วิธีซิมเพล็กซ์แบบกระโดดก่อน

นายณัฐดนัย กาฝากทอง

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิตสาขาวิชาคณิตศาสตร์ประยุกต์และวิทยาการคอมฯ ภาควิชาคณิตศาสตร์และวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์ จุฬาลงกรณ์มหาวิทยาลัยปีการศึกษา 2561

ลิขสิทธิ์ของจุฬาลงกรณ์มหาวิทยาลัย

บทคัดย่อและแฟ้มข้อมูลฉบับเต็มของวิทยานิพนธ์ตั้งแต่ปีการศึกษา 2554 ที่ให้บริการในคลังปัญญาจุฬาฯ (CUIR) เป็นไฟล์ข้อมูลของนิสิตเจ้าของวิทยานิพนธ์ที่ส่งผ่านทางบัณฑิตวิทยาลัย

The abstract and full text of theses from the academic year 2011 in Chulalongkorn University Intellectual Repository(CUIR) are the thesis authors' files submitted through the Graduate School.
PRECEDING-JUMP SIMPLEX METHOD

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

Copyright of Chulalongkorn University
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.)
ซอฟต์แวร์แก้ปัญหากำหนดการเชิงเส้นในเชิงปฏิบัติใช้วิธีซิมเพล็กซ์เพื่อระบุผลเฉลยที่เหมาะสมที่สุดโดยการดังนี้:

- ต้นแบบจากรูปที่ 12 ของขั้นตอนการกระโดดไปยังจุดที่เหมาะสมที่สุดก่อนใช้วิธีซิมเพล็กซ์. จุดยอดใหม่จะถูกระบุด้วยการกระโดดไปจุดยอดใกล้กับจุดยอดที่เหมาะสมที่สุดก่อนใช้วิธีซิมเพล็กซ์.
- ขั้นตอนนี้จะดำเนินการกระโดดคงที่ตามทิศทางของเวกเตอร์เกรดิเอนต์ของฟังก์ชันจุดประสงค์หรือทิศทางเลือกอื่นในกรณีที่ทิศทางของฟังก์ชันจุดประสงค์ชี้ออกจากบริเวณที่เป็นไปได้.
- จุดยอดดังกล่าวจะมีจุดที่เดิมที่เป็นไปได้ในจุดแรกถูกเลื่อนไปยังจุดที่เหมาะสมที่สุดก่อน.
- จุดยอดดังกล่าวจะถูกเลื่อนไปเรื่อยๆจนถึงจุดที่เหมาะสมที่สุดก่อน.
- จุดยอดดังกล่าวจะถูกเลื่อนไปเรื่อยๆจนถึงจุดที่เหมาะสมที่สุดก่อน.
- จุดยอดดังกล่าวจะถูกเลื่อนไปเรื่อยๆจนถึงจุดที่เหมาะสมที่สุดก่อน.
- จุดยอดดังกล่าวจะถูกเลื่อนไปเรื่อยๆจนถึงจุดที่เหมาะสมที่สุดก่อน.
- จุดยอดดังกล่าวจะถูกเลื่อนไปเรื่อยๆจนถึงจุดที่เหมาะสมที่สุดก่อน.
- จุดยอดดังกล่าวจะถูกเลื่อนไปเรื่อยๆจนถึงจุดที่เหมาะสมที่สุดก่อน.
- จุดยอดดังกล่าวจะถูกเลื่อนไปเรื่อยๆจนถึงจุดที่เหมาะสมที่สุดก่อน.
- จุดยอดดังกล่าวจะถูกเลื่อนไปเรื่อยๆจนถึงจุดที่เหมาะสมที่สุดก่อน.
- จุดยอดดังกล่าวจะถูกเลื่อนไปเรื่อยๆจนถึงจุดที่เหมาะสมที่สุดก่อน.
- จุดยอดดังกล่า
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.
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

ABSTRACT IN THAI ........................................ iv
ABSTRACT IN ENGLISH ..................................... v
ACKNOWLEDGEMENTS ....................................... vi
CONTENTS .................................................. vii
LIST OF TABLES ........................................... x
LIST OF FIGURES .......................................... xi

CHAPTER

1 INTRODUCTION ........................................... 1

1.1 Optimization problem in general ...................... 1
1.2 Why linear programming? ............................. 1
1.3 Linear programming model ........................... 2
1.4 The standard form of a linear programming problem ............. 4
1.5 Methods to solve a linear programming problem .......... 5
  1.5.1 The graphical method ............................. 6
  1.5.2 The simplex method ............................. 7
  1.5.3 The two-phase simplex method .................... 7
1.6 The jump method .................................. 8
  1.6.1 Objective of the thesis ......................... 9
  1.6.2 Scope of the thesis ............................ 10

2 BACKGROUND KNOWLEDGE AND LITERATURE REVIEW ... 12

2.1 Notations ........................................... 12
2.2 Basic algebra ...................................... 13
  2.2.1 Vector ........................................ 13
  2.2.2 Matrix and rank of a matrix .................. 14
  2.2.3 Inner product ................................ 14
  2.2.4 Euclidean norm ............................... 14
  2.2.5 Angle between vectors ....................... 15
2.3 Geometric definitions ................................ 15
<table>
<thead>
<tr>
<th>CHAPTER</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.4</td>
<td>The system of linear equations</td>
</tr>
<tr>
<td>2.4.1</td>
<td>Generalized inverse</td>
</tr>
<tr>
<td>2.5</td>
<td>The polyhedron theory</td>
</tr>
<tr>
<td>2.6</td>
<td>The basic feasible solution</td>
</tr>
<tr>
<td>2.7</td>
<td>The simplex algorithm</td>
</tr>
<tr>
<td>2.7.1</td>
<td>Pseudo code of the simplex algorithm</td>
</tr>
<tr>
<td>2.8</td>
<td>The two-phase simplex algorithm</td>
</tr>
<tr>
<td>2.8.1</td>
<td>Pseudo code of the two-phase simplex algorithm</td>
</tr>
<tr>
<td>2.9</td>
<td>Literature review</td>
</tr>
<tr>
<td>3</td>
<td>PRECEDING-JUMP SIMPLEX METHOD</td>
</tr>
<tr>
<td>3.1</td>
<td>The linear programming problem with the feasible origin point</td>
</tr>
<tr>
<td>3.1.1</td>
<td>The initial jump phase</td>
</tr>
<tr>
<td>3.1.1.1</td>
<td>The kinds of the initial jump</td>
</tr>
<tr>
<td>3.1.1.2</td>
<td>The search-direction process</td>
</tr>
<tr>
<td>3.1.2</td>
<td>The jump-to-vertex phase</td>
</tr>
<tr>
<td>3.1.2.1</td>
<td>The artificial hyperplane</td>
</tr>
<tr>
<td>3.2</td>
<td>The linear programming problem with the infeasible origin point out</td>
</tr>
<tr>
<td>3.2.1</td>
<td>The general preceding-jump simplex method</td>
</tr>
<tr>
<td>3.2.1.1</td>
<td>The initial jump phase</td>
</tr>
<tr>
<td>3.2.2</td>
<td>The search-direction process</td>
</tr>
<tr>
<td>3.2.2.1</td>
<td>Pseudo code of the general preceding-jump simplex algorithm</td>
</tr>
<tr>
<td>4</td>
<td>EXPERIMENTS AND RESULTS</td>
</tr>
<tr>
<td>4.1</td>
<td>Randomly generated problems with the feasible origin point</td>
</tr>
<tr>
<td>4.2</td>
<td>Randomly generated problems with the infeasible origin point</td>
</tr>
<tr>
<td>5</td>
<td>CONCLUSIONS AND FUTURE WORK</td>
</tr>
<tr>
<td>5.1</td>
<td>Conclusions</td>
</tr>
<tr>
<td>5.2</td>
<td>Future work</td>
</tr>
<tr>
<td>CHAPTER</td>
<td>Page</td>
</tr>
<tr>
<td>------------------</td>
<td>------</td>
</tr>
<tr>
<td>REFERENCES</td>
<td>72</td>
</tr>
<tr>
<td>APPENDICES</td>
<td>74</td>
</tr>
<tr>
<td>BIOGRAPHY</td>
<td>82</td>
</tr>
<tr>
<td>Table</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>--------</td>
</tr>
<tr>
<td>3.1</td>
<td>39</td>
</tr>
<tr>
<td>3.2</td>
<td>40</td>
</tr>
<tr>
<td>4.1</td>
<td>63</td>
</tr>
<tr>
<td>4.2</td>
<td>66</td>
</tr>
<tr>
<td>Table 3.1</td>
<td>The simplex tableau of the search-direction process with $x_1$ as the entering variable.</td>
</tr>
<tr>
<td>Table 3.2</td>
<td>The simplex tableau of the search-direction process with $x_2$ as the entering variable.</td>
</tr>
<tr>
<td>Table 4.1</td>
<td>The results of randomly generated problems with the origin point in the feasible region.</td>
</tr>
<tr>
<td>Table 4.2</td>
<td>The results of randomly generated problems with the infeasible origin point.</td>
</tr>
</tbody>
</table>
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Solving the linear programming problem by the graphical method</td>
<td>6</td>
</tr>
<tr>
<td>1.2</td>
<td>The initial jump point</td>
<td>9</td>
</tr>
<tr>
<td>2.1</td>
<td>Basic feasible solution of the example 1</td>
<td>21</td>
</tr>
<tr>
<td>3.1</td>
<td>The initial jump phase</td>
<td>32</td>
</tr>
<tr>
<td>3.2</td>
<td>The initial jump point on the bounded feasible region</td>
<td>34</td>
</tr>
<tr>
<td>3.3</td>
<td>Jump into the unbounded feasible region</td>
<td>35</td>
</tr>
<tr>
<td>3.4</td>
<td>The jump point in the infeasible region</td>
<td>36</td>
</tr>
<tr>
<td>3.5</td>
<td>The basic infeasible solution</td>
<td>36</td>
</tr>
<tr>
<td>3.6</td>
<td>The new direction $d'_0$ from the search-direction process</td>
<td>40</td>
</tr>
<tr>
<td>3.7</td>
<td>The jump-to-vertex phase</td>
<td>43</td>
</tr>
<tr>
<td>3.8</td>
<td>Creating direction in the intersecting hyperplane process</td>
<td>45</td>
</tr>
<tr>
<td>3.9</td>
<td>The feasible region of example 3.3</td>
<td>46</td>
</tr>
<tr>
<td>3.10</td>
<td>The initial jump phase of example 3.3</td>
<td>47</td>
</tr>
<tr>
<td>3.11</td>
<td>The jump-to-vertex phase of example 3.3</td>
<td>48</td>
</tr>
<tr>
<td>3.12</td>
<td>Jump into the infeasible region</td>
<td>51</td>
</tr>
<tr>
<td>3.13</td>
<td>The initial jump phase of example 3.4</td>
<td>53</td>
</tr>
<tr>
<td>3.14</td>
<td>The jump-to-vertex phase of example 3.4</td>
<td>54</td>
</tr>
<tr>
<td>3.15</td>
<td>Jump into the infeasible region</td>
<td>56</td>
</tr>
<tr>
<td>3.16</td>
<td>New direction from the search-direction process</td>
<td>59</td>
</tr>
<tr>
<td>4.1</td>
<td>The average number of iterations of PJS and SPX</td>
<td>64</td>
</tr>
<tr>
<td>4.2</td>
<td>The time average of PJS and SPX</td>
<td>65</td>
</tr>
<tr>
<td>4.3</td>
<td>The average number of iterations of PJS and two phase SPX</td>
<td>67</td>
</tr>
<tr>
<td>4.4</td>
<td>The time average of PJS and two phase SPX in general LP problems</td>
<td>67</td>
</tr>
<tr>
<td>4.5</td>
<td>The average running time of two phase SPX</td>
<td>68</td>
</tr>
<tr>
<td>4.6</td>
<td>The average of PJS running time</td>
<td>68</td>
</tr>
<tr>
<td>4.7</td>
<td>The improvement of PJS method</td>
<td>69</td>
</tr>
</tbody>
</table>
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

\[
\begin{align*}
\text{Maximize/Minimize} & \quad z = c_1x_1 + c_2x_2 + \ldots + c_jx_j + \ldots + c_nx_n \\
\text{subject to} & \quad a_{11}x_1 + a_{12}x_2 + \ldots + a_{1j}x_j + \ldots + a_{1n}x_n \leq b_1 \\
& \quad \vdots \\
& \quad a_{i1}x_1 + a_{i2}x_2 + \ldots + a_{ij}x_j + \ldots + a_{in}x_n \leq b_i \\
& \quad \vdots \\
& \quad a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mj}x_j + \ldots + a_{mn}x_n \leq b_m \\
\end{align*}
\]

\[x_1, x_2, \ldots, x_n \geq 0\]

The problem can be written using summations as follows.

\[
\begin{align*}
\text{Maximize} & \quad z = \sum_{j=1}^{n} c_jx_j \\
\text{subject to} & \quad \sum_{j=1}^{n} a_{ij}x_j \leq b_i, \quad i = 1, 2, \ldots, m \\
& \quad x_j \geq 0, \quad j = 1, 2, \ldots, n,
\end{align*}
\]

where \( m \) = number of constraints,

\( n \) = number of decision variables,

\( a_{ij} \) = coefficient of \( i^{th} \) constraint 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.

\[
\text{Maximize } c^T x \\
\text{subject to } Ax \leq b \\
x \geq 0
\]  

(1.1)

where \( A \) = the \( m \times n \) coefficient matrix.  
\( b \) = the \( m \times 1 \) vector of right-hand sides.  
\( c \) = the \( n \times 1 \) coefficient vector of the objective function.  
\( x \) = 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.

\[
\text{Maximize } z = c_1 x_1 + c_2 x_2 + \ldots + c_{m+n} x_{m+n} \\
\text{subject to } a_{11} x_1 + a_{12} x_2 + \ldots + a_{1(m+n)} x_{m+n} = b_1 \\
a_{21} x_1 + a_{22} x_2 + \ldots + a_{2(m+n)} x_{m+n} = b_2 \\
\vdots \\
a_{m1} x_1 + a_{m2} x_2 + \ldots + a_{m(m+n)} x_{m+n} = b_m \\
x_1, x_2, \ldots, x_{m+n} \geq 0
\]  

(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 $a_{i1}x_1 + a_{i2}x_2 + \ldots + a_{in}x_n \leq b_i$, convert it into an equality constraint by adding the nonnegative slack variable $s_i$ then the resulting constraint is $a_{i1}x_1 + a_{i2}x_2 + \ldots + a_{in}x_n + s_i = b_i$ where $s_i \geq 0$.
- If the constraint is $a_{i1}x_1 + a_{i2}x_2 + \ldots + a_{in}x_n \geq b_i$, convert it into an equality constraint by subtracting the nonnegative surplus variable $s_i$. The resulting constraint is $a_{i1}x_1 + a_{i2}x_2 + \ldots + a_{in}x_n - s_i = b_i$ 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

$$\begin{align*}
\text{Maximize} & \quad \sum_{j=1}^{m+n} c_j x_j \\
\text{subject to} & \quad \sum_{j=1}^{m+n} a_{ij} x_j = b_i, \quad i = 1, 2, \ldots, m \\
& \quad x_j \geq 0, \quad j = 1, 2, \ldots, m + n
\end{align*}$$

(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](image)

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. $\alpha_i = \frac{b_i}{\mathbf{A}_i^T \mathbf{c}}$ 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 \( \alpha_3 \) is equal to

\[
\alpha_3 = \frac{10}{\begin{bmatrix} 2 \\ 1 \\ 1.5 \end{bmatrix}} = 2.5.
\]

Thus, the jump point is \( x_1 = \alpha_3 c = 2.5(1.5, 1) = (3.75, 2.5) \) 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 \( c \) can be shifted to \( \alpha_{i_{min}} \cdot c \), where \( i_{min} \) is the minimum index of constraints when it hits the edge of the hyperplane having the minimum distance between \( x_0 \) and \( x_1^i, i = 1, 2, \ldots, m \).

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

\[ n = \text{the number of decision variables.} \]
\[ m = \text{the number of constraints.} \]
\[ c = \text{the gradient vector of the objective function.} \]
\[ I = \text{the set of all indices of constraints i.e. } \{1, 2, ..., m+n\}. \]
\[ A_i = \text{the } i^{th} \text{ row of the matrix } A. \]
\[ b_i = \text{the value of the right-hand side of the } i^{th} \text{ row, } i = 1, 2, ..., m+n. \]
\[ B = \text{the index set of the basic variables.} \]
\[ N = \text{the index set of the non-basic variables.} \]
\[ V = \text{the set of all indices visited constraints.} \]
\[ A_B = \text{the } m \times m \text{ invertible matrix, called the basic matrix.} \]
\[ A_V = \text{the system of linear equations from the index set } V. \]
\[ b_V = \text{the vector of the right-hand side forming from } V. \]
\( x_B \) = the basic variables.
\( x_N \) = the non-basic variables.
\( x_0 \) = the feasible starting point of the jump processes.
\( x_k \) = the feasible jump point corresponding to the \( k^{th} \) iteration.
\( d_k \) = the jump direction in the \( k^{th} \) iteration.
\( \alpha_k \) = the minimum distance between \( x_k \) 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, \( v^T = [1, 0, 0] \) is the row vector of size \( n = 3 \), and \( u^T = [1, 0] \) 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 \( 0 \). 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 \( e_i \).
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 \(I_n\).

\[
I_n = \begin{bmatrix}
1 & 0 & \ldots & 0 \\
0 & 1 & \ldots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
0 & 0 & \ldots & 1
\end{bmatrix}
\]

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 \(u\) and \(v\) 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.

\[
\mathbf{u} \cdot \mathbf{v} = u_1v_1 + u_2v_2 + \ldots + u_nv_n = \sum_{i=1}^{n} u_iv_i \tag{2.1}
\]

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 \( \mathbf{u} = (u_1, u_2, \ldots, u_n) \) and \( \mathbf{v} = (v_1, v_2, \ldots, v_n) \) 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 \( \theta \) is the angle between the two vectors. So \( \theta \) 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), \( \theta \) will be occurred 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 \( \{x_0 + \lambda d : \lambda > 0\} \), where \( d \) is the nonzero vector. Here, \( x_0 \) is called the vertex of the ray, and \( d \) is the direction of the ray.

Definition 2.3.2. Given an unbounded convex set \( X \), a non zero vector \( d \) is called the direction of \( X \) if for each \( x_0 \in X \), the ray \( \{x : x_0 + \lambda d, d \neq 0, \lambda \geq 0\} \) is contained within \( X \). Note that \( d_1 \) and \( d_2 \) are distinct if and only if there is no \( k \) that \( d_1 \neq k d_2 \).

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

\[
Ax = b
\]  

(2.3)

where \( A \) 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, \( A \) needs to be the full row rank and the square matrix. Thus, \( A^{-1} \), the inverse matrix of \( A \) exists and the unique solution is \( x = A^{-1}b \). Note that, the inverse of \( A \) is only \( A^{-1} \) when \( A^{-1} \times A = A \times A^{-1} = I \). The next case, if some inconsistencies arise between equations in \( Ax = b \), 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 \( A \).
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 $A$. If $A$ 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 $A$ has more variables than equations ($m < n$). However, the solution of $Ax = b$ 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 $A$.

Given matrix $A \in \mathbb{R}^{n \times m}$ and matrix $A^g \in \mathbb{R}^{m \times n}, A^g$ is the generalized inverse of $A$ if it satisfies the condition $AA^g A = A$. Thus the solution of the linear system with $A$ 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 $Ax = b$ where $A$ is non-square full rank so the solution may be occurred if $x = A^g b$. Thus the general inverse of $A$ will be separated into two types.

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

Right inverse : If $A$ has linearly independent rows i.e. rank $A = m$. Then the matrix $AA^T$ is the invertible $m$ by $m$ symmetric matrix. This implies that the right inverse of $A$ is $A^T (AA^T)^{-1}$. Thus $A^g = A^T (AA^T)^{-1}$ and $x = A^T (AA^T)^{-1} b$.

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 $x$ of a convex set $M$ is an extreme point or the vertex of $M$ then it is not possible to find two points $x_1, x_2$ in $M$ such that $x = \lambda x_1 + (1 - \lambda) x_2, 0 < \lambda < 1$. It means that the vertex does not lie on the line segment joining two other points in the
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 \( \{x | p^T x = k\} \) where \( p \) is the nonzero vector in \( \mathbb{R}^n \) and \( k \) is the scalar. Note \( p \) is called the normal or the gradient vector of the hyperplane. A closed half-space is a collection of points of the form \( \{x | p^T x \leq k\} \), where \( p \) 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 \( x \in X \) and suppose that the constraint \( j \) in the set of all constraints \( \{1, 2, ..., m + n\} \) is binding, or tight, or active, at \( x \) then \( A_j x = b_j \).

Moreover, the solution in \( P = \{x : Ax \leq b\} \) 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 \( x \) in convex set \( X \subseteq \mathbb{R}^n \) is called the basic feasible solution of \( X \) if \( x \) cannot be represented as the strict convex combination of two distinct points in \( X \). In other words, if \( x = \lambda x_1 + (1 - \lambda) x_2 \) with \( \lambda \in (0, 1) \) and \( x_1, x_2 \in X \), then \( x = x_1 = x_2 \). 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.

\[
Ax = b \quad \text{(2.4)}
\]

\[
x \geq 0
\]
where $A$ is the $m \times n$ matrix and $b$ is the $m$-vector. Suppose that the rank $(A) = m$ and $b$ belongs to the range that is spanned by the column space of $A$. 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 $Ax = b$. 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

\[
A_1 x_1 + A_2 x_2 + A_3 x_3 + \ldots + A_n x_n = b
\]

where $A_j, j = 1, 2, \ldots, n$ is the $j^{th}$ column of $A$. This implies that $A_j$ is in $\mathbb{R}^m$ with the full rank $A$ so after rearranging the columns of $A$ the column vectors $A_1, A_2, \ldots, A_m$ are linearly independent. Therefore, the set of $A_1, A_2, \ldots, A_m$ forms a basis. Moreover the solution of the equation above will be the $n$-vector $[x_1, x_2, \ldots, x_m, 0, 0, \ldots, 0]^T$ that corresponds to $Ax = b$. The $m$ linearly independent column vectors $A_j, j \in \{1, 2, \ldots, n\}$
will become the $m \times m$ invertible matrix $A_B$ and the remaining column vectors of $A$ denotes as $m \times (n-m)$ matrix $A_N$. Then the matrix $A$ will be expressed as $A = [A_B \ A_N]$.

The solution $x$ to $Ax = b$, where $x_B = A_B^{-1}b$ and $x_N = 0$ is called the basic solution of the system. If $x_B \geq 0$, then $x = [x_B \ x_N]$ is called the basic feasible solution of the system. Here $A_B$ is called the basic matrix (or simply the basis) and $A_N$ is called the non-basic matrix. The components of $x_B$ are called basic variables (or dependent variables) and the components of $x_N$ are called non-basic variables (or independent variables). Moreover, if all components of $x_B > 0$, then $x$ is called the nondegenerate basic feasible solution, but if at least one component of $x_B$ is zero, then $x$ is called the degenerate basic feasible solution. The example 1 shows the problem with two variables.

Maximize  

$$2x_1 + x_2$$

subject to  

$$x_1 + x_2 \leq 4$$

$$-x_1 + x_2 \leq 4$$

$$x_1 - x_2 \leq 3$$

$$x_1, x_2 \geq 0$$

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 $x_N$ and basic variables of $x_B$. The vertex $C$ can be computed by defining the matrix $A_B$ with basic variables $x_1, x_2$ and $x_4$. Now, this particular basis matrix $A_B$ is defined as follows:

$$A_B = \begin{bmatrix} 1 & 1 & 0 \\ -1 & 1 & 1 \\ 1 & -1 & 0 \end{bmatrix}$$
Then

\[ x_B = A_B^{-1}b = \begin{bmatrix} 0.5 & 1 & 0.5 \\ 0.5 & 0 & -0.5 \\ 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} 4 \\ 4 \\ 3 \end{bmatrix} = \begin{bmatrix} 3.5 \\ 0.5 \\ 7 \end{bmatrix} \]

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](image)

2.7 The simplex algorithm

From the LP model 1.1, the model contains the \( m \times n \) matrix \( A \), the \( n \times 1 \) vector \( x \geq 0 \), the \( m \times 1 \) vector \( b \geq 0 \), and the \( n \times 1 \) vector \( c \).

The model can be converted into the standard form as belows.
Maximize $\mathbf{c}^T \mathbf{x}$

subject to $\mathbf{A} \mathbf{x} + \mathbf{I} \mathbf{s} = \mathbf{b}$

$$x, s \geq 0$$

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 $\mathbf{A}$ 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 $\mathbf{A}_B$ which is the first basic matrix of the first simplex tableau and $\mathbf{A}$ is the first non-basic matrix representing by $\mathbf{A}_N$. 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.

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

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 $z_N^T$. The variable $x_k$ is called the entering variable.

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

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

1: procedure **Simplex**($c, b, A$)
2: Input: the LP problem $c, b, A$
3: Output: the optimal solution or unbounded solution
4: while True do
5:     $z_T^T \leftarrow c_T^T A_B^{-1} A_N - c_N^T$
6:     If $z_T^T \leq 0$ then STOP return the current BFS is optimal
7:     $k \leftarrow \text{argmax}\{z_N^T\}$
8:     $y_k \leftarrow A_B^{-1} A_{Nk}$
9:     If $y_k \leq 0$ then STOP return unbounded solution
10:    $r \leftarrow A_B^{-1} b$
11:    $l \leftarrow \text{argmin}\{\frac{r_i}{y_{ik}} | y_{ik} > 0, i \in B\}$
12:    $B \leftarrow (B \setminus \{l\}) \cup \{k\}$
13:    $N \leftarrow (N \setminus \{k\}) \cup \{l\}$
14: 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

\[
\text{Maximize } \sum_{j=1}^{n} c_j x_{ij} \\
\text{subject to } \sum_{j=1}^{n} a_{ij} x_{ij} \geq b_i, i = 1, 2, \ldots, m \tag{2.6}
\]

\[x_{ij} \geq 0, j = 1, 2, \ldots, n\]

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 (“ ≥ ” 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 \( b \). 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.

\[
\text{Minimize } \sum_{k=1}^{m} x_{a_k},
\]

subject to \(\sum_{j=1}^{n} a_{ij}x_j - x_{n+i} + x_{a_i} = b_i, i = 1, 2, \ldots, m,\) (2.7)

\[
x_j, x_{a_i} \geq 0, j = 1, 2, \ldots, n + m, i = 1, 2, \ldots, m
\]

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 \(j = 1, 2, \ldots, n + m\). 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.

\[
\text{Maximize } 2x_1 + x_2
\]

subject to \(x_1 + x_2 \leq 4\)

\(-x_1 + x_2 \leq 4\)

\(x_1 - x_2 \leq 3\)

\(x_1 + x_2 \geq 1\)

\(x_1, x_2 \geq 0\)

**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 \(x_a\).

Minimize \(x_a\)

subject to \(x_1 + x_2 + x_3 = 4\)

\(-x_1 + x_2 + x_4 = 4\)

\(x_1 - x_2 + x_5 = 3\)

\(x_1 + x_2 - x_6 + x_a = 1\)

\(x_1, x_2, x_3, x_4, x_5, x_6, x_a \geq 0\)

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.

Minimize \( 1 - x_1 - x_2 + x_6 \)

subject to

\[
\begin{align*}
-x_1 + x_2 + x_3 &= 4 \\
x_1 - x_2 + x_4 &= 4 \\
x_1 + x_2 - x_6 + x_a &= 1 \\
\end{align*}
\]

\( x_1, x_2, x_3, x_4, x_5, x_6, x_a \geq 0 \)

<table>
<thead>
<tr>
<th>Iter. 0</th>
<th>( x_1 )</th>
<th>( x_2 )</th>
<th>( x_3 )</th>
<th>( x_4 )</th>
<th>( x_5 )</th>
<th>( x_6 )</th>
<th>( x_a )</th>
<th>rsh</th>
</tr>
</thead>
<tbody>
<tr>
<td>( z )</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>( x_3 )</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>( x_4 )</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>( x_5 )</td>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>( x_a )</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>1</td>
</tr>
</tbody>
</table>

\( z_j - c_j \) is calculated as follows:

\[
\begin{align*}
-2 &-1 0 0 0 0 0 0 0 0
\end{align*}
\]
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.

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
1: **Input**: an LP problem \( c, b, A \).
2: **Output**: the optimal solution, unbounded solution, infeasible solution
3: **#Phase I**
4: \( c_l \leftarrow 1 \) if \( i \) is the index of the artificial variable otherwise \( 0 \)
5: **while** True **do**
6: \( z_N^T \leftarrow c_B^T A_B^{-1} A_N - c_N^T \)
7: **If** \( z_N^T \leq 0 \) **then** BREAK
8: \( k \leftarrow \argmax\{z_N^T\} \)
9: \( y_k \leftarrow A_B^{-1} A_{N_k} \)
10: **If** \( y_k \leq 0 \) **then**
11: **If** the artificial variables > 0 **then** STOP **return** infeasible solution
12: **else** STOP **return** unbounded solution
13: \( r \leftarrow A_B^{-1} b \)
14: \( l \leftarrow \argmin\{y_i^k | y_i^k > 0, i \in B\} \)
15: \( B \leftarrow (B \setminus \{l\}) \cup \{k\} \)
16: \( N \leftarrow (N \setminus \{k\}) \cup \{l\} \)
17: **If** the index of artificial variables is in \( B \) **then**
18: **If** the artificial variables == 0 **then**
19: \( l \leftarrow \) the \( i^{th} \) row of the artificial variable to be the leaving variable.
20: pivot the artificial indices out of \( B \)
21: remove the artificial variables from the simplex tableau
22: **If** the artificial variables > 0 **then** STOP
23: **return** infeasible solution
Algorithm 2 The two-phase simplex algorithm (continued)

24: \#Phase II
25: while True do
26:   $z_N^T \leftarrow c_B^T A_B^{-1} A_N - c_N^T$
27:   If $z_N^T \leq 0$ then STOP return the current BFS is the optimal solution
28:   $k \leftarrow \text{argmax}\{z_N^T\}$
29:   $y_k \leftarrow A_B^{-1} A_{N_k}$
30:   If $y_k \leq 0$ then STOP return unbounded solution
31:   $r \leftarrow A_B^{-1} b$
32:   $l \leftarrow \text{argmin}\{\frac{r}{y_{ik}} | y_{ik} > 0, i \in B\}$
33:   $B \leftarrow (B \setminus \{l\}) \cup \{k\}$
34:   $N \leftarrow (N \setminus \{k\}) \cup \{l\}$
35: 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, \( x_0 = 0 \in \mathbb{R}^n \), as the initial vertex. The LP problem contains the constraints \( A_i x_i \leq b_i, b_i \geq 0 \) for \( i = 1, 2, \ldots, m \) and \( x_j \geq 0 \) for \( j = 1, 2, \ldots, n \). So the number of all constraints is equal to \( m + n \). The point \( x_0 \) 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 \( x_0 \) to move along any direction. So the method needs to drop those hyperplanes or all hyperplanes binding at \( x_0 \), before doing the jump process. The index set of the first binding hyperplanes denoted by \( FB \). Since \( x_0 \) binds to the hyperplanes initially of the form \( x_j = 0 \) for \( j = 1, 2, \ldots, n \). Hence the set of the binding constraints, \( FB \), contains the indices of \( \{ m + 1, m + 2, \ldots, m + n \} \) which corresponds to the constraints of \( x_j \geq 0 \) for \( j = 1, 2, \ldots, n \). Moreover, if there is \( i^{th} \) constraint in \( \{ 1, 2, \ldots, m \} \) such that \( A_i x = 0 \) then \( i^{th} \) index will also be added into \( FB \). Let the gradient vector of the objective function, \( c \), be the first direction \( d_0 \). In the jump process, the point \( x_0 \) will jump along \( d_0 \) to \( x_i^1 \) on the hyperplanes which the point binds to \( i^{th} \) hyperplane as shown in Figure 3.1. It corresponds to \( A_i x_i^1 = b_i \) where \( x_i^1 = x_0 + \alpha_i^0 d_0 \) for \( i \in \{ 1, 2, \ldots, m + n \} \). 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 \( d_0 \) which the point corresponds to the minimum distance between the starting point \( x_0 \) and the point on the hyperplane i.e. \( x_i^1, i \in \{ 1, 2, \ldots, m + n \} \). Each point \( x_i^1 \) on \( i^{th} \) hyperplane
must be $A_i(x_0 + \alpha_0^i d_0) = b_i$ for $i \in \{1, 2, ..., m+n\} \setminus FB$. Since $x_0$ is equal to $0$, this implies that $\alpha_0^i = \frac{b_i}{A_i \cdot d_0}$. To maintain the feasibility of the jump point as shown in Figure 3.1, the jump point must be the minimum of the positive $\alpha_0^i$ which is defined by $x_1 = \alpha_0 d_0$ where $\alpha_0 = \min_{i \in \{1, 2, ..., m+n\} \setminus FB} \{ \alpha_0^i \} = \min_{i \in \{1, 2, ..., m+n\} \setminus FB} \left\{ \frac{b_i}{A_i \cdot d_0} > 0 \right\}$ and $A_i$ is the gradient vector of $i^{th}$ constraint. Then $x_1$ 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 $x_0$.

2. Denote the first direction by $d_0 = c$.

3. Compute $\alpha_0 = \min_{i \in \{1, 2, ..., m+n\} \setminus FB} \left\{ \frac{b_i}{A_i \cdot d_0} > 0 \right\}$.

4. Let $r_0 = \arg\min_{i \in \{1, 2, ..., m+n\} \setminus FB} \left\{ \frac{b_i}{A_i \cdot d_0} > 0 \right\}$.

5. Compute $x_1 = x_0 + \alpha_0 d_0$ binding $r_0^{th}$ constraint.

![Figure 3.1: The initial jump phase](image)

The starting point of the initial jump phase is $x_0 = (0, 0)$ and the jump direction is $d_0 = c = [1, 1.5]^T$. At this point, the hyperplanes $x_j = 0, j = 1, 2$ is binded by $x_0$ then
the set of the first binding constraints is $FB = \{3, 4\}$. From step 3 of the initial jump phase, $\alpha_0$ is calculated by

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

Then the initial jump point is $x_1 = x_0 + \alpha_0 c = (0, 0) + 1.25(1, 1.5) = (1.25, 1.88)$ which is represented by $x_1$ 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.
Example 3.1

Maximize \( x_1 + 2x_2 \)

subject to

1. \(-x_1 + x_2 \leq 2.5\)  
2. \(x_1 + 2x_2 \leq 10\)  
3. \(x_1 - x_2 \leq 0\)  
4. \(x_1 \geq 0\)  
5. \(x_2 \geq 0\)

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 \(x_1 = (2, 4)\) with \(d_0 = (1, 2), \alpha_0 = 2\). The initial jump point binds with the second hyperplane, this jump is called the initial bounded jump.

![Figure 3.2](image)

**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 \(x_1\) does not exist because there is no edge for supporting...
the minimum distance of the jump process along \( d_0 \). This implies that the \( \alpha_0 \) increases indefinitely i.e.

\[
\alpha_0 = \min\left\{ \frac{b_6}{A_6 \cdot d_0} \right\} = \min\left\{ \frac{3}{-2, 1, 1, 2} \right\} = \infty.
\]

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 \( d_0 \) 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 $x_1$ 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 $x_2$ 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 \( d_0 \) that points away from the feasible region. The method has to maintain this problem by jumping with the new direction \( d'_0 \) 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 \( F = \{ x \in \mathbb{R}^n | Ax \leq b, x \geq 0, b \geq 0 \} \) and \( F \) is nonempty. Let \( x^1 \) and \( x^2 \) be the vertices of the polyhedron. Since all vertices of the polyhedral correspond to \( F \) then \( Ax^1 \leq b, x^1 \geq 0 \) and \( Ax^2 \leq b, x^2 \geq 0 \). Let \( x \) be the point between \( x^1 \) and \( x^2 \) i.e. \( x = \lambda x^1 + (1 - \lambda) x^2 \) for \( \lambda \in [0, 1] \). Since both of \( x^1 \) and \( x^2 \) are nonnegative value then \( x \) is also nonnegative value. Considering for \( \lambda \in [0, 1] \) and \( x = \lambda x^1 + (1 - \lambda) x^2 \) such that

\[
Ax = A(\lambda x^1 + (1 - \lambda) x^2) = \lambda Ax^1 + (1 - \lambda) Ax^2 \leq \lambda b + (1 - \lambda) b = b
\]

So point \( x \) 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 \( x \). After that the new direction will be created by \( d'_0 = x - x_0 \). This direction will take the initial jump point \( x_1 \) into the feasible region.

Both vertices \( x^1 \) and \( x^2 \) 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 positive reduced cost. Performing simplex method until it founds the new vertex i.e. \( x^1 \). After that the method will repeat the simplex tableau to the first tableau using the entering variable which corresponds to the minimum positive reduced cost. Then the method will pivot until it found another vertex i.e. \( x^2 \). Now, the method has two vertices and it already creates the new direction \( d'_0 \) by \( \frac{x^1 + x^2}{2} - x_0 \).
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 $\mathbf{x}^2$ and $\mathbf{x}^1$. 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 second entering variable of the first simplex tableau from the minimum 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 positive 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 $\mathbf{x}^1 = (3.33, 3.33)$. 
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 $x^1 = (0, 2.5)$ in Figure 3.6.
Table 3.2: The simplex tableau of the search-direction process with $x_2$ as the entering variable.

<table>
<thead>
<tr>
<th>Iter.</th>
<th>$z$</th>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$x_4$</th>
<th>$x_5$</th>
<th>rhs</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3/2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$x_3$</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$x_4$</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$x_5$</td>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>5/2</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-5/2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$x_2$</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$x_4$</td>
<td>3</td>
<td>0</td>
<td>-2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$x_5$</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

The new direction for the jump process is $d_0' = \frac{x_1 + x_2}{2} - x_0 = (1.665, 2.915)$.

Figure 3.6: The new direction $d_0'$ from the search-direction process

From the initial jump phase, it tries to create the initial jump point along the
direction $d_0$. 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 $d'_0$ is used instead of $d_0$. 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 initial jump point by jumps. In this phase, the initial jump point $x_1$ will be shifted to another feasible point $x_2$ using the new direction $d_1$ which parallels to the hyperplanes in the set of visited hyperplane, $V$. The direction $d_1$ 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 $d_1$ is created by subtracting the point $x'_1$ and $x_1$, where $x'_1$ is derived by solving $A_i x'_1 = b_i$ and $A_i$ is the submatrix $A$ containing $i^{th}$ row of $A$ and $i \in V$. After that the $x_1$ will move along $d_1$ to bind the remaining constraints in $\{1, 2, ..., m+n\} \setminus V$. The new point $x_2$ will be blocked by constraint $r_1$ for some $r_1 \in \{1, 2, ..., m+n\} \setminus V$ i.e. $A_{r_1} x_2 = b_{r_1}$. In addition, the new jump point $x_2$ must stays within the feasible region. Since the jump point is $x_2 = x_1 + \alpha_1 d_1$ then $A_{r_1} (x_1 + \alpha_1 d_1) = b_{r_1}$. This implies that $\alpha_1 = \frac{b_{r_1} - A_{r_1} \cdot x_1}{A_{r_1} \cdot d_1}$. Then the new feasible jump point corresponds to $x_2 = x_1 + \alpha_1 d_1$ where $\alpha_1 = \frac{b_{r_1} - A_{r_1} \cdot x_1}{A_{r_1} \cdot d_1} = \min_{i \in \{1,2,\ldots,m+n\} \setminus V} \left\{ \frac{b_i - A_i \cdot x_k}{A_i \cdot d_k} \geq 0 \mid A_i \cdot d_k > 0 \right\}$ and
This process is repeated in 6 steps for finding a vertex.

1. Consider \( k^{th} \) jump with \( x_k \) 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 \( x_k' \). Note that \( A_i x_k' = b_i \) for \( i \in V \), and then the point is defined as \( x_k' = A_V^{-1} b_V \) where \( A_V^{-1} \) is the right inverse of \( A_V \).

3. The direction \( d_k \) is defined by \( d_k = x_k' - x_k \) if \( c \cdot d_k > 0 \) and \( d_k = x_k - x_k' \) if \( c \cdot d_k < 0 \). Note that if \( x_k = x_k' \) then \( d_k \) will be generated using the artificial hyperplane process which will be shown in the next subsection.

4. Compute \( \alpha_k = \min_{i \in \{1,2,...,m+n\}\setminus V} \left\{ \frac{b_i - A_i \cdot x_k}{A_i \cdot d_k} \geq 0 \right\} \) from the set of the nonbinding constraints, \( I \setminus V \).

5. Find \( x_{k+1} = x_k + \alpha_k d_k \) and add \( r_k = \arg\min_{i \in \{1,2,...,m+n\}\setminus V} \left\{ \frac{b_i - A_i \cdot x_k}{A_i \cdot d_k} \geq 0 \right\} \) 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 \( x_k \) will be the feasible vertex and it is the starting vertex for the simplex method.

Consider this process with the following LP problem.

**Example 3.2**

Maximize \( x_1 + 2x_2 + 2x_3 \)

subject to \( x_1 + x_2 + x_3 \leq 1 \) \hspace{1cm} (1)

\( x_1 \geq 0 \) \hspace{1cm} (2)

\( x_2 \geq 0 \) \hspace{1cm} (3)

\( x_3 \geq 0 \) \hspace{1cm} (4)
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 \( x_1 \) 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 \( d_1 \). The first step of creating the direction is to find another point on the hyperplane in \( V = \{1\} \) which is computed by solving \( A_1 x_1 = b_1 \) and let \( d_1 = x_1 - x_1 \) and it will be tested with the vector gradient of the objective function, \( c \). If \( d_1 \cdot c > 0 \) then it is the improving direction of the LP problem. Otherwise, reset \( d_1 = x_1 - x'_1 \). Now, \( x_2 \) 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, \( x_2 \) is still not the vertex of the LP problem. So the method will repeat this process until it reaches the vertex, \( x_3 \). After that the simplex method will be started at this point \( x_3 \).

The above example shows the process to create the direction of the jumps in the jump-to-vertex phase. Sometimes, the point \( x'_k \) and \( x_k \) are the same point. This implies
that the direction $d_k$ 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 $x_k'$ by $x_k$. However, it is possible that both points will be the same point. This implies that the direction $d_k$ will not exist. The method needs to find the new $x_k'$ from the intersection point between the artificial hyperplane and the hyperplane in $V$ as shown in Figure 3.8. Then $d_k'$ 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 $\beta$ i.e. $c^T x = z + \beta$ where $z = c^T x_k$. After that, the intersection point of the hyperplanes in $V$ and the artificial hyperplane will be solved for finding the new $x_k'$. Since the point $x_k'$ is on the improving artificial hyperplane and the hyperplanes in $V$ so the point $x_k'$ cannot be equal to $x_k$. The process performs 6 steps as follows.

1. Compute $z = c^T x_k$.
2. Add some positive $\beta$ to $z$ to guarantee increase of the objective function value.
3. The artificial constraint is $c^T x = z + \beta$ 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. $x_k' = A_i^b b_i$ for $i \in V$.
5. Remove the artificial hyperplane, $l$, from $V$.
6. The direction $d_k$ is defined by $d_k = x_k' - x_k$.

Considering the following Figure 3.8, the hyperplane $x_1 + x_2 + x_3 = 1$ is intersecting to the artificial hyperplane. Assume that the objective function is $2x_1 + x_2 + x_3$. Since the initial jump point is $x_1 = (0.5, 0.25, 0.25)$ thus the current objective function value
is 1.5. At the second steps of the above process, the objective value at \( x_1 \) will be added by the positive value, \( \beta \). Suppose that the given \( \beta \) 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 \( A_i x'_i = b_i \) for \( i \) in \( V = \{1, 5\} \). The solution of this linear subsystem is \( x'_1 = (0.6, 0.2, 0.2) \). Next, the artificial hyperplane \( l \) will be removed from \( V \). Hence the new improve direction is \( d'_1 = x'_1 - x_1 = [0.1, -0.05, -0.05]^T \) and \( V = \{1\} \). After that the method will begin the jump-to-vertex phase at \( x_1 \) using the direction \( d'_1 \).

**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

Maximize \( x_1 + 1.5x_2 \)

subject to

1. \(-x_1 + x_2 \leq 1\)
2. \(x_1 + 2x_2 \leq 5\)
3. \(x_1 \geq 0\)
4. \(x_2 \geq 0\)

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. \( \alpha_0 \). 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 \( \{3, 4\} \). \( \alpha_0 \) is calculated
as

$$\alpha_0 = \min_{i \in \{1,2,3,4\}\setminus FB} \{ \frac{b_i}{A_i^T c} \} = \left( \begin{array}{cccc} \frac{5}{2} & 1 & -1 & \frac{1}{2} \\ \end{array} \right) = \min \left( \frac{5}{4}, 2 \right) = \frac{5}{4}. $$

Now, the initial jump point is defined as $x_1 = \alpha_0 c = (1.25, 1.875)$ 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 $x_1$. 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 $A_2x_1' = b_2$ or $x_1' + 2x_2' = 5$. In this situation, the matrix...
A_2 is the full rank matrix but it is not the square matrix. To solve this problem, the right inverse of A_2 will be used instead of the regular inverse of A_2. The right inverse can be directly computed by A_2^T (A_2 A_2^T)^{-1}, then the solution of A_2 x_1' = b_2 is

\[x_2' = A_2^T (A_2 A_2^T)^{-1} b_2\]

The new point on the hyperplane is x_1' = (1, 2). The direction d_1 is created by subtracting x_1' and x_1 together that is d_1 = x_1' - x_1 = (1, 2) - (1, 25, 1.875) = \([-0.25, 0.125]^T\). Now, the starting point of the jump-to-vertex phase is x_1 and the direction of the jump process is d_1. It is ready to jump to the next hyperplanes. Before starting the jump process, the method will test d_1 whether this direction gives a better objective function value or not. Testing the direction by performing d_1 \cdot c. If d_1 \cdot c > 0 so this direction will improve the objective value otherwise the direction will become -d_1. Considering d_1 \cdot c = -0.0625 < 0 then the new direction is d_1' = -d_1 = \([0.25, -0.125]^T\).

![Figure 3.11: The jump-to-vertex phase of example 3.3](image)

In the jump process, the method needs to compute the minimum step of d_1 := d_1' from x_1 to the remaining constraints, α_k, using the following formula.

\[α_k = \min_{i \in \{2, \ldots, m+n\} \setminus V} \left\{ \frac{b_i - A_i \cdot x_k}{A_i \cdot d_k} \geq 0 \Bigg| A_i \cdot d_k > 0 \right\} \]
Then $\alpha_1$ is minimum of $\frac{b_i - A_i \cdot x_1}{A_i d_1} \geq 0$ where $A_i d_1 > 0, i \in \{1, 2, 3, 4\}$ and $i \notin V = \{1\}$. The remaining hyperplanes are the 2, 3, 4 hyperplanes, then the $\alpha_1$ is 15.

\[ \alpha_1 = \min \left\{ \begin{array}{c}
1 - \begin{bmatrix} -1 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} 1.25 \\ 1.875 \end{bmatrix}, \\
-1 \cdot 0.25, \\
1 \cdot [0 \ 1], \\
1 \cdot [-0.125 \ 0.25], \\
0 \cdot [-0.125 \ 0.25]
\end{array} \right\} = \{-1, -5, 15\} = 15 \]

Now the new jump point is $x_2 = x_1 + \alpha_1 d_1 = (1.25, 1.875) + 15(0.25, -0.125) = (1.25, 1.875) + (3.75, -1.875) = (5, 0)$ 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 $x_2$ is also the vertex of the LP problem. The last step, the preceding-jump simplex method will start the simplex method at $x_2$ 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 mechanic 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. Therefore, 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 immediately 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 $d_0 = c = [1, 1.5]^T$ and the method did the initial jump phase with the infeasible starting point $x_0 = (0, 0)$ then the jump point $x_1$ 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**

Maximize $x_1 + 1.5x_2$

subject to \[ x_1 + 2x_2 \leq 10 \] \hspace{1cm} (1)
\[-x_1 + x_2 \leq 4 \] \hspace{1cm} (2)
\[ x_1 \geq 2 \] \hspace{1cm} (3)
\[ x_2 \geq 2 \] \hspace{1cm} (4)
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

Minimize \( x_{a_1} + x_{a_2} \)
subject to \( x_1 + 2x_2 + x_3 = 10 \)
\(-x_1 + x_2 + x_4 = 4 \)
\( x_1 - x_5 + x_{a_2} = 2 \)
\( x_2 - x_6 + x_{a_2} = 2 \)
\( x_1, x_2 \geq 0 \)

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 \( x_0 = A \) will become
the starting point of the preceding-jump simplex method in the next phase. Note that
\( x_0 \) 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
\( I = \{1, 2, \ldots, m + n\} \) in the jump process.

Phase II (the preceding-jump simplex method)

From this example, phase I returns \( x_0 = (2, 2) \) then the jump process uses this
point to start the initial jump phase of phase II. The jump point will be completed at
\( x_1 = x_0 + \alpha_0 d_0 \) where \( \alpha_0 = \min \{ \frac{b_i - A_i x_0}{A_i d_0} | A_i d_0 > 0, d_0 = c \text{ and } x_0 = (2, 2) \} \). Similarly to the initial jump phase of the section 3.1, \( \alpha_0 \) is equal to 1 with the jump point
\( x_1 = (4, 3) = (2, 2) + 1(2, 1) \) which it binds the first constraint of the LP problem. Thus
\( r_0 = 1 \) with respect to \( \alpha_0 \) 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 $x_1$ on the hyperplanes in $V = \{1\}$ will be created and it is $x'_1 = (2, 4)$. So the new direction is $d_1 = x'_1 - x_1 = (2, 4) - (4, 3) = (-2, 1)$ and $d_1 \cdot c = -0.5 < 0$. This implies that this direction will give a decreasing objective value if it jumps along this direction. Thus the method will convert $d_1$ by $d'_1 := -d_1$. The last step of this phase is to calculate the minimum step length of the direction $d_1$ from $x_1$ to the remaining constraints, i.e. $\alpha_1$.

$$
\alpha_1 = \min_{i \in \{2, 3, 4\} \setminus V} \left\{ \begin{array}{c}
4 - \begin{bmatrix} -1 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} 4 \\ 3 \end{bmatrix}, \\
2 - \begin{bmatrix} 1 \\ 0 \end{bmatrix} \cdot \begin{bmatrix} 4 \\ 3 \end{bmatrix}, \\
2 - \begin{bmatrix} 0 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} 4 \\ 3 \end{bmatrix}
\end{array} \right\} = \{-1.667, -1, 1\} = 1
$$
The next jump point is \( x_2 = x_1 + \alpha_1 d_1 = x_1 + \alpha_1 d_1' = (4, 3) + 1(2, -1) = (6, 2) \) 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 \( x_2 \) is also the optimal solution.

![Figure 3.14: The jump-to-vertex phase of example 3.4](image)

The following subsections will explain all steps of the preceding-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 \( x_0 \in \mathbb{R}^n \) as the initial starting point from phase I. Denote \( d_0 \) as the first direction and it is equal to \( c \). The initial jump point is \( x_1 = x_0 + \alpha_0 d_0 \) where \( \alpha_0 = \min_{i \in \{1, 2, \ldots, m+n\} \setminus FB} \left\{ \frac{b_i - A_i x_0}{A_i \cdot d_0} > 0 \mid A_i \cdot d_0 > 0 \right\} \)
and $A_i$ 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 $x_0$ which is called the starting point of the preceding-jump simplex method.

4. Define the first jumping direction $c$ as $d_0$.

5. Calculate $\alpha_0 = \min_{i \in \{1, 2, \ldots, m+n\} \backslash FB} \left\{ \frac{b_i - A_i x_0}{A_i \cdot d_0} > 0 \left| A_i \cdot d_0 > 0 \right. \right\}$.

6. The initial jump point is $x_1 = x_0 + \alpha_0 d_0$ 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 $d_0$ will be changed to $d'_0$ which 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

$$r_0 = \arg\min_{i \in \{1, 2, \ldots, m+n\} \backslash FB} \left\{ \frac{b_i - A_i x_0}{A_i \cdot d_0} > 0 \left| A_i \cdot d_0 > 0 \right. \right\}.$$

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, $d_0$, points away from the feasible region. Then this problem will be fixed by creating the new feasible direction $d'_0$ using the search-direction process. Assume that example 3.4 has another infeasible direction $d_0 = [-1, 1]^T$. 
Maximize \(-x_1 + x_2\)
subject to \(x_1 + 2x_2 \leq 10\) \hspace{1cm} (1)
\(-x_1 + x_2 \leq 4\) \hspace{1cm} (2)
\(x_1 \geq 2\) \hspace{1cm} (3)
\(x_2 \geq 2\) \hspace{1cm} (4)

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

![Figure 3.15: Jump into the infeasible region](image)

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 \(x_1 = (1.5, 2.5)\). 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 \( c \) or \( d_0 \) 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, \( d_0' \). The direction is created by the summation 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 \( c \). Hence the direction \( d_0' \) 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 \( c \) i.e. the angle of \( c \) and \( A_i \) 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 \( x^1 \) and \( x^2 \) as shown in the search-direction process in the previous scenario then \( d_0 = \frac{x^1 + x^2}{2} - x_0 \). Otherwise, it has to create an artificial hyperplane by \( c^T x = c^T x_0 + \beta \) where \( x_0 \) is the given feasible vertex from phase I of the two-phase simplex method and \( \beta \) 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 \( A_j \) and \( c \), i.e. \( x'_0_j = A_j^{g^b_j}, j \in AC \). Note that if there are constraints in \( AC \) which is represented by "\( \leq " \) then \( x'_0 \) will be solved only these constraints.

5. Generate the direction from \( d_{0_j} = x'_0_j - x_0 \) for all \( j \) in \( AC \).

6. Do the initial jump phase with these directions \( d_{0_j} \) 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. \( d'_0 = \sum_{j=1}^{L} d_0 \), where \( j \in AC \).

If all jump points using \( d_0 \) violate some constraints then \( d'_0 \) will be create from the search-direction process from the previous phase.

7. Do the initial jump phase with the direction \( d'_0 \).

Figure 3.16 shows the new direction \( d'_0 \) 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 \( \theta_i = \arccos \left( \frac{A_i^c}{||A_i||} \right), i \in FB \). 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 \( \theta = \{ \arccos \left( \frac{A^c}{||A||} \right), \arccos \left( \frac{A^c}{||A||} \right) \} = \{59.03, 30.96\} \). Both make the acute angle to \( c \) then \( AC = \{3, 4\} \). The next step, the method has to create an artificial hyperplane using \( c^T x = c^T x_0 + \beta \) which assume \( \beta \) is equal to 1. Considering the value of the objective function at the vertex \( A \), \( c^T x_0 = -(2) + 1(2) = 0 \) 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 \( d'_{03} = x'_{03} - x_0 = (1, 2) - (2, 2) = [-1, 0]^T \) and \( d'_{04} = x'_{04} - x_0 = (2, 3) - (2, 2) = [0, 1]^T \).

The method will perform the initial jump with these directions and checking the feasibility of those jump points which are \{(1, 2), (2, 3)\} but (1,2) violates 4th constraint. Then the direction \( d'_0 \) is \([0, 1]^T\) which is shown in Figure 3.16. After that the preceding-jump will take \( d'_0 \) 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 finding 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

1: procedure PJS($c, b, A$)
2: \hspace{1em} \textbf{Input}: an LP problem $c, b, A$
3: \hspace{1em} \textbf{Output}: the optimal solution $x = \{x_1, \ldots, x_n\}$
4: #Phase 1
5: \hspace{1em} while True do
6: \hspace{2em} $z^T_N \leftarrow c_B^T A_B^{-1} A_N - c^T_N$
7: \hspace{2em} If $z^T_N \leq 0$ then STOP return $(x_B, x_N)$
8: \hspace{2em} $k \leftarrow \text{argmax}\{z^T_N\}$
9: \hspace{2em} $y_k \leftarrow A_B^{-1} A_{N_k}$
10: \hspace{2em} If $y_k \leq 0$ then STOP return infeasible solution
11: \hspace{2em} $r \leftarrow A_B^{-1} b$
12: \hspace{2em} $l \leftarrow \text{argmin}\{\frac{r}{y_{ik}} | y_{ik} > 0, i \in B\}$
13: \hspace{2em} $B \leftarrow (B \setminus \{l\}) \cup \{k\}$
14: \hspace{2em} $N \leftarrow (N \setminus \{k\}) \cup \{l\}$
15: \hspace{2em} If $x_{a_k} \in x_B$ and $x_{a_k} = 0$ for some $k = \{1, 2, \ldots, l\}$ then
16: \hspace{3em} $x_B \leftarrow$ pivot the artificial out of $x_B$
17: \hspace{2em} else $x_{a_k} \in x_B$ and $x_{a_k} > 0$ for some $k = \{1, 2, \ldots, l\}$ then
18: \hspace{3em} return infeasible solution
19: \hspace{2em} $x_0 \leftarrow (x_B, x_N)$
20: #Initial jump phase
21: \hspace{1em} $FB \leftarrow \text{Keep the indices of binding hyperplanes at } x_0$
22: \hspace{1em} $x_0 \leftarrow \text{Remove slack and surplus variables from } x_0$
23: \hspace{1em} $d_0 \leftarrow c$ the gradient objective function
24: \hspace{1em} $r_0 \leftarrow \text{argmin}\{\frac{b_i - \frac{A_i}{A_0}}{A_0} x_0 | A_i \cdot d_0 > 0, i \in \{1, 2, \ldots, m + n\} \setminus FB\}$
25: \hspace{1em} $\alpha_0 \leftarrow \frac{b_0 - A_0 x_0}{A_0 \cdot d_0}$
26: \hspace{1em} $x_1 \leftarrow x_0 + \alpha_0 d_0$
27: \hspace{1em} $V \leftarrow V \cup \{r_0\}$
28: \hspace{1em} If $x_1$ violates some constraints then
29: \hspace{2em} #The search-direction process
30: \hspace{3em} $AC \leftarrow \{i \in FB | \arccos(\frac{A_i \cdot c}{\|A_i\| \cdot \|c\|}) > 0\}$
31: \hspace{3em} If $AC$ is an empty set then
32: \hspace{4em} $x^1 \leftarrow$ the vertex from the simplex algorithm using the entering variable from the maximum positive reduce cost
33: \hspace{4em} $x^2 \leftarrow$ the vertex from the simplex algorithm using the entering variable from the minimum positive reduce cost
34: \hspace{4em} $d_0 \leftarrow \frac{x^1 + x^2}{2} - x_0$
35: \hspace{2em} else
36: \hspace{3em} $\psi \leftarrow c^T x_0 + \beta$ where $\beta \in \mathbb{R}^+$
Algorithm 3 The preceding-jump simplex algorithm (continued)

39: \( A_c \leftarrow \) the artificial hyperplane \( c^T x = \psi \)
40: \[ \text{for } i \text{ in } AC \]
41: \( G \leftarrow \{i\} \cup \{c\} \)
42: \( x'_i \leftarrow A^*_G b_G \)
43: \( G \leftarrow G \setminus \{i\} \)
44: \( d'_i \leftarrow x'_i - x_0 \)
45: \[ \text{If } d'_i \text{ does not violate any constraint then} \]
46: \( d_0 \leftarrow \sum_j d'_j \)
47: \[ \text{else do the step 32 to 36} \]
48: \( x_1 \leftarrow \) repeat steps 21 to 27
49: \[ \# \text{Jump-to-vertex phase} \]
50: \[ \text{while } |V| < n \text{ do} \]
51: \( x'_k \leftarrow A^*_V b_V \) where \( k \) is \( k^{th} \) jump point
52: \[ \text{If } x_k = x'_k \text{ then} \]
53: \[ \# \text{The artificial hyperplane process} \]
54: \( \psi \leftarrow c^T x_0 + \beta \) where \( \beta \in \mathbb{R}^+ \)
55: \( A_c \leftarrow \) the artificial hyperplane \( c^T x = \psi \)
56: \( V \leftarrow V \cup \{c\} \)
57: \( x'_k \leftarrow A^*_V b_V \)
58: \( V \leftarrow V \setminus \{c\} \)
59: \( d_0 \leftarrow x_k - x_k \)
60: \[ \text{else} \]
61: \( d_k \leftarrow x'_k - x_k \)
62: \[ \text{If } c \cdot d_k < 0 \text{ then} \]
63: \( d_k \leftarrow -d_k \)
64: \( x_{k+1} \leftarrow x_k + \alpha_k d_k \)
65: \( r_k \leftarrow \arg \min \{ \frac{b_i - A_i x_k}{A_i d_k} \geq 0 | A_i \cdot d_k > 0 \} \) in \( V \)
66: \( V \leftarrow V \cup \{r_k\} \)
67: \[ \text{end} \]
68: \( x_N \leftarrow \) the slack variables in \( V \)
69: \( x_B \leftarrow x_i \notin x_N \) where \( i \in \{1, 2, ..., m + n\} \)
70: \[ \# \text{The simplex algorithm} \]
71: \[ \text{while True do} \]
72: \( z_N^T \leftarrow c_N^T A_B^{-1} A_N - c_N^T \)
73: \[ \text{If } z_N^T \leq 0 \text{ then STOP return the current BFS is the optimal solution} \]
74: \( k \leftarrow \arg \max \{z_N^T \} \)
75: \( y_k \leftarrow A_B^{-1} A_{N_k} \)
76: \[ \text{If } y_k \leq 0 \text{ then STOP return infeasible solution} \]
77: \( r \leftarrow A_B^{-1} b \)
78: \( l \leftarrow \arg \min \{ \frac{r}{y_{ik}} | y_{ik} > 0, i \in B \} \)
79: \( B \leftarrow (B \setminus \{l\}) \cup \{k\} \)
80: \( N \leftarrow (N \setminus \{k\}) \cup \{l\} \)
81: \[ \text{end} \]
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 $Ax \leq b$ constraints, $x \geq 0$ and $b \geq 0$. 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 $b$ is computed from $b = Ax \cdot \text{sign}(Ax)$ where the function sign returns the negative or positive of $Ax$. In order to guarantee the positivity of $b$, let $b = Ax \cdot \text{sign}(Ax)$. Then the rhs vector $b$ is added by the positive vector $[5, \ldots, 5]^T$ to confirm $Ax \leq b$. 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,

\[
\text{Improvement of the number of iterations} = \left( \frac{\text{SPX iterations} - \text{PJS iterations}}{\text{SPX iterations}} \right) \times 100
\]

and

\[
\text{Improvement of time} = \left( \frac{\text{SPX time} - \text{PJS time}}{\text{SPX time}} \right) \times 100.
\]

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

<table>
<thead>
<tr>
<th>(m, n)</th>
<th>PJS</th>
<th>SPX</th>
<th>\begin{tabular}{c} PJS \end{tabular}</th>
<th>\begin{tabular}{c} SPX \end{tabular}</th>
<th>\begin{tabular}{c} Iterations \end{tabular}</th>
<th>\begin{tabular}{c} Time (sec.) \end{tabular}</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Jump</td>
<td>Total</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>114.700</td>
<td>290.900</td>
<td>0.052</td>
<td>0.142</td>
<td>0.200</td>
<td>60.571</td>
</tr>
<tr>
<td>200</td>
<td>403.700</td>
<td>1109.400</td>
<td>0.178</td>
<td>0.813</td>
<td>1.429</td>
<td>63.611</td>
</tr>
<tr>
<td>300</td>
<td>801.800</td>
<td>2399.900</td>
<td>1.771</td>
<td>3.615</td>
<td>5.078</td>
<td>66.590</td>
</tr>
<tr>
<td>400</td>
<td>1247.300</td>
<td>4384.700</td>
<td>4.398</td>
<td>8.095</td>
<td>12.147</td>
<td>71.553</td>
</tr>
<tr>
<td>500</td>
<td>2297.500</td>
<td>6911.200</td>
<td>9.512</td>
<td>18.141</td>
<td>24.888</td>
<td>66.757</td>
</tr>
<tr>
<td>600</td>
<td>2702.300</td>
<td>10070.100</td>
<td>16.674</td>
<td>29.385</td>
<td>45.456</td>
<td>73.165</td>
</tr>
<tr>
<td>700</td>
<td>3846.500</td>
<td>13766.000</td>
<td>28.601</td>
<td>50.475</td>
<td>76.190</td>
<td>72.058</td>
</tr>
<tr>
<td>800</td>
<td>5348.600</td>
<td>19258.800</td>
<td>51.570</td>
<td>88.151</td>
<td>129.567</td>
<td>72.228</td>
</tr>
<tr>
<td>900</td>
<td>6736.400</td>
<td>24028.600</td>
<td>77.805</td>
<td>132.438</td>
<td>212.230</td>
<td>71.965</td>
</tr>
<tr>
<td>1000</td>
<td>8152.200</td>
<td>29724.400</td>
<td>111.712</td>
<td>189.818</td>
<td>301.530</td>
<td>72.574</td>
</tr>
<tr>
<td>1200</td>
<td>11035.700</td>
<td>44402.800</td>
<td>207.542</td>
<td>351.501</td>
<td>559.043</td>
<td>75.146</td>
</tr>
<tr>
<td>1400</td>
<td>14257.300</td>
<td>60331.400</td>
<td>331.380</td>
<td>561.641</td>
<td>993.021</td>
<td>76.368</td>
</tr>
<tr>
<td>1600</td>
<td>18233.300</td>
<td>79008.700</td>
<td>562.844</td>
<td>922.446</td>
<td>1485.286</td>
<td>76.922</td>
</tr>
<tr>
<td>1800</td>
<td>24886.800</td>
<td>106142.200</td>
<td>781.395</td>
<td>1404.140</td>
<td>2185.535</td>
<td>76.553</td>
</tr>
<tr>
<td>2000</td>
<td>30190.300</td>
<td>132224.300</td>
<td>1115.499</td>
<td>2007.276</td>
<td>3122.775</td>
<td>77.167</td>
</tr>
<tr>
<td>2500</td>
<td>43452.700</td>
<td>203917.700</td>
<td>2273.835</td>
<td>4089.269</td>
<td>6363.104</td>
<td>78.691</td>
</tr>
</tbody>
</table>

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.

![Graph showing the average number of iterations of PJS and SPX](image)

**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.
4.2 Randomly generated problems with the infeasible origin point

The synthetic LP problems in this scenario consist of \( Ax \oplus b \) constraints \( x \geq 0 \) and \( b \oplus 0 \) where \( \oplus \) is one of \( \leq, =, \geq \). The number of variables and constraints 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 \( b \) is computed by \( b = Ax \). Then the components of vector \( b \) are separated into two cases that is \( b_i \geq 0 \) for some \( i \in \{1, 2, \ldots, m\} \) and \( b_j < 0 \) for some \( j \in \{1, 2, \ldots, m\} \). For \( i^{th} \) component of \( b_i \geq 0 \), the \( A_i x = b_i \) is converted to \( A_i x \leq b_i \) by adding 5 to \( b_i \). If \( b_j < 0 \) for some \( j \in \{1, 2, \ldots, m\} \), the coefficients of \( A_j \) and \( b_j \) are converted by \( -1 \times A_j x \) and \( -1 \times b_j \), respectively. These constraints are denoted by \( A_j x \geq b_j \) for some \( j \in \{1, 2, \ldots, m\} \). Hence the vector \( b \geq 0 \) 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

![Figure 4.2: The time average of PJS and SPX](image-url)
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.

<table>
<thead>
<tr>
<th>(m, n)</th>
<th>PJS</th>
<th>two phase SPX</th>
<th>PJS</th>
<th>two phase SPX</th>
<th>Iterations</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>367.800</td>
<td>1008.200</td>
<td>0.052</td>
<td>0.347</td>
<td>0.640</td>
<td>63.519</td>
</tr>
<tr>
<td>200</td>
<td>1026.300</td>
<td>4446.600</td>
<td>0.255</td>
<td>1.822</td>
<td>5.809</td>
<td>76.919</td>
</tr>
<tr>
<td>300</td>
<td>2141.600</td>
<td>10806.100</td>
<td>0.717</td>
<td>5.974</td>
<td>23.704</td>
<td>80.182</td>
</tr>
<tr>
<td>400</td>
<td>1879.600</td>
<td>20099.300</td>
<td>1.575</td>
<td>9.219</td>
<td>64.151</td>
<td>90.606</td>
</tr>
<tr>
<td>500</td>
<td>756.500</td>
<td>31549.500</td>
<td>3.285</td>
<td>9.708</td>
<td>141.146</td>
<td>97.602</td>
</tr>
<tr>
<td>600</td>
<td>196.900</td>
<td>48011.333</td>
<td>5.867</td>
<td>12.092</td>
<td>332.921</td>
<td>99.590</td>
</tr>
<tr>
<td>700</td>
<td>1038.400</td>
<td>66632.333</td>
<td>10.306</td>
<td>33.809</td>
<td>1003.971</td>
<td>98.442</td>
</tr>
<tr>
<td>800</td>
<td>80.500</td>
<td>85628.000</td>
<td>17.382</td>
<td>33.068</td>
<td>2578.906</td>
<td>99.906</td>
</tr>
<tr>
<td>900</td>
<td>176.100</td>
<td>111799.000</td>
<td>25.339</td>
<td>47.995</td>
<td>4769.367</td>
<td>99.842</td>
</tr>
<tr>
<td>1000</td>
<td>63.667</td>
<td>138894.000</td>
<td>41.221</td>
<td>63.911</td>
<td>6732.708</td>
<td>99.954</td>
</tr>
</tbody>
</table>

**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.
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 problems 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


<table>
<thead>
<tr>
<th>(m,n)</th>
<th>No. problems</th>
<th>PJS</th>
<th>two phase SPX</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Time (sec.)</td>
<td>Iterations</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Phase</td>
<td>Jump</td>
</tr>
<tr>
<td>100</td>
<td></td>
<td>Initial</td>
<td>direction</td>
</tr>
<tr>
<td>1</td>
<td>0.063 0.070 0.391</td>
<td>155.000</td>
<td>399.000</td>
</tr>
<tr>
<td>2</td>
<td>0.057 0.054 0.298</td>
<td>144.000</td>
<td>257.000</td>
</tr>
<tr>
<td>3</td>
<td>0.062 0.054 0.342</td>
<td>164.000</td>
<td>295.000</td>
</tr>
<tr>
<td>4</td>
<td>0.052 0.055 0.299</td>
<td>135.000</td>
<td>344.000</td>
</tr>
<tr>
<td>5</td>
<td>0.057 0.050 0.390</td>
<td>141.000</td>
<td>457.000</td>
</tr>
<tr>
<td>6</td>
<td>0.055 0.047 0.390</td>
<td>152.000</td>
<td>464.000</td>
</tr>
<tr>
<td>7</td>
<td>0.050 0.047 0.294</td>
<td>141.000</td>
<td>269.000</td>
</tr>
<tr>
<td>8</td>
<td>0.056 0.047 0.344</td>
<td>153.000</td>
<td>379.000</td>
</tr>
<tr>
<td>9</td>
<td>0.048 0.051 0.350</td>
<td>129.000</td>
<td>388.000</td>
</tr>
<tr>
<td>10</td>
<td>0.052 0.047 0.372</td>
<td>143.000</td>
<td>426.000</td>
</tr>
<tr>
<td></td>
<td>average</td>
<td>0.055 0.052 0.347</td>
<td>145.700</td>
</tr>
<tr>
<td></td>
<td>S.D.</td>
<td>0.005 0.007 0.039</td>
<td>10.339</td>
</tr>
<tr>
<td>200</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0.348 0.289 2.020</td>
<td>355.000</td>
<td>958.000</td>
</tr>
<tr>
<td>2</td>
<td>0.289 0.253 1.955</td>
<td>350.000</td>
<td>1089.000</td>
</tr>
<tr>
<td>3</td>
<td>0.310 0.254 1.991</td>
<td>353.000</td>
<td>1109.000</td>
</tr>
<tr>
<td>4</td>
<td>0.274 0.252 1.768</td>
<td>336.000</td>
<td>1029.000</td>
</tr>
<tr>
<td>5</td>
<td>0.290 0.232 1.818</td>
<td>352.000</td>
<td>1072.000</td>
</tr>
<tr>
<td>6</td>
<td>0.333 0.300 2.222</td>
<td>347.000</td>
<td>1131.000</td>
</tr>
<tr>
<td>7</td>
<td>0.295 0.230 1.564</td>
<td>368.000</td>
<td>849.000</td>
</tr>
<tr>
<td>8</td>
<td>0.277 0.248 1.848</td>
<td>351.000</td>
<td>1094.000</td>
</tr>
<tr>
<td>9</td>
<td>0.292 0.241 1.769</td>
<td>364.000</td>
<td>1031.000</td>
</tr>
<tr>
<td>10</td>
<td>0.274 0.245 1.268</td>
<td>349.000</td>
<td>901.000</td>
</tr>
<tr>
<td></td>
<td>average</td>
<td>0.298 0.255 1.822</td>
<td>352.500</td>
</tr>
<tr>
<td></td>
<td>S.D.</td>
<td>0.025 0.023 0.264</td>
<td>8.835</td>
</tr>
<tr>
<td>(m,n)</td>
<td>No. problems</td>
<td></td>
<td>PJS</td>
</tr>
<tr>
<td>-------</td>
<td>--------------</td>
<td>-----</td>
<td>-----</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Time (sec.)</td>
<td>Iterations</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Phase I</td>
<td>Jump</td>
</tr>
<tr>
<td>300</td>
<td>1</td>
<td>0.843</td>
<td>0.759</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0.909</td>
<td>0.729</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.790</td>
<td>0.735</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>0.779</td>
<td>0.684</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>0.945</td>
<td>0.866</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>0.781</td>
<td>0.656</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>0.805</td>
<td>0.685</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>0.758</td>
<td>0.694</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>0.745</td>
<td>0.683</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>0.735</td>
<td>0.677</td>
</tr>
<tr>
<td></td>
<td></td>
<td>average</td>
<td>0.809</td>
</tr>
<tr>
<td></td>
<td></td>
<td>S.D.</td>
<td>0.070</td>
</tr>
</tbody>
</table>

<p>| 400   | 1            | 1.692  | 1.717 | 14.011 | 877.000 | 3230.000 | $d_0$ | 1.670  | 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 | 3.335 | 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 | $d_0$ | 1.650  | 64.151 | 886.400 | 20009.300 |
|       |              | S.D.    | 0.057 | 0.056 | 5.850 | 34.645 | 432.344 |</p>
<table>
<thead>
<tr>
<th>(m,n)</th>
<th>No. problems</th>
<th>PJS</th>
<th>two phase SPX</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Time (sec.)</td>
<td>Iterations</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Phase I</td>
<td>Jump</td>
</tr>
<tr>
<td>500</td>
<td>1</td>
<td>3.314</td>
<td>3.208</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>3.195</td>
<td>3.196</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>3.326</td>
<td>3.262</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>3.034</td>
<td>3.190</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>3.325</td>
<td>3.180</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>3.098</td>
<td>3.937</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>2.973</td>
<td>3.349</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>3.004</td>
<td>3.176</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>3.064</td>
<td>3.191</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>2.851</td>
<td>3.162</td>
</tr>
<tr>
<td></td>
<td>S.D.</td>
<td>0.165</td>
<td>0.235</td>
</tr>
<tr>
<td>600</td>
<td>1</td>
<td>4.727</td>
<td>5.751</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>5.304</td>
<td>5.731</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>4.796</td>
<td>5.854</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>5.661</td>
<td>5.806</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>4.455</td>
<td>5.726</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>4.245</td>
<td>5.675</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>4.891</td>
<td>5.865</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>4.319</td>
<td>6.365</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>4.846</td>
<td>6.067</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>4.412</td>
<td>5.825</td>
</tr>
<tr>
<td></td>
<td>average</td>
<td>4.766</td>
<td>5.867</td>
</tr>
<tr>
<td></td>
<td>S.D.</td>
<td>0.448</td>
<td>0.206</td>
</tr>
<tr>
<td>(m,n)</td>
<td>No. problems</td>
<td>PJS</td>
<td>two phase SPX</td>
</tr>
<tr>
<td>-------</td>
<td>--------------</td>
<td>-----</td>
<td>---------------</td>
</tr>
<tr>
<td></td>
<td>Time (sec.)</td>
<td>Iterations</td>
<td>Time (sec.)</td>
</tr>
<tr>
<td></td>
<td>Phas I Jump Total</td>
<td>ph I SPX</td>
<td>Phase I Total</td>
</tr>
<tr>
<td>700</td>
<td>1</td>
<td>8.327 9.777 23.364</td>
<td>2014.000 341.000</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>7.981 11.574 20.743</td>
<td>1873.000 69.000</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>7.389 11.528 22.581</td>
<td>1800.000 238.000</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>7.778 9.869 19.726</td>
<td>1999.000 128.000</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>7.768 10.397 20.563</td>
<td>1912.000 135.000</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>8.477 9.655 20.676</td>
<td>1967.000 168.000</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>9.158 11.026 22.742</td>
<td>1983.000 138.000</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>6.830 9.507 20.294</td>
<td>1902.000 228.000</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>6.790 10.090 19.345</td>
<td>1866.000 165.000</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>6.780 9.632 14.060</td>
<td>1862.000 877.000</td>
</tr>
<tr>
<td></td>
<td>average</td>
<td>7.728 10.306 33.809</td>
<td>1907.800 1038.400</td>
</tr>
<tr>
<td></td>
<td>S.D.</td>
<td>0.798 0.793 40.166</td>
<td>69.637 2719.050</td>
</tr>
<tr>
<td>800</td>
<td>1</td>
<td>11.664 17.271 29.039</td>
<td>2303.000 0.000</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>12.189 16.752 29.038</td>
<td>2426.000 0.000</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>10.848 16.283 27.232</td>
<td>2265.000 0.000</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>15.945 19.394 40.561</td>
<td>2348.000 173.000</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>14.869 17.093 34.183</td>
<td>2450.000 84.000</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>11.635 16.401 28.137</td>
<td>2249.000 0.000</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>10.988 15.885 30.891</td>
<td>2334.000 137.000</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>15.552 20.363 39.282</td>
<td>2248.000 119.000</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>14.949 18.016 37.187</td>
<td>2389.000 151.000</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>13.875 16.361 35.132</td>
<td>2635.000 141.000</td>
</tr>
<tr>
<td></td>
<td>average</td>
<td>13.251 17.382 33.068</td>
<td>2364.700 80.500</td>
</tr>
<tr>
<td></td>
<td>S.D.</td>
<td>1.989 1.464 4.863</td>
<td>118.631 72.873</td>
</tr>
<tr>
<td>(m,n)</td>
<td>No. problems</td>
<td>PJS</td>
<td>two phase SPX</td>
</tr>
<tr>
<td>-------</td>
<td>--------------</td>
<td>-----</td>
<td>--------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Time (sec.)</td>
<td>Iterations</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Phase I</td>
<td>Jump</td>
</tr>
<tr>
<td>900</td>
<td>1</td>
<td>15.573</td>
<td>27.957</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>15.379</td>
<td>27.740</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>14.647</td>
<td>25.288</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>14.060</td>
<td>25.684</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>14.567</td>
<td>23.998</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>14.961</td>
<td>23.795</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>15.385</td>
<td>23.908</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>15.853</td>
<td>27.676</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>15.231</td>
<td>23.879</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>14.620</td>
<td>23.461</td>
</tr>
<tr>
<td></td>
<td></td>
<td>average</td>
<td>15.028</td>
</tr>
<tr>
<td></td>
<td></td>
<td>S.D.</td>
<td>0.551</td>
</tr>
<tr>
<td>1000</td>
<td>1</td>
<td>17.993</td>
<td>43.631</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>19.973</td>
<td>39.900</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>20.363</td>
<td>40.130</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>19.562</td>
<td>38.968</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>22.847</td>
<td>39.631</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>24.367</td>
<td>41.479</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>19.262</td>
<td>38.986</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>22.539</td>
<td>37.049</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>20.187</td>
<td>41.956</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>22.369</td>
<td>39.826</td>
</tr>
<tr>
<td></td>
<td></td>
<td>average</td>
<td>20.946</td>
</tr>
<tr>
<td></td>
<td></td>
<td>S.D.</td>
<td>1.978</td>
</tr>
<tr>
<td>(m,n)</td>
<td>No. problems</td>
<td>PJS</td>
<td>two phase SPX</td>
</tr>
<tr>
<td>-------</td>
<td>--------------</td>
<td>-----</td>
<td>---------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Time (sec.)</td>
<td>Iterations</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Initial</td>
<td>phase I</td>
</tr>
<tr>
<td>1</td>
<td>15.573</td>
<td>27.957</td>
<td>48.025</td>
</tr>
<tr>
<td>2</td>
<td>15.379</td>
<td>27.740</td>
<td>57.801</td>
</tr>
<tr>
<td>3</td>
<td>14.647</td>
<td>25.288</td>
<td>45.450</td>
</tr>
<tr>
<td>4</td>
<td>14.060</td>
<td>25.684</td>
<td>53.209</td>
</tr>
<tr>
<td>5</td>
<td>13.567</td>
<td>23.998</td>
<td>52.950</td>
</tr>
<tr>
<td>6</td>
<td>14.961</td>
<td>23.795</td>
<td>45.732</td>
</tr>
<tr>
<td>7</td>
<td>15.385</td>
<td>23.908</td>
<td>47.322</td>
</tr>
<tr>
<td>8</td>
<td>15.853</td>
<td>27.676</td>
<td>52.180</td>
</tr>
<tr>
<td>9</td>
<td>15.231</td>
<td>23.879</td>
<td>39.235</td>
</tr>
<tr>
<td>10</td>
<td>14.620</td>
<td>23.461</td>
<td>38.203</td>
</tr>
<tr>
<td></td>
<td>average</td>
<td>15.028</td>
<td>25.339</td>
</tr>
<tr>
<td></td>
<td>S.D.</td>
<td>0.551</td>
<td>1.828</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>Time (sec.)</th>
<th>Iterations</th>
<th>Time (sec.)</th>
<th>Iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Initial</td>
<td>phase I</td>
<td>phase II</td>
<td>direction</td>
</tr>
<tr>
<td>1</td>
<td>17.993</td>
<td>43.631</td>
<td>61.814</td>
<td>2902.000</td>
<td>0.000</td>
</tr>
<tr>
<td>2</td>
<td>19.973</td>
<td>39.900</td>
<td>69.238</td>
<td>3053.000</td>
<td>191.000</td>
</tr>
<tr>
<td>3</td>
<td>20.363</td>
<td>40.130</td>
<td>60.681</td>
<td>3292.000</td>
<td>0.000</td>
</tr>
<tr>
<td>4</td>
<td>19.562</td>
<td>38.968</td>
<td>56.508</td>
<td>2882.000</td>
<td>136.000</td>
</tr>
<tr>
<td>5</td>
<td>22.847</td>
<td>39.631</td>
<td>62.658</td>
<td>3165.000</td>
<td>0.000</td>
</tr>
<tr>
<td>6</td>
<td>24.367</td>
<td>41.479</td>
<td>71.580</td>
<td>3379.000</td>
<td>106.000</td>
</tr>
<tr>
<td>7</td>
<td>19.262</td>
<td>38.986</td>
<td>58.418</td>
<td>2930.000</td>
<td>0.000</td>
</tr>
<tr>
<td>8</td>
<td>22.539</td>
<td>37.049</td>
<td>59.760</td>
<td>3218.000</td>
<td>0.000</td>
</tr>
<tr>
<td>9</td>
<td>20.187</td>
<td>41.956</td>
<td>62.334</td>
<td>3121.000</td>
<td>0.000</td>
</tr>
<tr>
<td>10</td>
<td>22.369</td>
<td>39.826</td>
<td>78.841</td>
<td>3224.000</td>
<td>326.000</td>
</tr>
<tr>
<td></td>
<td>average</td>
<td>20.946</td>
<td>40.156</td>
<td>65.083</td>
<td>3116.600</td>
</tr>
<tr>
<td></td>
<td>S.D.</td>
<td>1.978</td>
<td>1.824</td>
<td>6.370</td>
<td>171.189</td>
</tr>
</tbody>
</table>
BIOGRAPHY

Name: Mr Natdanai Kafakthong

Date of Birth: 19 July 1994

Place of Birth: Tak, Thailand

Education: B.S. (Mathematics), Naresuan University, 2016