วิธีซิมเพล็กซ์แบบกระโดดก่อน

นายณัฐดนัย กาฝากทอง

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต
สาขาวิชาคณิตศาสตร์ประยุกต์และวิทยาการคณนา
ภาควิชาคณิตศาสตร์และวิทยาการคอมพิวเตอร์
คณะวิทยาศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2561

PRECEDING-JUMP SIMPLEX METHOD

Mr. Natdanai Kafakthong

A Thesis Submitted in Partial Fulfillment of the Requirements

for the Degree of Master of Science Program in Applied Mathematics and

Computational Science

Department of Mathematics and Computer Science

Faculty of Science

Chulalongkorn University

Academic Year 2018

Thesis Title	PRECEDING-JUMP SIMPLEX METHOD
By	Mr. Natdanai Kafakthong
Field of Study	Applied Mathematics and Computational Science
Thesis Advisor	Assistant Professor Krung Sinapiromsaran, Ph.D.

Accepted by the Faculty of Science, Chulalongkorn University in Partial Fulfillment of the Requirements for the Master's Degree

..... Dean of the Faculty of Science
(Professor Polkit Sangvanich, Ph.D.)

THESIS COMMITTEE

..... Chairman
(Associate Professor Phantipa Thipwiwatpotjana, Ph.D.)

..... Thesis Advisor
(Assistant Professor Krung Sinapiromsaran, Ph.D.)

..... Examiner
(Assistant Professor Boonyarit Intiyot, Ph.D.)

..... External Examiner
(Aua-aree Boonperm, Ph.D.)

ณัฐดนัย กาฝากทอง : วิธีซิมเพล็กซ์แบบกระโดดก่อน. (PRECEDING-JUMP SIMPLEX METHOD) อ.ที่ปรึกษาวิทยานิพนธ์หลัก : ผศ.ดร.กรุง สินอภิรมย์สราญ.

ซอฟต์แวร์แก้ปัญหากำหนดการเชิงเส้นในเชิงปฏิบัติใช้วิธีซิมเพล็กซ์เพื่อระบุผลเฉลยที่เหมาะที่สุดโดยการค้นแบบซ้ำจากบรรดาจุดยอดที่เป็นไปได้ที่อยู่ติดกัน วิธีนี้อาจเยี่ยมจุดยอดมากมายก่อนที่จะพบผลเฉลยที่เหมาะที่สุดสำหรับปัญหากำหนดการเชิงเส้นขนาดใหญ่ ซึ่งใช้ระยะเวลานานก่อนหยุด งานวิจัยนี้เสนอวิธีฮิวริสติกเพื่อบรรเทาประเด็นนี้โดยการกระโดดไปจุดยอดใหม่ใกล้กับจุดยอดที่เหมาะที่สุดก่อนใช้วิธีซิมเพล็กซ์ จุดยอดใหม่จะถูกระบุด้วยการ กระโดดครั้งแรกจากจุดยอดเริ่มต้นตามทิศทางของเวกเตอร์เกรเดียนต์ของฟังก์ชันจุดประสงค์หรือทิศทางเลือกอื่นในกรณีทิศทางของฟังก์ชันจุดประสงค์ชี้ออกนอกบริเวณที่เป็นไปได้ จุดยอดดังกล่าวจะหยุดที่จุดที่เป็นไปได้จุดแรกซึ่งผูกกับเงื่อนไขหนึ่งเงื่อนไข จุดที่เป็นไปได้ใหม่นั้นจะถูกเลื่อนไปยังจุดเพื่อนบ้านที่เป็นไปได้ถัดไปโดยคงเงื่อนไขที่ผูกไว้ก่อนหน้า จนกระทั่งวิธีนี้เลื่อนไปถึงจุดยอดที่เป็นไปได้เพื่อเริ่มวิธีซิมเพล็กซ์ ขั้นตอนวิธีนี้ถูกทดสอบกับปัญหาสังเคราะห์ซึ่งจุดศูนย์เป็นจุดยอดเริ่มต้นจาก 100 ถึง 2500 ตัวแปร ซึ่งมีจำนวนตัวแปรเท่ากับจำนวนเงื่อนไขบังคับ มากไปกว่านั้นวิธีซิมเพล็กซ์แบบกระโดดก่อนขยายเพื่อนำไปแก้ปัญหากำหนดการเชิงเส้นทั่วไปตั้งแต่ 100 ถึง 1000 ตัวแปร จากผลการทดลองพบว่าวิธีซิมเพล็กซ์แบบกระโดดก่อนช่วยลดจำนวนการทำซ้ำและเวลาทำงานอย่างมีนัยสำคัญ

ภาควิชา คณิตศาสตร์และวิทยาการคอมพิวเตอร์ ลายมือชื่อนิสิต .....

ลายมือชื่อ อ.ที่ปรึกษาหลัก .....

สาขาวิชา คณิตศาสตร์ประยุกต์และวิทยาการคณนา

ปีการศึกษา 2561

6072049323 : MAJOR APPLIED MATHEMATICS AND COMPUTATIONAL SCIENCE
KEYWORDS : LINEAR PROGRAMMING / SIMPLEX METHOD / FEASIBLE VERTEX,JUMP

NATDANAI KAFAKTHONG : PRECEDING-JUMP SIMPLEX METHOD. ADVISOR : ASST. PROF. KRUNG SINAPIROMSARAN, Ph.D.

A practical linear programming solver uses the simplex method to identify the optimal solution by iteratively searching among adjacent feasible vertices. It may visit numerous vertices before finding the optimal one for a large linear programming problem taking a long time to succeed. This research proposes a heuristic method to alleviate this issue by jumping to a new vertex close to the optimal one before performing the simplex method. The new vertex is identified by first jumping from the initial vertex along the gradient vector of the objective function or alternative direction if the objective direction points out of the feasible region and stops at the first feasible point binding at one constraint. It then shifts to a neighbor feasible point preserving previous binding constraints until it reaches the feasible vertex so the simplex method can start. The algorithm is tested on synthetic problems with the origin point as the initial vertex from 100 to 2500 variable having the same number of constraints. Moreover, the preceding jump simplex method is extended to solve general synthetic linear programming problems of 100 to 1000 variables. The experimental results show that the preceding-jump simplex method significantly reduces the number of iterations and total running time.

Department : Mathematics and Computer Science
Field of Study : Applied Mathematics and Computational Science
Academic Year : 2018

Student's Signature .....
Advisor's Signature .....

ACKNOWLEDGEMENTS

I would like to express my deepest and sincere gratitude to Assistant Professor Dr. Krung Sinapiromsaran, my advisor, for his invaluable advice, assistance and constructive criticism in the preparation of this Master thesis. Furthermore, I also sincerely thank Associate Professor Dr. Phantipa Thipwiwatpotjana, and Assistant Professor Dr. Boonyarit Intiyot, the examiners of this graduate thesis for their helpful suggestions. Moreover, I would like to thank the development and promotion of science and technology talents project (DPST) for my scholarship which always supports all my expenses during this work. I would like to express my deep thank to my parents for their supports and understanding. Finally, I would like to thank you all Ph.D. students who have given all suggestions to me.

CONTENTS

Page

CHAPTER Page

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

This chapter will provide an overview of optimization methods which covers important topics related to this research. The subchapter states an optimization problem in general, a linear programming model [1] and a review of the standard method for solving a linear programming problem including the basic concept of the jump method. The objectives and scopes of the thesis will be presented at the end of this chapter.

1.1 Optimization problem in general

Most real-world optimization problems are subject to many restrictions that it is difficult to intuitively decide what is the best solution. One way of solving this issue is to apply the mathematical concept to the problem. Normally, the problem will be transformed into a mathematical model and then be solved by an optimization solver. The model is a mathematical description that contains the goal of the problem and all restricted conditions, such as the number of raw materials. These restrictions will be formed as equations or inequalities which control the possible values of the solutions. Afterward, the model will be solved to obtain the solution that corresponds to the desired goal that called the optimal solution. In this study, a linear programming problem [1] is stated as the constraint optimization problem with all linear constraints.

1.2 Why linear programming?

Linear programming [1] is used to obtain optimal solutions for operations research to find the best, most economical solution to a problem within all of its limitations or constraints. Linear programming techniques are used in many fields to make their processes more efficient. These include food and agriculture [7], engineering [2], transportation [9], and energy [8]. For instances, it is used to maximize profitability, to determine the lowest

value for reducing production costs, and to control appropriate activities. These processes have been done with a linear mathematical model which is an advantage of this method because it is easy to manipulate and solve. The use of linear programming for optimization began in the 1940s by a Soviet economist Leonid Kantorovich [15], while looking for a way to assign resources appropriately during World War II. It is a compensation plan to reduce the cost of the Soviet army and increase the damage to the enemy. For a linear programming problem, which began to use in the US military, it was solved by George B. Dantzig [4]. He was the discoverer of the mathematical method for solving a linear programming problem, which was the simplex method. However, the method was kept secret until 1947 since it was being used in the war. The theory of linear programming had been developed for use in applications after the war ended [5]. Linear programming was published in the scientific journal and it became popular among researchers, including numerical analysts, mathematicians, and economists who controlled business benefits. At present, linear programming is widely used in many areas in operations research. There are many practical problems in the operations research which are described by linear programming problems such as network flow problems and product flow problems. Moreover, linear programming techniques are used in many fields to determine the best scenarios such as agricultural economics and management of industrial production [6].

Nowadays, there are many solution methods to search for the optimal solution of a linear programming problem. They rely on concepts such as the graphical method [10], the criss-cross method [13], the interior point method [11]. However, the simplex method is still a powerful method which is the most commonly used for solving a linear programming problem.

1.3 Linear programming model

The linear programming model is an algebraic description of the objective to be maximized/minimized and constraints to be satisfied. Variables are like activities that have to comply with constraints. These variables can be written as x_1 through x_n. Each constraint depends on the number of available units or required units which is represented

by b_i. In a general linear programming problem, the problem can be formulated as

$$\begin{aligned}
&\text{Maximize/Minimize} && z = c_1x_1 + c_2x_2 + ... + c_jx_j + ... + c_nx_n \\
&\text{subject to} && a_{11}x_1 + a_{12}x_2 + ... + a_{1j}x_j + ... + a_{1n}x_n \leq b_1 \\
& && \vdots \\
& && a_{i1}x_1 + a_{i2}x_2 + ... + a_{ij}x_j + ... + a_{in}x_n \leq b_i \\
& && \vdots \\
& && a_{m1}x_1 + a_{m2}x_2 + ... + a_{1j}x_j + ... + a_{1n}x_n \leq b_m \\
& && x_1, x_2, ..., x_n \geq 0
\end{aligned}$$

The problem can be written using summations as follows.

$$\begin{aligned}
&\text{Maximize} && z = \textstyle\sum_{j=1}^{n} c_j x_j \\
&\text{subject to} && \textstyle\sum_{j=1}^{n} a_{ij} x_j \leq b_i, i = 1, 2, ..., m \\
& && x_j \geq 0, j = 1, 2, ..., n,
\end{aligned}$$

where m = number of constraints,

n = number of decision variables,

a_{ij} = coefficient of i^{th} contraint and j^{th} variable,

b_i = right-hand side value of i^{th} constraint,

c_j = cost per unit of j^{th} variable,

x_j = j^{th} decision variable.

However, the constraints of the general linear programming model might contain " = " or " $\geq$ " constraints. Linear programming problems have many sizes from a small problem to a large problem. The small problem which contains less than 5 variables and 5 constraints can be easily solved. For the medium problem and the large problem, they are very difficult to handle without a computer. The linear programming problem can be

formulated using the vector and matrix.

$$\begin{aligned} &\text{Maximize } \mathbf{c^T x} \\ &\text{subject to } \mathbf{Ax} \leq \mathbf{b} \\ &\mathbf{x} \geq \mathbf{0} \end{aligned} \tag{1.1}$$

where $\mathbf{A}$ = the $m \times n$ coefficient matrix.

$\mathbf{b}$ = the $m \times 1$ vector of right-hand sides.

$\mathbf{c}$ = the $n \times 1$ coefficient vector of the objective function.

$\mathbf{x}$ = the $n \times 1$ decision variable vector.

This linear programming model is the canonical form having " $\leq$ " constraints for the maximization problem. For the general linear programming model, it might contain " $\geq$ " or " $=$ " constraints in the model. So to begin solving the linear programming model with the simplex method, slack variables will be added to each of inequalities converting them into equations. After that this model will be called the standard form.

1.4 The standard form of a linear programming problem

Any linear programming model can be converted to the standard form, which is solved by the simplex method. So before discussing the details of the method, there are steps of converting the model to the standard form.

$$\begin{aligned} &\text{Maximize } z = c_1x_1 + c_2x_2 + ... + c_{m+n}x_{m+n} \\ &\text{subject to } a_{11}x_1 + a_{12}x_2 + ... + a_{1(m+n)}x_{m+n} = b_1 \\ &\qquad\qquad a_{21}x_1 + a_{22}x_2 + ... + a_{2(m+n)}x_{m+n} = b_2 \\ &\qquad\qquad\qquad\qquad \vdots \\ &\qquad\qquad a_{m1}x_1 + a_{m2}x_2 + ... + a_{m(m+n)}x_{m+n} = b_m \\ &\qquad\qquad x_1, x_2, ..., x_{m+n} \geq 0 \end{aligned} \tag{1.2}$$

The linear programming model (1.1) is converted to (1.2) and it is called the standard form. The objective function is to maximize or minimize. The constraints are in the form of equations where all variables are nonnegative. The linear programming problem in the standard form always consists of the following:

- If the problem is $\min z$, convert it to $-\max -z$.
- If the constraint is $a_{i1}x_i + a_{i2}x_2 + ... + a_{in}x_n \leq b_i$, convert it into an equality constraint by adding the nonnegative slack variable s_i then the resulting constraint is $a_{i1}x_1 + a_{i2}x_2 + ... + a_{in}x_n + s_i = b_i$ where $s_i \geq 0$.
- If the constraint is $a_{i1}x_i + a_{i2}x_2 + ... + a_{in}x_n \geq b_i$, convert it into an equality constraint by subtracting the nonnegative surplus variable s_i. The resulting constraint is $a_{i1}x_i + a_{i2}x_2 + ... + a_{in}x_n - s_i = b_i$ where $s_i \geq 0$.
- If some variable x_j is unrestricted in sign, replace it everywhere in the formulation by $x_j^+ - x_j^-$ where $x_j^+ \geq 0$ and $x_j^- \geq 0$.

The standard form is

$$\begin{aligned} &\text{Maximize} \sum_{j=1}^{m+n} c_j x_j \\ &\text{subject to} \sum_{j=1}^{m+n} a_{ij} x_j = b_i, i = 1, 2, .., m \\ &\qquad x_j \geq 0, j = 1, 2, ..., m+n \end{aligned} \tag{1.3}$$

1.5 Methods to solve a linear programming problem

The optimal solution of the linear programming problem is searched among all possible solutions subject to constraints and nonnegativity of variables which gives the highest (or the lowest) z value. There are many techniques for solving the linear programming problem that depends on the structure of the problem. The widely used techniques or

methods are the graphical method, the simplex method and the method using artificial variables.

1.5.1 The graphical method

Previously, the linear programming problem is formulated as the linear programming model. It has an objective function which must be maximized or minimized while satisfying several constraints. If there are only two variables in the linear programming problem, it can use the graphical method to find the optimal solution. Consider the set of constraints as the system of inequalities which is shown in Figure 1.1. The solution of this system is the set of points, F, that belongs to all intersections of the constraints. The point in the set F is called the feasible solution and the shaded area (blue area in Figure 1.1) of this solution is called the feasible region. The objective function can be evaluated for the different feasible solution representing by the dashed line (red dash) in Figure 1.1.

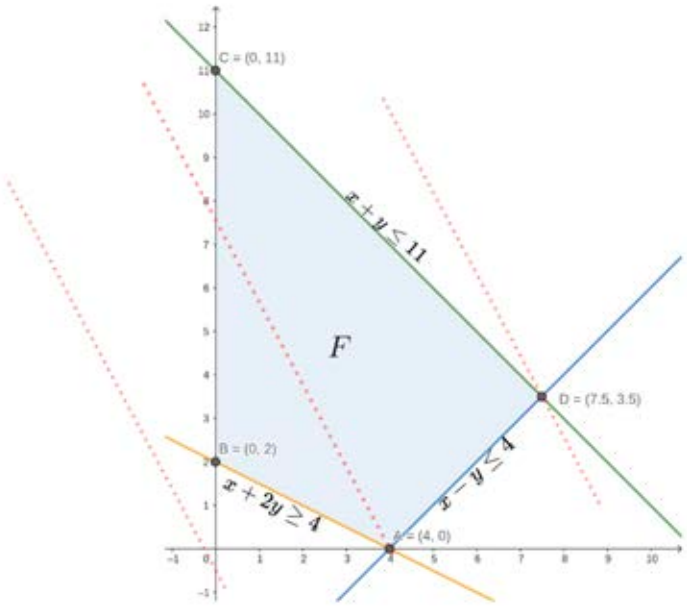

Figure 1.1: Solving the linear programming problem by the graphical method

The system of linear inequality constraints gives the set of the feasible solutions as shown in the Figure 1.1. To solve the linear programming problem, a user normally

determines all corner points of this feasible region and looks for the highest objective value. The highest value of the objective function is the optimal value and the corner point (the vertex) corresponding to the optimal value is called the optimal solution.

1.5.2 The simplex method

A linear programming problem involving two variables can be solved using the graphical method. However, the problem involving more than two variables or involving a large number of constraints is better solved by the solution method that can be implemented and solved using a computer. One such method is called the simplex method. It provides a systematic way of examining the vertices around the feasible region with a repetition starting from the initial vertex and systematically moves to the adjacent vertices which result in a better objective function value. The simplex method starts from the initial vertex in the feasible region. Usually, the simplex method starts at the origin point if it belongs to the feasible region. Otherwise, the simplex method needs other processes to find a new initial feasible vertex. Most of those rely on adding artificial variables to the linear programming model. The popular method for searching a starting vertex of the simplex method is called the two-phase simplex method.

1.5.3 The two-phase simplex method

This method was also introduced by Dantzig in 1963 [12] after he found that the simplex method could not start with the infeasible vertex. So this method is created for finding the new feasible starting vertex to the simplex method. The details of this method will be covered in the section of the two-phase simplex method of the next chapter. The general concept of this method is to split the process into two phases. The first phase aims to search for the initial feasible vertex by adding artificial variables into the model. These artificial variables allow the simplex method to start at the origin point. All artificial variables will leave out in the last simplex iteration of the phase I. At the end of the phase I, the method will find the feasible vertex of the original linear programming model. The second phase is the tradition simplex method that finds the optimal solution as mentioned in the previous subsection. In this thesis, the preceding-jump simplex

method investigates the phase I to find the feasible starting jump point when the origin point does not belong to the feasible region. Next section, the basic concept of the jump method will be explained.

1.6 The jump method

In the preceding-jump simplex method, the major key concept is to find the suitable initial feasible vertex before the jump. The goal of the jump is to avoid unnecessary vertices that the simplex method will pass through. This implies that the method will reduce the number of iterations and the total running time for solving the linear programming model. The jump point always resides in the feasible region if the starting jump point belongs to the feasible region and the jump direction points into the feasible region. The first jump, the method takes the gradient of the objective function to be the first direction (the first vector) of the jump because it gives the largest increasing objective function value comparing to other directions in term of unit vectors. Suppose that the linear programming model has the bounded feasible region F in $\mathbb{R}^n$. Then F contains all of the possible solutions. The jump method will take the direction to shift the point in the feasible region to another point. Assume that F is the feasible region as shown in Figure 1.2 and the direction is $[1.5, 1]^T$. The jump process will calculate the shortest distance from the starting point such as the origin point $(0, 0)$ to a point on every hyperplane along this direction. These hyperplanes are represented by the boundary on F that will block the jump before breaking out of the feasible region. So the jump point will belong to the hyperplane that it satisfies the shortest distance between the origin point and the point on these hyperplanes in Figure 1.2. The shortest distance is obtained from the minimum of the right-side values divided by the multiplying of the direction and the gradient vector of every constraint i.e. $\alpha_i = \frac{b_i}{\mathbf{A}_i \cdot \mathbf{c}}$ where i is the order of constraints in the linear programming model. In Figure 1.2, the minimum distance satisfies the plane

$2x_1 + x_2 = 10$ and it is the third constraint. Then α_3 is equal to

$$\alpha_3 = \frac{10}{\begin{bmatrix} 2 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} 1.5 \\ 1 \end{bmatrix}} = 2.5.$$

Thus, the jump point is $\mathbf{x}_1^1 = \alpha_3 \mathbf{c} = 2.5(1.5, 1) = (3.75, 2.5)$ because it corresponds to the minimum distance of those constraints and it also belongs to the feasible region.

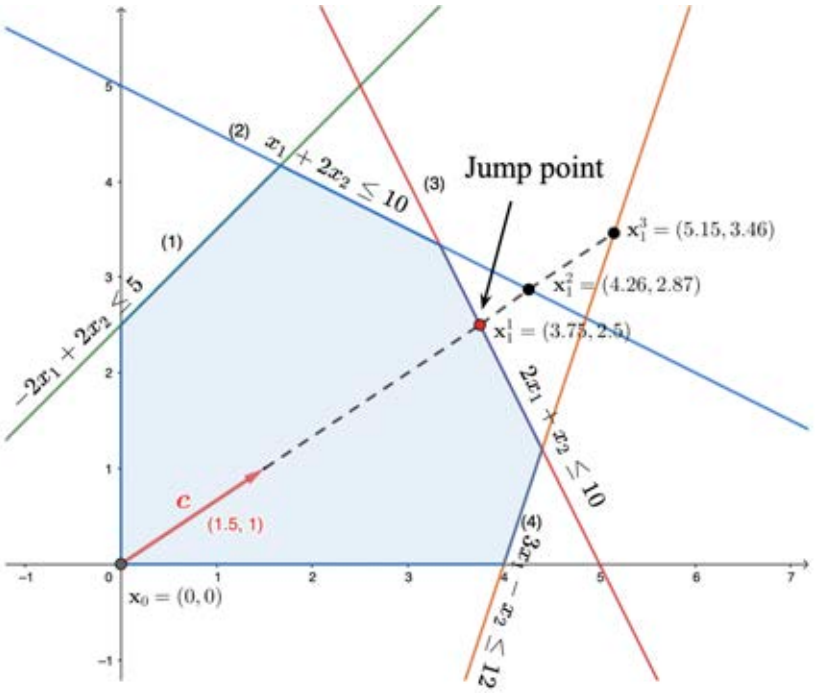

Figure 1.2: The initial jump point

Notice that the point along the direction of $\mathbf{c}$ can be shifted to $\alpha_{i_{min}} \cdot \mathbf{c}$, where i_{min} is the minimum index of constraints when it hits the edge of the hyperplane having the minimum distance between $\mathbf{x}_0$ and $\mathbf{x}_1^i, i = 1, 2, ..., m$.

1.6.1 Objective of the thesis

The purpose of this thesis is to present a new approach to find a basic feasible solution close to the optimal solution by jumps. Firstly, this thesis shows how to apply jumps in the scenario that the origin point is the initial vertex. Then it adapted to

apply to the general linear programming problems such the scenario of the origin point is not a feasible starting point of jumps. Performance evaluation of this new improved simplex method will be extended on the synthetic linear programming problems. Finally, the major objective of this method is to reduce the number of iterations and the total running time of the simplex method. The computational experiments will be presented by the synthetic LP problems from 100, 200, 300, 400 500 to 2500 variables. Moreover, the preceding-jump simplex method will be extended to solve the linear programming problem in the general form. It is also implemented with the synthetic linear programming problems from 100 to 1000 variables.

1.6.2 Scope of the thesis

Chapter 1: Introduction, this chapter serves as an introduction to linear programming which relates to the optimization method, an introduction to the solution methods for linear programming.

Chapter 2: Background and Literature Review provides the essential definitions and theories of linear programming that are used to produce the main results in chapter 3. It also explains the steps of the solution method such as the simplex method and the two-phase simplex method. The related work will be reviewed at the end of this chapter.

Chapter 3: Main results discuss the research methodology of this thesis which is split into two major topics. The first topic will introduce the preceding-jump simplex method with the origin vertex as the initial basic feasible solution. It also focuses on the implementation of the preceding-jump simplex method. The second topic explains the process of applying the preceding-jump simplex method in a general form of the linear programming problem.

Chapter 4: Computational results, the randomly generated linear programming problems were constructed and presented the results in this chapter. The results showed the compared number of iterations and running time of the traditional simplex method and the new method.

Chapter 5: Conclusion will be summarized the findings, states the managerial implications and highlights the limitations of this new method.

CHAPTER II

BACKGROUND KNOWLEDGE AND LITERATURE REVIEW

The main concept of this chapter involves definitions and theories which are essential to the preceding-jump simplex method. The first two sections start with the notations and basic algebra. Next section, the jump idea is proposed using the concept of geometric definitions of the linear programming (LP). In the fourth section to the sixth section, the important details of the LP, that are the system of linear equations, the polyhedron theory, and the basic feasible solution, are given for the simplex method. The next two sections, the important details of the simplex method and the two-phase simplex method will be presented. Finally, the literature reviews involving this thesis are shown in the last section.

2.1 Notations

n	=	the number of decision variables.
m	=	the number of constraints.
$\mathbf{c}$	=	the gradient vector of the objective function.
I	=	the set of all indices of constraints i.e. $\{1, 2, ..., m+n\}$.
$\mathbf{A}_i$	=	the i^{th} row of the matrix $\mathbf{A}$.
b_i	=	the value of the right-hand side of the i^{th} row, $i = 1, 2, ..., m+n$.
B	=	the index set of the basic variables.
N	=	the index set of the non-basic variables.
V	=	the set of all indices visited constraints.
$\mathbf{A}_B$	=	the $m \times m$ invertible matrix, called the basic matrix.
$\mathbf{A}_V$	=	the system of linear equations from the index set V.
$\mathbf{b}_V$	=	the vector of the right-hand side forming from V.

$\mathbf{x}_B$ = the basic variables.

$\mathbf{x}_N$ = the non-basic variables.

$\mathbf{x}_0$ = the feasible starting point of the jump processes.

$\mathbf{x}_k$ = the feasible jump point corresponding to the k^{th} iteration.

$\mathbf{d}_k$ = the jump direction in the k^{th} iteration.

α_k = the minimum distance between $\mathbf{x}_k$ and the hyperplanes in $I \setminus V$.

2.2 Basic algebra

The mathematical concept in this thesis depends on basic algebra, especially the concept of the linear system. It covers the detail of the vector, matrix, and rank of the matrix, including an angle between vectors.

2.2.1 Vector

An n-vector is a column of n numbers where the transpose of it, is called the row vector. For example, $\mathbf{v}^T = [1, 0, 0]$ is the row vector of size $n = 3$, and $\mathbf{u}^T = [1, 0]$ is the row vector of size 2. Usually, an n-vector is the column vector having the dimension $n \times 1$. The vector contains only zeroes in all components is called the null vector or the zero vector represented by $\mathbf{0}$. The vector consisting of 1 at the i^{th} row and the zeroes of the remaining rows is called the standard unit vector which is represented by $\mathbf{e}_i$.

$$\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \qquad \begin{bmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \\ 0 \end{bmatrix}$$

Zero vector The standard unit vector $\mathbf{e}_i$

2.2.2 Matrix and rank of a matrix

A matrix (m, n) is a collection of row vectors or column vectors which contains m rows and n columns. It is customary to enclose the elements of a matrix in parentheses, brackets, or braces. An identity matrix is a square matrix which has only 1 on the diagonal. The identity matrix is denoted as $\mathbf{I}_n$.

$$\mathbf{I}_n = \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{bmatrix}$$

The column rank of the matrix is the dimension of its column space that is the number of linearly independent columns of the matrix. Likewise, the row rank of the matrix is the number of linearly independent rows of the matrix. The row rank is always equal to the column rank, thus one normally uses the term rank of the matrix to identify the number of linearly independent rows or columns of the matrix.

2.2.3 Inner product

Any two n-vectors $\mathbf{u}$ and $\mathbf{v}$ can be multiplied together. The result of this multiplication is a real number which is called an inner or dot product of the two vectors. It is defined as follows.

$$\mathbf{u} \cdot \mathbf{v} = u_1 v_1 + u_2 v_2 + \ldots + u_n v_n = \sum_{i=1}^{n} u_i v_i \tag{2.1}$$

2.2.4 Euclidean norm

The measuring vector size or the length of the vector is done by specifying a norm. Generally, the vector size in Euclidean space is determined from the square root of the inner product of the vectors. The norm of the vector is the function that assigns the

positive length in the real number to the vector in the vector space excepting the zero vector. The length of the vector $\mathbf{u}$ is calculated from

$$\|\mathbf{u}\| = \sqrt{u_1^2 + u_2^2 + \cdots + u_n^2}.$$

2.2.5 Angle between vectors

Let $\mathbf{u} = (u_1, u_2, ..., u_n)$ and $\mathbf{v} = (v_1, v_2, ..., v_n)$ be any two vectors in $\mathbb{R}^n$. The inner product or the dot product of the two vectors is $\mathbf{u} \cdot \mathbf{v} = \|\mathbf{u}\|\|\mathbf{v}\| \cos\theta$ where θ is the angle between the two vectors. So θ is calculated from the following formula.

$$\theta = \arccos\left(\frac{\mathbf{u} \cdot \mathbf{v}}{\|\mathbf{u}\|\|\mathbf{v}\|}\right) \tag{2.2}$$

The equality above guarantees that the unique value of $\theta \in [0, \pi]$ that satisfies $\mathbf{u} \cdot \mathbf{v} = \|\mathbf{u}\|\|\mathbf{v}\| \cos\theta$. From the relation of two vectors in (2.2), θ will be occured in three cases.

1. If $\mathbf{u} \cdot \mathbf{v} = 0$, then both vectors are orthogonal to each other.
2. If $\mathbf{u} \cdot \mathbf{v} > 0$, then both vectors make an acute angle.
3. If $\mathbf{u} \cdot \mathbf{v} < 0$, then both vectors make an obtuse angle.

In the jump-to-vertex phase of chapter III, there is a step of choosing a jump direction before searching the jump point. In this step, the preceding-jump simplex method has to calculate an angle between the given direction (a vector) and the gradient vector of the objective function. The angle of both vectors will be used to check for the improving direction before doing the jump process.

2.3 Geometric definitions

The following definitions [1] relate to the LP problem which the jump idea relies on. There are three geometric definitions that support the searching of the jump point.

Definition 2.3.1. A ray is a collection of points of the form $\{\mathbf{x_0} + \lambda\mathbf{d} : \lambda > 0\}$, where $\mathbf{d}$ is the nonzero vector. Here, $\mathbf{x}_0$ is called the vertex of the ray, and $\mathbf{d}$ is the direction of the ray.

Definition 2.3.2. Given an unbounded convex set X, a non zero vector $\mathbf{d}$ is called the direction of X if for each $\mathbf{x}_0 \in X$, the ray $\{\mathbf{x} : \mathbf{x}_0 + \lambda\mathbf{d}, \mathbf{d} \neq \mathbf{0}, \lambda \geq 0\}$ is contained within X. Note that $\mathbf{d}_1$ and $\mathbf{d}_2$ are distinct if and only if there is no k that $\mathbf{d}_1 \neq k\mathbf{d}_2$.

Definition 2.3.3. An extreme direction is a direction that it cannot be expressed as the positive combination of two distinct directions. Any other direction can be expressed as a positive combination of extreme directions.

2.4 The system of linear equations

Mathematical problems are presented in the form of linear equations. The linear equations can be expressed in the matrix form such as

$$\mathbf{Ax} = \mathbf{b} \tag{2.3}$$

where $\mathbf{A}$ is the matrix of the coefficient variables of the constraints. The solution of this equation might occur in 3 cases: the solution is unique or the system has no solution or the system has infinitely many solutions.

In the first case, $\mathbf{A}$ needs to be the full row rank and the square matrix. Thus, $\mathbf{A^{-1}}$, the inverse matrix of $\mathbf{A}$ exists and the unique solution is $\mathbf{x} = \mathbf{A^{-1}b}$. Note that, the inverse of $\mathbf{A}$ is only $\mathbf{A^{-1}}$ when $\mathbf{A^{-1}} \times \mathbf{A} = \mathbf{A} \times \mathbf{A^{-1}} = \mathbf{I}$. The next case, if some inconsistencies arise between equations in $\mathbf{Ax} = \mathbf{b}$, the system cannot be solved. This implies that this system has no solution. The last case, if the number of variables is greater than the number of equations and there is no inconsistency system then the system has more than one solution. However, this system is still able to solve using the right or left inverse of $\mathbf{A}$.

2.4.1 Generalized inverse

The general linear system may not be square or it may be a square matrix which is not full rank matrix. This is a reason for the linear system that cannot be solved with the regular inverse of $\mathbf{A}$. If $\mathbf{A}$ has more rows than columns ($m > n$) which implies more equations than variables, and the system is sometimes referred to an overdetermined system. In the opposite argument $\mathbf{A}$ has more variables than equations ($m < n$). However, the solution of $\mathbf{Ax} = \mathbf{b}$ can be solved using the left or right inverse of the matrix. The left or the right inverse is the types of the generalized inverse of $\mathbf{A}$.

Given matrix $\mathbf{A} \in \mathbb{R}^{n\times m}$ and matrix $\mathbf{A}^{\mathrm{g}} \in \mathbb{R}^{m\times n}$, $\mathbf{A}^{\mathrm{g}}$ is the generalized inverse of $\mathbf{A}$ if it satisfies the condition $\mathbf{A}\mathbf{A}^{\mathrm{g}}\mathbf{A} = \mathbf{A}$. Thus the solution of the linear system with $\mathbf{A}$ non-square full rank can be found using the notion of the left or the right inverse of the matrix. Let consider the linear system $\mathbf{Ax} = \mathbf{b}$ where $\mathbf{A}$ is non-square full rank so the solution may be occured if $\mathbf{x} = \mathbf{A}^{\mathrm{g}}\mathbf{b}$. Thus the general inverse of $\mathbf{A}$ will be separated into two types.

Left inverse : If $\mathbf{A}$ has linearly independent columns i.e. rank $\mathbf{A} = n$. Then the matrix $\mathbf{A}^T\mathbf{A}$ is the invertible n by n symmetric matrix, so $(\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{A} = \mathbf{I}$. This implies that the left inverse of $\mathbf{A}$ is $(\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T$. Thus $\mathbf{A}^{\mathrm{g}} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T$ and $\mathbf{x} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{b}$.
Right inverse :If $\mathbf{A}$ has linearly independent rows i.e. rank $\mathbf{A} = m$. Then the matrix $\mathbf{A}\mathbf{A}^T$ is the invertible m by m symmetric matrix. This implies that the right inverse of $\mathbf{A}$ is $\mathbf{A}^T(\mathbf{A}\mathbf{A}^T)^{-1}$. Thus $\mathbf{A}^{\mathrm{g}} = \mathbf{A}^T(\mathbf{A}\mathbf{A}^T)^{-1}$ and $\mathbf{x} = \mathbf{A}^T(\mathbf{A}\mathbf{A}^T)^{-1}\mathbf{b}$.

2.5 The polyhedron theory

Basic importance theories of the LP problem [1] based on some properties of the polyhedron which are the particular significance in proving theorems of LP.

Definition 2.5.1. A point $\mathbf{x}$ of a convex set M is an extreme point or the vertex of M then it is not possible to find two points $\mathbf{x_1}, \mathbf{x_2}$ in M such that $\mathbf{x} = \lambda\mathbf{x}_1 + (1-\lambda)\mathbf{x}_2, 0 < \lambda < 1$. It means that the vertex does not lie on the line segment joining two other points in the

set.

Definition 2.5.2. The set of all convex combinations of a finite number of the extreme point $x_i, i \in \{1, 2, ..., k\}$, is the convex polyhedron spanned by these points. Note the set of vertices of the convex polyhedron is the subset of its spanning points.

Definition 2.5.3. A hyperplane H in $\mathbb{R}^n$ is a set of the form $\{\mathbf{x}|\mathbf{p}^T\mathbf{x} = k\}$ where $\mathbf{p}$ is the nonzero vector in $\mathbb{R}^n$ and k is the scalar. Note $\mathbf{p}$ is called the normal or the gradient vector of the hyperplane. A closed half-space is a collection of points of the form $\{\mathbf{x}|\mathbf{p}^T\mathbf{x} \leq k\}$, where $\mathbf{p}$ is the nonzero vector in $\mathbb{R}^n$ and k is the scalar i.e the hyperplane divides $\mathbb{R}^n$ into two regions, called half-spaces. The intersection of the finite number of closed half-spaces is called the polytope. The hyperplanes producing the half-spaces are called the generating hyperplanes of the polytope.

A constraint is declared to be "binding" or a constraint with the "slack" value of zero is said to be tight or binding if it satisfies with equality. Constraints which are not tight are called loose or not binding. Let $\mathbf{x} \in X$ and suppose that the constraint j in the set of all constraints $\{1, 2, ..., m + n\}$ is binding, or tight, or active, at $\mathbf{x}$ then $\mathbf{A}_j\mathbf{x} = b_j$. Moreover, the solution in $P = \{\mathbf{x} : \mathbf{Ax} \leq \mathbf{b}\}$ is called degenerate if it has more than n linearly independent active constraints.

2.6 The basic feasible solution

The notion of the basic feasible solution plays an especially important role in the theory of linear programming. If point $\mathbf{x}$ in convex set $X \subseteq \mathbb{R}^n$ is called the basic feasible solution of X if $\mathbf{x}$ cannot be represented as the strict convex combination of two distinct points in X. In other words, if $\mathbf{x} = \lambda\mathbf{x}_1 + (1 - \lambda)\mathbf{x}_2$ with $\lambda \in (0, 1)$ and $\mathbf{x}_1, \mathbf{x}_2 \in X$, then $\mathbf{x} = \mathbf{x}_1 = \mathbf{x}_2$. The solution set in the LP problem is equivalent to the intersection of every constraint in that problem. Thus the general basic feasible solution of $X \subseteq \mathbb{R}^n$ will be created by solving the linear system as below.

$$\begin{aligned} \mathbf{Ax} &= \mathbf{b} \\ \mathbf{x} &\geq \mathbf{0} \end{aligned} \tag{2.4}$$

where $\mathbf{A}$ is the $m \times n$ matrix and $\mathbf{b}$ is the m-vector. Suppose that the rank $(\mathbf{A}) = m$ and $\mathbf{b}$ belongs to the range that is spanned by the column space of $\mathbf{A}$. A solution of (2.4) is called the feasible solution. From chapter I, the set of the feasible solution or the feasible region is denoted by F. In some cases of the LP problems, it is possible that there is no feasible solution. In general, if the set F is not empty, it must be the closed convex set (polytope) bounded from below so it has at least one vertex.

From equation (2.4) with m equations and n variables, assume that $m < n$ and the equation are linearly independent. Generally, constraints appear as inequalities in mathematical models. By subtracting or adding variables to the model becomes the linear system $\mathbf{Ax} = \mathbf{b}$. If all of $n - m$ variables are given the zero values, then the remaining system of m equations in m variables has the unique solution. This solution along with the assumed zeros is the solution of equation (2.4). It is called the basic solution. The m variables remaining in the system after $n - m$ variables have been put equal to zero are called the basic feasible solution. The rest of the variables are called non-basic.

$$\begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1n} \\ A_{21} & A_{22} & \cdots & A_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ A_{m1} & A_{m2} & \cdots & A_{mn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}$$

or

$$\mathbf{A}_{\cdot 1}x_1 + \mathbf{A}_{\cdot 2}x_2 + \mathbf{A}_{\cdot 3}x_3 + ... + \mathbf{A}_{\cdot n}x_n = \mathbf{b}$$

where $\mathbf{A}_{\cdot j}, j = 1, 2, .., n$ is the j^{th} column of $\mathbf{A}$. This implies that $\mathbf{A}_{\cdot j}$ is in $\mathbb{R}^m$ with the full rank $\mathbf{A}$ so after rearranging the columns of $\mathbf{A}$ the column vectors $\mathbf{A}_{\cdot 1}, \mathbf{A}_{\cdot 2}, ..., \mathbf{A}_{\cdot m}$ are linearly independent. Therefore, the set of $\mathbf{A}_{\cdot 1}, \mathbf{A}_{\cdot 2}, ..., \mathbf{A}_{\cdot m}$ forms a basis. Moreover the solution of the equation above will be the n-vector $[x_1, x_2, ..., x_m, 0, 0, ..., 0]^T$ that corresponds to $\mathbf{Ax} = \mathbf{b}$. The m linearly independent column vectors $\mathbf{A}_{\cdot j}, j \in \{1, 2, ..., n\}$

will become the $m \times m$ invertible matrix $\mathbf{A}_B$ and the remaining column vectors of $\mathbf{A}$ denotes as $m \times (n-m)$ matrix $\mathbf{A}_N$. Then the matrix $\mathbf{A}$ will be expressed as $\mathbf{A} = [\mathbf{A}_B \ \mathbf{A}_N]$. The solution $\mathbf{x}$ to $\mathbf{Ax} = \mathbf{b}$, where $\mathbf{x}_B = \mathbf{A}_B^{-1}\mathbf{b}$ and $\mathbf{x}_N = \mathbf{0}$ is called the basic solution of the system. If $\mathbf{x}_B \geq \mathbf{0}$, then $\mathbf{x} = [\mathbf{x_B} \ \mathbf{x_N}]$ is called the basic feasible solution of the system. Here $\mathbf{A}_B$ is called the basic matrix (or simply the basis) and $\mathbf{A}_N$ is called the non-basic matrix. The components of $\mathbf{x}_B$ are called basic variables (or dependent variables) and the components of $\mathbf{x}_N$ are called non-basic variables (or independent variables). Moreover, if all components of $\mathbf{x}_B > 0$, then $\mathbf{x}$ is called the nondegenerate basic feasible solution, but if at least one component of $\mathbf{x}_B$ is zero, then $\mathbf{x}$ is called the degenerate basic feasible solution. The example 1 shows the problem with two variables.

$$\begin{aligned} \text{Maximize} \quad & 2x_1 + x_2 \\ \text{subject to} \quad & x_1 + x_2 \leq 4 \\ & -x_1 + x_2 \leq 4 \\ & x_1 - x_2 \leq 3 \\ & x_1, x_2 \geq 0 \end{aligned}$$

The LP problem contains three constraints with the feasible origin vertex. In the standard form, the problem needs the slack variables in order to transform these constraints to be the equations. This implies that the variables will be separated into two types as non-basic variables of $\mathbf{x}_N$ and basic variables of $\mathbf{x}_B$. The vertex C can be computed by defining the matrix $\mathbf{A}_B$ with basic variables x_1, x_2 and x_4. Now, this particular basis matrix $\mathbf{A}_B$ is defined as follows:

$$\mathbf{A}_B = \begin{bmatrix} 1 & 1 & 0 \\ -1 & 1 & 1 \\ 1 & -1 & 0 \end{bmatrix}$$

Then

$$\mathbf{x}_B = \mathbf{A}_B^{-1}\mathbf{b} = \begin{bmatrix} 0.5 & 1 & 0.5 \\ 0.5 & 0 & -0.5 \\ 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} 4 \\ 4 \\ 3 \end{bmatrix} = \begin{bmatrix} 3.5 \\ 0.5 \\ 7 \end{bmatrix}$$

Hence $x_1 = 3.5, x_2 = 0.5$ and $x_4 = 7$ is the basic feasible solution which it shows as the vertex C in Figure 2.1.

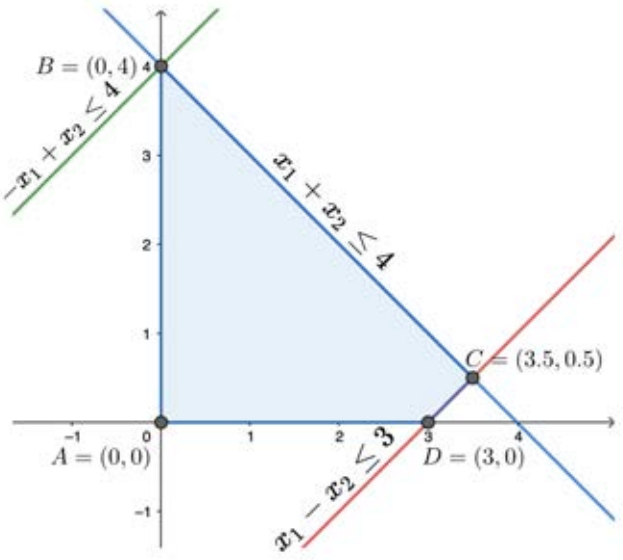

Figure 2.1: Basic feasible solution of the example 1

2.7 The simplex algorithm

From the LP model 1.1, the model contains the $m \times n$ matrix $\mathbf{A}$, the $n \times 1$ vector $\mathbf{x} \geq \mathbf{0}$, the $m \times 1$ vector $\mathbf{b} \geq \mathbf{0}$, and the $n \times 1$ vector $\mathbf{c}$.

The model can be converted into the standard form as belows.

$$\begin{aligned} &\text{Maximize } \mathbf{c}^T\mathbf{x} \\ &\text{subject to } \mathbf{A}\mathbf{x} + \mathbf{I}\mathbf{s} = \mathbf{b} \\ &\mathbf{x}, \mathbf{s} \geq \mathbf{0} \end{aligned} \tag{2.5}$$

where $\mathbf{x} = [x_1, x_2, ..., x_n]^T$ is the vector of the original variables of the LP model and $\mathbf{s} = [s_1, s_2, ..., s_m]^T$ is the vector of the slack variables. Moreover, the variables $x_j \geq 0, j = 1, 2, ..., n$ and $s_i \geq 0, i = 1, 2, ..., m$. The vector $\mathbf{c} = [c_1, c_2, ..., c_n]^T$ is the vector of the original coefficient variables of the objective function. The matrix $\mathbf{I}$ is the $m \times m$ identity matrix and $\mathbf{b} = [b_1, b_2, ..., b_m]^T$ is the right-hand-side vector of the constants with $b_i \geq 0, i = 1, 2, ..., m$. Suppose that the $m \times (n + m)$ matrix $\bar{\mathbf{A}}$ is denoted by $[\mathbf{A} \,|\, \mathbf{I}]$ which can be used to create the first simplex tableau. The slack variables are the basic variables of the first simplex tableau. Since the column vectors of the matrix $\mathbf{I}$ are linearly independent and they also span the column vectors of the model. Hence, the matrix $\mathbf{I}$ is represented by $\bar{\mathbf{A}}_B$ which is the first basic matrix of the first simplex tableau and $\mathbf{A}$ is the first non-basic matrix representing by $\bar{\mathbf{A}}_N$. The simplex method finds the optimal solution according to the following steps.

Step 1: The first step is to examine the negative reduced cost value of the non-basic variables.

$$\mathbf{z}_N^T = \mathbf{c}_B^T \bar{\mathbf{A}}_B^{-1} \bar{\mathbf{A}}_N - \mathbf{c}_N^T.$$

If all the negative reduced costs are negative value then return the current basic feasible solution is optimal.

Step 2: If not, pick out the variable x_k that corresponds to the largest (or most positive) value from $\mathbf{z}_N^T$. The variable x_k is called the entering variable.

Step 3: Let the entering cloumn vector be $\mathbf{y}_k = \bar{\mathbf{A}}_B^{-1} \bar{\mathbf{A}}_{N_k}$. If $\mathbf{y}_k \leq 0$ then return the LP problem has the unbounded solution. Let the right-hand-side vector be $\mathbf{r} = \bar{\mathbf{A}}_B^{-1}\mathbf{b}$.

Step 4: Calculate the ratio test of the basic variables from the ratio $\frac{r_i}{y_{ik}}$ where $y_{ik} > 0$ and

$i \in B$. The index of the basic variable corresponding to the minimum ratio will be denoted by l. The basic variable x_l which corresponds to the minimum ratio test, is called the leaving variable.

Step 5: The leaving variable x_l will be swapped with the entering variable x_k. Then B and N will be updated. Go to step 1.

2.7.1 Pseudo code of the simplex algorithm

Algorithm 1 The simplex algorithm

procedure SIMPLEX($\mathbf{c}, \mathbf{b}, \mathbf{A}$)
  Input: the LP problem $\mathbf{c}, \mathbf{b}, \mathbf{A}$
  Output: the optimal solution or unbounded solution
  while True **do**
    $\mathbf{z}_N^T \leftarrow \mathbf{c}_B^T \bar{\mathbf{A}}_B^{-1} \bar{\mathbf{A}}_N - \mathbf{c}_N^T$
    If $\mathbf{z}_N^T \leq 0$ **then** STOP **return** the current BFS is optimal
    $k \leftarrow \text{argmax}\{\mathbf{z}_N^T\}$
    $\mathbf{y}_k \leftarrow \bar{\mathbf{A}}_B^{-1} \bar{\mathbf{A}}_{N_k}$
    If $\mathbf{y}_k \leq 0$ **then** STOP **return** unbounded solution
    $\mathbf{r} \leftarrow \bar{\mathbf{A}}_B^{-1} \mathbf{b}$
    $l \leftarrow \text{argmin}\{\frac{r_i}{y_{ik}} | y_{ik} > 0, i \in B\}$
    $B \leftarrow (B \setminus \{l\}) \cup \{k\}$
    $N \leftarrow (N \setminus \{k\}) \cup \{l\}$
  end

2.8 The two-phase simplex algorithm

The LP model (1.1) with the slack variables transforms itself to the standard form before performing the simplex method. The simplex method can immediately start because the method has the basic feasible solution at the origin point. For the general LP

model, the simplex method cannot always start at the origin point due to its infeasibility. To remedy this issue, artificial variables are used to transform the original LP problem to the one with the basic feasible solution. Hence, the artificial variables will be added into the standard form for constructing the first basic feasible solution. After that, the simplex method can solve this model and the artificial variables will be driven out of the basic variables. This implies that the current basic feasible solution will be the basic feasible solution of the original LP problem. This thesis will present the artificial variable method for solving the general LP model called the two-phase simplex method.

Consider the following model

$$\begin{aligned} \text{Maximize } & \sum_{j=1}^{n} c_j x_{ij} \\ \text{subject to } & \sum_{j=1}^{n} a_{ij} x_{ij} \geq b_i, i = 1, 2, .., m \\ & x_{ij} \geq 0, j = 1, 2, ..., n \end{aligned} \tag{2.6}$$

Sometimes, the first basic feasible solution of this model is hard to derive since the origin point might not belong to the feasible region. Before performing the simplex method, it has to find the first basic feasible solution. In 1963, Danzig invented a new method, called the two-phase simplex method [12], which the first phase will give a basic feasible solution to the simplex method. It begins with assigning the surplus variables and the artificial variables to the constraints (“ $\geq$ ” or “ $=$ ”) and the objective function is changed to minimize the sum of these artificial variables. They allow the origin point with artificial variables assign to $\mathbf{b}$. This implies that the simplex method can start to solve the LP model at the origin point. However, the artificial variables will be driven out of basic in the final iteration of the first phase. From (2.6), the model will be transformed into (2.7)

by adding the slacks variables, the surpluses variables, and the artificial variables.

$$\begin{aligned} &\text{Minimize} \sum_{k=1}^{m} x_{a_i} \\ &\text{subject to} \sum_{j=1}^{n} a_{ij}x_j - x_{n+i} + x_{a_i} = b_i,\ i = 1, 2, .., m, \\ &\qquad x_j, x_{a_i} \geq 0,\ j = 1, 2, ..., n+m, i = 1, 2, ..., m \end{aligned} \tag{2.7}$$

where x_{a_i} is an artificial variable in the LP model and m is the number of artificial variables. The phase I begins with setting the coefficients of non-artificial variables in the objective function to zero i.e. $c_j = 0$ for $j = 1, 2, ..., n+m$. But the coefficient variable of the artificial variables will be assigned to 1. At the end of the phase I, the artificial variables will become the non-basic variables. Consider the following LP problem.

$$\begin{array}{lrcl} \text{Maximize} & 2x_1 + x_2 & & \\ \text{subject to} & x_1 + x_2 & \leq & 4 \\ & -x_1 + x_2 & \leq & 4 \\ & x_1 - x_2 & \leq & 3 \\ & x_1 + x_2 & \geq & 1 \\ & x_1, x_2 & \geq & 0 \end{array}$$

Phase I : It can be transformed into the standard form by introducing 3 slack variables, 1 surplus variables and 1 artificial variable i.e. x_3, x_4, x_5, x_6 and x_a.

$$\begin{array}{lrcl} \text{Minimize} & x_a & & \\ \text{subject to} & x_1 + x_2 + x_3 & = & 4 \\ & -x_1 + x_2 + x_4 & = & 4 \\ & x_1 - x_2 + x_5 & = & 3 \\ & x_1 + x_2 - x_6 + x_a & = & 1 \\ & x_1, x_2, x_3, x_4, x_5, x_6, x_a & \geq & 0 \end{array}$$

There is no obvious basic feasible solution in the original LP model. So the phase I of

the two-phase method will be used to determine the basic feasible solution of the original model. The objective function of the following LP model is changed by expressing the artificial variable x_a in term of the non-artificial variables i.e. $x_a = 1 - x_1 - x_2 + x_6$. After that, the simplex method will search for the optimal solution. Until x_a becomes the non-basic variable, then x_a will be dropped out of the model. The optimal solution of the phase I is the basic feasible solution of the original model. Note that, if x_a in the last simplex tableau is still greater than 0, then this model will be infeasible.

$$
\begin{array}{lccccccccccccccccc}
\text{Minimize} & 1 & - & x_1 & - & x_2 & + & x_6 & & & & & & & & & & \\
\text{subject to} & & & x_1 & + & x_2 & + & x_3 & & & & & & & & & = & 4 \\
& - & & x_1 & + & x_2 & & & + & x_4 & & & & & & & = & 4 \\
& & & x_1 & - & x_2 & & & & & + & x_5 & & & & & = & 3 \\
& & & x_1 & + & x_2 & & & & & & & - & x_6 & + & x_a & = & 1 \\
& & & & & & & & x_1 & , x_2 & , x_3 & , x_4 & , x_5 & , x_6 & , x_a & \geq & 0
\end{array}
$$

Iter. 0	x_1	x_2	x_3	x_4	x_5	x_6	x_a	rsh
z	1	1	0	0	0	-1	0	1
x_3	1	1	1	0	0	0	0	4
x_4	-1	1	0	1	0	0	0	4
x_5	1	-1	0	0	1	0	0	3
x_a	1	1	0	0	0	-1	1	1
$z_j - c_j$	-2	-1	0	0	0	0	0	0

Iter. 1	x_1	x_2	x_3	x_4	x_5	x_6	x_a	rsh
z	0	0	0	0	0	0	-1	0
x_3	0	0	1	0	0	1	-1	3
x_4	0	2	0	1	0	-1	1	5
x_5	0	-2	0	0	1	1	-1	2
x_1	1	1	0	0	0	-1	1	1
$z_j - c_j$	0	1	0	0	0	-2	2	2

Phase II : Now, the last simplex tableau of the phase I gives the basic feasible solution of the original LP model i.e. $(x_1, x_2, x_3, x_4, x_5) = (1, 0, 3, 5, 2, 0)$. Hence the next phase of the two-phase simplex method will search for the optimal solution.

		x_1	x_2	x_3	x_4	x_5	x_6	rsh
Iter.0	z	0	1	0	0	0	-2	2
	x_3	0	0	1	0	0	1	3
	x_4	0	2	0	1	0	-1	5
	x_5	0	-2	0	0	1	1	2
	x_1	1	1	0	0	0	-1	1
Iter. 1	z	-1	0	0	0	0	-1	1
	x_3	0	0	1	0	0	1	3
	x_4	-2	0	0	1	0	1	3
	x_5	2	0	0	0	1	-1	4
	x_1	1	1	0	0	0	-1	1

The optimal solution of the example above is $(x_1, x_2, x_3, x_4, x_5, x_6) = (1, 0, 3, 3, 4, 0)$ and the optimal value is 1. The last subsection of this chapter will show the pseudo code of the two-phase simplex method.

2.8.1 Pseudo code of the two-phase simplex algorithm

Algorithm 2 The two-phase simplex algorithm

Input: an LP problem $\mathbf{c}, \mathbf{b}, \mathbf{A}$.

Output: the optimal solution, unbounded solution, infeasible solution

#Phase I

$\mathbf{c}_l \leftarrow 1$ **if** i is the index of the artificial variable **otherwise** 0

while True **do**

    $\mathbf{z}_N^T \leftarrow \mathbf{c}_B^T \mathbf{A}_B^{-1} \mathbf{A}_N - \mathbf{c}_N^T$

    If $\mathbf{z}_N^T \leq 0$ **then** BREAK

    $k \leftarrow \text{argmax}\{\mathbf{z}_N^T\}$

    $\mathbf{y}_k \leftarrow \mathbf{A}_B^{-1} \mathbf{A}_{N_k}$

    If $\mathbf{y}_k \leq 0$ **then**

        If the artificial variables> 0 **then** STOP **return** infeasible solution

        else then STOP **return** unbounded solution

    $\mathbf{r} \leftarrow \mathbf{A}_B^{-1} \mathbf{b}$

    $l \leftarrow \text{argmin}\{\frac{r_i}{y_{ik}} | y_{ik} > 0, i \in B\}$

    $B \leftarrow (B \setminus \{l\}) \cup \{k\}$

    $N \leftarrow (N \setminus \{k\}) \cup \{l\}$

If the index of artificial variables is in B **then**

    If the artificial variables $== 0$ **then**

        $l \leftarrow$ the i^{th} row of the artificial variable to be the leaving variable.

        pivot the artificial indices out of B

        remove the artificial variables from the simplex tableau

    If the artificial variables > 0 **then** STOP

        return infeasible solution

Algorithm 2 The two-phase simplex algorithm (continued)

#Phase II
while True **do**
    $\mathbf{z}_N^T \leftarrow \mathbf{c}_B^T \mathbf{A}_B^{-1} \mathbf{A}_N - \mathbf{c}_N^T$
    If $\mathbf{z}_N^T \leq 0$ **then** STOP **return** the current BFS is the optimal solution
    $k \leftarrow \text{argmax}\{\mathbf{z}_N^T\}$
    $\mathbf{y}_k \leftarrow \mathbf{A}_B^{-1} \mathbf{A}_{N_k}$
    If $\mathbf{y}_k \leq 0$ **then** STOP **return** unbounded solution
    $\mathbf{r} \leftarrow \mathbf{A}_B^{-1} \mathbf{b}$
    $l \leftarrow \text{argmin}\{\frac{r_i}{y_{ik}} | y_{ik} > 0, i \in B\}$
    $B \leftarrow (B \setminus \{l\}) \cup \{k\}$
    $N \leftarrow (N \setminus \{k\}) \cup \{l\}$
end

2.9 Literature review

There are many researchers trying to improve some aspects of the simplex method such as the desirable choices of the pivot rule, the favorable starting basic feasible solution. In 2009, W-C. Yeh, H.W. Corley [3] introduced a pivot rule for reducing the number of iterations of the simplex method by the cosine technique. They used the gradient vector of constraints and the objective function to measure sizes of the angle in order to select the appropriate entering variable. In 2014, M. Tipawanna, K. Sinapiromsaran [14] proposed the max-out-in pivot rule which was guided by the maximum improvement before dealing with the objective improvement. They selected the leaving variable from the maximum of the rsh value to maximize the objective function value before selecting the entering variable. In 2016, an improvement of the initial vertex using the objective gradient direction was proposed by N. Yawila et. al. [16] which gave rise to the concept of the jump. They constructed artificial linearly independent hyperplanes containing the initial origin vertex forming the smaller feasible region for the simplex method to move to the next adjacent vertex. The simplex method would move along these artificial hyperplanes then it shifted to the adjacent vertex from the original LP model before dropping all artificial hyperplanes. In their result, this jump avoided quite a number of visiting vertices of the original LP models. But the algorithm added the n-1 artificial constraints into the LP problem which increased the number of constraints. This came the simplex method to pivot until all artificial constraints were driven out of the basis.

CHAPTER III

PRECEDING-JUMP SIMPLEX METHOD

The main idea of the preceding-jump simplex method for solving a linear programming (LP) problem is to avoid visiting unnecessary vertices by a jump. The jump is used for searching a new starting basic feasible solution or a new starting feasible vertex for the simplex method. This process is called the preceding-jump process. All jump points in this process will guarantee to be feasible. Therefore the starting point of jumps must belong to the feasible region and the direction of each jump must point into the feasible region. The process is divided into two phases which are the initial jump phase and the jump-to-vertex phase. In the initial jump phase, the method takes the gradient vector of the objective function to be the first feasible direction of the first jump. Then the initial jump point will be identified on a hyperplane of the constraints in the LP model using this direction. Since the initial jump point is normally not a vertex, the jump-to-vertex phase will be deployed for finding a vertex or a basic feasible solution near the initial jump point. The vertex becomes the starting vertex or the starting basic feasible solution of the simplex method.

In this thesis, the concept of the preceding-jump simplex method will be explained in two situations: the LP problem with the feasible origin point and the LP problem with the infeasible origin point. For the first situation, the preceding-jump simplex method can start at the origin point without any artificial technique while in the second situation it cannot start at the origin point. Therefore, phase I of the two-phase simplex method is applied before starting the preceding-jump simplex method.

3.1 The linear programming problem with the feasible origin point

In this situation, the origin point is feasible, so the preceding-jump simplex method will start at this point. In the initial jump phase of the preceding-jump process, the direction of the first jump is the gradient vector of the objective function. Because the

point along this direction will maximize the objective function value comparing to the other directions in term of the unit vector. Hence, the initial jump point will be identified on the hyperplane of the LP problem for the bounded feasible region. After that, the point will be shifted to other feasible jump points until it becomes the point of the n linearly intersecting hyperplanes using the jump-to-vertex phase. Finally, the method will start to find the optimal solution using the simplex method at the feasible vertex. The details of both phases will be described in the following subsections.

3.1.1 The initial jump phase

Assume that the feasible region is the nonempty set containing the origin point, $\mathbf{x}_0 = \mathbf{0} \in \mathbb{R}^{\mathbf{n}}$, as the initial vertex. The LP problem contains the constraints $\mathbf{A}_i\mathbf{x}_i \leq b_i, b_i \geq 0$ for $i = 1, 2, ..., m$ and $x_j \geq 0$ for $j = 1, 2, ..., n$. So the number of all constraints is equal to $m + n$. The point $\mathbf{x}_0$ expresses as the starting point of the initial jump phase. The jump process of this phase will begin with calculating the distance of the jump from starting point to another point on the hyperplanes along the direction. Since the starting point binds some hyperplanes then the distance of the jump will always equal to 0. This would not allow $\mathbf{x}_0$ to move along any direction. So the method needs to drop those hyperplanes or all hyperplanes binding at $\mathbf{x}_0$, before doing the jump process. The index set of the first binding hyperplanes denoted by FB. Since $\mathbf{x}_0$ binds to the hyperplanes initially of the form $x_j = 0$ for $j = 1, 2, ..., n$. Hence the set of the binding constraints, FB, contains the indices of $\{m+1, m+2, ..., m+n\}$ which coresponds to the constraints of $x_j \geq 0$ for $j = 1, 2, .., n$. Moreover, if there is i^{th} constraint in $\{1, 2, ..., m\}$ such that $\mathbf{A}_i\mathbf{x} = 0$ then i^{th} index will also be added into FB. Let the gradient vector of the objective function, $\mathbf{c}$, be the first direction $\mathbf{d}_0$. In the jump process, the point $\mathbf{x}_0$ will jump along $\mathbf{d}_0$ to $\mathbf{x}_1^i$ on the hyperplanes which the point binds to i^{th} hyperplane as shown in Figure 3.1. It corresponds to $\mathbf{A}_i\mathbf{x}_1^i = b_i$ where $\mathbf{x}_1^i = \mathbf{x}_0 + \alpha_0^i\mathbf{d}_0$ for $i \in \{1, 2, ..., m+n\}$. Before performing the jump process, the preceding-jump simplex method needs to drop i^{th} constraints in FB. Because the concept of the jump is to find the point on the hyperplanes along $\mathbf{d}_0$ which the point corresponds to the minimum distance between the starting point $\mathbf{x}_0$ and the point on the hyperplane i.e. $\mathbf{x}_1^i, i \in \{1, 2, ..., m+n\}$. Each point $\mathbf{x}_1^i$ on i^{th} hyperplane

must be $\mathbf{A}_i(\mathbf{x}_0 + \alpha_0^i \mathbf{d}_0) = b_i$ for $i \in \{1, 2, ..., m+n\} \setminus FB$. Since $\mathbf{x}_0$ is equal to $\mathbf{0}$, this implies that $\alpha_0^i = \frac{b_i}{\mathbf{A}_i \cdot \mathbf{d}_0}$. To maintain the feasiblity of the jump point as shown in Figure 3.1, the jump point must be the minimum of the positive α_0^i which is defined by $\mathbf{x}_1 = \alpha_0 \mathbf{d}_0$ where $\alpha_0 = \min\limits_{i \in \{1,2,...,m+n\} \setminus FB} \{\alpha_0^i\} = \min\limits_{i \in \{1,2,...,m+n\} \setminus FB} \left\{ \dfrac{b_i}{\mathbf{A}_i \cdot \mathbf{d}_0} > 0 \,\middle|\, \mathbf{A}_i \cdot \mathbf{d}_0 > 0 \right\}$ and $\mathbf{A}_i$ is the gradient vector of i^{th} constraint. Then $\mathbf{x}_1$ is called the initial jump point which belongs to i^{th} constraint. This process can be summarized in five steps

1. Let FB be the set of all binding hyperplanes at $\mathbf{x}_0$.
2. Denote the first direction by $\mathbf{d}_0 = \mathbf{c}$.
3. Compute $\alpha_0 = \min\limits_{i \in \{1,2,...,m+n\} \setminus FB} \left\{ \dfrac{b_i}{\mathbf{A}_i \cdot \mathbf{d}_0} > 0 \,\middle|\, \mathbf{A}_i \cdot \mathbf{d}_0 > 0 \right\}$.
4. Let $r_0 = \operatorname{argmin}\limits_{i \in \{1,2,...,m+n\} \setminus FB} \left\{ \dfrac{b_i}{\mathbf{A}_i \cdot \mathbf{d}_0} > 0 \,\middle|\, \mathbf{A}_i \cdot \mathbf{d}_0 > 0 \right\}$.
5. Compute $\mathbf{x}_1 = \mathbf{x}_0 + \alpha_0 \mathbf{d}_0$ binding r_0^{th} constraint.

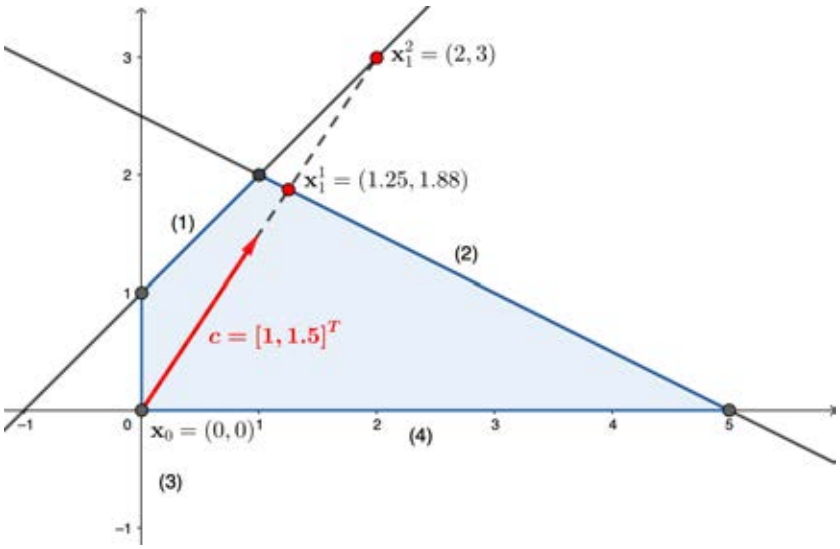

Figure 3.1: The initial jump phase

The starting point of the initial jump phase is $\mathbf{x}_0 = (0, 0)$ and the jump direction is $\mathbf{d}_0 = \mathbf{c} = [1, 1.5]^T$. At this point, the hyperplanes $x_j = 0, j = 1, 2$ is binded by $\mathbf{x}_0$ then

the set of the first binding constraints is $FB = \{3, 4\}$. From step 3 of the initial jump phase, α_0 is calculated by

$$\alpha_0 = \min_{i \in \{1,2,3,4\} \setminus FB} \{\frac{b_i}{\mathbf{A}_i \mathbf{d_0}} > 0\} = \left\{ \frac{5}{\begin{bmatrix} 1 \\ 2 \end{bmatrix} . \begin{bmatrix} 1 \\ 1.5 \end{bmatrix}}, \frac{1}{\begin{bmatrix} -1 \\ 1 \end{bmatrix} . \begin{bmatrix} 1 \\ 1.5 \end{bmatrix}} \right\} = \min\{\frac{5}{4}, 2\} = \frac{5}{4}.$$

Then the initial jump point is $\mathbf{x}_1 = \mathbf{x}_0 + \alpha_0 \mathbf{c} = (0, 0) + 1.25(1, 1.5) = (1.25, 1.88)$ which is represented by $\mathbf{x}_1^1$ in Figure 3.1. The new binding hyperplane is the second hyperplane so $r_0 = 2$. Notice that the initial jump point is not the vertex of the feasible region so the simplex method cannot start at this point. Thus it is mandatory to move to a vertex using the jump-to-vertex phase that shown in the subsection 3.1.2.

For an unbounded LP, the initial jump point may not exist. This implies that the method encounters the unbounded LP problem. In addition, if the first direction points away from the feasible region then there is no feasible initial jump point. Therefore, this direction cannot be used. The new direction that points into the feasible region will be derived.

3.1.1.1 The kinds of the initial jump

The initial jump point exists on the hyperplane from Figure 3.1 then this type of the initial jump is called **the bounded initial jump**. Sometimes the initial jump point may not exist because there is no hyperplane or edge for blocking the direction. This type of the initial jump is called **the unbounded initial jump**. If the jump point breaks out of the feasible region as shown in Figure 3.4, this type of the initial jump is called **the infeasible jump**. When the initial jump point does not belong to the feasible region then the last jump point or the vertex will not be feasible. This implies that the simplex method cannot start at that vertex. Thus, the initial jump phase needs to use a new feasible direction which will be shown in the next subsection.

Exmaple 3.1

$$\begin{aligned} \text{Maximize} \quad & x_1 + 2x_2 \\ \text{subject to} \quad & -x_1 + x_2 \leq 2.5 && (1) \\ & x_1 + 2x_2 \leq 10 && (2) \\ & x_1 - x_2 \leq 0 && (3) \\ & x_1 \geq 0 && (4) \\ & x_2 \geq 0 && (5) \end{aligned}$$

1. The bounded initial jump

The above problem contains five constraints covering the origin point $(0, 0)$ as shown in Figure 3.2. The initial jump point is $\mathbf{x}_1 = (2, 4)$ with $\mathbf{d}_0 = (1, 2), \alpha_0 = 2$. The initial jump point binds with the second hyperplane, this jump is called the initial bounded jump.

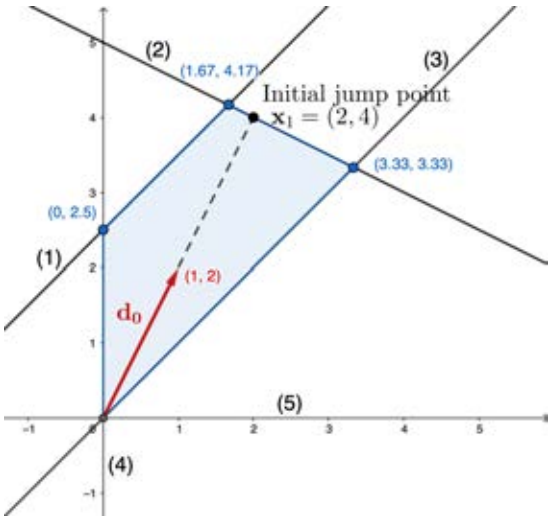

Figure 3.2: The initial jump point on the bounded feasible region

2. The unbounded initial jump

Suppose that the first two constraints are removed from the original problem and added the new constraint (6), $-2x_1 + 1x_2 \leq 3$ into the LP problem which is shown in Figure 3.3. The initial jump point $\mathbf{x}_1$ does not exist because there is no edge for supporting

the minimum distance of the jump process along $\mathbf{d}_0$. This implies that the α_0 increases indefinitely i.e.

$$\alpha_0 = \min\{\frac{b_6}{\mathbf{A}_6 \cdot \mathbf{d}_0}\} = \min\{\frac{3}{\begin{bmatrix} -2 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 2 \end{bmatrix}}\} = \infty.$$

This type of the initial jump is called the unbounded initial jump.

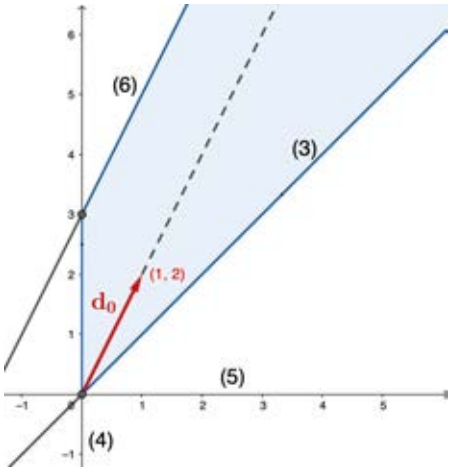

Figure 3.3: Jump into the unbounded feasible region

3. The infeasible jump

The important step of the initial jump phase is to create the initial jump point. Sometimes the direction $\mathbf{d}_0$ points away from the feasible region then increasing only a small positive value along the direction will violate some constraints. Suppose that the objective function is to maximize $1.5x_1 + x_2$. Then the new direction takes the point out of the feasible region as shown in Figure 3.4.

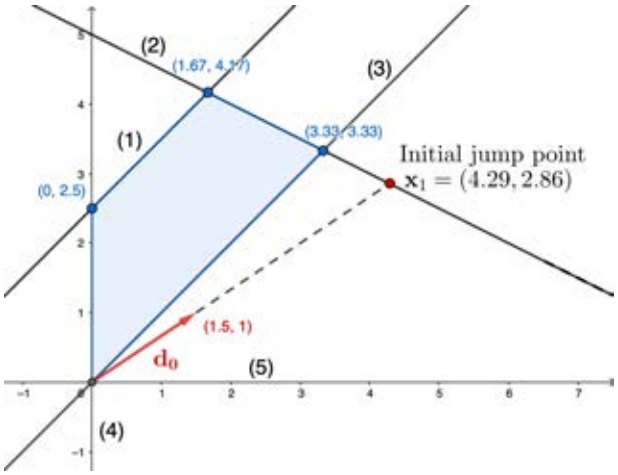

Figure 3.4: The jump point in the infeasible region

Point $\mathbf{x}_1$ violates the third constraint of the LP problem. Violating constraint of the initial jump point will cause the vertex of the last jump to be infeasible. In Figure 3.5, the new jump point $\mathbf{x}_2$ is found after it did the jump-to-vertex phase. Nevertheless, it does not belong to the feasible region then the simplex method cannot solve this LP problem using this point.

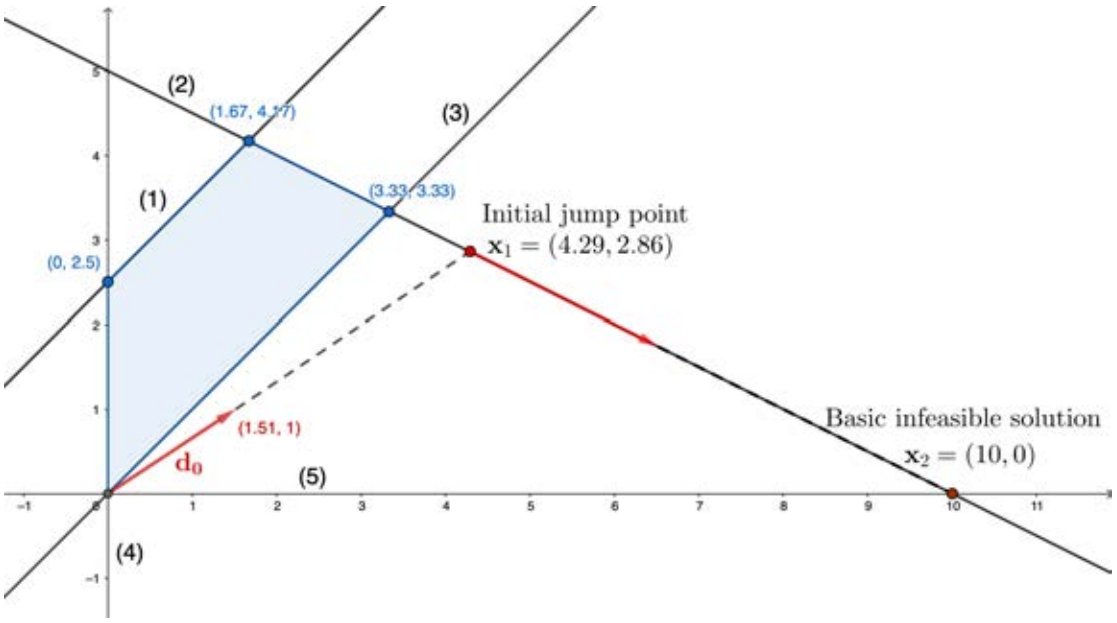

Figure 3.5: The basic infeasible solution

Therefore the preceding-jump simplex method needs to avoid using the given direction $\mathbf{d}_0$ that points away from the feasible region. The method has to maintain this problem by jumping with the new direction $\mathbf{d}_0^{'}$ which points into the feasible region. The new direction will be constructed by the search-direction process which will be shown in the next subsection.

3.1.1.2 The search-direction process

The basic idea of the search-direction process is to extract the feasible direction relies on creating the feasible point between two vertices of the polyhedron. Suppose that the feasible solution set of the LP problem is $F = \{\mathbf{x} \in \mathbb{R}^n | \mathbf{A}\mathbf{x} \leq b, \mathbf{x} \geq 0, \mathbf{b} \geq 0\}$ and F is nonempty. Let $\mathbf{x}^1$ and $\mathbf{x}^2$ be the vertices of the polyhedron. Since all vertices of the polyhedral correspond to F then $\mathbf{A}\mathbf{x}^1 \leq b, \mathbf{x}^1 \geq 0$ and $\mathbf{A}\mathbf{x}^2 \leq b, \mathbf{x}^2 \geq 0$. Let $\mathbf{x}$ be the point between $\mathbf{x}^1$ and $\mathbf{x}^2$ i.e. $\mathbf{x} = \lambda\mathbf{x}^1 + (1-\lambda)\mathbf{x}^2$ for $\lambda \in [0,1]$. Since both of $\mathbf{x}^1$ and $\mathbf{x}^2$ are nonnegative value then $\mathbf{x}$ is also nonnegative value. Considering for $\lambda \in [0,1]$ and $\mathbf{x} = \lambda\mathbf{x}^1 + (1-\lambda)\mathbf{x}^2$ such that

$$\mathbf{A}\mathbf{x} = \mathbf{A}(\lambda\mathbf{x}^1 + (1-\lambda)\mathbf{x}^2) = \lambda\mathbf{A}\mathbf{x}^1 + (1-\lambda)\mathbf{A}\mathbf{x}^2 \leq \lambda\mathbf{b} + (1-\lambda)\mathbf{b} = \mathbf{b}$$

So point $\mathbf{x}$ belongs to F, this implies that it always has the feasible point between two vertices of the polyhedron. Thus the searching-direction process will find two vertices to create a new feasible point $\mathbf{x}$. After that the new direction will be created by $\mathbf{d}_0^{'} = \mathbf{x} - \mathbf{x}_0$. This direction will take the initial jump point $\mathbf{x}_1$ into the feasible region.

Both vertices $\mathbf{x}^1$ and $\mathbf{x}^2$ are found by performing the simplex tableau. The first step of this process, the method will construct the first simplex tableau. The method will select the entering variable which corresponds to the maximum positve reduced cost. Performing simplex method until it founds the new vertex i.e. $\mathbf{x}^1$. After that the method will repeat the simplex tableau to the first tableau using the entering vaiable which corresponds to the minimum positive reduced cost. Then the method will pivot until it found another vertex i.e. $\mathbf{x}^2$. Now, the method has two vertices and it already creates the new direction $\mathbf{d}_0^{'}$ by $\frac{\mathbf{x}^1+\mathbf{x}^2}{2} - \mathbf{x}_0$.

The idea of choosing the entering variables of the first simplex tableau is to find a feasible improving vector $\mathbf{d}_0'$ that can avoid visiting unnecessary vertices. In other word, the initial jump point may move far away from these vertices using $\mathbf{d}_0'$. Hence the direction should lie between $\mathbf{x}^1$ and $\mathbf{x}^2$. The summary of the searching direction process is described below.

1. Create the first simplex tableau.
2. Choose the first entering variable from the maximum positive reduced cost of the non-basic variables.
3. Perform the simplex algorithm until it found the new vertex and denote it by $\mathbf{x}^1$.
4. Choose the scond entering variable of the first simplex tableau from the mimimum positive reduced cost of the nonbasic variables.
5. Perform the simplex algorithm until it found the new vertex and denote it by $\mathbf{x}^2$.
6. Transform both $\mathbf{x}^1$ and $\mathbf{x}^2$ into the normal dimension (n original variables).
7. The new direction is $\mathbf{d}_0' = \frac{\mathbf{x}^1+\mathbf{x}^2}{2} - \mathbf{x}_0$.

Figure 3.6 shows the new direction that it is created using the search-direction process. It begins by choosing the first entering variable as x_1 because the reduced cost of x_1 gives the maximum positve value. The second entering is x_2 which the reduced cost gives the minimum positive value. For the first entering variables, the simplex method will perform until it reached the new basic feasible solution, $(x_1, x_2, x_3, x_4, x_5) = (3.33, 3.33, 2.5, 0, 0)$ or the new vertex $\mathbf{x}^1 = (3.33, 3.33)$.

		x_1	x_2	x_3	x_4	x_5	rhs
Iter. 0	z	3/2	1	0	0	0	0
	x_3	-1	1	1	0	0	5/2
	x_4	1	2	0	1	0	10
	x_5	1	-1	0	0	1	0
Iter. 1	z	0	5/2	0	0	-3/2	0
	x_3	0	0	1	0	1	5/2
	x_4	0	3	0	0	-1	10
	x_1	1	-1	0	0	1	0
Iter. 2	z	0	0	0	0	-2/3	-25/3
	x_3	0	0	1	0	1	5/2
	x_2	0	1	0	0	-1/3	10/3
	x_1	1	0	0	0	2/3	10/3

Table 3.1: The simplex tableau of the search-direction process with x_1 as the entering variable.

Considering the first simplex tableau, the method has to resolve the simplex method from the first tableau by choosing x_2 as the entering variable. This time, the simplex tableau uses only one iteration to find the new basic feasible solution. Then it reaches out the basic feasible solution such a $(x_1, x_2, x_3, x_4, x_5) = (0, 2.5, 0, 5, 2.5)$ or the vertex $\mathbf{x}^1 = (0, 2.5)$ in Figure 3.6.

		x_1	x_2	x_3	x_4	x_5	rhs
Iter. 0	z	3/2	1	0	0	0	0
	x_3	-1	1	1	0	0	5/2
	x_4	1	2	0	1	0	10
	x_5	1	-1	0	0	1	0
Iter. 1	z	5/2	0	-1	0	0	-5/2
	x_2	-1	1	1	0	0	5/2
	x_4	3	0	-2	1	0	5
	x_5	0	0	1	0	1	5/2

Table 3.2: The simplex tableau of the search-direction process with x_2 as the entering variable.

The new direction for the jump process is $\mathbf{d}_0' = \frac{\mathbf{x}^1+\mathbf{x}^2}{2} - \mathbf{x}_0 = (1.665, 2.915)$.

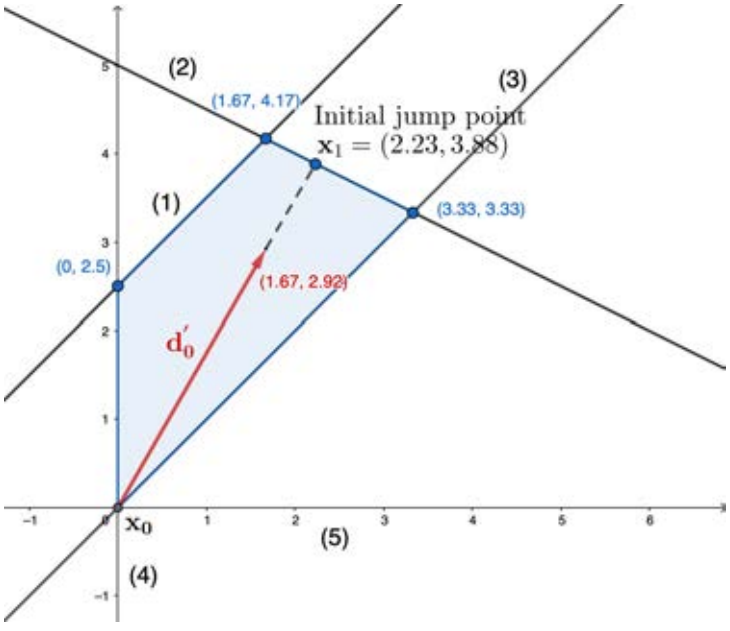

Figure 3.6: The new direction $\mathbf{d}_0'$ from the search-direction process

From the initial jump phase, it tries to create the initial jump point along the

direction $\mathbf{d}_0$. If the direction points into the bounded feasible region and the starting point belongs to the feasible region then the initial jump point will exist on the hyperplane of the LP problem. Some LP problems, the direction not always points into the feasible region then the new direction $\mathbf{d}_0^{'}$ is used instead of $\mathbf{d}_0$. After that, the initial jump point is embedded on some hyperplanes and it is also a feasible point. This hyperplane is called the visited hyperplane V. The method will keep the index of the binding hyperplane at the initial jump point into the set V. In the next phase, the initial jump point will be shifted to the vertex by jumps. Each jump will maintain the method will keep the index of the hyperplane which is visited by each jump point into V. The number of elements of V relates to the number of the intersection of hyperplanes. If the number of visited constraints V equals to the number of variables of the problem, (n), then it forms the basic feasible solution or the vertex. If the number of elements of V does not equal to the number of the variables n, the jump point needs to keep jumping until it binds n linearly constraints i.e. the vertex. This process will be performed in the jump-to-vertex phase, which is shown in the next subsection.

3.1.2 The jump-to-vertex phase

The goal of this phase is to find a vertex around the intial jump point by jumps. In this phase, the initial jump point $\mathbf{x}_1$ will be shifted to another feasible point $\mathbf{x}_2$ using the new direction $\mathbf{d}_1$ which parallels to the hyperplanes in the set of visited hyperplane, V. The direction $\mathbf{d}_1$ needs to parallel to the hyperplanes in V to guarantee all points will belong to all constraints in V to be the binding hyperplanes as the previous point. Thus the direction $\mathbf{d}_1$ is created by subtracting the point $\mathbf{x}_1^{'}$ and $\mathbf{x}_1$, where $\mathbf{x}_1^{'}$ is derived by solving $\mathbf{A}_i\mathbf{x}_1^{'} = \mathbf{b}_i$ and $\mathbf{A}_i$ is the submatrix $\mathbf{A}$ containing i^{th} row of $\mathbf{A}$ and $i \in V$. After that the $\mathbf{x}_1$ will move along $\mathbf{d}_1$ to bind the remaining constraints in $\{1, 2, ..., m+n\} \setminus V$. The new point $\mathbf{x}_2$ will be blocked by constraint r_1 for some $r_1 \in \{1, 2, ..., m+n\} \setminus V$ i.e. $\mathbf{A}_{r_1}\mathbf{x}_2 = b_{r_1}$. In addition, the new jump point $\mathbf{x}_2$ must stays within the feasible region. Since the jump point is $\mathbf{x}_2 = \mathbf{x}_1 + \alpha_1\mathbf{d}_1$ then $A_{r_1}(\mathbf{x}_1 + \alpha_1\mathbf{d}_1) = \mathbf{b}_{r_1}$. This implies that $\alpha_1 = \dfrac{\mathbf{b}_{r_1} - \mathbf{A}_{r_1} \cdot \mathbf{x}_1}{\mathbf{A}_{r_1} \cdot \mathbf{d}_1}$. Then the new feasible jump point corresponds to $\mathbf{x}_2 = \mathbf{x}_1 + \alpha_1\mathbf{d}_1$ where $\alpha_1 = \dfrac{\mathbf{b}_{r_1} - \mathbf{A}_{r_1} \cdot \mathbf{x}_1}{\mathbf{A}_{r_1} \cdot \mathbf{d}_1} = \min\limits_{i \in \{1,2,...,m+n\} \setminus V} \left\{ \dfrac{b_i - \mathbf{A}_i \cdot \mathbf{x}_k}{\mathbf{A}_i \cdot \mathbf{d}_k} \geq 0 \,\middle|\, \mathbf{A}_i \cdot \mathbf{d}_k > 0 \right\}$ and

$$r_1 = \underset{i\in\{1,2,\ldots,m+n\}\setminus V}{\operatorname{argmin}} \left\{ \frac{b_i - \mathbf{A}_i \cdot \mathbf{x}_k}{\mathbf{A}_i \cdot \mathbf{d}_k} \geq 0 \,\middle|\, \mathbf{A}_i \cdot \mathbf{d}_k > 0 \right\}.$$

This process is repeated in 6 steps for finding a vertex.

1. Consider k^{th} jump with $\mathbf{x}_k$ as the current feasible jump point and V is the set of all visited constraints.

2. Solve the linear subsystem from all visited hyperplanes to get $\mathbf{x}'_k$. Note that $\mathbf{A}_i\mathbf{x}'_k = b_i$ for $i \in V$, and then the point is defined as $\mathbf{x}'_k = \mathbf{A}^{\mathrm{g}}_V b_V$ where $\mathbf{A}^{\mathrm{g}}_V$ is the right inverse of $\mathbf{A}_V$.

3. The direction $\mathbf{d}_k$ is defined by $\mathbf{d}_k = \mathbf{x}'_k - \mathbf{x}_k$ if $\mathbf{c} \cdot \mathbf{d}_k > 0$ and $\mathbf{d}_k = \mathbf{x}_k - \mathbf{x}'_k$ if $\mathbf{c} \cdot \mathbf{d}_k < \mathbf{0}$. Note that if $\mathbf{x}_k = \mathbf{x}'_k$ then $\mathbf{d}_k$ will be generated using the artificial hyperplane process which will be shown in the next subsection.

4. Compute $\alpha_k = \min\limits_{i\in\{1,2,\ldots,m+n\}\setminus V} \left\{ \frac{b_i - \mathbf{A}_i \cdot \mathbf{x}_k}{\mathbf{A}_i \cdot \mathbf{d}_k} \geq 0 \,\middle|\, \mathbf{A}_i \cdot \mathbf{d}_k > \mathbf{0} \right\}$ from the set of the nonbinding constraints, $I \setminus V$.

5. Find $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{d}_k$ and add $r_k = \underset{i\in\{1,2,\ldots,m+n\}\setminus V}{\operatorname{argmin}} \left\{ \frac{b_i - \mathbf{A}_i \cdot \mathbf{x}_k}{\mathbf{A}_i \cdot \mathbf{d}_k} \geq 0 \,\middle|\, \mathbf{A}_i \cdot \mathbf{d}_k > 0 \right\}$ in V.

6. Repeat steps 1 - 5 until the number of elements of the set V is equal to the number of variables, n, of the LP problem. Then $\mathbf{x}_k$ will be the feasible vertex and it is the starting vertex for the simplex method.

Consider this process with the following LP problem.

Exmaple 3.2

$$\begin{aligned} &\text{Maximize } x_1 + 2x_2 + 2x_3 \\ &\text{subject to} \quad x_1 + x_2 + x_3 \leq 1 && (1) \\ &\qquad\qquad\quad x_1 \geq 0 && (2) \\ &\qquad\qquad\quad x_2 \geq 0 && (3) \\ &\qquad\qquad\quad x_3 \geq 0 && (4) \end{aligned}$$

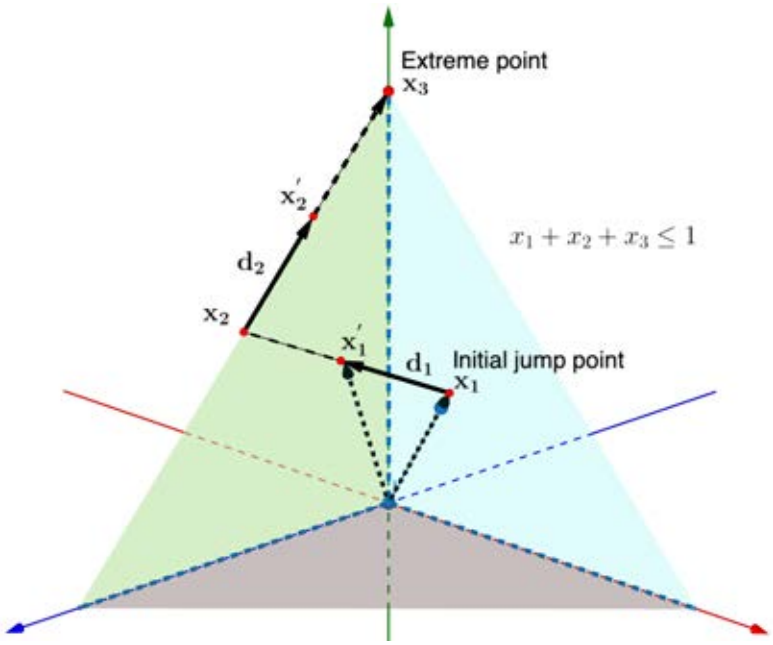

Figure 3.7: The jump-to-vertex phase

From Figure 3.7, the three dimensional LP problem has the constraint $x_1+x_2+x_3 \leq 1$ covering the origin point. Obviously, the initial jump point $\mathbf{x}_1$ of this problem is not a vertex. Thus, the initial jump point has to be shifted to a vertex before applying the simplex method. The jump-to-vertex phase needs to generate the new direction $\mathbf{d}_1$. The first step of creating the direction is to find another point on the hyperplane in $V = \{1\}$ which is computed by solving $\mathbf{A_1 x_1^{'}} = \mathbf{b}_1$ and let $\mathbf{d}_1 = \mathbf{x}_1^{'} - \mathbf{x}_1$ and it will be tested with the vector gradient of the objective function, $\mathbf{c}$. If $\mathbf{d}_1 \cdot \mathbf{c} > \mathbf{0}$ then it is the improving direction of the LP problem. Otherwise, reset $\mathbf{d}_1 = \mathbf{x}_1 - \mathbf{x}_1^{'}$. Now, $\mathbf{x}_2$ binds on the constraints $x_1 + x_2 + x_3 \leq 1$ and $x_1 \geq 0$ then the visited hyperplane is $V = \{1, 2\}$. However, $\mathbf{x}_2$ is still not the vertex of the LP problem. So the method will repeat this process until it reaches the vertex, $\mathbf{x}_3$. After that the simplex method will be started at this point $\mathbf{x}_3$.

The above example shows the process to create the direction of the jumps in the jump-to-vertex phase. Sometimes, the point $\mathbf{x}_k^{'}$ and $\mathbf{x}_k$ are the same point. This implies

that the direction $\mathbf{d}_k$ cannot be generated for the jump. Then it can be fixed by solving with a new linear system having an artificial hyperplane.

3.1.2.1 The artificial hyperplane

The step of generating a direction before doing the jump point is to subtract $\mathbf{x}_k^{'}$ by $\mathbf{x}_k$. However, it is possible that both points will be the same point. This implies that the direction $\mathbf{d}_k$ will not exist. The method needs to find the new $\mathbf{x}_k^{'}$ from the intersection point between the artificial hyperplane and the hyperplane in V as shown in Figure 3.8. Then $\mathbf{d}_k^{'}$ can be constructed by subtracting both points as shown in the jump-to-vertex phase to guarantee the increase of the objective value. Considering the artificial hyperplane, the plane is created from the objective function adding the positive value β i.e. $\mathbf{c}^T\mathbf{x} = z + \beta$ where $z = \mathbf{c}^T\mathbf{x}_k$. After that, the intersection point of the hyperplanes in V and the artificial hyperplane will be solved for finding the new $\mathbf{x}_k^{'}$. Since the point $\mathbf{x}_k^{'}$ is on the improving artificial hyperplane and the hyperplanes in V so the point $\mathbf{x}_k^{'}$ cannot be equal to $\mathbf{x}_k$. The process performs 6 steps as follows.

1. Compute $z = \mathbf{c}^T\mathbf{x}_k$.
2. Add some positive β to z to guarantee increase of the objective function value.
3. The artificial constraint is $\mathbf{c}^T\mathbf{x} = z + \beta$ which is denoted by l.
4. Put the artificial constraint, l, into the set of visited constraints, V, and the new point is solved using the right inverse as shown in the jump-to-vertex phase, i.e. $\mathbf{x}_k^{'} = \mathbf{A}_i^{\mathrm{g}}\mathbf{b}_i$ for $i \in V$.
5. Remove the artificial hyperplane, l, from V.
6. The direction $\mathbf{d}_k$ is defined by $\mathbf{d}_k = \mathbf{x}_k^{'} - \mathbf{x}_k$.

Considering the following Figure 3.8, the hyperplane $x_1 + x_2 + x_3 = 1$ is intesecting to the artificial hyperplane. Assume that the objective function is $2x_1 + x_2 + x_3$. Since the initial jump point is $\mathbf{x}_1 = (0.5, 0.25, 0.25)$ thus the current objective function value

is 1.5. At the second steps of the above process, the objective value at $\mathbf{x}_1$ will be added by the positive value, β. Suppose that the given β is 0.1 then the artificial hyperplane is $2x_1 + x_2 + x_3 = 1.5 + 0.1 = 1.6$ and the index of the hyperplane is denoted by $l = 5$. Next step, the method has to add l^{th} constraint into the set V. After that the method has to solve $\mathbf{A}_i\mathbf{x}_1^{'} = \mathbf{b}_i$ for i in $V = \{1, 5\}$. The solution of this linear subsystem is $\mathbf{x}_1^{'} = (0.6, 0.2, 0.2)$. Next, the artificial hyperplane l will be removed from V. Hence the new improve direction is $\mathbf{d}_1^{'} = \mathbf{x}_1^{'} - \mathbf{x}_1 = [0.1, -0.05, -0.05]^T$ and $V = \{1\}$. After that the method will begin the jump-to-vertex phase at $\mathbf{x}_1$ using the direction $\mathbf{d}_1^{'}$.

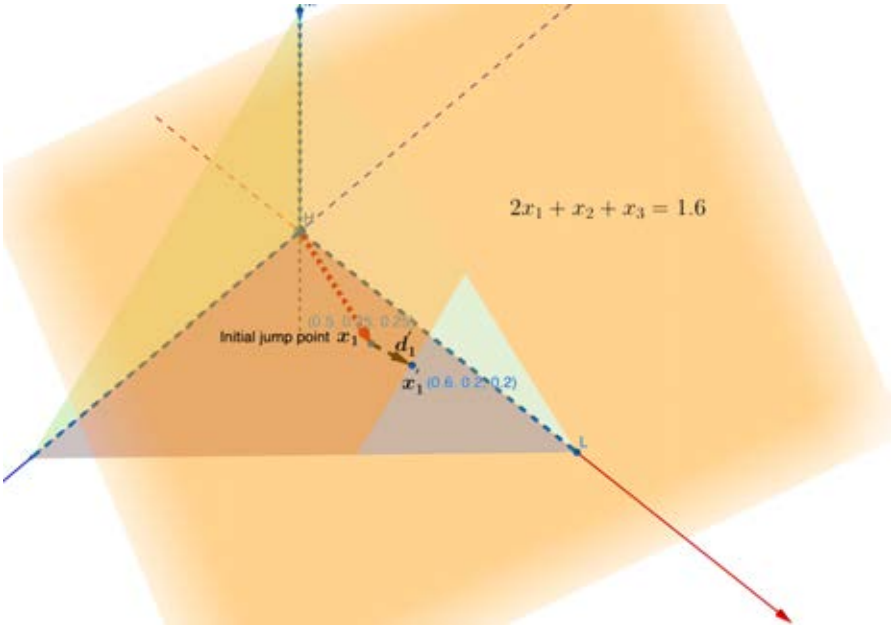

Figure 3.8: Creating direction in the artificial hyperplane process

All steps of the preceding-jump simplex method for the case of the origin point belonging to the feasible region of the LP problem will be demonstrated in the example 3.3. The method starts with doing the initial jump phase. After that, the jump-to-vertex phase will be deployed until it founds the feasible vertex.

Example 3.3

$$\text{Maximize } x_1 + 1.5x_2$$

$$\text{subject to } -x_1 + x_2 \leq 1 \quad (1)$$

$$x_1 + 2x_2 \leq 5 \quad (2)$$

$$x_1 \geq 0 \quad (3)$$

$$x_2 \geq 0 \quad (4)$$

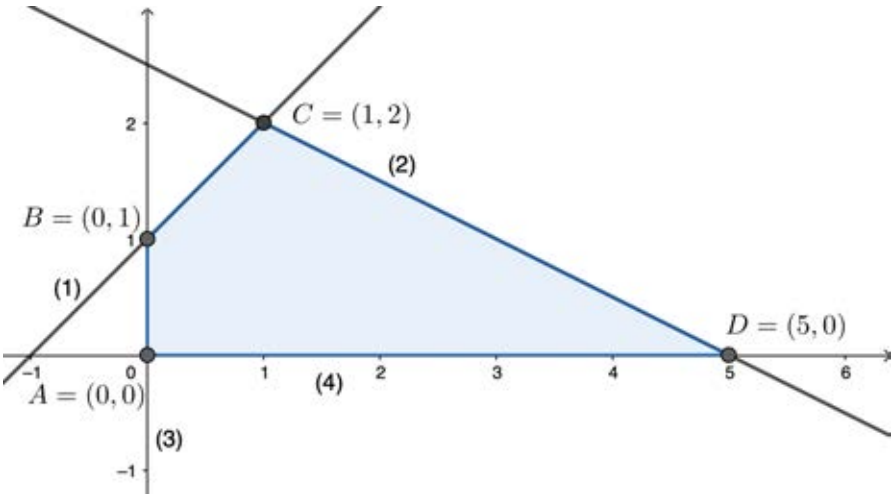

Figure 3.9: The feasible region of example 3.3

For the initial jump phase of example 3.3, the LP problem has to find the maximum solution that corresponds to all constraints. Figure 3.9, the origin point belongs to the feasible region of the four contraints. The initial jump phase will begin at this point with the direction $\mathbf{d}_0 = \mathbf{c} = [1, 1.5]^T$. In the initial jump phase, the initial jump point will be determined by calculating the minimum step of the $\mathbf{d}_0$ from the origin point to the hyperplanes, i.e. α_0. Those hyperplanes are $-x_1 + x_2 = 1, x_1 + 2x_2 = 5, x_1 = 0, x_2 = 0$ respectively and the set of the first binding hyperplanes, FB, is $\{3, 4\}$. α_0 is calculated

as

$$\alpha_0 = \min_{i\in\{1,2,3,4\}\setminus FB}\{\frac{b_i}{\mathbf{A}_i\mathbf{c}}\} = \left\{\frac{5}{\begin{bmatrix}1\\2\end{bmatrix}\cdot\begin{bmatrix}1\\1.5\end{bmatrix}}, \frac{1}{\begin{bmatrix}-1\\1\end{bmatrix}\cdot\begin{bmatrix}1\\1.5\end{bmatrix}}\right\} = \min\{\frac{5}{4}, 2\} = \frac{5}{4}.$$

Now, the initial jump point is defined as $\mathbf{x}_1 = \alpha_0\mathbf{c} = (1.25, 1.875)$ and the point is bounded by the second constraint, $V = \{2\}$. After that, the initial jump point moves to the vertex near it using the jump-to-vertex phase.

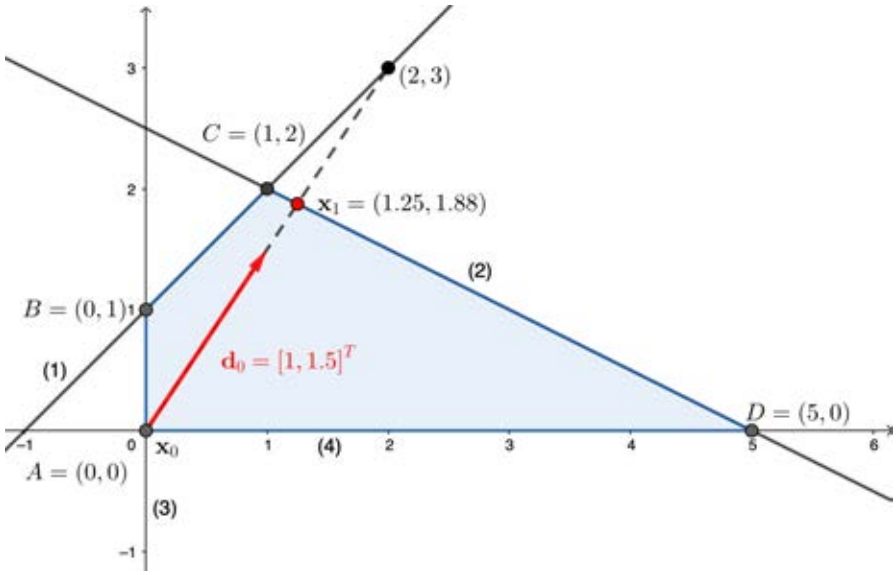

Figure 3.10: The initial jump phase of example 3.3

Figure 3.10, the initial jump point is not the vertex of the LP problem, so the method cannot start the simplex method at this point $\mathbf{x}_1$. It needs to perform the jump-to-vertex phase to find the feasible vertex for the simplex method. The process will be similar to the previous phase but it will use the new direction which gives the improved objective function value. Before getting that direction, the method will generate another point on the visited hyperplane $V = \{2\}$. Now, the visited hyperplane is $x_1 + 2x_2 = 5$ and then the method solves $\mathbf{A}_2\mathbf{x}_1^{'} = b_2$ or $x_1^{'} + 2x_2^{'} = 5$. In this situation, the matrix

$\mathbf{A}_2$ is the full rank matrix but it is not the square matrix. To solve this problem, the right inverse of $\mathbf{A}_2$ will be used instead of the regular inverse of $\mathbf{A}_2$. The right inverse can be directly computed by $\mathbf{A}_2^g = \mathbf{A}_2^T(\mathbf{A}_2\mathbf{A}_2^T)^{-1}$, then the solution of $\mathbf{A}_2\mathbf{x}_1^{'} = b_2$ is $\mathbf{x}_2^{'} = \mathbf{A}_2^T(\mathbf{A}_2\mathbf{A}_2^T)^{-1}b_2$ or

$$\mathbf{x}_\mathbf{1}^{'} = \mathbf{A}_2^T(\mathbf{A}_2\mathbf{A}_2^T)^{-1}b_2 = \begin{bmatrix}1\\2\end{bmatrix}\left([1,2]\begin{bmatrix}1\\2\end{bmatrix}\right)^{-1}(5) = \begin{bmatrix}1\\2\end{bmatrix}(\tfrac{1}{5})5 = \begin{bmatrix}1\\2\end{bmatrix}.$$

The new point on the hyperplane is $\mathbf{x}_1^{'} = (1,2)$. The direction $\mathbf{d}_1$ is created by subtracting $\mathbf{x}_1^{'}$ and $\mathbf{x}_1$ together that is $\mathbf{d}_1 = \mathbf{x}_1^{'} - \mathbf{x}_1 = (1,2) - (1.25, 1.875) = [-0.25, 0.125]^T$. Now, the starting point of the jump-to-vertex phase is $\mathbf{x}_1$ and the direction of the jump process is $\mathbf{d}_1$. It is ready to jump to the next hyperplanes. Before starting the jump process, the method will test $\mathbf{d}_1$ whether this direction gives a better objective function value or not. Testing the direction by performing $\mathbf{d}_1 \cdot \mathbf{c}$. If $\mathbf{d}_1 \cdot \mathbf{c} > 0$ so this direction will improve the objective value otherwise the direction will become $-\mathbf{d}_1$. Considering $\mathbf{d}_1 \cdot \mathbf{c} = -0.0625 < 0$ then the new direction is $\mathbf{d}_1^{'} = -\mathbf{d}_1 = [0.25, -0.125]^T$.

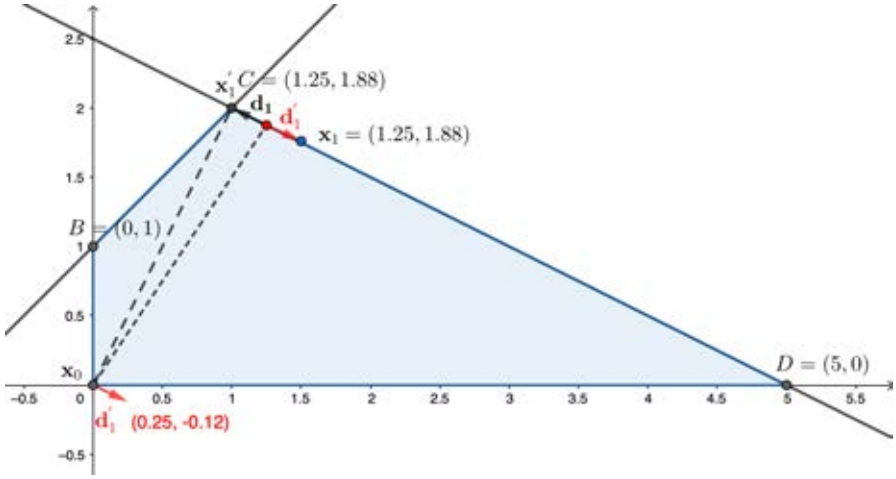

Figure 3.11: The jump-to-vertex phase of example 3.3

In the jump process, the method needs to compute the minimum step of $\mathbf{d}_1 := \mathbf{d}_1^{'}$ from $\mathbf{x}_1$ to the remaining constraints, α_1, using the following formula.

$$\alpha_k = \min_{i \in \{1,2,\ldots,m+n\} \setminus V} \left\{ \frac{\mathbf{b}_i - \mathbf{A}_i \cdot \mathbf{x}_k}{\mathbf{A}_i \cdot \mathbf{d}_k} \geq 0 \,\middle|\, \mathbf{A}_i \cdot \mathbf{d}_k > 0 \right\}$$

Then α_1 is minimum of $\dfrac{\mathbf{b}_i - \mathbf{A}_i \cdot \mathbf{x}_1}{\mathbf{A}_i \mathbf{d}_1} \geq \mathbf{0}$ where $\mathbf{A}_i \mathbf{d}_1 > 0, i \in \{1,2,3,4\}$ and $i \notin V = \{1\}$. The remaining hyperplanes are the $2,3,4$ hyperplanes, then the α_1 is 15.

$$\alpha_1 = \min \left\{ \frac{1 - \begin{bmatrix} -1 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} 1.25 \\ 1.875 \end{bmatrix}}{\begin{bmatrix} -1 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} 0.25 \\ -0.125 \end{bmatrix}}, \frac{0 - \begin{bmatrix} 1 \\ 0 \end{bmatrix} \cdot \begin{bmatrix} 1.25 \\ 1.875 \end{bmatrix}}{\begin{bmatrix} 1 \\ 0 \end{bmatrix} \cdot \begin{bmatrix} 0.25 \\ -0.125 \end{bmatrix}}, \frac{0 - \begin{bmatrix} 0 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} 1.25 \\ 1.875 \end{bmatrix}}{\begin{bmatrix} 0 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} 0.25 \\ -0.125 \end{bmatrix}} \right\} = \{-1, -5, 15\} = 15$$

Now the new jump point is $\mathbf{x}_2 = \mathbf{x}_1 + \alpha_1 \mathbf{d}_1 = (1.25, 1.875) + 15(0.25, -0.125) = (1.25, 1.875) + (3.75, -1.875) = (5, 0)$ belonging to the constraint of $x_2 \geq 0$ i.e. the fourth constraints in Figure 3.10. Now, the set of visited constraints is $V = \{1, 4\}$ and the number of elements in V is equal to the number of variables, n. Therefore the jump point $\mathbf{x}_2$ is also the vertex of the LP problem. The last step, the preceding-jump simplex method will start the simplex method at $\mathbf{x}_2$ to search for the optimal vertex or the optimal solution. The result of this problem gives the optimal solution at $D = (5, 0)$ with the objective value is 5.

The preceding-jump simplex method is an assistance method of the simplex method. This subsection purposed the complete machanic steps of the new method in case of the origin vertex is feasible. This is the primary condition for the preceding-jump simplex method to almost guarantee that all jump points are still feasible. However, the feasibility of the jump points also depends on the initial direction whether it takes the jump point to the feasible region or not. If the origin vertex or the starting point of the jump process belongs to the feasible region but the direction points away from the feasible region then the jump point will not exist. Then the method will fix this issue by using the search-direction process. In the next section, the preceding-jump simplex method will be extended to solve the LP problem in case of the infeasible origin point. The method will use the same technique to find a new feasible starting point for the preceding-jump simplex method.

3.2 The linear programming problem with the infeasible origin point out

The feasible starting point and the feasible direction of the jump process is the important steps of maintaining the feasibility of each jump point. If the starting point does not belong to the feasible region then the given vertex of the preceding-jump process will be infeasible. Thereofore, the starting point of the jump process must belong to feasible region before performing the jump process. From previous section 3.1, the feasible starting point immediatly found at the origin point then the method can start the jump process at this point. However, the general LP problems may not contain the feasible origin point. This implies that the method does not know where is the feasible starting point. For this situation, the preceding-jump simplex method needs to find the feasible starting point using phase I of the two-phase simplex method. The steps of this method will be described in the example 3.4, the LP problem of the example does not hold the origin point in the feasible region.

Suppose that the maximization problem contains four constraints and all variables are positive. If the first direction is $\mathbf{d}_0 = \mathbf{c} = [1, 1.5]^T$ and the method did the initial jump phase with the infeasible starting point $\mathbf{x}_0 = (0, 0)$ then the jump point $\mathbf{x}_1$ does not belong to the feasible region as shown in Figure 3.12. This implies that the vertex, after the preceding-jump process is done, might not belong to the feasible region then the simplex method will not be able to start. The following problem will show the infeasible jump point in Figure 3.12.

Example 3.4

$$\text{Maximize } x_1 + 1.5x_2$$

$$\begin{aligned} \text{subject to} \quad x_1 + 2x_2 &\leq 10 && (1)\\ -x_1 + x_2 &\leq 4 && (2)\\ x_1 &\geq 2 && (3)\\ x_2 &\geq 2 && (4) \end{aligned}$$

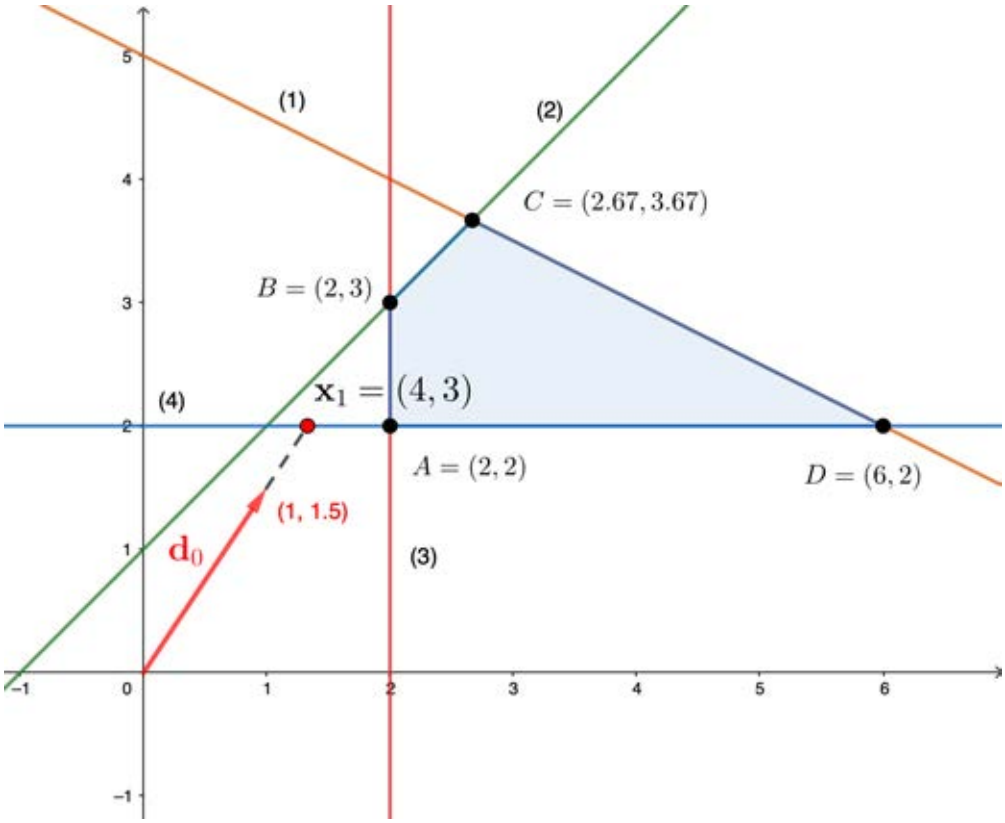

Figure 3.12: Jump into the infeasible region

The preceding-jump simplex method can start if the starting point is feasible. So the method needs the supporting method for finding the feasible starting point such the phase I of the two-phase simplex method.

3.2.1 The general preceding-jump simplex method

The general LP problem may not be solvable using the preceding-jump simplex method unless some feasible point is detected. This issue can be fixed by applying the first phase of two phases simplex method. Phase I of the two-phase simplex method deals with the computation of an initial feasible basis, which is then passed over to phase two of the simplex method described in Chapter II. From the example above, the phase I is calculated from the following problem.

Phase I

$$\begin{aligned} \text{Minimize} \quad & x_{a_1} + x_{a_2} \\ \text{subject to} \quad & x_1 + 2x_2 + x_3 = 10 \\ & -x_1 + x_2 + x_4 = 4 \\ & x_1 - x_5 + x_{a_1} = 2 \\ & x_2 - x_6 + x_{a_2} = 2 \\ & x_1, x_2 \geq 0 \end{aligned}$$

After the method has been completed in phase I, the given vertex from the phase I is the vertex $A = (2, 2)$ as shown in Figure 3.13. The feasible point $\mathbf{x}_0 = A$ will become the starting point of the preceding-jump simplex method in the next phase. Note that $\mathbf{x}_0$ belongs to the constraint $x_1 \geq 2$ and $x_2 \geq 2$ then the current binding hyperplanes are the hyperplanes of $x_1 = 2$ and $x_2 = 2$ i.e. the fifth and the sixth constraint. Thus the set $FB = \{5, 6\}$ will be denoted as the first binding hyperplanes. Before doing the initial jump phase, the method needs to remove the constraints in FB out of the set $I = \{1, 2, ..., m + n\}$ in the jump process.

Phase II (the preceding-jump simplex method)

From this example, phase 1 returns $\mathbf{x}_0 = (2, 2)$ then the jump process uses this point to start the initial jump phase of phase II. The jump point will be completed at $\mathbf{x}_1 = \mathbf{x}_0 + \alpha_0 \mathbf{d}_0$ where $\alpha_0 = \min_{i \in \{1,2,...,m+n\} \setminus FB} \{ \frac{\mathbf{b}_i - \mathbf{A}_i \mathbf{x}_0}{\mathbf{A}_i \mathbf{d}_0} | \mathbf{A}_i \mathbf{d}_0 > 0, \mathbf{d}_0 = \mathbf{c} \text{ and } \mathbf{x}_0 = (2, 2) \}$. Similarly to the initial jump phase of the section 3.1, α_0 is equal to 1 with the jump point $\mathbf{x}_1 = (4, 3) = (2, 2) + 1(2, 1)$ which it binds the first constraint of the LP problem. Thus $r_0 = 1$ with respect to α_0 will be added to V.

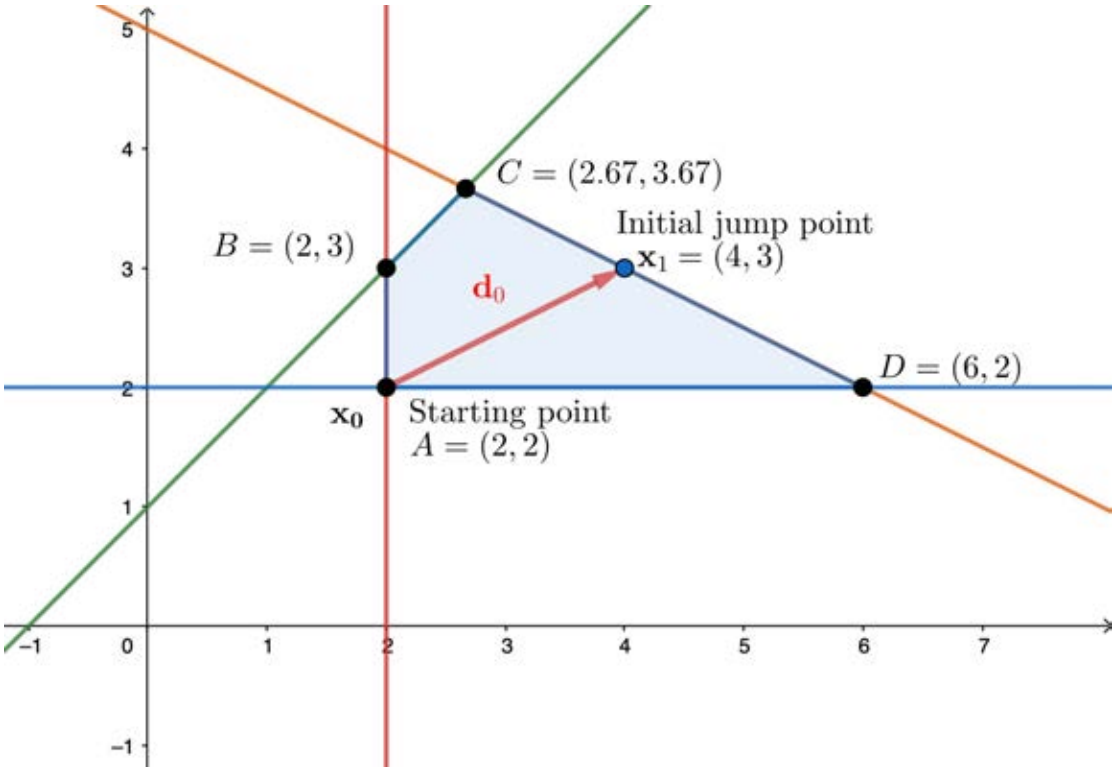

Figure 3.13: The initial jump phase of example 3.4

This implies that the index of the first constraint becomes an element in the set V. Now the jump point is not the vertex then the jump-to-vertex phase is applying. Searching the vertex in this phase is similar to the jump-to-vertex phase of the section 3.1. In this step, the new point $\mathbf{x}_1^{'}$ on the hyperplanes in $V = \{1\}$ will be created and it is $\mathbf{x}_1^{'} = (2, 4)$. So the new direction is $\mathbf{d}_1 = \mathbf{x}_1^{'} - \mathbf{x}_1 = (2, 4) - (4, 3) = (-2, 1)$ and $\mathbf{d}_1 \cdot \mathbf{c} = -0.5 < 0$. This implies that this direction will give a decreasing objective value if it jumps along this direction. Thus the method will convert $\mathbf{d}_1$ by $\mathbf{d}_1^{'} := -\mathbf{d}_1$. The last step of this phase is to calculate the minimum step length of the direction $\mathbf{d_1}$ from $\mathbf{x}_1$ to the remaining constraints, i.e. α_1.

$$\alpha_1 = \min_{i \in \{1,2,3,4\} \setminus V} \left\{ \frac{4 - \begin{bmatrix} -1 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} 4 \\ 3 \end{bmatrix}}{\begin{bmatrix} -1 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} 2 \\ -1 \end{bmatrix}}, \frac{2 - \begin{bmatrix} 1 \\ 0 \end{bmatrix} \cdot \begin{bmatrix} 4 \\ 3 \end{bmatrix}}{\begin{bmatrix} 1 \\ 0 \end{bmatrix} \cdot \begin{bmatrix} 2 \\ -1 \end{bmatrix}}, \frac{2 - \begin{bmatrix} 0 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} 4 \\ 3 \end{bmatrix}}{\begin{bmatrix} 0 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} 2 \\ -1 \end{bmatrix}} \right\} = \{-1.667, -1, 1\} = 1$$

The next jump point is $\mathbf{x}_2 = \mathbf{x}_1 + \alpha_1 \mathbf{d}_1 = \mathbf{x}_1 + \alpha_1 \mathbf{d}_1^{'} = (4,3) + 1(2,-1) = (6,2)$ where $V = \{1,4\}$. Hence this point is the point of the feasible region and the number of elements in V is equal to the variables of the LP problem. This implies that the method already has the vertex for the simplex method. Moreover, the vertex $\mathbf{x}_2$ is also the optimal solution.

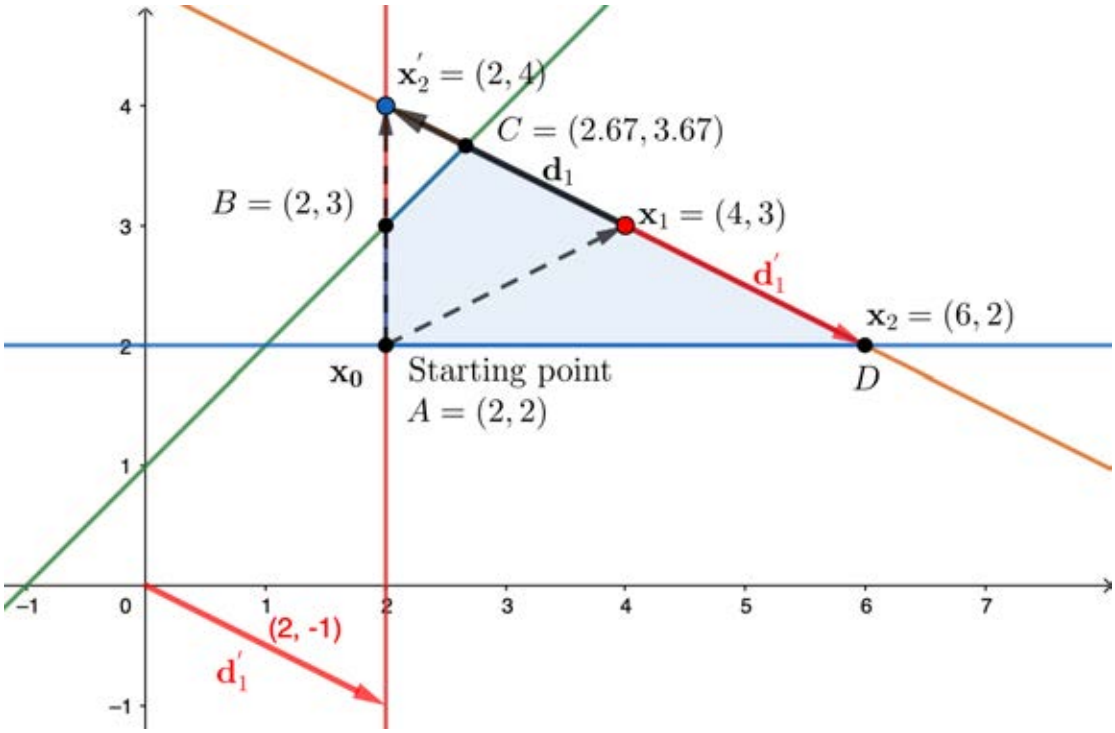

Figure 3.14: The jump-to-vertex phase of example 3.4

The following subsections will explain all steps of the precding-jump simplex method in the LP problem without the feasible origin point. It also contains both phases as appeared in the section 3.1. But in this situation, the method has to find the feasible starting point before starting the jump process. The details of both phases will be described in the next subsection.

3.2.1.1 The initial jump phase

Assume that the feasible region is a nonempty set containing $\mathbf{x}_0 \in \mathbb{R}^n$ as the initial starting point from phase I. Denote $\mathbf{d}_0$ as the first direction and it is equal to $\mathbf{c}$. The initial jump point is $\mathbf{x}_1 = \mathbf{x}_0 + \alpha_0 \mathbf{d}_0$ where $\alpha_0 = \min_{i \in \{1,2,\ldots,m+n\} \setminus FB} \left\{ \dfrac{\mathbf{b}_i - \mathbf{A}_i \mathbf{x}_0}{\mathbf{A}_i \cdot \mathbf{d}_0} > 0 \,\middle|\, \mathbf{A}_i \cdot \mathbf{d}_0 > 0 \right\}$

and $\mathbf{A}_i$ is the gradient vector of each constraint.

1. Denote FB is the set of the first binding constraint.
2. Convert the vertex or the basic feasible solution from phase I of the two-phase simplex method into its original dimension (remove the artificial variable and slack variable from phase I) then the point has only n components.
3. Denote the vertex by $\mathbf{x}_0$ which is called the starting point of the preceding-jump simplex method.
4. Define the first jumping direction $\mathbf{c}$ as $\mathbf{d}_0$.
5. Calculate $\alpha_0 = \min\limits_{i\in\{1,2,\ldots,m+n\}\setminus FB} \left\{ \dfrac{\mathbf{b}_i - \mathbf{A}_i\mathbf{x}_0}{\mathbf{A}_i \cdot \mathbf{d}_0} > 0 \,\middle|\, \mathbf{A}_i \cdot \mathbf{d}_0 > 0 \right\}$.
6. The initial jump point is $\mathbf{x}_1 = \mathbf{x}_0 + \alpha_0\mathbf{d}_0$ and the method has to check the feasibility of the point by substituting the point into every constraint.
7. If the jump point violates some constraints (the jump point breaking out of the feasible region) then the $\mathbf{d}_0$ will be changed to $\mathbf{d}_0^{'}$ which it is created from the search-direction process.
8. The initial jump phase will be repeated step 5 and 6 until it absolutely obtains the initial feasible jump point.
9. Keep the last visited constraint r_0 into V where
   $r_0 = \operatorname{argmin}\limits_{i\in\{1,2,\ldots,m+n\}\setminus FB} \left\{ \dfrac{\mathbf{b}_i - \mathbf{A}_i\mathbf{x}_0}{\mathbf{A}_i \cdot \mathbf{d}_0} > 0 \,\middle|\, \mathbf{A}_i \cdot \mathbf{d}_0 > 0 \right\}$.

The example 3.4 is shown an overview of the preceding-jump simplex method in the phase II with the feasible jump point. In the general LP problem, the jump point may not be feasible if the first direction, $\mathbf{d}_0$, points away from the feasible region. Then this problem will be fixed by creating the new feasible direction $\mathbf{d}_0^{'}$ using the search-direction process. Assume that example 3.4 has another infeasible direction $\mathbf{d}_0 = [-1, 1]^T$.

$$\text{Maximize } -x_1 + x_2$$

$$\text{subject to } x_1 + 2x_2 \leq 10 \quad (1)$$

$$-x_1 + x_2 \leq 4 \quad (2)$$

$$x_1 \geq 2 \quad (3)$$

$$x_2 \geq 2 \quad (4)$$

Phase I : the infeasible jump which shows in the following figure.

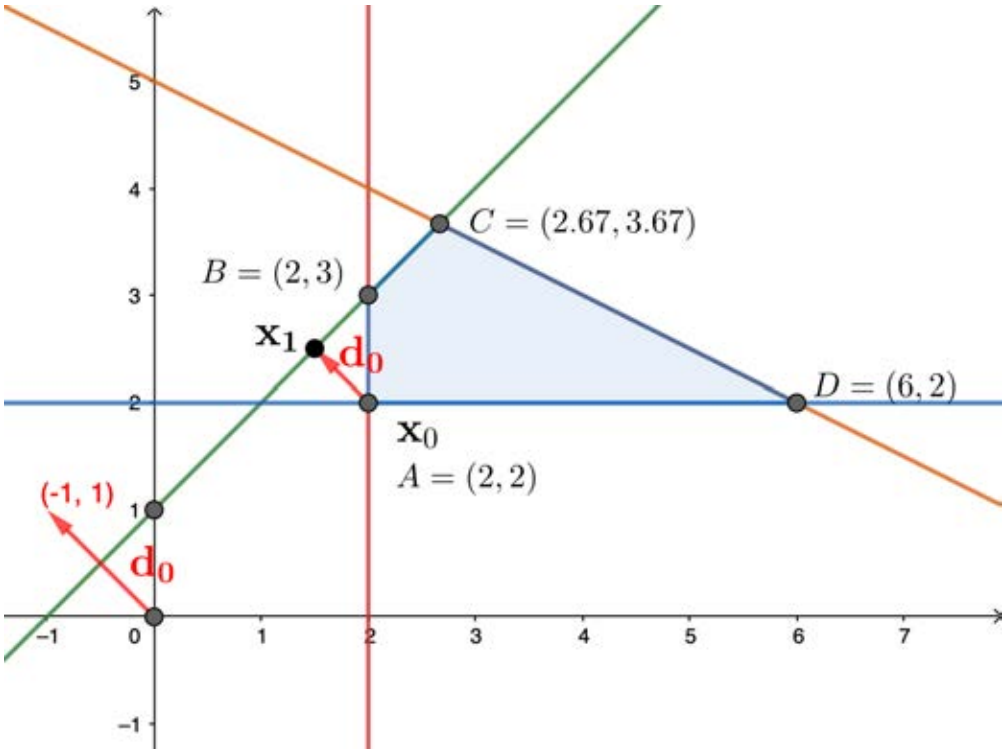

Figure 3.15: Jump into the infeasible region

Considering the initial jump point along the gradient vector of the objective function $\mathbf{c} = \mathbf{d}_0$. Then the new point from the initial jump phase is the point with the minimum distance between the initial vertex A and every hyperplane along that direction. The first hyperplane blocks points along the current direction. Thus the new jump point is $\mathbf{x}_1 = (1.5, 2.5)$. Notice that, the point binds hyperplane 2 but it is not feasible. Then the

method does not have enough the feasible basic variables to do the simplex method. It fixes this issue in the search-direction process.

3.2.2 The search-direction process

Since the direction $\mathbf{c}$ or $\mathbf{d}_0$ points away of the feasible region then every jump point will become the infeasible point. Fixing the issue, the preceding-jump simplex method has to use another direction, $\mathbf{d}_0^{'}$. The direction is created by the sumation of the directions that point into the feasible region. In addition, this direction needs to make an acute angle to the gradient objective function i.e. the direction $\mathbf{c}$. Hence the direction $\mathbf{d}_0^{'}$ defines as the average vector of intersection between the artificial hyperplane and the visited hyperplanes. This direction is going to be created in 7 steps.

1. Define AC is the set of the acute angle of each gradient vector of constraints and $\mathbf{c}$ i.e. the angle of $\mathbf{c}$ and $\mathbf{A}_i$ for $i = 1, 2, ..., m + n$.

2. After phase I completes, the method will select the acute angle from step 1 which the index of the hyperplanes in FB and put them into the AC.

3. If AC is an empty set then start the simplex method for finding the vertices $\mathbf{x}^1$ and $\mathbf{x}^2$ as shown in the search-direction process in the previous scenario then $\mathbf{d}_0 = \frac{\mathbf{x}^1+\mathbf{x}^2}{2} - \mathbf{x}_0$. Otherwise, it has to create an artificial hyperplane by $\mathbf{c}^T\mathbf{x} = \mathbf{c}^T\mathbf{x}_0 + \beta$ where $\mathbf{x}_0$ is the given feasible vertex from phase I of the two-phase simplex method and β is a positive instance.

4. The intersection point of the artificial hyperplane and the hyperplanes in AC will be solved using the right inverse of $\mathbf{A}_j$ and $\mathbf{c}$, i.e. $\mathbf{x}_{0_j}^{'} = \mathbf{A}_j^{\mathrm{g}} b_j, j \in AC$. Note that if there are constraints in AC which is represented by " = " then $\mathbf{x}_{0_j}^{'}$ will be solved only these constraints.

5. Generate the direction from $\mathbf{d}_{0_j} = \mathbf{x}_{0_j}^{'} - \mathbf{x}_0$ for all j in AC.

6. Do the initial jump phase with these directions $\mathbf{d}_{0_j}$ then checking those jump points whether it violates some constraints. If the jump point did not violate the con-

straints, it will be combined into the new direction i.e $\mathbf{d}_0^{'} = \sum_{j=1}^{L} \mathbf{d}_{0_j}$ where $j \in AC$. If all jump points using $\mathbf{d}_{0_j}$ violate some constraints then $\mathbf{d}_0^{'}$ will be create from the search-direction process from the previous phase.

7. Do the initial jump phase with the direction $\mathbf{d}_0^{'}$.

Figure 3.16 shows the new direction $\mathbf{d}_0^{'}$ that it was created using the search-direction process. It keeps the active constraints of the first binding hyperplanes FB into the set of acute angle constraints AC using $\theta_i = \arccos(\frac{\mathbf{A}_i\mathbf{c}}{\|\mathbf{A}_i\|\|\mathbf{c}\|}), i \in FB$. So the starting point is the vertex $A = (2, 2)$ and the active constraints are $x_1 \geq 2, x_2 \geq 2$. Note the angle of their gradients is computed by $\theta = \{\arccos \frac{\mathbf{A}_3 c}{\|\mathbf{A}_3\|\|\mathbf{c}\|}, \arccos \frac{\mathbf{A}_4\mathbf{c}}{\|\mathbf{A}_4\|\|\mathbf{c}\|}\} = \{59.03, 30.96\}$. Both make the acute angle to $\mathbf{c}$ then $AC = \{3, 4\}$. The next step, the method has to create an artificial hyperplane using $\mathbf{c}^T\mathbf{x} = \mathbf{c}^T\mathbf{x}_0 + \beta$ which assume β is equal to 1. Considering the value of the objective function at the vertex A, $\mathbf{c}^T\mathbf{x}_0 = -(2) + 1(2) = 0$ then the artificial hyperplane is $-x_1 + x_2 = 0 + 1 = 1$. The next step of the search-direction process is to solve the equations A_i, i in AC and $-x_1 + x_2 = 1$. Then the intercept point of both hyperplanes are $\{(1, 2), (2, 3)\}$ respectively. This implies that the new directions are $\mathbf{d}_{0_3}^{'} = \mathbf{x}_{0_3}^{'} - \mathbf{x}_0 = (1, 2) - (2, 2) = [-1, 0]^T$ and $\mathbf{d}_{0_4}^{'} = \mathbf{x}_{0_4}^{'} - \mathbf{x}_0 = (2, 3) - (2, 2) = [0, 1]^T$. The method will perform the initial jump with these directions and checking the feasiblity of those jump points which are $\{(1, 2), (2, 3)\}$ but (1,2) violates 4^{th} constraint. Then the direction $\mathbf{d}_0^{'}$ is $[0, 1]^T$ which is shown in Figure 3.16. After that the preceding-jump will take $\mathbf{d}_0^{'}$ to create the initial jump point in the initial jump phase.

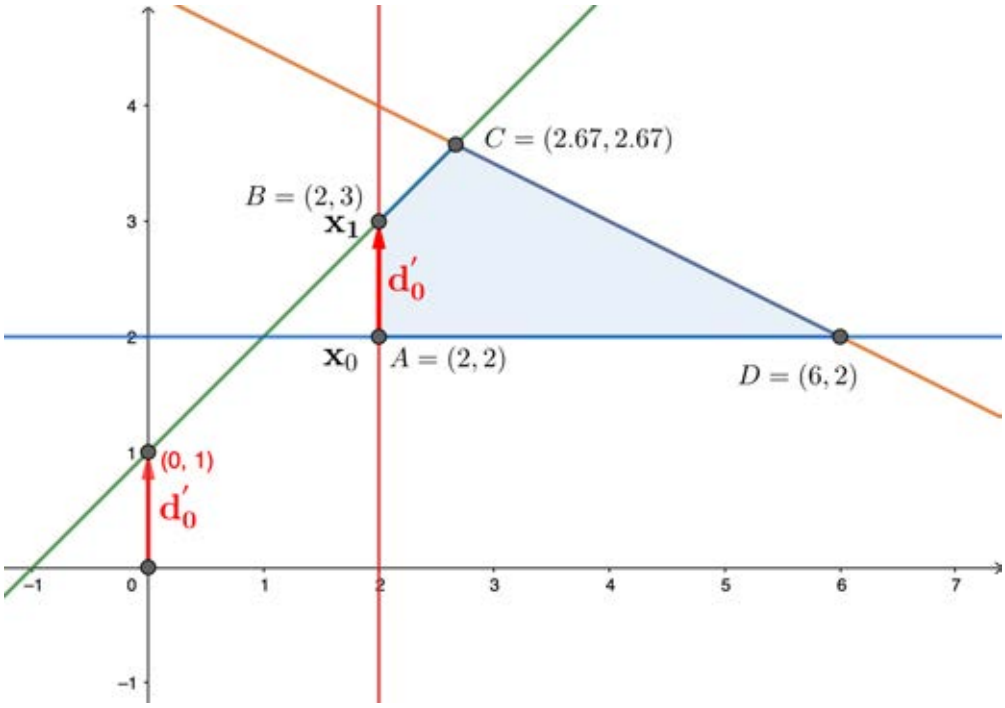

Figure 3.16: New direction from the search-direction process

From the figure above, the initial jump point is the point $B = (2, 3)$. Moreover the point B is also the vertex. Then another phase such a jump to vertex phase does not need to do anything at the jump point. Now it is ready to do the simplex method at this vertex for searching the optimal solution. In the general of the LP problem, the initial jump point might not be a vertex yet. So the method has to do such a jump-to-vertex phase as well.

3.2.2.1 Pseudo code of the general preceding-jump simplex algorithm

Before starting the preceding-jump simplex algorithm, the method has to find the feasible starting point from phase I of the two-phase simplex method. Then, both the initial jump phase and the jump-to-vertex phase will be combined together for findding a vertex. After that the simplex method will begin at the vertex. All steps of the algorithm will be shown in the following algorithm 3.

Algorithm 3 The general preceding-jump simplex algorithm

procedure PJS($\mathbf{c}, \mathbf{b}, \mathbf{A}$)
  Input: an LP problem $\mathbf{c}, \mathbf{b}, \mathbf{A}$
  Output: the optimal solution $\mathbf{x} = \{x_1, ..., x_n\}$
  #Phase 1
  while True **do**
    $\mathbf{z}_N^T \leftarrow \mathbf{c}_B^T \mathbf{A}_B^{-1} \mathbf{A}_N - \mathbf{c}_N^T$
    If $\mathbf{z}_N^T \leq 0$ **then** STOP **return** $(\mathbf{x}_B, \mathbf{x}_N)$
    $k \leftarrow \text{argmax}\{\mathbf{z}_N^T\}$
    $\mathbf{y}_k \leftarrow \mathbf{A}_B^{-1} \mathbf{A}_{N_k}$
    If $\mathbf{y}_k \leq 0$ **then** STOP **return** infeasible solution
    $\mathbf{r} \leftarrow \mathbf{A}_B^{-1} \mathbf{b}$
    $l \leftarrow \text{argmin}\{\frac{r_i}{y_{ik}} | y_{ik} > 0, i \in B\}$
    $B \leftarrow (B \setminus \{l\}) \cup \{k\}$
    $N \leftarrow (N \setminus \{k\}) \cup \{l\}$
  If $\mathbf{x}_{a_k} \in \mathbf{x}_B$ and $\mathbf{x}_{a_k} = 0$ for some $k = \{1, 2, .., l\}$ **then**
    $\mathbf{x}_B \leftarrow$ pivot the artificial out of $\mathbf{x}_B$
  else $\mathbf{x}_{a_k} \in \mathbf{x}_B$ and $\mathbf{x}_{a_k} > 0$ for some $k = \{1, 2, .., l\}$ **then**
    return infeasible solution
  $\mathbf{x}_0 \leftarrow (\mathbf{x}_B, \mathbf{x}_N)$
  #Initial jump phase
  $FB \leftarrow$ Keep the indices of bindding hyperplanes at $\mathbf{x}_0$
  $\mathbf{x}_0 \leftarrow$ Remove slack and surplus variables from $\mathbf{x}_0$
  $\mathbf{d}_0 \leftarrow \mathbf{c}$ the gradient objective function
  $r_0 \leftarrow \text{argmin}\{\frac{b_i - \mathbf{A}_i \cdot \mathbf{x}_0}{\mathbf{A}_i \cdot \mathbf{d}_0} > 0 | \mathbf{A}_i \cdot \mathbf{d}_0 > 0, i \in \{1, 2, ..., m+n\} \setminus FB\}$
  $\alpha_0 \leftarrow \frac{b_{r_0} - \mathbf{A}_{r_0} \cdot \mathbf{x}_0}{\mathbf{A}_{r_0} \cdot \mathbf{d}_0}$
  $\mathbf{x}_1 \leftarrow \mathbf{x}_0 + \alpha_0 \mathbf{d}_0$
  $V \leftarrow V \cup \{r_0\}$
  If $\mathbf{x}_1$ violates some constraints **then**
    #The search-direction process
    $AC \leftarrow \{i \in FB | \arccos(\frac{\mathbf{A}_i \cdot \mathbf{c}}{\|\mathbf{A}_i\| \|\mathbf{c}\|}) > 0\}$
    If AC is an empty set **then**
      $\mathbf{x}^1 \leftarrow$ the vertex from the simplex algorithm using the entering variable from the maximum positive reduce cost
      $\mathbf{x}^2 \leftarrow$ the vertex from the simplex algorithm using the entering variable from the minimum positive reduce cost
      $\mathbf{d}_0 \leftarrow \frac{\mathbf{x}^1 + \mathbf{x}^2}{2} - \mathbf{x}_0$
    else
      $\psi \leftarrow c^T \mathbf{x}_0 + \beta$ where $\beta \in \mathbb{R}^+$

Algorithm 3 The preceding-jump simplex algorithm (continued)

$\mathbf{A_c} \leftarrow$ the artificial hyperplane $\mathbf{c}^T\mathbf{x} = \psi$
for i **in** AC
$G \leftarrow \{i\} \cup \{c\}$
$\mathbf{x}_i^{'} \leftarrow \mathbf{A}_G^{\mathrm{g}}\mathbf{b}_G$
$G \leftarrow G \setminus \{i\}$
$\mathbf{d}_i^{'} \leftarrow \mathbf{x}_i^{'} - \mathbf{x}_0$
If $\mathbf{d}_i^{'}$ does not violate any constraint **then**
$\mathbf{d}_0 \leftarrow \sum_j \mathbf{d}_i^{'}$
else do the step 32 to36
$\mathbf{x}_1 \leftarrow$ repeat steps 21 to 27
#Jump-to-vertex phase
while $|V| < n$ **do**
$\mathbf{x}_k' \leftarrow \mathbf{A}_V^{\mathrm{g}}\mathbf{b}_V$ where k is k^{th} jump point
If $\mathbf{x}_k = \mathbf{x}_k^{'}$ **then**
#The artificial hyperplane process
$\psi \leftarrow \mathbf{c}^T\mathbf{x}_0 + \beta$ where $\beta \in \mathbb{R}^+$
$\mathbf{A}_c \leftarrow$ the artificial hyperplane $\mathbf{c}^T\mathbf{x} = \psi$
$V \leftarrow V \cup \{c\}$
$\mathbf{x}_k^{'} \leftarrow \mathbf{A}_V^{\mathrm{g}}\mathbf{b}_V$
$V \leftarrow V \setminus \{c\}$
$\mathbf{d}_0^{'} \leftarrow \mathbf{x}_k^{'} - \mathbf{x}_k$
else
$\mathbf{d}_k \leftarrow \mathbf{x}_k' - \mathbf{x}_k$
If $\mathbf{c} \cdot \mathbf{d}_k < 0$ **then**
$\mathbf{d}_k \leftarrow -\mathbf{d}_k$
$\mathbf{x}_{k+1} \leftarrow \mathbf{x}_k + \alpha_k\mathbf{d}_k$
$r_k \leftarrow \underset{i \in \{1,2,..,m+n\}\setminus \mathbf{V}}{\operatorname{argmin}} \left\{ \frac{b_i - \mathbf{A}_i \cdot \mathbf{x}_k}{\mathbf{A}_i \cdot \mathbf{d}_k} \geq 0 \,|\, \mathbf{A}_i \cdot \mathbf{d}_k > 0 \right\}$ in V
$V \leftarrow V \cup \{r_k\}$
end
$\mathbf{x}_N \leftarrow$ the slack variables in V
$\mathbf{x}_B \leftarrow x_i \notin \mathbf{x}_N$ where $i \in \{1, 2, ..., m+n\}$
#The simplex algorithm
while True **do**
$\mathbf{z}_N^T \leftarrow \mathbf{c}_B^T\mathbf{A}_B^{-1}\mathbf{A}_N - \mathbf{c}_N^T$
If $\mathbf{z}_N^T \leq 0$ **then** STOP **return** the current BFS is the optimal solution
$k \leftarrow \mathrm{argmax}\{\mathbf{z}_N^T\}$
$\mathbf{y}_k \leftarrow \mathbf{A}_B^{-1}\mathbf{A}_{N_k}$
If $\mathbf{y}_k \leq 0$ **then** STOP **return** infeasible solution
$\mathbf{r} \leftarrow \mathbf{A}_B^{-1}\mathbf{b}$
$l \leftarrow \mathrm{argmin}\{\frac{r_i}{y_{ik}} | y_{ik} > 0, i \in B\}$
$B \leftarrow (B \setminus \{l\}) \cup \{k\}$
$N \leftarrow (N \setminus \{k\}) \cup \{l\}$
end

CHAPTER IV

EXPERIMENTS AND RESULTS

Performance of the preceding-jump simplex method (PJS) is evaluated by applying to the randomly generated LP problems in two scenarios. The first scenario deals with the LP model having the feasible origin point. In other words, this model can be directly solved by the simplex method (SPX). In the second scenario, the assumption is that the feasible region does not contain the origin point. PJS together with the phase I of the two-phase simplex method had been used. The details of the randomly generated LP problems will be presented in the next section and the performance will be reported.

4.1 Randomly generated problems with the feasible origin point

The synthetic LP problems in this thesis consist of randomly generated $\mathbf{Ax} \leq \mathbf{b}$ constraints, $\mathbf{x} \geq \mathbf{0}$ and $\mathbf{b} \geq \mathbf{0}$. The number of variables and constraints are set to be the same n varying from $n = 100$ to $n = 2500$. Each of the results in the table 4.1 shows the average of iterations and the average of running time from 10 randomly LP problems. The randomly generated LP coefficients are created according to the following criteria, $c_i, a_{ij} \in [-9, 9]$ and $x_j \in [0, 9]$. Then the vector $\mathbf{b}$ is computed from $\mathbf{b} = \mathbf{Ax} \cdot \text{sign}(\mathbf{Ax})$ where the function sign returns the negative or positive of $\mathbf{Ax}$. In order to guarantee the positivity of $\mathbf{b}$, let $\mathbf{b} = \mathbf{Ax} \cdot \text{sign}(\mathbf{Ax})$. Then the rhs vector $\mathbf{b}$ is added by the positive vector $[5, \ldots, 5]^{\text{T}}$ to confirm $\mathbf{Ax} \leq \mathbf{b}$. Table 4.1 shows the experimental results of both methods by their iterations, time (sec.) and improvement (%). Each column of the table is separated into two subcolumns which contain the average number of iterations and the average total running time of both methods. In addition to the running-time column, the results are shown by the jumping time of the jump process and the total running time of PJS. The last two columns of the table show the improvement of PJS for saving the number of simplex iterations and running time. Improvement of iterations and time are

computed as follows,

$$\text{Improvement of the number of iterations} = (\frac{\text{SPX iterations} - \text{PJS iterations}}{\text{SPX iterations}}) \times 100$$

and

$$\text{Improvement of time} = (\frac{\text{SPX time} - \text{PJS time}}{\text{SPX time}}) \times 100.$$

Hence, the positive value indicates the improvement over SPX and the negative value indicates the deterioration over SPX.

(m, n)	Iterations		Time (sec.)			Improvement (%)	
	PJS	SPX	PJS		SPX	Iterations	Time
			Jump	Total	Total		
100	114.700	290.900	0.052	0.142	0.200	60.571	28.778
200	403.700	1109.400	0.178	0.813	1.429	63.611	43.149
300	801.800	2399.900	1.771	3.615	5.078	66.590	28.821
400	1247.300	4384.700	4.398	8.095	12.147	71.553	33.364
500	2297.500	6911.200	9.512	18.141	24.888	66.757	27.110
600	2702.300	10070.100	16.674	29.385	45.456	73.165	35.356
700	3846.500	13766.000	28.601	50.475	76.190	72.058	33.751
800	5348.600	19258.800	51.570	88.151	129.567	72.228	31.965
900	6736.400	24028.600	77.805	132.438	192.230	71.965	31.105
1000	8152.200	29724.400	111.712	189.818	279.815	72.574	32.163
1200	11035.700	44402.800	207.542	351.501	565.440	75.146	37.836
1400	14257.300	60331.400	331.380	561.641	962.533	76.368	41.650
1600	18233.300	79008.700	562.844	922.446	1545.439	76.922	40.312
1800	24886.800	106142.200	781.395	1404.140	2621.985	76.553	46.447
2000	30190.300	132224.300	1115.499	2007.276	3875.009	77.167	48.199
2500	43452.700	203917.700	2273.835	4089.269	8481.765	78.691	51.788

Table 4.1: The results of randomly generated problems with the origin point in the feasible region

The results of both methods are confirmed by the open software PULP to check the

optimal objective value from PJS and SPX which are the same for all cases. In the improvement column, it clearly shows the number of iterations that is reduced significantly. PJS can also save the total running time of solving the LP problems better than SPX.

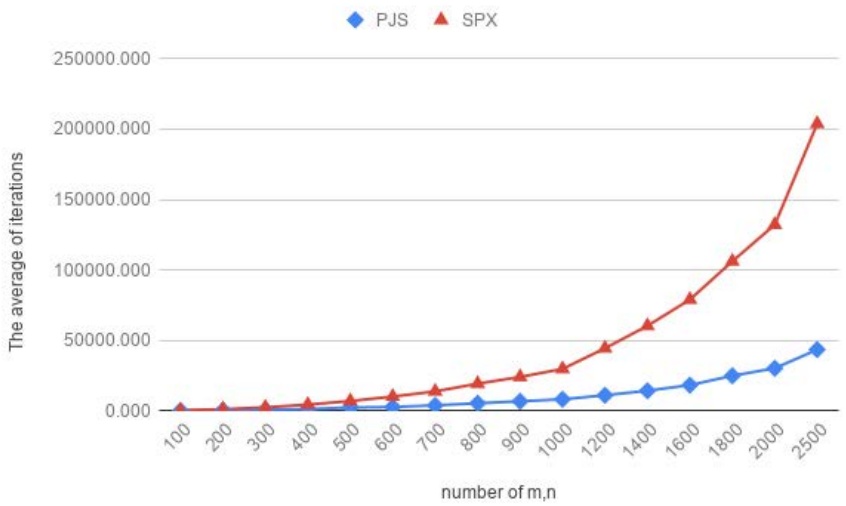

Figure 4.1: The average number of iterations of PJS and SPX

In Figure 4.1, the graph shows the average iterations of PJS comparing with the original simplex method, SPX. In this scenario, the method saved the number of iterations more than half of SPX for the LP problems having more than 2500 variables. Figure 4.2 shows the result of the average total running time that is spent by PJS and it can save about 30% of running time of SPX. In addition, if the problem has more variables, n, than 2500 variables, it can reduce more than 70% of the number of iterations.

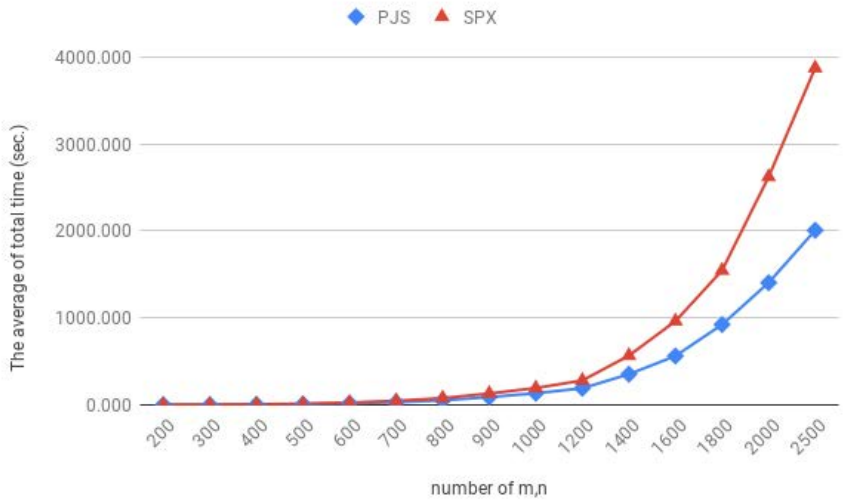

Figure 4.2: The time average of PJS and SPX

4.2 Randomly generated problems with the infeasible origin point

The synthetic LP problems in this scenario consist of $\mathbf{Ax} \oplus \mathbf{b}$ constraints $\mathbf{x} \geq \mathbf{0}$ and $\mathbf{b} \oplus \mathbf{0}$ where $\oplus$ is one of $\leq, =, \geq$. The number of variables and constaints are set to the same value n varying from $n = 100$ to $n = 1000$. Each of the results in Table 4.2 expresses as the average of iterations and the average total running time from 10 randomly LP problems. The randomly generated LP coefficients are created according to the following criteria, $c_i, a_{ij} \in [-9, 9]$ and $x_j \in [0, 9]$. The vector $\mathbf{b}$ is computed by $\mathbf{b} = \mathbf{Ax}$. Then the components of vector $\mathbf{b}$ are separated into two cases that is $b_i \geq 0$ for some $i \in \{1, 2, ..., m\}$ and $b_j < 0$ for some $j \in \{1, 2, ..., m\}$. For i^{th} component of $b_i \geq 0$, the $\mathbf{A}_i\mathbf{x} = b_i$ is converted to $\mathbf{A}_i\mathbf{x} \leq b_i$ by adding 5 to b_i. If $b_j < 0$ for some $j \in \{1, 2, ..., m\}$, the coefficients of $\mathbf{A}_j$ and b_j are converted by $-1 * \mathbf{A}_j\mathbf{x}$ and $-1 * b_j$, respectively. These constraints are denoted by $\mathbf{A}_j\mathbf{x} \geq b_j$ for some $j \in \{1, 2, ..., m\}$. Hence the vector $\mathbf{b} \geq \mathbf{0}$ and the problem contain the inequality. Table 4.2 shows the experimental results of both methods in three main columns such that the iteration, time (sec.) and the improvement (%). Each column of the table is separate into two subcolumns which contains the average number of iterations and the average total running of both methods. In addition to the running-time column, the results are shown by the jumping time of the jump process and

the total running time of PJS. The last two columns of the table show the improvement of PJS for saving the number of simplex iterations and the total running time.

(m, n)	Iterations		Time (sec.)			Improvement (%)	
	PJS	two phase SPX	PJS		two phase SPX	Iterations	Time
			Jump	Total	Total		
100	367.800	1008.200	0.052	0.347	0.640	63.519	45.768
200	1026.300	4446.600	0.255	1.822	5.809	76.919	68.631
300	2141.600	10806.100	0.717	5.974	23.704	80.182	74.798
400	1879.600	20009.300	1.575	9.219	64.151	90.606	85.629
500	756.500	31549.500	3.285	9.708	141.146	97.602	93.122
600	196.900	48011.333	5.867	12.092	332.921	99.590	96.368
700	1038.400	66632.333	10.306	33.809	1003.971	98.442	96.632
800	80.500	85628.000	17.382	33.068	2578.906	99.906	98.718
900	176.100	111799.000	25.339	47.995	4769.367	99.842	98.994
1000	63.667	138894.000	41.221	63.911	6732.708	99.954	99.051

Table 4.2: The results of randomly generated problems with the infeasible origin point

The table above shows the average number of iterations and the total running time. In addition, the last column shows the improvement of the average iterations and the average total running time. The improvement is computed by

$$\text{Improvement of time} = \left(\frac{\text{two phase SPX time} - \text{PJS time}}{\text{two phase SPX time}}\right) \times 100,$$

and

$$\text{Improvement of iterations} = \left(\frac{\text{two phase SPX iterations} - \text{PJS iterations}}{\text{two phase SPX iterations}}\right) \times 100.$$

Data from Table 4.2 is plotted as Figure 4.3 and Figure 4.4.

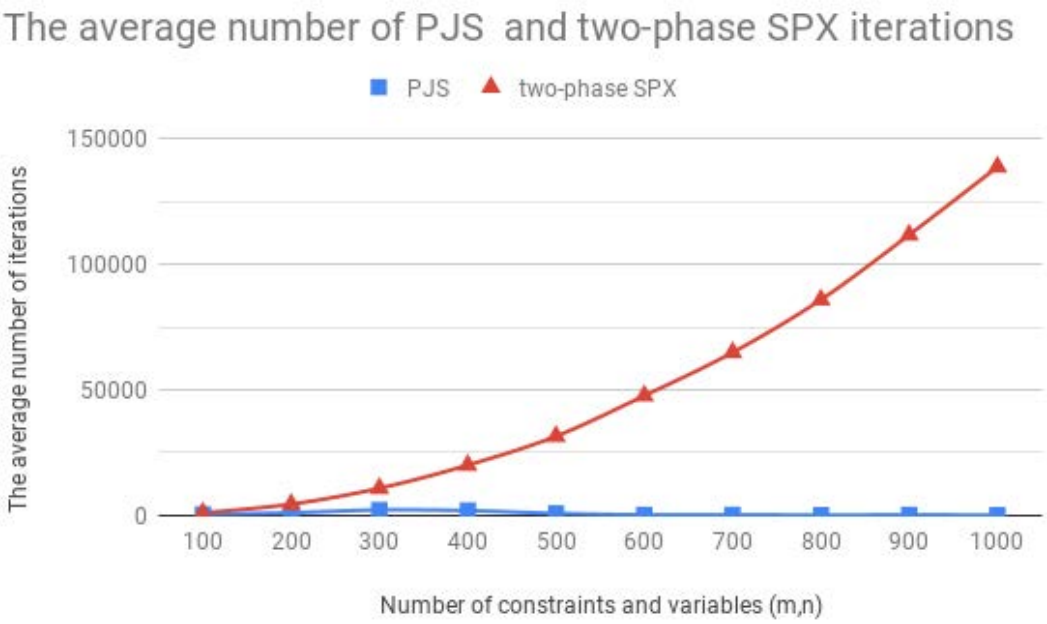

Figure 4.3: The average number of iterations of PJS and two phase SPX

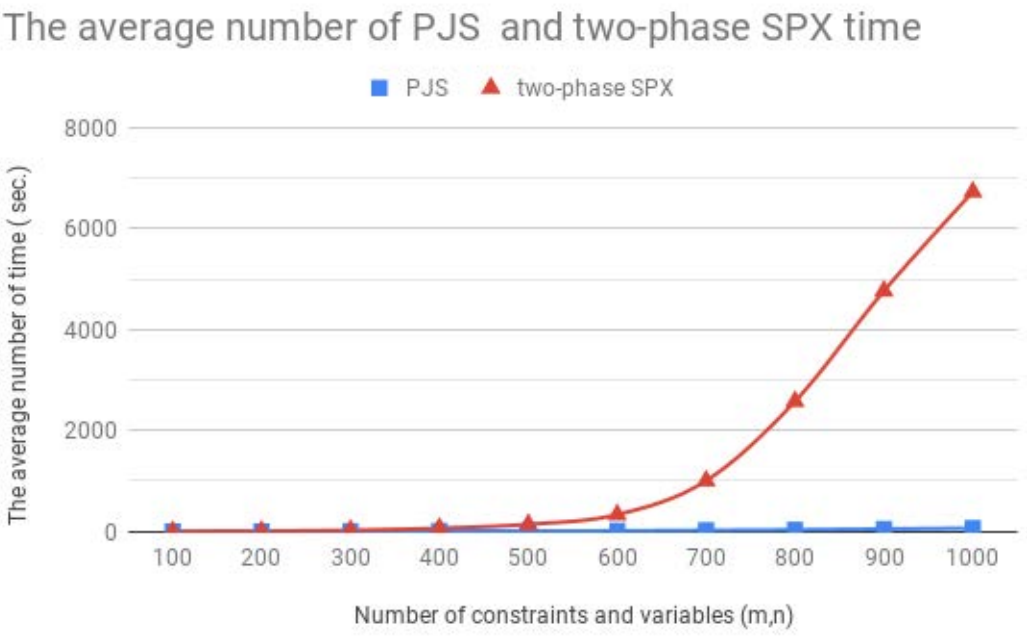

Figure 4.4: The time average of PJS and two phase SPX in general LP problems

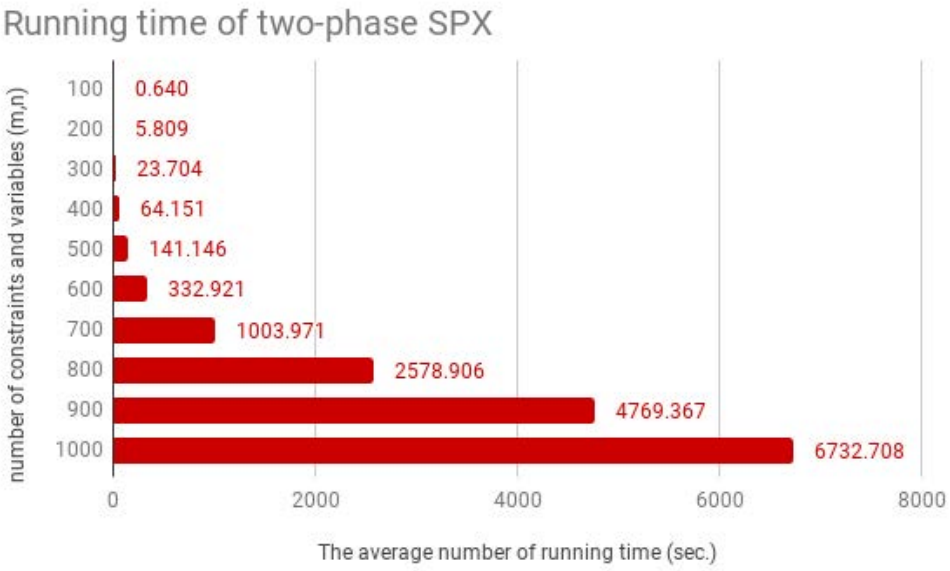

Figure 4.5: The average running time of two phase SPX

Figure 4.5 shows the average of the two-phase simplex running time from 100 to 1000 variables then the graph were compared with the total PJS running time in the next figure. In Figure 4.6, the running time of PJS method is shown into two parts: the time of the jumps and the time of performing the simplex iterations.

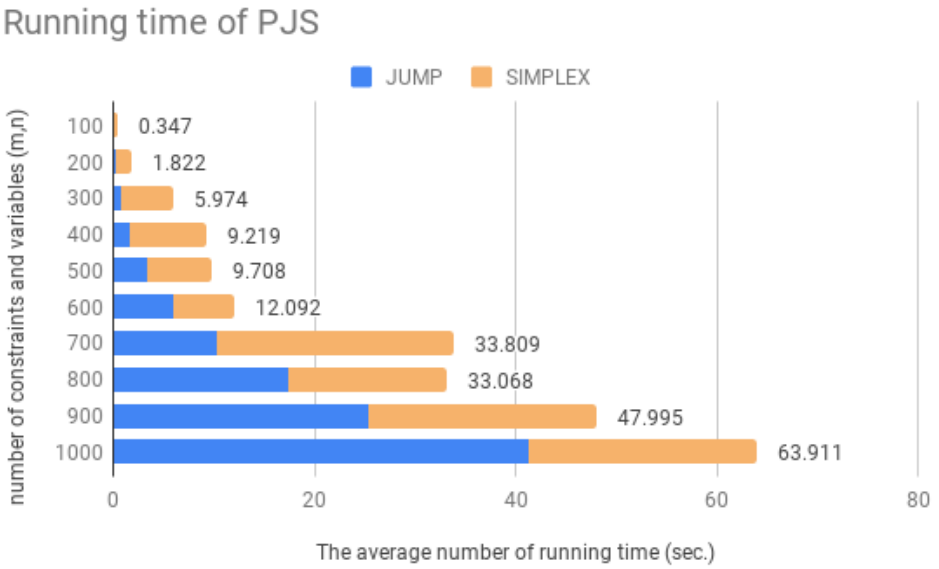

Figure 4.6: The average running time of PJS

It is clear that PJS method is able to save the running time more than the two-phase simplex method. Especially in the problems with 1000 variables, PJS can save the total running time more than 100 times of the two-phase simplex method.

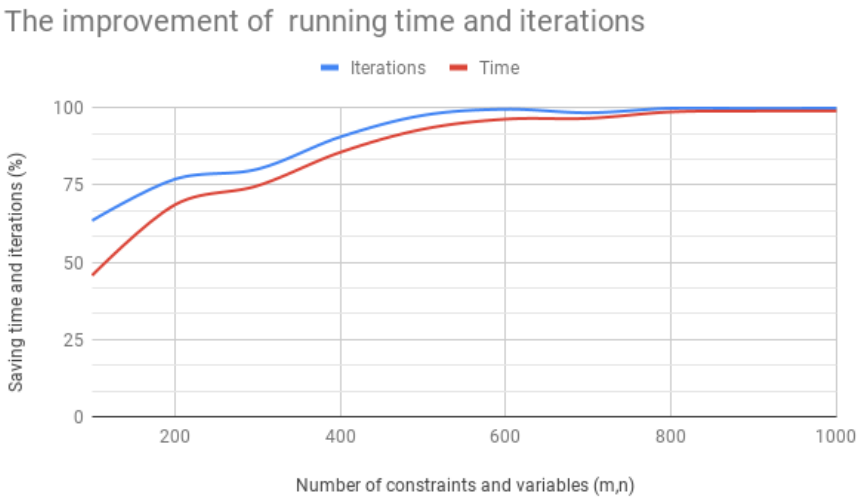

Figure 4.7: The improvement of PJS method

Figure 4.7 shows the improvements of PJS method in terms of the saving time percentage and the saving iteration percentage. For 100 variables, PJS method saved the average total running time about 60% and reduced the number of iterations about 48.95%. The graph continuously increases until 600 variables and then the graph is absolutely improved to 99% of the saving time and iterations.

CHAPTER V

CONCLUSIONS AND FUTURE WORK

The preceding-jump process is used to reduce the number of iterations of the simplex method and the total running time of solving LP problems. This process is separated into two phases. In the initial jump phase, the feasible starting point is shifted to the initial jump point. The jump point will be continuously shifted to the vertex using the jump-to-vertex phase. After that, the simplex method will start at the vertex searching for the optimal solution.

5.1 Conclusions

As a result of the preceding-jump simplex method (PJS) and the traditional simplex method (SPX), PJS outperforms the simplex method (SPX) and the two-phase simplex method (two-phase SPX). The computational experiments are separated into two scenarios, the first one uses PJS without any artificial technique. PJS can save about 30% on average total running time of SPX. Moreover, if the LP problems have the number of variables more than 2500 then the average total running time of solving the LP problems are reduced more than 50%. For SPX iterations, the average of iterations of SPX can be reduced more than 70% if the LP problems have the number of variables more than 2500 variables. The second scenario, PJS is extended to solve the general LP problems by applying the phase I of two-phase SPX. Both methods are also measured by the number of iterations and the total running time that PJS absolutely saves the iterations and the total running time more than 90%. Hence, PJS is significantly faster than SPX and two-phase SPX, especially the LP problems size bigger than 2,500 variables.

5.2 Future work

This thesis is the first to implement PJS on the randomly generated problems with equal number of variables and constrains. Further work is needed to test when the number of constraints is more than the number of variables. Moreover, it will be more practical to test this method on the LP prolems from nelib.

In the general LP problem, PJS still requires to perform phase I of the two-phase simplex method to determine the first feasible point before it starts the method. It may speed up computation time if phase I is not used. Then the next goal is to develop the preceding-jump simplex method to solve that problem without using phase I. Moreover, PJS may not need the simplex method to solve the LP problem. A user should be able to use other methods after finding the last jump point.

REFERENCES

[1] M. S. Bazaraa, J. J. Jarvis, and H. D. Sherali. *Linear Programming and Network Flows*. Wiley-Interscience, New York, NY, USA, 2004.

[2] D G Chetwynd. Applications of linear programming to engineering metrology. *Proceedings of the Institution of Mechanical Engineers, Part B: Management and engineering manufacture*, 199(2):93–100, 1985.

[3] H.W. Corley, J. Rosenberger, W.-C. Yeh, and T.K. Sung. The cosine simplex algorithm. *The International Journal of Advanced Manufacturing Technology*, 27(9): 1047–1050, 02 2006.

[4] G. B. Dantzig. *Linear Programming and Extensions*. Princeton University Press, Princeton, NJ, 1963.

[5] G. B. Dantzig. *Reminiscences About the Origins of Linear Programming*, pages 78–86. Springer Berlin Heidelberg, Berlin, Heidelberg, 1983.

[6] Ajibode Ilesanmi, FAGOYINBO I.S., AKINBO R.Y., and OLANIRAN Y.O.A. Maximization of profit in manufacturing industries using linear programming techniques: Geepee nigeria limited. 09 2011.

[7] Raluca Ion and Adrian Turek Rahoveanu. Linear programming in agriculture: Case study in region of development south-mountenia. *International Journal of Sustainable Economies Management*, 1, 01 2012.

[8] Fahad Javed and Naveed Arshad. On the use of linear programming in optimizing energy costs. pages 305–310, 11 2008.

[9] Muztoba Khan. Transportation cost optimization using linear programming. 12 2014.

[10] D. Mahto. *Linear Programming (Graphical Method)*. 03 2015.

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

[12] M. Sakarovitch. *The Two Phases of the Simplex Method: Theoretical Results Proved by Application of the Simplex Method*, pages 95–108. Springer New York, New York, NY, 1983.

[13] T. Terlaky and E. Klafszky. Some generalizations of the criss-cross method for quadratic programming. *Optimization*, 24, 09 1999.

[14] M. Tipawanna and K. Sinapiromsaran. Max-out-in pivot rule with dantzig's safeguarding rule for the simplex method. *Journal of Physics: Conference Series*, 490, 02 2014.

[15] Anatoly Vershik. L.V. kantorovich and linear programming. 08 2007.

[16] N. Yawila, B. Intiyot, and K. Sinapiromsaran. Simplex method with objective jump. *International Conference on Applied Statistics*, 2016.

APPENDIX

(m,n)	No. problems	PJS						two phase SPX			
		Time(sec.)			Iterations		Initial direction	Time (sec.)		Iterations	
		Phase1	Jump	Total	phase I	SPX		Phase I	Total	Phase I	phase II
100	1	0.063	0.070	0.391	155.000	399.000	d_0'	0.059	0.693	155.000	1111.000
	2	0.057	0.054	0.298	144.000	257.000	d_0'	0.055	0.654	144.000	1030.000
	3	0.062	0.054	0.342	164.000	295.000	d_0'	0.070	0.816	164.000	1120.000
	4	0.052	0.055	0.299	135.000	344.000	d_0'	0.052	0.651	135.000	990.000
	5	0.057	0.050	0.390	141.000	457.000	d_0'	0.050	0.571	141.000	914.000
	6	0.055	0.047	0.390	152.000	464.000	d_0'	0.055	0.608	152.000	981.000
	7	0.050	0.047	0.294	141.000	269.000	d_0'	0.051	0.612	141.000	989.000
	8	0.056	0.047	0.344	153.000	379.000	d_0'	0.054	0.609	153.000	986.000
	9	0.048	0.051	0.350	129.000	388.000	d_0'	0.047	0.616	129.000	1018.000
	10	0.052	0.047	0.372	143.000	426.000	d_0'	0.051	0.573	143.000	943.000
	average	0.055	0.052	0.347	145.700	367.800		0.055	0.640	145.700	1008.200
	S.D.	0.005	0.007	0.039	10.339	74.543		0.006	0.072	10.339	65.571
200	1	0.348	0.289	2.020	355.000	958.000	d_0'	0.346	6.542	355.000	4417.000
	2	0.289	0.253	1.955	350.000	1089.000	d_0'	0.302	5.753	350.000	4356.000
	3	0.310	0.254	1.991	353.000	1109.000	d_0'	0.310	5.682	353.000	4298.000
	4	0.274	0.252	1.768	336.000	1029.000	d_0'	0.268	5.445	336.000	4378.000
	5	0.290	0.232	1.818	352.000	1072.000	d_0'	0.278	5.976	352.000	4440.000
	6	0.333	0.300	2.222	347.000	1131.000	d_0'	0.340	6.172	347.000	4483.000
	7	0.295	0.230	1.564	368.000	849.000	d_0'	0.295	5.864	368.000	4732.000
	8	0.277	0.248	1.848	351.000	1094.000	d_0'	0.279	5.742	351.000	4616.000
	9	0.292	0.241	1.769	364.000	1031.000	d_0'	0.290	5.286	364.000	4245.000
	10	0.274	0.245	1.268	349.000	901.000	d_0'	0.284	5.625	349.000	4501.000
	average	0.298	0.255	1.822	352.500	1026.300		0.299	5.809	352.500	4446.600
	S.D.	0.025	0.023	0.264	8.835	94.419		0.026	0.360	8.835	145.676

(m,n)	No. problems	PJS						two phase SPX			
		Time(sec.)			Iterations		Initial direction	Time (sec.)		Iterations	
		Phase1	Jump	Total	phase I	SPX		Phase I	Total	Phase I	phase II
300	1	0.843	0.759	7.157	605.000	2607.000	d_0'	0.831	24.639	605.000	11070.000
	2	0.909	0.729	6.978	685.000	2493.000	d_0'	0.946	24.638	685.000	10482.000
	3	0.790	0.735	5.317	586.000	1789.000	d_0'	0.793	24.962	586.000	11268.000
	4	0.779	0.684	6.415	596.000	2483.000	d_0'	0.768	23.082	596.000	10660.000
	5	0.945	0.866	6.377	688.000	1915.000	d_0'	1.077	24.013	688.000	11166.000
	6	0.781	0.656	1.731	603.000	124.000	d_0	0.777	22.386	603.000	10724.000
	7	0.805	0.685	6.462	629.000	2503.000	d_0'	0.801	23.772	629.000	10833.000
	8	0.758	0.694	5.664	590.000	2103.000	d_0'	0.742	21.615	590.000	10357.000
	9	0.745	0.683	6.875	587.000	2725.000	d_0'	0.751	23.749	587.000	10582.000
	10	0.735	0.677	6.760	578.000	2674.000	d_0'	0.811	24.178	578.000	10919.000
	average	0.809	0.717	5.974	614.700	2141.600		0.830	23.704	614.700	10806.100
	S.D.	0.070	0.061	1.597	40.343	778.811		0.104	1.062	40.343	300.191
400	1	1.692	1.717	14.011	877.000	3230.000	d_0'	1.709	65.341	877.000	19854.000
	2	1.570	1.579	3.646	816.000	139.000	d_0	1.542	66.353	816.000	19716.000
	3	1.737	1.585	4.165	902.000	246.000	d_0	1.737	66.877	902.000	20556.000
	4	1.629	1.592	14.419	884.000	3603.000	d_0'	1.612	60.843	884.000	19188.000
	5	1.642	1.524	3.608	884.000	127.000	d_0	1.642	63.433	884.000	19857.000
	6	1.610	1.561	15.428	871.000	3990.000	d_0'	1.613	62.463	871.000	19865.000
	7	1.726	1.567	14.636	942.000	3655.000	d_0'	1.724	65.775	942.000	20386.000
	8	1.589	1.548	3.470	863.000	91.000	d_0	1.587	64.568	863.000	20514.000
	9	1.686	1.575	15.273	924.000	3610.000	d_0'	1.680	64.240	924.000	20351.000
	10	1.642	1.508	3.535	901.000	105.000	d_0	1.652	61.619	901.000	19806.000
	average	1.652	1.575	9.219	886.400	1879.600		1.650	64.151	886.400	20009.300
	S.D.	0.057	0.056	5.850	34.645	1841.257		0.063	2.034	34.645	432.344

(m,n)	No. problems	PJS						two phase SPX			
		Time(sec.)			Iterations		Initial direction	Time (sec.)		Iterations	
		Phase1	Jump	Total	phase I	SPX		Phase I	Total	Phase I	phase II
500	1	3.314	3.208	7.139	1295.000	131.000	d_0	3.314	146.583	1295.000	32593.000
	2	3.195	3.196	7.119	1253.000	158.000	d_0	3.163	142.741	1253.000	31291.000
	3	3.326	3.262	31.419	1318.000	5842.000	d_0'	3.311	144.378	1318.000	32179.000
	4	3.034	3.190	7.170	1213.000	211.000	d_0	2.980	143.140	1213.000	32341.000
	5	3.325	3.180	7.391	1188.000	198.000	d_0	2.957	134.950	1188.000	30743.000
	6	3.098	3.937	8.509	1263.000	270.000	d_0	3.720	142.361	1263.000	31595.000
	7	2.973	3.349	7.249	1213.000	174.000	d_0	3.537	135.773	1213.000	30616.000
	8	3.004	3.176	6.820	1207.000	136.000	d_0	3.006	145.347	1207.000	32715.000
	9	3.064	3.191	7.533	1242.000	290.000	d_0	3.052	133.524	1242.000	30121.000
	10	2.851	3.162	6.733	1147.000	155.000	d_0	2.826	142.666	1147.000	31301.000
	average	3.118	3.285	9.708	1233.900	756.500		3.186	141.146	1233.900	31549.500
	S.D.	0.165	0.235	7.644	50.924	1787.664		0.282	4.634	50.924	892.135
600	1	4.727	5.751	12.152	1468.000	210.000	d_0	4.685	356.490	1468.000	47662.000
	2	5.304	5.731	12.450	1522.000	188.000	d_0	4.863	354.404	1522.000	48448.000
	3	4.796	5.854	13.305	1509.000	336.000	d_0	4.766	370.014	1509.000	47924.000
	4	5.661	5.806	13.026	1642.000	226.000	d_0	4.879	334.665	1642.000	49333.000
	5	4.455	5.726	11.184	1529.000	135.000	d_0	4.472	310.199	1529.000	45935.000
	6	4.245	5.675	10.800	1459.000	128.000	d_0	4.331	317.823	1459.000	48661.000
	7	4.891	5.865	11.992	1683.000	173.000	d_0	5.201	308.138	1683.000	46626.000
	8	4.319	6.365	12.946	1488.000	301.000	d_0	5.090	328.192	1488.000	47056.000
	9	4.846	6.067	11.495	1680.000	74.000	d_0	4.886	316.535	1680.000	45900.000
	10	4.412	5.825	11.566	1528.000	198.000	d_0	4.412	332.750	1528.000	49329.000
	average	4.766	5.867	12.092	1550.800	196.900		4.759	332.921	1550.800	47687.400
	S.D.	0.448	0.206	0.839	85.170	78.556		0.286	21.201	85.170	1281.355

(m,n)	No. problems	PJS						two phase SPX			
		Time(sec.)			Iterations		Initial direction	Time (sec.)		Iterations	
		Phase1	Jump	Total	phase I	SPX		Phase I	Total	Phase I	phase II
700	1	8.327	9.777	23.364	2014.000	341.000	d_0	7.957	999.221	2014.000	63191.000
	2	7.981	11.574	20.743	1873.000	69.000	d_0	7.391	1040.541	1873.000	66863.000
	3	7.389	11.528	22.581	1800.000	238.000	d_0	7.091	1120.498	1800.000	64748.000
	4	7.778	9.869	19.726	1999.000	128.000	d_0	7.844	1029.734	1999.000	65753.000
	5	7.768	10.397	20.563	1912.000	135.000	d_0	8.326	1032.925	1912.000	66219.000
	6	8.477	9.655	20.676	1867.000	168.000	d_0	7.405	1011.362	1867.000	65396.000
	7	9.158	11.026	22.742	1983.000	138.000	d_0	8.125	977.702	1983.000	63983.000
	8	6.830	9.507	20.294	1902.000	228.000	d_0	6.989	952.830	1902.000	65252.000
	9	6.790	10.090	19.345	1866.000	165.000	d_0	7.274	932.438	1866.000	64169.000
	10	6.780	9.632	148.060	1862.000	8774.000	d_0'	6.732	942.454	1862.000	63814.000
	average	7.728	10.306	33.809	1907.800	1038.400		7.513	1003.971	1907.800	64938.800
	S.D.	0.798	0.793	40.166	69.637	2719.050		0.526	56.472	69.637	1163.044
800	1	11.664	17.271	29.039	2303.000	0.000	d_0	11.447	2821.391	2303.000	86899.000
	2	12.189	16.752	29.038	2426.000	0.000	d_0	11.486	2713.168	2426.000	84814.000
	3	10.848	16.283	27.232	2265.000	0.000	d_0	10.832	2757.333	2265.000	85171.000
	4	15.945	19.394	40.561	2348.000	173.000	d_0	15.581	2431.543	2348.000	88882.000
	5	14.869	17.093	34.183	2450.000	84.000	d_0	14.731	2097.580	2450.000	85288.000
	6	11.635	16.401	28.137	2249.000	0.000	d_0	10.535	2665.978	2249.000	83379.000
	7	10.988	15.885	30.891	2334.000	137.000	d_0	12.090	2623.907	2334.000	88393.000
	8	15.552	20.363	39.282	2248.000	119.000	d_0	15.436	2332.834	2248.000	83644.000
	9	14.949	18.016	37.187	2389.000	151.000	d_0	15.740	2686.879	2389.000	87052.000
	10	13.875	16.361	35.132	2635.000	141.000	d_0	14.145	2658.449	2635.000	86037.000
	average	13.251	17.382	33.068	2364.700	80.500		13.202	2578.906	2364.700	85955.900
	S.D.	1.989	1.464	4.863	118.631	72.873		2.116	223.656	118.631	1854.435

(m,n)	No. problems	PJS						two phase SPX			
		Time(sec.)			Iterations		Initial direction	Time (sec.)		Iterations	
		Phase1	Jump	Total	phase I	SPX		Phase I	Total	Phase I	phase II
900	1	15.573	27.957	48.025	2791.000	103.000	d_0	15.734	4845.705	2791.000	114803.000
	2	15.379	27.740	57.801	2753.000	351.000	d_0	16.055	4900.313	2753.000	116198.000
	3	14.647	25.288	45.450	2656.000	119.000	d_0	15.935	4634.840	2656.000	109603.000
	4	14.060	25.684	53.209	2566.000	326.000	d_0	14.065	4719.611	2566.000	109528.000
	5	14.567	23.998	52.950	2661.000	332.000	d_0	15.715	4908.506	2661.000	114838.000
	6	14.961	23.795	45.573	2720.000	158.000	d_0	15.346	4649.021	2720.000	109136.000
	7	15.385	23.908	47.322	2788.000	163.000	d_0	16.272	4772.700	2788.000	111168.000
	8	15.853	27.676	52.180	2882.000	209.000	d_0	15.939	4744.860	2882.000	111274.000
	9	15.231	23.879	39.235	2757.000	0.000	d_0	16.694	4684.147	2757.000	108861.000
	10	14.620	23.461	38.203	2729.000	0.000	d_0	14.128	4833.966	2729.000	112581.000
	average	15.028	25.339	47.995	2730.300	176.100		15.588	4769.367	2730.300	111799.000
	S.D.	0.551	1.828	6.243	87.356	128.916		0.863	99.687	87.356	2679.123
1000	1	17.993	43.631	61.814	2902.000	0.000	d_0	18.159	6620.545	2902.000	135949.000
	2	19.973	39.900	69.238	3053.000	191.000	d_0	19.964	6808.821	3053.000	140465.000
	3	20.363	40.130	60.681	3292.000	0.000	d_0	21.792	6768.758	3292.000	140268.000
	4	19.562	38.968	65.508	2882.000	136.000	d_0	17.881	7256.058	2882.000	140022.000
	5	22.847	39.631	62.658	3165.000	0.000	d_0	20.919	7125.823	3165.000	135636.000
	6	24.367	41.479	71.580	3379.000	106.000	d_0	22.842	7370.465	3379.000	135527.000
	7	19.262	38.986	58.418	2930.000	0.000	d_0	18.021	6940.086	2930	138687
	8	22.539	37.049	59.760	3218.000	0.000	d_0	20.487	7141.803	3218	140161
	9	20.187	41.956	62.334	3121.000	0.000	d_0	21.076	7253.590	3121	140316
	10	22.369	39.826	78.841	3224.000	326.000	d_0	21.725	6722.600	3224	137859
	average	20.946	40.156	65.083	3116.600	75.900		20.287	7000.855	3116.600	138489.000
	S.D.	1.978	1.824	6.370	171.189	112.980		1.747	261.958	171.189	2091.086

(m,n)	No. problems	PJS						two phase SPX			
		Time(sec.)			Iterations		Initial direction	Time (sec.)		Iterations	
		Phase1	Jump	Total	phase I	SPX		Phase I	Total	Phase I	phase II
900	1	15.573	27.957	48.025	2791.000	103.000	d_0	15.734	4845.705	2791.000	114803.000
	2	15.379	27.740	57.801	2753.000	351.000	d_0	16.055	4900.313	2753.000	116198.000
	3	14.647	25.288	45.450	2656.000	119.000	d_0	15.935	4634.840	2656.000	109603.000
	4	14.060	25.684	53.209	2566.000	326.000	d_0	14.065	4719.611	2566.000	109528.000
	5	14.567	23.998	52.950	2661.000	332.000	d_0	15.715	4908.506	2661.000	114838.000
	6	14.961	23.795	45.573	2720.000	158.000	d_0	15.346	4649.021	2720.000	109136.000
	7	15.385	23.908	47.322	2788.000	163.000	d_0	16.272	4772.700	2788.000	111168.000
	8	15.853	27.676	52.180	2882.000	209.000	d_0	15.939	4744.860	2882.000	111274.000
	9	15.231	23.879	39.235	2757.000	0.000	d_0	16.694	4684.147	2757.000	108861.000
	10	14.620	23.461	38.203	2729.000	0.000	d_0	14.128	4833.966	2729.000	112581.000
	average	15.028	25.339	47.995	2730.300	176.100		15.588	4769.367	2730.300	111799.000
	S.D.	0.551	1.828	6.243	87.356	128.916		0.863	99.687	87.356	2679.123
1000	1	17.993	43.631	61.814	2902.000	0.000	d_0	18.159	6620.545	2902.000	135949.000
	2	19.973	39.900	69.238	3053.000	191.000	d_0	19.964	6808.821	3053.000	140465.000
	3	20.363	40.130	60.681	3292.000	0.000	d_0	21.792	6768.758	3292.000	140268.000
	4	19.562	38.968	65.508	2882.000	136.000	d_0	17.881	7256.058	2882.000	140022.000
	5	22.847	39.631	62.658	3165.000	0.000	d_0	20.919	7125.823	3165.000	135636.000
	6	24.367	41.479	71.580	3379.000	106.000	d_0	22.842	7370.465	3379.000	135527.000
	7	19.262	38.986	58.418	2930.000	0.000	d_0	18.021	6940.086	2930	138687
	8	22.539	37.049	59.760	3218.000	0.000	d_0	20.487	7141.803	3218	140161
	9	20.187	41.956	62.334	3121.000	0.000	d_0	21.076	7253.590	3121	140316
	10	22.369	39.826	78.841	3224.000	326.000	d_0	21.725	6722.600	3224	137859
	average	20.946	40.156	65.083	3116.600	75.900		20.287	7000.855	3116.600	138489.000
	S.D.	1.978	1.824	6.370	171.189	112.980		1.747	261.958	171.189	2091.086

BIOGRAPHY

Name	Mr Natdanai Kafakthong
Date of Birth	19 July 1994
Place of Birth	Tak, Thailand
Education	B.S. (Mathematics), Naresuan University, 2016