

NOVEL LINEAR AND NONLINEAR OPTIMIZATION ALGORITHMS
WITH APPLICATION TO THE DIAL-A-RIDE PROBLEM

BY

MUNIRAH FARHAN ALQARNI

A Dissertation Presented to the
COLLEGE OF GRADUATE AND INTERDISCIPLINARY STUDIES
KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

DOCTOR OF PHILOSOPHY

In

MATHEMATICS

November, 2024

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS
DHAHRAN 31261, SAUDI ARABIA

DEANSHIP OF GRADUATE STUDIES

This dissertation, written by **MUNIRAH FARHAN AL-QARNI** under the direction of his dissertation adviser and approved by his dissertation committee, has been presented to and accepted by the Dean of Graduate Studies, in partial fulfillment of the requirements for the degree of **DOCTOR OF PHILOSOPHY IN MATHEMATICS**.

Dissertation Committee

Slim Belhaiza

Dr. Slim Belhaiza (Adviser)

Mohammed Alshahrani

Dr. Mohammed Alshahrani (Member)

Muhammad Riaz

Dr. Muhammad Riaz (Member)

Izhar Ahmad

Dr. Izhar Ahmad (Member)

Ahmed Al Hanbali

Dr. Ahmed Al Hanbali (Member)

Monther Alfuraidean

Department Chairman

Suliman Al-Homidan

Dean of Graduate Studies

Date



©Munirah Al-Qarni
2024

To the memory of my beloved Mother and Father .. The light in my life .. For instilling in me the love of reading and the pursuit of knowledge from my early sapling years May Allah reward them with the best possible rewards.

To my twin soul, my beloved sister, Dr. Garsah AlQarni... I am truly grateful for your boundless and unwavering support, your heartfelt prayers, and the endless love you always give.

ACKNOWLEDGMENTS

All praise and thanks be to Allah Almighty for giving me the knowledge, strength and patience to complete this work. I am deeply grateful to my supervisor Dr. Slim Belhaiza for his excellent supervision over the past three years. I sincerely thank him for his wisdom, invaluable comments and encouragement as well as his cooperation. I would like to express my gratitude to Dr. Mohammad Alshahrani and Dr. Adel Khalfallah for their support in this work and for giving very useful feedback. I also want to thank my thesis committee members for giving very useful feedback and recommendations. I would like to thank the faculty and staff of the Department of Mathematics and the Deanship of Graduate Studies at KFUPM, especially Professor Monther Alfuraidean, Professor Suliman Al-Homidan, and all my course instructors. Their support and guidance were crucial to my success in completing this thesis.

Finally, my special appreciation and thanks also go to my family; my husband, brother, sisters, sons and daughters for their direct and indirect motivation and support in helping me complete my thesis. May Allah reward them with the best possible rewards.

TABLE OF CONTENTS

ACKNOWLEDGMENTS	vi
LIST OF TABLES	x
LIST OF FIGURES	xi
ABSTRACT (ENGLISH)	xiv
ABSTRACT (ARABIC)	xvi
CHAPTER 1 GENERAL INTRODUCTION	1
CHAPTER 2 A NEW MULTIPLE PIVOT SIMPLEX ALGORITHM	4
2.1 Introduction	4
2.2 Linear Programming and Simplex Background	6
2.2.1 General Simplex Procedure	7
2.2.2 Modified Simplex Literature Review	9
2.3 The Multiple Pivot Simplex Algorithm	12
2.3.1 Updating rule for the Inverse Basis Matrix B^{-1}	15
2.3.2 Multiple Pivot Simplex Procedure	24
2.4 Computational Results	36
2.5 Conclusion	46
CHAPTER 3 A MODIFIED PSEUDO- INVERSE NEWTON- RAPHSON ALGORITHM FOR ARTIFICIAL NEURAL NET- WORKS	48

3.1	Introduction	48
3.2	Newton-Raphson methods: Literature Review	52
3.2.1	The general Newton method	52
3.2.2	Generalized-Inverse Newton-Raphson methods (GINR)	54
3.3	The Modified Pseudo-Inverse Newton-Raphson Method (MPINR)	55
3.3.1	The Moore-Penrose inverse or pseudo-inverse	56
3.3.2	Computing the pseudo-inverse matrix H^+	58
3.3.3	Convergence conditions	60
3.4	Computational Results	68
3.5	Conclusion	76

CHAPTER 4 FORECASTING TECHNIQUE FOR THE DIAL-A-RIDE PROBLEM WITH TIME WINDOWS 77

4.1	Introduction	77
4.2	Mathematical Description of DARPTW	78
4.3	Literature Review	80
4.3.1	Exact Approaches	81
4.3.2	Heuristic Approaches	83
4.3.3	Statistical and Machine Learning Approaches	88
4.4	Forecasting Approach: Definition and Applications	95
4.5	Neural Networks: Mathematical Background	96
4.5.1	Neural Network Architecture	97
4.5.2	Fundamentals of Artificial Neural Networks	98
4.5.3	Applications of Neural Networks	105
4.6	ANN Forecasting Technique for DARPTW	106
4.6.1	Training Data Set	107
4.6.2	Data Pre-Processing	108
4.6.3	Variance and Component Analysis	109
4.6.4	Singular Value Decomposition (SVD)	111
4.6.5	Principal Component Analysis (PCA)	117

4.6.6	Seasonality Decomposition	118
4.6.7	Multilayer Neural Network for DARPTW	121
4.6.8	Gradient Descent-based algorithms	122
4.7	Forecasting Experimental Results	127
4.8	Conclusion	133
CHAPTER 5	GENERAL CONCLUSION	134
APPENDIX A	PROOFS	136
APPENDIX B	SUPPLEMENTARY MATERIAL	137
REFERENCES		138
VITAE		157

LIST OF TABLES

2.1	Simplex Tableau Format	8
2.2	Simplex Tableau	9
2.3	Simplex Tableau	13
2.4	Initial Simplex Tableau	17
2.5	Simplex Tableau	22
2.6	Test Problems Details	38
2.7	Version 1 - Double Simplex Algorithm Results with Order and with Cuts	39
2.8	Version 2 - Double Simplex Algorithm Results with No Order and with Cuts	40
2.9	Version 3 - Double Simplex Algorithm Results with Order and No Cuts	41
2.10	Version 4 - Double Simplex Algorithm Results with No Order and No Cuts	42
3.1	Functions and Hessian Singularity Points	69
3.2	MPINR vs. GINR on functions f_1 to f_{15}	70
4.1	DARPMTW notation	80
4.2	RMSE of travel time forecasts using a two-layer ANN	129
4.3	RMSE of travel time forecasts using a three-layer ANN	130
4.4	RMSE of travel time forecasts using MPINR vs. GINR	132

LIST OF FIGURES

2.1	The flow chart of Multiple Pivot Simplex algorithm	30
2.2	Tree diagram of example 1	35
2.3	Comparative Performance I	43
2.4	Comparative Performance II	44
2.5	Comparative Performance III	45
2.6	Comparative Performance III	45
2.7	Comparative Performance IV	46
3.1	3D Graph - Example 4.	67
3.2	f_8 with starting point (4,4).	73
3.3	f_{11} with starting point (-2,4).	74
3.4	f_{15} with starting point (3,0.5).	75
4.1	A biological neuron	97
4.2	An Artificial Neuron [1]	98
4.3	Activation functions and their derivatives [2].	101
4.4	ANN Forecasting Steps.	106
4.5	Vancouver, Burrard st. to Highbury st.	108
4.6	Additive Seasonality Decomposition Results	121
4.7	Actual versus predicted travel times with SGD and $f_1() = f_2() =$ $f_3() : \text{relu}$	131

DISSERTATION ABSTRACT

NAME: Munirah Farhan Al-Qarni

TITLE OF STUDY: NOVEL LINEAR AND NON-LINEAR OPTIMIZATION
ALGORITHMS WITH APPLICATION TO THE DIAL-
A-RIDE PROBLEM

MAJOR FIELD: Mathematics

DATE OF DEGREE: November 2024

This thesis focuses on developing advanced optimization algorithms for both linear and non-linear problems, specifically a multiple pivot simplex algorithm and a modified pseudo-inverse Newton-Raphson algorithm, with applications to the Dial-A-Ride Problem with time windows. The research introduces a new multiple-pivot simplex algorithm to improve the computational efficiency of traditional linear programming techniques, particularly the simplex method. The dissertation explores optimization enhancements in initialization, iteration, and termination stages, with a focus on the reduction of computational costs and improvement in efficiency. The proposed algorithm was evaluated using randomly generated problems and the findings suggest that it yields promising results for larger problems.

For non-linear optimization problems, a Modified Pseudo-Inverse Newton-Raphson (MPINR) is proposed. The MPINR method utilizes the Moore-Penrose pseudo-inverse to overcome challenges found in traditional Newton-Raphson techniques, where the Hessian matrix may be singular or nearly singular. The results show that the proposed algorithm is competitive with the Generalized Inverse Newton-Raphson (GINR) method in terms of efficiency and number of iterations. It also enhances the toolkit available for optimizing complex error functions in neural networks, laying the groundwork for more efficient training and improved performance in various applications. One of the main contributions includes developing of an artificial neural networks-based forecasting technique to enhance decision-making in the Dial-A-Ride Problem, thereby minimizing operational costs while maximizing customer satisfaction. The findings suggest that the proposed methods yield promising results for addressing complex optimization problems in real-world scenarios, demonstrating their potential for practical applications in transportation and logistics.

ملخص الرسالة

الاسم: منيرة فرحان القرني

عنوان الدراسة: عنوان الرسالة

التخصص: الرياضيات

تاريخ الدرجة العلمية: أدخل الشهر والسنة

تقدم هذه الدراسة خوارزمية جديدة متعددة المحاور لتحسين الكفاءة الحسابية لطريقة السيمبلكس التقليدية، حيث تستكشف هذه الطريقة امكانية اجراء التحسينات في مراحل التهيئة والتكرار والتوقف، مع التركيز على تقليل التكاليف الحاسوبية. تم تقييم الخوارزمية المقترحة باستخدام مسائل تم توليدها عشوائيا باحجام مختلفة. وتشير النتائج أنها تحقق نتائج واعدة للمسائل الأكبر. كذلك تم اقتراح طريقة نيوتن رافسون المعدلة للمعكوس الزائف للتغلب على التحديات الموجودة في تقنيات نيوتن رافسون التقليدية وأبرز هذه التحديات كون مصفوفة هيسيان شاذة أو شبه شاذة. وقد أظهرت النتائج أن الخوارزمية المقترحة تنافس خوارزمية نيوتن رافسون للمعكوس المعمم من حيث كفاءة الحل وعدد التكرارات. كما أنها يمهد الطريق لتحسين دوال الخطأ المعقدة في الشبكات العصبية بهدف الحصول على تدريب أكثر كفاءة و أداء أفضل لهذه الشبكات في مختلف التطبيقات. ومن المساهمات الرئيسة تطوير تقنية للتنبؤ معتمدة على الشبكات العصبية الاصطناعية لتحسين اتخاذ القرار في واحدة من مشاكل النقل(DARP) بهدف تقليل التكاليف التشغيلية وزيادة رضا العملاء وقد تم تطبيق الطريقة المقترحة على مسائل نقل واقعية حيث أظهرت امكانية التطبيق العملي في مجال النقل اللوجستي.

CHAPTER 1

GENERAL INTRODUCTION

This thesis explores novel optimization algorithms and their applications to both linear and non-linear problems, with a primary aim of enhancing computational efficiency, convergence stability, and practical applicability. The research presents three main contributions: a multiple-pivot simplex algorithm for linear programming, an improved Newton-Raphson method for neural network optimization, and a forecasting technique for a transportation scheduling problem. Each contribution provides theoretical advancements and practical insights into optimization in diverse domains.

Chapter 2 presents a multiple-pivot simplex algorithm designed to improve upon the classical simplex method in linear programming. Traditional simplex methods iterate by pivoting a single variable at a time, which can lead to high computational costs, particularly for large problems. The proposed algorithm addresses this by selecting and pivoting multiple variables simultaneously, thereby reducing the number of iterations and improving overall computational efficiency. This chapter includes a comprehensive background on the simplex method, along with a literature review that highlights the recent advancements in pivot selection strategies. The chapter then

details the development of the multiple-pivot approach, including the mathematical basis for selecting pivots and maintaining solution feasibility. Computational results using test problems demonstrate the efficiency gains, numerical stability, and versatility of the multiple-pivot algorithm over conventional simplex methods, especially in handling large-scale linear programming tasks.

Chapter 3 introduces the Modified Pseudo-Inverse Newton-Raphson (MPINR) algorithm, a novel approach to solving non-linear optimization problems in artificial neural network training. Traditional Newton-Raphson methods rely on the Hessian matrix for determining optimal steps; however, this matrix can become singular or nearly singular, leading to convergence issues. The MPINR algorithm addresses these limitations by using the Moore-Penrose pseudo-inverse of the Hessian matrix, which allows the method to operate effectively even in cases of matrix singularity. This chapter provides an overview of Newton-Raphson methods, followed by a description of the pseudo-inverse approach and the derivation of the MPINR algorithm. Convergence conditions for the method are examined, establishing the robustness of MPINR in ensuring stable and rapid convergence. Computational experiments illustrate that MPINR outperforms the Generalized Inverse Newton-Raphson (GINR) method in terms of iteration efficiency and computational stability, making it a powerful tool for optimizing error functions in neural network models.

Chapter 4 applies artificial neural networks (ANNs) to address the Dial-A-Ride Problem with Time Windows (DARPTW), a scheduling and routing problem in transportation logistics. The DARPTW involves scheduling pick-ups and drop-offs for passengers within specified time windows, aiming to optimize factors such as travel time,

cost, and customer satisfaction. This chapter introduces a forecasting technique that uses ANNs to predict travel times and inform routing decisions in the DARPTW framework. The chapter provides a mathematical formulation of DARPTW, reviews existing exact and heuristic methods, and discusses the integration of machine learning techniques into forecasting. A detailed explanation of the neural network architecture is presented, along with pre-processing techniques like variance analysis and principal component analysis (PCA) to improve data quality. Experimental results demonstrate that the ANN-based forecasting model yields accurate travel time predictions, contributing to better decision-making in route planning, reduced operational costs, and increased customer satisfaction. This work showcases the potential of ANN-based forecasting for real-world optimization in transportation and logistics.

Overall, this thesis introduces a set of innovative algorithms that enhance the capabilities of optimization and forecasting in both theoretical and applied contexts. Together, these contributions provide valuable tools for addressing complex optimization problems across various domains, demonstrating potential impacts in fields such as transportation, logistics, and machine learning.

CHAPTER 2

A NEW MULTIPLE PIVOT SIMPLEX ALGORITHM

2.1 Introduction

In 1947, George Dantzig [3] introduced the simplex method, a groundbreaking approach for solving linear programs (LP). Numerous researchers have sought to advance Dantzig's ideas, contributing significantly to the widespread adoption of linear programming as the primary optimization technique in various fields. In addition to considering the theoretical underpinnings of the simplex method, the development of robust computer codes has been a key objective in this area, significantly enhancing its algorithmic efficiency and computational capabilities, which largely accounts for the simplex method's success.

Extension research on the simplex method has primarily focused on three stages: initialization, iteration, and termination. These extensions aim to enhance algorithm efficiency and reduce computational costs. The first significant breakthrough occurred

with the development of the revised simplex method by Dantzig and Orchard-Hays [4]. This method, although identical in algorithmic approach, demands less computational time due to its utilization of matrix operations. The runtime of the revised simplex method is bounded by $O(2^n)$ in the worst-case scenario [5]. Meanwhile, Spielman and Teng [6] proved that the revised simplex method may operate in polynomial time under certain conditions, particularly when instances are randomly generated and subject to slight perturbations. Other notable contributions are to be mentioned, such as: Huang *et al.*'s improvements to the initialization stage [7], Azlan *et al.*'s review of methods for modifying the termination stage [8], the use of sparse matrices by Tolla [9] and Suhl and Suhl [10], the reduced bases approach to address degeneracy by Elhallaoui *et al.* [11] and Raymand *et al.* [12], column generation technique developed by Ford and Fulkerson [13], the decomposition methods by Dantzig and Wolfe [14], Gilmore and Gomory's cutting approach [15], and Appelgren's column generation approach [16]. These approaches have collectively enhanced the computational efficiency of the simplex method.

More recently, Vitor and Easton [17] introduced the Double Pivot Simplex Method (DPSM), featuring double pivots and the Slope Algorithm to improve efficiency and accuracy in linear program optimization. Similarly, Yang [18] proposed a new method for solving linear programming problems, updating two variables at each iteration to consider longer-term benefits, especially for larger problem sizes.

Inspired by the work of Vitor and Easton [17] and Yang [18], and building upon their valuable contributions, this chapter introduces a novel multiple-pivot simplex algorithm. Our adaptation of the Simplex algorithm is designed to select the most

efficient subset of decision variables to enter the basis under specific conditions. Our proposed approach aims to improve the computational efficiency, numerical stability, and versatility of the Simplex algorithm in handling specific problem structures compared to traditional approaches.

2.2 Linear Programming and Simplex Background

Widely regarded as one of the most significant developments in science, the simplex method was acknowledged as one of the top 10 algorithms of the twentieth century [19]. Some of the earliest applications of linear programs were pioneered by Kantorovich [20] and Koopmans [21]. The simplex method transformed linear programs from mere theoretical models to practical tools [22].

This paper enhances the simplex method by devising an efficient algorithm capable of executing multiple pivots, and designed to solve linear programs in standard form (LPS):

$$\begin{aligned}
 (LPS) \quad & \max z = C^t x \\
 & \text{subject to } Ax = b \\
 & x \geq 0.
 \end{aligned}$$

We assume that LPS initially consists of $n+m$ variables and m constraints. Specifically, C^t is of size $1 \times (n+m)$, x of size $(n+m) \times 1$, A of size $m \times (n+m)$, and b of size $m \times 1$. For simplification purposes, we also presume that LPS is both feasible and bounded.

2.2.1 General Simplex Procedure

The general simplex procedure assumes a division of the decision variables vector x into two sub-vectors: x_B and x_N , where x_B represents the sub-vector for basic variables—those allowed to have non-zero values—and x_N represents the sub-vector for non-basic variables, which must be zero at each iteration. At any given iteration, there are consistently m variables in the basis, indicating that x_B is of size $m \times 1$ and x_N is of size $n \times 1$. Following this division, C^t also splits into two sub-vectors C_B^t and C_N^t , and A splits into two sub-matrices: B of size $m \times m$ and N of size $m \times n$.

LPS can be rewritten as follows:

$$\begin{aligned} (LPS) \quad & \max z = C^t x = C_B^t x_B + C_N^t x_N \\ & \text{subject to } Bx_B + Nx_N = b \\ & x_B, x_N \geq 0. \end{aligned}$$

Therefore, at every feasible basic solution (with non-singular B) we can write:

$$x_B = B^{-1}b - B^{-1}Nx_N,$$

and

$$z = C_B^t B^{-1}b + (C_N^t - C_B^t B^{-1}N) x_N.$$

In common simplex terminology, the basic feasible solution is said to be optimal if and only if all the components of that reduced costs vector $RC_N^t = C_N^t - C_B^t B^{-1}N$ are negative or zeros. The Simplex algorithm is generally presented using its famous

Tableau representation which reproduces the previous algebraic notation but with a more practical computational format. Hence, a feasible basic solution would be represented by the Simplex Tableau in Table 2.1.

var	x_B	x_N	Value
x_B	I	$B^{-1}N$	$B^{-1}b$
RC	0	$C_N^t - C_B^t B^{-1}N$	$z = C_B^t B^{-1}b$

Table 2.1: Simplex Tableau Format

The simplex algorithm initiates at a vertex of the feasible region and iteratively moves along the edges to enhance the objective function value until an optimal solution is achieved.

Let us consider a scenario where, at a particular iteration, we encounter the Tableau representation depicted in Table 2.2. The Simplex algorithm typically selects a candidate entering variable x_i with a strictly positive reduced cost $RC_i > 0$. This selection may be based on the highest reduced cost value, or a simpler lexicographic rule that prioritizes variables with the lowest order index in the linear program (LPS) formulation to ensure non-cycling. Once the entering variable is chosen, the minimum ratio comes into play to determine the variable leaving the basis, ensuring the feasibility of the new basis. The variable departing from the basis is then the one that yields the smallest positive value for the ratio $\frac{b_k}{a_{ki}}$ among all variables x_k in the basis.

var	$\cdots$	x_i	$\cdots$	Value	Ratio
$\cdots$	$\cdots$		$\cdots$	$\cdots$	$\cdots$
x_k	$\cdots$	a_{ki}	$\cdots$	b_k	$\frac{b_k}{a_{ki}}$
$\cdots$	$\cdots$		$\cdots$	$\cdots$	
RC	$\cdots$	RC_i	$\cdots$	$\cdots$	$\cdots$

Table 2.2: Simplex Tableau

The classical simplex method [3] updates the entire simplex tableau at each iteration, although not all numbers in the tableau are needed in each iteration. Choosing the entering basic variable requires only the numbers in the row of objective function coefficients, and choosing the leaving basic variable requires the numbers in the column corresponding to that entering basic variable and the numbers in the right-hand side of the tableau just one row and two columns out of the entire tableau. The revised simplex method [4] employs the right matrix implementation and rules, avoiding the need to invert the basis matrix at every iteration, thus speeding up the simplex method’s performance significantly.

2.2.2 Modified Simplex Literature Review

Millions of linear programs are solved daily, contributing to the creation of a more efficient world. Presently, linear programs play a pivotal role in society by offering improved solutions to a myriad of problems across various industries including medicine (Lee *et al.* [23]), finance (Mansini *et al.* [24]), manufacturing (Gomes and Oliveira [25]), logistics (Spitter *et al.* [26]), and governance (Gautier *et al.* [27]).

The imperative need to solve linear programs more efficiently has spurred researchers to pursue various advancement. Harris [28] introduced pivot column and

row selection methods utilized by the Devex code for the first time. This paper introduces dynamic column weighting factors to estimate gradients for selecting a maximum gradient column and examines the resulting impact on rounding errors. It also discusses the implications of imprecise constraint positioning, which broadens the choice of pivot rows and introduces a further selection criterion based on pivot size. The paper includes three examples of problems with 2500 to 5000 rows, demonstrating the overall time and iteration advantages of the Devex code over standard simplex methods in solving large-scale linear programming problems. The Devex code has shown a significant reduction in the number of iterations required to reach the optimal solution, leading to faster convergence and shorter computation times.

Goldfarb and Sit [29] investigated the behavior of the steepest-edge simplex method in linear programming. This method differs from other pivot rules by maximizing the gradient of the objective function in the space of all variables. While efficient in practice, they demonstrate that it can exhibit worst-case behavior in certain scenarios, constructing a sequence of linear programs requiring $2^n - 1$ iterations to solve.

Bland [30] introduced new finite pivoting rules impacting the efficiency and effectiveness of the simplex method by ensuring finiteness. It provides a simple proof of finiteness for the simplex method under these new rules, guaranteeing termination within a finite number of steps. The new rules are designed to reduce or eliminate cycling, thereby improving efficiency by minimizing unnecessary iterations. Bland [30] emphasized the ease of implementing these rules, enhancing their practical applicability in solving linear programming problems across various domains.

Chen *et. al* [31] presented an algorithm maintaining primal and dual feasibility

in linear programming problems, employing a new pivot rule potentially leading to improved computational efficiency and reduced iteration numbers compared to traditional simplex algorithms.

Junior and Lins [32] proposed a new approach to improving pivoting algorithms by suggesting a better initial basis for the Simplex algorithm. By selecting a vertex of the feasible region closer to the optimal vertex, they improved the iteration efficiency of the Simplex algorithm.

Djannaty and Rostamy [33] explored different strategies for selecting pivot variables in the simplex method, focusing on LP relaxation of set problems, to propose new strategies that can enhance the method's efficiency.

Pan [34] proposed a new pivot rule, the largest-distance pivot rule, aiming to improve the performance of the simplex algorithm by providing a simpler and more computationally efficient alternative. To further improve efficiency, Pan [35] combined the largest-distance rule with the nest pricing rule, demonstrating its potential to outperform other pivot rules in computational experiments.

Omer *et al.* [36] suggested a general theoretical framework for an algorithm derived from the primal simplex, guaranteeing a strict improvement in the objective after each iteration, and provided insights into its practical implications.

Chankong *et al.* [37] explored the absolute change pivot rule, aiming to reduce the number of iterations needed to reach the optimal solution, focusing on maximizing the improvement in the objective value function.

In more recent developments, Vitor and Easton [17] introduced the Double Pivot Simplex Method (DPSM), which incorporates double pivots and the Slope Algorithm

to enhance the efficiency and accuracy of linear program optimization. This algorithm is applicable when the following conditions are met: the objective function coefficients are positive, the right-hand side values of all constraints are nonnegative, and both variables are constrained to be nonnegative. Similarly, Yang [18] proposed a novel approach to address linear programming problems, updating two variables during each iteration to account for longer-term advantages, particularly beneficial for larger problem sizes.

Overall, the simplex method continues to evolve through innovative research, paving the way for more efficient and effective optimization techniques in various domains.

2.3 The Multiple Pivot Simplex Algorithm

This section will illustrate the techniques which are used in the proposed method. Actually, there are differences of what we have implemented vs the double pivot simplex method proposed by Vitor and Easton, where their method follows the spirit of Dantzig's rule and selects the two indices with the most negative reduced cost for the entering variables. Each iteration of this method relies on the slope algorithm to determine the optimal basis and identify the leaving variables. This process is repeated until the optimal solution to the original problem is achieved. Their Computational results show that the algorithm reduces the number of iterations required by the simplex method. However, it requires additional time to use the slope algorithm for identifying two leaving variables per iteration. Furthermore, in some instances, it may only identify a single variable.

Now let us assume that during a particular iteration, we have multiple candidates for entering the basis. For instance, the tableau representation provided in Table 2.3 assumes that both RC_i and RC_j are greater than zero.

var	$\cdots$	x_i	$\cdots$	x_j	$\cdots$	Value	$Ratio_i$	$Ratio_j$
$\cdots$	$\cdots$		$\cdots$	$\cdots$				
x_k	$\cdots$	a_{ki}	$\cdots$	a_{kj}		b_k	$\frac{b_k}{a_{ki}}$	$\frac{b_k}{a_{kj}}$
$\cdots$	$\cdots$		$\cdots$	$\cdots$				
x_l	$\cdots$	a_{li}	$\cdots$	a_{lj}		b_l	$\frac{b_l}{a_{li}}$	$\frac{b_l}{a_{lj}}$
RC	$\cdots$	RC_i	$\cdots$	RC_j	$\cdots$		$\cdots$	

Table 2.3: Simplex Tableau

Furthermore, let us assume that the following relationships hold:

$$\frac{b_k}{a_{ki}} \leq \frac{b_l}{a_{li}} \text{ and } \frac{b_l}{a_{lj}} \leq \frac{b_k}{a_{kj}}. \quad (2.1)$$

Consequently, x_i is a candidate for entering the basis and replacing x_k , while x_j is a candidate for entering the basis and replacing x_l . Extracted from the simplex tableau, the corresponding rows for x_k and x_l can be expressed as follows:

$$\begin{aligned} x_k &= b_k - \cdots - a_{ki}x_i - \cdots - a_{kj}x_j - \cdots \\ x_l &= b_l - \cdots - a_{li}x_i - \cdots - a_{lj}x_j - \cdots \end{aligned}$$

If the two moves were to be executed simultaneously, we would obtain the following rows for x_i and x_j :

$$\begin{aligned}
x_i &= \frac{b_k}{a_{ki}} - \dots - \frac{a_{kj}}{a_{ki}} x_j - \dots - \frac{1}{a_{ki}} x_k - \dots \\
x_j &= \frac{b_l}{a_{lj}} - \dots - \frac{a_{li}}{a_{lj}} x_i - \dots - \frac{1}{a_{lj}} x_l - \dots
\end{aligned}$$

which yield:

$$\begin{aligned}
x_i &= \frac{b_k}{a_{ki}} - \dots - \frac{a_{kj}}{a_{ki}} \left(\frac{b_l}{a_{lj}} - \frac{a_{li}}{a_{lj}} x_i \right) - \dots \\
x_j &= \frac{b_l}{a_{lj}} - \dots - \frac{a_{li}}{a_{lj}} \left(\frac{b_k}{a_{ki}} - \frac{a_{kj}}{a_{ki}} x_j \right) - \dots
\end{aligned}$$

and again:

$$\begin{aligned}
x_i^* &= \frac{b_k a_{lj} - a_{kj} b_l}{a_{ki} a_{lj} - a_{kj} a_{li}} - \dots \\
x_j^* &= \frac{b_l a_{ki} - a_{li} b_k}{a_{ki} a_{lj} - a_{kj} a_{li}} - \dots
\end{aligned}$$

This corresponds to the well-known Cramer's rule solution for the linear system $Ax = b$. Each variable x_i or x_j value is expressed as the ratio of determinants of two sub-matrices:

$$x_i^* = \frac{\det(B_i^{kl})}{\det(A_{ij}^{kl})} \quad \text{and} \quad x_j^* = \frac{\det(B_j^{kl})}{\det(A_{ij}^{kl})}$$

where B_i^{kl} contains rows k and l from column a_i augmented with the row elements k and l from b , while A_{ij}^{kl} contains rows k and l , along with columns i and j from A (2×2 matrices).

Based on conditions (2.1), and assuming that all a_{ki}, a_{kj}, a_{li} and a_{lj} are positive, we obtain $b_k a_{li} \leq b_l a_{ki}$ and $b_l a_{kj} \leq b_k a_{lj}$. Consequently, for the simultaneous move to be feasible for the entering variables x_i and x_j , the value of $(a_{ki} a_{lj} - a_{kj} a_{li})$ must be strictly positive. In simpler terms, a strictly positive determinant for the sub-matrix $\begin{pmatrix} a_{ki} & a_{kj} \\ a_{li} & a_{lj} \end{pmatrix}$ is necessary. However, this condition may not be sufficient as we need to make sure that all variables are positive. In the next section, we show how a

generalized rule for the inverse of the basis matrix B^{-1} updating is obtained.

One may also speculate in the short term that this move is worthwhile if the following condition holds:

$$RC_i x_i^* + RC_j x_j^* \geq \max \left\{ RC_i \frac{b_k}{a_{ki}}, RC_j \frac{b_l}{a_{lj}} \right\}.$$

This condition will be used to improve the performance of the multiple pivot simplex procedure and cut out unnecessary moves.

2.3.1 Updating rule for the Inverse Basis Matrix B^{-1}

At every iteration of the generalized multiple simplex algorithm, the basis matrix $B_{m \times m}$ is updated, and a new basis matrix B' is obtained. Let us assume that

$$B' = B + \Delta B$$

where

$$\Delta B = UV^T = B' - B.$$

In the case where two variables x_i and x_j are entering the basis and the two variables x_k and x_l are leaving the basis, we can write U as a $m \times 2$ matrix, and V as a $2 \times m$ matrix using the columns from A involved in the basis matrix B updating: col_i , col_j , col_k and col_l .

We then have:

$$\begin{aligned}
UV^T &= \begin{pmatrix} 0 & 0 & \dots & col_i - col_k & \dots & col_j - col_l & 0 & 0 \end{pmatrix}_{m \times m} \\
&= \begin{pmatrix} 0 & \dots & a_{1i} - a_{1k} & 0 & \dots & 0 & a_{1j} - a_{1l} & 0 & \dots & 0 \\ 0 & \dots & a_{mi} - a_{mk} & 0 & \dots & 0 & a_{mj} - a_{ml} & 0 & \dots & 0 \end{pmatrix} \\
&= \begin{pmatrix} a_{1i} - a_{1k} \\ \vdots \\ a_{mi} - a_{mk} \end{pmatrix} \begin{pmatrix} 0 & \dots & 1 & 0 & \dots & 0 & 0 & \dots & 0 \end{pmatrix} + \begin{pmatrix} a_{1j} - a_{1l} \\ \vdots \\ a_{mj} - a_{ml} \end{pmatrix} \begin{pmatrix} 0 & \dots & 0 & 0 & \dots & 1 & 0 & \dots & 0 \end{pmatrix} \\
&= \begin{pmatrix} a_{1i} - a_{1k} & a_{1j} - a_{1l} \\ \vdots & \vdots \\ a_{mi} - a_{mk} & a_{mj} - a_{ml} \end{pmatrix}_{m \times 2} \begin{pmatrix} 0 & \dots & 1 & 0 & \dots & 0 & 0 & \dots & 0 \\ 0 & \dots & 0 & 0 & \dots & 1 & 0 & \dots & 0 \end{pmatrix}_{2 \times m}
\end{aligned}$$

From the Sherman- Morrison- Woodbury (SMW) formula, the inverse of B' is written as:

$$(B')^{-1} = B^{-1} - B^{-1}U(I_h + V^T B^{-1}U)^{-1}V^T B^{-1}.$$

Therefore the condition

$$B'^{-1}b \geq 0$$

can be written as:

$$(B^{-1} - B^{-1}U(I_h + V^T B^{-1}U)^{-1}V^T B^{-1})b \geq 0$$

$$B^{-1}b - B^{-1}U(I_h + V^T B^{-1}U)^{-1}V^T B^{-1}b \geq 0$$

$$(I_m - B^{-1}U(I_h + V^T B^{-1}U)^{-1}V^T)B^{-1}b \geq 0$$

In the general multiple-pivot case where h variables enter the basis simultaneously and h variables leave, U is a $m \times h$, and V^T is a $h \times m$ matrix.

Example 1 Let us consider the following linear program in standard form:

$$\begin{aligned}
(LPS1) \quad & \max 2x_1 + 3x_2 + 2x_3 \\
\text{subject to} \quad & x_1 + 3x_2 + 2x_3 + x_4 = 4 \\
& 2x_1 + 2x_2 + x_3 + x_5 = 2 \\
& x_1 + x_2 + 2x_3 + x_6 = 3 \\
& x_1, x_2, x_3, x_4, x_5, x_6 \geq 0.
\end{aligned}$$

The first simplex tableau writes as:

var	x_1	x_2	x_3	x_4	x_5	x_6	value	$Ratio_1$	$Ratio_2$	$Ratio_3$
x_4	1	3	2	1	0	0	4	4	$\frac{4}{3}$	2
x_5	2	2	1	0	1	0	2	1	1	2
x_6	1	1	2	0	0	1	3	3	3	$\frac{3}{2}$
RC	2	3	2	0	0	0				

Table 2.4: Initial Simplex Tableau

Let λ estimate the potential improvement brought by one iteration of the Simplex algorithm. In the case of one single entering variable, the estimated λ is the product of the RC of the entering variable by its expected new value, represented here by the minimum *Ratio*. In this example, we have the possibility to choose the entering variable as follows:

$$x_1 \text{ where : } \lambda_1 = RC_1 \times Ratio_1 = 2 \times 1 = 2$$

$$x_2 \text{ where : } \lambda_2 = RC_2 \times Ratio_2 = 3 \times 1 = 3$$

$$x_3 \text{ where : } \lambda_3 = RC_3 \times Ratio_3 = 2 \times \frac{3}{2} = 3$$

In the case where the two variables x_1 and x_3 are simultaneously entering the basis, and the two variables x_5, x_6 are leaving, the estimated improvement would then be the linear combination of the reduced costs of x_1 and x_3 and their expected new values obtained by the Cramer rule:

$$\begin{aligned} \lambda_{1,3} &= RC_1 \times \frac{\det B_1^{5,6}}{\det A_{1,3}^{5,6}} + RC_3 \times \frac{\det B_3^{5,6}}{\det A_{1,3}^{5,6}} \\ &= 2 \times \frac{\begin{vmatrix} 2 & 1 \\ 3 & 2 \end{vmatrix}}{\begin{vmatrix} 2 & 1 \\ 1 & 2 \end{vmatrix}} + 2 \times \frac{\begin{vmatrix} 2 & 2 \\ 1 & 3 \end{vmatrix}}{\begin{vmatrix} 2 & 1 \\ 1 & 2 \end{vmatrix}} = 2 \times \frac{1}{3} + 2 \times \frac{4}{3} = \frac{10}{3} \end{aligned}$$

While, if x_2, x_3 are the two entering variables, then x_5, x_6 will be leaving variables,

and consequently the estimated improvement would be:

$$\lambda_{2,3} = RC_2 \times \frac{\det B_2^{5,6}}{\det A_{2,3}^{5,6}} + RC_3 \times \frac{\det B_3^{5,6}}{\det A_{2,3}^{5,6}} = 3 \times \frac{1}{3} + 2 \times \frac{4}{3} = \frac{11}{3}$$

Note here that x_1 and x_2 cannot enter simultaneously the basis as x_5 will be only the leaving variable. Similarly, there is no possibility here to have all three variables enter simultaneously the basis.

It can be observed that the variables x_2 and x_3 provide the highest potential improvement as $\lambda_{2,3} > \lambda_{1,3}$. One can also notice here that:

$$\lambda_{2,3} = RC_2 x_2^* + RC_3 x_3^* \geq \max \left\{ \frac{2}{2} RC_2, \frac{3}{2} RC_3 \right\}$$

as we have:

$$3 \frac{1}{3} + 2 \frac{4}{3} = \frac{11}{3} \geq \max \{3, 3\}.$$

The newly obtained basis comprises the variables x_4 , x_2 , and x_3 , while the non-basic variables are x_1 , x_5 , and x_6 . Therefore, the updated basic matrix is: $B = \begin{pmatrix} 1 & 3 & 2 \\ 0 & 2 & 1 \\ 0 & 1 & 2 \end{pmatrix}$, and the non-basic matrix is: $N = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}$.

The coefficients of the basic and non-basic variables are then $C_B^t = (0, 3, 2)$ and $C_N^t = (2, 0, 0)$, respectively.

Therefore, $B^{-1} = \frac{1}{3} \begin{pmatrix} 3 & -4 & -1 \\ 0 & 2 & -1 \\ 0 & -1 & 2 \end{pmatrix}$, and the reduced costs of the non-basic variables are:

$$RC_N^t = C_N^t - C_B^t B^{-1} N = (-1, -\frac{4}{3}, -\frac{1}{3}).$$

The reduced costs of the non-basic variables are all negative. Hence, the optimal solution is reached in one single double pivot iteration with:

$$\begin{pmatrix} x_4^* \\ x_2^* \\ x_3^* \end{pmatrix} = B^{-1}b = \begin{pmatrix} \frac{1}{3} \\ \frac{1}{3} \\ \frac{4}{3} \end{pmatrix}.$$

In this example the best choice is x_2 , x_3 are entering variables, while x_5 , x_6 are leaving variables, hence:

$$UV^T = \begin{pmatrix} 0 & a_{12} - a_{15} & a_{13} - a_{16} \\ 0 & a_{22} - a_{25} & a_{23} - a_{26} \\ 0 & a_{32} - a_{35} & a_{33} - a_{36} \end{pmatrix} = \begin{pmatrix} 0 & 3 & 2 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} 3 & 2 \\ 1 & 1 \\ 1 & 1 \end{pmatrix}_{3 \times 2} \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}_{2 \times 3}$$

Therefore:

$$U = \begin{pmatrix} 3 & 2 \\ 1 & 1 \\ 1 & 1 \end{pmatrix}, V^T = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Then the updated B is:

$$B' = B + UV^T = \begin{pmatrix} 1 & 3 & 2 \\ 0 & 2 & 1 \\ 0 & 1 & 2 \end{pmatrix}$$

Applying SMW formula yields as expected:

$$(B')^{-1} = \frac{1}{3} \begin{pmatrix} 3 & -4 & -1 \\ 0 & 2 & -1 \\ 0 & -1 & 2 \end{pmatrix}$$

Example 2 Let us now consider a modified version of the linear program ($LP1$)

with:

$$(LP2) \quad \max 2x_1 + 3x_2 + 2x_3$$

$$\text{subject to } 5x_1 + 3x_2 + 2x_3 \leq 4$$

$$2x_1 + 2x_2 + x_3 \leq 2$$

$$x_1 + x_2 + 2x_3 \leq 3$$

$$x_1, x_2, x_3 \geq 0.$$

The first simplex tableau writes as:

var	x_1	x_2	x_3	x_4	x_5	x_6	b	R_1	R_2	R_3
x_4	5	3	2	1	0	0	4	$\frac{4}{5}$	$\frac{4}{3}$	2
x_5	2	2	1	0	1	0	2	1	1	2
x_6	1	1	2	0	0	1	3	3	3	$\frac{3}{2}$
RC	2	3	2	0	0	0				

Table 2.5: Simplex Tableau

If we consider x_1 as the entering variable, then x_4 will be leaving variable, consequently: $\lambda_1 = RC_1 \times Ratio_1 = 2 \times \frac{4}{5} = \frac{8}{5}$.

In the case where x_2 is entering: $\lambda_2 = RC_2 \times Ratio_2 = 3 \times 1 = 3$.

In the case where x_3 is entering: $\lambda_3 = RC_3 \times Ratio_3 = 2 \times \frac{3}{2} = 3$.

Taking two entering variables yields the following possibilities:

$$x_1, x_2 \longrightarrow \lambda_{1,2} = RC_1 \times \frac{\det B_1^{4,5}}{\det A_{1,2}^{4,5}} + RC_2 \times \frac{\det B_2^{4,5}}{\det A_{1,2}^{4,5}} = 2 \times \frac{2}{4} + 2 \times \frac{2}{4} = \frac{5}{2} = 2.5$$

$$x_1, x_3 \longrightarrow \lambda_{1,3} = RC_1 \times \frac{\det B_1^{5,6}}{\det A_{1,3}^{5,6}} + RC_3 \times \frac{\det B_3^{5,6}}{\det A_{1,3}^{5,6}} = 2 \times \frac{2}{8} + 2 \times \frac{11}{8} = \frac{13}{4} = 3.25$$

$$x_2, x_3 \longrightarrow \lambda_{2,3} = RC_2 \times \frac{\det B_2^{5,6}}{\det A_{2,3}^{5,6}} + RC_3 \times \frac{\det B_3^{5,6}}{\det A_{2,3}^{5,6}} = 3 \times \frac{1}{3} + 2 \times \frac{4}{3} = \frac{11}{3} = 3.66$$

One can notice here that variables x_1 , x_2 , and x_3 have strictly positive reduced costs, while x_1 is a candidate to take the position of x_4 in the basis, x_2 is a candidate to take the position of x_5 in the basis, while x_3 is a candidate to take the position of

x_6 . We also have a positive determinant for the new sub-matrix $B' = \begin{pmatrix} 5 & 3 & 2 \\ 2 & 2 & 1 \\ 1 & 1 & 2 \end{pmatrix}$.

Therefore, it is possible here to have all three variables x_1 , x_2 and x_3 enter simultaneously the basis with an expected improvement:

$$x_1, x_2, x_3 \longrightarrow \lambda_{1,2,3} = RC_1 \times \frac{\det B_1^{4,5,6}}{\det A_{1,2,3}^{4,5,6}} + RC_2 \times \frac{\det B_2^{4,5,6}}{\det A_{1,2,3}^{4,5,6}} + RC_3 \times \frac{\det B_3^{4,5,6}}{\det A_{1,2,3}^{4,5,6}} = 2 \times \frac{4}{5} + 3 \times 1 + 2 \times \frac{3}{2} = 7.6$$

Notice here that $\lambda_{1,2,3}$ has a larger value than any other expected improvement.

In fact, if the three variables move simultaneously to the basis, we get

$$\begin{pmatrix} x_1^* \\ x_2^* \\ x_3^* \end{pmatrix} = (B')^{-1}b = \begin{pmatrix} \frac{1}{6} \\ \frac{1}{6} \\ \frac{4}{3} \end{pmatrix}.$$

One can also notice that:

$$\lambda_{1,2,3} = RC_1x_1^* + RC_2x_2^* + RC_3x_3^* \geq \max \left\{ \frac{4}{5}RC_1, \frac{2}{2}RC_2, \frac{3}{2}RC_3 \right\}$$

as we have:

$$2\left(\frac{1}{6}\right) + 3\left(\frac{1}{6}\right) + 2\left(\frac{4}{3}\right) = \frac{21}{6} \geq \max \left\{ \frac{8}{5}, 3, 3 \right\}.$$

The new reduced costs of the three new non-basic variables would then be:

$$RC_N^t = C_N^t - C_B^t(B')^{-1}N = \left(\frac{1}{2}, -2, -\frac{1}{2}\right).$$

Hence, the solution is still not optimal. The simultaneous move of the two variables x_2 and x_3 to the basis would have been optimal. This could not have been easily predicted. Therefore, to design an effective multiple pivot simplex, we need to set-up some clear rules to follow when trying to prioritize a particular move with respect to other ones.

In this particular case, we also have a positive determinant when x_2 and x_3 enter the

basis, with the new sub-matrix $B' = \begin{pmatrix} 1 & 3 & 2 \\ 0 & 2 & 1 \\ 0 & 1 & 2 \end{pmatrix}$, while

$$N' = \begin{pmatrix} 5 & 0 & 0 \\ 2 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} \quad \text{and} \quad (B')^{-1} = \frac{1}{3} \begin{pmatrix} 3 & -4 & -1 \\ 0 & 2 & -1 \\ 0 & -1 & 2 \end{pmatrix}.$$

Hence, if the x_2, x_3 move simultaneously to the basis, we get:

$$\begin{pmatrix} x_1^* \\ x_2^* \\ x_3^* \end{pmatrix} = (B')^{-1}b = \begin{pmatrix} \frac{1}{3} \\ \frac{1}{3} \\ \frac{4}{3} \end{pmatrix}.$$

This solution is optimal as indicated by the new reduced costs of the these non-basic variables would then be:

$$RC_N^t = C_N^t - C_B^t(B')^{-1}N' = (-1, -\frac{4}{3}, -\frac{1}{3}).$$

2.3.2 Multiple Pivot Simplex Procedure

We now formally propose a multiple pivot simplex algorithm to solve linear programming problems. It is an extension of the traditional simplex algorithm. It develops the concept of multiple pivot simplex iteration, allowing for updates to the basis and more effective exploration of the solution space.

The logic behind the multiple pivot simplex algorithm revolves around improving the objective function value and reaching an optimal solution. It begins with an initial feasible solution and iteratively moves from one feasible solution to another, always improving the objective function value. Similar to the traditional simplex algorithm, it starts with an initial feasible solution and proceeds to move from one feasible solution to another.

The multiple simplex pivot selects a number of entering variables and their corresponding leaving variables, which may be two or more, depending (or not) on the largest value of the expected improvement value (which we already defined as λ) among all combinations of a set of variables with positive reduced cost.

Before we start describing the algorithm, we define the following sets. The sets of indices of the basic and non-basic variables

$$\mathcal{I}_B = \{i \mid x_i \in x_B\}, \quad \mathcal{I}_N = \{i \mid x_i \in x_N\}. \quad (2.2)$$

Now for every non-basic variable x_i , we define the set of all indices of non-basic variables that can enter the basis with x_i

$$\gamma_i = \{j \in \mathcal{I}_N \mid x_i, x_j \text{ satisfy (2.1) for some } k, l \in \mathcal{I}_B\} \quad (2.3)$$

The set S_Q of all variables with a positive reduced cost; that is

$$S_Q = \{x_i \in x_N \mid RC_i > 0\}, \quad (2.4)$$

with Q as the number of variables in this set. Further, we define the set C_Q of all possible variables eligible for entering the basis as

$$C_Q = \{\gamma_i \mid i \in \mathcal{I}_N\}, \quad (2.5)$$

Now the algorithm is initialized with a feasible solution and updating the reduced costs. If the set S_Q is empty, it means that we already have an optimal solution. Otherwise, we examine iteratively the combinations C_r where $r \in C_Q$.

After determining the entering and leaving variables, the algorithm updates the basis matrix B_r using a UV^T decomposition, which represents the value of the changes obtained. As detailed earlier, U is an $m \times h$ matrix and V^T is an $h \times m$ matrix, with h being the number of chosen entering variables. This UV^T is then added to the previous basic matrix to obtain a new basic matrix B'_r . Subsequently, the algorithm recalculates the basic variables and computes the new solution.

Notice here that the UV^T decomposition is either performed by extracting U and V^t from A as detailed before, or using CR factorization based on the Singular value Decomposition of $B' - B$.

From the Sherman–Morrison–Woodbury (SMW) formula, the inverse of B_r is written as: $(B_r)^{-1} = B^{-1} - B^{-1}U(I_h + V^TB^{-1}U)^{-1}V^TB^{-1}$. Therefore the condition $B_r^{-1}b \geq 0$ needs to be satisfied to generate feasible solutions. In addition, the current solution is improved if: $\Delta z(C_r) = C_{B_r}^t B_r^{-1}b - C_B^t B^{-1}b \geq 0$.

By iteratively following this process, the algorithm progressively improves the objective function value until an optimal solution with all reduced costs negative or zero

is reached.

A QMatrix is a $Q \times (Q + 1)$ matrix is primarily used to identify all valid combinations without conflicts and their corresponding improvements. The value of the parameter Q represents the number of variables with $RC > 0$. The last column of the *QMatrix* stores the value of the expected improvement λ_r for each combination C_r . The reordering or not of the *QMatrix* rows in decreasing order of the λ_r yields different prioritization strategies for the algorithm.

For instance, starting from the initial solution in Example 1, the QMatrix writes as follows:

$$QMatrix = \begin{pmatrix} 1 & 0 & 1 & 5 \\ 0 & 1 & 1 & 6 \\ 0 & 0 & 1 & 3 \end{pmatrix}$$

The entry in $QMatrix(1,1) = 1$ indicates that x_1 is a possible entering variable, while $QMatrix(1,2) = 0$ means that x_1 and x_2 cannot be used together as entering variables due to a conflict, as both would make x_5 leave the basis. Also, the entry $QMatrix(1,3) = 1$ means that x_1 and x_3 can be used together as entering variables without conflict, since x_5 and x_6 will be, respectively, leaving the basis.

For the entering variable x_1 , the maximum improvement value is determined by its reduced cost 2 multiplied by the maximum new value for x_1 , which is equal to 1 (minimum ratio rule). For x_3 , the maximum improvement value is determined by its reduced cost 3 multiplied by the maximum new value for x_3 , which is equal to 1 (minimum ratio rule). Therefore, the maximum possible improvement for the pair x_1 and x_3 appears in the first row and cannot exceed 5. And so on for the other rows.

Hence, for each variable q in S_Q , we calculate the *Ratios* with the minimum ratio rule. We then generate C_Q , the set of all combinations generated by S_Q . For each combination $C_r \in C_Q$, we use the Cramer rule where for each variable q in C_r : $\hat{x}_q = \frac{\det(B(C_r))}{\det(A(C_r))}$ to calculate the estimated improvement $\lambda_r = \sum_{q \in C_r} RC_q \times \hat{x}_q$.

The new multiple pivot simplex algorithm illustrated in the flow chart Figure 4.5 can now be formally stated in Algorithm 1.

Algorithm 1: Multiple Pivot Simplex algorithm

```
1 Step 0:
2   Initialize Feasible solution basis  $B$ ;
3   Set best basis  $B^* = B$ ;
4 Step 1:
5   a. Let:  $\Delta z = 0$ ;  $\overline{\Delta z} = 0$ ;
6        $x_B = B^{-1}b$  ;  $Z = C_B^T x_B$ 
7        $RC = C_N^T - C_B^T B^{-1}A$  ;
8   b. Let  $S_Q$ : Set of variables with  $RC > 0$ 
9       Let  $Q = |S_Q|$ ; be the number of  $RC > 0$ ;
10      if  $Q = 0$  ; Go To Step 3
11      else Go To Step 2
12 Step 2:
13 - For each variable  $q$  in  $S_Q$ , calculate the  $Ratios_q$  with the minimum ratio
    rule.
14 - For each combination  $C_r \in C_Q$ : Calculate the estimated improvement  $\lambda_r$ .
15 - Create  $QMatrix$  with a dimension of  $(Q \times Q + 1)$ , fill in its columns, and
    reorder its rows if needed.
16 - For each combination  $C_r$ , where  $r = 1, \dots, |C_Q|$ :
17     if  $\lambda_r \geq \overline{\Delta z}$ 
18         - Let  $B_r = A(C_r)$ 
19         - Get  $UV^t = B_r - B$ 
20         - Compute  $(B_r)^{-1} = B^{-1} - B^{-1}U(I_h + V^T B^{-1}U)^{-1}V^T B^{-1}$ 
21         - if  $(B_r)^{-1}b \geq 0$ 
22             - Compute  $\Delta z(C_r) = C_{B_r}^t (B_r)^{-1}b - C_B^t B^{-1}b$ 
23             - if  $\Delta z > \overline{\Delta z}$ 
24                 -  $\overline{\Delta z} = \Delta z$ ,
25                 - Record best basis  $B^* = B_r$ .
26             - Record basis  $B = B_r$  and Go to Step 1.
27         else Unfeasible basis.
28     else Cut  $C_r$  Combination.
29 - EndFor
30 Step 3: Optimal Solution is reached.
31 Return  $B^*$ 
```

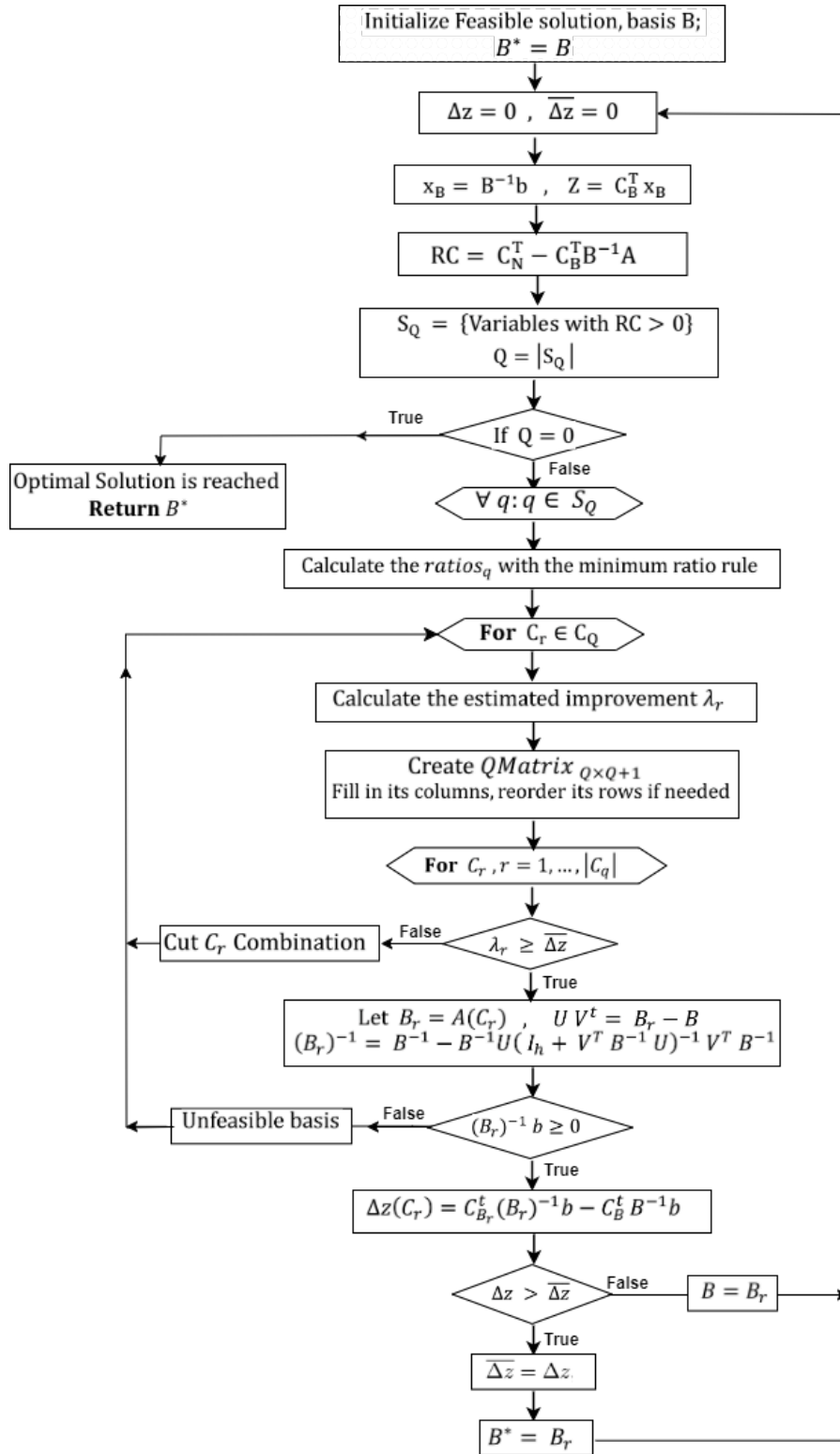


Figure 2.1: The flow chart of Multiple Pivot Simplex algorithm

Herein, we provide a formal proof on the convergence of the proposed Multiple Pivot Simplex as described in the algorithm pseudocode [1](#). The Main theorem proof follows one main assumption and three propositions.

Assumption 1 *We assume that LPS is a feasible and bounded linear programming problem in standard form:*

$$\begin{aligned}
 (LPS) \quad & \max z = C^t x \\
 & \text{subject to } Ax = b \\
 & x \geq 0.
 \end{aligned}$$

Proposition 1 *At each iteration of the Multiple Simplex algorithm, the current basis B_r represents a feasible solution.*

Proof.

During the algorithm:

- The process selects a new candidate basis B_r by swapping variables, checking the feasibility condition $(B_r)^{-1}b \geq 0$. If this condition holds, the new basis B_r is feasible.
- If a basis is infeasible, the algorithm discards it and does not proceed with that combination. This ensures that the algorithm only operates on a feasible basis.

Consequently, the algorithm always maintains feasibility, ensuring that all iterations respect the constraints of the original problem. |

Proposition 2 *Each new iteration of the Multiple Simplex algorithm indicates a potential improvement in the objective function.*

Proof. The algorithm computes the reduced costs $RC = C_N^T - C_B^T B^{-1} A$ and selects variables in S_Q , which have a positive reduced cost ($RC > 0$). This indicates potential improvement in the objective function.

- In Step 2, for each combination $C_r \in C_Q$, the algorithm calculates the expected improvement λ_r and updates the objective improvement $\overline{\Delta z}$ if the new objective increase Δz is larger.
- If a combination results in a better basis with a higher objective value, the algorithm updates the best basis B^* .
- Since the algorithm only accepts improving solutions and keeps track of the best basis, the objective value improves (or remains the same) at each iteration.

This ensures that the objective function is non-decreasing throughout the algorithm.

■

Proposition 3 *The Multiple Simplex algorithm is guaranteed to terminate in a finite number of steps.*

Proof. The set of possible basic feasible solutions in a linear programming problem is finite. Each basic feasible solution corresponds to a basis, and the total number of possible bases is limited by the number of variables and constraints.

- In every iteration, the algorithm checks for optimality by evaluating the reduced costs. If no improving directions remain (i.e., $Q = 0$, all reduced costs are non-positive), the algorithm terminates in Step 3 and returns the best basis B^* , which corresponds to the optimal solution.

- Since there are finitely many basic feasible solutions, and each iteration either improves the objective or terminates, the algorithm is guaranteed to terminate in a finite number of steps.

■

Theorem 2.1 (Convergence of the Multiple Pivot Simplex Algorithm)

Under assumption 1, the Multiple Pivot Simplex Algorithm, as described in the pseudocode of Algorithm 1, converges to the optimal solution of LPS in a finite number of iterations.

Proof. The Multiple Pivot Simplex Algorithm terminates with an optimal solution because:

- It only considers feasible bases at each iteration (Proposition 1).
- The objective function improves or stays the same at each step (Proposition 2).
- The number of possible bases is finite, which ensures finite termination (Proposition 3).

Thus, the algorithm converges to the optimal solution in a finite number of iterations.

■

The tree representation of Example 1 is shown in Figure 4.2, demonstrating the trace of the Multiple Pivot Simplex algorithm. This representation begins with an initial feasible solution at the root node, identifying the corresponding initial basis, the value of z , and the value of Δz , the reduced costs for the non-basic variables as

listed. The Set S_Q represents the set of candidate variables that could enter the basis, while C_q shows all possible combinations of entering variables $\{x_1\}$, $\{x_2\}$, $\{x_3\}$, and all possible pairs of entering variables. However, introducing x_1, x_2 , and x_3 into the basis simultaneously is infeasible, as x_5 would be the same departing variable when both x_1 and x_2 enter the basis.

At each node, the tree diagram shows whether $\overline{\Delta z}$ and the best basis are updated. In the case where $\lambda_r < \overline{\Delta z}$, the algorithm stops, as no improvement can be made, while in the other case, it continues until reaching the optimal solution or finding new updates for the basis. In this example, the algorithm explores all possible ways of introducing variables from the non-basic set to achieve the optimal solution. It starts by selecting x_1 as the entering variable, with $\lambda_1 = 2$, which effectively updates the value of $\overline{\Delta z}$ to 2.

Next, the algorithm considers introducing x_1 and x_3 simultaneously as entering variables. In this case, $\lambda_{1,3} = 3.33$, which is greater than the previous $\overline{\Delta z}$ value, prompting an update to $\overline{\Delta z} = 3.33$, along with a basis update.

After that, to allow x_2 to enter the basis, $\overline{\Delta z}$ resets to zero as we move back to the root node, this means $\overline{\Delta z} = 0$ again. When the algorithm introduces x_2 as the entering variable with $\lambda_2 = 3$, the value of $\overline{\Delta z}$ will be updated to $\overline{\Delta z} = 3$.

In the following step, x_2 and x_3 are introduced together as entering variables with $\lambda_{2,3} = 3.66$, which is greater than $\overline{\Delta z} = 3$. Therefore, the algorithm updates both $\overline{\Delta z}$ and the basis.

At this stage, all RC values are negative, indicating that the algorithm has reached the optimal solution.

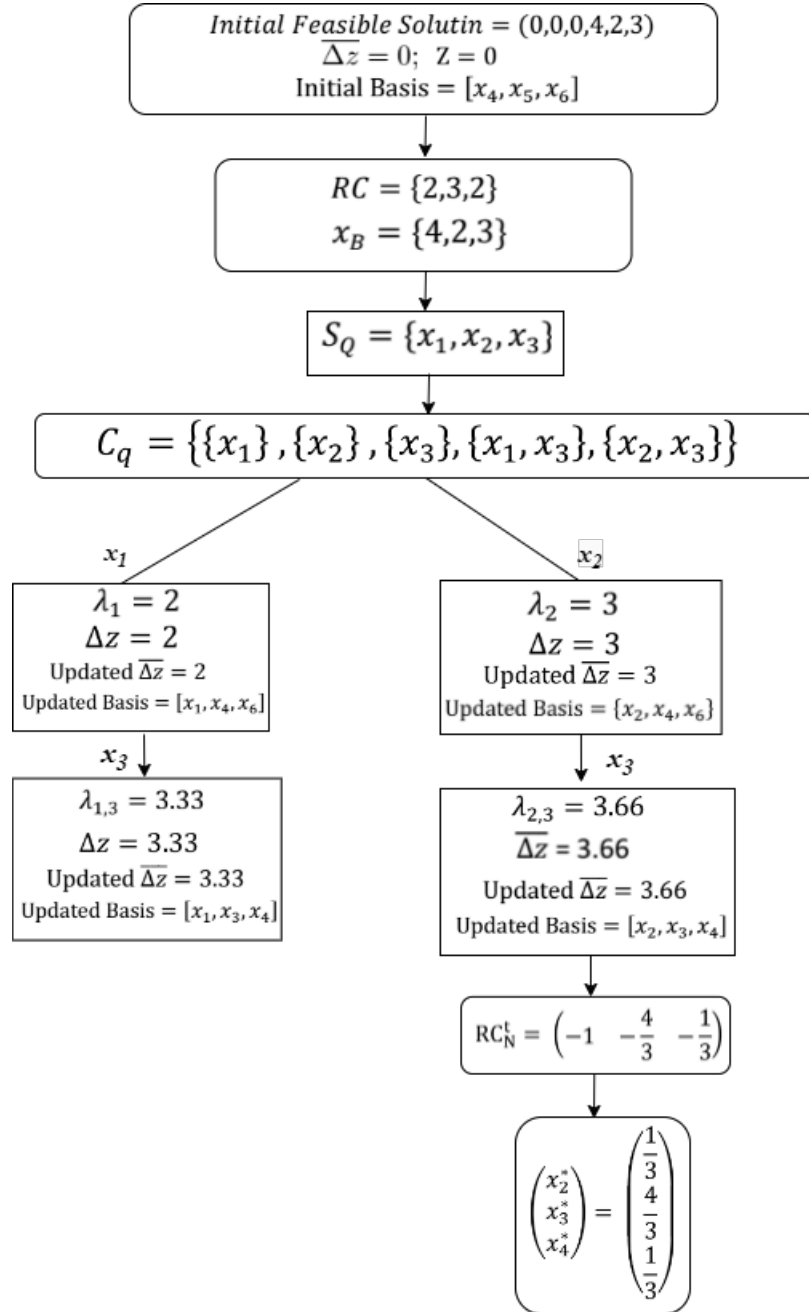


Figure 2.2: Tree diagram of example 1

2.4 Computational Results

A Double Pivpt Simplex algorithm is implemented in Python 3 using Jupyter Notebook (Jupyter Project). The computations were carried out on 64-bit workstations equipped with 2.9 GHz Intel Core i7-7600 processors and 8 GB of RAM, running Windows 11. Its performance was tested on 25 test problems, which are detailed in Table 2.6. The number of variables n varies between 3 and 50, while the number of constraints m ranges from 3 to 50. The "size" refers to the total number of entries in the corresponding LPS form A matrix, calculated as $m \times (n + m)$, which varies between 18 and 4100. The columns "Iter" and "Time" indicate respectively the number of iterations and time (in seconds) taken by the state-of-the-art parallelized simplex implementation by Huangfu and Hall [38]. For each problem, the optimal objective value is denoted as Z^* .

Tables 2.7 to 2.10 present the results of four versions (Version 1 to Version 4) of the double simplex algorithm tested on the 25 test problems, where:

- **Version 1:** Double Simplex Algorithm with Order and with Cuts.
- **Version 2:** Double Simplex Algorithm with No Order and with Cuts.
- **Version 3:** Double Simplex Algorithm with Order and No Cuts.
- **Version 4:** Double Simplex Algorithm with No Order and No Cuts.

Where:

- **Cuts:** mean that at the node if $\lambda_r < \overline{\Delta z}$, which means no improvement can be made, the algorithm stops.

► **Order:** means reordering QMatrix rows in decreasing order of λ_r .

The "Cuts" column indicates the number of cuts ($\lambda < \Delta \bar{z}$) performed during the execution of the algorithm. The "Inv" column denotes the number of matrix inversions carried out, where a fractional inversion equal to $\frac{1}{m^2}$ is added when a double pivot is performed. The "Inf" column shows the number of infeasible nodes explored, while the "Nodes" column indicates the total number of nodes examined during the algorithm's execution. The "Nodes 1" column specifies the number of nodes explored involving a single variable pivot, and the "Nodes s" column details the number of nodes explored with two-variable pivots. The "Iter" column represents the total number of iterations executed, and finally, the "Time" column indicates the total time, in seconds, taken by the algorithm to reach the optimal solution.

Problem	n	m	Size	Iter	Time	z^*
Ex1	3	3	18	3	0.001985	3.667
Ex2	3	3	18	5	0.000482	3.667
Ex3	7	3	30	5	0.000535	12
Ex4	10	6	96	11	0.000681	6.397
Ex5	5	3	24	5	0.000568	5
Ex6	6	7	91	7	0.000540	12.367
Ex7	4	5	45	8	0.000480	706
Ex8	6	8	112	9	0.000497	12.178
Ex9	5	3	24	5	0.000389	23.222
Ex10	10	7	119	13	0.000498	44.581
Ex11	15	20	700	28	0.000958	42.214
Ex12	12	8	160	13	0.000685	48.781
Ex13	14	10	240	18	0.000690	61.822
Ex14	16	14	420	19	0.000821	16.774
Ex15	18	25	1075	38	0.001250	48.048
Ex16	23	28	1428	54	0.001621	43.605
Ex17	20	10	300	14	0.000725	23.35
Ex18	26	20	920	34	0.001249	21.522
Ex19	24	11	385	16	0.000785	21.375
Ex20	30	32	1984	42	0.001929	261.25
Ex21	34	40	2960	49	0.002477	293.275
Ex22	40	26	1716	49	0.002216	401.2
Ex23	36	32	2176	48	0.002246	192
Ex24	50	42	3864	64	0.003545	230
Ex25	32	50	4100	89	0.003390	248.612
Average	17.96	16.64	454.36	25.84	0.001250	111.3163

Table 2.6: Test Problems Details

Problem	Cuts	Inv	Inf	Nodes	Nodes1	Nodes2	Iter	Time
1	1	3.556	0	6	3	3	2	0.0016
2	1	3.667	0	8	4	4	2	0.0014
3	1	5.444	4	26	8	18	2	0.0038
4	5	4.180	6	20	6	14	3	0.0035
5	3	3.667	0	7	3	4	2	0.0012
6	5	5.449	8	29	10	19	4	0.0044
7	3	5.400	2	10	6	4	4	0.0017
8	2	2.188	5	14	4	10	1	0.0022
9	3	5.444	3	16	7	9	3	0.0024
10	4	3.796	5	48	10	38	2	0.0071
11	3	6.245	73	114	22	92	5	0.0187
12	3	2.734	32	57	9	48	1	0.0082
13	4	4.800	32	94	23	71	3	0.0140
14	9	6.597	86	138	23	115	5	0.0204
15	8	7.238	94	172	26	146	6	0.0316
16	29	9.318	185	281	39	242	8	0.0505
17	6	4.620	151	187	19	168	2	0.0257
18	18	7.763	239	344	40	304	6	0.0551
19	13	8.826	273	381	46	335	5	0.0533
20	18	6.242	99	267	46	221	5	0.0522
21	17	4.173	121	293	29	264	3	0.0889
22	15	5.828	234	581	61	520	4	0.1083
23	12	6.201	78	231	38	193	5	0.0459
24	49	3.006	0	10	5	5	2	0.0074
25	6	4.058	27	172	34	138	3	0.0741
Average	9.52	5.218	70.28	140.24	20.84	119.4	3.52	0.0273

Table 2.7: Version 1 - Double Simplex Algorithm Results with Order and with Cuts

Problem	Cuts	Inv	Inf	Nodes	Nodes1	Nodes2	Iter	Time
1	1	3.444	0	5	3	2	2	0.0011
2	1	3.667	0	8	4	4	2	0.0016
3	0	4.111	4	23	7	16	1	0.0033
4	5	4.160	5	17	6	11	3	0.0042
5	3	2.333	0	6	2	4	1	0.0010
6	2	4.367	8	22	8	14	3	0.0032
7	2	4.360	2	9	6	3	3	0.0014
8	2	3.219	2	18	6	12	2	0.0028
9	1	5.222	3	14	6	8	3	0.0021
10	4	4.980	6	61	14	47	3	0.0091
11	8	3.233	40	108	17	91	2	0.0186
12	6	3.766	32	62	12	50	2	0.0110
13	2	4.660	13	91	25	66	3	0.0159
14	10	6.750	76	167	30	137	5	0.0262
15	5	7.218	66	160	27	133	6	0.0323
16	20	9.328	191	311	44	267	8	0.0626
17	8	8.050	145	232	31	201	5	0.0324
18	18	7.763	239	344	40	304	6	0.0551
19	6	7.826	149	249	33	216	5	0.0348
20	17	4.183	12	204	29	175	3	0.0413
21	10	3.163	15	278	27	251	2	0.0880
22	24	4.714	283	521	49	472	3	0.1063
23	7	3.160	24	187	32	155	2	0.0472
24	46	2.002	0	4	4	0	1	0.0023
25	5	4.052	4	158	37	121	3	0.0738
Average	8.52	4.789	52.76	130.36	19.96	110.4	3.16	0.0271

Table 2.8: Version 2 - Double Simplex Algorithm Results with No Order and with Cuts

Problem	Cuts	Inv	Inf	Nodes	Nodes1	Nodes2	Iter	Time
1	0	3.556	0	7	4	3	2	0.0012
2	0	3.667	0	9	5	4	2	0.0014
3	0	6.667	4	28	10	18	3	0.0041
4	0	4.210	6	26	11	15	3	0.0051
5	0	4.000	0	11	6	5	2	0.0015
6	0	5.490	9	35	14	21	4	0.0052
7	0	5.440	2	13	9	4	4	0.0020
8	0	2.203	6	17	6	11	1	0.0026
9	0	5.667	4	19	9	10	3	0.0027
10	0	3.878	7	54	14	40	2	0.0078
11	0	6.248	73	117	25	92	5	0.0185
12	0	2.750	32	62	12	50	1	0.0083
13	0	4.810	32	99	27	72	3	0.0159
14	0	6.628	88	151	32	119	5	0.0231
15	0	7.250	94	186	34	152	6	0.0363
16	0	12.412	197	365	81	284	11	0.0701
17	0	4.670	151	194	25	169	2	0.0256
18	0	7.808	246	373	55	318	6	0.0576
19	0	8.959	285	406	57	349	5	0.0548
20	0	6.267	103	296	62	234	5	0.0554
21	0	4.195	133	339	46	293	3	0.1060
22	0	5.868	234	609	76	533	4	0.1183
23	0	6.211	79	249	50	199	5	0.0558
24	0	3.057	0	105	54	51	2	0.0349
25	0	4.063	32	187	40	147	3	0.0884
Average	0	5.439	72.68	158.28	30.56	127.72	3.68	0.0321

Table 2.9: Version 3 - Double Simplex Algorithm Results with Order and No Cuts

Problem	Cuts	Inv	Inf	Nodes	Nodes1	Nodes2	Iter	Time
1	0	3.444	0	6	4	2	2	0.0011
2	0	3.667	0	9	5	4	2	0.0021
3	0	4.111	4	23	7	16	1	0.0039
4	0	4.180	5	23	11	12	3	0.0036
5	0	2.444	0	9	5	4	1	0.0016
6	0	4.388	8	24	10	14	3	0.0035
7	0	4.400	2	11	8	3	3	0.0017
8	0	3.219	2	20	8	12	2	0.0032
9	0	5.333	3	15	7	8	3	0.0022
10	0	5.000	6	65	18	47	3	0.0101
11	0	3.268	43	126	25	101	2	0.0207
12	0	3.844	32	68	18	50	2	0.0092
13	0	4.670	13	93	27	66	3	0.0136
14	0	6.811	81	184	40	144	5	0.0290
15	0	7.224	67	166	32	134	6	0.0350
16	0	9.351	192	333	64	269	8	0.0593
17	0	8.100	147	243	39	204	5	0.0331
18	0	7.808	246	373	55	318	6	0.0576
19	0	5.777	144	247	35	212	3	0.0343
20	0	5.202	12	236	49	187	4	0.0469
21	0	3.168	15	288	37	251	2	0.0891
22	0	4.750	283	552	73	479	3	0.1044
23	0	3.165	24	196	39	157	2	0.0438
24	0	2.028	0	50	50	0	1	0.0141
25	0	4.055	5	168	42	126	3	0.0770
Average	0	4.776	53.36	141.12	28.32	112.8	3.12	0.0280

Table 2.10: Version 4 - Double Simplex Algorithm Results with No Order and No Cuts

A close examination reveals distinct differences in the algorithm's behavior across the versions. Version 2 (with no order and with cuts) stands out as the fastest, with an average execution time of 0.0271 seconds, outperforming Version 1 (with order and cuts) and Version 4 (with no order and no cuts), which have average times of 0.0273 seconds and 0.028 seconds, respectively (Figure 2.3). This suggests that Version 2 is more efficient in processing the test problems. In terms of the number of inversions (Inv), Version 4 demonstrates superior performance with an average of 4.776 inversions, which is lower than the 4.789 and 5.218 averages seen in the second and first versions (Figure 2.4). The reduced number of inversions indicates that Version 4 achieves better numerical stability and fewer updates to the inverse matrix.

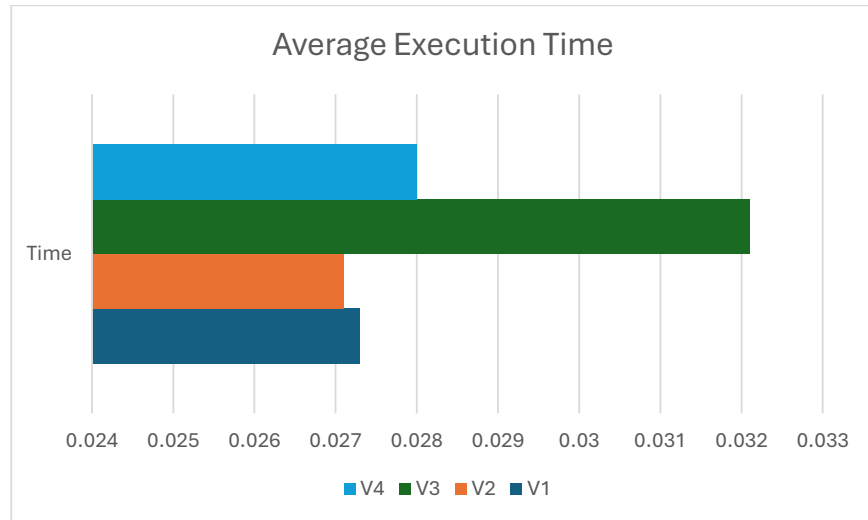


Figure 2.3: Comparative Performance I

When considering the number of iterations (Iter), all four versions show similar

performance, with averages around 3.52 for Version 1, 3.16 for Version 2, 3.68 for Version 3, and 3.12 for Version 4. While these differences in iteration count are relatively small, they still suggest that Version 4 achieves convergence slightly more efficiently on average (Figure 2.5). Additionally, the second version with no order consistently shows fewer cuts (Cuts) and infeasibilities, than the first version with order and cuts, indicating that this version potentially finds feasible solutions with fewer adjustments (Figure 2.6).

Overall, Version 2 demonstrates a combination of faster execution time, fewer inversions, fewer nodes explored (Figure 2.7), and comparable iteration counts, making it the most efficient version across the tested metrics.

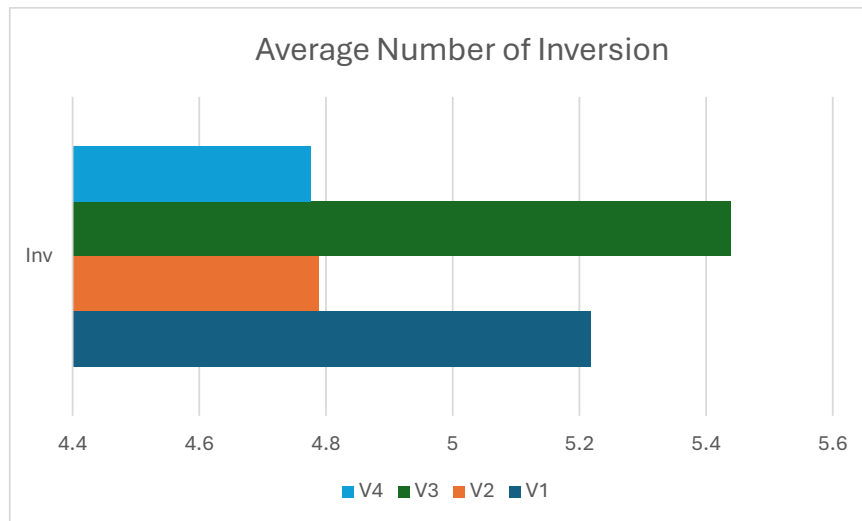


Figure 2.4: Comparative Performance II

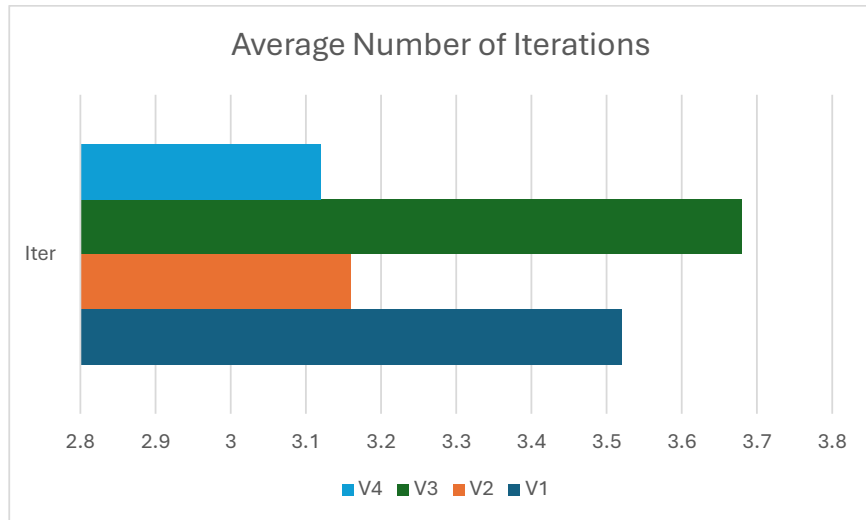


Figure 2.5: Comparative Performance III

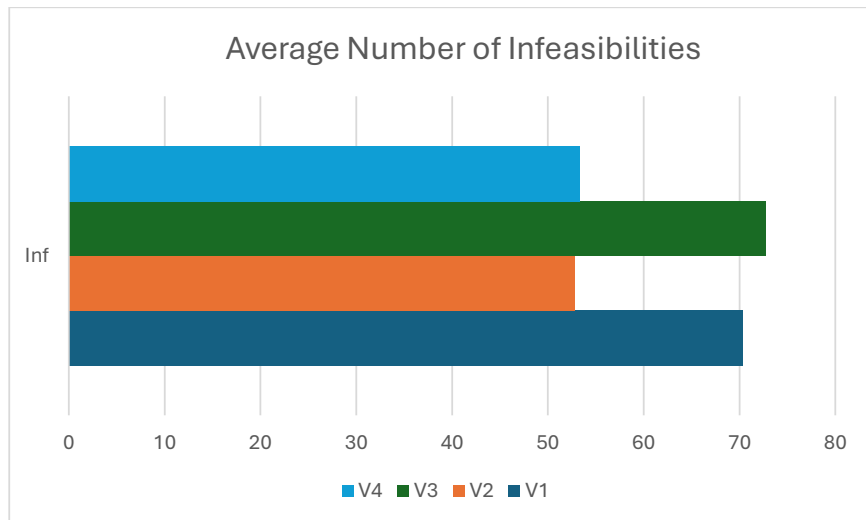


Figure 2.6: Comparative Performance III

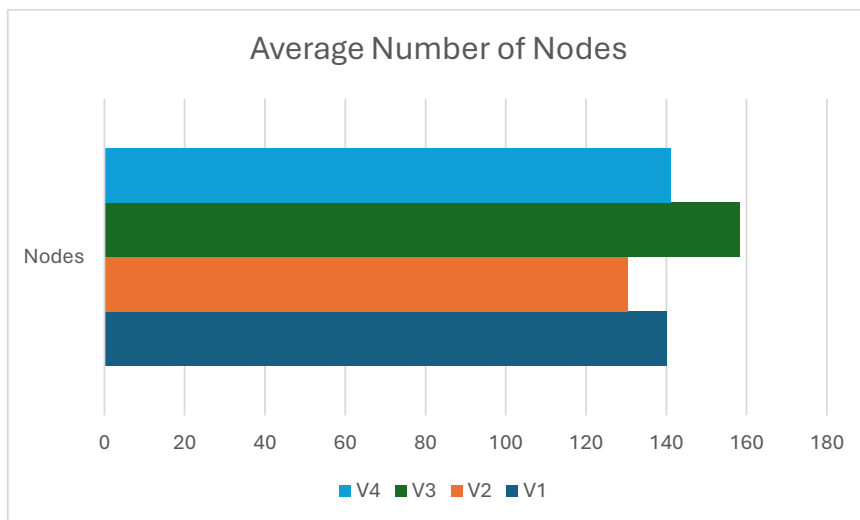


Figure 2.7: Comparative Performance IV

2.5 Conclusion

Over the years, researchers have worked to advance Dantzig's ideas, contributing to the widespread use of linear programming as a key optimization technique in various fields. In addition to exploring the theoretical properties of the simplex method, researchers have focused on developing powerful computer codes, crucial for improving algorithmic and computational techniques, thereby contributing to the success of the Simplex method. Modified simplex methods represent enhancements of the original simplex algorithm. The modified versions incorporate generally a more efficient implementation. Meanwhile, these modifications can also tackle various aspects of the algorithm, including the selection of entering and leaving variables, the choice of pivot

rules, or the handling of degeneracy.

In this chapter, we introduced a multiple-pivot simplex method that enhances one of the most well-known and effective algorithms in science. Unlike the traditional Simplex Method, which pivots on a single variable at each iteration, the Multiple Pivot Simplex Method pivots on two variables simultaneously. During its development, we adapted two types of cut statuses and two types of ordering statuses, leading to four versions of the proposed algorithm limited to double pivots. These versions were evaluated using randomly generated problems ranging from sizes 18 to 4,100.

The computational results indicate that Version 2, characterized as "No Order with Cut," is the fastest in identifying the optimal basis and solution. Additionally, the findings suggest that the proposed method yields promising results for larger problems.

The possible extensions of this work will include the implementation of a Triple Pivot implementation of the proposed algorithm, which can be tested on more extensive sets of benchmark instances.

CHAPTER 3

A MODIFIED PSEUDO- INVERSE NEWTON-RAPHSON ALGORITHM FOR ARTIFICIAL NEURAL NETWORKS

3.1 Introduction

Minimizing non-linear functions is crucial in a wide range of applications, as many real-world systems exhibit complex, non-linear relationships that cannot be effectively captured by linear models. Non-linear optimization allows for more accurate modeling of phenomena in fields such as machine learning, engineering, finance, and logistics, where capturing intricate dependencies and interactions among variables is essential. In machine learning, for instance, optimizing non-linear error functions enhances

model accuracy and robustness, leading to more reliable predictions. In engineering design, non-linear optimization aids in refining complex system parameters for improved performance and efficiency. Overall, minimizing non-linear functions is a foundational aspect of modern optimization that drives advancements and innovation across diverse industries.

The minimization of error functions within artificial neural networks (ANN) is of major importance. To do so, different activation (transfer) functions are commonly used within the different layers of the network. The input received $p^t = (p_1^t, \dots, p_r^t, \dots, p_R^t, 1)$ is used to approximate the measure of the output y^t at time or observation index t . To avoid any confusion with the time or observation index t , the transpose notation here used is $'$.

Common ANN activation functions, for instance the *logistic sigmoid* function $f(x) = \frac{1}{1+e^{-x}}$, assume that

$$x^t = w \cdot (p^t)' = \sum_{r=1}^R w_r p_r^t + b,$$

where $w = (w_1, \dots, w_r, \dots, w_R, b)$ gathers the input weights and the bias b . The neurons at every layer l ($l = 1 \dots L$) of the ANN are dotted with their activation function $f_l(\cdot)$. The output estimation following one complete forward pass through the ANN is then

$$\hat{y}^t = F(p^t) = f_L(\dots f_l(\dots f_1(p^t))),$$

where F is the obtained composed function. Hence, the total estimation error function

obtained for each output following one full forward pass through the L layers of the ANN is given by

$$E(T) = \sum_{t=1}^T (y^t - \hat{y}^t)^2 = \sum_{t=1}^T (y^t - F(p^t))^2.$$

Gradient methods are widely used to optimize differentiable functions with or without constraints. Gradient methods are generally based on information extracted from first and second order partial derivatives. Within a classic neural network gradient descent scheme, the weights are updated after each forward pass h following the sequence

$$\begin{aligned} w_r^{h+1} &= w_r^h - \mu \frac{1}{\|\nabla E(T)\|} \frac{\partial E(T)}{\partial w_r}, \\ b^{h+1} &= b^h - \mu \frac{1}{\|\nabla E(T)\|} \frac{\partial E(T)}{\partial b}. \end{aligned}$$

In vector form, the classic gradient descent scheme writes as

$$w^{h+1} = w^h - \frac{\mu}{\|\nabla E(T)\|} \cdot \nabla E(T).$$

While the derivation of the classic gradient descent algorithm is based on the first-order Taylor series expansion of $E(x_{h+1})$. Newton-Raphson's method is based on the second-order Taylor series

$$E(x_{h+1}) = E(x_h + \Delta x_h) \approx E(x_h) + \nabla E(x_h)^t \Delta x_h + \frac{1}{2} \Delta x_h^t H(x_h) \Delta x_h,$$

where $\Delta x_h = (x - x_h)$ and $H(x_h)$ is the Hessian matrix of $E(x_h)$. The principle behind Newton-Raphson's method is to locate a critical point of this quadratic approximation of $E(x)$. The first order condition yields

$$\nabla E(x_h) + H(x_h)\Delta x_h = 0.$$

Which leads to the original Newton-Raphson's method where

$$x^{h+1} = x^h - H(x_h)^{-1}\nabla E(x_h).$$

The convergence of the Newton-Raphson algorithm is a crucial aspect of its effectiveness. The Newton-Raphson method exhibits local convergence, meaning that its effectiveness depends on the proximity of the initial guess to the exact root. The algorithm tends to converge rapidly when the initial guess is sufficiently close to the root. When the Newton-Raphson algorithm converges, it typically does so quadratically. This implies that, with each iteration, the number of correct digits roughly doubles. The quadratic convergence makes the Newton method highly efficient compared to linearly converging methods.

Meanwhile, convergence is not guaranteed for all functions or initial guesses. The function must be differentiable, and the initial guess should be reasonably close to a root. In some cases, the algorithm may fail to converge or converge to a root that is not of interest (if there are multiple roots).

Hence, while the Newton-Raphson method is locally convergent, its global conver-

gence is not assured. There are cases where the algorithm may diverge or get stuck in cycles, especially if the initial guess is far from any root or the Hessian matrix is singular or quasi-singular.

This chapter proposes a modified Newton-Rapshon scheme based on pseudo-inverse matrices to overcome such issues.

3.2 Newton-Raphson methods: Literature Review

The Newton method and its extensions are widely acknowledged as the most popular iterative techniques for solving nonlinear equations of the form

$$\nabla E(x) = F(x) = 0.$$

This acclaim stems from their simplicity and swift convergence properties. This iterative process generates a sequence of approximations $\{w^k\}$ that converge quadratically to the root w^* .

3.2.1 The general Newton method

The general form of Newton method can be written as following :

$$x_{h+1} = x_h - H^{-1}(x_h)\nabla E(x_h). \quad (3.1)$$

The Newton method serves as the foundation for many fast, local algorithms designed to solve smooth equations. These algorithms exhibit excellent convergence rates,

provided the initial point is chosen within an appropriate neighborhood around the target solution.

The original method, introduced by Newton in 1669, was specifically devised to address polynomial functions. However, it was J. Raphson in 1690 who proposed the general form of the method.

Significant advancements in this method were made by Fourier, Cauchy, and others. In 1818, Fourier demonstrated that in the proximity of a root, the method exhibits quadratic convergence. Cauchy presented the multidimensional extension of equations in 1829. Moreover, in 1847, Cauchy utilized this method to establish the existence of a root for an equation. Later, in 1916, Fine [39] explored the convergence in the n -dimensional case, and Bennet [40] extended this result to the infinite-dimensional case.

Kantorovich [41] established the local linear convergence of the original Newton-Raphson method. He later refined this result to encompass local quadratic convergence and introduced a new method named the Newton-Kantorovich method, extending Newton's method for functional spaces as described in [42]. Around the same period, Mysovskikh [43] demonstrated local quadratic convergence algorithms with slightly different theoretical assumptions.

Other notable contributions include the convergence theorems by Ortega and Rheinboldt [44] and the affine invariant theorems by Deuffhard [45], [46], [47].

In the context of rectangular systems, extending the Newton approach poses challenges due to the rectangular nature of the Jacobian matrix, rendering its inverse undefined. Consequently, the Pseudo-inverse of the Jacobian becomes a viable alter-

native to Newton Methods.

3.2.2 Generalized-Inverse Newton-Raphson methods (GINR)

The generalized Newton method suggests employing the generalized Jacobian of F , i.e., the generalized inverse of $H()$ noted H^+ , instead of F' , i.e., instead of the inverse of $H()$ in the Newton method, and using it for convergence analysis. The generalized inverse Newton-Raphson method (GINR) can be formulated as following:

$$x_{h+1} = x_h - H^+(x_h)\nabla E(x_h). \quad (3.2)$$

One of the pioneering works on pseudo-inverse Newton methods is credited to Ben-Israel [48]. His contributions in this area are significant. Allgower *et al.* [49] proposed Newton-type methods, utilizing the generalized inverse or the pseudo-inverse (the Moore-Penrose) in place of J^{-1} . Chen *et al.* [50] conducted a local convergence analysis of generalized Newton methods for singular, smooth, and non-smooth operator equations.

Dedieu and Shub [51] delved into Newton's method for over-determined systems, where the number of equations exceeds the number of unknown variables. They also integrated the concept of the pseudo-inverse of the chosen linear operator. Levin and Ben-Israel [52] examined the conditions for the local convergence of pseudo-inverse Newton and established quadratic convergence.

Galántai [53] contributed results on the Newton method, covering aspects of convergence, numerical stability, computational complexity, and error estimates. His

work addressed convergence considerations for both smooth and non-smooth equations, including under-determined equations and equations with singular Jacobians.

Qi and Sun [54] extended the Newton method for solving non-linear equations to non-smooth cases by utilizing the generalized Jacobian instead of the derivative. They established convergence theorems under the condition of semi-smoothness. Additionally, Newton methods employing generalized inverses and pseudo-inverses were investigated by Deuffhard and Heindl [46], Nashed and Chen [55], Walker [56], Decker and Kelley [57], among others.

Conventional studies on the convergence of Newton methods typically focus on two dimensions semi-local and local convergence. Semi-local convergence relies on information around an initial point to establish criteria ensuring the method's convergence. Conversely, local convergence is based on information around a solution w^* to provide estimates of the radii of convergence balls.

3.3 The Modified Pseudo-Inverse Newton-Raphson Method (MPINR)

Before detailing our proposed Modified Pseudo-Inverse Newton-Raphson (MPINR) method, we need to review the main definitions and rules on the computation and updating of the pseudo-inverse matrix H^+ . The successful application of the proposed MPINR relies on the efficient computation and updating of H^+ , more especially if the method is to be integrated within a neural network.

3.3.1 The Moore-Penrose inverse or pseudo-inverse

The Moore-Penrose inverse, also known as the pseudo-inverse, arises from the need to generalize matrix inversion for non-square or singular matrices. In many applications, such as solving systems of linear equations, finding least-squares solutions, we encounter matrices that are not invertible in the traditional sense. The Moore-Penrose inverse provides a way to extend the concept of a matrix inverse to a broader class of matrices. It allows us to compute a "best approximate inverse" even for non-square or singular matrices. This is particularly useful when dealing with overdetermined or underdetermined systems of linear equations.

Throughout this chapter, we use the following standard notation:

- $A \in \mathbb{C}^{m \times n}$: A complex matrix with m rows and n columns
- A^* : The conjugate transpose of A
- $\mathcal{R}(A)$: The range (column space) of A
- $\mathcal{N}(A)$: The null space of A
- $E^\perp$ consists of all vectors that are orthogonal to every vector in E , In other words, if $\mathbf{v} \in E^\perp$, then for every $\mathbf{u} \in E$, the dot product (or inner product in the case of complex vector spaces) satisfies: $\mathbf{u} \cdot \mathbf{v} = 0$

The Moore-Penrose inverse was independently defined by E.H. Moore in 1920 and Roger Penrose in 1955.

Definition 3.1 Let $A \in \mathbb{C}^{m \times n}$ be a complex matrix. The Moore-Penrose inverse A^+ is the unique matrix satisfying the following four conditions:

$$(i) \quad AA^+A = A$$

$$(ii) \quad A^+AA^+ = A^+$$

$$(iii) \quad (AA^+)^* = AA^+$$

$$(iv) \quad (A^+A)^* = A^+A$$

where $*$ denotes the conjugate transpose.

These conditions are often referred to as the Moore-Penrose equations or Penrose conditions.

Properties: The Moore-Penrose inverse possesses several important properties. For any matrix A :

$$(P1) \quad (A^+)^+ = A$$

$$(P2) \quad (A^*)^+ = (A^+)^*$$

$$(P3) \quad (kA)^+ = \frac{1}{k}A^+ \text{ for any non-zero scalar } k$$

$$(P4) \quad (A^*A)^+ = A^+(A^*)^+$$

$$(P5) \quad AA^+ \text{ and } A^+A \text{ are Hermitian}$$

$$(P6) \quad AA^+ \text{ is the orthogonal projector onto } \mathcal{R}(A), \text{ the range of } A$$

$$(P7) \quad I - AA^+ \text{ is the orthogonal projector onto } \mathcal{N}(A^*), \text{ the null space of } A^*$$

$$(P8) \quad \mathcal{R}(A) = \mathcal{R}(AA^+) = \mathcal{R}(AA^*)$$

$$(P9) \quad \mathcal{R}(A^+) = \mathcal{R}(A^*) = \mathcal{R}(A^+A) = \mathcal{R}(A^*A)$$

$$(P10) \quad \mathcal{R}(I - AA^+) = \mathcal{N}(AA^+) = \mathcal{N}(A^*) = \mathcal{N}(A^+) = \mathcal{R}(A)^\perp$$

$$(P11) \quad \mathcal{R}(I - A^+A) = \mathcal{N}(A^+A) = \mathcal{N}(A) = \mathcal{R}(A^*)^\perp$$

3.3.2 Computing the pseudo-inverse matrix H^+

If H is invertible then H^+ is simply H^{-1} . Otherwise, the following rules need to be applied

- Rule 1: If H has independent columns, then $H^+ = (H^t H)^{-1} H^t$ and so $H^+ H = I$.
- Rule 2: If H has independent rows, then $H^+ = H^t (H H^t)^{-1}$ and so $H H^+ = I$.
- Rule 3: The pseudoinverse of $H = U \Sigma V^t$ is $H^+ = V \Sigma^+ U^t$, where the diagonal matrix Σ is inverted where possible. That is; Σ^+ is diagonal and its diagonal elements are calculated by

$$\Sigma_{ii}^+ = \begin{cases} \frac{1}{\Sigma_{ii}}, & \text{if } \Sigma_{ii} \neq 0, \\ 0, & \text{if } \Sigma_{ii} = 0, \end{cases}$$

For example, with

$$\Sigma = \begin{pmatrix} \sigma_1 & 0 & 0 & 0 \\ 0 & \sigma_2 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}, \text{ we get } \Sigma^+ = \begin{pmatrix} 1/\sigma_1 & 0 & 0 & 0 \\ 0 & 1/\sigma_2 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

Example 1

Let us assume that we have to minimize

$$f(x_1, x_2) = 2x_1^2 + \frac{1}{2}x_2^2 + 2x_1x_2.$$

$$\frac{\partial f}{\partial x_1} = 4x_1 + 2x_2 \quad \text{and} \quad \frac{\partial f}{\partial x_2} = 2x_1 + x_2.$$

Meanwhile, the Hessian matrix obtained is not only singular, but its columns as well as its rows are linearly dependent.

$$H = \begin{bmatrix} 4 & 2 \\ 2 & 1 \end{bmatrix}.$$

To compute the pseudo-inverse of H , we will use the Singular Value Decomposition (SVD) approach.

The SVD decomposes H into three matrices:

$$H = U\Sigma V^T,$$

where

- U is an orthogonal matrix containing the left singular vectors.
- Σ is a diagonal matrix with the singular values.
- V is an orthogonal matrix containing the right singular vectors.

In this case, the matrices obtained from the SVD are

$$U = \begin{bmatrix} -\frac{4}{\sqrt{20}} & -\frac{2}{\sqrt{20}} \\ -\frac{2}{\sqrt{20}} & \frac{4}{\sqrt{20}} \end{bmatrix} = \begin{bmatrix} -\frac{2}{\sqrt{5}} & -\frac{1}{\sqrt{5}} \\ -\frac{1}{\sqrt{5}} & \frac{2}{\sqrt{5}} \end{bmatrix},$$

$$\Sigma = \begin{bmatrix} 5 & 0 \\ 0 & 0 \end{bmatrix},$$

$$V = \begin{bmatrix} -\frac{4}{\sqrt{20}} & -\frac{2}{\sqrt{20}} \\ \frac{2}{\sqrt{20}} & -\frac{4}{\sqrt{20}} \end{bmatrix} = \begin{bmatrix} -\frac{2}{\sqrt{5}} & -\frac{1}{\sqrt{5}} \\ \frac{1}{\sqrt{5}} & -\frac{2}{\sqrt{5}} \end{bmatrix}.$$

The pseudo-inverse of the diagonal matrix Σ , denoted as Σ^+ , is given by taking the reciprocal of each non-zero singular value

$$\Sigma^+ = \begin{bmatrix} \frac{1}{5} & 0 \\ 0 & 0 \end{bmatrix}.$$

The pseudo-inverse is given by

$$H^+ = V\Sigma^+U^T = \frac{1}{25} \begin{bmatrix} 4 & 2 \\ 2 & 1 \end{bmatrix}.$$

3.3.3 Convergence conditions

Classic investigations into the convergence of Newton methods traditionally concentrate on two dimensions: semi-local and local convergence. Semi-local convergence utilizes information around an initial point to establish criteria ensuring the method's convergence. In contrast, local convergence is grounded in information surrounding a solution w^* providing estimates of the radii of convergence balls.

Theorem 3.1 *Let $E : \mathbb{R}^n \rightarrow \mathbb{R}$ be a function, let x_0 be a point in $\mathbb{R}^n$, and $r > 0$ such that $E \in \mathcal{C}^2(S(x_0, r))$ where $S(x_0, r)$ is a ball of radius r and center x_0 , and consider the sequence*

$$x_{h+1} = H^+(x_h)H(x_h)x_h - H^+(x_h)\nabla E(x_h) \quad \text{for } h \geq 0. \quad (3.3)$$

If there exist positive constants M , N and R such that

$$M\|H^+(x)\| + N + R \leq k < 1 \quad \text{for all } x \in S(x_0, r) \quad (3.4)$$

and

$$\|x_1 - x_0\| < (1 - k)r, \quad (3.5)$$

$$\|H^+(v)(u - v) - \nabla E(u) + \nabla E(v)\| \leq M\|u - v\|, \quad (3.6)$$

$$\text{and } \|(H^+(u) - H^+(v))\nabla E(u)\| \leq N\|u - v\|, \quad (3.7)$$

$$\|(H^+(u)H(u) - H^+(v)H(v))u\| \leq R\|u - v\|, \quad (3.8)$$

for all $u, v \in S(x_0, r)$, then the sequence $\{x_h\}_{h \geq 1}$ lies in $S(x_0, r)$ and converges to a solution of the equation:

$$H^+(x)\nabla E(x) = 0. \quad (3.9)$$

Proof. To simplify the notations, let $H_h := H(x_h)$, $H_h^+ = H^+(x_h)$, $E_h := E(x_h)$.

First, we prove by induction that

$$x_h \in S(x_0, r) \text{ for all } h \geq 1. \quad (3.10)$$

The case $h = 1$ is guaranteed by (3.5). Let us assume that (3.10) is true for all subscripts $\leq h$, then, using equation (3.3), we get

$$x_{h+1} - x_h = H_h^+ H_h x_h - H_h^+ \nabla E_h - H_{h-1}^+ H_{h-1} x_{h-1} + H_{h-1}^+ \nabla E_{h-1} + x_h - x_h.$$

By definition, notice that $x_h \in \mathcal{R}(H_{h-1}^+)$, hence $H_{h-1}^+ H_{h-1} x_h = x_h$. Therefore,

$$x_{h+1} - x_h = (H_h^+ H_h x_h - H_{h-1}^+ H_{h-1} x_h) + (H_{h-1}^+ H_{h-1} (x_h - x_{h-1}) + H_{h-1}^+ \nabla E_{h-1} - H_h^+ \nabla E_h).$$

Besides, we can rewrite

$$\begin{aligned} H_{h-1}^+ H_{h-1} (x_h - x_{h-1}) + H_{h-1}^+ \nabla E_{h-1} - H_h^+ \nabla E_h &= H_{h-1}^+ [H_{h-1} (x_h - x_{h-1}) - \nabla E_h + \nabla E_{h-1}] \\ &+ (H_{h-1}^+ - H_h^+) \nabla E_h. \end{aligned}$$

For $u = x_h$ and $v = x_{h-1}$ in the (3.6), (3.7), (3.8), we get:

$$\|H_h^+ H_h x_h - H_{h-1}^+ H_{h-1} x_h\| \leq R \|x_h - x_{h-1}\|,$$

$$\|H_{h-1}^+ [H_{h-1}(x_h - x_{h-1}) - \nabla E_h + \nabla E_{h-1}]\| \leq M\|H_{h-1}^+\| \|x_h - x_{h-1}\|,$$

and

$$\|(H_{h-1}^+ - H_h^+)\nabla E_h\| \leq N\|x_h - x_{h-1}\|.$$

Therefore, we can write:

$$\|x_{h+1} - x_h\| \leq (M\|H_{h-1}^+\| + N + R) \|x_h - x_{h-1}\|.$$

Given the condition (3.4), we have:

$$\|x_{h+1} - x_h\| \leq k\|x_h - x_{h-1}\|.$$

Therefore,

$$\|x_{h+1} - x_0\| \leq k^h \|x_1 - x_0\|.$$

Hence, by telescoping series and condition (3.5) we obtain:

$$\|x_{h+1} - x_0\| \leq \sum_{l=0}^h k^l \|x_1 - x_0\| \leq \frac{1}{1-k} \|x_1 - x_0\| \leq r$$

which proves (3.10).

The function

$$N(x) := H^+(x)H(x)x - H^+(x)\nabla E(x)$$

is a contraction from $S(x_0, r)$ to $S(x_0, r)$. By Banach contraction mapping theorem, N has a unique fixed point x^* inside the closure of $S(x_0, r)$, that is,

$$x^* = H^+(x^*)H(x^*)x^* - H^+(x^*)\nabla E(x^*) = H^+(x^*)[H(x^*)x^* - \nabla E(x^*)]. \quad (3.11)$$

The equation (3.11) implies that x^* belongs to $\mathcal{R}(H^+(x^*)) = \mathcal{R}(H^t(x^*))$ (see property (P9)). As H^+H is the projection on $R(H^t)$, we deduce that

$$H^+(x^*)H(x^*)x^* = x^*,$$

Thus

$$H^+(x^*)\nabla E(x^*) = 0.$$

■

Remark: Since $\nabla E(x^*) \in \mathcal{N}(H^+(x^*))$ and $\mathcal{N}(H^+(x^*)) = \mathcal{N}(H^t(x^*))$, then $\nabla E(x^*) \in \mathcal{N}(H^t(x^*))$ and we also have $H^t(x^*)\nabla E(x^*) = 0$.

Example 3

The function defined in example 1 is now minimized with our proposed MPINR method:

$$f(x_1, x_2) = 2x_1^2 + \frac{1}{2}x_2^2 + 2x_1x_2.$$

Starting from the initial point $X_0^t = (3, 0)$, the proposed MPINR method

converges immediately to the point

$$X_1^t = H^+ H X_0^t - H^+ \nabla f(X_0) = \frac{1}{5} \begin{bmatrix} 4 & 2 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} 3 \\ 0 \end{bmatrix} - \frac{1}{25} \begin{bmatrix} 4 & 2 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} 12 \\ 6 \end{bmatrix} = (0, 0).$$

This solution X_1 does not only minimize f , reaching the minimum value $f^* = 0$, but it is also the minimum with the least magnitude, i.e. $\|X_1\| = 0$, as it is naturally the least squares solution.

For comparison, starting from the same point $X_0^t = (3, 0)$, the GNINR method would have generated the point

$$X_1^t = X_0^t - H^+ \nabla f(X_0) = \begin{bmatrix} 3 \\ 0 \end{bmatrix} - \frac{1}{25} \begin{bmatrix} 4 & 2 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} 12 \\ 6 \end{bmatrix} = \left(\frac{3}{5}, -\frac{6}{5}\right).$$

which has definitely a larger magnitude.

Starting from a different initial point $X_0^t = (-1, 1)$, the proposed New Pseudo-Inverse Newton-Raphson method converges again immediately to the same point

$$X_1^t = H^+ H X_0^t - H^+ \nabla f(X_0) = \frac{1}{5} \begin{bmatrix} 4 & 2 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} -1 \\ 1 \end{bmatrix} - \frac{1}{25} \begin{bmatrix} 4 & 2 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} -2 \\ -1 \end{bmatrix} = (0, 0).$$

While the generalized inverse Newton-Raphson method starting from $X_0^t = (-1, 1)$ would have converged to a different point

$$X_1^t = X_0^t - H^+ \nabla f(X_0) = \begin{bmatrix} -1 \\ 1 \end{bmatrix} - \frac{1}{25} \begin{bmatrix} 4 & 2 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} -2 \\ 1 \end{bmatrix} = \left(-\frac{3}{5}, \frac{6}{5}\right).$$

The first example demonstrated that the proposed New Pseudo-Inverse Newton Raphson (NPINR) procedure and the Generalized Inverse Newton Raphson (GINR) procedure can converge to different minimum points, even when starting from the same initial point. The second example shows that the proposed MPINR procedure may converge more quickly than the GINR procedure in cases where the initial point represents a singularity for the Hessian.

Example 4

Let us assume now that we have to minimize

$$f(x_1, x_2) = (x_1 - 2)^4 + (x_2 - 3)^2 + 3x_1^2 + 6x_1x_2 + 2x_2.$$

The absolute minimum of this function lies at

$$x_1^* = 4.1038034 \text{ and } x_2^* = -10.3114102, \text{ with } f^* = -27.212196.$$

This minimum point is displayed in green on Figure [3.1](#).

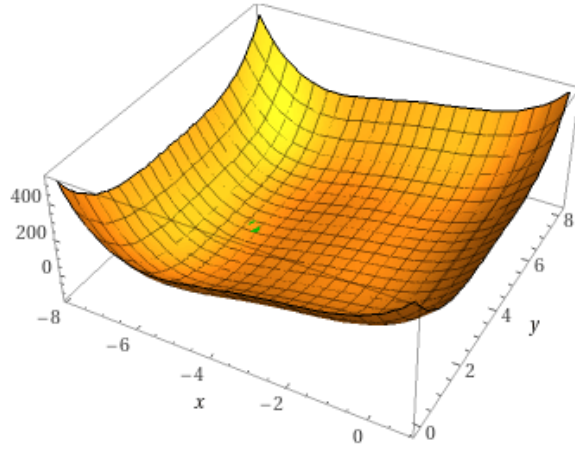


Figure 3.1: 3D Graph - Example 4.

We have here

$$\frac{\partial f}{\partial x_1} = 6x_1 + 6x_2 + 4(x_1 - 2)^3 \text{ and } \frac{\partial f}{\partial x_2} = 6x_1 + 2x_2 - 4.$$

Meanwhile, the Hessian matrix has two singularity points at $x_1 = 1$ and $x_1 = 3$:

$$H = \begin{bmatrix} 12(x_1 - 2)^2 + 6 & 6 \\ 6 & 2 \end{bmatrix}.$$

Starting from the initial point $X_0^t = (1, 0)$, the proposed NPINR method converges after 10 iterations to the point

$$X_{10} = (4.346205, -11.038616), \quad \text{with } f_{10} = -25.8810416.$$

This solution X_{10} does not minimize f yet, but represents a much better solution than the one obtained by the GINR method, which starts from the same initial point

$X_0^t = (1, 0)$ and reaches after 10 iterations the point

$$X_{10} = (5.155287, -13.465862), \quad \text{with} \quad f_{10} = 6.5202561.$$

3.4 Computational Results

A collection of unconstrained non-linear optimization test problems and their corresponding singularity points is listed in Table 3.1. These fifteen test functions are used to validate the performance of the proposed MPINR method with the Generalized Inverse Newton Raphson GINR method. The code is implemented with Julia language and run on a Intel(R) Core(TM) i9-11900H CPU desktop, with 2.50 GHz x64-based processor and 21.0 GB RAM.

Table 3.2 illustrates the behavior of these two methods with various initial points. Each row in this table provides the information about the minimum point value (Min. Point), the number of iterations (No.Iters) required to reach that minimum, and the final optimal point value for the corresponding function. The presence of multiple starting points per function tests the method's robustness, demonstrating that the optimization is effective regardless of the initial points or not. The convergence behavior of each method varies, indicating the need for careful selection of starting points. In the reported results, a tolerance for convergence less or equal to 10^{-6} is used as the stopping condition, and the result are rounded with 4 digit after the decimal.

For almost all chosen starting points, the new MPINR method reaches the optimal minimum points across all instances, while GINR gets stuck at some initial points.

Table 3.1: Functions and Hessian Singularity Points

Functions	Singularity Points
$f_1(x_1, x_2) = (x_1 - 1)^4 + (x_2 - 2)^2 - 2x_1^2 + 4x_1x_2 - 2x_2$	$x_1 = 0$, or $x_1 = 2$
$f_2(x_1, x_2) = x_1^4 + x_2^4$	$x_1 = 0$, or $x_2 = 0$
$f_3(x_1, x_2) = (x_1 - 3)^4 + 2(x_2 - 1)^2 - 4x_1^2 + 4x_1x_2$	$x_1 = 2$, or $x_1 = 4$
$f_4(x_1, x_2) = \frac{1}{4}x_1^4 + x_1^2x_2 + x_2^2$	$x_2 = -\frac{1}{2}x_1^2$
$f_5(x_1, x_2) = x_1^4 + 2x_2^2 - 6x_1x_2$	$x_1 = \pm\frac{\sqrt{3}}{2}$
$f_6(x_1, x_2) = 2(x_1 - 3)^4 + (x_2 - 2)^2 + 4x_1^2 + 4x_1x_2 + 2x_2$	$x_1 = 3$
$f_7(x_1, x_2) = (x_1 + 4)^2 + (x_2 - 3)^4 + x_1x_2$	$x_2 = 3 \pm \frac{1}{2\sqrt{6}}$
$f_8(x_1, x_2) = (x_1 + 4)^2 + (x_2 - 3)^4 + 4x_1 + 6x_1x_2 + 3x_2^2$	$x_2 = 2$, or $x_2 = 4$
$f_9(x_1, x_2) = (x_1 + 1)^2 + (x_2 - 1)^4 + x_2^2 + 2x_1x_2$	$x_2 = 1$
$f_{10}(x_1, x_2) = (x_1 + 1)^2 + \frac{1}{12}(x_2 - 1)^4 + x_1x_2$	$x_2 = 1 \pm \frac{1}{\sqrt{2}}$
$f_{11}(x_1, x_2) = (x_1 + 3)^4 + (x_2 - 4)^4 - 6x_1^2$	$x_1 = -2$, or $x_1 = -4$, or $x_2 = 4$
$f_{12}(x_1, x_2) = (x_1 + 3)^4 + (x_2 - 4)^4 - 6x_2^2$	$x_1 = -3$, or $x_2 = 3$, or $x_2 = 5$
$f_{13}(x_1, x_2) = (x_1 - 3)^4 - x_1^2 + 10x_1x_2 + 5x_2^2$	$x_1 = 2$ or $x_1 = 4$
$f_{14}(x_1, x_2)^{\text{a}} = 2x_1^2 + \frac{1}{2}x_2^2 + 2x_1x_2$	$\mathbb{R}^2$
$f_{15}(x_1, x_2)^{\text{b}} = (x_1 - 2)^4 + (x_2 - 3)^2 + 3x_1^2 + 6x_1x_2 + 2x_2$	$x_1 = 1$, or $x_1 = 3$

^a This function is discussed in Example 3.

^b This function is discussed in Example 4.

Table 3.2: MPINR vs. GINR on functions f_1 to f_{15}

	Start Point	MPINR		GINR		Optimal Point
		Min.Point	No. Iters	Min.Point	No. Iters	
f_1	(2, 0)	(1.0, 1.0)	∞	(2.73205, -2.4641)	8	(2.73205, -2.4641)
	(0, 2)	(-0.73205, 4.4641)	8	(2.73205, -2.4641)	8	or
	(-1, 5)	(-0.73205, 4.4641)	5	(-0.73205, 4.4641)	5	(-0.73205, 4.4641)
f_2	(1, 0)	(0, 0)	23	(0, 0)	23	(0, 0)
	(-1, 0)	(0, 0)	23	(0, 0)	23	
	(0, 1)	(0, 0)	23	(0, 0)	23	
f_3	(2, 0)	(5.4920, -4.4920)	91	(5.4920, -4.4920)	26	(5.4920, -4.4920)
	(4, -2)	(5.4920, -4.4920)	26	(5.4920, -4.4920)	6	
	(0, 0)	(5.4920, -4.4920)	15	(5.4920, -4.4920)	15	
f_4	(0, 1)	(0, 0)	2	(0, 0)	2	$x_2 = -\frac{1}{2}x_1^2$
	(0, -1)	(0, 0)	2	(0, 0)	2	
	(0, 2)	(0, 0)	2	(0, 0)	2	
f_5	$(-\frac{\sqrt{3}}{2}, 1)$	(1.5, 2.25)	26	(1.5, 2.25)	26	(1.5, 2.25)
	$(-\frac{\sqrt{3}}{2}, 0)$	(1.5, 2.25)	26	(1.5, 2.25)	26	or
	$(-\frac{\sqrt{3}}{2}, -2)$	(1.5, 2.25)	26	(1.5, 2.25)	26	(-1.5, -2.25)
f_6	(3, 0)	(2.2063, -3.4126)	9	(2.2063, -3.4126)	9	(2.2063, -3.4126)
	(3, -5)	(2.2063, -3.4126)	9	(2.2063, -3.4126)	9	
	(3, 5)	(2.2063, -3.4126)	9	(2.2063, -3.4126)	10	
f_7	$(1, 3 + \frac{1}{2\sqrt{6}})$	(-6.0747, 4.1495)	19	(-6.0747, 4.1495)	19	(-6.0747, 4.1495)
	$(0, 3 - \frac{1}{2\sqrt{6}})$	(-6.0747, 4.1495)	17	(-6.0747, 4.1495)	17	
	$(-1, 3 + \frac{1}{2\sqrt{6}})$	(-6.0747, 4.1495)	19	(-6.0747, 4.1495)	19	

	Start Point	MPINR		GINR		Optimal Point
		Min.Point	No. Iters	Min.Point	No. Iters	
f_8	(0, 2)	(−24.0, 6.0)	26	(−24.0, 6.0)	11	(−24, 6)
	(4, 4)	(−24.0, 6.0)	11	(−24.0, 6.0)	23	
	(0, 0)	(−24.0, 6.0)	3	(−24.0, 6.0)	3	
f_9	(0, 1)	(−2.7937, 1.7937)	14	(−2.7937, 1.7937)	12	(−2.7937, 1.7937)
	(1, 1)	(−2.7937, 1.7937)	14	(−2.7937, 1.7937)	14	
	(−1, 1)	(−2.7937, 1.7937)	14	(−2.7937, 1.7937)	10	
f_{10}	$(0, 1 - \frac{1}{\sqrt{2}})$	(−2.4755, 2.9510)	13	(−2.4755, 2.9510)	36	(−2.4755, 2.9510)
	$(1, 1 + \frac{1}{\sqrt{2}})$	(−2.4755, 2.9510)	21	(−2.4755, 2.9510)	36	
	$(−1, 1 - \frac{1}{\sqrt{2}})$	(−2.4755, 2.9510)	13	(−2.4755, 2.9510)	16	
f_{11}	(−2, 4)	(−5.5541, 4.0)	81	(−2.0, 4.0)	∞	(−5.4441, 4)
	(−4, 4)	(−5.5541, 4.0)	81	(−4.0, 4.0)	∞	
	(0, 4)	(−5.5541, 4.0)	46	(−5.5541, 4.0)	47	
f_{12}	(−3, 3)	(−3.0, 6.7219)	80	(−3.0, 3.0)	∞	(−3.0, 6.7219)
	(−3, 5)	(−3.0, 6.7219)	27	(−3.0, 5.0)	∞	
	(−3, 0)	(−3.0, 6.7219)	22	(−3.0, 6.7219)	9	
f_{13}	(2, 0)	(5.55415, −5.55415)	10	(5.55415, −5.55415)	7	(5.55415, −5.55415)
	(4, 1)	(5.55415, −5.55415)	32	(5.55415, −5.55415)	12	
	(2, −1)	(5.55415, −5.55415)	10	(5.55415, 5.55415)	26	
f_{14}	(3, 0)	(0, 0)	1	(0.6, −1.2)	1	$x_2 = -2x_1$
	(−1, 1)	(0, 0)	1	(−0.6, 1.2)	1	
	(3, 2)	(0, 0)	1	(−0.2, 0.4)	1	
f_{15}	(1, 1)	(4.1038, −10.3114)	13	(4.1038, −10.3114)	21	(4.1038, −10.3114)
	(3, 0.5)	(4.1038, −10.3114)	12	(4.1038, −10.3114)	24	
	(3, −2)	(4.1038, −10.3114)	12	(4.1038, −10.3114)	27	

For instances f_2 and f_4 to f_7 , GINR and MPINR require exactly the same number of iterations to converge to the minimum point. However, some notable differences are observed on instances $f_1, f_3, f_8, f_{10}, f_{11}, f_{12}, f_{13}, f_{14}$, and f_{15} .

On instance f_1 , starting from the Hessian singularity point $(0, 2)$, MPINR gets stuck at $(1, 1)$ where the gradient is $(0, 0)$ and does not converge to the minimum, while GINR requires 8 iterations to converge. Starting from the Hessian singularity point $(2, 0)$, MPINR converges in 8 iterations to the second minimum point, while GINR converges in the same number of iterations to the first minimum point. On instance f_3 , starting from the Hessian singularity point $(4, -2)$, MPINR takes 26 iterations to converge to the minimum, while GINR requires only 6 iterations to converge.

On instance f_8 , starting from the Hessian singularity point $(0, 2)$, MPINR requires 26 iterations to converge to the minimum, while GINR requires 11 iterations. Meanwhile, starting from the Hessian singularity point $(4, 4)$, MPINR requires 11 iterations to converge, while GINR requires 23 iterations. Figures 3.2 displays how MPINR and GINR iterate starting from the singularity point $(4, 4)$ on instance f_8 .

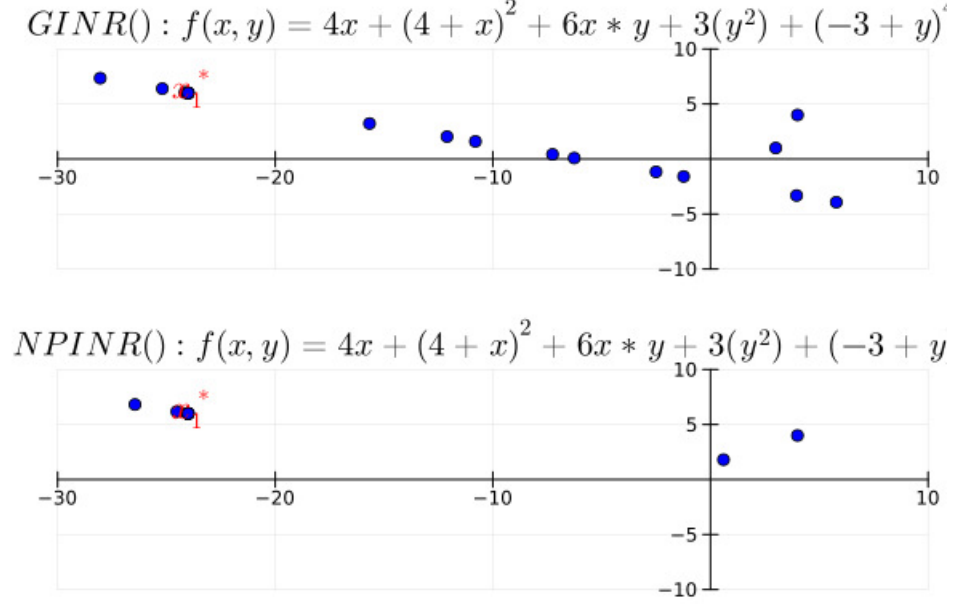


Figure 3.2: f_8 with starting point (4,4).

On instance f_{10} , starting from the Hessian singularity point $(0, 1 - \frac{1}{\sqrt{2}})$, MPINR requires 13 iterations to converge to the minimum, while GINR requires 36 iterations. Similarly, starting from the Hessian singularity point $(1, 1 + \frac{1}{\sqrt{2}})$, MPINR requires 21 iterations to converge, while GINR requires 36 iterations.

On instances f_{11} and f_{12} , MPINR can iterate starting from the Hessian double singularity points, whereas GINR gets stuck. At these points, the Hessian matrix becomes a zero matrix. MPINR corrects the initial point to $(0, 0)$ using pseudo-inverse zero-matrix multiplication, while GINR does not. Figures 3.3 displays how MPINR iterates starting from the double singularity point $(-2, 4)$ on instance f_{11} , while GINR cannot iterate.

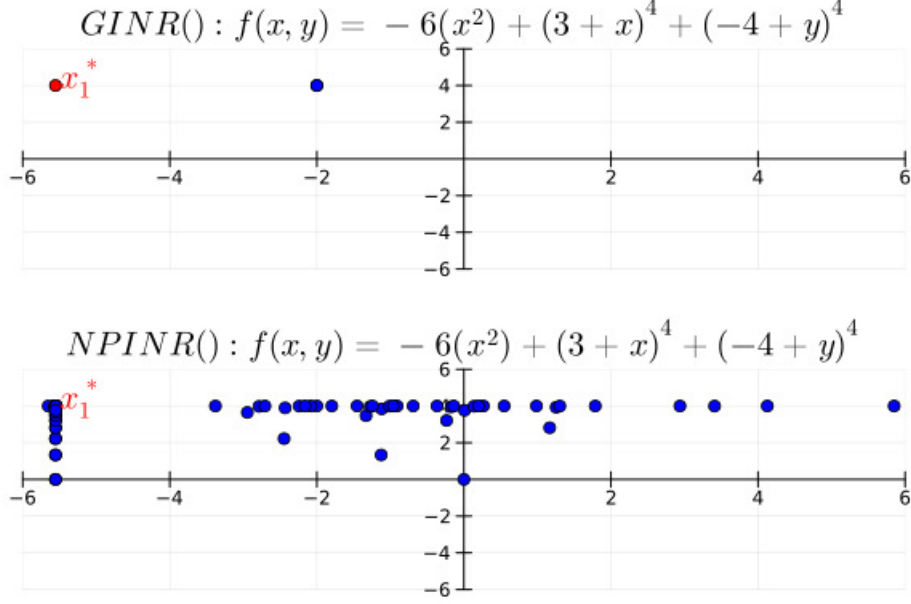


Figure 3.3: f_{11} with starting point $(-2, 4)$.

On instance f_{13} , starting from the Hessian singularity point $(2, 1)$, MPINR requires 10 iterations to converge to the minimum, while GINR requires 26 iterations. Meanwhile, starting from the Hessian singularity point $(4, 1)$, MPINR requires 32 iterations to converge, while GINR requires only 12 iterations.

On instance f_{14} , MPINR and GINR converge to two different optimal minimum points with different magnitudes, as explained in Example 3. On instance f_{15} , MPINR requires 12 to 13 iterations to converge to the minimum, while GINR requires 24 to 27 iterations. Figure 3.4 shows how MPINR and GINR iterate starting from the singularity point $(3, 0.5)$ on instance f_{15} , with MPINR requiring fewer iterations than GINR to converge to the minimum.

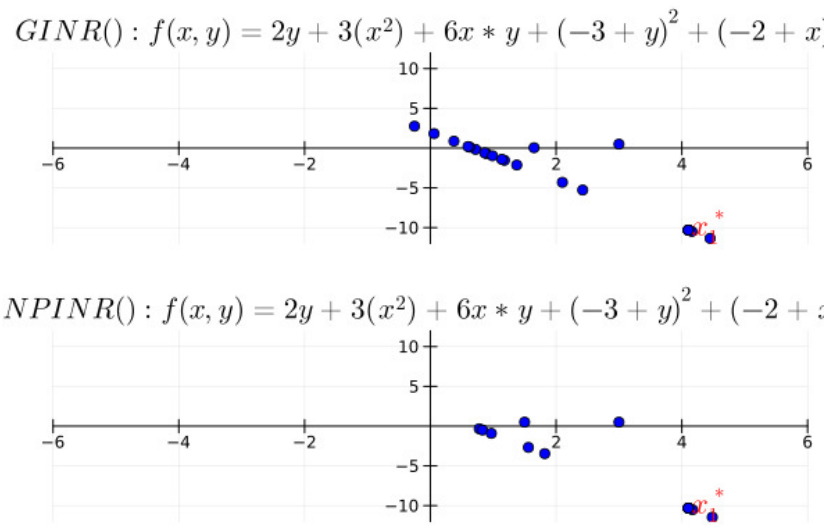


Figure 3.4: f_{15} with starting point (3,0.5).

3.5 Conclusion

In this chapter, we proposed a Modified Pseudo-Inverse Newton-Raphson (MPINR) method as an effective optimization technique for non-linear optimization problems. The MPINR method utilizes the Moore-Penrose pseudo-inverse to overcome challenges found in traditional Newton-Raphson techniques, where the Hessian matrix may be singular or nearly singular.

The MPINR method not only maintains the rapid convergence characteristics of the original Newton method but also provides a robust framework for handling singular matrices.

Through several examples, we illustrated the method's ability to achieve better convergence properties compared to GINR, emphasizing its efficiency in finding optimal solutions. Generally, the results show that the proposed algorithm is competitive with GINR in terms of efficiency and number of iterations.

Overall, the proposed MPINR method enhances the toolkit available for optimizing complex error functions in neural networks, laying the groundwork for more efficient training and improved performance in various applications. Future work could focus on further refining this method and exploring its applicability in more diverse and complex problem domains.

CHAPTER 4

FORECASTING TECHNIQUE

FOR THE DIAL-A-RIDE

PROBLEM WITH TIME

WINDOWS

4.1 Introduction

Public transport is designed for cost efficiency by utilizing fixed routes, schedules, and stops. However, it lacks the flexibility in timing and location that private transportation offers. To become a more prominent mode of transport, public transport must become more adaptable to customer constraints, such as providing door-to-door services with specific time windows.

The Dial-A-Ride Problem (DARP) addresses door-to-door public transport for seniors

or individuals with impairments. It uses a fleet of diverse vehicles to meet customer needs while aiming to minimize total travel costs. This goal is achieved by adhering to constraints like vehicle load, customer transit times, and overall route duration. DARP can be categorized into static and dynamic versions, depending on whether all requests are known during planning or received during the execution of a routing plan. This chapter focuses on the Dial-A-Ride Problem with Time Windows (DARPTW), where pickups and drop-offs must occur within designated time frames.

Fluctuating traffic and road conditions introduce uncertainty to static DARP parameters, which further complicates the creation of efficient routing plans. To address this uncertainty, the current chapter employs two-layer and three-layer artificial neural network (ANN) to forecast travel times and service times.

Section 4.2 is dedicated to the mathematical description of DARPTW. Section 4.3 reviews the existing literature on solutions related to DARP. Section 4.4 covers the general techniques and key applications of the Forecasting Approach. The theoretical background of Neural Networks are discussed in details in Section 4.5. The proposed two-layer and three-layer ANN for forecasting travel times and service times is introduced in Section 4.6. Section 4.7 presents and discusses the experimental results. Finally, Section 4.8 summarizes and concludes this chapter.

4.2 Mathematical Description of DARPTW

The dial-a-ride problem (DARP) is a subclass of Vehicle Routing Problems (VRP). It is a door-to-door transport for seniors and people with special needs. The aim of

the DARP is to design a set of minimum cost vehicle routes capable of containing as many demands as possible, under a set of constraints such as vehicle load, customer transit time and overall route duration. The DARPTW involves a set of clients, as well as a set of vehicles. It is formally defined with V as the vertices set and A as the arcs set, on a graph $G = (V, A)$. For $i = 1, 2, \dots, n$, a client i has two stop requests: (i^+) which signals its pick-up, and (i_-) which signals its drop. Each request i , either pick-up (i^+) or drop (i_-) , has a positive service time s_i and a time window: $[l_i, u_i]$. A pick-up stop is denoted by a positive demand d_{i^+} , generally equal to $+1$, while a drop stop is denoted by a negative demand d_{i_-} , generally equal to -1 . Between i^+ and i_- , a customer i has a maximum transit time r_i . A vehicle k has a maximum capacity q_k , a maximum route duration t_k , and a work time window $[l_k, u_k]$.

The main target of DARPTW is to propose a set of feasible routes on the graph G . Each client i has to be picked-up and dropped by the same vehicle, during its respective time windows. A vehicle must observe a waiting time in the case of early arrival at a customer's pick-up or drop. Waiting times are considered as being part of the route duration. Hence, a given route duration includes the total travel, waiting and service times. Additionally, any route performed by a vehicle k must not exceed the overall load capacity of the vehicle q_k and its total duration must not exceed the maximum duration t_k . Table 4.1 summarizes the adopted DARPTW notation.

The DARPTW is NP-hard and highly combinatorial. Therefore, solution methods have mostly been focused on heuristic methods in the past decades. The following section reviews some of the existing literature on exact and approximate solution approaches, as well existing forecasting approaches for different DARP and VRP vari-

ants.

Table 4.1: DARPMTW notation

General	Type	Description
$\mathcal{I}$	Set	Set of all customers;
$\mathcal{N}$	Set	Set of all pick-up and drop nodes;
$\mathcal{K}$	Set	Set of all vehicles;
$\{D\}$	Set	Set of depots;
$\mathcal{V}$	Set	Vertex set = $N \cup \{D\}$;
$\mathcal{A}$	Set	Arc set of links between pairs of nodes in $\mathcal{V}$;
$\mathcal{G}$	Graph	DARMPMTW graph;
Customers	Type	Description
n	Integer	Number of customers;
i, j	Index	Customer i or j ;
i^+, i^-	Node	Pick-up and drop of customer i ;
i^-, j^+	Node	Pick-up or drop of customer i or j , or depot D ;
s_i	Real	Service time at i ;
P_i	Set	Set of time windows at i ;
w_i	Integer	Number of time windows at i ;
l_i^p	Real	Lower bound of time window p at i ;
u_i^p	Real	Upper bound of time window p at i ;
d_i	Integer	Number of customers to serve at i ;
τ_i	Real	Maximum transit time of customer i ;
Drivers	Type	Description
m	Integer	Number of drivers/vehicles/routes;
k	Index	Driver or vehicle k ;
D	Node	Depot;
l_k, u_k	Real	Lower and upper bound of the length of driver k 's work shift;
L_k	Real	Loading capacity of vehicle k ;
T_k	Real	Maximum route duration of vehicle k ;

4.3 Literature Review

The dial-a-ride problem (DARP) has been among the most attractive operations research problems since its wide-range applicability in health, transportation, service, and manufacturing environment in real-life. Many different variants of DARP have been considered and will have been developed by extending the problem into a different environment with additional constraints and objectives. The DARP was first introduced by Wilson [58] in 1971. There are lots of literature survey papers for DARPs in different dates. Although our focus is on studies conducted after 2000, there are numerous milestones before 2000 that we briefly introduce.

Cordeau and Laporte [59] presented a literature survey for dynamic and static DARPs until their date 2003. Different approaches were presented for the DARP and its different variants. These techniques can be classified as follows:

4.3.1 Exact Approaches

Exact approaches for the DARP were mainly developed based on the concept of the branch and bound algorithm. They can be categorized into three techniques: branch and cut, branch and price, and branch and price and cut algorithms. These approaches guarantee optimal solutions while consuming large running times [60]. Thus, these types of methods are appropriate for small-size instances. In addition, a reduction approach was proposed for specific small DARPs.

Branch-and-Cut

This approach is based upon the branch and-bound with cutting planes which are used for tightening linear programming relaxations. This procedure verifies optimal solutions by providing stronger bounds. Cordeau [61] introduced an integer programming (IP) model and a Branch-and-Cut (BAC) algorithm using various families of valid inequalities as cutting planes for the DARP with time windows, aiming to minimize the total vehicle routing cost. The problem they investigated included constraints such as ride time, pairing, time windows, vehicle capacity, and precedence. Computational experiments demonstrated that the proposed BAC algorithm is effective for small and medium-sized problems. In a similar way, Liu *et al.* [62] used eight different families of valid inequalities to define a branch and cut method applied on a set of problems

generated according to real life data. they designed a BAC algorithm to solve their investigated problem. Their experimental results showed that their valid inequalities improve the lower bounds. They also reported that their proposed BAC algorithm is effective. Heilporn *et al.* [63] implemented Branch and Cut algorithm in the stochastic context of the DARP.

Branch-and-Price

Unlike Branch and Cut algorithms, Branch and Price algorithms base on column generation rather than generating cuts for LP relaxations in a branch and bound method. Branch and Price algorithms need to reformulate the problem into a restricted master problem and a pricing subproblem. Garaix *et al.*[64] used dynamic programming to solve the pricing subproblem. The algorithm associates partial paths with labels and extends these labels. It takes into account the subproblem's resource constraints and reduced cost. It is also able to deal successfully with a linear fractional objective function. Florio *et al.* [65] considered VRPSD-PDC problem which represents the vehicle routing problem with stochastic demands and probabilistic duration constraints. They modeled the problem as a set-partitioning problem and solved it with a branch-and-price algorithm.

To verify route feasibility, three strategies were employed. The first one consisted in the application of Chebyshev's bounds on the routes' duration. In order to keep track of the variance of the route duration, this procedure was incorporated into the pricing algorithm. The second strategy used Monte-Carlo sampling and statistical inference to determine the probability that the route duration exceeds a specific limit. The last

strategy derived the route duration probability distribution.

Branch-and-Price-and-Cut

Branch-and-Price-and-Cut methods are Branch-and-Price methods with cutting planes added to the LP relaxations through the solution process. They can benefit from both Branch-and-Price and Branch-and-Cut algorithms by (i) handling smaller size problems with columns which result from the solution of the sub-problems and (ii) tightening the values of bounds for the Linear Programming relaxation. In order to reduce the number of paths to be enumerated for solving the sub-problems, Qu and Bard [66] used dominance rules. They applied subset row inequalities and showed the advantages of these cuts on reducing the number of nodes to be explored in the Branch-and-Price tree and the overall solution time.

The addition of cuts is effective in reducing the number of nodes to be visited in the branch-and-bound tree. Hence, the majority of the overall run time of the Branch-and-Price-and-Cut is spent solving the sub-problems, as reported in [66]. This suggests that more research on faster algorithms to solve the sub-problems is needed to improve the overall efficiency of Branch-and-price-and-cut algorithms.

4.3.2 Heuristic Approaches

Approximation approaches do not guarantee optimal solutions but provide very good near optimal solutions in a short time. Thus, these types of methods are suitable for a realistic scale instances. For this reason, researchers mainly focused their efforts on developing approximation methods for the DARPs during the last decades.

The literature on heuristic approaches for the DARP can be classified into several subcategories as shown below.

Insertion heuristics

Jaw *et al.* [67] proposed a sequential insertion heuristic. The customer requests in this method are sorted depending on their earliest arrival time, and inserted iteratively into the best feasible locations of the vehicle routes. Diana and Dessouky [68] presented a parallel insertion method with a regret criterion with a new procedure to initialize the route to improve the quality of the final solutions. An insertion-based heuristic was proposed by Madsen *et al.* [69] for the DARPTW with additional constraints, multiple objectives, and multiple capacities. The method was applied for the online scheduling of new requests in a dynamic environment.

Local-search heuristics

The tabu search (TS) heuristic was introduced by Glover [70], and Cordeau and Laporte [71] proposed a TS for DARP with a time window where the objective was minimizing the routing cost of vehicles departing from a common depot. The constraints of their problem were vehicle capacity, ride times and route duration. They used real-life and randomly generated data sets to execute their algorithm. Attanasio *et al.* [72]. considered dynamic DARP by generating several parallel TS algorithms that were designed for static DARP. Their experiment revealed that their proposed algorithms are effective to response user requests. Paquette *et al.* [73] designed a multicriteria

DARP considering service quality and cost minimization. They proposed a TS algorithm that uses the reference point method for multicriteria optimization. Their experimental study reveals that the proposed algorithm may obtain non-dominated solutions.

Variable neighborhood search (VNS)

The variable neighborhood search heuristic (VNS) was proposed by Mladenović and Hansen [74]. The main concept of VNS is to change between multiple neighborhood search operators in order to escape from local optima. Parragh and Doerner [75] used VNS to solve the DARP. They introduced three operators: exchange, chain, and zero-split. These operators permit infeasible solutions to be accepted through the search process by applying the relaxation technique of Cordeau and Laporte [59]. In their experimental study, they reported new best results of most of the existing benchmark instances. Schilde *et al.* [76] modeled the stochastic dynamic dial-a-ride problem with predictable return transports. They showed that considering stochastic information of the return transports has a significant positive effect on the quality of the problem solution. Additionally, they compared four different meta-heuristics for their problem. These approaches are: multiple plan approach (MPA), the multiple scenario approach (MSA), variable neighborhood search (VNS) and stochastic-VNS (SVNS). By using real-life data, they generated test instances with different scenarios and tested their proposed algorithms. Belhaiza [77] modeled the VRP from a game theoretic perspective and adopted a Pareto non-dominated solution approach. Belhaiza [78] designed a hybrid ALNS algorithm with additional heuristics to solve DARP with time win-

dows where the objective is to increase the performance of routing operations. They tested their solution algorithm by using test instances derived from real-life data of the city Vancouver. Masson *et al.* [79] proposed an adaptive large neighborhood search algorithm (ALNS) for a variant of the DARP where customers can change vehicles during their travel at predefined transfer nodes (DARPT). The presented algorithm generated an 8 Recently, Molenbruch *et al.* [80] combined a VNS approach within a multi-directional local search algorithm.

Hybrid algorithms

A hybrid large neighborhood heuristic was proposed by Parragh and Schmid [81] for the DARP. They were able to improve many solutions on benchmark instances using a variety of neighborhoods. Belhaiza *et al.* [82], proposed a Hybrid Variable Neighborhood Tabu Search heuristic (HVNTS) and considered minimizing the total duration as the main objective. Belhaiza *et al.* [83] proposed a hybridization meta-heuristic called hybrid genetic variable neighborhood search heuristic (HGVNS). A genetic crossover operator heuristic was integrated into the shaking phase of VNS. Belhaiza [84] proposed a data-driven hybrid evolutionary algorithm for the DARPTW. The heuristic used evolutionary crossovers within a hybrid VNS algorithm guided by data extracted from routes with infeasibilities. Chevrier *et al.* [85] addressed a demand responsive transport service (DRT) as a special case of the DARP and solved it using a hybrid evolutionary multi-objective approach. Beaudry *et al.* [86] studied a patient transportation DARP with additional constraints that are specific for the concept of the hospital. The authors presented a two-phase heuristic solution

approach that uses an insertion based initial solution approach in the first phase and a TS algorithm to enhance the solution in the second phase. According to their computational study with real-life instances, their proposed algorithm can decrease the waiting times of patients and it can provide good solutions. Detti *et al.* [87] addressed a multi-depot DARP with healthcare operations where there are the quality of service concerns, preferences of patients, heterogeneous vehicles, vehicle-patient compatibility and tariffs for vehicle waiting times. They proposed Tabu Search (TS) and VNS algorithms to solve their problem. Furthermore, they presented a mixed integer linear programming (MILP) model for the problem. They reported that their proposed solution approaches perform after an experiment with instances with real-life data.

Other methods

A dynamic programming-based approach was explored to solve the single vehicle routing problems in [88] and multi-vehicle routing problems in [89]. Other heuristic approaches, such as Simulated Annealing (SA), have also shown to be of great benefit. Braekers *et al.* [90] presented a Deterministic Annealing (DA) metaheuristic for a heterogeneous dial-a-ride with multiple depots problem. They concluded that it is an efficient method for solving larger instances compared to branch-and-cut algorithm. Zidi *et al.* [91] solved the DARP problem with multi-objectives using simulated annealing approach. Their approach was tested on benchmark instances proposed by Cordeau and Laporte [59]. The results proved that SA can provide good quality solutions in short computing times. Kirchler and Calvo [92] designed a granular TS algorithm for static DARP with time windows. They compared their algorithms with

classical VNS, TS and Genetic Algorithm (GA). They stated that their proposed algorithm outperforms other compared algorithms. Rekiek *et al.* [93] introduced a GA to solve DARP problem that is to transport handicapped people in Brussels. They stated that their algorithm outperforms the manually generated solution comparing service quality and computational effort. They reported that their algorithm can find routes by satisfying all constraints in a short time period. Jorgensen *et al.* [94] proposed a GA for DARP where the objective was minimizing the total routing cost of vehicles by satisfying service level constraints. The proposed GA first assigns costumers to vehicles and then solves independent routing problems. They compared their proposed algorithm with state-of-the-art methods and they stated that their algorithm is effective in terms of solution quality.

4.3.3 Statistical and Machine Learning Approaches

One of the challenges in dealing with the DARP in particular, and VRP in general, is the lack of knowledge of future traffic variables such as flow, speed, density, time, demand, etc. Predicting such variables would tremendously help during the operations planning and route execution processes. This section reviews and compiles the most important predictive models built for demand travel, traffic flow and travel time forecasting for VRP variants.

Demand forecasting

Predictive analytic techniques were used for demand requests forecasting over short and medium terms in routing networks. These techniques are deemed necessary in

problems where user demands are of stochastic nature and virtually unknown. Several works proposed to develop predictive algorithms and efficient models to handle the routing of vehicles based on customers' future demand prediction [95], [96]. Many of the techniques proposed to predict demand relied on historical users' request information and traffic related values collected from different sensing devices. Powell[97] considered the problem of vehicle routing with stochastic demand, and used a backpropagation neural network approach to forecast users' demand. The model was applied to a postal delivery application and achieved a high accuracy level. Marković *et al.* [98] proposed a VRP method which applies a constructive heuristic algorithm to find the solution of a problem with stochastic demand. They used a backpropagation Neural Network to predict mail delivery to large customers. The proposed methodology proved to be effective at different uncertainty levels.

Within the same context, the dynamic origin destination (OD) demand estimation problem to solve DVRP was explored. Toledo and Kolehkina [99] presented a solution to the offline estimation of the dynamic (OD) demand matrices, using historical demand data and traffic statistics on some of the network links. This method was tested on a case-study network in Stockholm. It was shown to be scalable and efficient in terms of OD demand matrices' estimation. Ma and Qian [100] proposed an iterative generalized least squares (IGLS) approach to iteratively estimate the variance, mean and covariance of the OD demand matrix, by considering the day to day data collected over several years. Lasso regularization also was used to generate sparse covariance matrix to improve computational efficiency and to provide better interpretations. The framework also predicts the probability distributions of flow across paths and network

links, as well as their travel cost in an environment with multiple variation sources; OD demand, choice of routes, errors with unknown origin. Zhou *et al.* [101] proposed an iterative bilevel estimation framework to minimize both deviation between real time link counts and estimated link flows, and the deviation between given historical static demand and estimated time-dependent demand. Then these two deviation types were combined into one weighted objective function, with the weighting values to be determined. Zhou and Mahmassani [102] proposed an ordinary nonlinear least squares estimation method to merge automatic vehicle identification counts, historical demand and link counts information into a multi-objective model. The obtained optimization problems were solved by using an iterative bilevel estimation method. Their approach showed its effectiveness under different rates on a synthetic data set. Finally, Zhou *et al.* [103] proposed a model that uses a Kalman filter to obtain the demand deviation from a priori regular demand estimation pattern. They designed an optimal adaptive method to capture day to day demand evolution. Their algorithm was tested on a case-study depending on the Irvine test-bed network.

Traffic Flow Forecasting techniques

The most important predictive techniques proposed for traffic flow forecasting were based on time series analysis, statistical methods and artificial intelligence. Ermagun and Levinson [104] reviewed several forecasting studies on conditions of short term traffic using spatial dependence between the links. They concluded that spatial information boosts the predictions' accuracy, specially under congested traffic systems with longer horizons.

Machine Learning and Data Analytic techniques attracted a lot of attention due to their ability in predicting future variables with good accuracy levels and reducing route congestion. Artificial Neural Networks (ANN) were considered for traffic flow prediction [105] [106] [107] [108].

Goves *et al.* [105], for instance, opted for ANN based model for predicting traffic flow 15 minutes ahead of time. The model was tested on a data collected by system of the Highways England's Motorway Incident Detection and Automatic Signaling (MIDAS), and achieved an accuracy level of 90%. Lv *et al.* [106] used a deep learning approach for predicting traffic flows. An unsupervised greedy layerwise method was used to train the model and help fine-tuning the model's parameters. The model was tested on Caltrans Performance Measurement System (PeMS) database, with measurements by over 15000 individual detectors every 30 seconds deployed in the state of California. Similarly, Polson and Solokov [107] proposed a deep learning model where the layers modeled the nonlinear relations among predictors, except the first layer which identifies the relations in space and time. This model was tested on real data provided by the Illinois Department of Transportation, representing data collected from Chicago's Interstate highway. Van *et al.* [108] proposed a framework for predicting a freeway travel time using a technique called state-space neural network (SSNN) which is used for the identification of non-linear systems [109]. Their results indicate that the SSNN model generates robust and accurate travel time predictions, on both real and synthetic data.

While machine learning approaches outperformed classic statistical approaches, such

as exponential smoothing and historical average, they found no guarantee for machine learning approaches' superiority when compared with advanced statistical approaches, such as spatio-temporal autoregressive integrated moving average. Oh *et al.* [110] developed a guideline for transportation engineers and decision makers, while considering model-based methods for highways. They reviewed model-based trip time prediction approaches by categorizing them into four classes according to the amount of details involved in the model: Macroscopic, Microscopic, CA based, and Mesoscopic. They evaluated each method depending on five main perspectives: Prediction range, Applicability, Accuracy, Efficiency, and Robustness. Finally, they evaluated and discussed these approaches with respect to data-driven approaches. Vlahogianni *et al.* [111] reviewed recent developments in technology and mathematical models that allowed researchers to expand horizons towards new research directions on traffic volumes or travel times prediction. Abadi *et al.* [112] developed a model to forecast traffic flow on all links of the transportation network. Their prediction was developed based on an autoregressive model, which adjusts to unpredictable conditions. The model was tested on a real life study (California area) in a simulated environment, and was able to achieve an acceptable level of accuracy.

Furtlehner *et al.* [113] proposed a leveraging of the traffic network temporal and spatial dependency to achieve prediction accuracy. The probabilistic based-model was able to handle 80% of missing data properly, in addition to online network constraints. The missing data are mainly due to incomplete information on the network state at any time. Hence, both the learning and the running of the model needed to be able to perform under missing data. The performance of the model was tested on three

different case studies with various characteristics collected in Vienna (over a period of four years), Turin (over a period of four years), and San Francisco Bay area (over a period of eleven months). Attanasi *et al.* [114] proposed a hybrid model for flow output forecasting by data sensor devices. The model is based on Bayesian and Neural Networks. The approach works in two phases. The first phase involves an offline clustering analysis based on affinity propagation, which is an unsupervised technique where the clusters number is not preset. The second phase involves an online forecasting using a Neural Network and a Bayesian Network. Bayesian inference method is such that the posterior probability density function is derived by combining a prior knowledge with some evidence information observed during the experiments. Tests were conducted on a large data set collected on a three years period over the city of Vienna's road network. The results showed that the Bayesian Network gave a slightly better performance than Neural Networks.

Travel Time Forecasting

Travel time measurements are fundamental in intelligent transportation systems (ITSs). Travel time forecasting is generally studied through the analysis of travel time reliability (TTR). TTR determines the extent of unexpected travel time delays, and formally refers to the dependability or consistency in travel times, from day to day or/and across different times of the day. TTR modeling gained a lot of attention due to its ability to improve routing applications' performance by providing accurate predictions of trip travel times. To the best of our knowledge, there are no travel time forecasting DARP or DVRP related papers published yet.

Yang *et al.* [115] proposed a generalized autoregressive conditional heteroscedasticity model (GARCH) to address travel times volatility, and provide reliable information for travel time forecasting. They applied their approach on three different real data sets to predict standard deviation (PSD), and detect reliable sample points. Their results show that the reliability of travel time forecasting increases with more data acquisition. Wang *et al.* [116] developed a methodology to forecast TTR for freeway transportation planning based on GPS data. TTR is measured using the instantaneous GPS spot speed variation distribution. They established a correlation between TTR and traffic density. To do so, they segmented TTR and traffic density data sets into distinct homogeneous clusters based on the K-means approach, and fitted the reliability-density correlation of every cluster using squared errors minimization. Juhász *et al.* [117] analyzed travel time reliability (TTR) in urban transport. They used automatic measurements of times and traffic volumes from a data set of the city of Budapest. They studied a forecasting model where travel times standard deviation depends on volume-capacity ratios and free-flow travel times. They performed a real-life experiment which proved that travel times can reliably be predicted using volume capacity ratio.

Chen and Fan [118] developed a systematic method for analyzing TTR on segments of roadway. They illustrated TTR models based on data collected on a freeway corridor in Charlotte, North Carolina. They considered a number of influential operators, such as segment location, time, day, and year. They developed a time series pattern for forecasting the TTR. Their experimental results indicated that the TTR models are unique, under every case and under different days of week and weather conditions.

Kidando *et al.* [119] analyzed the impact of TTR and other factors on crash severity probability on arterial roads. They applied two random effect regressions; the traditional random effect (TRE) logistic regression and the Dirichlet random effect (DRE). For both models, they developed a Markov Chain Monte Carlo simulation to obtain posterior distributions of the parameters. Using four years reported crash police data, as well as trip speeds in Northeast Florida, the analysis of goodness of fit concluded that the DRE model fits the data with more accuracy.

4.4 Forecasting Approach: Definition and Applications

Forecasting techniques are methods used to predict or estimate future events, trends, or outcomes based on historical data. These techniques apply statistical, mathematical, and computational models to analyze past and current information, identifying patterns that can inform projections about future events. Forecasting techniques are widely used in various fields, including finance, economics, weather prediction, supply chain management, and transportation, to aid in decision-making and strategic planning. Forecasting techniques can vary depending on the nature of the data and the specific context in which they are applied.

Examples of forecasting techniques include time series analysis, regression analysis, machine learning (ML) models, and artificial Intelligence (AI). ML and AI techniques have gained popularity in forecasting due to their ability to handle complex data sets, adapt to changing patterns, and provide accurate predictions in various domains. It's

important to note that ML and AI forecasting techniques require careful consideration of data quality, model selection, hyperparameter tuning, and validation procedures. Artificial Neural Networks (ANNs) are a subset of machine learning algorithms inspired by the structure and functioning of biological neural networks. More details in the next section.

4.5 Neural Networks: Mathematical Background

Artificial Neural Networks are powerful tools for a wide range of tasks due to their ability to model complex relationships and adapt to new data. Considerable research has focused on addressing one fundamental question: "How does the human brain work?" Artificial Neural Networks have been employed in attempts to answer this question. Hagan et al.[\[1\]](#) reported that early research in the field of neural networks dates back to 1943, when McCulloch and Pitts proposed a simple mathematical model to explain the biological functioning of neurons. This work is recognized as one of the first significant studies on artificial neural networks. The technique of ANNs presents a promising alternative methodology.

Over more than five decades, numerous investigations have found that the biological neural system is a suitable model for applying ANNs in real-world scenarios. However, it was not until the 1980s that ANNs began to be applied on a large scale to real-world problems. Consequently, common applications were not trained by algorithms due to their lack of sophistication.

The brain is made up of numerous neural cells (biological neurons), each comprising a cell body, dendrites, and an axon. These neurons are interconnected, forming a larger biological neural system. The connections, known as *synapses*, occur between the dendrite of one neuron and the axon of another. When a neuron receives a signal through the axon, it either excites or inhibits this signal and transmits it through its dendrites to all connected neurons. The concept of ANNs is designed to emulate the neural network of the brain. To achieve this, an ANN comprises a set of interconnected artificial neurons that mimic biological neurons.

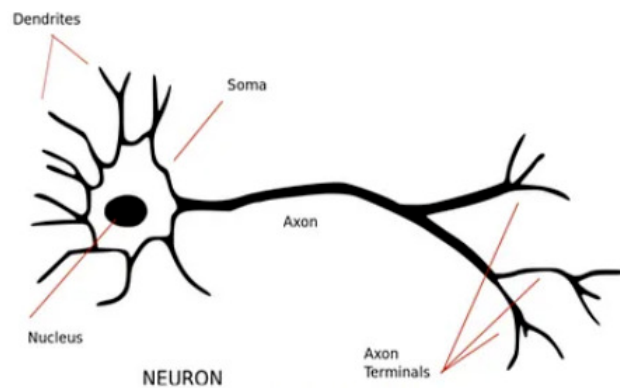


Figure 4.1: A biological neuron

4.5.1 Neural Network Architecture

ANNs consist of interconnected nodes known as neurons, are organized into distinct layers. The neural network typically comprises three main types of layers: Input Layer, Hidden Layers, Output Layer.

Each neuron in the network receives inputs, applies a specified mathematical function which is called *activation function*, and passes the resulting output to the neurons

in the next layer. The connections between neurons have associated weights, which determine the strength and significance of each connection. During the training process, these weights are iteratively adjusted to minimize the error between the network's outputs and the desired or target outputs.

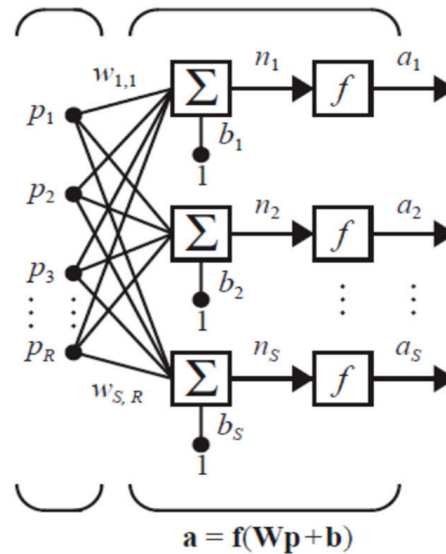


Figure 4.2: An Artificial Neuron [1]

4.5.2 Fundamentals of Artificial Neural Networks

There are three main aspects which characterize an artificial neural network (ANN) and affect its performance: (1) the architecture of the network (2) the training or learning algorithm (3) the activation functions. The fundamental elements of a neural network include:

1. **Neurons:** Neurons also known as *Nodes*, are the basic units within a neural network. Each neuron receives input, processes it through a specific mathematical function, and produces an output

2. **Layers:** Neurons are organized into layers which are:
 - **Input Layer:** The first layer that receives the initial data input into the network.
 - **Hidden Layers:** Intermediate layers where computations are performed. There can be multiple hidden layers in a network and allow the network to learn complex patterns.
 - **Output Layer:** The final layer that produces the final output of the network.
3. **Weights:** Weights Connections (Parameters) between neurons that represent the strength of the connection between neurons in adjacent layers. These weights are adjusted during training to minimize the error in predictions.
4. **Biases:** Biases are extra parameters added to the weighted sum of inputs to neurons to shift the activation function horizontally to enable a better fitting of the model to the data. in another words; The bias serves to shift the output of the activation function, allowing the neuron to model more complex, nonlinear relationships in the data.
5. **Activation Functions:** In ANN, the weighted sum which is the sum of products of inputs and their corresponding weights is computed. Then, an activation (Transfer) function is applied to the weighted sum to obtain the output of that particular layer and consider it as the input to the next layer. The activation function must be differentiable for backpropagation implementability.

The different types of activation function can be classified into two main groups: linear activation functions and non-linear activation functions. Non-linear activation functions are more widely used compared with the linear activation functions. With the help of these functions, ANN can extract information from data and represent non-linear functional mappings from input to output.

The basic role of activation functions is to introduce nonlinearity into the network's modeling capabilities. If ANN does not incorporate any activation function, the output of each neuron would be a simple linear function of its inputs. In other words, without the inclusion of nonlinear activation functions, the neural network would effectively be reduced to a linear regression model. A linear regression model is limited in its ability to capture complex, nonlinear relationships between the input data and the desired output. By omitting activation functions, the neural network would be constrained to modeling only linear associations, failing to take advantage of the network's potential to learn and represent complicated, nonlinear patterns in the data.

Applying a nonlinear transformation to the weighted sum of inputs for each neuron, the activation function enables the neuron to produce a nonlinear output. This enhanced modeling capability is a key factor in the success of neural networks in solving a wide range of challenging in real-world data. Figure 4.3 displays a table which contains some of the activation functions. For instance, *wave* is a wavelet-inspired function which was proposed by Hara and Nakayamma [120]. Efe [121] presented two activation functions labeled *sincos*

Label	Activation function	Corresponding derivative function
logsig	$\phi_i(u_i(t)) = \frac{1}{1 + \exp(-u_i(t))}$	$\phi'_i(u_i(t)) = \phi_i(u_i(t)) \cdot (1 - \phi_i(u_i(t)))$
tanh	$\phi_i(u_i(t)) = \frac{2}{1 + \exp(-2 \cdot u_i(t))} - 1$	$\phi'_i(u_i(t)) = 1 - (\phi_i(u_i(t)))^2$
cloglog	$\phi_i(u_i(t)) = 1 - \exp(-\exp(u_i(t)))$	$\phi'_i(u_i(t)) = \exp(u_i(t)) \cdot (1 - \phi_i(u_i(t)))$
cloglogm	$\phi_i(u_i(t)) = 1 - 2 \cdot \exp(-0.7 \cdot \exp(u_i(t)))$	$\phi'_i(u_i(t)) = 1.4 \cdot \exp(u_i(t)) \cdot \exp(-0.7 \cdot \exp(u_i(t)))$
probit	$\phi_i(u_i(t)) = \Phi(u_i(t))$	$\phi'_i(u_i(t)) = \frac{1}{\sqrt{2\pi}} \exp(-u_i(t)^2/2)$
loglog	$\phi_i(u_i(t)) = \exp(-\exp(-u_i(t)))$	$\phi'_i(u_i(t)) = \exp(-u_i(t)) \cdot \phi_i(u_i(t))$
sech	$\phi_i(u_i(t)) = \frac{2}{\exp(u_i(t)) + \exp(-u_i(t))}$	$\phi'_i(u_i(t)) = -\phi_i(u_i(t)) \cdot \tanh$
sinc	$\phi_i(u_i(t)) = \begin{cases} \frac{\sin(\pi u_i(t))}{\pi u_i(t)} , & u_i(t) \neq 0 \\ 1 , & u_i(t) = 0 \end{cases}$	$\phi'_i(u_i(t)) = \begin{cases} \frac{\cos(\pi u_i(t)) - \text{sinc}(u_i(t))}{u_i(t)} , & u_i(t) \neq 0 \\ 0 , & u_i(t) = 0 \end{cases}$
wave	$\phi_i(u_i(t)) = (1 - u_i^2(t)) \cdot \exp(-u_i^2(t))$	$\phi'_i(u_i(t)) = 2 \cdot u_i(t) \cdot \exp(-u_i^2(t)) \cdot (-2 + u_i^2(t))$
sincos	$\phi_i(u_i(t)) = \sin(u_i(t)) + \cos(u_i(t))$	$\phi_i(u_i(t)) = \cos(u_i(t)) - \sin(u_i(t))$
rootsig	$\phi_i(u_i(t)) = \frac{u_i(t)}{1 + \sqrt{1 + u_i^2(t)}}$	$\phi'_i(u_i(t)) = \frac{1}{(1 + \sqrt{1 + u_i^2(t)}) \cdot \sqrt{1 + u_i^2(t)}}$
logsigm	$\phi_i(u_i(t)) = \left(\frac{1}{1 + \exp(-u_i(t))} \right)^2$	$\phi'_i(u_i(t)) = \frac{2 \exp(-u_i(t))}{(1 + \exp(-u_i(t)))^3}$

Figure 4.3: Activation functions and their derivatives [2].

and *sinc*. Duch and Jankowski [122] presented a sigmoid function with roots called *rootsig*. Singh and Chandra [123] applied a class of sigmoidal functions *logsigm* with very good results. Gomes *et al* [2] present a comparative study between the neural networks models with different activation functions and two different machine learning algorithms. Since each one has its own characteristics, advantages and disadvantages, it is not easy to say which activation function is the best. The only thing that can be done, is to perform a deep experimentation to be able to select the most appropriate activation function(s).

Some commonly used activation functions in artificial neural networks include:

- Sigmoid: also known as the logistic function, takes any real number as input and produces an output value within the range $(0, 1)$. it is formulated as follows:

$$f(x) = \frac{1}{1 + e^{-x}}$$

- ReLU (Rectified Linear Unit): the ReLU function works great in most applications, and is therefore widely used. It is calculated as follows:

$$f(x) = \max(0, x)$$

- Tanh (Hyperbolic Tangent): It is computed using the following formula:

$$\text{Tanh}(x) = \frac{2}{1 + e^{-2x}} - 1 = \frac{e^{2x} - e^{-2x}}{e^{2x} + e^{-2x}}$$

This function is similar to Sigmoid function as both have an S-shaped curve, However, while the Sigmoid function produces outputs in the range $(0, 1)$, the Tanh function produces outputs in the range $(-1, 1)$.

Consider $f(\cdot)$ as the activation function used within a given neuron with common illustration displayed in Figure 4.2. With $p^t = (p_1^t, \dots, p_r^t, \dots, p_R^t, 1)$, recall that the purpose of $f(p^t) = \hat{y}^t$ is to approximate the measure of the output y^t at time or observation index t . To avoid any confusion with the time or observation index t , the transpose notation here used is $'$.

assume that:

$$x^t = w \cdot (p^t)' = \sum_{r=1}^R w_r p_r^t + b,$$

where $w = (w_1, \dots, w_r, \dots, w_R, b)$ gathers the input weights and the bias b . The neurons at every layer l ($l = 1 \dots L$) of the ANN are dotted with their activation function $f_l(\cdot)$. The output estimation following one complete forward pass through the ANN is then:

$$\hat{y}^t = F(p^t) = f_L(\dots f_l(\dots f_1(p^t))),$$

where F is the obtained composed function.

Selecting the appropriate activation function is a critical step in the design of an artificial neural network, the optimal choice depends on the specific problem which is tried to solve.

6. Loss (Error) Function: A function that measures the difference between the

network's predicted outputs and the actual target values. Common loss functions include Cross-Entropy Loss for classification tasks, While Mean Squared Error (MSE) is common loss function used for regression tasks. For example, the total estimation error function obtained for each output following one full forward pass through the L layers of the ANN is given by:

$$E(T) = \sum_{t=1}^T (y^t - \hat{y}^t)^2 = \sum_{t=1}^T (y^t - F(p^t))^2.$$

7. **Optimizer:** An algorithm used to update the weights and biases during training by minimizing the loss function. Popular optimizers include:

- Gradient Descent (GD)
- Stochastic Gradient Descent (SGD)
- Adaptive Moment Estimation (Adam)

8. **Training a Neural Network:**

- Forward Propagation: The process where input data is passed through the network layer by layer to generate an output.
- Backpropagation: The process of adjusting the weights and biases in the network based on the loss function's output to minimize the error. This is done by calculating the gradient of the loss function with respect to each weight using the chain rule.

Collectively, these components enable a neural network to learn from data, refine

its parameters, and make accurate predictions or decisions depending on the learned patterns.

4.5.3 Applications of Neural Networks

Neural networks have transformed numerous fields by allowing machines to learn from data and make decisions or predictions with a high degree of accuracy. Some common applications of neural networks include:

- **Image and Speech Recognition:** Neural networks, particularly Convolutional Neural Networks (CNNs), are used for tasks like recognizing objects in images or understanding spoken language.
- **Natural Language Processing:** Recurrent Neural Networks (RNNs) and Transformers are used for tasks such as language translation, sentiment analysis, and text generation.
- **Healthcare:** Diagnosing diseases from medical images, predicting patient outcomes, and personalizing treatment plans.
- **Time Series Forecasting:** Neural networks are utilized for forecasting tasks in finance, weather prediction, sales forecasting, and demand forecasting. Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks are popular choices for time series analysis.

The next section will focus on exploiting forecasting-based ANN technique for solving DARPTW. The framework steps of this method are shown in figure [4.4](#) . More details

about these steps will be discussed in the next section.

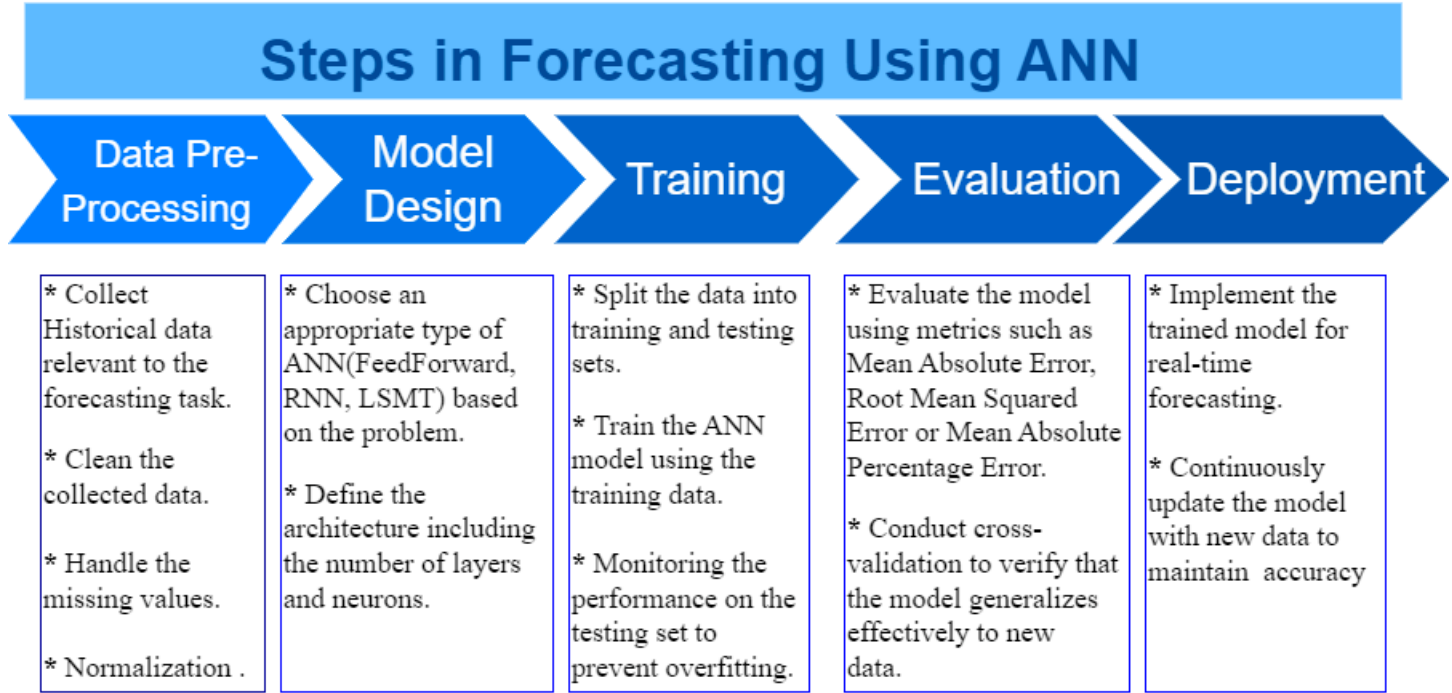


Figure 4.4: ANN Forecasting Steps.

4.6 ANN Forecasting Technique for DARPTW

This section details the proposed multi-layer neural network approach to predict DARP demands, service and travel times. We first present our data format. Then, we describe the pre-processing stage of the data. Additionally, two famous techniques for component analysis, i.e. Singular Value Decomposition (SVD) and Principle Component Analysis(PCA), are detailed. We then discuss the Seasonality Decomposition and the multi-layer neural network architecture and the subsequent transfer functions chosen. Finally, the gradient descent methods used for weight optimization are described.

4.6.1 Training Data Set

Our proposed ANN approach is trained on the Vancouver data set. A Canadian-based company, time-stamped the GPS locations of 40 DAR vehicles as they traveled through the city and suburbs of Vancouver for a one-year period. It further recorded vehicles' arrival at customers' pick-up and drop-off locations, waiting (if any), loading and unloading, and travel times. It mapped each GPS location to a node on the road map of Vancouver and its suburbs and reported the most frequently-used arcs of the road map. There are over 1500 arcs in the network. For each arc of the network, there are at least 300 and up to 500 observations. For instance, the arc at 4th Avenue-West connecting node (0) Burrard St. to node (1) Highbury St. (cf. Figure 4.5) was crossed 430 times. Although the distance in both directions is approximately 3.19 kilometers, the recorded travel times differ depending on the day, time period, road and traffic conditions, and other random events such as planned public activities at the neighboring Kitsilano Park and Beach. For instance, the average travel time from (0) to (1) is approximately 6.46 minutes, while the average travel time from (1) to (0) is 6.83 minutes. The fastest travel time from (0) to (1) is 4.86 minutes, and the slowest from (1) to (0) is 8.71 minutes.

To each arc (i, j) , connecting two DAR stops i and j , is associated with a set of historical temporal data. This data consists of observations of independent (input) and dependent (output) variables. At the time t , input includes the longitude and latitude of i and j , the day order, season, day of the week, and time of day. Output includes the arrival time, waiting time, loading/unloading time, and the observed



Figure 4.5: Vancouver, Burrard st. to Highbury st.

travel time between i and j at t . The day order ranges from 1 to 365 and follows the sequence of days within a year. We consider three seasons, each lasting four months. The day of the week ranges from 1 on Monday to 7 on Sunday. A day is divided into four slots: 6 a.m. to 9 a.m., 9 a.m. to 12 p.m., 12 p.m. to 3 p.m., and 3 p.m. to 6 p.m.

4.6.2 Data Pre-Processing

Data preprocessing is an essential step in the data mining and machine learning process, and analytics because the quality of the data significantly affects the performance of any model or analysis. This process includes transforming raw data into a clean, standardized, and suitable format for analysis, modeling, or further processing. First of all, the acquired data needed to be transformed into input-output times series samples to be able to train the model. In particular, we had to clean the inputs with 0 values most likely due to measurement errors. The entries in the corresponding cells were then replaced by the average values observed for the travel and service times, par-

ticularly taking into account the average travel time in minute per kilometer ratio observed on similar days, similar seasons or similar periods.

4.6.3 Variance and Component Analysis

Variance and component analysis are important statistical techniques used to understand and interpret data, particularly in high-dimensional datasets. Variance and component analysis are foundational statistical tools in data analysis, enabling researchers to make informed decisions based on the structure and relationships within their data. The goal of variance analysis is to identify correlations between features and outputs. These techniques are essential for extracting useful patterns and insights from large datasets. Both variance and component analysis are widely applicable across various fields, including: control systems, power networks, variance analysis, image processing, stability analysis, Data Mining.

Component analysis, particularly Principal Component Analysis (PCA), is a technique used to reduce the dimensionality of data while retaining as much information as possible. This involves several key steps: aims to find patterns in data and reduce the number of variables to the most significant ones, while preserving essential information. To carry out PCA, specific mathematical and statistical steps are necessary: Calculate the covariance matrix, determine the eigenvectors and eigenvalues, conduct singular value decomposition (SVD).

Covariance Matrix

The covariance matrix is a key concept in data analysis and statistics that offers insights into the relationships among multiple variables. It reflects how two variables vary together and is crucial for various applications, including financial analysis, Principal Component Analysis (PCA), and multivariate statistics. It is also known as the variance-covariance matrix, it displays the variance of each element along the main diagonal, while the covariances between different elements are found in the off-diagonal elements.

The variance-covariance matrix is a square matrix where the diagonal elements indicate the variance, while the off-diagonal elements represent covariance. The covariance between variables can be any real number—positive, negative, or zero. A positive covariance indicates that as one variable increases, the other variable tends to increase as well, whereas a negative covariance suggests that as one variable increases, the other variable tends to decrease. A covariance close to zero implies that the variables are uncorrelated, meaning changes in one variable do not predict changes in the other. The dimensions of a covariance matrix depend on the number of variables in a data set. For instance, if there are two variables, the covariance matrix will have two rows and two columns, and so on.

Mathematically, the covariance between two features X , Y can be calculated as:

$$Cov(x, y) = \frac{\sum (x - \bar{x})(y - \bar{y})}{n - 1}$$

where: n represent the observations, $\bar{x}$ and $\bar{y}$ are respectively, the means of x and y .

4.6.4 Singular Value Decomposition (SVD)

The singular value decomposition (SVD) of matrices is considered as one of the most important concepts in matrix theory, because of its variety of applications in mathematics and many other areas of science.

SVD is a generalization of the spectral decomposition. While some decompositions are just applied for real square matrices, the SVD can be found to any rectangular matrix. It was presented by Beltrami and Jordan in 1873-1874 for square matrices, while Eckart and Young gave the first complete proof of the singular value decomposition for rectangular complex matrix in 1936-1939. More details about the early history of SVD are discussed by Stewart [124].

SVD has many applications in different fields. In statistical data analysis, the SVD is a particularly useful tool for finding least-squares solutions and approximations. In numerical analysis, the SVD provides a measure of the effective rank of a given matrix. Additionally, it has many applications in biology, signal processing, control system analysis, system identification and mathematical modeling.

SVD is used to compute the Moore-Penrose pseudo inverse that can be used in solving a system of linear equations and can give the optimal solution of the least squares problem when solving an overdetermined system.

SVD of a matrix is similar to the diagonalization of a matrix. Diagonalization of a matrix decomposes the matrix into factors using the eigenvalues and eigenvectors. It is of the form $A = VDV^T$, where the columns of V are eigenvectors of A , and D is a diagonal matrix since the diagonal elements are the eigenvalues. Whereas, the SVD

factorization is of the form $A = U\Sigma V$, U and V are called left and right singular vectors for A , Σ is a diagonal matrix consisting of the singular values of A .

Eigenvectors and Eigenvalues

Definition: Let $A \in \mathbb{R}^{n \times n}$. The number $\lambda \in \mathbb{R}$ is called an eigenvalue of A , if there exists a non zero vector $v \in \mathbb{R}^n$ such that $Av = \lambda v$. In this case, v is called eigenvector of A associated with the eigenvalue λ . The set of all eigenvalues of A is called the spectrum of A and is denoted by $\Lambda(A)$.

Example: For the matrix $A = \begin{bmatrix} 1 & 6 \\ 5 & 2 \end{bmatrix}$ has eigenvectors:

$$v_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, v_2 = \begin{bmatrix} 6 \\ -5 \end{bmatrix}$$

Since:

$$Av_1 = \begin{bmatrix} 1 & 6 \\ 5 & 2 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 7 \\ 7 \end{bmatrix} = 7 \begin{bmatrix} 1 \\ 1 \end{bmatrix} = 7v_1$$

$$Av_2 = \begin{bmatrix} 1 & 6 \\ 5 & 2 \end{bmatrix} \begin{bmatrix} 6 \\ -5 \end{bmatrix} = \begin{bmatrix} -24 \\ 20 \end{bmatrix} = -4 \begin{bmatrix} 6 \\ -5 \end{bmatrix} = -4v_2$$

Therefore, v_1 is an eigenvector associated with the eigenvalue $\lambda_1 = 7$, and v_2 is an eigenvector associated with the eigenvalue $\lambda_2 = -4$ and $\Lambda(A) = \{7, -4\}$

Definition: Let $A \in \mathbb{R}^{n \times n}$, then :

$$f(\lambda) = \det(\lambda I_n - A)$$

is called the characteristic polynomial of A

Theorem:

The eigenvalues of A are the roots of the characteristic polynomial of A . In order to find the eigenvalues of a given matrix A , it must find the roots of the characteristic polynomial. The corresponding eigenvectors are obtained by substituting the values of λ in the system of equations $(\lambda I_n - A)x = 0$, which means that the eigenvectors of A corresponding to λ span the null space of $(\lambda I_n - A)$.

Example: Let $A = \begin{bmatrix} 1 & i \\ i & 1 \end{bmatrix}$, the characteristic polynomial is:

$$f(\lambda) = \det(\lambda I_2 - A) = (\lambda - 1)^2 - i^2$$

$$f(\lambda) = ((\lambda - 1 - i)(\lambda - 1 + i))$$

Therefore, the eigenvalues are: $\lambda_1 = 1 + i$, $\lambda_2 = 1 - i$. By substituting $\lambda_1 = 1 + i$ with λ in $(\lambda I_2 - A)x = 0$, gives:

$$\begin{bmatrix} i & -i \\ -i & i \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Its solution is : $x_1 = x_2$. Hence, $v_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ is an eigenvector corresponding to:

$\lambda_1 = 1 + i$. Likewise, for $\lambda_2 = 1 - i$ gives the eigenvector $v_2 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$.

Computation of the SVD

The decomposition of an $m \times n$ matrix A consists of an $m \times n$ diagonal matrix Σ of the form:

$$\Sigma = \begin{bmatrix} D & 0 \\ 0 & 0 \end{bmatrix}$$

where: D is an $r \times r$ diagonal matrix.

Definition Let $A \in \mathbb{R}^{m \times n}$, A non negative real number σ is called a singular value of A if there exist two unit length vectors $u \in \mathbb{R}^m$ and $v \in \mathbb{R}^n$, such that:

$$Av = \sigma u, A^T u = \sigma v$$

Definition: The matrix A is orthogonal matrix if it satisfies $A^{-1} = A^T$.

Theorem:

Let $A \in \mathbb{R}^{m \times n}$ matrix with rank r . There exists an $m \times n$ matrix Σ where the diagonal entries in D are the first r singular values of A , $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_r > 0$ since σ_i is the square root of an eigenvalue of $A^T A$, and $U \in \mathbb{R}^{m \times m}$, $V \in \mathbb{R}^{n \times n}$ are orthogonal matrices such that:

$$A = U\Sigma V^T$$

Definition: A matrix $A \in \mathbb{C}^{n \times n}$ is called unitary matrix if :

$$U^*U = UU^* = I$$

where: $U^* = \overline{U}^T$

Remark : For complex matrix same arguments work, using Hermitian transpose instead of the usual one. Then every complex matrix can be written in the form $A = U\Sigma V^*$, where Σ diagonal, with non negative entries. U and V are unitary matrices.

Proposition : The symmetric matrices $A^T A$ and AA^T have the same eigenvalues

Example Let $A \in \mathbb{R}^{2 \times 3}$, and $A = \begin{bmatrix} -1 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$ To construct a singular value decomposition of A , we follow the steps:

- Step 1: Computing AA^T and $A^T A$

$$AA^T = \begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix}$$

$$A^T A = \begin{bmatrix} 1 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 1 \end{bmatrix}$$

- Step 2 : Finding the eigenvalue of AA^T . The characteristic polynomial is:

$$f(\lambda) = \lambda^2 - 4\lambda + 3$$

The roots of this polynomial gives the eigenvalues which are $\lambda_1 = 3, \lambda_2 = 1$

- Step 3 : Using the equation $\lambda I - A^T A = 0$ to find the eigenvectors of $A^T A$ corresponding to λ_1, λ_2

For $\lambda_1 = 3$, the corresponding eigenvector is $u_1 = (-1, 1)^T$.

For $\lambda_2 = 1$, the corresponding eigenvector is $u_2 = (1, 1)^T$.

- Step 4: All these eigenvectors u_1, u_2 must to be normalized, the matrix U will be written as:

$$U = \frac{1}{\sqrt{2}} \begin{bmatrix} -1 & 1 \\ 1 & 1 \end{bmatrix}$$

- Step 5: Likewise for AA^T , it must finding the eigenvalues and eigenvectors.

Since AA^T is 3×3 , it has the same eigenvalues of $A^T A$, and the third must be 0. Then:

for $\lambda_1 = 3$, the corresponding eigenvector is: $v_1 = (1, -2, 1)^T$.

for $\lambda_2 = 1$, the corresponding eigenvector is: $v_2 = (-1, 0, 1)^T$.

for $\lambda_3 = 0$, the corresponding eigenvector is: $v_3 = (1, 1, 1)^T$.

After normalizing the eigenvectors v_1, v_2 and v_3 , the matrix V^T is written as:

$$V^T = \begin{bmatrix} \frac{1}{\sqrt{6}} & \frac{-2}{\sqrt{6}} & \frac{1}{\sqrt{6}} \\ \frac{-1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} \end{bmatrix}$$

- Step 6: Finding the singular values of A can be done by taking the square roots of eigenvalues of $A^T A$:

$$\sigma_1 = \sqrt{3}, \sigma_2 = 1, \sigma_3 = 0$$

Then the matrix $\Sigma \in \mathbb{R}^{2 \times 3}$ will be written as:

$$\Sigma = \begin{bmatrix} \sqrt{3} & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

- Step 7: Putting all of the matrices U , V , Σ into the equation: $A = U\Sigma V^T$, the singular value decomposition will take the form:

$$A = \frac{1}{\sqrt{2}} \begin{bmatrix} -1 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} \sqrt{3} & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{6}} & \frac{-2}{\sqrt{6}} & \frac{1}{\sqrt{6}} \\ \frac{-1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} \end{bmatrix}$$

4.6.5 Principal Component Analysis (PCA)

Principal Component Analysis (PCA) is a dimensionality reduction technique that reduces the number of variables of a data set, while preserving as much information as possible. It aims to find the directions (principal components) that capture the most variance in the data. By projecting the data onto these principal components, PCA can help in visualizing the data, reducing noise, and identifying patterns and relationships within the data set.

To use the data for training the ANN, we first ‘clean it,’ by replacing 0-valued outputs with their corresponding average observed values over similar days, seasons, and periods. Second, we apply a principal component analysis (PCA) to reduce the input parameters. PCA reveals that geo-code coordinates do not have a direct impact on travel and service times at a given stop, i.e., the longitude and latitude have much

smaller entries in the closest SVD rank-one approximation matrix of our data matrix. Meanwhile, the remaining feature importance analysis comes in the following order: day of the week, the period, the day order and season.

4.6.6 Seasonality Decomposition

A time series is a sequence of observations taken over times. It illustrates the relationship between two variables, where one is time and the other is a quantitative variable. This relationship does not always show an increase or decrease in the variable over time; it can vary in different ways. These can include daily temperature in a city, weekly Retail sales, annual rainfall in a region, or numerous other metrics. Forecasting the future values of an observed time series is a crucial challenge in various fields, such as production planning, economic, sales forecasting and inventory management. Mathematically, a time series is formed as:

$$y_t = f(t)$$

where: y_t is the value of the variable at time t .

According to [125], time series data can be categorized into five key components:

- Level: The simplest form of a time series is one that only exhibits a level, indicating a constant demand over time. This is the foundation of any time series.
- Trend: The trend component reflects the growth or decline in demand over time.

It shows the general direction in which the data is moving.

- Seasonal Variations: These are fluctuations that occur at regular intervals due to human behavior, or: natural forces.
- Cyclical Movements: Unlike seasonal variations, these are longer-term, less predictable fluctuations in the data and don't necessarily repeat annually.
- Noise: This is the random, unpredictable element in the data that cannot be explained by any analytical formula. It is often assumed to be normally distributed around zero.

To represent the relationship between the components of a time series, either an additive or multiplicative models can be used.

- Additive Model for Time Series Analysis: If T_t , S_t , C_t , and R_t are the trend value, seasonal, cyclic and random fluctuations at time t respectively. A time series can be expressed as:

$$y_t = T_t + S_t + C_t + R_t$$

- Multiplicative model for Time Series Analysis: A time series can be expressed according to this models as follows:

$$y_t = T_t \times S_t \times C_t \times R_t$$

Time series can be categorized based on different characteristics. The following

are some common types of time series:

- Based on the nature of the data: A time series that consists of observations of a single variable over time is called Univariate Time Series. On the other hand, Multivariate Time Series involves multiple variables observed over time.
- Based on trend: a time series is referred as stationary if its statistical properties do not change over time. In other words, it has a constant mean and variance and it is independent of time [126]. It does not show trends or seasonality. This contrasts with non-stationary data, where data points have means, variances and covariance that change over time. Non-Stationary time series often exhibit a strong trend and seasonality, making them difficult to analyze directly. Thus, they must be converted to stationary before the modeling and forecasting operations can take place.

There are some techniques which can be used to make a non-stationary time series stationary, such as trend removal and seasonal decomposition. The first technique can be done using differencing which involves subtracting the previous value from the current value or by fitting a trend line and subtracting it from the data. The second one is used to remove the seasonal component after separating the data into its trend, seasonal, and residual components.

To choose the right forecasting approach, it is necessary to conduct a seasonality decomposition. Seasonality decomposition, whether additive or multiplicative, is made easy with the available statistics Python packages. For example, the travel times along arc $(0, 1)$ of Figure 4.5 have a periodic trend and a seasonality component. This is

reflected by Figure 4.6, which shows the seasonality decomposition, and by the 0.1692 p-value of the Dickey-Fuller test.

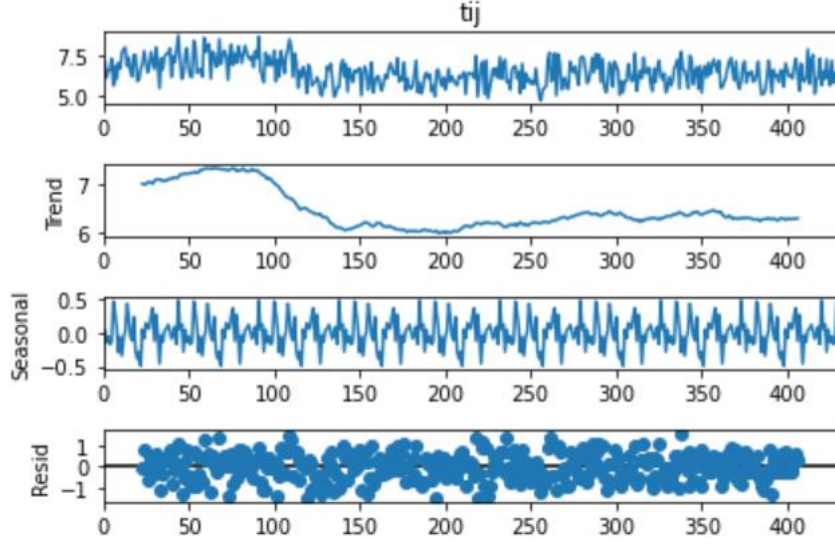


Figure 4.6: Additive Seasonality Decomposition Results

4.6.7 Multilayer Neural Network for DARPTW

The pre-processing of the data suggests that each arc (i, j) should be characterized by four input and four output variables. The input variables are the day order $p_{ij}^{5t} \in \{1, 2, \dots, 365\}$, the season $p_{ij}^{6t} \in \{1, 2, 3, 3\}$, the day of the week $p_{ij}^{7t} \in \{1, 2, 3, 6\}$, and the period of the day $p_{ij}^{8t} \in \{1, 2, 3, 4\}$. The output variables at time t are: the arrival time y_i^{1t} at node i , the waiting time y_i^{2t} , the (un)loading service times y_i^{3t} , and the travel time y_{ij}^{4t} from i to j .

Herein, we consider ANN designs that use different numbers of layers, activation

functions, and weight optimization techniques. We use a two- and a three-layer ANN.

We test four activation functions of the first (input) layer:

- the logistic-sigmoid (log-sigmoid) function $f_1(a) = \frac{1}{1+e^{-a}}$,
- the hyperbolic tangent function (tanh) $f_1(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}}$,
- the sigmoid linear unit function (selu) $f_1(a) = \frac{a}{1+e^{-a}}$, and
- the rectified linear unit function (relu) $f_1(a) = \max\{0, a\}$,

where $a_s = W_s \cdot (p_{ij}^5, \dots, p_{ij}^8) + b_s$, W_s is the weight vector at node s , $s = 1, \dots, S$, and b_s is the bias value.

4.6.8 Gradient Descent-based algorithms

We use four gradient-based weight optimization techniques. In fact, after each forward pass h , the ANN updates its weights based on its total estimation error E of the output.

When it uses a training data set with T observations,

$$E(T) = \sum_{t=1}^T (y^t - \hat{y}^t)^2 = \sum_{t=1}^T (y^t - F(p^t))^2.$$

When using a classic steepest descent, the ANN updates the weights, based on the average gradient of the entire data set:

$$w_r^{h+1} = w_r^h - \frac{\alpha}{\|\nabla E(T)\|} \frac{\partial E(T)}{\partial w_r},$$

$$b^{h+1} = b^h - \frac{\alpha}{\|\nabla E(T)\|} \frac{\partial E(T)}{\partial b},$$

where $r = 1, \dots, R$, and α is a prefixed learning rate parameter. In vector form,

$$w^{h+1} = w^h - \frac{\alpha}{\|\nabla E(T)\|} \cdot \nabla E(T).$$

Gradient Descent(GD) is fundamental to training neural networks where Neural networks are trained to minimize a loss function that measures the difference between the network's predictions and the actual target values. Actually, gradient descent provides a systematic way to adjust the network's parameters (weights and biases) to minimize the loss function. Also, gradient descent is used in conjunction with backpropagation, which efficiently computes the gradients for for each layer of the network. This allows for end-to-end training of ANN.

It operates on a simple principle: starting from a random point, we continuously move in the direction of the steepest decline until we reach the function's minimum. This method works when the function is both convex and differentiable at all points. To determine the steepest descent direction, we calculate the negative gradient of the function at the current point. By following the negative of this direction, Gradient Descent ensures we're always moving towards the minimum of the function in the most efficient way possible, given our current position. There are several variants of Gradient Descent, each designed to address specific challenges or improve performance in certain scenarios. Each of these variants has its own strengths and is suited for different types of problems or datasets. The choice of which to use often depends on the specific characteristics of the problem at hand.

Herein, in lieu of using a classic steepest descent, we test four gradient optimizers:

stochastic gradient descent (SGD), adaptive moment estimation (ADAM), adaptive gradient algorithm (AdaGrad), and an extended ADAM (AdaMax).

SGD is a gradient descent variant that is specifically designed to handle large data sets. It estimates the true gradient of the loss function using a randomly selected “mini-batch” of the data; i.e., a sample of size n_s . It then updates the model parameters in the direction of the negative gradient to minimize the loss function. Generally, SGD is computationally efficient, reaching a faster convergence and handling much larger data sets than traditional gradient descent methods. However, it may require careful tuning of its hyper-parameters and may fail to converge to the global minimum. Its pseudo-code is detailed in Algorithm 2, where ∇Q_j^h is an approximation of the gradient ∇E_h obtained using sample j at iteration h , and ϵ is the precision parameter set herein at 10^{-8} .

Algorithm 2: SGD

Start: Set $h \leftarrow 0$

Do While $\|w^{h+1} - w^h\| > \epsilon$:

- $h \leftarrow h + 1$
- Randomly shuffle samples in training set
- Select $j \in \{1, 2, \dots, n_s\}$
- $w^{h+1} \leftarrow w^h - \alpha \nabla Q_j^h$

End Do While

End: return w^h

Adam improves the efficiency and convergence of SGD by using the first and

second moments of the gradients of the loss function to update the model parameters while requiring relatively little memory. In addition, it can handle sparse gradients; thus, is suitable for deep learning models with a large number of parameters. However, it can be sensitive to the choice of its hyper-parameters and may sometimes converge to sub-optimal solutions if the learning rate is not set correctly. Its pseudo-code, given in Algorithm 3, uses a step size $\alpha > 0$, exponential decay rates $(\beta_1, \beta_2) \in [0, 1]^2$ of the gradient and its moment estimates, and iteration index h . Initially, it sets the weight vector w_0 , the first moment vector m_0 , and the initial second moment vector v_0 to zero vectors and $\alpha = 0.001$, $\beta_1 = 0.900$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$.

Algorithm 3: Adam

Start: Set $h \leftarrow 0$, $w_0 = m_0 = v_0 = \mathbf{0}$.

Do While $\|w^{h+1} - w^h\| > \epsilon$:

- $h \leftarrow h + 1$
- $\nabla E_h \leftarrow \nabla E(w_{h-1})$
- $m_h \leftarrow \beta_1 \cdot m_{h-1} + (1 - \beta_1) \cdot \nabla E_h$
- $v_h \leftarrow \beta_2 \cdot v_{h-1} + (1 - \beta_2) \cdot \nabla E_h^2$
- $\hat{m}_h \leftarrow \frac{m_h}{(1 - \beta_1^h)}$
- $\hat{v}_h \leftarrow \frac{v_h}{(1 - \beta_2^h)}$
- $w^h \leftarrow w^{h-1} - \alpha \cdot \hat{m}_h / (\sqrt{\hat{v}_h} + \epsilon)$

End Do While

End: return w^h

Adagrad adapts the learning rate for each parameter by dividing the learning rate by the sum of the squared gradients for that parameter. Thus, parameters whose gradients are large have small learning rates while those whose gradients are small have large learning rates. However, summing squared gradients over time may cause the learning rate to become too small, prohibiting the algorithm from escaping local minima of the loss function. Regardless, Adagrad handles sparse data very well and eliminates the need to manually set the learning rate, which is automatically adjusted for each parameter. Its basic version updates the weights as:

$$w^{h+1} = w^h - \alpha \cdot \frac{\nabla E_h}{\sqrt{\sum_{i=1}^h \nabla E_i^2}}.$$

Adamax generalizes Adam to the infinite norm (max). In addition, it uses the maximum value so far observed instead of computing the moving average of the squared gradients as Adam does. Thus, Adamax is more robust for large gradients, handles sparse data better, and converges faster than Adam. It is relatively easy to implement and requires little memory. However, it may be unsuitable for some problems and requires fine-tuning of its hyper-parameters. Algorithm 4 gives the pseudo-code of Adamax, where $(\frac{\alpha}{1-\beta_1^h})$ is the learning rate with a bias-correction term for the first moment. The first moment vector m_0 and the initial exponentially weighted infinity norm u_0 are initialized to zero, $\alpha = 0.002$, $\beta_1 = 0.900$ and $\beta_2 = 0.999$.

Algorithm 4: Adamax

Start: Set $h \leftarrow 0$

Do While $||w^{h+1} - w^h|| > \epsilon$:

- $h \leftarrow h + 1$
- $\nabla E_h \leftarrow \nabla E_h(w_{h-1})$
- $m_h \leftarrow \beta_1 \cdot m_{h-1} + (1 - \beta_1) \cdot \nabla E_h$
- $u_h \leftarrow \max(\beta_2 \cdot u_{h-1}, |\nabla E_h|)$
- $w^h \leftarrow w^{h-1} - \left(\frac{\alpha}{1-\beta_1^h}\right) \cdot \frac{m_h}{u_h}$

End Do While

End: return w^h

4.7 Forecasting Experimental Results

We conducted our experiments using Python 3 on Jupyter Notebook. The computations were carried out on 64-bit workstations with 2.9 GHz Intel Core i7-7600 processors and 8 GB RAM running under MS Windows. The accuracy results of the two and three-layer ANN are summarized in Tables 4.2 and 4.3 respectively. The entries in the table indicate the root mean squared error RMSE of the residuals.

RMSE is one of the most popular measures for assessing the accuracy of predictions. It illustrates the Euclidean distance between measured true values and forecasts. To compute RMSE, we first calculate the residual (difference between prediction and true value) for each data point, then compute the norm of residual for each data point. Finally, we compute the mean of residuals and take the square root of that mean using

the formula:

$$RMSE = \sqrt{\frac{\sum_{t=1}^T (y^t - \hat{y}^t)^2}{N}}.$$

All the results are obtained with a fixed number of epochs (50). An epoch refers to a single forward and backward pass through the entire training data set.

Table 4.2 reports the residual mean squared errors (RMSE) for the travel time data set using a two-layered ANN with $S_1 = 5, 10, 25, 50$ neurons in the first layer and activation function(s) $f_1 = \mathbf{log-sigmoid}, \mathbf{tanh}, \mathbf{selu}, \mathbf{relu}$, on the road segment $(0, 1)$ output features. Table 4.3 has the same information for a three-layer ANN with $S_1 = 5, 10, 25, 50$ neurons of the first layer, $S_2 = 5, 10, 20$ neurons of the second (hidden) layer, and activation functions f_1 and f_2 on the road segment $(0, 1)$ output features. The output layer uses activation function **relu** and has from 1 to 10 neurons in Tables 4.2 and 4.3. The training set is fixed at $\frac{2}{3}$ of the data set. For example, for arc $(0, 1)$: $\frac{2}{3} \times 430 \approx 288$. The results are the average RMSE over 50 consecutive runs of 50 epochs, where an epoch is a single forward and backward run over the entire training set.

Table 4.2 suggests that coupling the activation function **relu** with SGD gives, on average, better accuracy than **tanh** or **selu** for a two-layer ANN. This inference holds for the predicted output features and road network. The RMSE is lowest (0.6609) with SGD, $S_1 = 50$ and $S_2 = 10$. SGD performs better and is more stable than the other optimizers across all activation functions. We conjecture that its stochastic component mitigated the large initial error gradients whereas the other optimizers were unable to overcome this erroneous random initialization.

Table 4.2: RMSE of travel time forecasts using a two-layer ANN

		$f_1():$ log-sigmoid				
S_1	S_2	SGD	ADAM	AdaGrad	AdaMax	Avg.
5	1	0.7619	1.8845	5.9605	2.1290	2.6840
10	2	0.7779	2.1016	6.0239	2.2453	2.7872
25	5	0.7781	1.9932	5.7287	2.2049	2.6762
50	10	0.7982	2.0297	5.3079	2.2845	2.6051
	Avg.	0.7790	2.0022	5.7552	2.2159	2.6881
		$f_1():$ tanh				
S_1	S_2	SGD	ADAM	AdaGrad	AdaMax	Avg.
5	1	0.7265	1.7430	5.8112	2.1774	2.6145
10	2	0.7315	1.5831	5.3674	1.8465	2.3821
25	5	0.7299	1.6973	4.9278	1.7996	2.2887
50	10	0.7517	1.7001	5.0110	2.0857	2.3871
	Avg.	0.7349	1.6809	5.2794	1.9773	2.4181
		$f_1():$ selu				
S_1	S_2	SGD	ADAM	AdaGrad	AdaMax	Avg.
5	1	0.7628	1.4445	5.6493	1.6198	2.3691
10	2	0.7487	1.9182	5.5104	1.9673	2.5362
25	5	0.7507	1.7651	4.8466	1.8726	2.3088
50	10	0.7603	1.7294	3.9477	1.6215	2.0147
	Avg.	0.7556	1.7143	4.9885	1.7703	2.3072
		$f_1():$ relu				
S_1	S_2	SGD	ADAM	AdaGrad	AdaMax	Avg.
5	1	0.7204	2.0262	6.0948	1.8979	2.6848
10	2	0.6803	1.5727	5.9224	1.8549	2.5076
25	5	0.6771	1.4832	5.6420	1.9471	2.4374
50	10	0.6609	1.7316	5.3504	1.6817	2.3562
	Avg.	0.6847	1.7034	5.7524	1.8454	2.4965

Table 4.3: RMSE of travel time forecasts using a three-layer ANN

			$f_1()$: log-sigmoid ; $f_2()$: selu				
S_1	S_2	S_3	SGD	ADAM	AdaGrad	AdaMax	Avg.
5	2	1	0.772	1.9569	5.7671	2.2063	2.6756
10	5	2	0.7695	1.6658	4.4777	1.6279	2.1352
25	10	5	0.7493	0.9752	2.1198	1.0509	1.2238
50	25	10	0.7452	0.7317	0.7871	0.7637	0.7569
		Avg.	0.759	1.3324	3.287925	1.4122	1.6979
			$f_1()$: tanh ; $f_2()$: selu				
S_1	S_2	S_3	SGD	ADAM	AdaGrad	AdaMax	Avg.
5	2	1	0.7401	1.8111	5.522	1.8459	2.4798
10	5	2	0.7079	1.3616	3.847	1.2994	1.8040
25	10	5	0.6735	0.7267	2.0284	0.7936	1.0556
50	25	10	0.6653	0.6557	1.3772	0.6949	0.8483
		Avg.	0.6967	1.1388	3.1937	1.1585	1.5469
			$f_1()$: tanh ; $f_2()$: log-sigmoid				
S_1	S_2	S_3	SGD	ADAM	AdaGrad	AdaMax	Avg.
5	2	1	0.7557	2.0378	6.3373	2.1743	2.8263
10	5	2	0.7349	1.7355	5.8134	1.5562	2.4600
25	10	5	0.7159	1.0192	4.3961	0.9616	1.7732
50	20	10	0.7321	0.6595	2.2364	0.7001	1.0820
		Avg.	0.7347	1.363	4.6958	1.3481	2.0354
			$f_1()$: relu ; $f_2()$: relu				
S_1	S_2	S_3	SGD	ADAM	AdaGrad	AdaMax	Avg.
5	2	1	0.7381	1.8389	6.1883	2.2629	2.7571
10	5	2	0.6567	1.3032	5.6233	1.1182	2.1754
25	10	5	0.5261	0.7425	3.5542	0.8187	1.4104
50	20	10	0.4227	0.4303	1.7282	0.6144	0.7989
		Avg.	0.5859	1.0787	4.2735	1.2036	1.7854

Table 4.3 suggests that coupling the $f_1()$:**relu** and $f_2()$:**relu** activation functions with SGD generally provides the best accuracy for the three-layer ANN. The RMSE is lowest (0.4227) when $S_1 = 50$, $S_2 = 20$, and $S_3 = 10$ neurons. SGD is more stable than ADAM when S_1 varies from 5 to 50 and S_2 varies from 5 to 20 neurons. The activation functions $f_1()$:**tanh** and $f_2()$:**log-sigmoid** provide the largest RMSE on average. The best RMSE (0.6595) is obtained by the ADAM optimizer with $S_1 = 50$ and $S_2 = 20$ neurons. According to the results of the two-sample T-test, no particular architecture is significantly dominant. This result is also confirmed for the different

predicted output features. Figure 4.7 shows the actual versus predicted travel times with SGD, $f_1() = f_2() = f_3() : \text{relu}$, $S_1 = 256$, $S_2 = 128$ and $S_3 = 64$ nodes. This combination gives the lowest RMSE= 0.31806. In summary, architectures using an SGD optimizer with two or three layers of neurons dotted with relu activation functions provide the most accurate predictions.

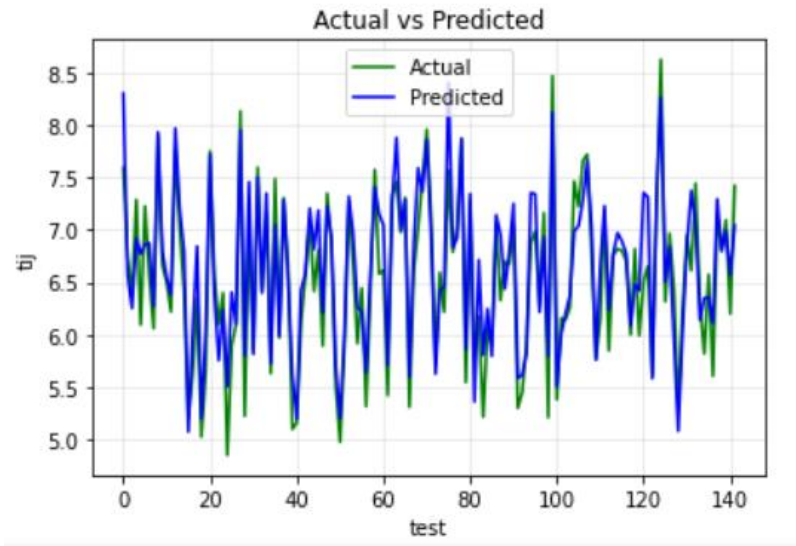


Figure 4.7: Actual versus predicted travel times with SGD and $f_1() = f_2() = f_3() : \text{relu}$.

Finally, we have also tested the proposed Modified Pseudo-Inverse Newton-Raphson (MPINR) Method in chapter 3 of this thesis and the Generalized Inverse Newton-Raphson (GINR) method on the provided data. Three different activation functions were tested within a single neuron:

- A Linear function (LU): $f(X) = WX + b$.
- A Sigmoid Linear Unit function (SELU): $f(X) = \frac{WX+b}{1+\exp[-(WX+b)]}$

- A Quadratic function (QD) $f(X) = (W_1X') \cdot (W_2X) + b$

Table 4.4 reports the residual mean squared errors (RMSE) for the travel time data on the same $(0, 1)$ arc using MPINR and GINR. These results show that both MPINR and GINR are effective and competitive with the two and three-layer ANNs tested. The lowest RMSE obtained (0.6265) for both method using the QD function is already better than all of those obtained by two-layer ANNs, and most of the three-layer ANNs tested. These results confirm the promising potential offered by Newton-Raphson methods for Neural Networks gradient descent optimizers.

Table 4.4: RMSE of travel time forecasts using MPINR vs. GINR

Function	MPINR	GINR	Avg.
LU	0.7273	0.7273	0.7273
SELU	0.7275	0.7275	0.7275
QD	0.6265	0.6265	0.6265
Avg.	0.6937	0.6937	0.6937

4.8 Conclusion

This chapter presented a complete forecasting framework for the dial-a-ride problem with time windows (DARPTW). It reviewed forecasting and solution methods for the DARPTW, and then presented a two-layer and a three-layer artificial neural network (ANN) approach for DARPTW travel and service times.

This approach was used to predict travel times and service times on a real-life dataset. Four types of activation functions and four different gradient descent optimizers were tested. Experimental results showed that the proposed ANN method produces low RMSE. Additionally, the results show that the ReLU activation functions combined with stochastic gradient descent (SGD) optimizer provide the best forecasting results for two-layer and three-layer neural networks.

Finally, the experimental results also confirm the promising potential of Newton-Raphson gradient descent methods if properly integrated within artificial neural networks.

CHAPTER 5

GENERAL CONCLUSION

This thesis has introduced novel optimization techniques aimed at advancing both linear and non-linear optimization for complex real-world problems. By developing a multiple-pivot simplex algorithm, a Modified Pseudo-Inverse Newton-Raphson (MP-INR) method, and a neural network-based forecasting approach for the Dial-A-Ride Problem with Time Windows (DARPTW), this work provides new tools that enhance computational efficiency, stability, and practical applicability across diverse optimization contexts.

The multiple-pivot simplex algorithm significantly improves the efficiency of linear programming by allowing multiple pivots in each iteration, reducing the computational burden for large-scale problems. The MPINR method for non-linear optimization leverages the Moore-Penrose pseudo-inverse to address singularity challenges in the Hessian matrix, enhancing the convergence and stability of optimization in artificial neural networks. Additionally, the ANN-based forecasting approach applied to the DARPTW demonstrates the practical potential of machine learning in addressing complex scheduling and routing problems, ultimately leading to more efficient and

customer-oriented transportation services.

Together, these contributions emphasize the importance of innovation in optimization algorithms to address emerging challenges in computational and applied mathematics. Future work could focus on extending these methods to broader problem domains, exploring their adaptability to different model structures, and investigating additional applications in fields such as logistics, finance, and engineering. This thesis thus lays a foundation for continued advancement in optimization, supporting more effective solutions to complex problems across various industries.

APPENDIX A

PROOFS

APPENDIX B

SUPPLEMENTARY MATERIAL

REFERENCES

- [1] M. Hagan, H. Demuth, M. Beale, and O. De Jesús, *Neural Network Design*.
Martin Hagan, 2014. [Online]. Available: <https://books.google.com.sa/books?id=4EW9oQEACAAJ>
- [2] G. S. da S. Gomes, T. B. Ludermir, and L. M. Lima, “Comparison of new activation functions in neural network for forecasting financial time series,” *Neural Computing and Applications*, vol. 20, pp. 417–439, 2011.
- [3] G. B. Dantzig, “Maximization of a linear function of variables subject to linear inequalities,” *Activity analysis of production and allocation*, vol. 13, pp. 339–347, 1951.
- [4] G. B. Dantzig and W. Orchard-Hays, “The product form for the inverse in the simplex method,” *Mathematical Tables and Other Aids to Computation*, pp. 64–67, 1954.
- [5] V. Klee, G. J. Minty, and O. Shisha, “Inequalities, iii,” *How Good is the Simplex Algorithm*, pp. 159–175, 1972.

- [6] D. A. Spielman and S.-H. Teng, “Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time,” *Journal of the ACM (JACM)*, vol. 51, no. 3, pp. 385–463, 2004.
- [7] M. Huang, Y. Zhong, H. Yang, J. Wang, F. Zhang, B. Bai, and L. Shi, “Simplex initialization: A survey of techniques and trends,” *arXiv preprint arXiv:2111.03376*, 2021.
- [8] N. A. A. N. Azlan, E. Mohamad, M. R. Salleh, O. Novareza, D. Yuniawan, M. A. A. Rahman, A. Saptari, and M. A. Sulaiman, “Multiplicity of approach and method in augmentation of simplex method: A review,” *Mathematics and Statistics*, vol. 8, no. 1, pp. 53–80, 2020.
- [9] P. Tolla, “A stable and sparsity exploiting lu factorization of the basis matrix in linear programming,” *European journal of operational research*, vol. 24, no. 2, pp. 247–251, 1986.
- [10] U. H. Suhl and L. Suhl, “Computing sparse lu factorizations for large-scale linear programming bases,” *INFORMS J. Comput.*, vol. 2, pp. 325–335, 1990.
[Online]. Available: <https://api.semanticscholar.org/CorpusID:207225465>
- [11] I. Elhallaoui, A. Metrane, G. Desaulniers, and F. Soumis, “An improved primal simplex algorithm for degenerate linear programs,” *INFORMS Journal on Computing*, vol. 23, no. 4, pp. 569–577, 2011.

- [12] V. Raymond, F. Soumis, and D. Orban, “A new version of the improved primal simplex for degenerate linear programs,” *Computers & Operations Research*, vol. 37, no. 1, pp. 91–98, 2010.
- [13] L. R. Ford Jr and D. R. Fulkerson, “A suggested computation for maximal multi-commodity network flows,” *Management Science*, vol. 50, no. 12_supplement, pp. 1778–1780, 2004.
- [14] G. B. Dantzig and P. Wolfe, “Decomposition principle for linear programs,” *Operations research*, vol. 8, no. 1, pp. 101–111, 1960.
- [15] P. C. Gilmore and R. E. Gomory, “A linear programming approach to the cutting-stock problem,” *Operations research*, vol. 9, no. 6, pp. 849–859, 1961.
- [16] L. H. Appelgren, “A column generation algorithm for a ship scheduling problem,” *Transportation Science*, vol. 3, no. 1, pp. 53–68, 1969.
- [17] F. Vitor and T. Easton, “The double pivot simplex method,” *Mathematical Methods of Operations Research*, vol. 87, pp. 109–137, 2018.
- [18] Y. Yang, “A double-pivot simplex algorithm and its upper bounds of the iteration numbers,” *Research in the Mathematical Sciences*, vol. 7, no. 4, p. 34, 2020.
- [19] F. Sullivan and J. Dongarra, “Guest editors’ introduction: The top 10 algorithms,” *Computing in Science & Engineering*, vol. 2, no. 1, pp. 22–23, 2000.

- [20] L. V. Kantorovich, “Mathematical methods of organizing and planning production,” *Management science*, vol. 6, no. 4, pp. 366–422, 1960.
- [21] T. C. Koopmans, “Optimum utilization of the transportation system,” *Econometrica: Journal of the Econometric Society*, pp. 136–146, 1949.
- [22] R. Dorfman, “The discovery of linear programming,” *Annals of the History of Computing*, vol. 6, no. 3, pp. 283–295, 1984.
- [23] E. K. Lee, R. J. Gallagher, and D. A. Patterson, “A linear programming approach to discriminant analysis with a reserved-judgment region,” *INFORMS Journal on Computing*, vol. 15, no. 1, pp. 23–41, 2003.
- [24] R. Mansini, W. Ogryczak, and M. G. Speranza, “Conditional value at risk and related linear programming models for portfolio optimization,” *Annals of operations research*, vol. 152, pp. 227–256, 2007.
- [25] A. M. Gomes and J. F. Oliveira, “Solving irregular strip packing problems by hybridising simulated annealing and linear programming,” *European Journal of Operational Research*, vol. 171, no. 3, pp. 811–829, 2006.
- [26] J. Spitter, C. A. Hurkens, A. De Kok, J. K. Lenstra, and E. G. Negenman, “Linear programming models with planned lead times for supply chain operations planning,” *European Journal of operational research*, vol. 163, no. 3, pp. 706–720, 2005.

- [27] A. Gautier, B. F. Lamond, D. Pare, and F. Rouleau, “The quebec ministry of natural resources uses linear programming to understand the wood-fiber market,” *Interfaces*, vol. 30, no. 6, pp. 32–48, 2000.
- [28] P. M. Harris, “Pivot selection methods of the devex lp code,” *Mathematical programming*, vol. 5, pp. 1–28, 1973.
- [29] D. Goldfarb and W. Y. Sit, “Worst case behavior of the steepest edge simplex method,” *Discrete Applied Mathematics*, vol. 1, no. 4, pp. 277–285, 1979.
- [30] R. G. Bland, “New finite pivoting rules for the simplex method,” *Mathematics of operations Research*, vol. 2, no. 2, pp. 103–107, 1977.
- [31] H.-D. Chen, P. M. Pardalos, and M. A. Saunders, “The simplex algorithm with a new primal and dual pivot rule,” *Operations Research Letters*, vol. 16, no. 3, pp. 121–127, 1994.
- [32] H. V. Junior and M. P. E. Lins, “An improved initial basis for the simplex algorithm,” *Computers Operations Research*, vol. 32, no. 8, pp. 1983–1993, 2005.
- [33] F. Djannaty and B. Rostami, “A computational approach to pivot selection in the lp relaxation of set problems,” *Journal of Applied Mathematics and Decision Sciences*, vol. 2006, 11 2006.
- [34] P.-Q. Pan, “A largest-distance pivot rule for the simplex algorithm,” *European journal of operational research*, vol. 187, no. 2, pp. 393–402, 2008.

- [35] P. Pan, “A fast simplex algorithm for linear programming,” *Journal of Computational Mathematics*, pp. 837–847, 2010.
- [36] J. Omer, S. Rosat, V. Raymond, and F. Soumis, “Improved primal simplex: a more general theoretical framework and an extended experimental analysis,” *INFORMS Journal on Computing*, vol. 27, no. 4, pp. 773–787, 2015.
- [37] K. Chankong, B. Intiyot, and K. Sinapiromsaran, “Absolute change pivot rule for the simplex algorithm,” in *Proceedings of the International MultiConference of Engineers and Computer Scientists*, vol. 2, 2014.
- [38] Q. Huangfu and J. J. Hall, “Parallelizing the dual revised simplex method,” *Mathematical Programming Computation*, vol. 10, no. 1, pp. 119–142, 2018.
- [39] H. B. Fine, “On newton’s method of approximation,” *Proceedings of the National Academy of Sciences*, vol. 2, no. 9, pp. 546–552, 1916.
- [40] A. A. Bennett, “Newton’s method in general analysis,” *Proceedings of the National Academy of Sciences*, vol. 2, no. 10, pp. 592–598, 1916.
- [41] L. Kantorovitch, “The method of successive approximation for functional equations,” *Acta Mathematica*, vol. 71, no. 1, pp. 63–97, 1939.
- [42] L. V. Kantorovich, “On newton’s method,” *Trudy Matematicheskogo Instituta imeni VA Steklova*, vol. 28, pp. 104–144, 1949.
- [43] I. P. Mysovskikh, “On the convergence of newton’s method,” *Trudy Matematicheskogo Instituta imeni VA Steklova*, vol. 28, pp. 145–147, 1949.

- [44] J. M. Ortega and W. C. Rheinboldt, *Iterative solution of nonlinear equations in several variables*. SIAM, 2000.
- [45] P. Deuffhard, *Newton methods for nonlinear problems: affine invariance and adaptive algorithms*. Springer Science & Business Media, 2011, vol. 35.
- [46] P. Deuffhard and G. Heindl, “Affine invariant convergence theorems for newton’s method and extensions to related methods,” *SIAM Journal on Numerical Analysis*, vol. 16, no. 1, pp. 1–10, 1979.
- [47] P. Deuffhard and F. A. Potra, “Asymptotic mesh independence of newton–galerkin methods via a refined mysovskii theorem,” *SIAM Journal on Numerical Analysis*, vol. 29, no. 5, pp. 1395–1412, 1992.
- [48] A. Ben-Israel, “A newton-raphson method for the solution of systems of equations,” *Journal of Mathematical analysis and applications*, vol. 15, no. 2, pp. 243–252, 1966.
- [49] E. L. Allgower, K. Böhmer, A. Hoy, and V. Janovský, “Direct methods for solving singular nonlinear equations,” *ZAMM-Journal of Applied Mathematics and Mechanics/Zeitschrift für Angewandte Mathematik und Mechanik: Applied Mathematics and Mechanics*, vol. 79, no. 4, pp. 219–231, 1999.
- [50] X. Chen, Z. Nashed, and L. Qi, “Convergence of newton’s method for singular smooth and nonsmooth equations using adaptive outer inverses,” *SIAM Journal on Optimization*, vol. 7, no. 2, pp. 445–462, 1997.

- [51] J. Dedieu and M. Shub, “Newton’s method for overdetermined systems of equations,” *Mathematics of Computation*, vol. 69, no. 231, pp. 1099–1115, 2000.
- [52] Y. Levin and A. Ben-Israel, “A newton method for systems of m equations in n variables,” *Nonlinear Analysis-Theory Methods and Applications*, vol. 47, no. 3, pp. 1961–1972, 2001.
- [53] A. Galántai, “The theory of newton’s method,” *Journal of Computational and Applied Mathematics*, vol. 124, no. 1-2, pp. 25–44, 2000.
- [54] L. Qi and J. Sun, “A nonsmooth version of newton’s method,” *Mathematical programming*, vol. 58, no. 1, pp. 353–367, 1993.
- [55] M. Nashed and X. Chen, “Convergence of newton-like methods for singular operator equations using outer inverses,” *Numerische Mathematik*, vol. 66, no. 1, pp. 235–257, 1993.
- [56] H. F. Walker, “Newton-like methods for underdetermined systems,” *Computational solution of nonlinear systems of equations*, pp. 679–699, 1990.
- [57] D. W. Decker and C. T. Kelley, “Newton’s method at singular points. i,” *SIAM Journal on Numerical Analysis*, vol. 17, no. 1, pp. 66–70, 1980.
- [58] N. Wilson and M. I. of Technology. Urban Systems Laboratory, *Scheduling Algorithms for a Dial-a-ride System*, ser. PB 201 808. Massachusetts Institute of Technology, Urban Systems Laboratory, 1971. [Online]. Available: <https://books.google.com.sa/books?id=LRTMZwEACAAJ>

- [59] J.-F. Cordeau and G. Laporte, “The dial-a-ride problem (darp): Variants, modeling issues and algorithms,” *Quarterly Journal of the Belgian, French and Italian Operations Research Societies*, vol. 1, pp. 89–101, 2003. [Online]. Available: <https://api.semanticscholar.org/CorpusID:38556368>
- [60] S. Ho, W. Szeto, Y.-H. Kuo, J. Leung, M. Petering, and T. Tou, “A survey of dial-a-ride problems: Literature review and recent developments,” *Transportation Research Part B: Methodological*, vol. 111, pp. 395–421, 02 2018.
- [61] J.-F. Cordeau, “A branch-and-cut algorithm for the dial-a-ride problem,” *Operations Research*, vol. 54, pp. 573–586, 06 2006.
- [62] M. Liu, Z. Luo, and A. Lim, “A branch-and-cut algorithm for a realistic dial-a-ride problem,” *Transportation Research Part B: Methodological*, vol. 81, pp. 267–288, 2015.
- [63] G. Heilporn, J.-F. Cordeau, and G. Laporte, “An integer l-shaped algorithm for the dial-a-ride problem with stochastic customer delays,” *Discrete Applied Mathematics*, vol. 159, no. 9, pp. 883–895, 2011.
- [64] T. Garaix, C. Artigues, D. Feillet, and D. Josselin, “Optimization of occupancy rate in dial-a-ride problems via linear fractional column generation,” *Computers & Operations Research*, vol. 38, no. 10, pp. 1435–1442, 2011.
- [65] A. Florio, R. Hartl, S. Minner, and J.-J. Salazar-González, “A branch-and-price algorithm for the vehicle routing problem with stochastic demands and

- probabilistic duration constraints,” *Transportation Science*, vol. 55, no. 1, pp. 122–138, 2021.
- [66] Y. Qu and J. F. Bard, “A branch-and-price-and-cut algorithm for heterogeneous pickup and delivery problems with configurable vehicle capacity,” *Transportation Science*, vol. 49, no. 2, pp. 254–270, 2015. [Online]. Available: <https://api.semanticscholar.org/CorpusID:21755720>
- [67] J.-J. Jaw, A. R. Odoni, H. N. Psaraftis, and N. H. Wilson, “A heuristic algorithm for the multi-vehicle advance request dial-a-ride problem with time windows,” *Transportation Research Part B: Methodological*, vol. 20, no. 3, pp. 243–257, 1986.
- [68] M. Diana and M. M. Dessouky, “A new regret insertion heuristic for solving large-scale dial-a-ride problems with time windows,” *Transportation Research Part B: Methodological*, vol. 38, no. 6, pp. 539–557, 2004.
- [69] O. Madsen, Ravn, H.F., and J. Rygaard, “A heuristic algorithm for a dial-a-ride problem with time windows, multiple capacities, and multiple objectives,” *Annals of Operations Research*, vol. 60, p. 193–208, 1995.
- [70] F. Glover, “Tabu search - part i,” *ORSA Journal on Computing*, vol. 1, no. 3, pp. 190–206, 1989.
- [71] J.-F. Cordeau and G. Laporte, “A tabu search heuristic for the static multi-vehicle dial-a-ride problem,” *Transportation Research Part B: Methodological*, vol. 37, no. 6, pp. 579–594, 2003.

- [72] A. Attanasio, J.-F. Cordeau, G. Ghiani, and G. Laporte, “Parallel tabu search heuristics for the dynamic multi-vehicle dial-a-ride problem,” *Parallel Computing*, vol. 30, pp. 377–387, 03 2004.
- [73] J. Paquette, J.-F. Cordeau, and G. Laporte, “Quality of service in dial-a-ride operations,” *Computers & Industrial Engineering*, vol. 56, no. 4, pp. 1721–1734, 2009.
- [74] N. Mladenović and P. Hansen, “Variable neighborhood search,” *Computers Operations Research*, vol. 24, no. 11, pp. 1097–1100, 1997.
- [75] H. R. Parragh S, Doerner K, “Variable neighborhood search for the dial-a-ride problem,” *Computers & operations research*, vol. 37, no. 6, p. 1129–1138, 2010.
- [76] M. Schilde, K. Doerner, and R. Hartl, “Metaheuristics for the dynamic stochastic dial-a-ride problem with expected return transports,” *Computers Operations Research*, vol. 38, no. 12, pp. 1719–1730, 2011.
- [77] S. Belhaiza, “A game theoretic approach for the real-life multiple-criterion vehicle routing problem with multiple time windows,” *IEEE Systems Journal*, vol. 12, no. 2, pp. 1251–1262, 2018.
- [78] —, “A hybrid adaptive large neighborhood heuristic for a real-life dial-a-ride problem,” *Algorithms*, vol. 12, no. 2, 2019.
- [79] R. Masson, F. Lehuédé, and O. Péton, “The dial-a-ride problem with transfers,” *Computers & Operations Research*, vol. 41, pp. 12–23, 2014.

- [80] Y. Molenbruch, K. Braekers, A. Caris, and G. V. Berghe, “Multi-directional local search for a bi-objective dial-a-ride problem in patient transportation,” *Computers & Operations Research*, vol. 77, pp. 58–71, 2017.
- [81] S. N. Parragh and V. Schmid, “Hybrid column generation and large neighborhood search for the dial-a-ride problem,” *Computers & Operations Research*, vol. 40, no. 1, pp. 490–497, 2013.
- [82] S. Belhaiza, P. Hansen, and G. Laporte, “A hybrid variable neighborhood tabu search heuristic for the vehicle routing problem with multiple time windows,” *Computers & Operations Research*, vol. 52, 08 2013.
- [83] S. Belhaiza, R. M’Hallah, and G. Brahim, “A new hybrid genetic variable neighborhood search heuristic for the vehicle routing problem with multiple time windows,” 06 2017, pp. 1319–1326.
- [84] S. Belhaiza, “A data driven hybrid heuristic for the dial-a-ride problem with time windows,” in *IEEE Symposium Series on Computational Intelligence (SSCI)*, 2017, pp. 1–8.
- [85] R. Chevrier, A. Liefvooghe, L. Jourdan, and C. Dhaenens, “Solving a dial-a-ride problem with a hybrid evolutionary multi-objective approach: Application to demand responsive transport,” *Applied Soft Computing*, vol. 12, no. 4, pp. 1247–1258, 2012.
- [86] A. Beaudry, G. Laporte, T. Melo, and S. Nickel, “Dynamic transportation of patients in hospitals,” *OR Spectrum*, vol. 32, pp. 77–107, 04 2010.

- [87] P. Detti, F. Papalini, and G. Z. M. de Lara, “A multi-depot dial-a-ride problem with heterogeneous vehicles and compatibility constraints in healthcare,” *Omega*, vol. 70, pp. 1–14, 2017.
- [88] Y. D. Jacques Desrosiers and F. Soumis, “A dynamic programming solution of the large-scale single-vehicle dial-a-ride problem with time windows,” *American Journal of Mathematical and Management Sciences*, vol. 6, no. 3-4, pp. 301–325, 1986.
- [89] A. Ziliaskopoulos and G. Kozanidis, “Dynamic programming strategies for the dial a ride problem with time window constraints,” *85th Annual Meeting of the Transportation Research Board, Washington, DC, USA*, pp. 1–12, January 22-26 2006.
- [90] K. Braekers, A. Caris, and G. K. Janssens, “Exact and meta-heuristic approach for a general heterogeneous dial-a-ride problem with multiple depots,” *Transportation Research Part B: Methodological*, vol. 67, no. C, pp. 166–186, 2014.
- [91] I. Zidi, K. Mesghouni, K. Zidi, and K. Ghedira, “A multi-objective simulated annealing for the multi-criteria dial a ride problem,” *Engineering Applications of Artificial Intelligence*, vol. 25, no. 6, pp. 1121–1131, 2012.
- [92] D. Kirchler and R. W. Calvo, “A granular tabu search algorithm for the dial-a-ride problem,” *Transportation Research Part B: Methodological*, vol. 56, pp. 120–135, 2013.

- [93] B. Rekiek, A. Delchambre, and H. A. Saleh, “Handicapped person transportation: An application of the grouping genetic algorithm,” *Engineering Applications of Artificial Intelligence*, vol. 19, no. 5, pp. 511–520, 2006.
- [94] R. M. Jorgensen, J. Larsen, and K. B. Bergvinsdottir, “Solving the dial-a-ride problem using genetic algorithms,” *Journal of the operational research society*, vol. 58, no. 10, pp. 1321–1331, 2007.
- [95] G. De Luca and M. Gallo, “Artificial neural networks for forecasting user flows in transportation networks: literature review, limits, potentialities and open challenges,” in *2017 5th IEEE international conference on models and technologies for intelligent transportation systems (MT-ITS)*. IEEE, 2017, pp. 919–923.
- [96] W. B. Powell, *A Comparative Review of Alternative Algorithms for the Dynamic Vehicle Allocation Problem. Vehicle Routing: Methods and Studies. Studies in Manegment Science and Systems-Volume 16*. Dalctraf, 1988.
- [97] —, “A stochastic formulation of the dynamic assignment problem, with an application to truckload motor carriers,” *Transportation Science*, vol. 30, no. 3, pp. 195–219, 1996.
- [98] H. Marković, I. Cavar, and T. Caric, “Using data mining to forecast uncertain demands in stochastic vehicle routing problem,” in *13th International Symposium on Elecronics in Transport (ISEP), Slovenia*, 2005, pp. 1–6.

- [99] T. Toledo and T. Kolechkina, “Estimation of dynamic origin–destination matrices using linear assignment matrix approximations,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 14, no. 2, pp. 618–626, 2012.
- [100] W. Ma and Z. S. Qian, “Statistical inference of probabilistic origin-destination demand using day-to-day traffic data,” *Transportation Research Part C: Emerging Technologies*, vol. 88, pp. 227–256, 2018.
- [101] X. Zhou, X. Qin, and H. S. Mahmassani, “Dynamic origin-destination demand estimation with multiday link traffic counts for planning applications,” *Transportation Research Record*, vol. 1831, no. 1, pp. 30–38, 2003.
- [102] X. Zhou and H. S. Mahmassani, “Dynamic origin-destination demand estimation using automatic vehicle identification data,” *IEEE Transactions on intelligent transportation systems*, vol. 7, no. 1, pp. 105–114, 2006.
- [103] —, “A structural state space model for real-time traffic origin–destination demand estimation and prediction in a day-to-day learning framework,” *Transportation Research Part B: Methodological*, vol. 41, no. 8, pp. 823–840, 2007.
- [104] A. Ermagun and D. Levinson, “Spatiotemporal traffic forecasting: review and proposed directions,” *Transport Reviews*, vol. 38, no. 6, pp. 786–814, 2018.
- [105] C. Goves, R. North, R. Johnston, and G. Fletcher, “Short term traffic prediction on the uk motorway network using neural networks,” *Transportation Research Procedia*, vol. 13, pp. 184–195, 2016.

- [106] Y. Lv, Y. Duan, W. Kang, Z. Li, and F.-Y. Wang, “Traffic flow prediction with big data: A deep learning approach,” *Ieee transactions on intelligent transportation systems*, vol. 16, no. 2, pp. 865–873, 2014.
- [107] N. G. Polson and V. O. Sokolov, “Deep learning for short-term traffic flow prediction,” *Transportation Research Part C: Emerging Technologies*, vol. 79, pp. 1–17, 2017.
- [108] J. Van Lint, S. Hoogendoorn, and H. J. van Zuylen, “Accurate freeway travel time prediction with state-space neural networks under missing data,” *Transportation Research Part C: Emerging Technologies*, vol. 13, no. 5-6, pp. 347–369, 2005.
- [109] J. Zamarreno, P. Vega, L. Garcia, and M. Francisco, “State-space neural network for modelling, prediction and control,” *Control Engineering Practice*, vol. 8, no. 9, pp. 1063–1075, 2000.
- [110] S. Oh, Y.-J. Byon, K. Jang, and H. Yeo, “Short-term travel-time prediction on highway: a review of the data-driven approach,” *Transport Reviews*, vol. 35, no. 1, pp. 4–32, 2015.
- [111] E. I. Vlahogianni, M. G. Karlaftis, and J. C. Golias, “Short-term traffic forecasting: Where we are and where we’re going,” *Transportation Research Part C: Emerging Technologies*, vol. 43, pp. 3–19, 2014.

- [112] A. Abadi, T. Rajabioun, and P. A. Ioannou, “Traffic flow prediction for road transportation networks with limited traffic data,” *IEEE transactions on intelligent transportation systems*, vol. 16, no. 2, pp. 653–662, 2014.
- [113] C. Furtlehner, J.-M. Lasgouttes, A. Attanasi, L. Meschini, and M. Pezzulla, “Spatio-temporal probabilistic short-term forecasting on urban networks,” Ph.D. dissertation, Inria Saclay-Île de France; Inria de Paris; PTV-SISTeMA, 2019.
- [114] A. Attanasi, L. Meschini, M. Pezzulla, G. Fusco, G. Gentile, and N. Isaenko, “A hybrid method for real-time short-term predictions of traffic flows in urban areas,” in *2017 5th IEEE International Conference on Models and Technologies for Intelligent Transportation Systems (MT-ITS)*. IEEE, 2017, pp. 878–883.
- [115] M. Yang, Y. Liu, and Z. You, “The reliability of travel time forecasting,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 11, no. 1, pp. 162–171, 2009.
- [116] Z. Wang, A. Goodchild, and E. McCormack, “A methodology for forecasting freeway travel time reliability using gps data,” *Transportation research procedia*, vol. 25, pp. 842–852, 2017.
- [117] M. Juhász, T. Mátrai, and C. Koren, “Forecasting travel time reliability in urban road transport,” *Archives of Transport*, vol. 43, no. 3, pp. 53–67, 2017.
- [118] Z. Chen and W. Fan, “Data analytics approach for travel time reliability pattern analysis and prediction,” *Journal of Modern Transportation*, vol. 27, no. 4, pp. 250–265, 2019.

- [119] E. Kidando, R. Moses, E. E. Ozguven, and T. Sando, “Incorporating travel time reliability in predicting the likelihood of severe crashes on arterial highways using non-parametric random-effect regression,” *Journal of traffic and transportation engineering (English edition)*, vol. 6, no. 5, pp. 470–481, 2019.
- [120] K. Hara and K. Nakayamma, “Comparison of activation functions in multilayer neural network for pattern classification,” in *Proceedings of 1994 IEEE International Conference on Neural Networks (ICNN’94)*, vol. 5. IEEE, 1994, pp. 2997–3002.
- [121] M. Ö. Efe, “Novel neuronal activation functions for feedforward neural networks,” *Neural Processing Letters*, vol. 28, pp. 63–79, 2008.
- [122] W. Duch and N. Jankowski, “Survey of neural transfer functions,” *Neural computing surveys*, vol. 2, no. 1, pp. 163–212, 1999.
- [123] Y. Singh and P. Chandra, “A class+ 1 sigmoidal activation functions for ffnns,” *Journal of Economic Dynamics and Control*, vol. 28, no. 1, pp. 183–187, 2003.
- [124] G. Stewart, “Stewart, g.w.: On the early history of the singular value decomposition. siam rev. 35(4), 551-566,” *SIAM Review*, vol. 35, 03 2001.
- [125] E. Silver, D. Pyke, and D. Thomas, *Inventory and Production Management in Supply Chains*. Taylor & Francis, 2016.
- [126] D. Kwiatkowski, P. C. Phillips, P. Schmidt, and Y. Shin, “Testing the null hypothesis of stationarity against the alternative of a unit root: How sure are

we that economic time series have a unit root?" *Journal of econometrics*, vol. 54,
no. 1-3, pp. 159–178, 1992.

VITAE

- Name: Munirah Farhan Saad AlQarni
- Nationality: Saudi
- Date of Birth: September 21,1973.
- Email: *mfs.alqarni@gmail.com*
- Permenant Address: 4187,13 street, AlYarmok Dist.,Postal Code 34443, AL Khobar, Kingdom of Saudi Arabia.

Academic qualifications:

- PhD in Mathematics from King Fahd University of Petroleum and Minerals (KFUPM), Dhahran, Saudi Arabia, November 2024.
- Master of Science in Applied Mathematics from Damman University, Damman, Saudi Arabia, 2011.

Thesis title: "Structure Methods for Solving the Nearest Correlation Matrix Problem".

- Bachelor of Science (with honors) in Mathematics from the Science College, Dammam, Saudi Arabia, 1996.

Publication:

- Slim Belhaiza, Rym M'Hallah, Munirah Al-Qarni, A data-driven game theoretic multi-objective hybrid algorithm for the Dial-A-Ride Problem with multiple time windows, Transportation Research Part A: Policy and Practice, Volume 178, 2023, 103862, ISSN 0965-8564, <https://doi.org/10.1016/j.tra.2023.103862>.
- S. Belhaiza, R. M'Hallah and M. Al-Qarni, "A Data-Driven Forecasting and Solution Approach for the Dial-A-Ride Problem with Time Windows," 2022 IEEE Symposium Series on Computational Intelligence (SSCI), Singapore, Singapore, 2022, pp. 101-110, doi: 10.1109/SSCI51031.2022.10022259.
- Al-Homidan, S., AlQarni, M. Structure methods for solving the nearest correlation matrix problem. Positivity 16, 497–508 (2012). <https://doi.org/10.1007/s11117-012-0180-x>