วิธีซิมเพล็กซ์ที่ใช้การกระโดดตามจุดประสงค์

นางสาวณัฐชา ยาวิละ

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต
สาขาวิชาคณิตศาสตร์ประยุกต์และวิทยาการคณนา ภาควิชาคณิตศาสตร์และวิทยาการคอมพิวเตอร์
คณะวิทยาศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2559

Simplex method with objective jump

Miss Nutcha Yawila

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 2016

Thesis Title	Simplex method with objective jump
By	Miss Nutcha Yawila
Field of Study	Applied Mathematics and Computational Science
Thesis Advisor	Assistant Professor Boonyarit Intiyot, Ph.D.
Thesis Co-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
(Associate Professor Polkit Sangvanich, Ph.D.)

THESIS COMMITTEE

.....Chairman
(Associate Professor Pornchai Satravaha, Ph.D.)

.....Thesis Advisor
(Assistant Professor Boonyarit Intiyot, Ph.D.)

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

.....Examiner
(Assistant Professor Phantipa Thipwiwatpotjana, Ph.D.)

.....External Examiner
(Associate Professor Peerayuth Charnsetikul, Ph.D.)

ณัฐชา ยาวิละ : วิธีซิมเพล็กซ์ที่ใช้การกระโดดตามจุดประสงค์ (Simplex method with objective jump) อ.ที่ปรึกษาวิทยานิพนธ์หลัก: ผศ. ดร.บุญฤทธิ์ อินทิยศ, อ.ที่ปรึกษาวิทยานิพนธ์ร่วม: ผศ. ดร. กรุง สินอภิรมย์สราญ, 114 หน้า.

เรานำเสนอแนวคิด “การกระโดดตามจุดประสงค์” ซึ่งเป็นวิธีการใหม่ที่ใช้หาผลเฉลยที่เป็นไปได้พื้นฐานจากจุดกำเนิดก่อนเริ่มวิธีซิมเพล็กซ์ สำหรับปัญหากำหนดการเชิงเส้นที่อยู่ในรูปแบบบัญญัติ ถ้าเวกเตอร์ทางด้านขวาของเงื่อนไขบังคับมีค่าไม่ติดลบ จุดกำเนิดจะอยู่ภายในอาณาบริเวณที่เป็นไปได้ และมักจะใช้เป็นผลเฉลยเริ่มต้นของวิธีซิมเพล็กซ์ กระบวนกระโดดตามจุดประสงค์มีวัตถุประสงค์ที่จะเริ่มต้นวิธีซิมเพล็กซ์ด้วยผลเฉลยที่เป็นไปได้พื้นฐานอื่น โดยการกระโดดจากจุดกำเนิดในทิศทางเกรเดียนต์ของจุดประสงค์ กระบวนการนี้ต้องการเงื่อนไขบังคับเพิ่มเติมเพื่อสร้างปัญหากำหนดการเชิงเส้นย่อยซึ่งมีอาณาบริเวณที่เป็นไปได้เล็กลง โดยด้านที่เพิ่มมาของอาณาบริเวณที่เล็กกว่านี้เป็นด้านที่มีแนวไปตามเกรเดียนต์เวกเตอร์ของฟังก์ชันจุดประสงค์ จากนั้นจะมีการจัดรูปเมทริกซ์แบบพิเศษ เพื่อเคลื่อนจากจุดกำเนิดไปยังจุดสุดขีดอีกอันหนึ่งบนด้านนี้ หลังจากกระบวนการนี้ จำเป็นต้องทำการหมุนแบบพิเศษเพื่อเคลื่อนไปจุดสุดขีดเดิมที่อยู่ใกล้ ซึ่งกลายมาเป็นผลเฉลยที่เป็นไปได้พื้นฐานเริ่มต้นใหม่สำหรับวิธีซิมเพล็กซ์ ตัวอย่างเชิงตัวเลขแสดงให้เห็นว่าผลเฉลยที่เป็นไปได้พื้นฐานเริ่มต้นอันใหม่ให้ผลที่ดีกว่าจุดกำเนิดในแง่ของจำนวนการทำซ้ำและเวลาในการคำนวณ อย่างไรก็ตามในการที่จะได้ผลเฉลยที่เป็นไปได้พื้นฐานเริ่มต้นนั้นมา ก็มีความยุ่งยากในแง่ของจำนวนการทำซ้ำและเวลาในการคำนวณที่เพิ่มขึ้น เมื่อนำความยุ่งยากดังกล่าวมาพิจารณาร่วมด้วย วิธีซิมเพล็กซ์ที่ใช้การกระโดดตามจุดประสงค์จะให้ผลดีกว่าวิธีซิมเพล็กซ์แบบเดิมเพียงแค่ปัญหาที่มีตัวแปร 2 - 5 ตัวและเงื่อนไขบังคับไม่เกิน 1000 เงื่อนไขทั้งในแง่ของจำนวนการทำซ้ำและเวลาในการคำนวณโดยรวม นอกจากนี้ วิธีซิมเพล็กซ์ที่ใช้การกระโดดตามจุดประสงค์จะใช้เวลาในการคำนวณน้อยกว่าวิธีซิมเพล็กซ์แบบเดิมถึงแม้ว่าจำนวนการทำซ้ำจะมากกว่า สำหรับส่วนใหญ่ของปัญหาที่มีจำนวนเงื่อนไขบังคับอย่างน้อย 300 จนถึง 1000 ตัว และจำนวนตัวแปรอย่างน้อย 40 ตัวแต่ไม่เกินจำนวนของเงื่อนไขบังคับในแต่ละปัญหานั้น

ภาควิชา	คณิตศาสตร์และวิทยาการคอมพิวเตอร์	ลายมือชื่อนิสิต	.....
สาขาวิชา	คณิตศาสตร์ประยุกต์และวิทยาการคณนา	ลายมือชื่อ อ.ที่ปรึกษาหลัก	.....
ปีการศึกษา	2559	ลายมือชื่อ อ.ที่ปรึกษาร่วม	.....

5771972723 : MAJOR APPLIED MATHEMATICS AND COMPUTATIONAL SCIENCE
KEYWORDS: LINEAR PROGRAMMING, SIMPLEX METHOD, OBJECTIVE JUMP, INITIAL BASIC FEASIBLE SOLUTION.

NUTCHA YAWILA: Simplex method with objective jump. ADVISOR: ASST. PROF. BOONYARIT INTIYOT, Ph.D., CO-ADVISOR: ASST. PROF. KRUNG SINAPIROMSARAN, Ph.D., 114 pp.

We propose an idea of “objective jump”, a novel approach to obtain an initial basic feasible solution from the origin point before starting the simplex method. For a linear programming problem expressed in the canonical form, if its right-hand-side vector is non-negative, the origin point is guaranteed to be in the feasible region and is normally used as the starting point of the simplex method. Our objective jump procedure aims to start at another basic feasible solution by jumping from the origin along the objective gradient direction. This procedure requires additional constraints to construct a sub-LP problem with a smaller feasible region where an additional edge of this smaller region is formed such that it aligns with the gradient vector of the objective function. Then, a special matrix manipulation is performed to move from the origin to another extreme point on this edge. After this procedure is done, some special pivots are required to move to a nearby extreme point, which becomes our new initial basic feasible solution for the simplex method. The numerical examples have shown that the new initial basic feasible solutions outperformed the origin point in terms of the number of iterations and the running time. However, there is a price to pay in terms of additional iterations and running time to obtain such initial basic feasible solutions. When taking the price into consideration, the simplex method with objective jump outperformed the traditional simplex method only on problems with 2 – 5 variables and no more than 1000 constraints in terms of both the total number of iterations and the running time. In addition, the simplex method with objective jump performed with less running time than the traditional simplex although the number of iterations was larger for most cases when the number of constraints is at least 300 up to 1000 and the number of variables is at least 40 but does not exceed the number of constraints for each problem.

Department:	Mathematics and Computer Science	Student's Signature	.....
		Advisor's Signature	.....
Field of Study:	Applied Mathematics and Computational Science	Co-Advisor's Signature	.....
Academic Year:	2016		

ACKNOWLEDGEMENTS

First of all, I would like to exceptionally thank my advisor, Assistant Professor Dr. Boonyarit Intiyot and my co-advisor, Assistant Professor Dr. Krung Sinapiromsaran for their advice and helps throughout the course of this thesis. This thesis would not have been completed without all the support that I have always received from them.

Also, thanks to my thesis committees, Associate Professor Dr. Pornchai Satravaha, Assistant Professor Dr. Phantipa Thipwiwatpotjana and my external examiner, Associate Professor Dr. Peerayuth Charnsethikul, Department of Industrial Engineering, Faculty of Engineering, Kasetsart University. Moreover, I would like to thank all lecturers for teaching knowledge not only methodologies in research but also many other methodologies in life.

In addition, I am grateful for my parents who always support me no matter how far away I live and thanks back to all my lovely friends who have cheered me up in several hard times.

Finally, I also would like to acknowledge Applied Mathematics and Computational Science Program, Faculty of Science, Chulalongkorn University for their scholarships that allow me to be here.

CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

A linear programming problem is a problem of maximizing or minimizing a linear function (such as maximum profit or minimize cost) based on a set of linear constraints to ensure the best outcome of a mathematical model. Linear programming is a special case of mathematical programming, sometimes known as linear optimization.

There are several fields of study that linear programming can be applied such as business, economic and industries including transportation, manufacturing, and telecommunications. Moreover, linear programming is heavily used in microeconomics and business management such as planning, production, technology and related issues. Typically, most companies want to maximize the profit or minimize the cost with limited resources that can be casted as a linear programming problem. This gives rise to the popular use of linear programming in the field of optimization.

A linear programming problem can be solved by one of the most effective algorithms called the simplex method. It performs very well in practice on a problem of small to medium size. In general, the simplex method obtains an optimal solution by moving progressively from one extreme point to a better adjacent extreme point in the feasible region. An important step in solving the linear programming problem with the simplex method is known as a pivot rule. In 1947, Dantzig [4] suggested the most negative rate of change rule for the minimized problems. Since the Dantzig's rule could not guarantee the termination of the simplex method, a new pivot that avoids cycling, in case of degeneracy was introduced by Bland [2]. In 1973, Klee and Minty [9] showed that the Dantzig's simplex method was not a polynomial-time algorithm when applied to their special problems. Moreover, many pivot rules were proposed to improve the speed and the number of iterations such as the steepest edge pivot rule [5], the devex rule [6] and the largest-distance pivot rule [11] which avoid visiting all extreme points. These effect the number of iterations required for solving linear programming problems. Furthermore, the polynomial-time algorithm for solving a large linear programming problem was introduced called the interior point method (IPM) [12]

which finds an optimal solution by moving through an interior of the feasible region. Currently, a large linear programming problem is commonly solved by the interior point method. Even though, the solution finding method by the interior point method is very different than the simplex one. There is no dominant method for solving a linear programming problem in general.

1.1 Definition of linear programming

A linear programming model is an optimization model of maximizing or minimizing the linear function subject to linear equality or inequality constraints. It can be expressed in the canonical form of the maximize problem as:

$$\begin{aligned}
\max \quad & c_1x_1 + c_2x_2 + \cdots + c_jx_j + \cdots + c_nx_n \\
\text{s.t.} \quad & a_{11}x_1 + a_{12}x_2 + \cdots + a_{1j}x_j + \cdots + a_{1n}x_n \le b_1 \\
& \vdots \qquad\qquad \vdots \qquad\qquad \vdots \\
& a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{ij}x_j + \cdots + a_{in}x_n \le b_i \\
& \vdots \qquad\qquad \vdots \qquad\qquad \vdots \\
& a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mj}x_j + \cdots + a_{mn}x_n \le b_m \\
& x_1, x_2, x_3, \ldots, x_j, \ldots, x_n \ge 0.
\end{aligned}$$

Known coefficients of the linear programming model include cost coefficients c_j for $j = 1, 2, 3, \ldots, n$, coefficients a_{ij} for $i = 1, 2, 3..., m,\ j = 1, 2, 3..., n$ and the right-hand-sided value b_i for $i = 1, 2, 3..., m$. The unknown variables are decision variables x_j for $j = 1, 2, 3..., n$. This linear programming model can be written in summation format as

$$\begin{aligned}
\max \quad & \sum_{j=1}^{n} c_j x_j \qquad\qquad & & \min \quad \sum_{j=1}^{n} c_j x_j \\
\text{s.t.} \quad & \sum_{j=1}^{n} a_{ij} x_j \le b_i \qquad & \text{or} \qquad & \text{s.t.} \quad \sum_{j=1}^{n} a_{ij} x_j \ge b_i \\
& x_j \ge 0 & & x_j \ge 0
\end{aligned}$$

for $i = 1, 2, 3..., m$ and $j = 1, 2, 3..., n$.

1.1.1 Standard form

A linear programming model is said to be in the standard form if all constraints are equalities and all variables are nonnegative; i.e.,

$$\begin{aligned} \max \ & \sum_{j=1}^{n} c_j x_j \\ \text{s.t.} \ & \sum_{j=1}^{n} a_{ij} x_j = b_i \\ & x_j \geq 0 \end{aligned} \qquad \text{or} \qquad \begin{aligned} \min \ & \sum_{j=1}^{n} c_j x_j \\ \text{s.t.} \ & \sum_{j=1}^{n} a_{ij} x_j = b_i \\ & x_j \geq 0, \end{aligned}$$

for $i = 1, 2, 3..., m$ and $j = 1, 2, 3..., n.$

Additionally, the inequality constraints $\sum_{j=1}^{n} a_{ij} x_j \geq b_i$ can be converted to equality constraints by subtracting the nonnegative surplus variable s_i from the summation leading to $\sum_{j=1}^{n} a_{ij} x_j - s_i = b_i$ and $s_i \geq 0$. On the other hand, inequality constraints $\sum_{j=1}^{n} a_{ij} x_j \leq b_i$ can be converted to equality constraints by adding the nonnegative slack variable s_i to the summation leading to $\sum_{j=1}^{n} a_{ij} x_j + s_i = b_i$ and $s_i \geq 0.$

1.1.2 Matrix notation

The linear programming problem can be written in the matrix form as

$$\begin{aligned} \max \ & z = \mathbf{c}^{\mathrm{T}} \mathbf{x} \\ \text{s.t.} \ & \mathbf{A}\mathbf{x} \leq \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0}, \end{aligned}$$

where $\mathbf{A}$ is an $m \times n$ matrix. The vectors $\mathbf{x}$, $\mathbf{c}$ are $n \times 1$ column vectors and $\mathbf{b}$ is the $m \times 1$ column vector.

1.1.3 Geometric definitions

Geometric definitions of a linear programming problem are stated as follows.

1. Convex set:

   A set $\mathbf{X}$ in $\mathbb{R}^n$ is called a *convex set* if given any two points $\mathbf{x}_1$ and $\mathbf{x}_2$ in $\mathbf{X}$, then $\lambda\mathbf{x}_1 + (1-\lambda)\mathbf{x}_2 \in \mathbf{X}$ for each $\lambda \in [0,1]$.

2. Ray and direction:

   A *ray* is a collection of points of the form $\{\mathbf{x}_0 + \lambda\mathbf{d} : \lambda \geq 0\}$, where $\mathbf{d}$ is a nonzero vector. The point $\mathbf{x}_0$ is called the vertex of the ray, and $\mathbf{d}$ is the *direction* of the ray.

3. Direction of a convex set:

   A nonzero vector $\mathbf{d}$ is called a direction of the convex set $\mathbf{X}$ if for each $\mathbf{x}_0 \in \mathbf{X}$, the ray $\{\mathbf{x}_0 + \lambda\mathbf{d} : \lambda \geq 0\}$ also belongs to the set.

4. Extreme point:

   A point $\mathbf{x}$ in a convex set $\mathbf{X}$ is called an *extreme point* of $\mathbf{X}$ if $\mathbf{x} = \lambda\mathbf{x}_1 + (1-\lambda)\mathbf{x}_2$ with $\lambda \in (0,1)$ and $\mathbf{x}_1, \mathbf{x}_2 \in \mathbf{X}$, then $\mathbf{x} = \mathbf{x}_1 = \mathbf{x}_2$.

5. Extreme direction:

   An *extreme direction* of a convex set $\mathbf{X}$ is a direction of the set that cannot be represented as a positive combination of two distinct directions of the set. Vectors $\mathbf{d}_1$ and $\mathbf{d}_2$ are said to be distinct if $\mathbf{d}_1$ cannot be represented as a positive multiple of $\mathbf{d}_2$.

6. Hyperplane:

   A *hyperplane* $\mathbf{H} \in \mathbb{R}^n$ is a set of the form $\{\mathbf{x} : \mathbf{p}^{\mathrm{T}}\mathbf{x} = k\}$, where $\mathbf{p}$ is a nonzero vector in $\mathbb{R}^n$ and k is a scalar. This $\mathbf{p}$ is called the *normal* or the *gradient* of the hyperplane.

7. Half-Space:

   A hyperplane divides $\mathbb{R}^n$ into two regions, called *half-spaces*. A half-space is a collection of points of the form $\{\mathbf{x} : \mathbf{p}^{\mathrm{T}}\mathbf{x} \geq k\}$ or $\{\mathbf{x} : \mathbf{p}^{\mathrm{T}}\mathbf{x} \leq k\}$. The union of the two half-spaces is $\mathbb{R}^n$.

8. Polyhedral set:

   A polyhedral set represents a special case of a convex set. A *polyhedral set* or a *polyhedron* is the intersection of a finite number of half-

spaces. It can be represented by $\{\mathbf{x} : \mathbf{Ax} \leq \mathbf{b}\}$. A bounded polyhedral set is called a *polytope*.

9. Binding Constraint:

   Let $g(\mathbf{x}) \leq b$ be a constraint in an optimization problem. If at point $\mathbf{x_0} \in \mathbb{R}^n$ satisfies $g(\mathbf{x}_0) \leq b$, and $g(\mathbf{x}_0) = \mathbf{b}$, then the constraint $g(\mathbf{x}) \leq b$, is said to be *binding* at $\mathbf{x_0}$.

10. Face and Edge:

    Let $\mathbf{P}$ be a polyhedral set defined by $\mathbf{P} = \{\mathbf{x} \in \mathbb{R}^n \mid \mathbf{Ax} \leq \mathbf{b}\}$ where $\mathbf{A} \in \mathbb{R}^{m \times n}$ and $\mathbf{b} \in \mathbb{R}^m$. If $\mathbf{X} \subseteq \mathbf{P}$ is defined by a non-empty set of binding linearly independent hyperplanes, then $\mathbf{X}$ is a *face* of $\mathbf{P}$. An *edge* of a polyhedral is defined by any face of dimension one.

1.2 Related work and literature review

In 2014, Lawanyawut, Sirirat, and Yawila [10] proposed a new approach to obtain an initial basis for the simplex method by moving (or "jumping") from the origin in the direction of the objective gradient to an extreme point on the other side of the feasible region in 2 and 3-dimensional linear programming models.

The jump required additional constraints to construct a sub-problem, with a smaller feasible region where an edge was in line with the objective gradient. After starting at the origin, a number of pivots were performed to move along that edge and to a nearby extreme point, which became the initial basis for the simplex method. Once the initial basis was obtained, additional constraints were removed before starting the simplex method. The approach was empirically compared with the regular simplex method with the standard initial basis using a number of generated small-sized 2 and 3-dimensional linear programming problems. The performance improvement provided by this approach was shown in terms of the number of iterations.

Moreover, there are some literatures having an idea to improve the initial basis of the simplex method, for example, an improved initial basis for the simplex algorithm by Junior and Lins in 2005 [7]. This paper suggested an initial basis which determined the vertex in the feasible region that was closer to the optimal vertex than the initial

solution applied by the simplex method. The main idea presented one constraint that made the closest angle with the objective function. They considered which constraints of the primal problem:

$$\begin{aligned} \max\ & \mathbf{b}^{\mathrm{T}}\mathbf{y} \\ \text{s.t.}\ & \mathbf{A}^{\mathrm{T}}\mathbf{y}+\mathbf{s} = \mathbf{c} \\ & \mathbf{s} \geq \mathbf{0}, \end{aligned}$$

forming the basis of the dual problem. The angles between the gradient vectors to the hyperplanes defining to each constraint and gradient vector to the objective function was calculated by the angle $\theta(x_j) = \arccos \dfrac{(\mathbf{a}_j)^{\mathbf{T}}\mathbf{b}}{\|\mathbf{a}_j\| \cdot \|\mathbf{b}\|}$, $j = 1, 2, 3, ..., n,$ where x_j is the variable of primal problem. The vector $\mathbf{a}_j$ is the column vector of the matrix $\mathbf{A}$.

1.3 Objective of the thesis

In this thesis, a principal objective is to offer the concept and insights of a new approach for determining an efficient initial basis for the simplex method solving in the n dimensional linear programming problem.

1.4 Scope of the thesis

Maximizing a linear programming problem is considered in a canonical form (all constraints are of $\leq$ type) such that all cost coefficients in the objective function and the right-hand-side values of all constraints are nonnegative. This guarantees that the origin is in the feasible region and it is normally used as the starting point of the simplex method.

Following the first chapter, some related algebraic definitions and the simplex method are explained in Chapter 2. In Chapter 3, the initial concept of the thesis is discussed along geometric concepts of the linear programming problem. There are some definitions considered before the objective jump procedure, and some examples are illustrated in this chapter such as examples of Klee and Minty problems and other examples from the text books. Chapter 4 presents the computational results of testing the simplex method with objective jump on several problems of various sizes. The

results which include the average number of iterations and average running times are compared with the simplex method using Dantzig's pivot rule

CHAPTER 2
PRELIMINARIES

In this chapter, the main study of solving a linear programming problem is presented with some algebraic definitions, theoretical concepts and the simplex method [1].

2.1 Algebraic definitions

Some algebraic definitions used in this paper are

1. Zero vector:

   Zero vector $\mathbf{0}$ is the vector which all components equal to zero. Here, it also is called the *origin*.

2. (Standard) Basis vector:

   The vector $\mathbf{e}_i$ is a *(standard) basis vector* if all components equal to zero except the i^{th} position is 1.

3. Inner product:

   For any two vectors $\mathbf{a},\mathbf{b} \in \mathbb{R}^n$, the *inner product* or *dot product* of two vectors is defined as $\mathbf{a}\cdot\mathbf{b} = a_1b_1 + a_2b_2 + \cdots + a_nb_n = \sum_{j=1}^{n} a_jb_j$. Moreover, the inner product can be defined by $\mathbf{a}\cdot\mathbf{b} = \|\mathbf{a}\|\|\mathbf{b}\|\cos(\theta)$, where $\|\mathbf{a}\|, \|\mathbf{b}\|$ are the Euclidean norm of vectors $\mathbf{a}$ and $\mathbf{b}$, θ is the angle between $\mathbf{a}$ and $\mathbf{b}$. If $\mathbf{a}\cdot\mathbf{b} = 0$ then $\mathbf{a}$ and $\mathbf{b}$ are *orthogonal* and $\theta = 90^\circ$. The inner product of $\mathbf{a}$ and $\mathbf{b}$ can also be written in the matrix multiplication form as $\mathbf{a}^{\mathrm{T}}\mathbf{b}$.

4. Identity matrix:

   An $n \times n$ matrix $\mathbf{I}_n$ is an *identity matrix* if the diagonal elements are ones and other elements are all zeros.

5. Matrix inversion:

Let $\mathbf{A}, \mathbf{B}$ be the $n \times n$ square matrices. If $\mathbf{B}$ is the *inverse* of $\mathbf{A}$, then $\mathbf{AB} = \mathbf{I}_n = \mathbf{BA}$. The inverse matrix is denoted by $\mathbf{A}^{-1}$ and is unique. If $\mathbf{A}$ has the inverse, $\mathbf{A}$ is called *nonsingular*. Otherwise, $\mathbf{A}$ is called *singular*.

2.2 Simplex method

The simplex method is a method for solving a linear programming problem. This method is presented in 1947 by George B. Dantzig. A system of linear inequalities defines a polytope as a feasible region of a linear programming problem. The simplex method begins at a starting extreme point of the polytope and moves along the edges of this polytope to the adjacent extreme point. So that at each new extreme point, the objective function is improved until the optimal solution is reached, or else until the algorithm determines that the optimal value is unbounded.

The simplex method is based on the iterative improvement that starting a basic feasible solution at the origin without enumerating all extreme points. This method starts the operation on the standard form of the following linear programming problem.

$$\begin{aligned} \max \quad & z = \mathbf{c}^{\mathrm{T}}\mathbf{x} \\ \text{s.t.} \quad & \mathbf{Ax} = \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0}, \end{aligned}$$

where $\mathbf{A}$ is an $m \times n$ matrix with $\operatorname{rank}(\mathbf{A}) = m$ and $\mathbf{b}$ is a vector of size m.

2.1.1 Basic feasible solution

Consider the standard form of a linear programming model, let $\mathbf{A} = [\mathbf{B}, \mathbf{N}]$ where $\mathbf{B}$ is an $m \times m$ invertible matrix, called the basic matrix. The $m \times (n-m)$ matrix $\mathbf{N}$ is called the nonbasic matrix. Let $\mathbf{x} = \begin{bmatrix} \mathbf{x_B} \\ \mathbf{x_N} \end{bmatrix}$ be the solution of the system of linear equations $\mathbf{Ax} = \mathbf{b}$, where $\mathbf{x_N} = \mathbf{0}$ and $\mathbf{x_B} = \mathbf{B}^{-1}\mathbf{b}$, is called a basic solution. A basic solution $\mathbf{x}$ with the component $\mathbf{x_B} \geq \mathbf{0}$ is called a basic feasible solution.

2.1.2 Objective value

Suppose that a basic feasible solution is given by $\mathbf{x} = \begin{bmatrix} \mathbf{B}^{-1}\mathbf{b} \\ \mathbf{0} \end{bmatrix}$, then the objective value z is calculated by $z = \mathbf{c}^{\mathrm{T}}\mathbf{x} = \begin{bmatrix} \mathbf{c}_{\mathbf{B}}^{\mathrm{T}} & \mathbf{c}_{\mathbf{N}}^{\mathrm{T}} \end{bmatrix} \begin{bmatrix} \mathbf{B}^{-1}\mathbf{b} \\ \mathbf{0} \end{bmatrix} = \mathbf{c}_{\mathbf{B}}^{\mathrm{T}}\mathbf{B}^{-1}\mathbf{b}$.

2.1.3 Theoretical concepts

1. Existence of an optimal solution

   Assume that the feasible region is not empty. Then a finite optimal solution exists if and only if $\mathbf{c}^{\mathrm{T}}\mathbf{d}_j \geq \mathbf{0}$ for $j = 1, 2, 3, ..., l$, where $\mathbf{d}_1, \mathbf{d}_2, ..., \mathbf{d}_l$ are the extreme directions of the feasible region. Otherwise, the optimal solution value is unbounded.

2. Correspondence between basic feasible solutions and extreme points

   The collection of extreme points are equivalent to the collection of basic feasible solutions. In other words, a point is a basic feasible solution if and only if it is an extreme point. Moreover, for every extreme point, there is a corresponding basis. Conversely, for every basis, there is a corresponding (unique) extreme point.

3. Existence of an optimal solution and optimal extreme point

   If an optimal solution exists, then an optimal extreme point or equivalently an optimal basic feasible solution exists.

2.1.4 Tableau form

A linear programming model can be rewritten in part of basic and nonbasic variables; i.e.

$$\max \quad z$$

$$\text{s.t} \quad z - \mathbf{c}_{\mathbf{B}}^{\mathrm{T}}\mathbf{x}_{\mathbf{B}} - \mathbf{c}_{\mathbf{N}}^{\mathrm{T}}\mathbf{x}_{\mathbf{N}} = \mathbf{0} \tag{2.1}$$

$$\mathbf{B}\mathbf{x}_{\mathbf{B}} + \mathbf{N}\mathbf{x}_{\mathbf{N}} = \mathbf{b} \tag{2.2}$$

$$\mathbf{x}_{\mathbf{B}}, \mathbf{x}_{\mathbf{N}} \geq \mathbf{0}.$$

By multiplying $\mathbf{B}^{-1}$ to equation (2.2), we get

$$\mathbf{x_B} + \mathbf{B}^{-1}\mathbf{N}\mathbf{x_N} = \mathbf{B}^{-1}\mathbf{b}. \tag{2.3}$$

Multiplying (2.3) by $\mathbf{c_B^T}$,

$$\mathbf{c_B^T}\mathbf{x_B} + \mathbf{c_B^T}\mathbf{B}^{-1}\mathbf{N}\mathbf{x_N} = \mathbf{c_B^T}\mathbf{B}^{-1}\mathbf{b}. \tag{2.4}$$

Adding to equation (2.1),

$$z - \mathbf{c_B^T}\mathbf{B}^{-1}\mathbf{b} + \mathbf{c_B^T}\mathbf{B}^{-1}\mathbf{N}\mathbf{x_N} - \mathbf{c_N^T}\mathbf{x_N} = 0. \tag{2.5}$$

Then, we get

$$z + 0\mathbf{x_B} + (\mathbf{c_B^T}\mathbf{B}^{-1}\mathbf{N} - \mathbf{c_N^T})\mathbf{x_N} = \mathbf{c_B^T}\mathbf{B}^{-1}\mathbf{b}. \tag{2.6}$$

Currently, $\mathbf{x_N} = \mathbf{0}$, and by the equations (2.3) and (2.6), we obtain $\mathbf{x_B} = \mathbf{B}^{-1}\mathbf{b}$ and $z = \mathbf{c_B^T}\mathbf{B}^{-1}\mathbf{b}$. From the equations (2.3) and (2.6), the tableau with the current basic feasible solution represents in the following tableau.

	$\mathbf{x_N}$	$\mathbf{x_B}$	RHS	
z	$\mathbf{c_B^T}\mathbf{B}^{-1}\mathbf{N} - \mathbf{c_N^T}$	$\mathbf{0}$	$\mathbf{c_B^T}\mathbf{B}^{-1}\mathbf{b}$	*row0*
$\mathbf{x_B}$	$\mathbf{B}^{-1}\mathbf{N}$	$\mathbf{I}$	$\mathbf{B}^{-1}\mathbf{b}$	

For the above tableau, the objective row is referred to *row0.* The value of the objective function is $z = \mathbf{c_B^T}\mathbf{B}^{-1}\mathbf{b}$ and the basic feasible solutions is $\mathbf{B}^{-1}\mathbf{b}$. If $\mathbf{c_B^T}\mathbf{B}^{-1}\mathbf{N} - \mathbf{c_N^T} \geq \mathbf{0}$, then the current basic feasible solution is optimal. Otherwise, the current tableau is improved for some nonbasic variable $\mathbf{x_N}$.

2.1.5 Nonbasic variable space

From the equation (2.3),

$$\begin{aligned}\mathbf{x_B} &= \mathbf{B}^{-1}\mathbf{b} - \mathbf{B}^{-1}\mathbf{N}\mathbf{x_N} \\ &= \mathbf{B}^{-1}\mathbf{b} - \sum_{j \in J} \mathbf{B}^{-1}\mathbf{a}_j x_j.\end{aligned}$$

Let $\bar{\mathbf{b}} = \mathbf{B}^{-1}\mathbf{b}$ and $\mathbf{y}_j = \mathbf{B}^{-1}\mathbf{a}_j$. Thus, $\mathbf{x_B} = \bar{\mathbf{b}} - \sum_{j \in J} \mathbf{y}_j x_j$ where $\mathbf{a}_j$ is the j^{th} nonbasic column of the matrix $\mathbf{A}$ and J is the current set of the indices of the nonbasic variables.

From the equation (2.6), we have the objective function value:

$$\begin{aligned} z &= \mathbf{c}_\mathbf{B}^{\mathrm{T}}\mathbf{B}^{-1}\mathbf{b} - \mathbf{c}_\mathbf{B}^{\mathrm{T}}\mathbf{B}^{-1}\mathbf{N}\mathbf{x}_\mathbf{N} + \mathbf{c}_\mathbf{N}^{\mathrm{T}}\mathbf{x}_\mathbf{N} \\ &= z_0 - \sum_{j \in J}(z_j - c_j)x_j, \end{aligned}$$

where $z_0 = \mathbf{c}_\mathbf{B}^{\mathrm{T}}\mathbf{B}^{-1}\mathbf{b}$ and $z_j - c_j = \mathbf{c}_\mathbf{B}^{\mathrm{T}}\mathbf{y}_j - c_j$ for each nonbasic variable.

A linear programming model can be rewritten in terms of the nonbasic variables using the foregoing transformation:

$$\begin{aligned} \max \quad & z = z_0 - \sum_{j \in J}(z_j - c_j)x_j \\ \text{s.t.} \quad & \sum_{j \in J} \mathbf{y}_j x_j \le \bar{\mathbf{b}} \\ & x_j \ge 0,\ j \in J. \end{aligned} \tag{2.7}$$

The feasible region of an LP model is defined in terms of $n+m$ intersecting half-spaces which composes of m inequality constraints in equation (2.7), and n nonnegativity constraints.

2.1.6 Entering and leaving variables

From tableau of the simplex method, an entering variable is chosen from among the columns containing $\mathbf{c}_\mathbf{B}^{\mathrm{T}}\mathbf{B}^{-1}\mathbf{N} - \mathbf{c}_\mathbf{N}^{\mathrm{T}}$ and matrix $\mathbf{B}^{-1}\mathbf{N}$. When the column with $\mathbf{c}_\mathbf{B}^{\mathrm{T}}\mathbf{B}^{-1}\mathbf{N} - \mathbf{c}_\mathbf{N}^{\mathrm{T}} < 0$ is chosen, then the simplex method will choose a leaving variable by performing the minimum ratio test on the chosen column and the right-hand-side (RHS) column. Choosing entering variables and leaving variables is summarized below:

1. Entering variable: x_k enters, if $z_k - c_k < 0$ such that $|z_k - c_k| = \max_{j \in J} |z_j - c_j|$.
2. Leaving variable: x_{B_r} leaves, if $\dfrac{\bar{b}_r}{y_{rk}} = \min_{1 \le i \le m} \left\{ \dfrac{\bar{b}_i}{y_{ik}} \middle| y_{ik} > 0 \right\}$.

2.1.7 Pivoting operation

By a pivoting operation, if x_k enters the basis and x_{B_r} leaves the basis, then y_{rk} is considered to be the pivot. This is shown in the table below.

	$\cdots\ x_j\ \cdots\ x_k\ \cdots$	$x_{B_1}\cdots x_{B_r}\ \cdots\ x_{B_m}$	RHS
z	$\cdots\ z_j-c_j\ \cdots\ z_k-c_k\ \cdots$	$0\ \cdots\ 0\ \cdots\ 0$	$\mathbf{c_B^T B^{-1} b}$
x_{B_1}	$\cdots\ y_{1j}\ \cdots\ y_{1k}\ \cdots$	$1\ \cdots\ 0\ \cdots\ 0$	$\bar{b}_1$
$\vdots$	$\vdots\quad\vdots$	$\vdots\quad\vdots\quad\vdots$	$\vdots$
x_{B_r}	$\cdots\ y_{rj}\ \cdots\ (y_{rk})\ \cdots$	$0\ \cdots\ 1\ \cdots\ 0$	$\bar{b}_r$
$\vdots$	$\vdots\quad\vdots$	$\vdots\quad\vdots\quad\vdots$	$\vdots$
x_{B_m}	$\cdots\ y_{mj}\ \cdots\ y_{mk}\ \cdots$	$0\ \cdots\ 0\ \cdots\ 1$	$\bar{b}_m$

Operating y_{rk} can be as follows:

1. Divide row r by y_{rk}.
2. Update the i^{th} row for $i=1,2,3...,m$ and $i\neq r$ by adding the new r^{th} row multiplied by $-y_{ik}$.
3. Update *row0* by adding to the new r^{th} row multiplied by $c_k - z_k$.

2.1.8 Simplex algorithm

Initial step:

Find and initial basic feasible solution. Form the following initial tableau:

	$\mathbf{x_N}$	$\mathbf{x_B}$	RHS
z	$\mathbf{c_B^T B^{-1} N - c_N^T}$	$\mathbf{0}$	$\mathbf{c_B^T B^{-1} b}$
$\mathbf{x_B}$	$\mathbf{B^{-1} N}$	$\mathbf{I}$	$\mathbf{B^{-1} b}$

Main steps

1. Let $|z_k - c_k| = \max_{j \in J} |z_j - c_j|$. If $z_k - c_k \geq 0$, then stop as the current solution is optimal. Otherwise, check column of $\mathbf{y}_k$. If $\mathbf{y}_k \leq \mathbf{0}$, then stop as the original linear programming problem has unbounded optimal objective value.
2. If $\mathbf{y}_k \not\leq \mathbf{0}$, determine the index r as a minimum ratio test:

$$\frac{\bar{b}_r}{y_{rk}} = \min_{1 \leq i \leq m} \left\{ \frac{\bar{b}_i}{y_{ik}} \middle| y_{ik} > 0 \right\}.$$

3. Update the tableau by pivoting at y_{rk}. Update the basic and nonbasic variables where x_k enters the basis and x_{B_r} leaves the basis, then repeat the main steps.

CHAPTER 3
SIMPLEX METHOD WITH OBJECTIVE JUMP

In this thesis, we propose a method for finding an initial basic feasible solution and designing a new direction of the path along the direction of the objective function to avoid visiting all vertices to reduce the number of iterations and running time. In this chapter, we discuss the main idea of the simplex method with objective jump and explains the process for our method. Finally, we illustrate examples of the simplex method with objective jump. For the remaining of the thesis, we will refer to the simplex method with objective jump as SOJU.

3.1 Main idea

The main idea of this thesis is to approach the new initial feasible basis for the simplex method. Initially, this approach creates the artificial edge before starting the simplex method. This edge is aligned with the direction of the objective function. (see in Figure 3.1) and created from the intersecting of new hyperplanes and a hyperplane of the constraint. After this step, this obtains the new point, called a jump point and a sub-problem which has smaller feasible region than the original one. This process is called “objective jump”. After that, the objective jump process pivots out from this jump point to the new adjacent point of the origin feasible region. Then, the new point will be the initial point of the simplex method.

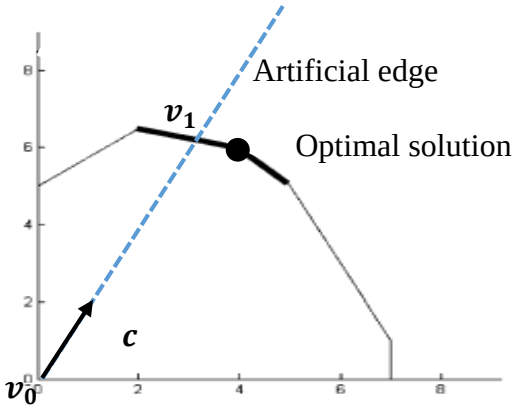


Figure 3.1: An artificial edge along with the direction of the objective function through the feasible region.

We solve a linear programming model that contains the origin in its feasible solution set.

$$\begin{aligned}
\max \quad & z = c_1x_1 + c_2x_2 + \ldots + c_nx_n \\
\text{s.t.} \quad & a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n \le b_1 \\
& a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n \le b_2 \\
& \vdots \qquad\qquad \vdots \qquad\qquad \vdots \\
& a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n \le b_m \\
& x_1, x_2, x_3, \ldots, x_n \ge 0,
\end{aligned}$$

where $b_1, b_2, b_3 \ldots, b_m \ge 0$.

This is equivalent to

$$\begin{aligned}
\max \quad & z = \sum_{j=1}^{n} c_j x_j \\
\text{s.t.} \quad & \sum_{j=1}^{n} a_{ij} x_j \le b_i \quad \text{for } i = 1, 2, 3, ..., m \\
& x_j \ge 0 \quad \text{for } j = 1, 2, 3, ..., n,
\end{aligned}$$

where $b_i \ge 0$.

3.2 Direction of the objective function

According to the scope of the thesis, a direction or gradient of the objective function is $\mathbf{c} \ge \mathbf{0}$. It can be divided into two cases:

Case I: all $c_j > 0$. In this case, the vector $\mathbf{c}$ makes acute angles with all basis vectors $\mathbf{e}_i$; i.e., $\mathbf{c}^{\mathrm{T}}\mathbf{e}_i > 0$.

Case II: Some $c_j = 0$ (but not all zero). In this case, the vector $\mathbf{c}$ is orthogonal to some basis vector $\mathbf{e}_i$; i.e., $\mathbf{c}^{\mathrm{T}}\mathbf{e}_i = 0$.

In a linear programming model with $n = 3$, two examples of these cases are illustrated in Figure 3.2. In the first example, the direction of the objective function $\mathbf{c}_1 = \begin{bmatrix} 2 \\ 2 \\ 2 \end{bmatrix}$ makes an acute angle ($< 90°$) with all basis vectors $\mathbf{e_1} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$, $\mathbf{e_2} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$ and

$\mathbf{e}_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$; i.e., $\mathbf{c}_1{}^{\mathrm{T}}\mathbf{e}_1 = 2 > 0$, $\mathbf{c}_1{}^{\mathrm{T}}\mathbf{e}_2 = 2 > 0$ and $\mathbf{c}_1{}^{\mathrm{T}}\mathbf{e}_3 = 2 > 0$. In the second example, the direction of the objective function $\mathbf{c}_2 = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}$ makes the right angle $(= 90°)$ with the basis vector $\mathbf{e}_3 \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$; i.e., $\mathbf{c}_2{}^{\mathrm{T}}\mathbf{e}_3 = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}^{\mathrm{T}} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = 0$ and acute angle with $\mathbf{e}_1$ and $\mathbf{e}_2$.

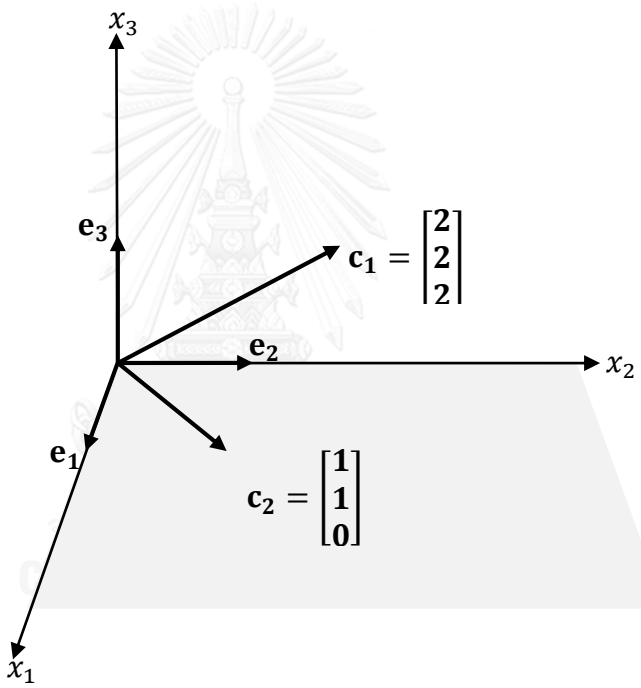


Figure 3.2: The directions of the objective function point into the feasible region and make acute or right angle with each axis.

3.3 Case I $(\text{All } c_j > 0)$

3.3.1 The gradient of all ones

Initially, the simple idea of the thesis uses the gradient of all ones as the simple gradient of the objective function which is denoted by $\mathbf{c}_{\{c_j=1\}} = \mathbf{1} \in \mathbb{R}^n$.

Definition 1.1 (Orthogonal objective vectors)

Let $\mathbf{c}_{\{c_j=1\}} = \mathbf{1} \in \mathbb{R}^n$ be a column vector of all ones and n be the number of variables. Define a set of $n-1$ column vectors $\mathbf{V} = \{\mathbf{v}_1, \mathbf{v}_2, ..., \mathbf{v}_i, ..., \mathbf{v}_{n-1}\}$ in $\mathbb{R}^n$ where the j^{th} element of a column vector $\mathbf{v}_i$ is given by

$$\left(\mathbf{v}_i\right)_j = \begin{cases} 1 & \text{if} \quad j < i+1 \\ -i & \text{if} \quad j = i+1 \\ 0 & \text{if} \quad j > i+1 \end{cases} \quad \text{, for } 1 \le i \le n-1,\ 1 \le j \le n. \qquad (3.1)$$

The vector $\mathbf{v}_i$ is called an *orthogonal objective vector*.

Properties

1) These vectors together with $\mathbf{c}_{\{c_j=1\}}$ are pairwise orthogonal; i.e.,

   $\mathbf{v}_i^{\mathrm{T}} \mathbf{c}_{\{c_j=1\}} = 0,\ \forall i = 1, 2, 3, \ldots, n-1,$ and $\mathbf{v}_i^{\mathrm{T}} \mathbf{v}_r = 0,\ \forall i \ne r$.

2) These vectors are also linearly independent.

In Figure 3.3, for $n = 2,$ the orthogonal objective vector $\mathbf{v}_1 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$ is perpendicular with the gradient of the objective function $\mathbf{c} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$; i.e.,

$\mathbf{v}_1^{\mathrm{T}} \mathbf{c} = \begin{bmatrix} 1 \\ -1 \end{bmatrix}^{\mathrm{T}} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = 0$. For $n = 3,$ the orthogonal objective vectors $\mathbf{v}_1 = \begin{bmatrix} 1 \\ -1 \\ 0 \end{bmatrix}, \mathbf{v}_2 = \begin{bmatrix} 1 \\ 1 \\ -2 \end{bmatrix}$ and the direction of the objective function $\mathbf{c} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$ are pairwise orthogonal; i.e.,

$$\mathbf{v}_1^{\mathrm{T}} \mathbf{v}_2 = \begin{bmatrix} 1 \\ -1 \\ 0 \end{bmatrix}^{\mathrm{T}} \begin{bmatrix} 1 \\ 1 \\ -2 \end{bmatrix} = 0,\ \mathbf{v}_1^{\mathrm{T}} \mathbf{c} = \begin{bmatrix} 1 \\ -1 \\ 0 \end{bmatrix}^{\mathrm{T}} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = 0,\ \text{and } \mathbf{v}_2^{\mathrm{T}} \mathbf{c} = \begin{bmatrix} 1 \\ 1 \\ -2 \end{bmatrix}^{\mathrm{T}} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = 0.$$

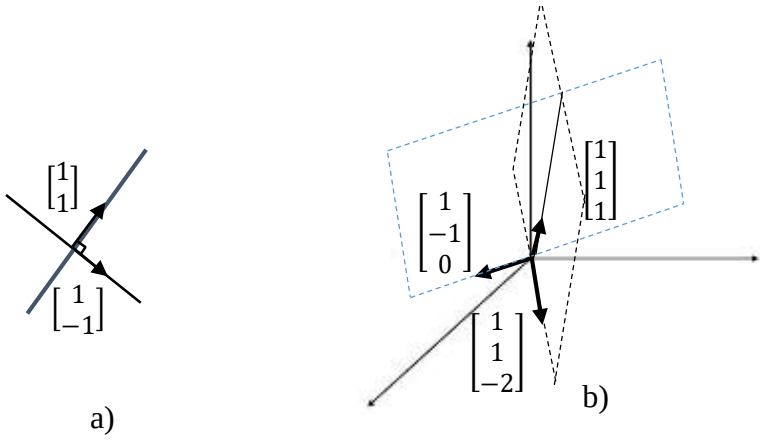


Figure 3.3: a) one orthogonal objective vectors for $n=2$ and b) two orthogonal objective vectors for $n=3$.

LP model with the simple gradient of the objective function

Let $\mathbf{c}_{\{c_j=1\}} = \mathbf{1} \in \mathbb{R}^n$, consider the following LP model:

$$\begin{aligned} \max z &= \mathbf{c}^{\mathrm{T}}_{\{c_j=1\}} \mathbf{y} \\ \text{s.t.} \quad \mathbf{A}\mathbf{y} &\le \mathbf{b} \\ \mathbf{y} &\ge \mathbf{0} \end{aligned} \tag{3.2}$$

where $\mathbf{y} \in \mathbb{R}^n$ is a vector of variables.

Artificial edge and orthogonal objective matrix

Our idea is to create the artificial edge that is aligned with the gradient of the objective function $\mathbf{c}_{\{c_j=1\}} = \mathbf{1}$ in order to jump from the origin point along this edge.

To form such edge, the intersecting of $n-1$ linearly independent hyperplanes are needed. The hyperplanes can be defined by the orthogonal objective vectors $\mathbf{v}_i$, $i=1,2,3,...,n-1$ and the edge can be obtained from the system of linear equations $\mathbf{v}_i^{\mathrm{T}}\mathbf{y}=0,\ i=1,2,3,...,n\text{-}1$. The system of equations is written in a matrix form as $\mathbf{T}_0\mathbf{y}=\mathbf{0}$ where the matrix

$$\mathbf{T}_0 = \begin{pmatrix} \mathbf{v}_1^{\mathrm{T}} \\ \mathbf{v}_2^{\mathrm{T}} \\ \mathbf{v}_3^{\mathrm{T}} \\ \vdots \\ \mathbf{v}_{n-1}^{\mathrm{T}} \end{pmatrix} = \begin{pmatrix} 1 & -1 & 0 & 0 & \dots & 0 \\ 1 & 1 & -2 & 0 & \dots & 0 \\ 1 & 1 & 1 & -3 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & 1 & 1 & \dots & -(n-1) \end{pmatrix}_{(n-1)\times n} \tag{3.3}$$

The matrix $\mathbf{T}_0$ is called the *orthogonal objective matrix.*

Claim 1.1 The system of equations $\mathbf{T}_0\mathbf{y} = \mathbf{0}$ has the general solution of the form

$$y_1 = y_2 = \ldots = y_j = \ldots = y_{n-1} = y_n \in \mathbb{R}.$$

Proof.

The system of equations $\mathbf{T}_0\mathbf{y} = \mathbf{0}$ can be expressed as

$\sum_{j=1}^{k-1} y_j - (k-1)\, y_k = 0,\ k = 2,3,4,...,n$ and this is equivalent to $\sum_{j=1}^{k-1} y_j = (k-1)\, y_k$. Hence, the system of equations can be rewritten as:

$$\begin{array}{lcll}
y_1 & = & y_2 & \cdots(1) \\
y_1 + y_2 & = & 2y_3 & \cdots(2) \\
y_1 + y_2 + y_3 & = & 3y_4 & \cdots(3) \\
\vdots \quad \vdots \quad \vdots & & \vdots \quad \vdots & \vdots \\
y_1 + y_2 + y_3 + y_4 + \cdots + y_{n-2} & = & (n-2)\, y_{n-1} & \cdots(\text{p-2}) \\
y_1 + y_2 + y_3 + y_4 + \cdots + y_{n-2} + y_{n-1} & = & (n-1)\, y_n & \cdots(\text{p-1})
\end{array}$$

Initially, the equation (1) has $y_1 = y_2$.

Substitute $y_1 = y_2$ to the equation (2); $2y_2 = 2y_3$ then $y_2 = y_3$.

Substitute $y_1 = y_2 = y_3$ to the equation (3); $3y_3 = 3y_4$ then $y_3 = y_4$.

Repeat the substitution in the similar manner. Therefore, the general solution of these equations is $y_1 = y_2 = \ldots = y_j = \ldots = y_{n-1} = y_n$ ■

The artificial edge is represented by this general solution $y_1 = y_2 = \ldots = y_j = \ldots = y_{n-1} = y_n$. It will be used to define the jump point which is the point along the line $y_1 = y_2 = \ldots = y_j = \ldots = y_{n-1} = y_n$ such that it is feasible and at the boundary of the feasible region.

Definition 2.1 (Jump point)

For the LP model (3.2), let $\sum_{j=1}^{n} a_{ij} y_j \le b_i$ be the constraints such that $b_i \ge 0$ and not all $b_i = 0$. A point $\mathbf{y} = \left[y_j \right] \in \mathbb{R}^n$ is said to be the *jump point* if

Condition (1): The point **y** satisfies the condition

$$y_1 = y_2 = \ldots = y_j = \ldots = y_{n-1} = y_n = \mu > 0 \text{ for some scalar } \mu .$$

Condition (2): The point **y** satisfies at least one equality constraint $\sum_{j=1}^{n} a_{qj} y_j + s_q = b_q$

such that the slack variable $s_q = 0$ and $b_q > 0$.

Condition (3): The point **y** is feasible for the LP model (3.2).

Preposition 2.2

For the LP model (3.2), let $\sum_{j=1}^{n} a_{ij} y_j \le b_i$ be constraints such that $b_i \ge 0$ and not all $b_i = 0$. If there exists some i such that $\sum_{j=1}^{n} a_{ij} > 0$ and $b_i > 0$, the jump point is $\mathbf{y}^J = \left[y_j^J \right] \in \mathbb{R}^n$ where each component $y_j^J = \mu = \min_i \left\{ \frac{b_i}{\sum_{j=1}^{n} a_{ij}} > 0 \;\middle|\; \sum_{j=1}^{n} a_{ij} > 0 \right\}$.

Otherwise, the jump point does not exist.

Proof.

After adding slacks variable $s_i \ge 0$ to constraints $\sum_{j=1}^{n} a_{ij} y_j \le b_i$ such that $b_i \ge 0$,

the constraints become $\sum_{j=1}^{n} a_{ij} y_j + s_i = b_i$.

Case 1

Suppose that for some r, $\sum_{j=1}^{n} a_{rj} > 0$ and $b_r > 0$.

Let $q = \arg\min_r \left\{ \frac{b_r}{\sum_{j=1}^{n} a_{rj}} > 0 \;\middle|\; \sum_{j=1}^{n} a_{rj} > 0 \right\}$ and $\mathbf{y} = \left[y_j \right] \in \mathbb{R}^n$ such that $y_j = \mu = \frac{b_q}{\sum_{j=1}^{n} a_{qj}}$.

Clearly, the point **y** satisfies condition (1) in Definition 2.1.

Moreover, the point $\mathbf{y}$ satisfies condition (2) since substituting $y_j = \mu = \frac{b_q}{\sum_{j=1}^{n} a_{qj}}$ into the constraint $\sum_{j=1}^{n} a_{qj} y_j + s_q = b_q$ will result in $s_q = 0$.

The point $\mathbf{y}$ also satisfies all constraints which can be explained as follows.

For the constraint i such that $\sum_{j=1}^{n} a_{ij} > 0$,

$$\sum_{j=1}^{n} a_{ij} y_j = \left(\sum_{j=1}^{n} a_{ij} \right) \mu = \left(\sum_{j=1}^{n} a_{ij} \right) \frac{b_q}{\sum_{j=1}^{n} a_{qj}} \leq \left(\sum_{j=1}^{n} a_{ij} \right) \frac{b_i}{\sum_{j=1}^{n} a_{ij}} = b_i .$$

For the constraint i such that $\sum_{j=1}^{n} a_{ij} \leq 0$, $\sum_{j=1}^{n} a_{ij} y_j = \sum_{j=1}^{n} a_{ij} \mu \leq 0 \leq b_i$. Therefore, the point $\mathbf{y}$ is feasible and satisfies condition (3) in Definition 2.1.

Hence, the point $\mathbf{y}$ is the jump point for the LP model (3.2).

Case 2

Suppose $\sum_{j=1}^{n} a_{ij} \leq 0$ or $b_i = 0$ for all $i = 1, 2, 3, ..., m$. Assume a jump point $\mathbf{y}$ exist. Hence, from condition (1), $y_1 = y_2 = \ldots = y_j = \ldots = y_n = \mu$ for some $\mu > 0$.

Since $\mathbf{y}$ is feasible for any constraint i, we must have $\sum_{j=1}^{n} a_{ij} y_j \leq b_i$. Because $\sum_{j=1}^{n} a_{ij} y_j = \mu \sum_{j=1}^{n} a_{ij} \leq 0$ and $b_i \geq 0$, this implies $\sum_{j=1}^{n} a_{ij} y_j = b_i = 0$. Thus, condition (2) in definition 2.1 cannot be satisfied, which is a contradiction. Thus, a jump point does not exist. ■

3.3.2 General case I

In general case I, the gradient of the objective function is denoted by $\mathbf{c}_{\{c_j > 0\}} = \left[c_j \right] \in \mathbb{R}^n, c_j > 0$.

In this case, we consider the LP model:

$$\begin{aligned} \max\ & \mathbf{c}^{\mathrm{T}}_{\{c_j>0\}}\mathbf{x} \\ \text{s.t.}\ & \mathbf{A}\mathbf{x} \le \mathbf{b} \\ & \mathbf{x} \ge 0, \end{aligned} \tag{3.4}$$

where the vector of variables $\mathbf{x} \in \mathbb{R}^n$.

Transformation of orthogonal objective matrix

From the LP model (3.2), the artificial edge for the jump can be constructed through the system of equations $\mathbf{T}_0\mathbf{y} = \mathbf{0}$ where $\mathbf{T}_0$ is defined in (3.3). The similar system for the model (3.4) can also be constructed. Notice that the model (3.4) can be transformed into (3.2) by setting the variable $y_j = c_j x_j$. Thus, $\mathbf{c}^{\mathrm{T}}_{\{c_j>0\}}\mathbf{x} = \sum_{j=1}^{n} c_j x_j = \sum_{j=1}^{n} 1(c_j x_j) = \sum_{j=1}^{n} 1 y_j = \mathbf{1}^{\mathrm{T}}\mathbf{y}$.

Let $a'_{ij} = \frac{a_{ij}}{c_j}$. Hence, $\sum_{j=1}^{n} a_{ij}x_j = \sum_{j=1}^{n} \frac{a_{ij}}{c_j} c_j x_j = \sum_{j=1}^{n} \frac{a_{ij}}{c_j} y_j = \sum_{j=1}^{n} a'_{ij} y_j$. Therefore, the model (3.4) is equivalent to

$$\begin{aligned} \max\ & \mathbf{1}^{\mathrm{T}}\mathbf{y} \\ \text{s.t.}\ & \mathbf{A}'\mathbf{y} \le \mathbf{b} \\ & \mathbf{y} \ge 0, \end{aligned} \tag{3.5}$$

where the matrix $\mathbf{A}' = \left[a'_{ij} \right] \in \mathbb{R}^{m \times n}$.

Suppose the orthogonal objective matrix is $\mathbf{T}_0 = [t_{ij}] \in \mathbb{R}^{(n-1)\times n}$. Consider the system of equations $\mathbf{T}_0\mathbf{y} = \mathbf{0}$ for the LP model (3.5). For any constraint i, $\sum_{j=1}^{n} t_{ij} y_j = \sum_{j=1}^{n} t_{ij}\left(c_j x_j\right) = \sum_{j=1}^{n} \left(t_{ij} c_j\right) x_j = 0$. Therefore, if we set $\mathbf{T} = [c_j t_{ij}]$, the system of equations $\mathbf{T}_0\mathbf{y} = \mathbf{0}$ is equivalent to $\mathbf{T}\mathbf{x} = \mathbf{0}$; i.e.,

$$\begin{bmatrix} c_1 & -c_2 & 0 & 0 & \cdots & 0 \\ c_1 & c_2 & -2c_3 & 0 & \cdots & 0 \\ c_1 & c_2 & c_3 & -3c_4 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ c_1 & c_2 & c_3 & c_4 & \cdots & -(n-1)c_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{3.6}$$

Note that the row vectors in the matrix $\mathbf{T}$ may no longer be pairwise orthogonal.

Jump point

According to Preposition 2.2, the jump point of the LP model (3.5) is $\mathbf{y}^J = [y_j^J] \in \mathbb{R}^n$ where $y_j^J = \min_i \left\{ \frac{b_i}{\sum_{j=1}^{n} a'_{ij}} > 0 \;\middle|\; \sum_{j=1}^{n} a'_{ij} > 0 \right\} = \mu'$ provided that there are some i such that $\sum_{j=1}^{n} a'_{ij} > 0$ and $b_i > 0$. This jump point can be translated to the jump point of the LP model (3.4) as $\mathbf{x}^J = [x_j^J] \in \mathbb{R}^n$ where $x_j^J = \frac{\mu'}{c_j} = \frac{1}{c_j}\left[\min_i \left\{ \frac{b_i}{\sum_{j=1}^{n} \frac{a_{ij}}{c_j}} > 0 \;\middle|\; \sum_{j=1}^{n} \frac{a_{ij}}{c_j} > 0 \right\} \right]$.

For example 1,

$$\begin{aligned} \max \quad & x_1 + x_2 \\ \text{s.t.} \quad & x_1 + x_2 \le 1 \\ & -x_1 - x_2 \le 2 \\ & -x_1 - x_2 \le 3 \\ & x_1, x_2 \ge 0. \end{aligned}$$

Notice that $c_1, c_2 = 1$. Then, we consider each constraints such that $a_{i1} + a_{i2} > 0$ where $i = 1,2,3$. Since $a_{11} + a_{12} = 1 + 1 = 2 > 0$, so that the jump point is $(x_1^J, x_2^J) = \left(\frac{1}{2}, \frac{1}{2}\right)$.

This jump point is illustrated in Figure 3.4.

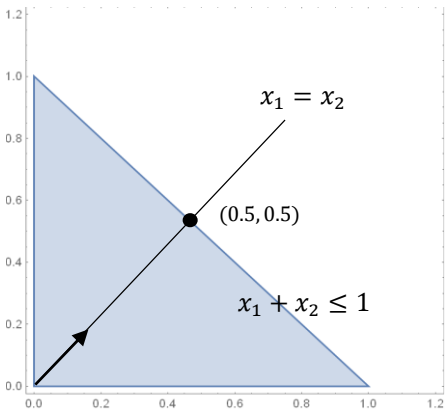


Figure 3.4: An example of a Jump point (0.5, 0.5).

For example 2,

$$\begin{aligned} \max \quad & x_1 + x_2 \\ \text{s.t.} \quad & x_1 - 2x_2 \le 4 \\ & -x_1 - x_2 \le 3 \\ & x_1, x_2 \ge 0 \end{aligned}$$

Notice that $c_1, c_2 = 1$. Since $a_{11} + a_{12} = 1 - 2 = -1 < 0$ and $a_{21} + a_{22} = -1 - 1 = -2 < 0$, so that the jump point does not exist for this example. Furthermore, the region is also unbounded. This is illustrated in Figure 3.5.

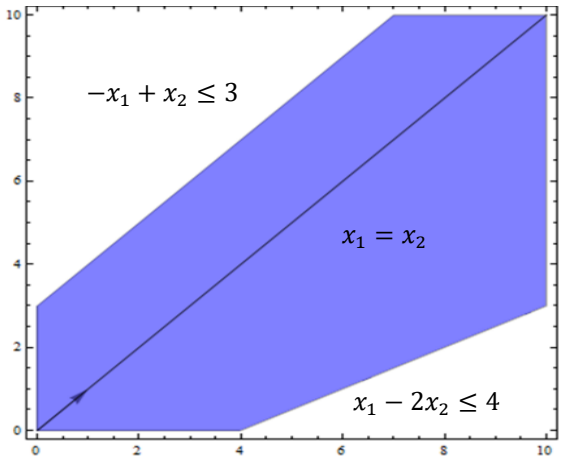


Figure 3.5: An example where a jump point does not exist.

Sub-LP model

To incorporate the artificial edge $\mathbf{Tx} = \mathbf{0}$ into the model (3.4) where $\mathbf{T} = \left[t'_{ij} \right] \in \mathbb{R}^{(n-1)\times n}$ defined in (3.6), this intersection of hyperplanes is transformed into the intersection of half-spaces $\sum_{j=1}^{n} t'_{ij} x_j \le 0$ or $\sum_{j=1}^{n} t'_{ij} x_j \ge 0$. The intersection of these half-spaces together with the feasible region in the model (3.4) form a smaller problem, which is called a *sub-LP model*. Note that there are 2^{n-1} possible sub-LP models due to the possibilities of $\sum_{j=1}^{n} t'_{ij} x_j \le 0$ or $\sum_{j=1}^{n} t'_{ij} x_j \ge 0$ for each artificial constraint. In this thesis, we only select one sub-LP model of the form:

$$\begin{aligned} \max \quad & z = \mathbf{c}_{\{c_j>0\}}^{T}\mathbf{x} \\ \text{s.t.} \quad & \mathbf{Ax} \le \mathbf{b} \\ & \mathbf{Tx} \le \mathbf{0} \\ & \mathbf{x} \ge \mathbf{0} \end{aligned} \tag{3.7}$$

Figure 3.6 illustrates an example of the feasible region of the LP model and its sub-LP model.

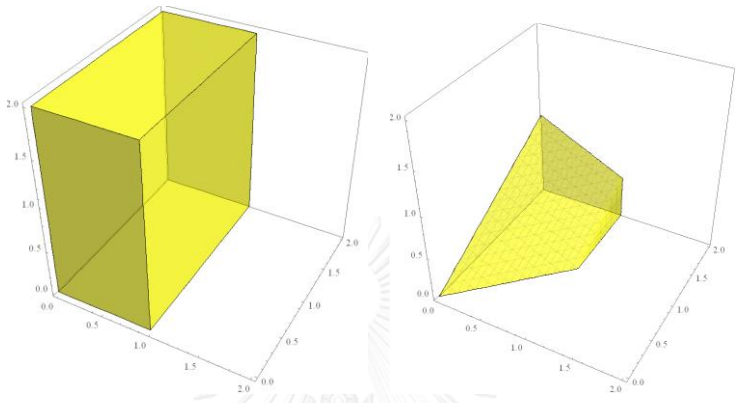


Figure 3.6: Feasible region of a) LP model and b) Sub-LP model with $n = 3$.

Objective jump model

To apply the simplex tableau to the sub-LP model (3.7), it need to be converted into a standard form. Assuming a jump point exists, let $q = \arg\min_i \left\{ \frac{b_i}{\sum_{j=1}^{n} \frac{a_{ij}}{c_j}} > 0 \;\middle|\; \sum_{j=1}^{n} \frac{a_{ij}}{c_j} > 0 \right\}$. Rearrange the indices $i = 1, 2, 3, ..., m$ of the constraints in $\mathbf{Ax} \le \mathbf{b}$ so that $m = q$. The problem (3.7) is turned into the objective jump model below.

$$\begin{aligned} \max \; & z = \sum_{j=1}^{n} c_j x_j \\ \text{s.t.} \; & \sum_{j=1}^{n} a_{ij} x_j + s_i = b_i && \text{for } i = 1, 2, ..., m-1 \\ & \sum_{j=1}^{n} a_{mj} x_j + s_m = b_m \\ & \sum_{j=1}^{i-m} c_j x_j - (i-m) c_{i-m+1} x_{i-m+1} + s_i = 0 && \text{for } i = m+1, m+2, ..., m+n-1 \\ & x_j, s_i \ge 0 && \text{for } j = 1, 2, ..., n \text{ and } i = 1, 2, ..., m+n-1 \end{aligned} \tag{3.8}$$

Basis of objective jump model

The main idea of SOJU is to start the simplex method from the jump point. Hence we want to determine the initial basis that represent the point. This can be done using the objective jump model (3.8) as follows. First, we choose a set of nonbasic variables as $\{s_m, s_{m+1}, s_{m+2}, \ldots, s_{m+n-1}\}$ and the remainders are the set of basic variables: $\{x_1, x_2, x_3, \ldots, x_n, s_1, s_2, \ldots, s_{m-1}\}$. Then, the current of basic (subscript $\mathbf{B}$) and nonbasic (subscript $\mathbf{N}$) are separated as follows:

$$\begin{aligned} \max \quad & z = \mathbf{0}_n^{\mathrm{T}}\mathbf{x}_{\mathbf{N}} + \begin{bmatrix} \mathbf{c}^{\mathrm{T}} & \mathbf{0}_{m-1} \end{bmatrix}\mathbf{x}_{\mathbf{B}} \\ \text{s.t.} \quad & \begin{bmatrix} \mathbf{0}_{(m-1)\times n} \\ \mathbf{I}_n \end{bmatrix}\mathbf{x}_{\mathbf{N}} + \begin{bmatrix} \mathbf{Q}_{(m-1)\times n} & \mathbf{I}_{m-1} \\ \mathbf{J}_{n\times n} & \mathbf{0}_{n\times(m-1)} \end{bmatrix}\mathbf{x}_{\mathbf{B}} = \begin{bmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \end{bmatrix} \\ & \mathbf{x}_{\mathbf{N}}, \mathbf{x}_{\mathbf{B}} \geq \mathbf{0}, \end{aligned} \tag{3.9}$$

where $\mathbf{Q} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1,n-1} & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2,n-1} & a_{2n} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{m-1,1} & a_{m-1,2} & \cdots & a_{m-1,n-1} & a_{m-1,n} \end{bmatrix}_{(m-1)\times n}$, $\mathbf{b}_1 = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_{m-1} \end{bmatrix}$,

$$\mathbf{J} = \begin{bmatrix} a_{m1} & a_{m2} & a_{m3} & \cdots & a_{mn} \\ c_1 & -c_2 & 0 & \cdots & 0 \\ c_1 & c_2 & -2c_3 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_1 & c_2 & c_3 & \cdots & -(n-1)c_n \end{bmatrix}_{n\times n}, \quad \mathbf{b}_2 = \begin{bmatrix} b_m \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}.$$

The notation of nonbasic and basic are shown in Table 3.1.

Table 3.1: Nonbasic and basic notations.

	Nonbasic	Basic
Variable: $\mathbf{x}$	$\mathbf{x}_{\mathbf{N}}$	$\mathbf{x}_{\mathbf{B}}$
Cost coefficient: $\mathbf{c}^{\mathrm{T}}$	$\mathbf{c}_{\mathbf{N}}^{\mathrm{T}} = \mathbf{0}_n^{\mathrm{T}}$	$\mathbf{c}_{\mathbf{B}}^{\mathrm{T}} = \begin{bmatrix} \mathbf{c}^{\mathrm{T}} & \mathbf{0}_{m-1} \end{bmatrix}$
Coefficient matrix: $\mathbf{A}$	$\mathbf{N} = \begin{bmatrix} \mathbf{0}_{(m-1)\times n} \\ \mathbf{I}_n \end{bmatrix}$	$\mathbf{B} = \begin{bmatrix} \mathbf{Q}_{(m-1)\times n} & \mathbf{I}_{m-1} \\ \mathbf{J}_{n\times n} & \mathbf{0}_{n\times(m-1)} \end{bmatrix}$

Table 3.2 represents the tableau of the current basis from the model (3.8). This tableau will be used in SOJU.

Table 3.2: Current basis of the objective jump tableau.

	s_m	s_{m+1}	$\cdots$	s_{m+n-1}	x_1	x_2	x_3	$\cdots$	x_n	s_1	$\cdots$	s_{m-1}	RHS
z	0	0	$\cdots$	0	c_1	c_2	c_3	$\cdots$	c_n	0	$\cdots$	0	0
	0	0	$\cdots$	0	a_{11}	a_{12}	a_{13}	$\cdots$	a_{1n}	1	$\cdots$	0	b_1
	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
	0	0	$\cdots$	0	$a_{m-1,1}$	$a_{m-1,2}$	$a_{m-1,3}$	$\cdots$	$a_{m-1,n}$	0	$\cdots$	1	b_{m-1}
	1	0	0	$\cdots$ 0	a_{m1}	a_{m2}	a_{m3}	$\cdots$	a_{mn}	0	$\cdots$	0	b_m
	0	1	0	$\cdots$ 0	c_1	$-c_2$	0	$\cdots$	0	0	$\cdots$	0	0
	0	0	1	$\cdots$ 0	c_1	c_2	$-2c_3$	$\cdots$	0	0	$\cdots$	0	$\vdots$
	$\vdots$	$\vdots$	$\vdots$	$\ddots$ $\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	0
	0	0	0	$\cdots$ 1	c_1	c_2	c_3	$\cdots$	$-(n-1)c_n$	0	$\cdots$	0	0

Inverse of the basic matrix in SOJU

The inverse of $\mathbf{B}$ can be determined by using Gauss-Jordan eliminations as:

$$
\begin{aligned}
[\mathbf{B}\,|\,\mathbf{I}] &= \left[\begin{array}{cc|cc} \mathbf{Q}_{(m-1)\times n} & \mathbf{I}_{m-1} & \mathbf{I}_{m-1} & \mathbf{0}_{(m-1)\times n} \\ \mathbf{J}_{n\times n} & \mathbf{0}_{n\times(m-1)} & \mathbf{0}_{n\times(m-1)} & \mathbf{I}_n \end{array}\right] \\
&\sim \left[\begin{array}{cc|cc} \mathbf{J}_{n\times n} & \mathbf{0}_{n\times(m-1)} & \mathbf{0}_{n\times(m-1)} & \mathbf{I}_n \\ \mathbf{Q}_{(m-1)\times n} & \mathbf{I}_{m-1} & \mathbf{I}_{m-1} & \mathbf{0}_{(m-1)\times n} \end{array}\right] \\
&\sim \left[\begin{array}{cc|cc} \mathbf{I}_n & \mathbf{0}_{n\times(m-1)} & \mathbf{0}_{n\times(m-1)} & \mathbf{J}^{-1} \\ \mathbf{Q} & \mathbf{I}_{m-1} & \mathbf{I}_{m-1} & \mathbf{0}_{(m-1)\times n} \end{array}\right] \\
&\sim \left[\begin{array}{cc|cc} \mathbf{I}_n & \mathbf{0}_{n\times(m-1)} & \mathbf{0}_{n\times(m-1)} & \mathbf{J}^{-1} \\ \mathbf{0}_{(m-1)\times n} & \mathbf{I}_{m-1} & \mathbf{I}_{m-1} & -\mathbf{Q}\mathbf{J}^{-1} \end{array}\right] \sim [\mathbf{I}\,|\,\mathbf{B}^{-1}].
\end{aligned}
$$

Therefore, the inverse of $\mathbf{B}^{-1} = \begin{bmatrix} \mathbf{0}_{n\times(m-1)} & \mathbf{J}^{-1} \\ \mathbf{I}_{m-1} & -\mathbf{Q}\mathbf{J}^{-1} \end{bmatrix}$.

Objective jump tableau

Objective jump tableau can be created by calculating the following simplex tableau,

	$\mathbf{x_N}$	$\mathbf{x_B}$	RHS
z	$\mathbf{c_B^T B^{-1} N - c_N^T}$	$\mathbf{0}_{m-1}^{\mathrm{T}}$	$\mathbf{c_B^T B^{-1} b}$
$\mathbf{x_B}$	$\mathbf{B^{-1} N}$	$\mathbf{I}_{m-1}$	$\mathbf{B^{-1} b}$

After calculation, the objective jump tableau is

	$\mathbf{x_N}$	$\mathbf{x_B}$	RHS
z	$\mathbf{c}^{\mathrm{T}}\mathbf{J}^{-1}$	$\mathbf{0}_{m-1}^{\mathrm{T}}$	$\mathbf{c}^{\mathrm{T}}\mathbf{J}^{-1}\mathbf{b}_2$
$\mathbf{x_B}$	$\begin{bmatrix}\mathbf{J}^{-1}\\ -\mathbf{Q}\mathbf{J}^{-1}\end{bmatrix}$	$\mathbf{I}_{m-1}$	$\begin{bmatrix}\mathbf{J}^{-1}\mathbf{b}_2\\ \mathbf{b}_1-\mathbf{Q}\mathbf{J}^{-1}\mathbf{b}_2\end{bmatrix}$

Recall that the jump point for (3.4) or (3.8) is given by $\mathbf{x}^J$ where

$x_i^J = \frac{1}{c_i}\cdot\frac{b_m}{\sum_{j=1}^{n}\frac{a_{mj}}{c_j}} = \frac{\mu}{c_i},\ i=1,2,3,\ldots,n$. Accordingly,

$$\mathbf{J}^{-1}\mathbf{b}_2 = \begin{bmatrix}(\mathbf{J}^{-1})_{11} & (\mathbf{J}^{-1})_{12} & (\mathbf{J}^{-1})_{13} & \cdots & (\mathbf{J}^{-1})_{1n}\\ (\mathbf{J}^{-1})_{21} & (\mathbf{J}^{-1})_{22} & (\mathbf{J}^{-1})_{23} & \cdots & (\mathbf{J}^{-1})_{2n}\\ (\mathbf{J}^{-1})_{31} & (\mathbf{J}^{-1})_{32} & (\mathbf{J}^{-1})_{33} & \cdots & (\mathbf{J}^{-1})_{3n}\\ \vdots & \vdots & \vdots & \ddots & \vdots\\ (\mathbf{J}^{-1})_{n1} & (\mathbf{J}^{-1})_{n1} & (\mathbf{J}^{-1})_{n3} & \cdots & (\mathbf{J}^{-1})_{nn}\end{bmatrix}\begin{bmatrix}b_m\\0\\0\\\vdots\\0\end{bmatrix} = \begin{bmatrix}(\mathbf{J}^{-1})_{11}b_m\\(\mathbf{J}^{-1})_{21}b_m\\(\mathbf{J}^{-1})_{31}b_m\\\vdots\\(\mathbf{J}^{-1})_{n1}b_m\end{bmatrix} = \begin{bmatrix}x_1^J\\x_2^J\\x_3^J\\\vdots\\x_n^J\end{bmatrix} = \mathbf{x}^J.$$

Therefore, $(\mathbf{J}^{-1})_{i1} = x_i^J\cdot\frac{1}{b_m}$.

Hence, this tableau can be expressed in details as follows:

Table 3.3: Objective jump tableau.

	s_m	s_{m+1} $\cdots$ s_{m+n-1}	$x_1\,x_2\cdots x_n\,s_1\cdots s_{m-1}$	RHS
z	$\frac{1}{b_m}\sum_{j=1}^{n}c_jx_j^J$	$\sum_{j=1}^{n}c_j(\mathbf{J}^{-1})_{j2}$ $\cdots$ $\sum_{j=1}^{n}c_j(\mathbf{J}^{-1})_{jn}$	$0\,0\cdots0\,0\cdots0$	$\sum_{j=1}^{n}c_jx_j^J$
x_1	$\frac{x_1^J}{b_m}$	$(\mathbf{J}^{-1})_{12}$ $\cdots$ $(\mathbf{J}^{-1})_{1n}$	$1\,0\cdots0\,0\cdots0$	x_1^J
x_2	$\frac{x_2^J}{b_m}$	$(\mathbf{J}^{-1})_{22}$ $\cdots$ $(\mathbf{J}^{-1})_{2n}$	$0\,1\cdots0\,0\cdots0$	x_2^J
$\vdots$	$\vdots$	$\vdots$ $\quad$ $\vdots$	$\vdots\ \vdots\ \ddots\ \vdots\ \vdots\ \ddots\ \vdots$	$\vdots$
x_n	$\frac{x_n^J}{b_m}$	$(\mathbf{J}^{-1})_{n2}$ $\cdots$ $(\mathbf{J}^{-1})_{nn}$	$0\,0\cdots1\,0\cdots0$	x_n^J
s_1	$-\frac{1}{b_m}\sum_{j=1}^{n}a_{1j}x_j^J$	$-\sum_{k=1}^{n}a_{1k}(\mathbf{J}^{-1})_{k1}$ $\cdots$ $-\sum_{k=1}^{n}a_{1k}(\mathbf{J}^{-1})_{kn}$	$0\,0\cdots0\,1\cdots0$	$b_1-\sum_{j=1}^{n}a_{1j}x_j^J$
$\vdots$	$\vdots$	$\vdots$ $\quad$ $\vdots$	$\vdots\ \vdots\ \ddots\ \vdots\ \vdots\ \ddots\ \vdots$	$\vdots$
s_{m-1}	$-\frac{1}{b_m}\sum_{j=1}^{n}a_{m-1,j}x_j^J$	$-\sum_{j=1}^{n}a_{m-1,k}(\mathbf{J}^{-1})_{k1}$ $\cdots$ $-\sum_{j=1}^{n}a_{m-1,k}(\mathbf{J}^{-1})_{kn}$	$0\,0\cdots0\,0\cdots1$	$b_{m-1}-\sum_{j=1}^{n}a_{m-1,j}x_j^J$

3.4 Case II (Some $c_j = 0$ but not all zeroes)

3.4.1 The 0-1 gradient

The simple idea of Case II uses the objective gradient consisting of 0 and 1, denoted by $\mathbf{c}_{\{c_j \in \{0,1\}\}} = \left[c_j \right] \in \mathbb{R}^n$ where $c_j \in \{0,1\}$.

We rearrange indices j so that the gradient of objective function contains the components one $\mathbf{1} \in \mathbb{R}^p$ and zero $\mathbf{0} \in \mathbb{R}^l$ such that $\mathbf{c}_{\{c_j \in \{0,1\}\}} = \begin{bmatrix} \mathbf{1} \\ \mathbf{0} \end{bmatrix} \in \mathbb{R}^n$ where $n = p + l$ and $p, l \geq 1$.

Orthogonal objective vectors

A set of orthogonal objective vectors of the vector $\mathbf{c}_{\{c_j \in \{0,1\}\}}$ is denoted by $\mathbf{V} = \{\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3, \ldots, \mathbf{v}_{p-1}, \mathbf{v}_p, \mathbf{v}_{p+1}, \mathbf{v}_{p+2}, \ldots, \mathbf{v}_{p+l-1}\}$ where $\mathbf{v}_i \in \mathbb{R}^n$.

1) For $1 \leq i \leq p-1$, the j^{th} component of the orthogonal objective vector $\mathbf{v}_i$ is given by

$$\left(\mathbf{v}_i\right)_j = \begin{cases} 1 & ;1 \leq j < i+1 \\ -i & ; j = i+1 \\ 0 & ; j > i+1 \end{cases}$$

2) For $i = p$, the j^{th} component of the orthogonal objective vector $\mathbf{v}_p$ is given by

$$\left(\mathbf{v}_p\right)_j = \begin{cases} 0 & ;1 \leq j \leq p \\ 1 & ; j > p \end{cases}$$

3) For $p+1 \leq i \leq p+l-1$, let $r = i - p$. The j^{th} component of the orthogonal objective vector $\mathbf{v}_i$ is given by

$$\left(\mathbf{v}_i\right)_j = \begin{cases} 0 & ;1 \leq j \leq p \text{ or } p+r+1 < j \leq p+r+l \\ 1 & ; p < j < p+r+1 \\ -r & ; j = p+r+1 \end{cases}$$

<u>Properties</u>

1) These vectors together with $\mathbf{c}_{\{c_j \in \{0,1\}\}} = \begin{bmatrix} \mathbf{1} \\ \mathbf{0} \end{bmatrix}$ are pairwise orthogonal; i.e.,

- For $1 \le i \le p-1$, $\mathbf{c}^{\mathrm{T}}_{\{c_j \in \{0,1\}\}} \mathbf{v}_i = \mathbf{1}^{\mathrm{T}} \mathbf{v}_i + \mathbf{0}^{\mathrm{T}} \mathbf{v}_i = \sum_{j=1}^{p} 1(\mathbf{v}_i)_j + 0$

$$= \sum_{j=1}^{i} (\mathbf{v}_i)_j + (\mathbf{v}_i)_{i+1} + \sum_{j=i+1}^{p} (\mathbf{v}_i)_j$$

$$= i - i + 0 = 0$$

- $\mathbf{c}^{\mathrm{T}}_{\{c_j \in \{1,0\}\}} \mathbf{v}_p = \mathbf{1}^{\mathrm{T}} \mathbf{v}_p + \mathbf{0}^{\mathrm{T}} \mathbf{v}_p = \sum_{j=1}^{p} 1(\mathbf{v}_p)_j + 0 = \sum_{j=1}^{p} 1 \cdot 0 = 0$

- For $p+1 \le i \le p+l-1$, $\mathbf{c}^{\mathrm{T}}_{\{c_j \in \{1,0\}\}} \mathbf{v}_i = \mathbf{1}^{\mathrm{T}} \mathbf{v}_i + \mathbf{0}^{\mathrm{T}} \mathbf{v}_i = \sum_{j=1}^{p} 1 \cdot 0 + 0 = 0$

- For $1 \le i \le p-1$, $\mathbf{v}_i^{\mathrm{T}} \mathbf{v}_p = \sum_{j=1}^{p} (\mathbf{v}_i)_j (\mathbf{v}_p)_j + \sum_{j=p+1}^{p+l} (\mathbf{v}_i)_j (\mathbf{v}_p)_j$

$$= \sum_{j=1}^{p} (\mathbf{v}_i)_j \cdot 0 + \sum_{j=p+1}^{p+l} 0 \cdot 1 = 0$$

- For $1 \le i \le p-1$ and $p+1 \le k \le p+l-1$,

$$\mathbf{v}_i^{\mathrm{T}} \mathbf{v}_k = \sum_{j=1}^{p} (\mathbf{v}_i)_j (\mathbf{v}_k)_j + \sum_{j=p+1}^{p+l} (\mathbf{v}_i)_j (\mathbf{v}_k)_j$$

$$= \sum_{j=1}^{p} (\mathbf{v}_i)_j 0 + \sum_{j=p+1}^{p+l} 0 (\mathbf{v}_k)_j = 0$$

- For $p+1 \le i \le p+l-1$,

$$\mathbf{v}_p^{\mathrm{T}} \mathbf{v}_i = \sum_{j=1}^{p} (\mathbf{v}_p)_j (\mathbf{v}_i)_j + \sum_{j=p+1}^{p+l} (\mathbf{v}_p)_j (\mathbf{v}_k)_j$$

$$= \sum_{j=1}^{p} 0 (\mathbf{v}_i)_j + \sum_{j=p+1}^{p+l} 1 (\mathbf{v}_i)_j = 0 + \sum_{j=p+1}^{p+l} 1 (\mathbf{v}_i)_j$$

$$= \sum_{j=p+1}^{p+k} (\mathbf{v}_i)_j + (\mathbf{v}_i)_{i+1} + \sum_{j=p+i+2}^{p+k+l} (\mathbf{v}_i)_j = i - i + 0 = 0$$

For example, the gradient of the objective function is $\mathbf{c} = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 1 \end{bmatrix}$. After rearranging the indices of the components of the gradient of the objective function,

orthogonal objective vectors of $\mathbf{c}=\begin{bmatrix}1\\1\\1\\0\\0\end{bmatrix}$ are $\mathbf{v}_1=\begin{bmatrix}1\\-1\\0\\0\\0\end{bmatrix}$, $\mathbf{v}_2=\begin{bmatrix}1\\1\\-2\\0\\0\end{bmatrix}$, $\mathbf{v}_3=\begin{bmatrix}0\\0\\0\\1\\1\end{bmatrix}$, and

$\mathbf{v}_4=\begin{bmatrix}0\\0\\0\\1\\-1\end{bmatrix}$ such that $\mathbf{c}^{\mathrm{T}}\mathbf{v}_1=0,\ \mathbf{c}^{\mathrm{T}}\mathbf{v}_2=0,\mathbf{c}^{\mathrm{T}}\mathbf{v}_3=0,\ \mathbf{c}^{\mathrm{T}}\mathbf{v}_4=0,\ \mathbf{v}_1^{\mathrm{T}}\mathbf{v}_2=0,\ \mathbf{v}_1^{\mathrm{T}}\mathbf{v}_3=0,$

$\mathbf{v}_1^{\mathrm{T}}\mathbf{v}_4=0,\ \mathbf{v}_2^{\mathrm{T}}\mathbf{v}_3=0,\ \mathbf{v}_2^{\mathrm{T}}\mathbf{v}_4=0$, and $\mathbf{v}_3^{\mathrm{T}}\mathbf{v}_4=0$.

LP model with 0-1 gradient of the objective function

Let $\mathbf{c}_{\{c_j\in\{0,1\}\}}=[c_j]\in\mathbb{R}^n$ where $c_j\in\{0,1\}$. After rearranging $\mathbf{c}_{\{c_j\in\{0,1\}\}}=\begin{bmatrix}\mathbf{1}\\\mathbf{0}\end{bmatrix}\in\mathbb{R}^n$,

the following LP model is considered.

$$\begin{aligned}\max z &= \begin{bmatrix}\mathbf{1}^{\mathrm{T}} & \mathbf{0}^{\mathrm{T}}\end{bmatrix}\mathbf{y}\\ \text{s.t.}\quad \mathbf{A}\mathbf{y} &\le \mathbf{b}\\ \mathbf{y} &\ge \mathbf{0}\end{aligned} \tag{3.9}$$

where $\mathbf{y}\in\mathbb{R}^n$ is a vector of variables.

Artificial edge and orthogonal objective matrix

The orthogonal objective matrix is

$$\mathbf{T}_0=\begin{bmatrix}\mathbf{v}_1^{\mathrm{T}}\\ \mathbf{v}_2^{\mathrm{T}}\\ \mathbf{v}_3^{\mathrm{T}}\\ \vdots\\ \mathbf{v}_{p-1}^{\mathrm{T}}\\ \mathbf{v}_p^{\mathrm{T}}\\ \mathbf{v}_{p+1}^{\mathrm{T}}\\ \mathbf{v}_{p+2}^{\mathrm{T}}\\ \mathbf{v}_{p+3}^{\mathrm{T}}\\ \vdots\\ \mathbf{v}_{p+l-1}^{\mathrm{T}}\end{bmatrix}=\begin{bmatrix}1&-1&0&0&\cdots&0&0&0&0&0&\cdots&0\\ 1&1&-2&0&\cdots&0&0&0&0&0&\cdots&0\\ 1&1&1&-3&\cdots&0&0&0&0&0&\cdots&0\\ \vdots&\vdots&\vdots&\vdots&\ddots&\vdots&\vdots&\vdots&\vdots&\vdots&\ddots&\vdots\\ 1&1&1&1&\cdots&-(p-1)&0&0&0&0&\cdots&0\\ 0&0&0&0&\cdots&0&1&1&1&1&\cdots&1\\ 0&0&0&0&\cdots&0&1&-1&0&0&\cdots&0\\ 0&0&0&0&\cdots&0&1&1&-2&0&\cdots&0\\ 0&0&0&0&\cdots&0&1&1&1&-3&\cdots&0\\ \vdots&\vdots&\vdots&\vdots&\ddots&\vdots&\vdots&\vdots&\vdots&\vdots&\ddots&\vdots\\ 0&0&0&0&\cdots&0&1&1&1&1&\cdots&-(l-1)\end{bmatrix} \tag{3.10}$$

The system of equations representing the artificial edge is written in a matrix form as $\mathbf{T}_0\mathbf{y}=\mathbf{0}$.

Claim1.2. The system of equations $\mathbf{T}_0\mathbf{y}=\mathbf{0}$ has the general solution of the form $y_1=y_2=y_3\ldots=y_p$ and $y_{p+1}=y_{p+2}=y_{p+3}=\ldots=y_{p+l}=0$.

Proof.

The system of equations $\mathbf{T}_1\mathbf{y}=\mathbf{0}$ can be rewritten as:

$$
\begin{array}{lcll}
y_1 & = & y_2 & \cdots(1)\\
y_1+y_2 & = & 2y_3 & \cdots(2)\\
y_1+y_2+y_3 & = & 3y_4 & \cdots(3)\\
\vdots\quad\vdots\quad\vdots\quad\ddots & \vdots & \vdots & \\
y_1+y_2+y_3+\cdots+y_{p-1} & = & (p-1)y_p & \cdots(p\text{-}1)\\
y_{p+1}+y_{p+2}+y_{p+3}+\cdots+y_{p+l-1}+y_{p+l} & = & 0 & \cdots(p)\\
y_{p+1} & = & y_{p+2} & \cdots(p+1)\\
y_{p+1}+y_{p+2} & = & 2y_{p+3} & \cdots(p+2)\\
y_{p+1}+y_{p+2}+y_{p+3} & = & 3y_{p+4} & \cdots(p+3)\\
\vdots\quad\vdots\quad\vdots\quad\ddots & \vdots & \vdots & \\
y_{p+1}+y_{p+2}+y_{p+3}+\cdots+y_{p+l-1} & = & (l-1)y_{p+l} & \cdots(p+l\text{-}1)
\end{array}
$$

It follows from Claim 1.1 that the first $p-1$ equations give the solution $y_1=y_2=y_3\ldots=y_p$. Similarly, the last $l-1$ equations give the solutions $y_{p+1}=y_{p+2}=y_{p+3}=\ldots=y_{p+l}$.

However, the equation (p) forces $y_{p+1}=y_{p+2}=y_{p+3}=\ldots=y_{p+l}=0$. Therefore, the general solution is given by $y_1=y_2=y_3\ldots=y_p$ and

$y_{p+1}=y_{p+2}=y_{p+3}=\ldots=y_{p+l}=0$. ■

Definition 2.2 (Jump point)

For the LP model (3.9), a point $\mathbf{y}=\left[y_j\right]\in\mathbb{R}^n$ is the *jump point* if

Condition (1): The point $\mathbf{y}$ satisfies the condition $y_1=y_2=y_3\ldots=y_p=\mu>0$ for some scalar μ and $y_{p+1}=y_{p+2}=y_{p+3}=\ldots=y_{p+l}=0$.

Condition (2): The point $\mathbf{y}$ satisfies at least one equality constraint $\sum_{j=1}^{n}a_{qj}y_j+s_q=b_q$ such that the slack variable $s_q=0$ and $b_q>0$.

Condition (3): The point $\mathbf{y}$ is feasible for the LP model (3.9).

Preposition 2.3

For the LP model (3.9), let $\sum_{j=1}^{n} a_{ij} y_j \le b_i$ be constraints such that $b_i \ge 0$ and not all $b_i = 0$. If there exists some i such that $\sum_{j=1}^{p} a_{ij} > 0$ and $b_i > 0$, the jump point is $\mathbf{y}^J = \left[y_j^J \right] \in \mathbb{R}^n$ where the component $y_j^J = \begin{cases} \mu & ; 1 \le j \le p \\ 0 & ; p+1 \le j \le p+l \end{cases}$ and $\mu = \min_i \left\{ \frac{b_i}{\sum_{j=1}^{p} a_{ij}} > 0 \;\middle|\; \sum_{j=1}^{p} a_{ij} > 0 \right\}$. Otherwise, the jump point does not exist.

Proof.

After adding slacks variable $s_i \ge 0$ to constraints $\sum_{j=1}^{n} a_{ij} y_j \le b_i$ such that $b_i \ge 0$, the constraints become $\sum_{j=1}^{n} a_{ij} y_j + s_i = b_i$.

Case 1

Suppose that there exists some r such that $\sum_{j=1}^{p} a_{rj} > 0$ and $b_r > 0$.

Let $q = \arg\min_r \left\{ \frac{b_r}{\sum_{j=1}^{p} a_{rj}} > 0 \;\middle|\; \sum_{j=1}^{p} a_{rj} > 0 \right\}$ and $\mathbf{y} = \left[y_j \right] \in \mathbb{R}^n$ such that

$$y_j = \begin{cases} \dfrac{b_q}{\sum_{j=1}^{p} a_{qj}} = \mu & ; 1 \le j \le p \\ 0 & ; p+1 \le j \le p+l. \end{cases}$$

Clearly, the point $\mathbf{y}$ satisfies condition (1) in Definition 2.2.

Moreover, the point $\mathbf{y}$ satisfies condition (2) since substituting $y_j = \begin{cases} \mu & ; 1 \le j \le p \\ 0 & ; p+1 \le j \le p+l \end{cases}$ into the constraint $\sum_{j=1}^{n} a_{qj} y_j + s_q = b_q$ will result in $s_q = 0$.

The point $\mathbf{y}$ also satisfies all constraints which can be explained as follows.

For the constraint i such that $\sum_{j=1}^{p} a_{ij} > 0$,

$$\sum_{j=1}^{n} a_{ij} y_j = \left(\sum_{j=1}^{p} a_{ij} \right) \mu + \left(\sum_{j=p+1}^{p+l} a_{ij} \right) 0 = \left(\sum_{j=1}^{p} a_{ij} \right) \frac{b_q}{\sum_{j=1}^{p} a_{qj}} \leq \left(\sum_{j=1}^{p} a_{ij} \right) \frac{b_i}{\sum_{j=1}^{p} a_{ij}} = b_i .$$

For the constraint i such that $\sum_{j=1}^{p} a_{ij} \leq 0$,

$\sum_{j=1}^{n} a_{ij} y_j = \left(\sum_{j=1}^{p} a_{ij} \right) \mu + \left(\sum_{j=p+1}^{p+l} a_{ij} \right) 0 = \left(\sum_{j=1}^{p} a_{ij} \right) \mu \leq 0 \leq b_i$. Therefore, the point $\mathbf{y}$ is feasible and satisfies condition (3) in Definition 2.2.

Hence, the point $\mathbf{y}$ is the jump point for the LP model (3.9).

Case 2

Suppose $\sum_{j=1}^{p} a_{ij} \leq 0$ or $b_i = 0$ for all $i = 1, 2, 3, ..., m$. Assume a jump point $\mathbf{y}$ exists. Hence from condition (1), $y_1 = y_2 = y_3 \ldots = y_p = \mu$ for some $\mu > 0$ and $y_{p+1} = y_{p+2} = y_{p+3} = \ldots = y_{p+l} = 0$. Since $\mathbf{y}$ is feasible, for any constraint i, we have $\sum_{j=1}^{n} a_{ij} y_j \leq b_i$. Because $\sum_{j=1}^{n} a_{ij} y_j = \left(\sum_{j=1}^{p} a_{ij} \right) \mu + \left(\sum_{j=p+1}^{p+l} a_{ij} \right) 0 = \left(\sum_{j=1}^{p} a_{ij} \right) \mu \leq 0$ and $b_i \geq 0$, this implies $\sum_{j=1}^{n} a_{ij} y_j = b_i = 0$. Condition (2) from definition 2.2 cannot be satisfied, which is a contradiction. So, a jump point does not exist. ■

3.4.2 General case II

In this case, the gradient of the objective function is denoted by $\mathbf{c}_{\{c_j \geq 0\}} = \left[c_j \right] \in \mathbb{R}^n$ where $c_j \geq 0$. We consider the LP model after separating $\mathbf{c}_{\{c_j \geq 0\}}$ to the vector of $\mathbf{c}_{\{c_j > 0\}} = [c_j] \in \mathbb{R}^p, c_j > 0$ and the zero vector $\mathbf{0} \in \mathbb{R}^l$ where $n = p + l$ and $p,l \geq 1$:

$$\begin{aligned} \max \quad & \left[\mathbf{c}^{\mathrm{T}}_{\{c_j > 0\}} \quad \mathbf{0}^{\mathrm{T}} \right] \mathbf{x} \\ \text{s.t.} \quad & \mathbf{A}\mathbf{x} \leq \mathbf{b} \\ & \mathbf{x} \geq 0, \end{aligned} \tag{3.11}$$

where the vector of variables $\mathbf{x} \in \mathbb{R}^n$.

Transformation of orthogonal objective matrix

In the LP model (3.9), the artificial edge for the jump can be constructed through the system of equations $\mathbf{T}_0 \mathbf{y} = \mathbf{0}$ where $\mathbf{T}_0$ is defined in (3.10). We want to construct the similar system for the model (3.11). Notice that we can transform the model (3.11) into (3.9) by setting the variable $y_j = \begin{cases} c_j x_j & ; 1 \leq j \leq p \\ x_j & ; p+1 \leq j \leq p+l \end{cases}$.

Thus, $\sum_{j=1}^{p} 1(c_j x_j) + \sum_{j=p+1}^{p+l} 0 x_j = \sum_{j=1}^{p} 1 y_j + \sum_{j=p+1}^{p+l} 0 y_j = \left[\mathbf{1}^{\mathrm{T}} \quad \mathbf{0}^{\mathrm{T}} \right] \mathbf{y}$.

Let $a'_{ij} = \begin{cases} \dfrac{a_{ij}}{c_j} & ; 1 \leq j \leq p \\ a_{ij} & ; p+1 \leq j \leq p+l \end{cases}$. Hence, the constraint i can be transformed as

$$\sum_{j=1}^{n} a_{ij} x_j = \sum_{j=1}^{p} \frac{a_{ij}}{c_j} c_j x_j + \sum_{j=p+1}^{p+l} a_{ij} x_j = \sum_{j=1}^{n} a'_{ij} y_j \leq b_i .$$

Therefore, the model (3.11) is equivalent to

$$\begin{aligned} \max \quad & \left[\mathbf{1}^{\mathrm{T}} \quad \mathbf{0}^{\mathrm{T}} \right] \mathbf{y} \\ \text{s.t.} \quad & \mathbf{A}'\mathbf{y} \leq \mathbf{b} \\ & \mathbf{y} \geq 0 \end{aligned} \tag{3.12}$$

where the matrix $\mathbf{A}' = \left[a'_{ij} \right] \in \mathbb{R}^{m \times n}$.

Suppose the orthogonal objective matrix is $\mathbf{T}_0 = [t_{ij}] \in \mathbb{R}^{(n-1)\times n}$ defined in (3.10). Consider the system of equations $\mathbf{T}_0\mathbf{y} = \mathbf{0}$. For any equation i,

$$\sum_{j=1}^{n} t_{ij} y_j = \sum_{j=1}^{p} t_{ij} y_j + \sum_{j=p+1}^{p+l} t_{ij} y_j = \sum_{j=1}^{p} t_{ij} \left(c_j x_j\right) + \sum_{j=p+1}^{p+l} t_{ij} \left(x_j\right) = \sum_{j=1}^{p} \left(t_{ij} c_j\right) x_j + \sum_{j=p+1}^{p+l} t_{ij} x_j = 0.$$

Therefore, if we set $\mathbf{T} = [t'_{ij}]$ where $t'_{ij} = \begin{cases} t_{ij} c_j & \text{if } 1 \le j \le p \\ t_{ij} & \text{if } p+1 \le j \le p+l \end{cases}$, the system of equations $\mathbf{T}_0\mathbf{y} = \mathbf{0}$ is equivalent to $\mathbf{T}\mathbf{x} = \mathbf{0}$; i.e.,

$$\begin{bmatrix} c_1 & -c_2 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 & 0 & \cdots & 0 \\ c_1 & c_2 & -2c_3 & 0 & \cdots & 0 & 0 & 0 & 0 & 0 & \cdots & 0 \\ c_1 & c_2 & c_3 & -3c_4 & \cdots & 0 & 0 & 0 & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ c_1 & c_2 & c_3 & c_4 & \cdots & -(p-1)c_p & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 1 & 1 & 1 & 1 & \cdots & 1 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 1 & -1 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 1 & 1 & -2 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 1 & 1 & 1 & -3 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 0 & 1 & 1 & 1 & 1 & \cdots & -(l-1) \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ \vdots \\ x_{p-1} \\ x_p \\ x_{p+1} \\ x_{p+2} \\ x_{p+3} \\ \vdots \\ x_{p+l} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{3.13}$$

Note that the row vectors in the matrix $\mathbf{T}$ may no longer be pairwise orthogonal.

Jump point

According to Preposition 2.3, the jump point of the LP model (3.12) is $\mathbf{y}^J = [y_j^J] \in \mathbb{R}^n$ where

$$y_j^J = \begin{cases} \min_i \left\{ \dfrac{b_i}{\sum_{j=1}^{p} a'_{ij}} > 0 \;\middle|\; \sum_{j=1}^{p} a'_{ij} > 0 \right\} = \mu' & \text{if } 1 \le j \le p \\ 0 & \text{if } p+1 \le j \le p+l, \end{cases}$$

provided that there are some r such that $\sum_{j=1}^{p} a'_{rj} > 0$ and $b_r > 0$. This jump point can be translated to the jump point of the LP model (3.11) as $\mathbf{x}^J = [x_j^J] \in \mathbb{R}^n$ where

$$x_j^J = \begin{cases} \dfrac{\mu'}{c_j} & \text{if } 1 \le j \le p \\ 0 & \text{if } p+1 \le j \le p+l. \end{cases}$$

Sub-LP model

To incorporate the artificial edge $\mathbf{Tx} = \mathbf{0}$ into the model (3.11) where $\mathbf{T} = \left[t'_{ij} \right] \in \mathbb{R}^{(n-1)\times n}$ is defined in (3.13), we need to transform this intersection of hyperplanes into the intersection of half-spaces $\sum_{j=1}^{n} t'_{ij} x_j \le 0$ or $\sum_{j=1}^{n} t'_{ij} x_j \ge 0$. The intersection of these half-spaces together with the feasible region in the model (3.11) form the sub-LP model. Note that there are 2^{n-1} possible sub-LP models due to the possibilities of $\sum_{j=1}^{n} t'_{ij} x_j \le 0$ or $\sum_{j=1}^{n} t'_{ij} x_j \ge 0$ for each artificial constraint. In this thesis, we only select one sub-LP model of the form:

$$\begin{aligned} \max \quad & \mathrm{z} = \mathbf{c}^{\mathrm{T}}_{\{c_j \ge 0\}} \mathbf{x} \\ \text{s.t.} \quad & \mathbf{Ax} \le \mathbf{b} \\ & \mathbf{Tx} \le \mathbf{0} \\ & \mathbf{x} \ge \mathbf{0} \end{aligned} \tag{3.14}$$

Objective jump model

To apply the simplex tableau to the sub-LP model (3.14), we need to convert this problem into a standard form. Assuming a jump point exists, let $q = \arg\min_i \left\{ \frac{b_i}{\sum_{j=1}^{p} \frac{a_{ij}}{c_j}} > 0 \;\middle|\; \sum_{j=1}^{p} \frac{a_{ij}}{c_j} > 0 \right\}$. Rearrange the indices $i = 1, 2, 3, ..., m$ of the constraint in $\mathbf{Ax} \le \mathbf{b}$ so that $m = q$. The problem (3.14) is turned into the objective jump model below.

$$\begin{aligned} \max \; & z = \sum_{j=1}^{p} c_j x_j + \sum_{j=p+1}^{p+l} c_j x_j \\ \text{s.t.} \; & \sum_{j=1}^{n} a_{ij} x_j + s_i = b_i && \text{for } i = 1, 2, ..., m-1 \\ & \sum_{j=1}^{n} a_{mj} x_j + s_m = b_m \\ & \sum_{j=1}^{i-m} c_j x_j - (i-m) c_{i-m+1} x_{i-m+1} + s_i = 0 && \text{for } i = m+1, m+2, ..., m+p-1 \\ & \sum_{j=p+1}^{p+l} x_j + s_{m+p} = 0 \\ & \sum_{j=p+1}^{i-m} x_j - (i-m) x_{i-m+1} + s_i = 0 && \text{for } i = m+p+1, m+p+2, ..., m+p+l-1 \\ & x_j, s_i \ge 0 && \text{for } j = 1, 2, ..., n \text{ and } i = 1, 2, ..., m+n-1 \end{aligned} \tag{3.15}$$

Basis of the objective jump model

The main idea of SOJU is to start the simplex method from the jump point. Hence we want to determine the initial basis that represent the point. This can be done using the objective jump model (3.15) as follows. First, we choose a set of nonbasic variables as $\{s_m, s_{m+1}, s_{m+2}, \ldots, s_{m+n-1}\}$ and the remainders are the set of basic variables: $\{x_1, x_2, x_3, \ldots, x_n, s_1, s_2, \ldots, s_{m-1}\}$. Then, the current of basic (subscript $\mathbf{B}$) and nonbasic (subscript $\mathbf{N}$) are separated as follows:

$$\begin{aligned} \max \quad & z = \mathbf{0}_n^{\mathrm{T}}\mathbf{x}_{\mathbf{N}} + \begin{bmatrix}\mathbf{c}^{\mathrm{T}} & \mathbf{0}_{m-1}\end{bmatrix}\mathbf{x}_{\mathbf{B}} \\ \text{s.t.} \quad & \begin{bmatrix}\mathbf{0}_{(m-1)\times n} \\ \mathbf{I}_n\end{bmatrix}\mathbf{x}_{\mathbf{N}} + \begin{bmatrix}\mathbf{Q}_{(m-1)\times n} & \mathbf{I}_{m-1} \\ \mathbf{J}_{n\times n} & \mathbf{0}_{n\times(m-1)}\end{bmatrix}\mathbf{x}_{\mathbf{B}} = \begin{bmatrix}\mathbf{b}_1 \\ \mathbf{b}_2\end{bmatrix} \\ & \mathbf{x}_{\mathbf{N}}, \mathbf{x}_{\mathbf{B}} \geq \mathbf{0}, \end{aligned} \tag{3.16}$$

where $\mathbf{Q} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1,n-1} & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2,n-1} & a_{2n} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{m-1,1} & a_{m-1,2} & \cdots & a_{m-1,n-1} & a_{m-1,n} \end{pmatrix}_{(m-1)\times n}$, $\mathbf{b}_1 = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_{m-1} \end{bmatrix}$,

$$\mathbf{J} = \begin{bmatrix} a_{m1} & a_{m2} & a_{m3} & \cdots & a_{mp} & a_{m,p+1} & a_{m,p+2} & a_{m,p+3} & \cdots & a_{m,p+l} \\ c_1 & -c_2 & 0 & \cdots & 0 & 0 & 0 & 0 & \cdots & 0 \\ c_1 & c_2 & -2c_3 & \cdots & 0 & 0 & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ c_1 & c_2 & c_3 & \cdots & -(p-1) & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & \cdots & 0 & 1 & 1 & 1 & \cdots & 1 \\ 0 & 0 & 0 & \cdots & 0 & 1 & -1 & 0 & \cdots & 0 \\ 0 & 0 & 0 & \cdots & 0 & 1 & 1 & -2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & 1 & 1 & 1 & \cdots & -(l-1) \end{bmatrix}, \quad \mathbf{b}_2 = \begin{bmatrix} b_m \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}.$$

Table 3.4 represents the tableau of the current basis from the model (3.16). This tableau will be used in SOJU.

Table 3.4: Current basis of the objective jump tableau.

	s_m s_{m+1} s_{m+2} $\cdots$ s_{m+p} s_{m+p+1} s_{m+p+2} s_{m+p+3} $\cdots$ s_{m+n-1}	x_1 x_2 x_3 $\cdots$ x_p x_{p+1} x_{p+2} x_{p+3} $\cdots$ x_n	s_1 $\cdots$ s_{m-1}	RHS
Z	$0\ 0\ 0\ \cdots\ 0\ 0\ 0\ 0\ \cdots\ 0$	$c_1\ c_2\ c_3\ \cdots\ c_p\ c_{p+1}\ c_{p+2}\ c_{p+3}\ \cdots\ c_n$	$0\ \cdots\ 0$	0
	$0\ 0\ 0\ \cdots\ 0\ 0\ 0\ 0\ \cdots\ 0$	$a_{11}\ a_{12}\ a_{13}\ \cdots\ a_{1p}\ a_{1,p+1}\ a_{1,p+2}\ a_{1,p+3}\ \cdots\ a_{1,p+l}$	$1\ \cdots\ 0$	b_1
	$\vdots\ \vdots\ \vdots\ \ddots\ \vdots\ \vdots\ \vdots\ \vdots\ \ddots\ \vdots$	$\vdots\ \vdots\ \vdots\ \ddots\ \vdots\ \vdots\ \vdots\ \vdots\ \ddots\ \vdots$	$\vdots\ \ddots\ \vdots$	$\vdots$
	$0\ 0\ 0\ \cdots\ 0\ 0\ 0\ 0\ \cdots\ 0$	$a_{m-1,1}\ a_{m-1,2}\ a_{m-1,3}\ \cdots\ a_{m-1,p}\ a_{m-1,p+1}\ a_{m-1,p+2}\ a_{m-1,p+3}\ \cdots\ a_{m-1,p+l}$	$0\ \cdots\ 1$	b_{m-1}
	$1\ 0\ 0\ \cdots\ 0\ 0\ 0\ 0\ \cdots\ 0$	$a_{m1}\ a_{m2}\ a_{m3}\ \cdots\ a_{mp}\ a_{m,p+1}\ a_{m,p+2}\ a_{m,p+3}\ \cdots\ a_{m,p+l}$	$0\ \cdots\ 0$	b_m
	$0\ 1\ 0\ \cdots\ 0\ 0\ 0\ 0\ \cdots\ 0$	$c_1\ -c_2\ 0\ \cdots\ 0\ 0\ 0\ 0\ \cdots\ 0$	$0\ \cdots\ 0$	0
	$0\ 0\ 1\ \cdots\ 0\ 0\ 0\ 0\ \cdots\ 0$	$c_1\ c_2\ -2c_3\ \cdots\ 0\ 0\ 0\ 0\ \cdots\ 0$	$0\ \cdots\ 0$	0
	$\vdots\ \vdots\ \vdots\ \ddots\ \vdots\ \vdots\ \vdots\ \vdots\ \ddots\ \vdots$	$\vdots\ \vdots\ \vdots\ \ddots\ \vdots\ \vdots\ \vdots\ \vdots\ \ddots\ \vdots$	$\vdots\ \ddots\ \vdots$	$\vdots$
	$0\ 0\ 0\ \cdots\ 1\ 0\ 0\ 0\ \cdots\ 0$	$c_1\ c_2\ c_3\ \cdots\ -(p-1)\ 0\ 0\ 0\ \cdots\ 0$	$0\ \cdots\ 0$	0
	$0\ 0\ 0\ \cdots\ 0\ 1\ 0\ 0\ \cdots\ 0$	$0\ 0\ 0\ \cdots\ 0\ 1\ 1\ 1\ \cdots\ 1$	$0\ \cdots\ 0$	0
	$0\ 0\ 0\ \cdots\ 0\ 0\ 1\ 0\ \cdots\ 0$	$0\ 0\ 0\ \cdots\ 0\ 1\ -1\ 0\ \cdots\ 0$	$0\ \cdots\ 0$	0
	$0\ 0\ 0\ \cdots\ 0\ 0\ 0\ 1\ \cdots\ 0$	$0\ 0\ 0\ \cdots\ 0\ 1\ 1\ -2\ \cdots\ 0$	$0\ \cdots\ 0$	0
	$\vdots\ \vdots\ \vdots\ \ddots\ \vdots\ \vdots\ \vdots\ \vdots\ \ddots\ \vdots$	$\vdots\ \vdots\ \vdots\ \ddots\ \vdots\ \vdots\ \vdots\ \vdots\ \ddots\ \vdots$	$\vdots\ \ddots\ \vdots$	$\vdots$
	$0\ 0\ 0\ \cdots\ 0\ 0\ 0\ 0\ \cdots\ 1$	$0\ 0\ 0\ \cdots\ 0\ 1\ 1\ 1\ \cdots\ -(l-1)$	$0\ \cdots\ 0$	0

Objective jump tableau

Objective jump tableau can be created by calculating the following simplex tableau,

	$\mathbf{x_N}$	$\mathbf{x_B}$	RHS
Z	$\mathbf{c_B^T B^{-1} N - c_N^T}$	$\mathbf{0}_{m-1}^{\mathrm{T}}$	$\mathbf{c_B^T B^{-1} b}$
$\mathbf{x_B}$	$\mathbf{B^{-1} N}$	$\mathbf{I}_{m-1}$	$\mathbf{B^{-1} b}$

where the inverse of $\mathbf{B}^{-1} = \begin{bmatrix} \mathbf{0}_{n\times(m-1)} & \mathbf{J}^{-1} \\ \mathbf{I}_{m-1} & -\mathbf{Q}\mathbf{J}^{-1} \end{bmatrix}$.

After calculation, the objective jump tableau is

	$\mathbf{x_N}$	$\mathbf{x_B}$	RHS
Z	$\mathbf{c}^{\mathrm{T}}\mathbf{J}^{-1}$	$\mathbf{0}_{m-1}^{\mathrm{T}}$	$\mathbf{c}^{\mathrm{T}}\mathbf{J}^{-1}\mathbf{b}_2$
$\mathbf{x_B}$	$\begin{bmatrix} \mathbf{J}^{-1} \\ -\mathbf{Q}\mathbf{J}^{-1} \end{bmatrix}$	$\mathbf{I}_{m-1}$	$\begin{bmatrix} \mathbf{J}^{-1}\mathbf{b}_2 \\ \mathbf{b}_1 - \mathbf{Q}\mathbf{J}^{-1}\mathbf{b}_2 \end{bmatrix}$

Note that $\mathbf{J}^{-1}\mathbf{b}_2 = \left[(\mathbf{J}^{-1})_{i1} b_m\right] = \left[x_i^J\right] = \mathbf{x}^J$ .Therefore, $(\mathbf{J}^{-1})_{i1} = x_i^J \cdot \dfrac{1}{b_m}$ where

$$x_i^J = \begin{cases} \dfrac{1}{c_i} \cdot \dfrac{b_m}{\sum_{j=1}^{p} \dfrac{a_{mj}}{c_j}} = \dfrac{\mu}{c_i} & \text{if } i = 1,2,3,\ldots,p \\ 0 & \text{if } i = p+1, p+2, p+3, ..., p+l \end{cases} .$$

Hence, this tableau can be expressed in more details as follows:

Table 3.5: Objective jump tableau.

	s_m	s_{m+1}	$\cdots$	$s_{m+p+l-1}$	$x_1 \cdots x_p$	$x_{p+1} \cdots x_{p+l}$	$s_1 \cdots s_{m-1}$	RHS
z	$\frac{1}{b_m}\sum_{j=1}^{p} c_j x_j^J$	$\sum_{j=1}^{p} c_j (\mathbf{J}^{-1})_{j2}$	$\cdots$	$\sum_{j=1}^{p} c_j (\mathbf{J}^{-1})_{jp}$	$0 \cdots 0$	$0 \cdots 0$	$0 \cdots 0$	$\sum_{j=1}^{p} c_j x_j^J$
x_1	$\frac{x_1^J}{b_m}$	$(\mathbf{J}^{-1})_{12}$	$\cdots$	$(\mathbf{J}^{-1})_{1p}$	$1 \cdots 0$	$0 \cdots 0$	$0 \cdots 0$	x_1^J
$\vdots$	$\vdots$	$\vdots$		$\vdots$	$\vdots \ \vdots \ \vdots$	$\vdots \ \vdots \ \vdots$	$\vdots \ \vdots \ \vdots$	$\vdots$
x_p	$\frac{x_p^J}{b_m}$	$(\mathbf{J}^{-1})_{p2}$	$\cdots$	$(\mathbf{J}^{-1})_{pp}$	$0 \cdots 1$	$0 \cdots 0$	$0 \cdots 0$	x_p^J
x_{p+1}	0	$(\mathbf{J}^{-1})_{p+1,2}$	$\cdots$	$(\mathbf{J}^{-1})_{p+1,p}$	$0 \cdots 0$	$1 \cdots 0$	$0 \cdots 0$	0
$\vdots$	$\vdots$	$\vdots$		$\vdots$	$\vdots \ \vdots \ \vdots$	$\vdots \ \vdots \ \vdots$	$\vdots \ \vdots \ \vdots$	$\vdots$
x_{p+l}	0	$(\mathbf{J}^{-1})_{p+l,2}$	$\cdots$	$(\mathbf{J}^{-1})_{p+l,p}$	$0 \cdots 0$	$0 \cdots 1$	$0 \cdots 0$	0
s_1	$-\frac{1}{b_m}\sum_{j=1}^{p} a_{1j} x_j^J$	$-\sum_{k=1}^{p} a_{1k} (\mathbf{J}^{-1})_{k1}$	$\cdots$	$-\sum_{k=1}^{p} a_{1k} (\mathbf{J}^{-1})_{kn}$	$0 \cdots 0$	$0 \cdots 0$	$1 \cdots 0$	$b_1 - b_m \sum_{j=1}^{p} a_{1j} x_j^J$
$\vdots$	$\vdots$	$\vdots$		$\vdots$	$\vdots \ \vdots \ \vdots$	$\vdots \ \vdots \ \vdots$	$\vdots \ \vdots \ \vdots$	$\vdots$
s_{m-1}	$-\frac{1}{b_m}\sum_{j=1}^{p} a_{m-1,j} x_j^J$	$-\sum_{j=1}^{n} a_{m-1,k} (\mathbf{J}^{-1})_{k1}$	$\cdots$	$-\sum_{j=1}^{n} a_{m-1,k} (\mathbf{J}^{-1})_{kn}$	$0 \cdots 0$	$0 \cdots 0$	$0 \cdots 1$	$b_{m-1} - b_m \sum_{j=1}^{p} a_{m-1,j} x_j^J$

3.5 Simplex method with objective jump (SOJU)

The simplex method with objective jump were divided into two stages. The first stage of SOJU would find an initial basic feasible solution using the objective jump procedure. The second stage of SOJU would determine the optimal solution starting from the initial basic feasible solution from the first stage using the traditional simplex method.

3.5.1 Algorithm

1st stage

6. Generate the objective jump tableau as shown in Table 3.3 for LP model (3.4) or Table 3.5 for LP model (3.11).
7. Choose a set of entering variables $s_{m+1}, s_{m+2}, s_{m+3}, \ldots, s_{m+n-1}$, respectively.
8. Choose a set of leaving variables corresponding to the entering variables by the minimum ratio test.
9. Operate pivoting.
10. Eliminate the set of rows and columns of these variables $s_{m+1}, s_{m+2}, \ldots, s_{m+n-1}$, respectively.

2nd stage

After the 1st stage, we can apply Dantzig's pivot rule of the standard simplex method to find the optimal solution of a linear programming problem.

3.6 Examples

Klee and Minty example [9] is given by

$$\begin{aligned} \max \quad & \sum_{j=1}^{n} 10^{n-j} x_j \\ \text{s.t.} \quad & 2\sum_{j=1}^{i-1} 10^{i-j} x_j + x_i \le 100^{i-1} \quad \text{for} \quad i = 1, 2, 3, ..., n \\ & x_j \ge 0. \end{aligned}$$

Example 1. Consider the Klee and Minty problem with 3 variables.

$$\begin{aligned} \max \quad & z = 100x_1 + 10x_2 + x_3 \\ \text{s.t.} \quad & x_1 \le 1 \\ & 20x_1 + x_2 \le 100 \\ & 200x_1 + 20x_2 + x_3 \le 10000 \\ & x_1, x_2, x_3 \ge 0 \end{aligned}$$

1st stage

1) Calculate b_i and $\sum_{j=1}^{3} \frac{a_{ij}}{c_j}$. Choose the i^{th} constraint which has a positive smallest ratio $\frac{b_i}{\sum_{j=1}^{3} \frac{a_{ij}}{c_j}}$ and find the jump point.

i^{th} constraint	$\sum_{j=1}^{3} \frac{a_{ij}}{c_j}$	b_i	$\frac{b_i}{\sum_{j=1}^{3} \frac{a_{ij}}{c_j}}$
1	0.01	1	100
2	0.3	100	333.33
3	5	10000	2000

All ratios are positive and the first constraint has a smallest ratio. So, the jump point is $(x_1^J, x_2^J, x_3^J) = \left(\frac{100}{100}, \frac{100}{10}, \frac{100}{1}\right) = (1,\ 10,\ 100)$.

2) Add artificial constraints and generate the sub-LP problem in the standard form.

$$
\begin{aligned}
\max \quad & z = 100x_1 + 10x_2 + x_3 \\
\text{s.t.} \quad & 20x_1 + x_2 + s_1 = 100 \\
& 200x_1 + 20x_2 + x_3 + s_2 = 10000 \\
& x_1 + s_q = 1 \\
& 100x_1 - 10x_2 + s_4 = 0 \\
& 100x_1 + 10x_2 - 2x_3 + s_5 = 0 \\
& x_1, x_2, x_3, s_1, s_2, s_3, s_4, s_5 \geq 0
\end{aligned}
$$

3) Create the objective jump tableau by calculating $\mathbf{J}^{-1}$, $-\mathbf{Q}\mathbf{J}^{-1}$, $\mathbf{c}^{\mathrm{T}}\mathbf{J}^{-1}$, $\mathbf{c}^{\mathrm{T}}\mathbf{J}^{-1}\mathbf{b}_2$ and $\mathbf{b}_1 - \mathbf{Q}\mathbf{J}^{-1}\mathbf{b}_2$.

Since $\mathbf{J}=\begin{bmatrix}1 & 0 & 0\\ 100 & -10 & 0\\ 100 & 10 & -2\end{bmatrix}$, $\mathbf{J}^{-1}=\begin{bmatrix}1 & 0 & 0\\ 10 & -0.1 & 0\\ 100 & -0.5 & -0.5\end{bmatrix}$. Moreover,

$\mathbf{Q}=\begin{bmatrix}20 & 1 & 0\\ 200 & 20 & 1\end{bmatrix}$, $\mathbf{c}=\begin{bmatrix}100\\ 10\\ 1\end{bmatrix}$, $\mathbf{x}^{J}=\begin{bmatrix}1\\ 10\\ 100\end{bmatrix}$, $\mathbf{b}_1=\begin{bmatrix}100\\ 10000\end{bmatrix}$, and $\mathbf{b}_2=\begin{bmatrix}1\\ 0\\ 0\end{bmatrix}$.

So that $-\mathbf{Q}\mathbf{J}^{-1}=\begin{bmatrix}-20 & -1 & 0\\ -200 & -20 & -1\end{bmatrix}\begin{bmatrix}1 & 0 & 0\\ 10 & -0.1 & 0\\ 100 & -0.5 & -0.5\end{bmatrix}=\begin{bmatrix}-30 & 0.1 & 0\\ -500 & 2.5 & 0.5\end{bmatrix}$,

$$\mathbf{c}^{\mathrm{T}}\mathbf{J}^{-1}=\begin{bmatrix}100 & 10 & 1\end{bmatrix}\begin{bmatrix}1 & 0 & 0\\ 10 & -0.1 & 0\\ 100 & -0.5 & -0.5\end{bmatrix}=\begin{bmatrix}300 & -1.5 & -0.5\end{bmatrix},$$

$$\mathbf{c}^{\mathrm{T}}\mathbf{J}^{-1}\mathbf{b}_2=\begin{bmatrix}300 & -1.5 & -0.5\end{bmatrix}\begin{bmatrix}1\\ 0\\ 0\end{bmatrix}=300, \text{ and}$$

$$\mathbf{b}_1-\mathbf{Q}\mathbf{J}^{-1}\mathbf{b}_2=\begin{bmatrix}100\\ 10000\end{bmatrix}+\begin{bmatrix}-30 & 0.1 & 0\\ -500 & 2.5 & 0.5\end{bmatrix}\begin{bmatrix}1\\ 0\\ 0\end{bmatrix}=\begin{bmatrix}70\\ 9500\end{bmatrix}.$$

The objective jump tableau is shown as follows.

	s_3	s_4	s_5	x_1	x_2	x_3	s_1	s_2	RHS
Z	300	-1.5	-0.5	0	0	0	0	0	300
x_1	1	0	0	1	0	0	0	0	1
x_2	10	-0.1	0	0	1	0	0	0	10
x_3	100	-0.5	-0.5	0	0	1	0	0	100
s_1	-30	0.1	0	0	0	0	1	0	70
s_2	-500	2.5	0.5	0	0	0	0	1	9500

4) Enter nonbasic variables s_4, s_5 to basis as well as leave corresponding variables which are determined by the minimum ratio test, respectively. Finally, remove the artificial constraints from the tableau. The simplex method will starting with the following tableau.

s_3	s_1	s_2	x_1	x_2	x_3	s_4	s_5	RHS

Z	100	-10	1	0	0	0	0	0	9100
x_1	1	0	0	1	0	0	0	0	1
x_2	-20	1	0	0	1	0	0	0	80
x_3	200	-20	1	0	0	1	0	0	8200
s_4	-300	10	0	0	0	0	1	0	700
s_5	500	50	2	0	0	0	0	1	15500

2nd stage

1) Apply Dantzig's rule of the simplex method

	s_3	s_1	s_2	x_1	x_2	x_3	RHS
Z	100	-10	1	0	0	0	9100
x_1	1	0	0	1	0	0	1
x_2	-20	1	0	0	1	0	80
x_3	200	-20	1	0	0	1	8200

Iteration 1: Exchange the nonbasic variable s_1 with the basic variable x_2.

	s_3	x_2	s_2	x_1	s_1	x_3	RHS
Z	-100	10	-1	0	0	0	9900
x_1	1	0	0	1	0	0	1
s_1	-20	1	0	0	1	0	80
x_3	-200	20	1	0	0	1	9800

Iterations 2: Exchange the nonbasic variable s_3 with the basic variable x_1.

	x_1	x_2	s_2	s_3	s_1	x_3	RHS
Z	100	10	1	0	0	0	10000
s_3	1	0	0	1	0	0	1
s_1	20	1	0	0	1	0	100
x_3	200	20	1	0	0	1	10000

The optimal solution (x_1, x_2, x_3) is $(1, 100, 10000)$ and the optimal value is 10,000.This SOJU take 2 iterations.

Example 2. Consider the LP problem with redundant constraints [8].

$$
\begin{array}{lrcrcrcrcl}
\max & z = x_1 + x_2 + x_3 + x_4 & & & & & & & & \\
\text{s.t.} & x_1 & + & x_2 & + & x_3 & & & \leq & 2 \\
 & x_1 & + & x_2 & & & + & x_4 & \leq & 2 \\
 & x_1 & & & + & x_3 & + & x_4 & \leq & 2 \\
 & x_1 & & & & & & & \leq & 2 \\
 & & & x_2 & + & x_3 & + & x_4 & \leq & 2 \\
 & x_1 & - & 2x_2 & + & 2x_3 & + & 2x_4 & \leq & 4 \\
 & 2x_1 & + & 4x_2 & + & 3x_3 & + & 6x_4 & \leq & 24 \\
 & & & & & x_1, x_2, x_3, x_4 & & & \geq & 0
\end{array}
$$

1st stage

1) Calculate b_i and $\sum_{j=1}^{3} \frac{a_{ij}}{c_j}$. Choose the i^{th} constraint which has a positive smallest ratio $\frac{b_i}{\sum_{j=1}^{3} \frac{a_{ij}}{c_j}}$ and find the jump point.

i^{th} constraint	1	2	3	4	5	6	7
b_i	2	2	2	2	2	4	24
$\sum_{j=1}^{4} a_{ij}$	3	3	3	1	3	3	1
$\frac{b_i}{\sum_{j=1}^{4} a_{ij}}$	0.67	0.67	0.67	2	0.67	1.33	1.6

This objective jump is obtained by SOJU as follows:

	s_1	s_8	s_9	s_{10}	x_1	x_2	x_3	x_4	s_2	s_3	s_4	s_5	s_6	s_7	RHS
z	1.33	0	0	-0.33	0	0	0	0	0	0	0	0	0	0	2.67
x_1	0.33	0.5	0.17	0	1	0	0	0	0	0	0	0	0	0	0.67
x_2	0.33	-0.5	0.17	0	0	1	0	0	0	0	0	0	0	0	0.67
x_3	0.33	0	-0.33	0	0	0	1	0	0	0	0	0	0	0	0.67
x_4	0.33	0	0	-0.33	0	0	0	1	0	0	0	0	0	0	0.67
s_2	-1	0	-0.33	0.33	0	0	0	0	1	0	0	0	0	0	0
s_3	-1	-0.5	0.17	0.33	0	0	0	0	0	1	0	0	0	0	0
s_4	-0.33	-0.5	-0.17	0	0	0	0	0	0	0	1	0	0	0	1.33
s_5	-1	0.5	0.17	0.33	0	0	0	0	0	0	0	1	0	0	0
s_6	-1	-1.5	0.83	0.67	0	0	0	0	0	0	0	0	1	0	2
s_7	-5	1	0	2	0	0	0	0	0	0	0	0	0	1	14

2) Exchange the nonbasic variable s_8 with the basic variable s_5. Eliminate rows and columns of s_8.

	s_1	s_5	s_9	s_{10}	x_1	x_2	x_3	x_4	s_2	s_3	s_4	s_6	s_7	RHS
z	1.33	0	0	-0.33	0	0	0	0	0	0	0	0	0	2.67
x_1	1.33	-1	0	-0.33	1	0	0	0	0	0	0	0	0	0.67
x_2	-0.67	1	0.33	0.33	0	1	0	0	0	0	0	0	0	0.67
x_3	0.33	0	-0.33	0	0	0	1	0	0	0	0	0	0	0.67
x_4	0.33	0	0	-0.33	0	0	0	1	0	0	0	0	0	0.67
s_2	-1	0	-0.33	0.33	0	0	0	0	1	0	0	0	0	0
s_3	-2	1	0.33	0.67	0	0	0	0	0	1	0	0	0	0
s_4	-1.33	1	0	0.33	0	0	0	0	0	0	1	0	0	1.33
s_6	-4	3	1.33	1.67	0	0	0	0	0	0	0	1	0	2
s_7	-3	-2	-0.33	1.33	0	0	0	0	0	0	0	0	1	14

3) Exchange the nonbasic variable s_9 with the basic variable s_3. Eliminate rows and columns of s_9.

	s_1	s_5	s_3	s_{10}	x_1	x_2	x_3	x_4	s_2	s_4	s_8	s_6	s_7	RHS
z	-1.33	0	0	3.33	0	0	0	0	0	0	0	0	0	2.67
x_1	1.33	0	-1	-0.33	1	0	0	0	0	0	0	0	0	0.67
x_2	1.33	0	-1	-0.33	0	1	0	0	0	0	0	0	0	0.67
x_3	-1.67	0	2	0.67	0	0	1	0	0	0	0	0	0	0.67
x_4	0.33	0	0	-0.33	0	0	0	1	0	0	0	0	0	0.67
s_2	-3	0	2	1	0	0	0	0	1	0	0	0	0	0
s_4	-1.33	0	1	0.33	0	0	0	0	0	1	0	0	0	1.33
s_8	0	1	-1	0	0	0	0	0	0	0	1	0	0	0
s_6	4	0	-5	-1	0	0	0	0	0	0	0	1	0	2
s_7	-5	0	0	2	0	0	0	0	0	0	0	0	1	14

4) Exchange the nonbasic variable s_{10} with the basic variable s_2. Eliminate rows and columns of s_{10}.

	s_1	s_5	s_3	s_2	x_1	x_2	x_3	x_4	s_4	s_6	s_7	RHS
z	0.33	0	1	0	0	0	0	0	0	0	0	2.67
x_1	0.33	0	-0.33	0.3	1	0	0	0	0	0	0	0.67
x_2	0.33	0	-0.33	0.3	0	1	0	0	0	0	0	0.67
x_3	0.33	0	0.67	-1	0	0	1	0	0	0	0	0.67
x_4	-0.67	0	0.67	0.3	0	0	0	1	0	0	0	0.67
s_4	0.33	0	0.33	-0	0	0	0	0	1	0	0	1.33
s_6	1	-3	1	0	0	0	0	0	0	1	0	2
s_7	1	0	-4	-2	0	0	0	0	0	0	1	14

After the first stage of SOJU, this tableau is the optimal tableau for the simplex method. Finally, the optimal solution is $(x_1, x_2, x_3, x_4) = \left(\frac{2}{3}, \frac{2}{3}, \frac{2}{3}, \frac{2}{3}\right)$ and the optimal value is 2.67.

Example 3. Consider the following problem.

$$\begin{aligned} \max \quad & z = 6x_1 + 5x_3 \\ \text{s.t.} \quad & -4x_1 + 2x_2 + x_3 \le 4 \\ & 2x_1 + \quad x_3 \le 20 \\ & x_1 - x_2 - x_3 \le 10 \\ & x_1,\ x_2,\ x_3 \ge 0 \end{aligned}$$

1st stage

1) The gradient of the objective function is $(c_1, c_2, c_3) = (6, 0, 5)$. Therefore, the value $p = 2$ and l=1. Since the 2nd constraint has $\frac{a_{21}}{c_1} + \frac{a_{23}}{c_3} = \frac{2}{5} + \frac{1}{5} = \frac{8}{15} > 0$ and a positive minimum ratio is equal to $20\left(\frac{15}{8}\right) = \frac{75}{2}$. So, the jump point is $(x_1^J, x_2^J, x_3^J) = \left(\frac{1}{6} \times \frac{75}{2},\ 0,\ \frac{1}{5} \times \frac{75}{2}\right) = \left(\frac{25}{4},\ 0,\ \frac{75}{10}\right)$.

2) Add artificial constraints and generate the sub-LP problem in the standard form as follow:

$$\begin{aligned} \max \quad & z = 6x_1 + 5x_3 \\ \text{s.t.} \quad & -4x_1 + 2x_2 + x_3 + s_1 = 4 \\ & x_1 - \ x_2 - x_3 + s_2 = 10 \\ & 2x_1 \quad + x_3 + s_3 = 20 \\ & 6x_1 \quad - 5x_3 + s_4 = 0 \\ & \quad x_2 \quad + s_5 = 0 \\ & x_1, x_2, x_3, s_1, s_2, s_3, s_4, s_5 \ge 0 \end{aligned}$$

3) Create the objective jump tableau by calculating $\mathbf{J}^{-1}$, $-\mathbf{Q}\mathbf{J}^{-1}$, $\mathbf{c}^{\mathrm{T}}\mathbf{J}^{-1}$, $\mathbf{c}^{\mathrm{T}}\mathbf{J}^{-1}\mathbf{b}_2$, and $\mathbf{b}_1 - \mathbf{Q}\mathbf{J}^{-1}\mathbf{b}_2$.

Since, $\mathbf{J} = \begin{bmatrix} 2 & 0 & 1 \\ 6 & 0 & -5 \\ 0 & 1 & 0 \end{bmatrix}$ then $\mathbf{J}^{-1} = \begin{bmatrix} \frac{5}{16} & \frac{1}{16} & 0 \\ 0 & 0 & 1 \\ \frac{3}{8} & -\frac{1}{8} & 0 \end{bmatrix}$. Moreover, $\mathbf{Q} = \begin{bmatrix} -4 & 2 & 1 \\ 1 & -1 & -1 \end{bmatrix}$,

$$\mathbf{c} = \begin{bmatrix} 6 \\ 0 \\ 5 \end{bmatrix}, \mathbf{x}^J = \begin{bmatrix} \frac{25}{4} \\ 0 \\ \frac{75}{10} \end{bmatrix}, \mathbf{b}_1 = \begin{bmatrix} 4 \\ 10 \end{bmatrix}, \text{ and } \mathbf{b}_2 = \begin{bmatrix} 20 \\ 0 \\ 0 \end{bmatrix}.$$

$$\text{So that } -\mathbf{Q}\mathbf{J}^{-1} = \begin{bmatrix} 4 & -2 & -1 \\ -1 & 1 & 1 \end{bmatrix} \begin{bmatrix} \frac{5}{16} & \frac{1}{16} & 0 \\ 0 & 0 & 1 \\ \frac{3}{8} & -\frac{1}{8} & 0 \end{bmatrix} = \begin{bmatrix} \frac{7}{8} & \frac{3}{8} & -2 \\ \frac{1}{16} & -\frac{3}{16} & 1 \end{bmatrix},$$

$$\mathbf{c}^{\mathrm{T}}\mathbf{J}^{-1} = \begin{bmatrix} 6 & 0 & 5 \end{bmatrix} \begin{bmatrix} \frac{5}{16} & \frac{1}{16} & 0 \\ 0 & 0 & 1 \\ \frac{3}{8} & -\frac{1}{8} & 0 \end{bmatrix} = \begin{bmatrix} \frac{15}{4} & -\frac{1}{4} & 0 \end{bmatrix},$$

$$\mathbf{c}^{\mathrm{T}}\mathbf{J}^{-1}\mathbf{b}_2 = \begin{bmatrix} \frac{15}{4} & -\frac{1}{4} & 0 \end{bmatrix} \begin{bmatrix} 20 \\ 0 \\ 0 \end{bmatrix} = 75,$$

$$\text{and} \quad \mathbf{b}_1 - \mathbf{Q}\mathbf{J}^{-1}\mathbf{b}_2 = \begin{bmatrix} 4 \\ 10 \end{bmatrix} + \begin{bmatrix} \frac{7}{8} & \frac{3}{8} & -2 \\ \frac{1}{16} & -\frac{3}{16} & 1 \end{bmatrix} \begin{bmatrix} 20 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 4 \\ 10 \end{bmatrix} + \begin{bmatrix} \frac{35}{2} \\ \frac{5}{4} \end{bmatrix} = \begin{bmatrix} \frac{43}{2} \\ \frac{45}{4} \end{bmatrix}.$$

The objective jump tableau is shown as follows.

	s_3	s_4	s_5	x_1	x_2	x_3	s_1	s_2	RHS
z	15/4	-1/4	0	0	0	0	0	0	75
x_1	5/16	1/16	0	1	0	0	0	0	25/4
x_2	0	0	1	0	1	0	0	0	0
x_3	3/8	-1/8	0	0	0	1	0	0	15/2
s_1	7/8	3/8	-2	0	0	0	1	0	0
s_2	1/16	-3/16	1	0	0	0	0	1	0

Enter nonbasic variables s_4, s_5 to basis as well as leave corresponding variables which are determined by minimum ratio test, respectively. Finally, remove the artificial constraints from the tableau.

	s_3	s_4	s_5	x_1	x_2	x_3	s_1	s_2	RHS
z	13/3	0	-4/3	0	0	0	2/3	0	75
x_1	1/6	0	1/3	1	0	0	-1/6	0	25/4
x_2	0	0	1	0	1	0	0	0	0
x_3	2/3	0	-2/3	0	0	1	1/3	0	0
s_4	7/3	1	-16/3	0	0	0	8/3	0	0
s_2	1/2	0	0	0	0	0	1/2	1	0

	s_3	s_5	x_1	x_2	x_3	s_1	s_2	RHS
z	13/3	0	0	4/3	0	2/3	0	75
x_1	1/6	0	1	-1/3	0	-1/6	0	25/4
s_5	0	1	0	1	0	0	0	0
x_3	2/3	0	0	-2/3	1	1/3	0	15/2
s_2	1/2	0	0	0	0	1/2	1	0

The simplex method will starting with the following tableau.

	s_3	x_1	x_2	x_3	s_1	s_2	RHS
z	13/3	0	4/3	0	2/3	0	75
x_1	1/6	1	-1/3	0	-1/6	0	25/4
x_3	2/3	0	-2/3	1	1/3	0	0
s_2	1/2	0	0	0	1/2	1	15/2

Finally, the optimal solution is $(x_1, x_2, x_3) = \left(\frac{25}{4}, 0, \frac{15}{2}\right)$ and the optimal value is 75.

CHAPTER 4
EXPERIMENTS AND RESULTS

In this chapter, a number of test problems were randomly generated to test the performance of SOJU comparing with that of the traditional simplex method with Dantzig's rule, which we will refer to as SIMP in this chapter. All tests were performed on Intel(R) Core(TM) i7-5500U CPU @2.40GHz and 8.00GB RAM with Windows 10 Pro laptop, using Python 3.5.2 by Jupyter Notebook [3]. Time is measured in seconds.

All problems are generated in the form of LP model (3.4) with the following parameters: (1) the number of variables, (2) the number of constraints, (3) the range of values for the objective coefficients, (4) the range for constraint coefficients, and (5) the initial point that will be used to calculate a valid value of RHS. These parameters are explained as follows:

- The parameters m and n represent the number of constraints and variables, respectively, where $n \geq m$.
- The range of a random objective coefficients for each positive entry is given by (0, 10], i.e. $c_i \in (0,10]$ for $j = 1,2,3,\ldots,n$.
- The range of a random entry of the coefficient matrix is given by [-10, 10], i.e. $a_{ij} \in [-10,10]$ for $i = 1,2,3,\ldots,m,\ j = 1,2,3,\ldots,n,$.
- A values for RHS entry is calculated from $\mathbf{Ax}$ where the value of the variable a random variable x_i, $i = 1,2,3,\ldots,m$ is randomly selected from [0, 10]. If $(\mathbf{Ax})_i \geq 0$ $(\mathbf{Ax})_i \geq 0$, set $\mathbf{b}_i = (\mathbf{Ax})_i$. If $(\mathbf{Ax})_i < 0$, then let $\mathbf{b}_i = -(\mathbf{Ax})_i$. This computing guarantees nonnegative RHS entries.

For each row, a sum of the quotient of coefficient entry divided by the cost coefficient is also computed to make sure that at least one constraint i must satisfy the condition $\sum_{j=1}^{n} \frac{a_{ij}}{c_j} > 0$. If they are all negative, the coefficients in the last constraint will be multiplied by -1.

4.1 Test problems

Test I

Test problems were generated to create a set of linear programming problems with three different sizes: small sizes for problems with 10, 20, 30, 40 and 50 constraints; medium sizes for problems with 100, 200, 300 and 400 constraints; and large sizes for the problems with 500 and 1000 constraints. For each problem size, the number of variables was varied as 2, 3, 4, 5, 10, 50, 100, 200, 300, 400, 500, 1000 variables as long as it does not exceed the number of constraints. Moreover, 10 problem instances were generated for each size configuration.

Test II

This test was carried out due to the speculation that SOJU would perform well when the number of variables is much smaller than the number of constraints. Test problems were generated to create a set of linear programming problems with small variables: 2, 3, 4, 5, 10 and 15 variables. For each variables, the large number of constraints was varied as 1000, 2000, 3000, …, 15000 constraints. Moreover, 5 problem instances were generated for each size configuration.

4.2 Results and analysis

The computational experiments of this simplex method with objective jump were divided into two stages represented by SOJU1 and SOJU2. The first stage, SOJU1, would find an initial basic feasible solution using the objective jump procedure. The second stage, SOJU2, would determine the optimal solution starting from the initial basic feasible solution from the first stage using traditional simplex method.

Total stages, denoted by TSOJU, was compared to SIMP algorithm in which the optimal solution were found by the simplex method with the Dantzig's pivoting rule. For comparison, the experiments reported average number of iterations and running time of SOJU1, SOJU2 and TSOJU versus SIMP.

Tables 4.1-4.4 provide each case of the average number of iterations and running times in Test I. To compare the algorithm, we present plots of average number of iterations in Figures 4.1-4.15 and average running time reported in Figures 4.16-4.30.

Table 4.1: The average number of iterations and average running time by SIMP and SOJU with small problem sizes.

Problem size		Average no. of iterations				Average running time (sec.)				
m	*n*	SOJU 1st stage	SOJU 2nd stage	Total SOJU	SIMP	SOJU 1st stage	SOJU 2nd stage	Total SOJU	SIMP	% Diff.
10	2	1	0.5	1.5	3.0	0.0006	0.0002	0.0008	0.0005	59.8
	3	2	1.3	3.3	4.5	0.0004	0.0005	0.0009	0.0005	80.2
	4	3	2.1	5.1	5.5	0.0005	0.0004	0.0009	0.0009	0.1
	5	4	2.5	6.5	5.7	0.0012	0.0003	0.0015	0.0016	-8.0
	10	9	6.6	15.6	10.1	0.0023	0.0020	0.0043	0.0024	79.4
20	2	1	1.3	2.3	2.7	0.0002	0.0006	0.0008	0.0014	-42.8
	3	2	1.4	3.4	4.2	0.0005	0.0006	0.0011	0.0017	-35.2
	4	3	4.0	7.0	6.8	0.0008	0.0019	0.0027	0.0026	3.9
	5	4	5.0	9.0	9.1	0.0010	0.0026	0.0036	0.0049	-26.5
	10	9	9.6	18.6	12.4	0.0026	0.0060	0.0086	0.0070	23.0
	20	11	20.6	39.6	26.0	0.0085	0.0145	0.0230	0.0166	38.7
30	2	1	1.0	2.0	3.2	0.0006	0.0009	0.0015	0.0025	-39.9
	3	2	2.5	4.5	5.1	0.0010	0.0015	0.0025	0.0039	-35.8
	4	3	3.6	6.6	6.7	0.0012	0.0029	0.0041	0.0051	-19.5
	5	4	5.8	9.8	7.9	0.0017	0.0061	0.0078	0.0062	25.9
	10	9	13.5	22.5	14.7	0.0044	0.0166	0.0210	0.0137	53.4
	20	19	28.5	47.6	29.2	0.0131	0.0378	0.0509	0.0287	77.4
	30	29	39.8	68.8	43	0.0277	0.0528	0.0805	0.0661	29.8
40	2	1	0.7	1.7	3.0	0.0007	0.0008	0.0015	0.0039	-61.5
	3	2	3.7	5.7	4.6	0.0014	0.0046	0.0060	0.0061	-1.6
	4	3	4.4	7.4	7.1	0.0019	0.0060	0.0079	0.0124	-36.3
	5	4	6.2	10.2	9.3	0.0026	0.0091	0.0117	0.0122	-4.0
	10	9	14.5	23.5	17.1	0.0065	0.0214	0.0279	0.0252	10.8
	20	19	28.6	47.6	34.4	0.0186	0.0603	0.0789	0.0561	40.7
	30	29	46.1	75.1	57.8	0.0407	0.1098	0.1505	0.1272	18.3
	40	39	64.0	103.0	74.0	0.0720	0.1538	0.2258	0.2017	12.0
50	2	1	1.6	2.6	3.0	0.0012	0.0034	0.0046	0.0057	-19.2
	3	2	2.7	4.7	5.4	0.0019	0.0048	0.0067	0.0118	-43.2
	4	3	4.3	7.3	8.0	0.0027	0.0078	0.0105	0.0161	-34.7
	5	4	6.7	10.7	10.0	0.0041	0.0146	0.0187	0.0181	3.4
	10	9	17.6	26.6	18.8	0.0092	0.0441	0.0533	0.0397	34.3
	20	19	32.9	51.9	41.9	0.0255	0.0985	0.1240	0.1130	9.7
	30	29	54.9	83.9	60.5	0.0543	0.1742	0.2286	0.1902	20.2
	40	39	78.5	117.5	81.5	0.0920	0.2124	0.3044	0.2994	1.7
	50	49	88.4	137.4	99.4	0.1362	0.1651	0.3013	0.3270	-7.9

Table 4.2: The average number of iterations and average running time by SIMP and SOJU with medium problem sizes.

Problem size		Average no. of iterations				Average running time (sec.)				
m	n	SOJU 1st stage	SOJU 2nd stage	Total SOJU	SIMP	SOJU 1st stage	SOJU 2nd stage	Total SOJU	SIMP	% Diff.
100	2	1	1.3	2.3	3.5	0.0043	0.0092	0.0135	0.0251	-46.17
	3	2	3.2	5.2	5.8	0.0067	0.0172	0.0239	0.0520	-53.98
	4	3	5.5	8.5	7.7	0.0097	0.0311	0.0408	0.0518	-21.19
	5	4	7.4	11.4	12.2	0.0127	0.1616	0.1744	0.3058	-42.99
	10	9	21.6	30.6	26.3	0.0295	0.1614	0.1909	0.1817	5.05
	20	19	44.3	63.3	47.2	0.0771	0.3632	0.4403	0.3676	19.77
	30	29	65.3	94.3	66.9	0.1445	0.5108	0.6553	0.5399	21.36
	40	39	92.6	131.6	98.5	0.2097	0.7206	0.9303	0.8269	12.49
	50	49	136.6	185.6	143.7	0.3196	1.0986	1.4182	1.3046	8.71
	100	99	350.4	432.5	350.2	0.9831	3.4265	4.4096	4.6653	-5.48
200	2	1	1	2.0	3.2	0.0133	0.0240	0.0373	0.0868	-56.97
	3	2	2.3	4.3	5.1	0.0238	0.0480	0.0718	0.1481	-51.54
	4	3	5.5	8.5	7.7	0.0376	0.0311	0.0687	0.0518	32.61
	5	4	7.4	11.4	12.2	0.0461	0.1616	0.2077	0.3058	-32.07
	10	9	24.8	33.8	30.2	0.1026	0.6168	0.7194	0.7629	-5.71
	20	19	52.7	71.7	64.7	0.2577	1.3103	1.5680	1.7392	-9.84
	30	29	89.4	118.4	100.9	0.4181	2.2768	2.6949	2.6925	0.09
	40	39	132.7	171.7	131.3	0.6072	3.3757	3.9829	3.6750	8.38
	50	49	142.0	191.0	151.5	0.8098	3.8147	4.6245	4.4547	3.81
	100	99	394.2	493.2	453.1	2.2339	18.2148	20.4487	24.8362	-17.67
	200	199	1605.9	1804.9	1622.0	7.9871	217.8548	225.8419	254.5809	-11.29

Table 4.3: The average number of iterations and average running time by SIMP and SOJU with medium problem sizes.

Problem size		Average no. of iterations				Average running time (sec.)				
m	n	SOJU 1st stage	SOJU 2nd stage	Total SOJU	SIMP	SOJU 1st stage	SOJU 2nd stage	Total SOJU	SIMP	% Diff.
300	2	1	1.0	2.0	3.1	0.0307	0.0518	0.0825	0.1885	-56.24
	3	2	3.1	5.1	5.6	0.0527	0.1437	0.1964	0.3404	-42.29
	4	3	5.6	8.6	7.6	0.0806	0.2592	0.3399	0.4471	-23.99
	5	4	7.2	11.2	11.1	0.0998	0.3417	0.4415	0.6459	-31.65
	10	9	22.0	31.0	31.9	0.2316	1.1453	1.3769	1.8083	-23.86
	20	19	57.0	76.0	70.7	0.5312	3.2218	3.7530	4.1833	-10.29
	30	29	114.7	143.7	122.3	0.8319	6.6137	7.4455	7.4521	-0.09
	40	39	141.8	180.8	144.9	1.2412	7.5880	8.8292	9.0594	-2.54
	50	49	184.6	233.6	212.8	1.5623	9.9544	11.5166	14.3144	-19.55
	100	99	430.3	529.3	437.8	3.9801	29.6278	33.6079	32.2917	4.08
	200	199	1549.7	1748.7	1603.0	12.7937	321.4763	334.2699	385.8594	-13.37
	300	299	2715.9	3014.9	2815.5	26.4250	292.6399	319.0649	364.7053	-12.51
400	2	1	0.9	1.9	3.4	0.0537	0.0851	0.1389	0.3874	-64.16
	3	2	2.7	4.7	5.9	0.0949	0.2357	0.3306	0.6568	-49.67
	4	3	6.2	9.2	8.1	0.1392	0.5408	0.6800	0.8617	-21.09
	5	4	9.1	13.1	12.1	0.1762	0.7903	0.9665	1.2657	-23.63
	10	9	24.4	33.4	25.4	0.4223	2.3234	2.7457	2.6512	3.57
	20	19	59.4	78.4	68.3	0.9143	6.0508	6.9651	7.3323	-5.01
	30	29	107.3	136.3	111.2	1.4159	10.9527	12.3686	12.3417	0.22
	40	39	159.5	198.5	163.7	1.9726	15.6701	17.6427	18.6046	-5.17
	50	49	200.6	249.6	217.5	2.5378	20.5882	23.1260	25.4054	-8.97
	100	99	464.1	495.5	14.7	6.1652	45.3392	51.5044	62.3063	-17.34
	200	199	1605.9	1804.9	1622	18.3344	217.8548	236.1892	254.5809	-7.22
	300	299	3222.5	3521.5	3425.2	36.2887	553.6082	589.8969	691.8511	-14.74
	400	399	4749.0	5148.0	4855.5	63.3971	977.0197	1040.4168	1177.6710	-11.65

From Tables 4.1-4.3, the second stage of SOJU almost always yields smaller number of iterations and running time than SIMP. This implies that the new initial basis provided by SOJU gives better performance than traditional basis used in SIMP. Unfortunately, there is a price to pay to obtain the SOJU initial basis which occurs during the first stage of SOJU. To be fair, the total number of iterations and running time from both stages of SOJU should be used to compare with SIMP. From the tables, if the total number of iterations of SOJU is less than SIMP, the running time will also be smaller. These cases often occur when n is much smaller than m. However, if the total number of iterations of SOJU is larger than SIMP, SOJU may still be faster in some cases, which often occurs when $m \geq 300$ and $n \geq 40$.

Table 4.4: The average number of iterations and running time by SIMP and SOJU with large problem sizes.

Problem size		Average no. of iterations				Average running time (sec.)				
m	*n*	SOJU 1st stage	SOJU 2nd stage	Total SOJU	SIMP	SOJU 1st stage	SOJU 2nd stage	Total SOJU	SIMP	% Diff.
500	2	1	1.3	2.3	3.3	0.0831	0.2103	0.2934	0.5749	-49.0
	3	2	3.6	5.6	5.9	0.1538	0.5035	0.6574	0.9996	-34.2
	4	3	5.4	8.4	8.2	0.2133	0.7619	0.9752	1.3906	-29.9
	5	4	7.9	11.9	12.3	0.2747	1.1025	1.3771	2.0675	-33.4
	10	9	25.8	34.8	30.3	0.6382	3.9505	4.5886	5.0520	-9.2
	20	19	65	84.0	67.4	1.4126	10.3405	11.7531	11.2153	4.8
	30	29	110.6	139.6	120.8	2.1356	17.3395	19.4751	19.2092	1.4
	40	39	163.4	202.4	170.6	2.9036	25.2479	28.1515	29.8839	-5.8
	50	49	207.7	256.7	238.4	3.8553	31.8784	35.7337	42.6190	-16.2
	100	99	486.3	585.3	491.6	8.9480	86.2108	95.1588	93.3028	2.0
	200	199	1549.7	1748.7	1603.0	24.6166	321.4763	346.0929	385.8594	-10.3
	300	299	3467.1	3766.1	3552.2	47.6427	904.2718	951.9145	1079.2690	-11.8
	400	399	5041.2	5440.2	5478.2	79.8347	1522.8154	1602.6501	1880.8721	-5.8
	500	499	6824.0	7323.0	7083.0	123.9207	2347.2343	2471.1549	2851.0051	-13.3
1000	2	1	1.0	1.0	3.2	0.3173	0.6383	1.1086	2.6267	-58.7
	3	2	2.4	2.4	5.6	0.5668	1.4337	2.2217	4.1371	-61.2
	4	3	5.3	5.3	9.2	0.8152	3.2922	4.3969	6.6869	-69.4
	5	4	8.7	8.7	12.1	1.0744	5.2069	6.6381	9.0701	-68.0
	10	9	29.1	29.1	29.4	2.3949	18.0354	18.9534	19.9798	-62.9
	20	19	59.4	59.4	67.7	5.0440	36.1956	38.0356	46.5377	-60.5
	30	29	111.5	111.5	125.1	7.7980	67.9288	70.7025	87.9675	-66.1
	40	39	171.8	171.8	180.9	10.7085	106.0429	109.7645	124.6988	-66.0
	50	49	222.6	222.6	248.7	13.7127	148.3861	153.0553	189.5792	-70.6
	100	99	616.4	616.4	608.7	30.4475	434.5165	471.2869	443.5627	-65.0
	200	199	1391.0	1391.0	1323.6	72.9461	1100.3228	1184.8193	1107.4488	-65.6
	300	299	2813.9	2813.9	2741.0	128.0439	2606.6703	2750.3465	2685.5203	1.8
	400	399	5577.6	5577.6	5478.2	195.8104	5584.9723	5799.3454	6119.5753	-5.5
	500	499	9502.0	9502.0	9419.5	281.8883	10338.3137	10640.0304	11968.5490	-11.3
	1000	999	29038.0	29038.0	30756.8	1002.4307	46174.9741	46211.8328	55297.7483	-15.6

For Test II, the results are shown in Figures 4.31-4.36 which present plots of average number of iterations and average running time report in Tables 4.5-4.7.

Table 4.5: The average number of iterations and running time by SIMP and SOJU with small number of variables and large number of constraints.

Problem size		Average no. of iterations				Average running time (sec.)				
n	*m*	SOJU 1st stage	SOJU 2nd stage	Total SOJU	SIMP	SOJU 1st stage	SOJU 2nd stage	Total SOJU	SIMP	% Diff.
2	1000	1	2.2	3.2	3.4	0.3127	0.5709	0.8836	1.0183	-13.2
	2000	1	2.0	3.0	3.2	1.2970	1.9410	3.2380	3.9053	-17.1
	3000	1	2.6	3.6	3.0	2.9807	5.6843	8.6651	8.2034	5.6
	4000	1	2.0	3.0	2.8	5.2758	7.9578	13.2336	13.3881	-1.2
	5000	1	1.6	2.6	2.8	8.8431	9.7991	18.6422	21.1736	-12.0
	6000	1	2.0	3.0	2.8	12.5082	17.8418	30.3500	30.5334	-0.6
	7000	1	1.4	2.4	3.2	17.4497	17.5091	34.9588	394.0257	-91.1
	8000	1	1.2	2.2	2.6	22.7509	19.3734	42.1243	52.6989	-20.1
	9000	1	1.4	2.4	3.4	28.6950	28.9376	57.6326	85.2961	-32.4
	10000	1	1.6	2.6	3.0	35.5583	41.0473	76.6057	92.9273	-17.6
	11000	1	1.4	2.4	3.8	44.4834	43.1784	87.6618	143.7342	-39.0
	12000	1	2.0	3.0	2.6	56.3240	74.9676	131.2917	121.4375	8.1
	13000	1	1.8	2.8	2.8	66.1342	78.3266	144.4608	150.8353	-4.2
	14000	1	2.2	3.2	3.0	79.0778	114.6711	193.7488	186.1854	4.1
	15000	1	2.4	3.4	3.8	88.5608	144.9023	233.4631	261.5471	-10.7
3	1000	2	4.0	6.0	5.6	0.5673	0.9560	1.5233	1.6123	-5.5
	2000	2	2.6	4.6	5.0	2.3278	2.5482	4.8760	5.8632	-16.8
	3000	2	3.6	5.6	5.0	5.2303	8.0000	13.2303	13.2712	-0.3
	4000	2	2.8	4.8	6.2	9.4271	11.0120	20.4390	29.9503	-31.8
	5000	2	4.4	6.4	6.4	14.5702	27.1917	41.7619	47.2784	-11.7
	6000	2	3.4	5.4	6.2	20.8962	30.2846	51.1808	67.0965	-23.7
	7000	2	3.6	5.6	5.4	28.8421	43.9705	72.8127	80.2835	-9.3
	8000	2	4.0	6.0	6.4	38.3835	63.6617	102.0452	118.1041	-13.6
	9000	2	4.0	6.0	5.6	49.3019	80.8846	130.1866	131.2194	-0.8
	10000	2	4.0	6.0	5.6	59.6988	99.5071	159.2060	164.0864	-3.0
	11000	2	4.4	6.4	6.6	75.6571	131.9914	207.6484	233.5866	-11.1
	12000	2	4.0	6.0	5.2	90.2816	147.3178	237.5994	220.2686	7.9
	13000	2	4.4	6.4	5.8	106.0539	189.0512	295.1051	287.9528	2.5
	14000	2	2.8	4.8	6.0	118.4918	149.7953	268.2871	349.7494	-23.3
	15000	2	3.2	5.2	6.0	133.6533	197.7966	331.4500	392.6448	-15.6

Table 4.6: The average number of iterations and running time by SIMP and SOJU with small number of variables and large number of constraints.

Problem size		Average no. of iterations				Average running time (sec.)				
n	*m*	SOJU 1st stage	SOJU 2nd stage	Total SOJU	SIMP	SOJU 1st stage	SOJU 2nd stage	Total SOJU	SIMP	% Diff.
4	1000	3	6.2	9.2	9.0	0.8368	1.4966	2.3335	2.5617	-8.9
	2000	3	6.6	9.6	8.0	3.2380	6.3507	9.5887	9.1129	5.2
	3000	3	5.0	8.0	10.4	7.4146	11.1363	18.5509	26.7125	-30.6
	4000	3	6.0	9.0	9.0	13.1959	23.9269	37.1228	41.3230	-10.2
	5000	3	5.2	8.2	9.8	20.3515	32.8178	53.1693	69.3708	-23.4
	6000	3	5.0	8.0	8.4	29.4183	46.3045	75.7228	85.9936	-11.9
	7000	3	7.8	10.8	9.6	41.1318	96.8856	138.0174	138.2478	-0.2
	8000	3	6.8	9.8	8.6	53.3547	111.6801	165.0349	157.4819	4.8
	9000	3	5.0	8.0	6.4	67.0141	107.0023	174.0163	150.2575	15.8
	10000	3	7.0	10.0	7.6	82.4976	183.6653	266.1629	217.0185	22.6
	11000	3	5.4	8.4	8.2	106.1110	173.0695	279.1805	287.0784	-2.8
	12000	3	6.8	9.8	10.8	126.4462	260.8316	387.2778	440.3629	-12.1
	13000	3	6.4	9.4	8.4	152.1075	288.1232	440.2307	409.7139	7.4
	14000	3	5.0	8.0	8.0	180.0171	280.5067	460.5238	457.7387	0.6
	15000	3	7.0	10.0	8.4	204.3029	447.5344	651.8373	538.8584	21.0
5	1000	4	8.2	12.2	11.2	1.0895	1.9675	4.8115	3.1978	-4.4
	2000	4	8.4	12.4	10.0	4.3197	8.1373	19.4318	11.3658	9.6
	3000	4	6.4	10.4	10.0	9.9386	14.2884	40.3892	25.7296	-5.8
	4000	4	10.0	14.0	12.8	18.4244	40.0878	87.9724	58.5290	0.0
	5000	4	10.0	14.0	11.8	28.6239	63.1243	138.5021	83.5535	9.8
	6000	4	9.4	13.4	10.4	41.2515	86.4931	196.2464	106.0221	20.5
	7000	4	8.8	12.8	12.6	57.0492	110.3759	258.6079	175.1819	-4.4
	8000	4	8.4	12.4	11.8	74.7882	139.7157	336.2271	214.0058	0.2
	9000	4	9.6	13.6	12.4	94.2494	205.5763	452.1835	285.7649	4.9
	10000	4	9.0	13.0	12.8	116.5282	237.8949	541.9302	361.5506	-2.0
	11000	4	10.0	14.0	13.6	144.6622	316.3620	687.7191	470.7470	-2.1
	12000	4	8.8	12.8	11.8	172.2180	344.2547	792.8823	484.8409	6.5
	13000	4	7.2	11.2	10.4	203.6305	337.0466	860.3991	505.9683	6.9
	14000	4	8.8	12.8	9.6	239.6545	491.7583	1105.1184	544.9246	34.2
	15000	4	9.6	13.6	12.6	271.0079	621.8891	1343.1265	803.1660	11.2

Table 4.7: The average number of iterations and running time by SIMP and SOJU with small number of variables and large number of constraints.

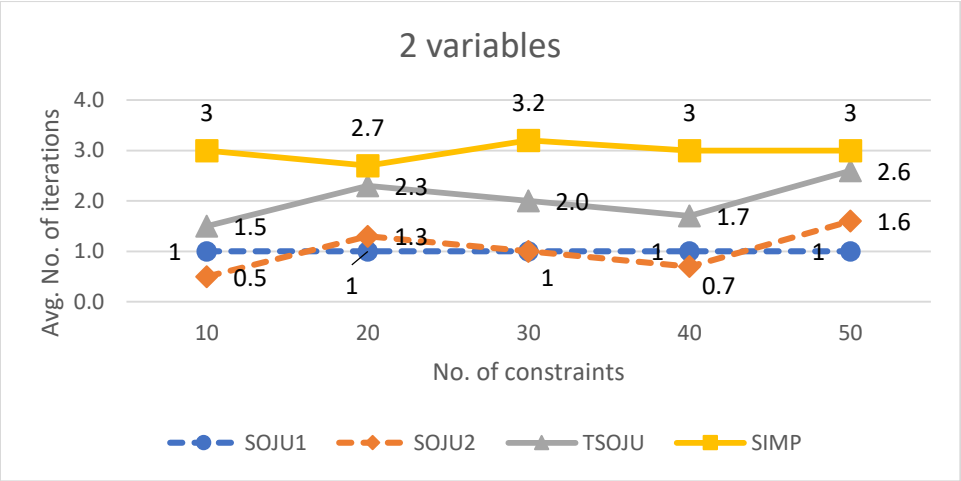
Problem size		Average no. of iterations				Average running time (sec.)				
n	*m*	SOJU 1st stage	SOJU 2nd stage	Total SOJU	SIMP	SOJU 1st stage	SOJU 2nd stage	Total SOJU	SIMP	% Diff.
10	1000	9	28.4	37.4	28.2	2.4824	6.8525	9.3349	7.9794	17.0
	2000	9	24.6	33.6	29.0	9.6035	23.7059	33.3094	32.2646	3.2
	3000	9	23.0	32.0	30.6	23.3050	51.0358	74.3408	77.0919	-3.6
	4000	9	26.0	35.0	28.2	43.0517	103.0584	146.1101	127.1255	14.9
	5000	9	22.0	31.0	25.4	68.6751	138.7661	207.4412	177.6019	16.8
	6000	9	22.8	31.8	29.6	100.2209	206.3161	306.5370	299.9554	2.2
	7000	9	21.0	30.0	27.4	139.2391	262.9681	402.2071	377.1423	6.6
	8000	9	25.4	34.4	32.8	184.2237	417.1254	601.3490	586.8790	2.5
	9000	9	20.0	29.0	27.8	233.0995	429.6708	662.7703	629.8877	5.2
	10000	9	22.0	31.0	29.8	287.5293	569.4184	856.9477	829.0433	3.4
	11000	9	22.0	31.0	28.6	353.7068	683.7364	1037.4432	983.3912	5.5
	12000	9	22.2	31.2	26.6	420.6811	836.0746	1256.7557	1080.1659	16.3
	13000	9	24.6	33.6	26.8	502.4979	1106.9756	1609.4735	1295.1735	24.3
	14000	9	26.2	35.2	32.0	581.7821	1439.6674	2021.4494	1813.1138	11.5
	15000	9	22.0	31.0	24.6	675.4173	1387.2463	2062.6636	1564.2370	31.9
15	1000	14	38.4	52.4	47.6	3.7358	9.3589	13.0947	13.3696	-2.1
	2000	14	44.8	58.8	50.8	14.7246	43.1557	57.8803	56.4810	2.5
	3000	14	42.4	56.4	47.8	36.2643	92.7929	129.0572	120.5445	7.1
	4000	14	42.6	56.6	49.6	68.4397	168.3889	236.8286	223.0386	6.2
	5000	14	35.0	49.0	49.4	108.6200	220.9233	329.5433	348.3895	-5.4
	6000	14	46.2	60.2	53.8	157.7627	413.4616	571.2243	545.8562	4.6
	7000	14	41.6	55.6	52.4	218.2647	509.5913	727.8560	721.5752	0.9
	8000	14	46.8	60.8	48.4	288.8969	761.6478	1050.5447	873.4027	20.3
	9000	14	45.4	59.4	54.4	363.4264	944.0076	1307.4340	1230.0626	6.3
	10000	14	43.6	57.6	52.0	448.6884	1111.3986	1560.0870	1458.9940	6.9
	11000	14	41.2	55.2	54.4	554.9419	1279.0549	1833.9968	1869.0985	-1.9
	12000	14	41.8	55.8	45.6	662.0624	1559.6373	2221.6997	1856.8739	19.6
	13000	14	42.6	56.6	51.6	781.8030	1896.5801	2678.3831	2487.2534	7.7
	14000	14	48.8	62.8	61.6	922.2734	2617.4280	3539.7014	3267.6504	8.3
	15000	14	38.2	52.2	43.8	1070.0026	2369.3886	3439.3912	3009.9744	14.3

Tables 4.5 – 4.7 contains the results from Test II, which used average values from 5 problem samples instead of 10 problem samples as being used in Test I. Hence, the problems with the same size $m \times n$ in Test I may give different average results comparing with results from Test II. Therefore, we should compare results from within the same test configuration only.

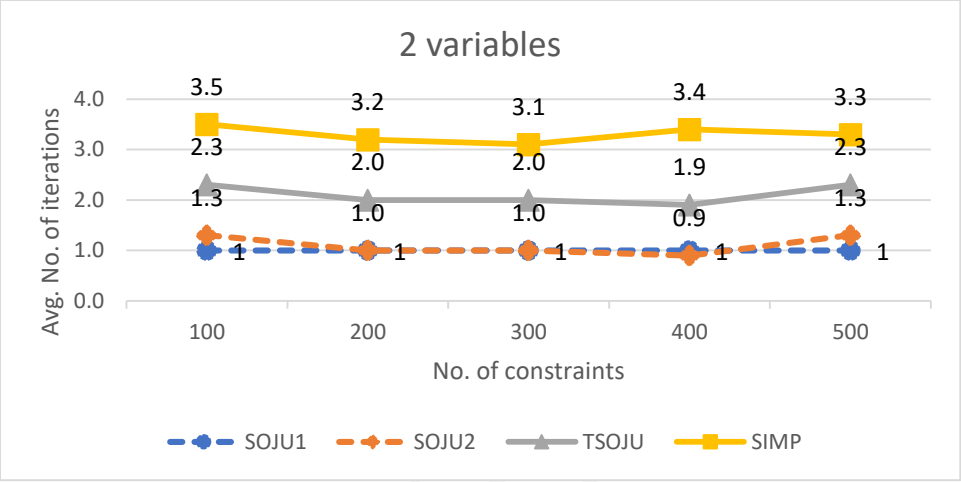
Table 4.5 shows that when the number of variables is 2 or 3 and the number of constraints ranges from 1000 to 15000, SOJU gave smaller number of iterations in most cases and, therefore, had better running time than SIMP in most cases. However, when the number of variables are 4 to 15 and the number of constraints ranges from 1000 to 15000, SOJU started to use more iterations than SIMP. Nonetheless, the running time difference percentages in those cases still ranges between -31% and 32%.

4.2.1 Average number of iterations (Test I)

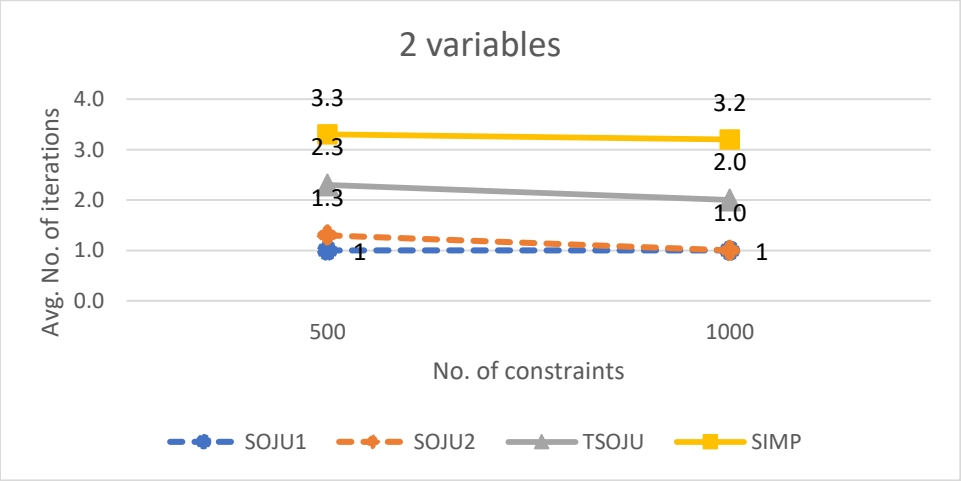
For two variable problem, average number of iterations solved by SOJU method was fewer than SIMP method in all sizes of constraints (see Figure 4.1). Furthermore, the 1st stage of SOJU method was solved using 1 iteration and the 2nd stage of SOJU was solved using about 1 iteration. In total, SOJU method had better performance than SIMP method for this case.



a) Small number of constraints



b) Medium number of constraints



c) Large number of constraints

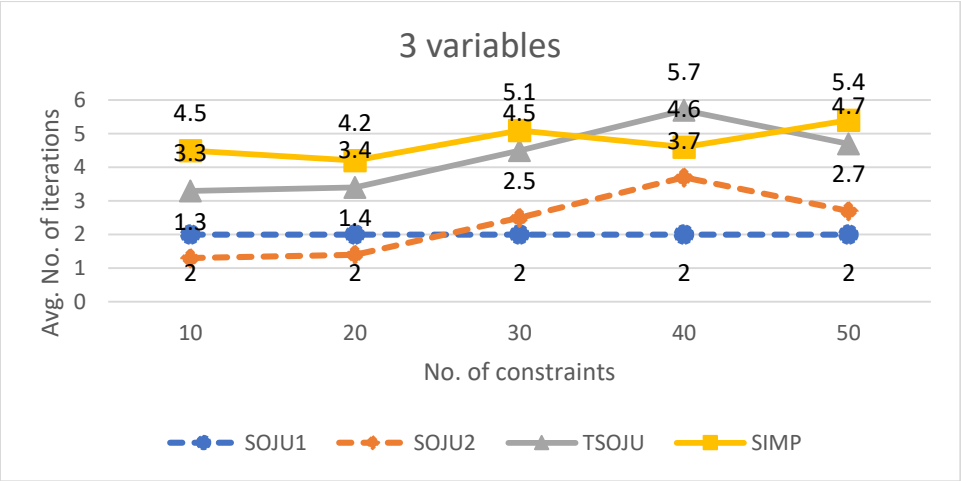
Figure 4.1: The average number of iterations by SIMP and SOJU with 2 variables in Test I.

For three variable problem, the average number of iterations solved by SOJU method was less than SIMP except for 40 constraints. (See Figure 4.2.) However, the average number of iterations in SOJU method was similar to SIMP method for 40 constraints.

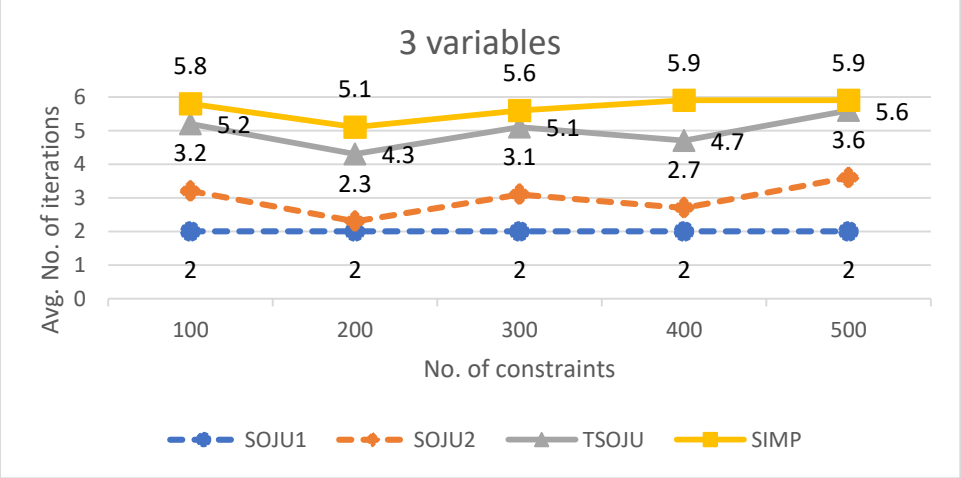
Figures 4.3 and 4.4, which are the results from problems with 3 and 4 variables, respectively, show comparable results where SOJU and SIMP take about the same number of iterations in all cases.

Figures 4.5 – 4.9, which are the results from problems with 10 - 50 variables, show that the second stage SOJU takes fewer number of iterations than SIMP. However, the number of iterations in the first stage SOJU grows with the number of variables (i.e. $n-1$ iterations for a problem with n variables). In total, SOJU still takes more number of iterations than SIMP.

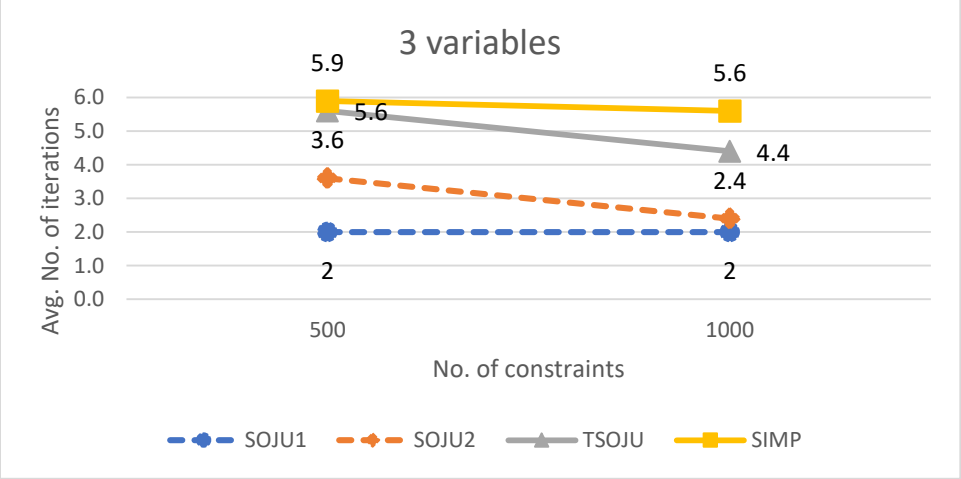
Figures 4.10 – 4.15, which are the results from problems with 100 - 1000 variables, show that the second stage SOJU takes about the same number of iterations as SIMP. Hence, SOJU still takes more number of iterations in total.



a) Small number of constraints

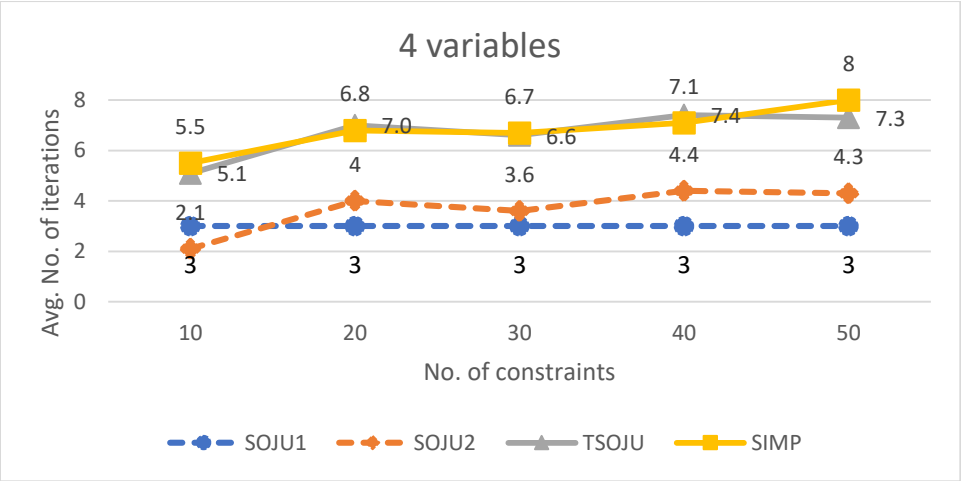


b) Medium number of constraints

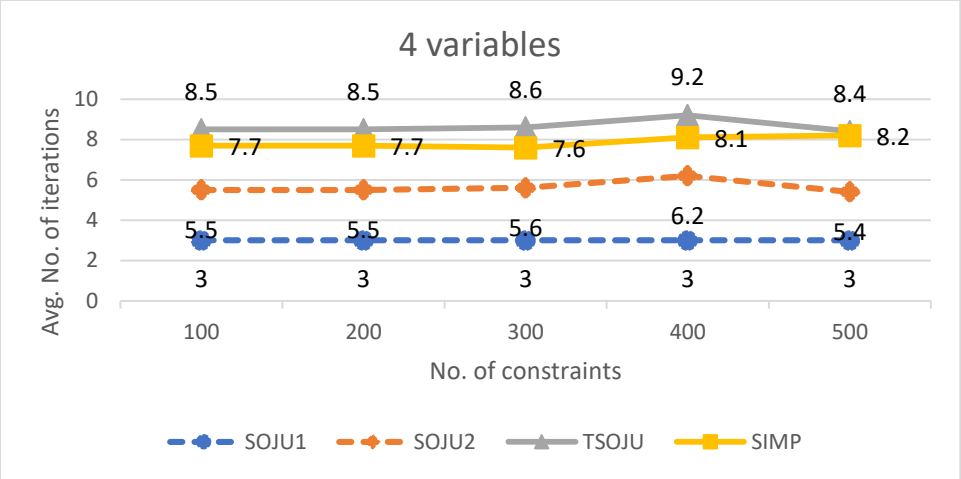


c) Large number of constraints

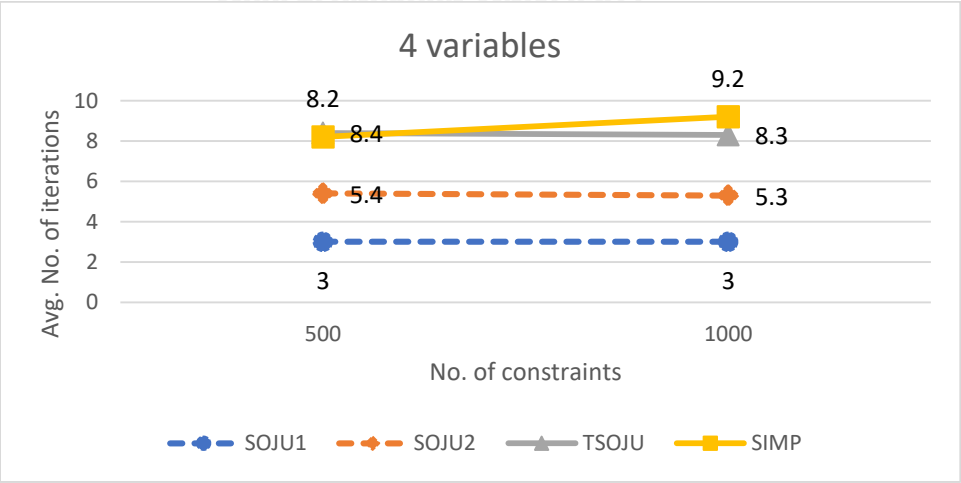
Figure 4.2: The average number of iterations by SIMP and SOJU with 3 variables in Test I.



a) Small number of constraints

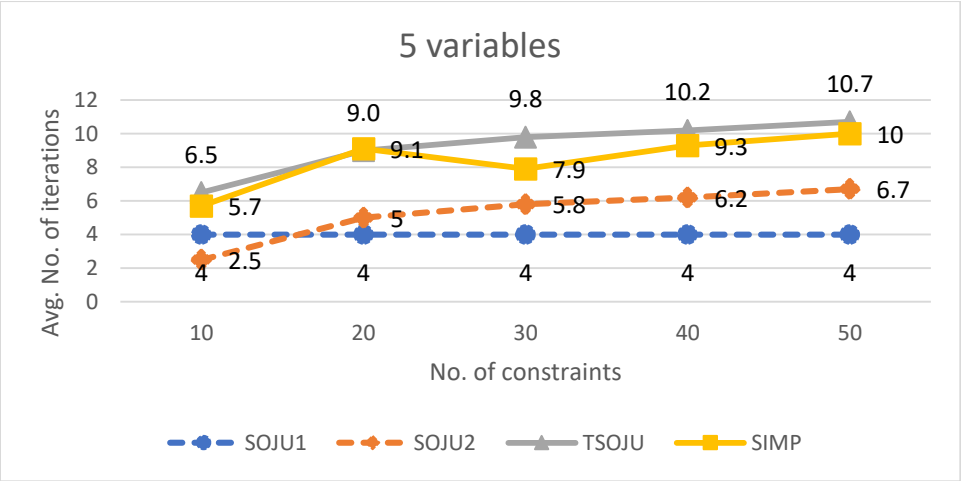


b) Medium number of constraints

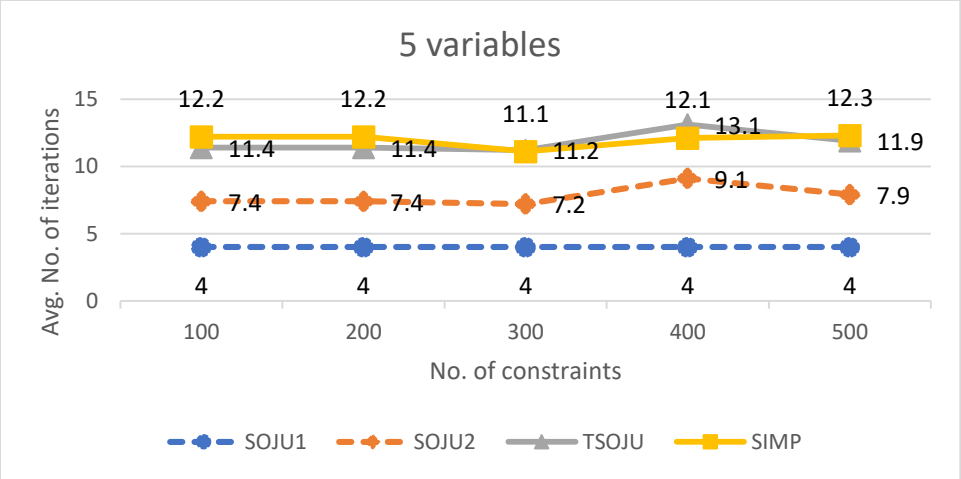


c) Large number of constraints

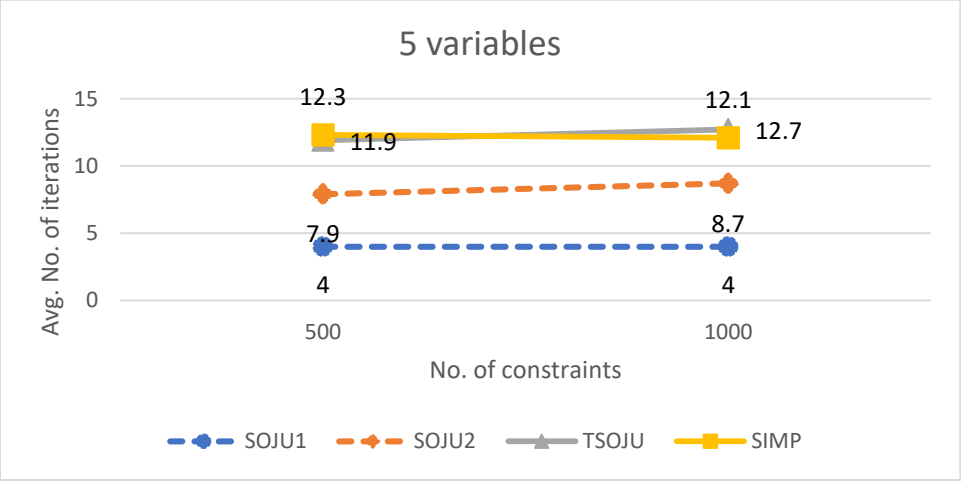
Figure 4.3: The average number of iterations by SIMP and SOJU with 4 variables in Test I.



a) Small number of constraints

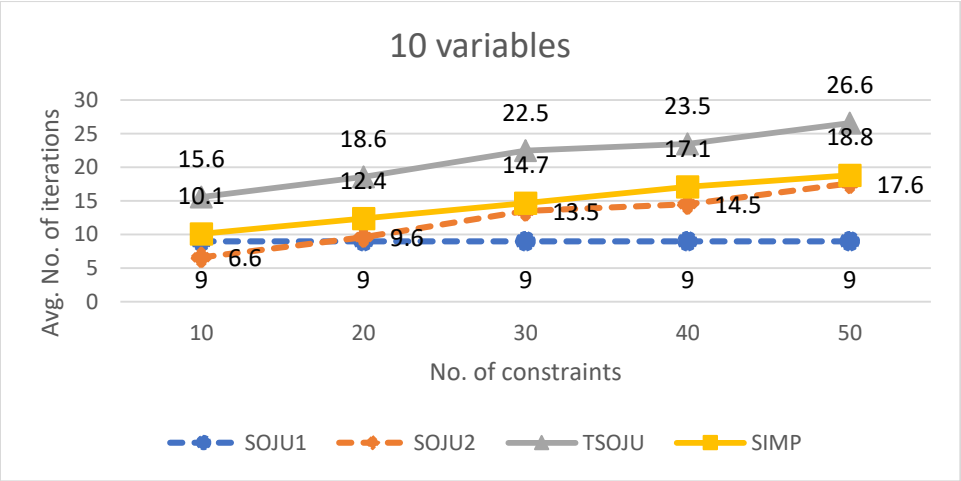


b) Medium number of constraints

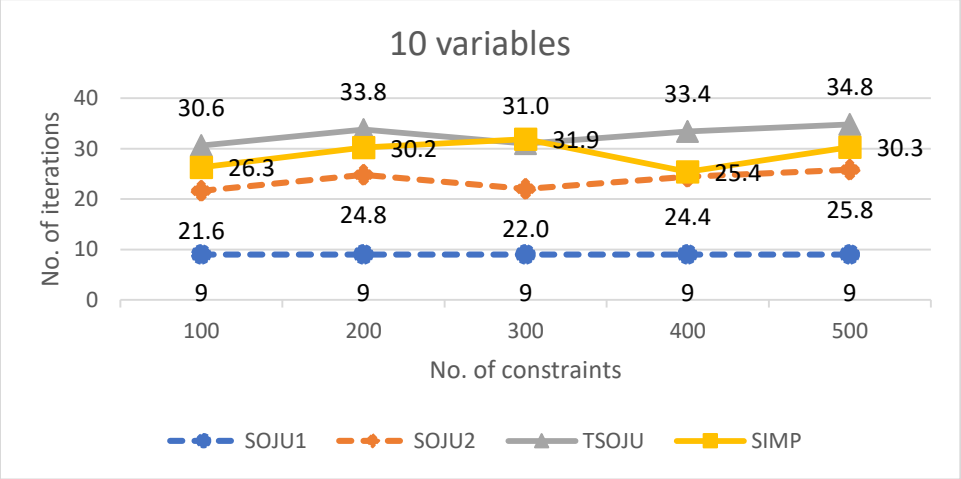


c) Large number of constraints

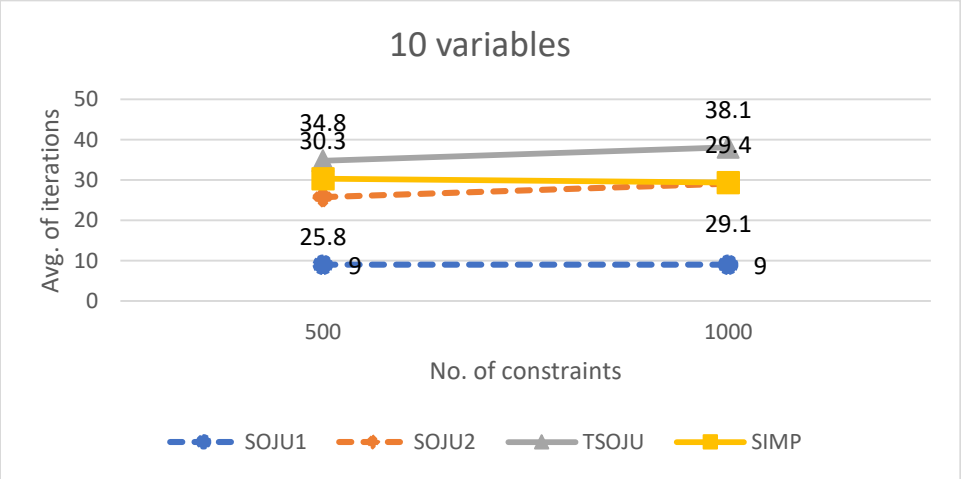
Figure 4.4: The average number of iterations by SIMP and SOJU with 5 variables of Test I.



a) Small number of constraints

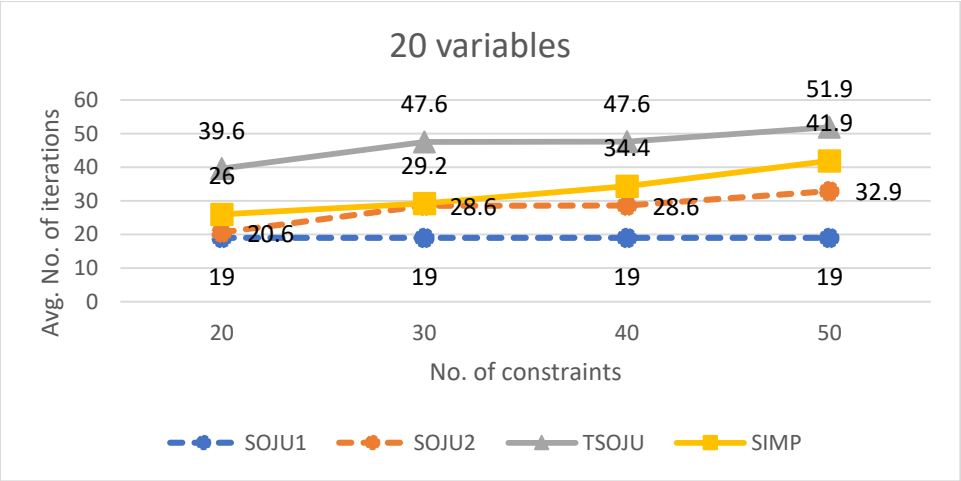


b) Medium number of constraints

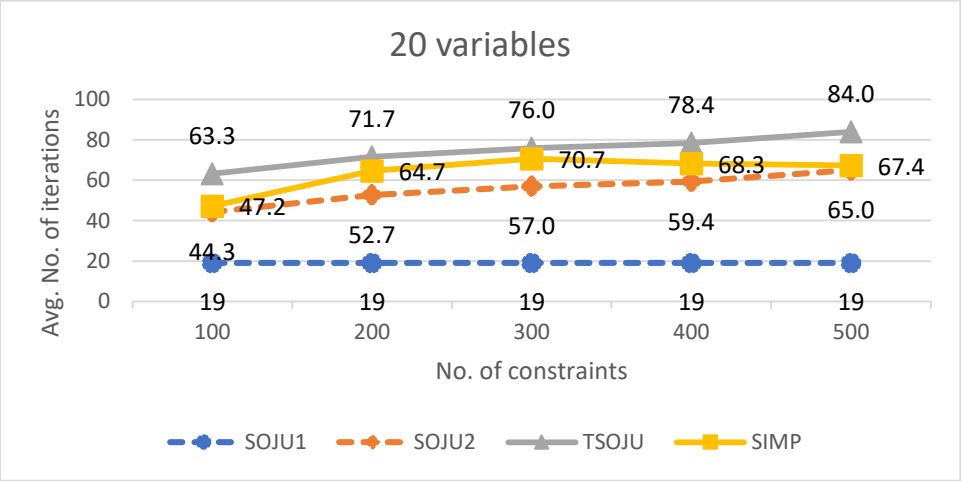


c) Large number of constraints

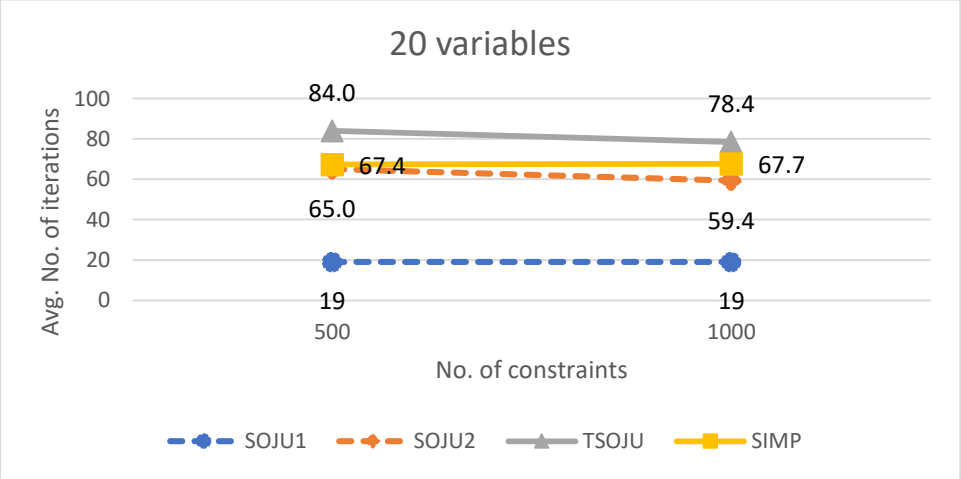
Figure 4.5: The average number of iterations by SIMP and SOJU with 10 variables in Test I.



a) Small number of constraints

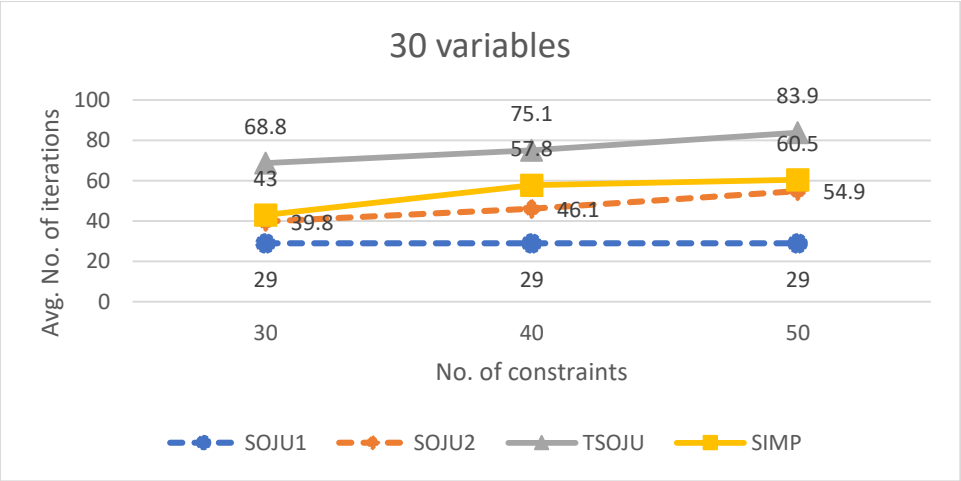


b) Medium number of constraints

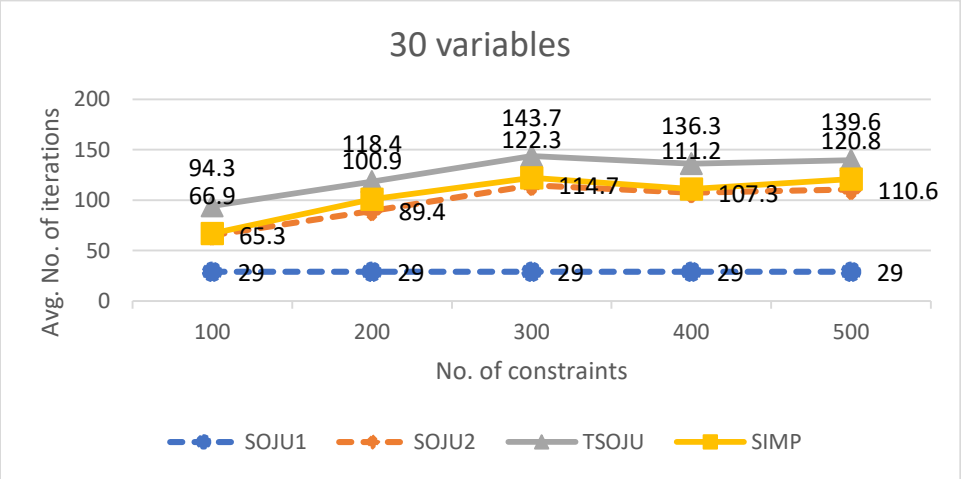


c) Large number of constraints

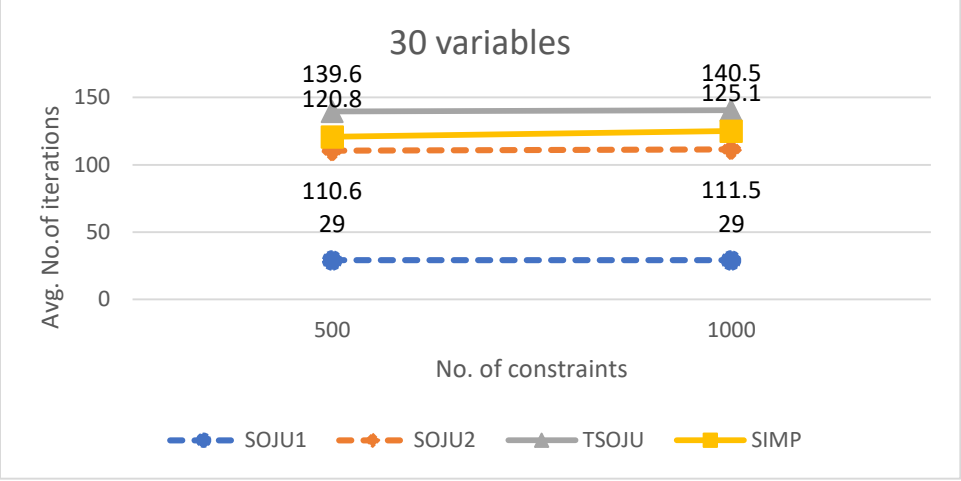
Figure 4.6: The average number of iterations by SIMP and SOJU with 20 variables in Test I.



a) Small number of constraints

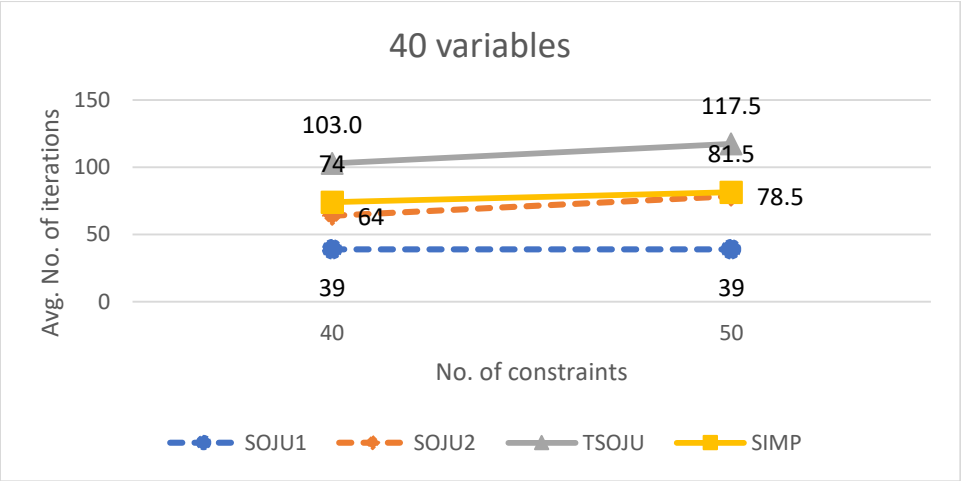


b) Medium number of constraints

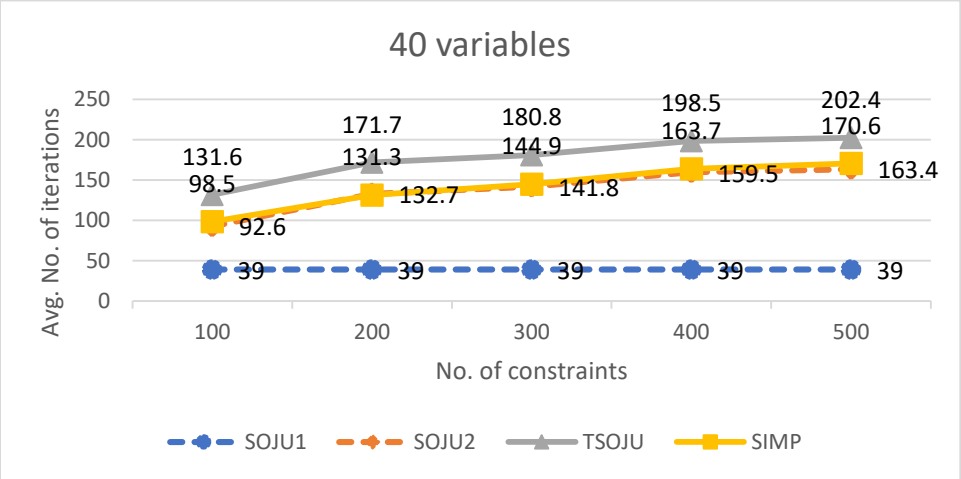


c) Large number of constraints

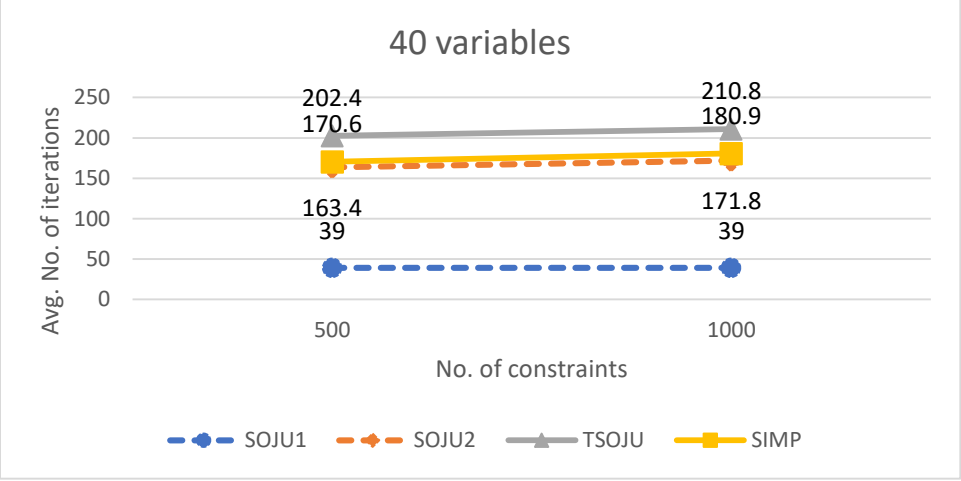
Figure 4.7: The average number of iterations by SIMP and SOJU with 30 variables of Test I.



a) Small number of constraints

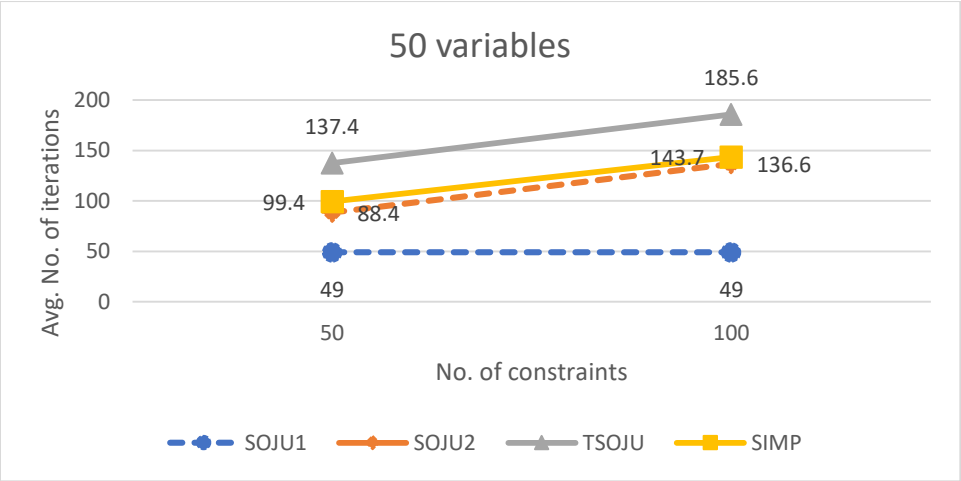


b) Medium number of constraints

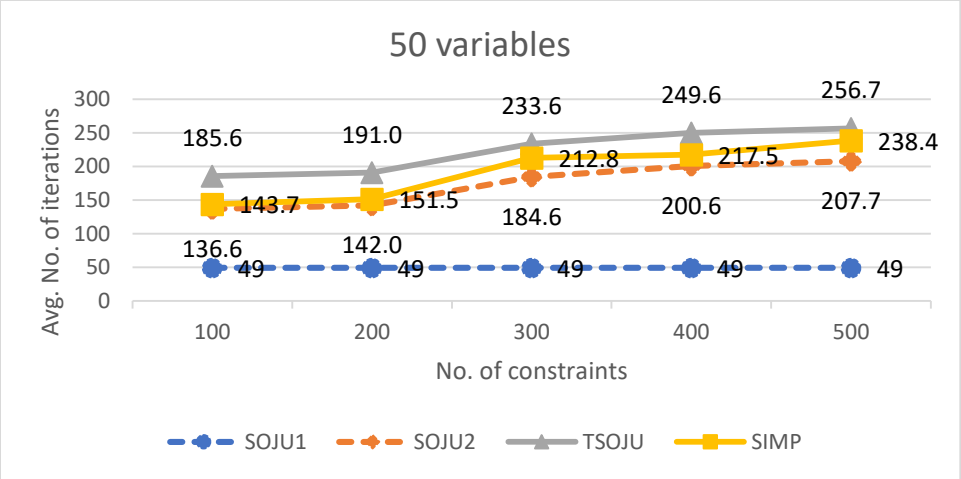


c) Large number of constraints

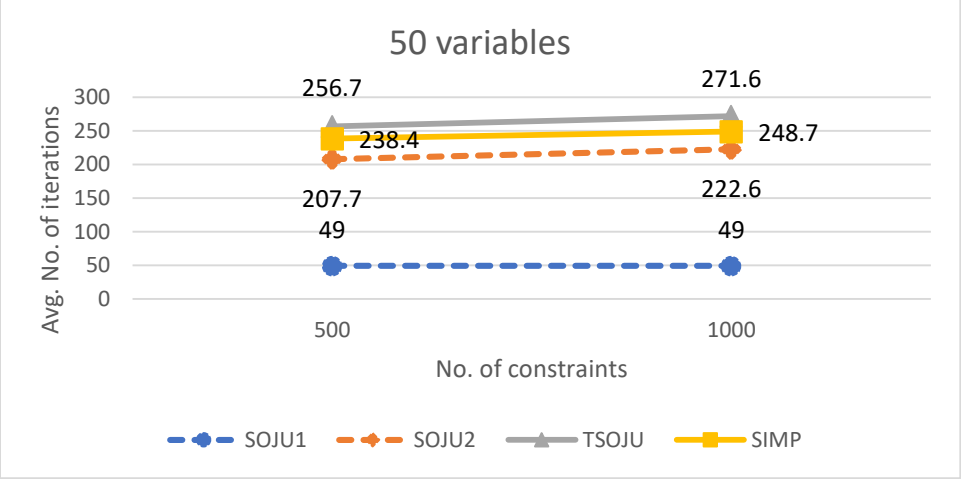
Figure 4.8: The average number of iterations by SIMP and SOJU with 40 variables in Test I.



a) Small number of constraints

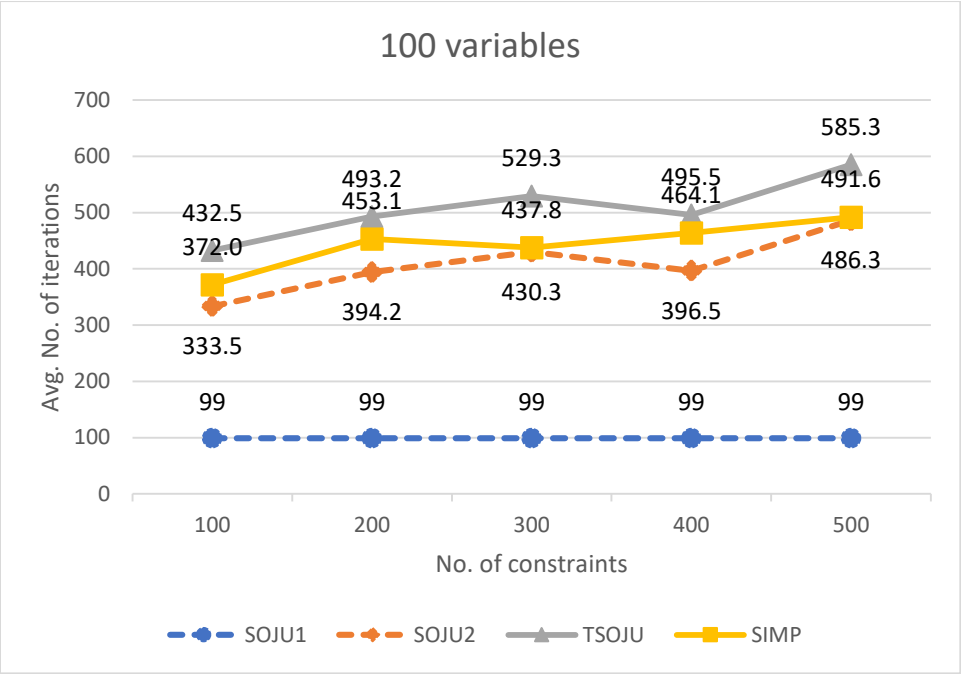


b) Medium number of constraints

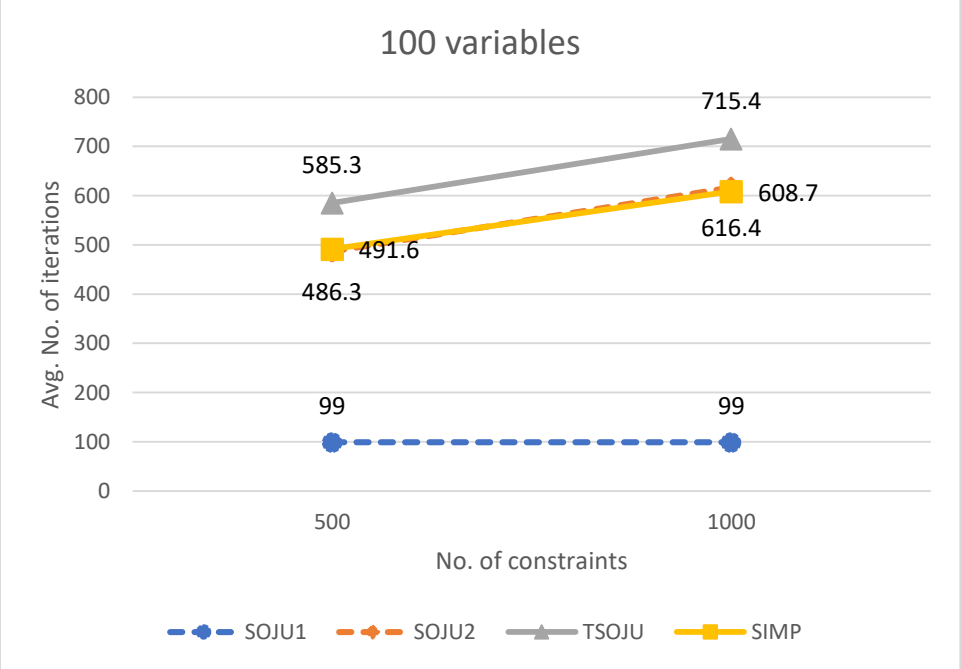


c) Large number of constraints

Figure 4.9: The average number of iterations by SIMP and SOJU with 50 variables in Test I.

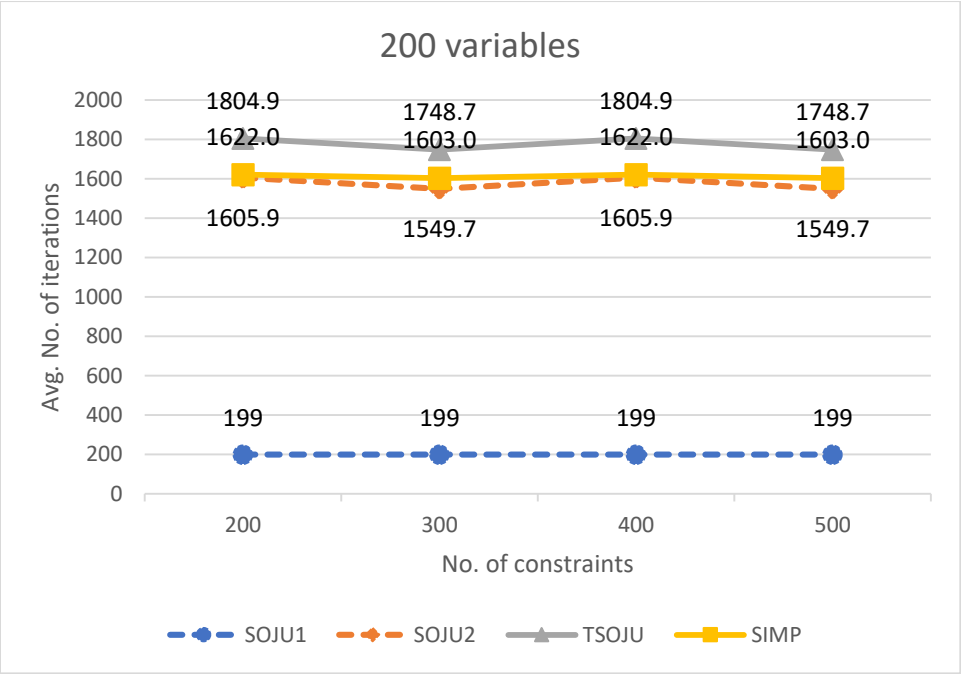


a) Medium number of constraints

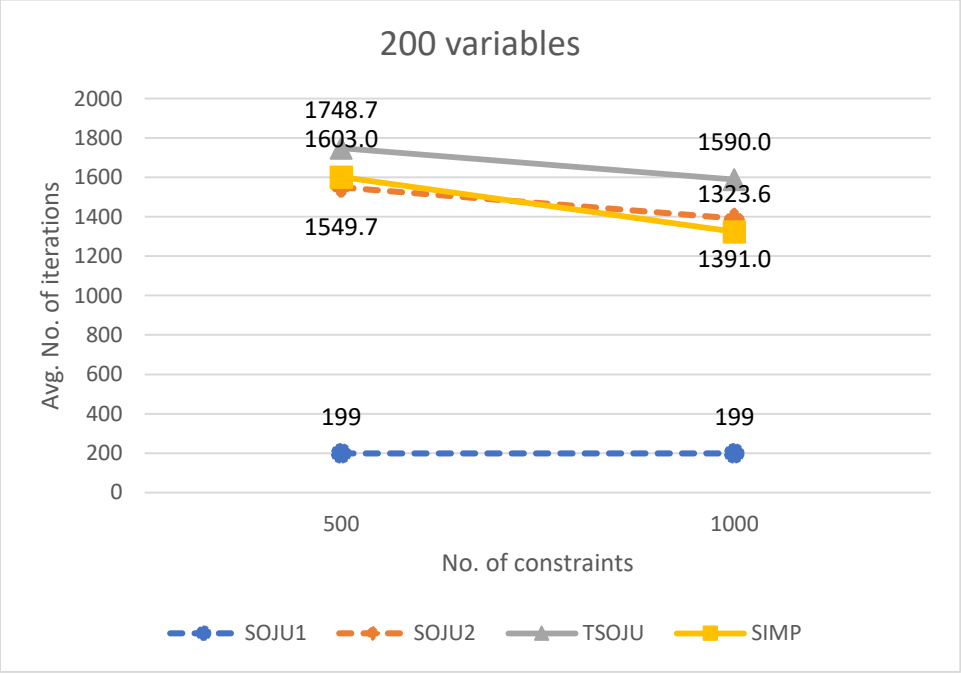


b) Large number of constraints

Figure 4.10: The average number of iterations by SIMP and SOJU with 100 variables in Test I.

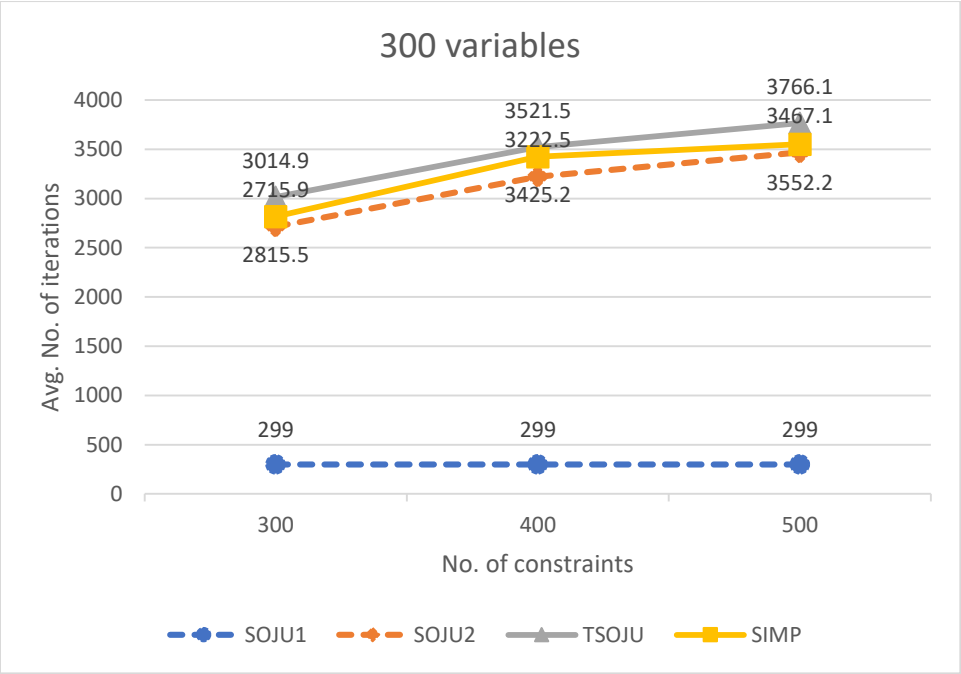


a) Medium number of constraints

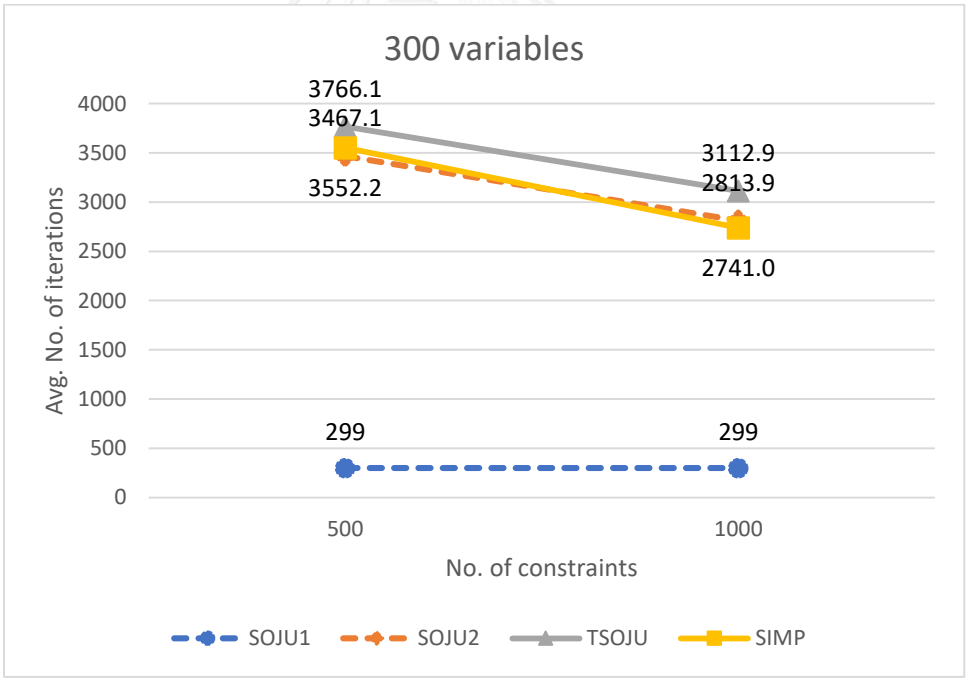


b) Large number of constraints

Figure 4.11: The average number of iterations by SIMP and SOJU with 200 variables in Test I.

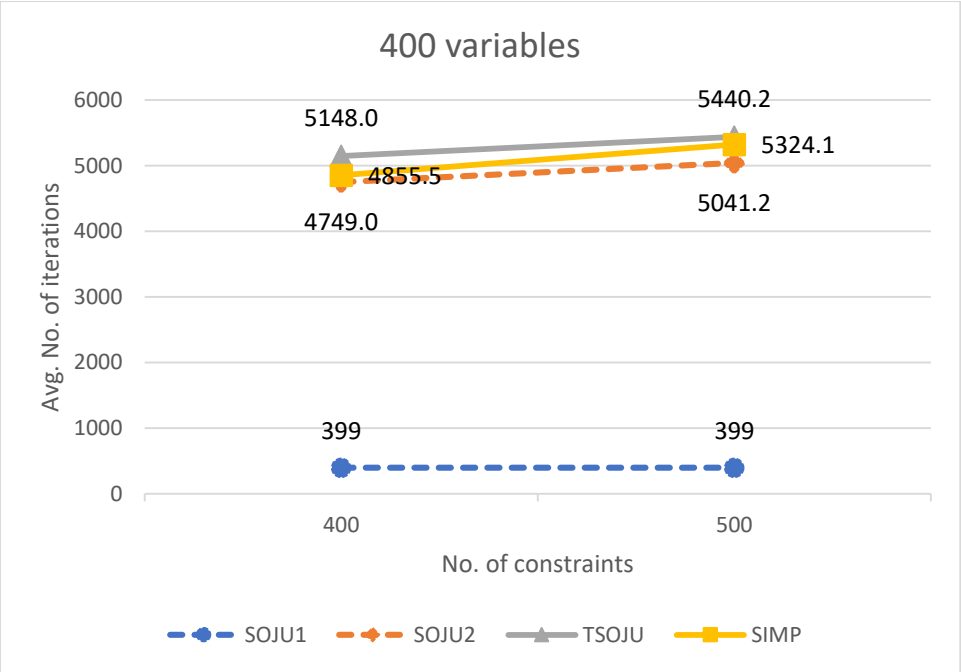


a) Medium number of constraints

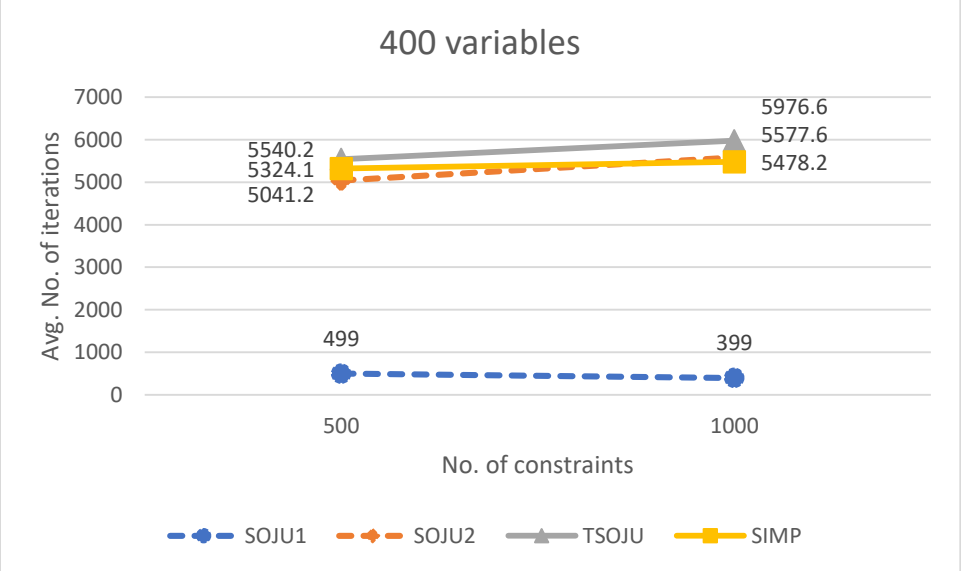


b) Large number of constraints

Figure 4.12 The average number of iterations by SIMP and SOJU with 300 variables in Test I.



b) Medium number of constraints



c) Large number of constraints

Figure 4.13: The average number of iterations by SIMP and SOJU with400 variables in Test I.

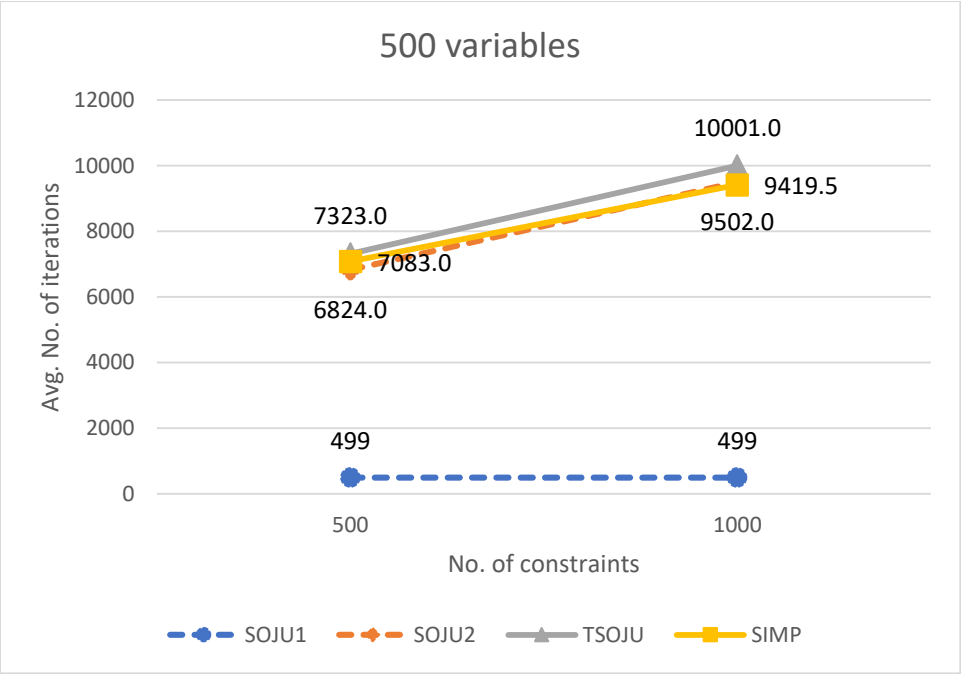


Figure 4.14: The average number of iterations by SIMP and SOJU with 500 variables in Test I.

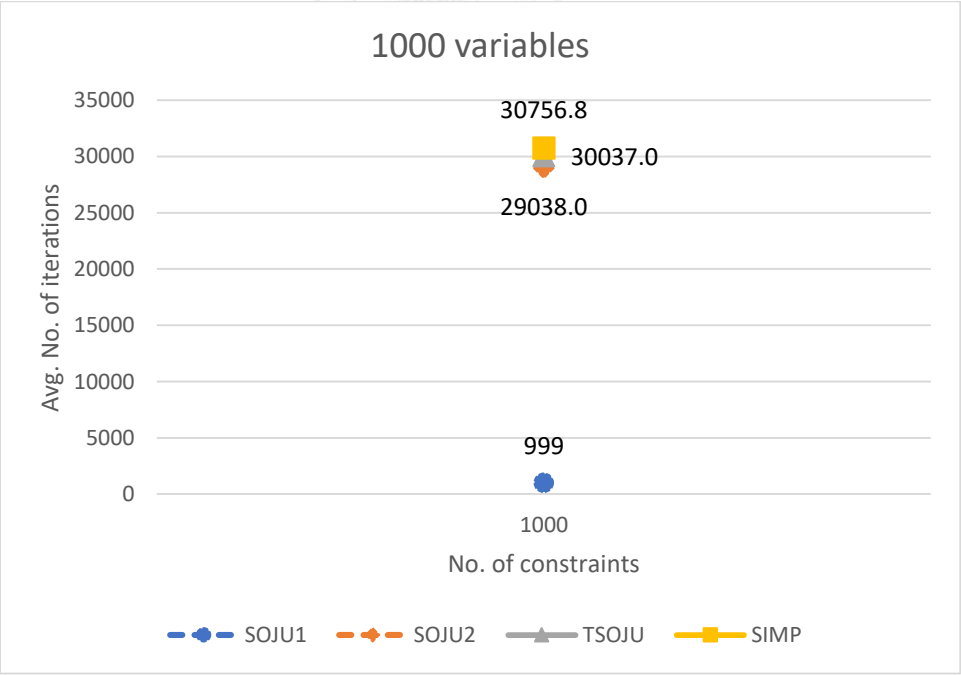
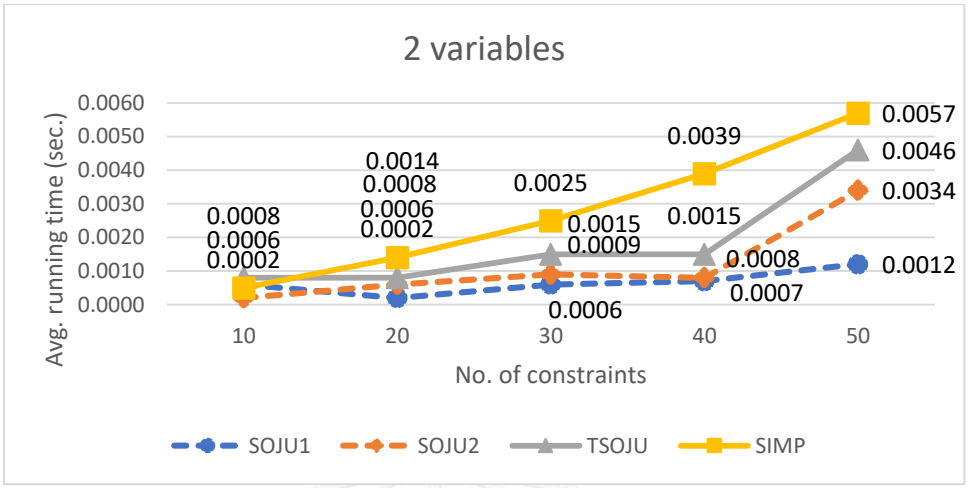


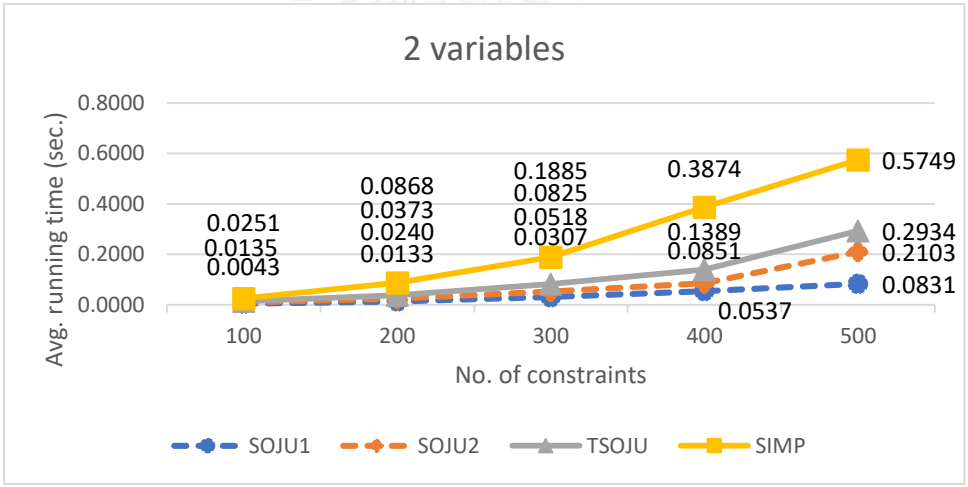
Figure 4.15: The average number of iterations by SIMP and SOJU with 1000 variables in Test I.

4.2.2 Average running time (Test I)

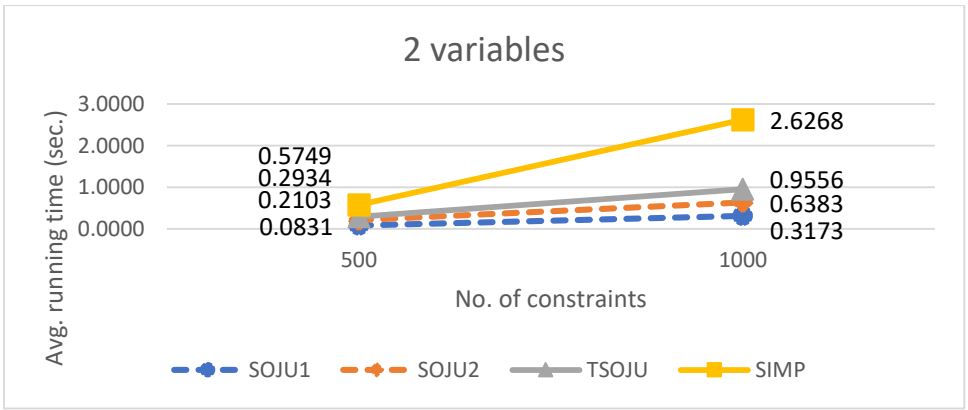
For problems with 2 – 5 variables, SOJU performed better than SIMP in most cases in terms of running time. (See Figures 4.16 – 4.19.)



a) Small number of constraints

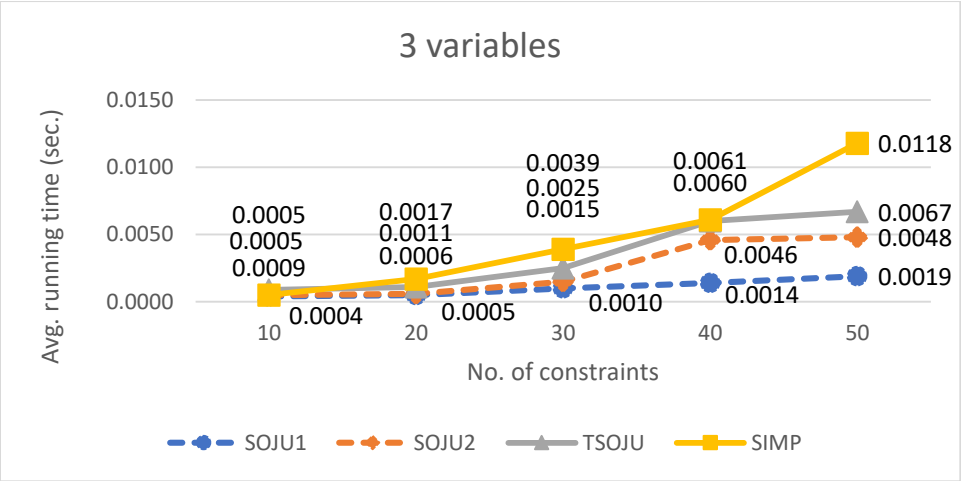


b) Medium number of constraints

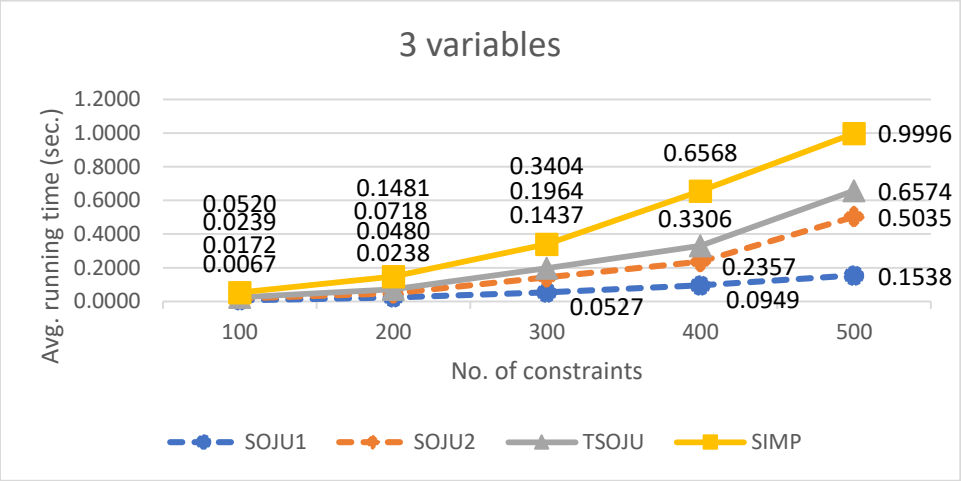


c) Large number of constraints

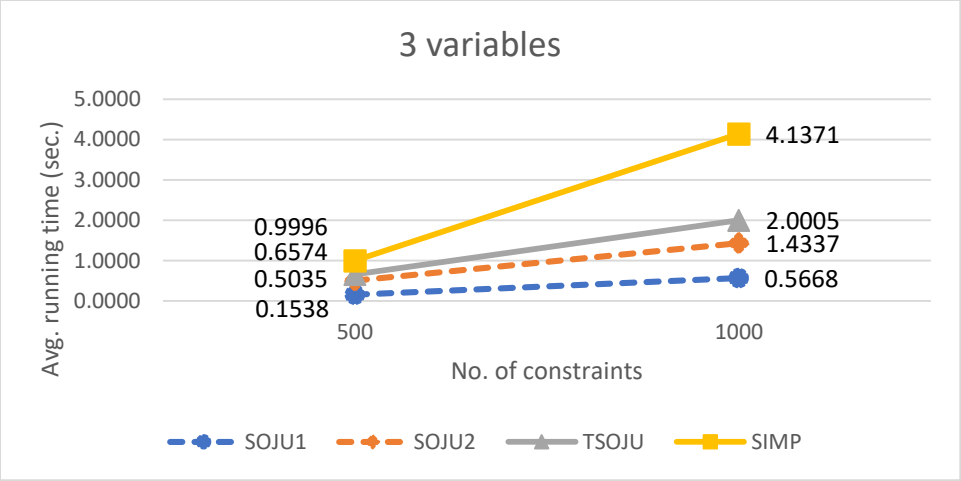
Figure 4.16: The average number of iterations by SIMP and SOJU with 2 variables in Test I.



a) Small number of constraints

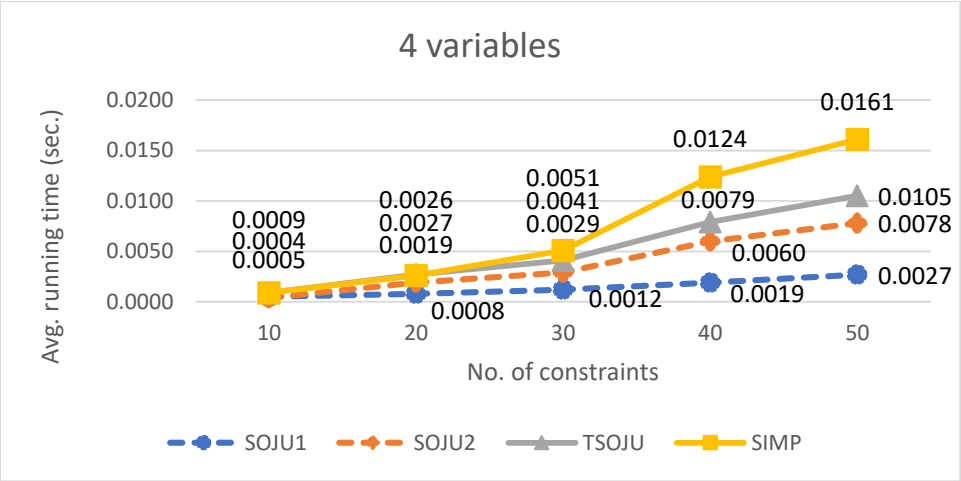


b) Medium number of constraints

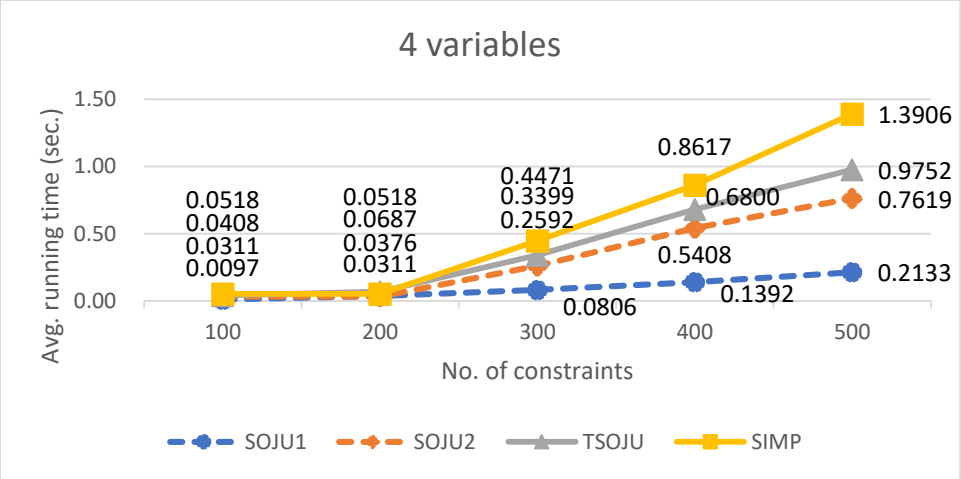


c) Large number of constraints

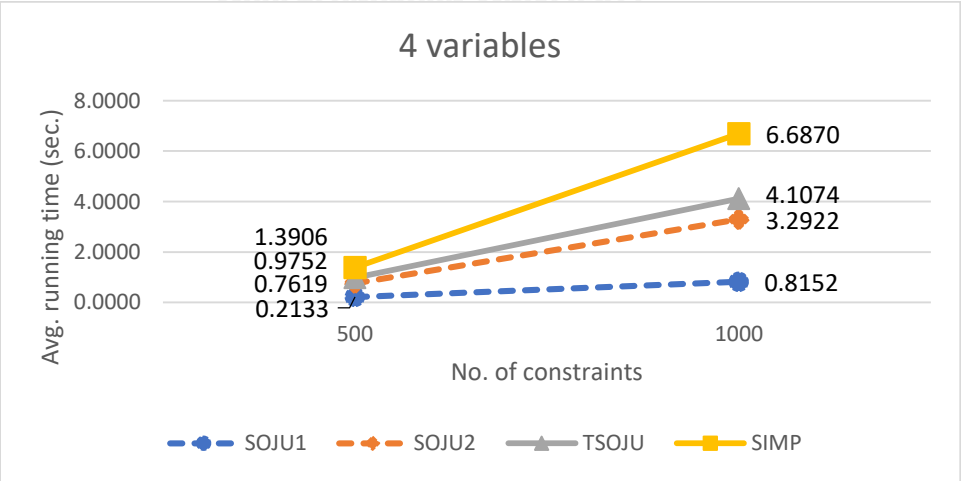
Figure 4.17: The average number of iterations by SIMP and SOJU with 3 variables in Test I.



a) Small number of constraints

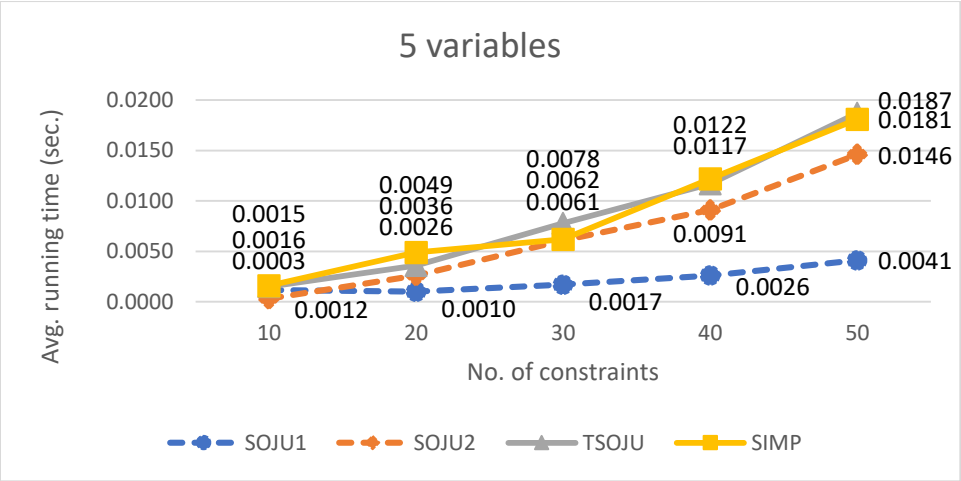


b) Medium number of constraints

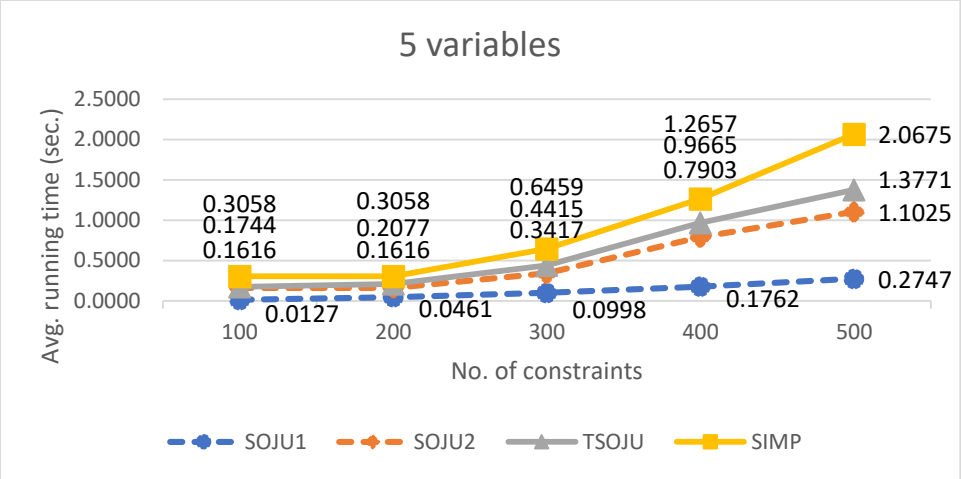


c) Large number of constraints

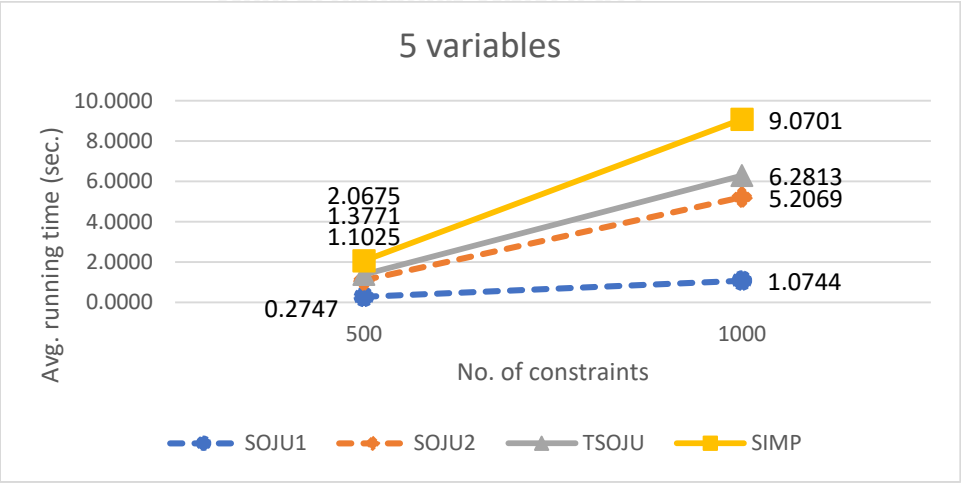
Figure 4.18: The average number of iterations by SIMP and SOJU with 4 variables in Test I.



a) Small number of constraints



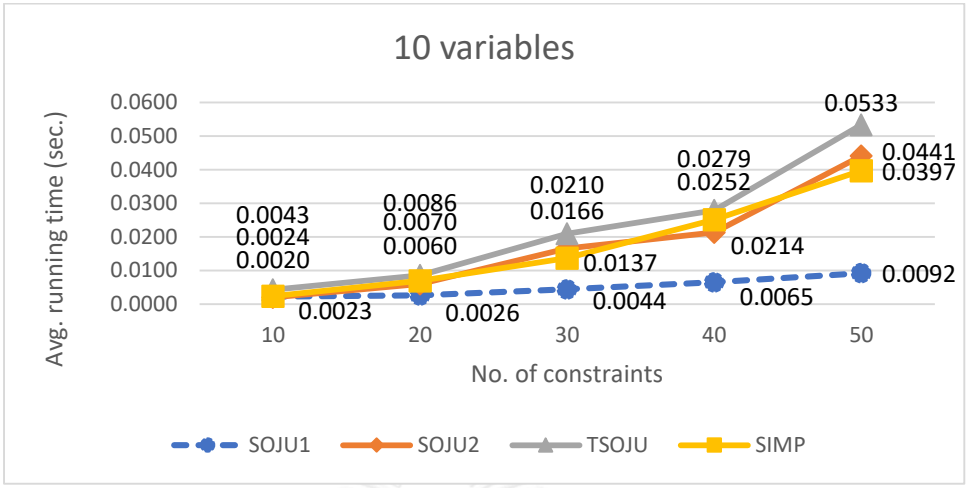
b) Medium number of constraints



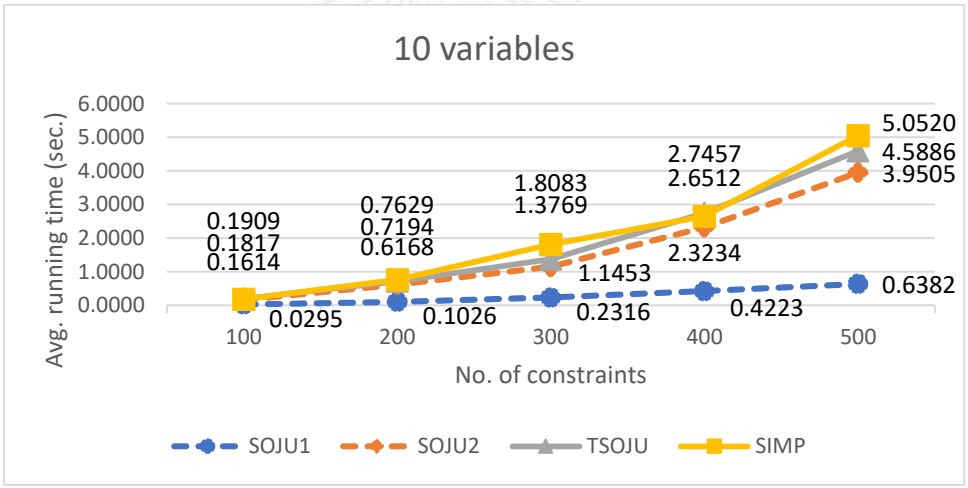
c) Large number of constraints

Figure 4.19: The average number of iterations by SIMP and SOJU with 5 variables in Test I.

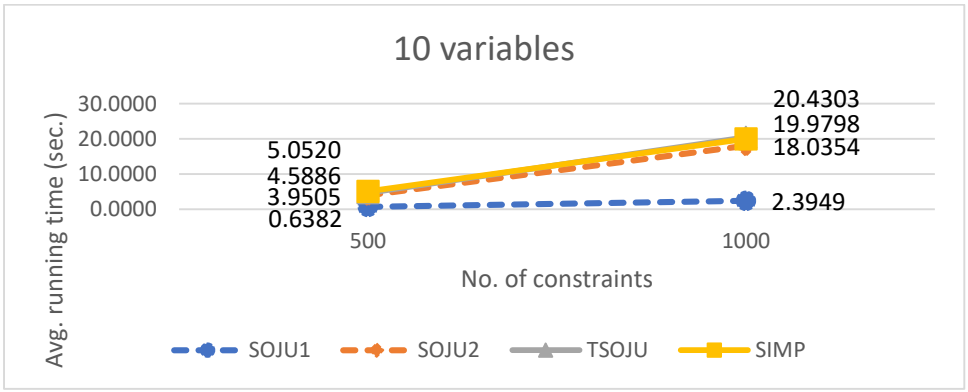
For problems with 10 – 1000 variables, SOJU and SIMP had comparable performance in most cases in terms of running time. (See Figures 4.20 – 4.30.)



a) Small number of constraints

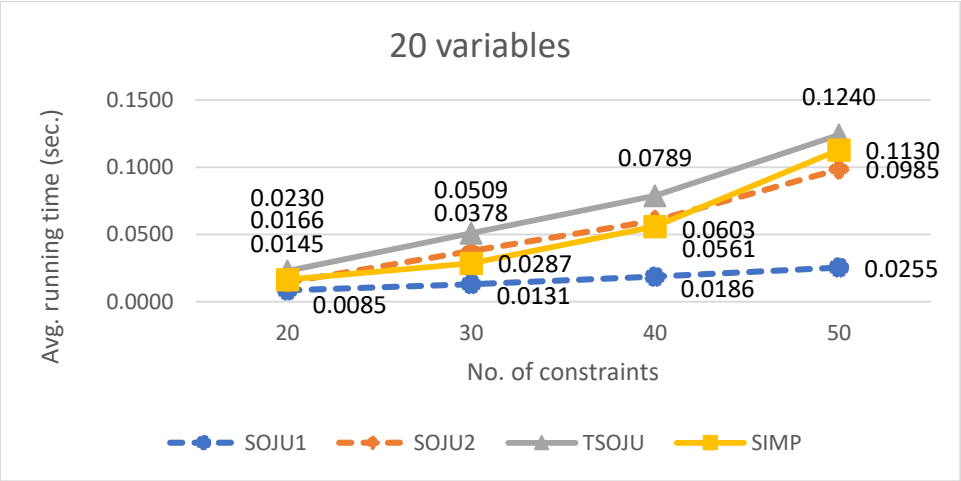


b) Medium number of constraints

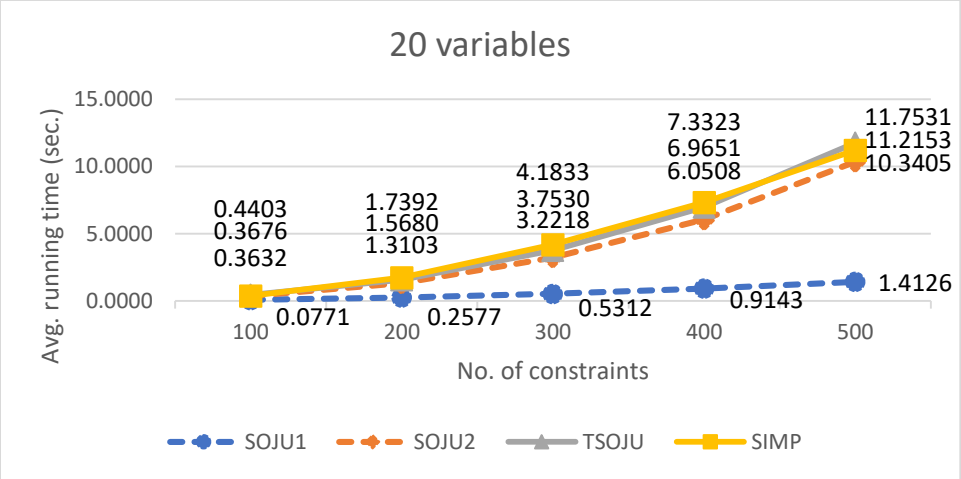


c) Large number of constraints

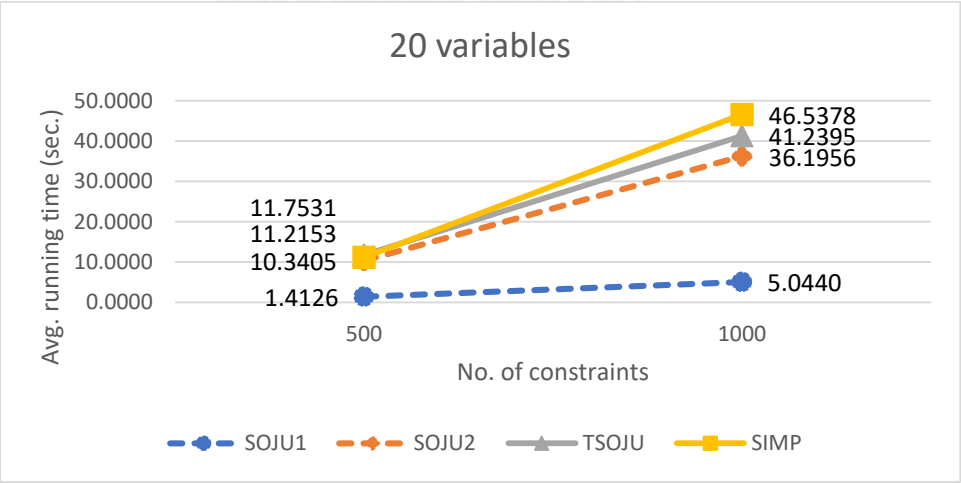
Figure 4.20: The average number of iterations by SIMP and SOJU with 10 variables in Test I.



a) Small number of constraints

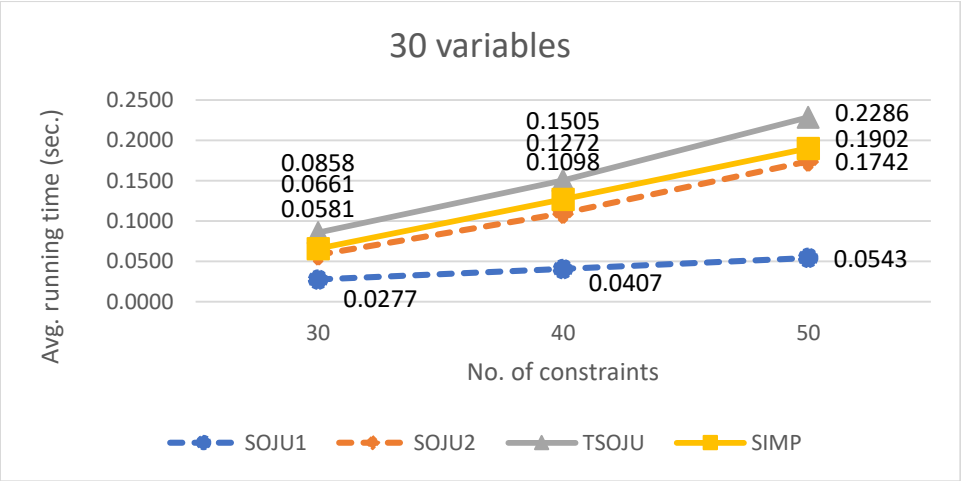


b) Medium number of constraints

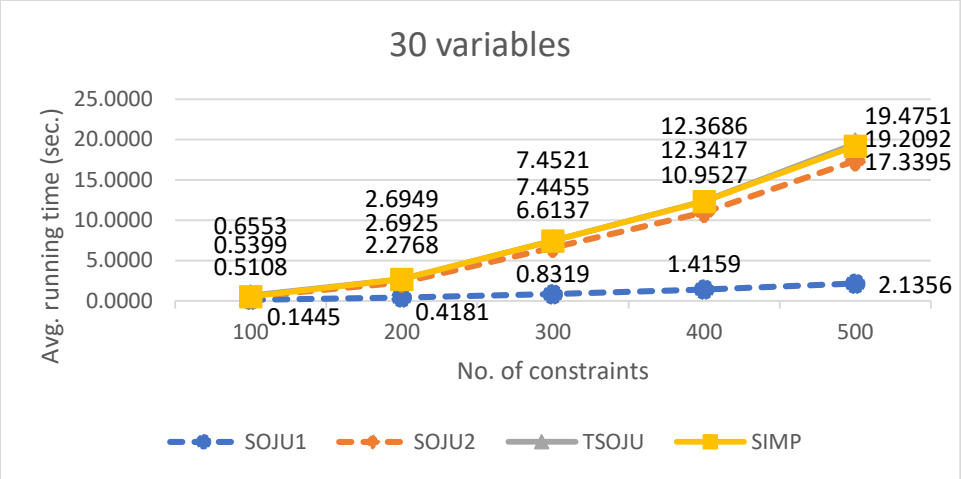


c) Large number of constraints

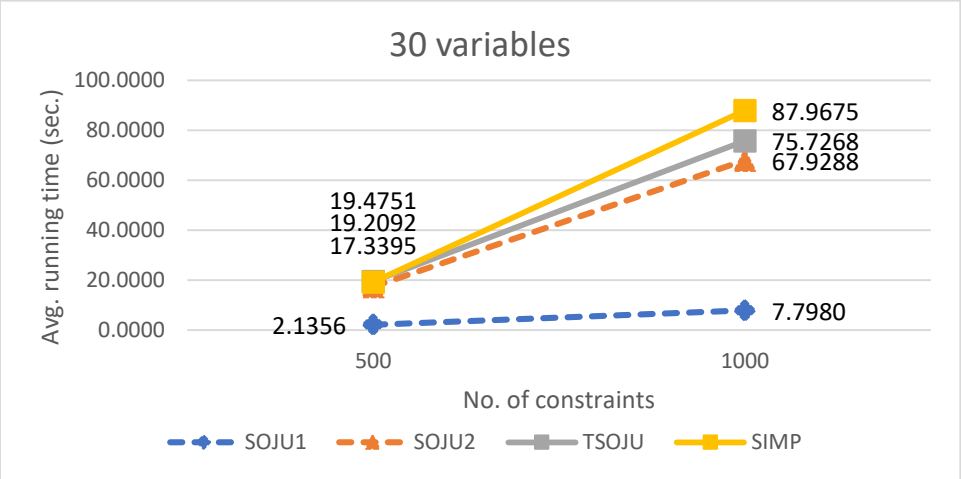
Figure 4.21: The average number of iterations by SIMP and SOJU with 20 variables of Test I.



a) Small number of constraints

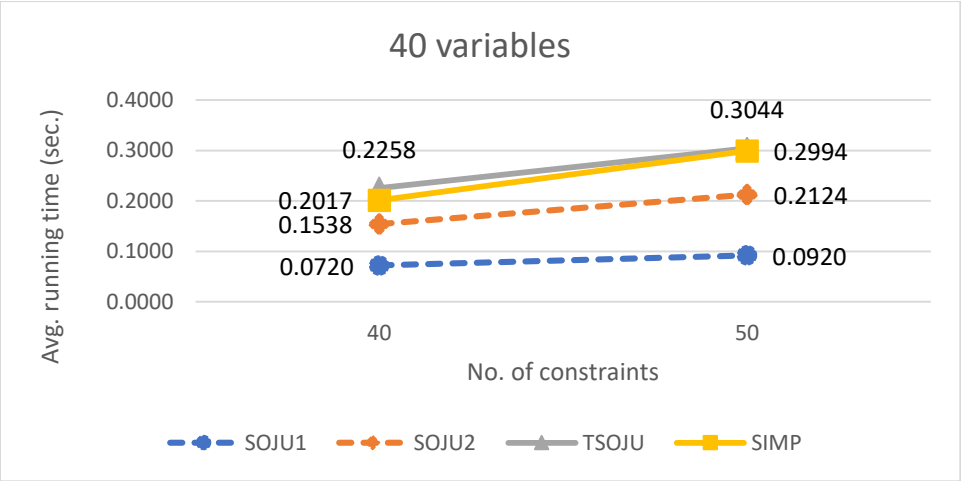


b) Medium number of constraints

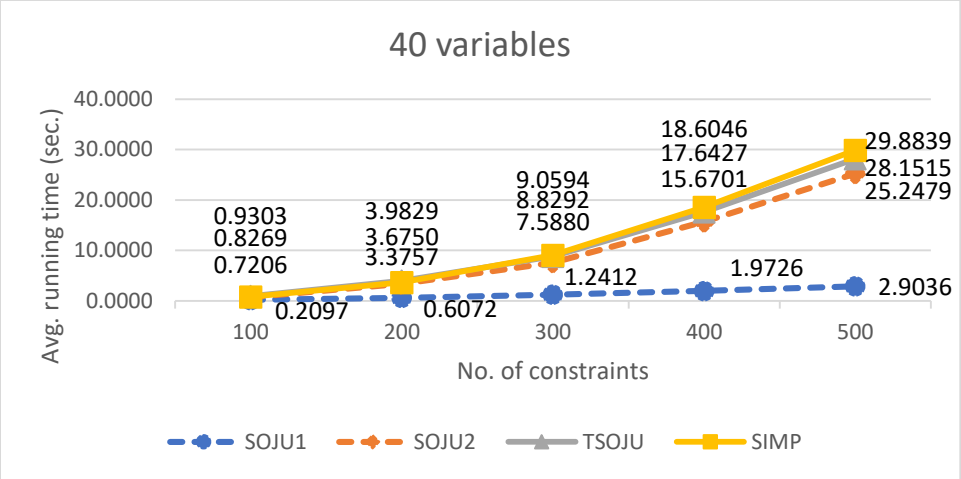


c) Large number of constraints

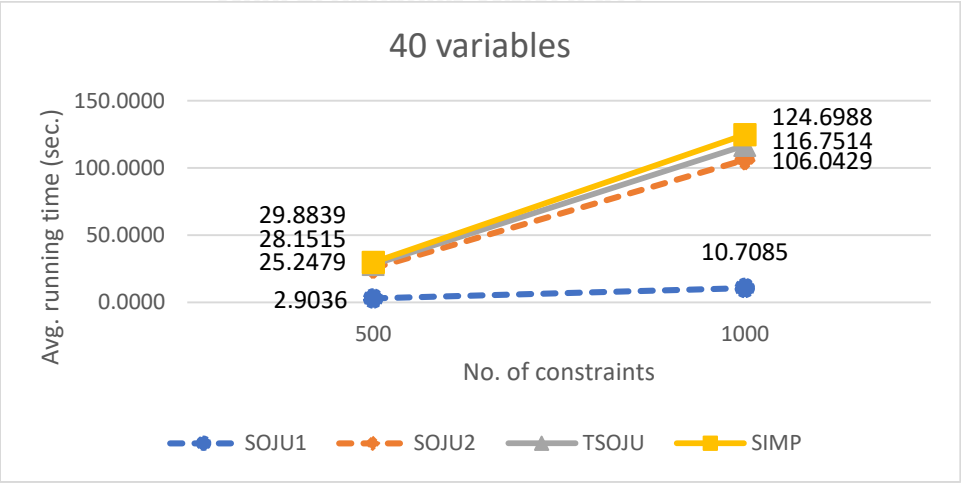
Figure 4.22: The average number of iterations by SIMP and SOJU with 30 variables in Test I.



a) Small number of constraints

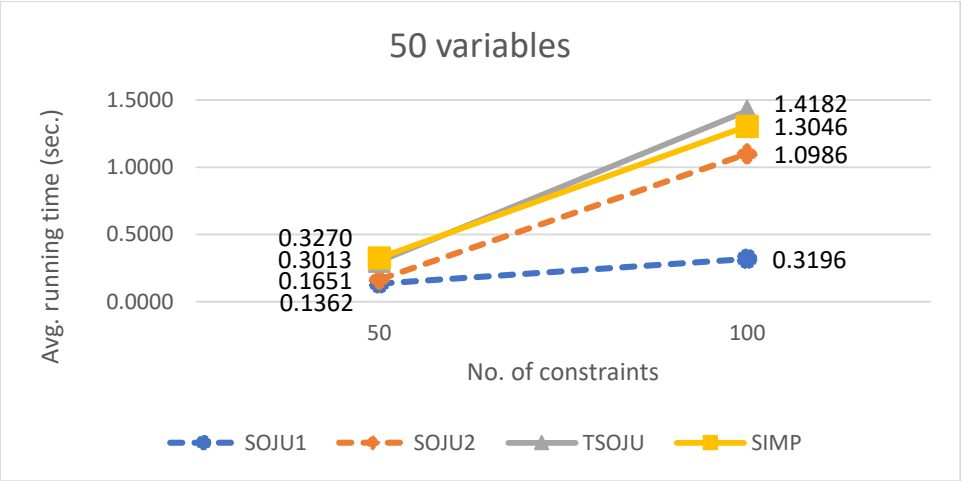


b) Medium number of constraints

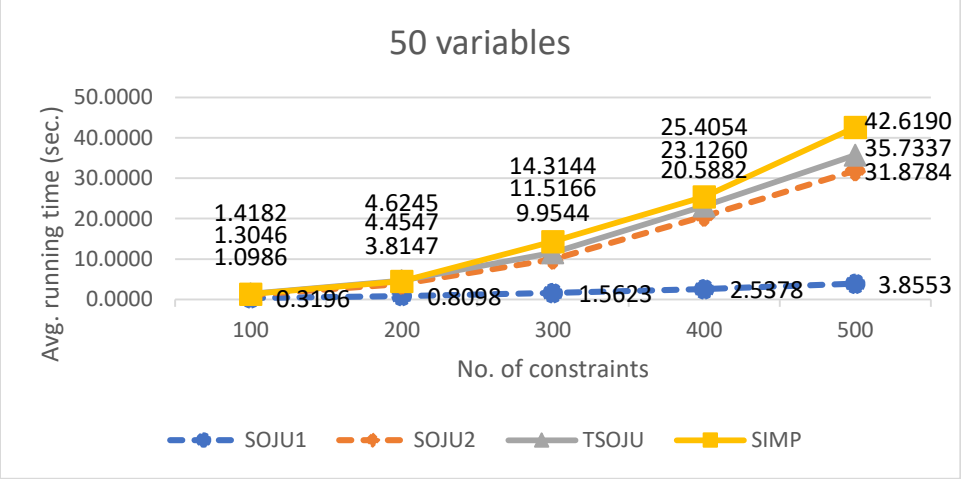


c) Large number of constraints

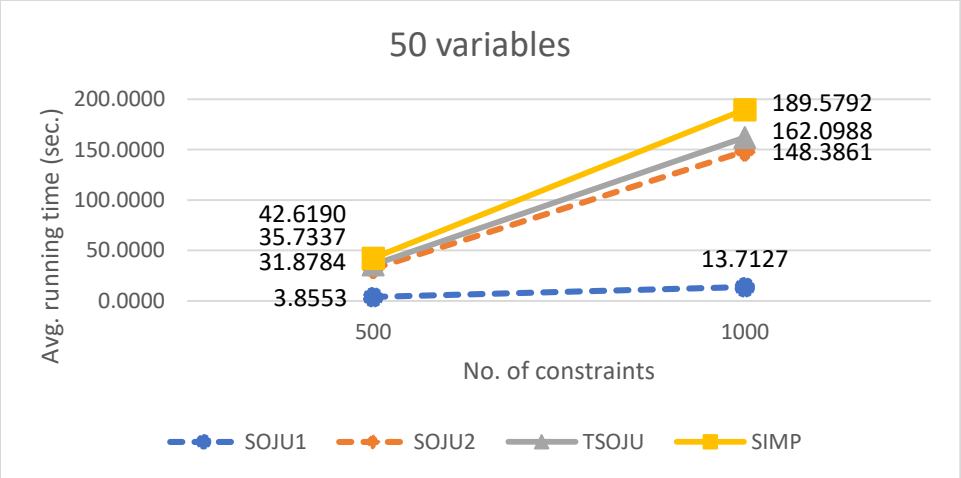
Figure 4.23: The average number of iterations by SIMP and SOJU with 40 variables in Test I.



a) Small number of constraints

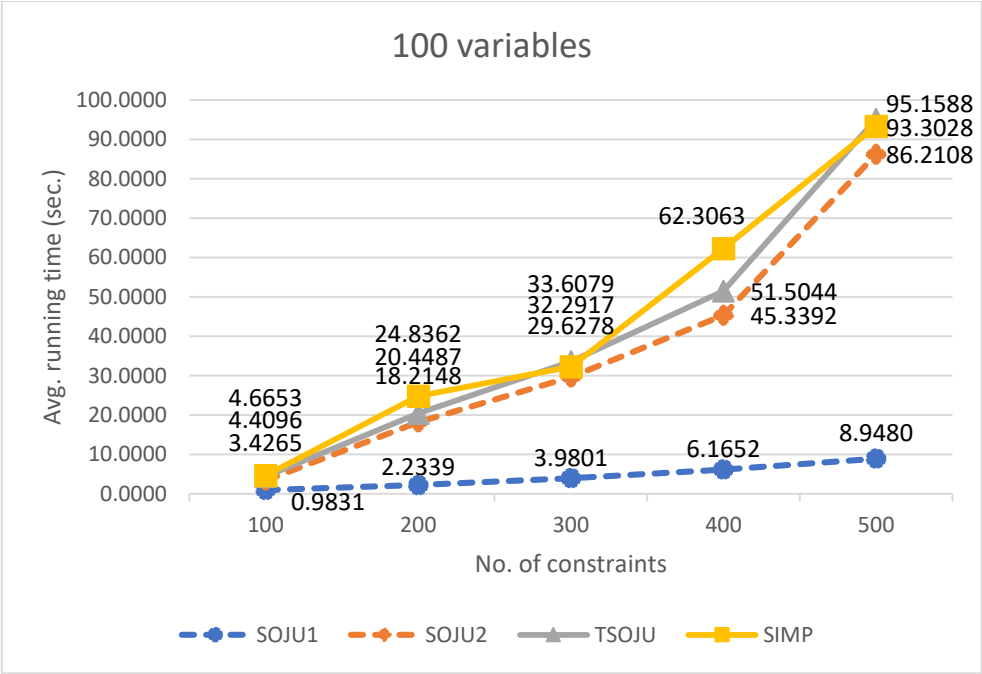


b) Medium number of constraints

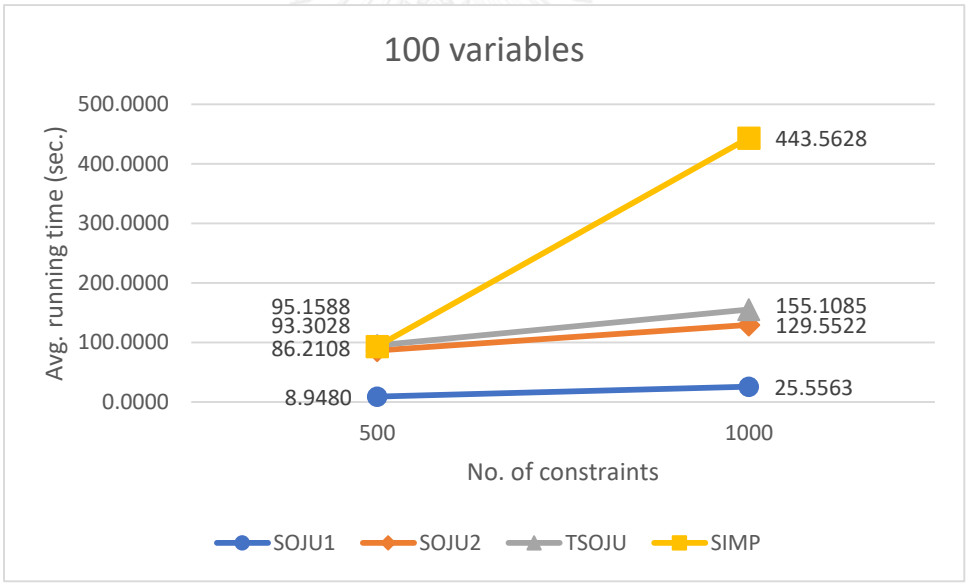


c) Large number of constraints

Figure 4.24: The average number of iterations by SIMP and SOJU with 50 variables in Test I.

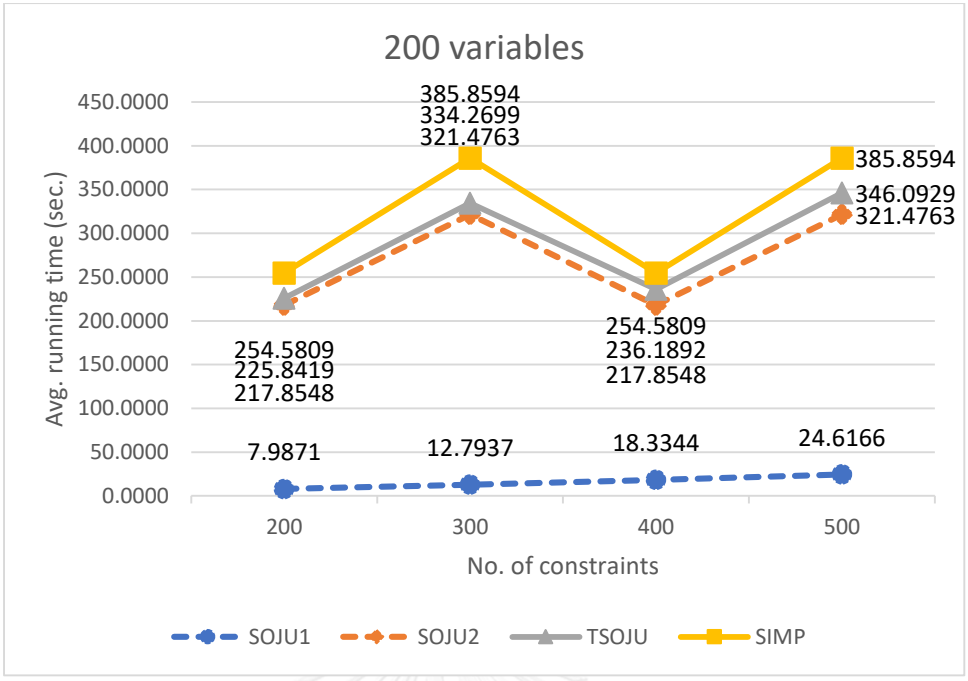


a) Medium number of constraints

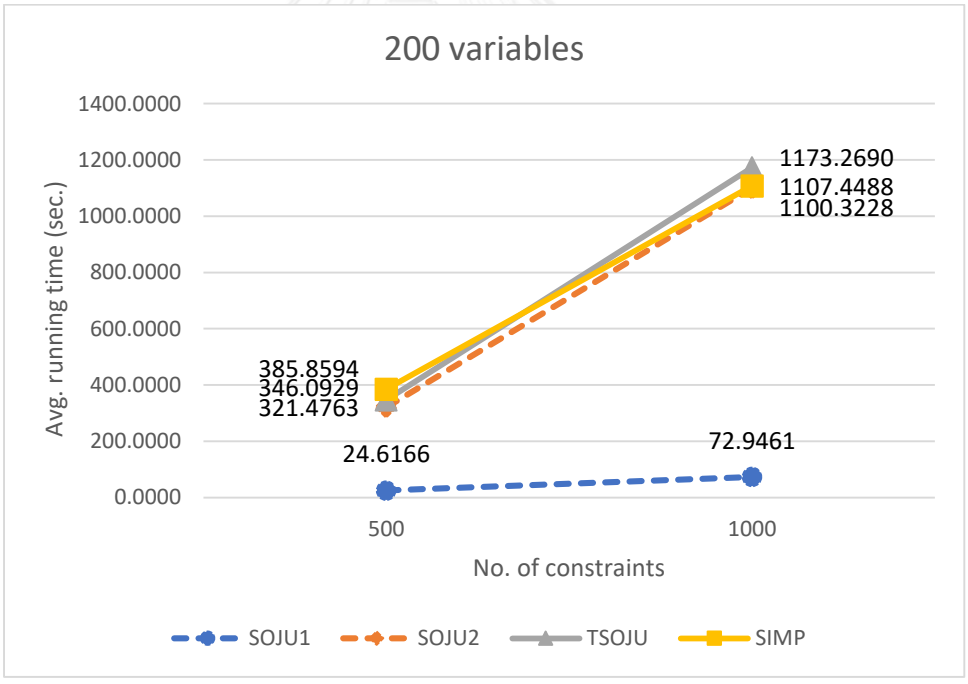


b) Large number of constraints

Figure 4.25: The average number of iterations by SIMP and SOJU with 100 variables in Test I.

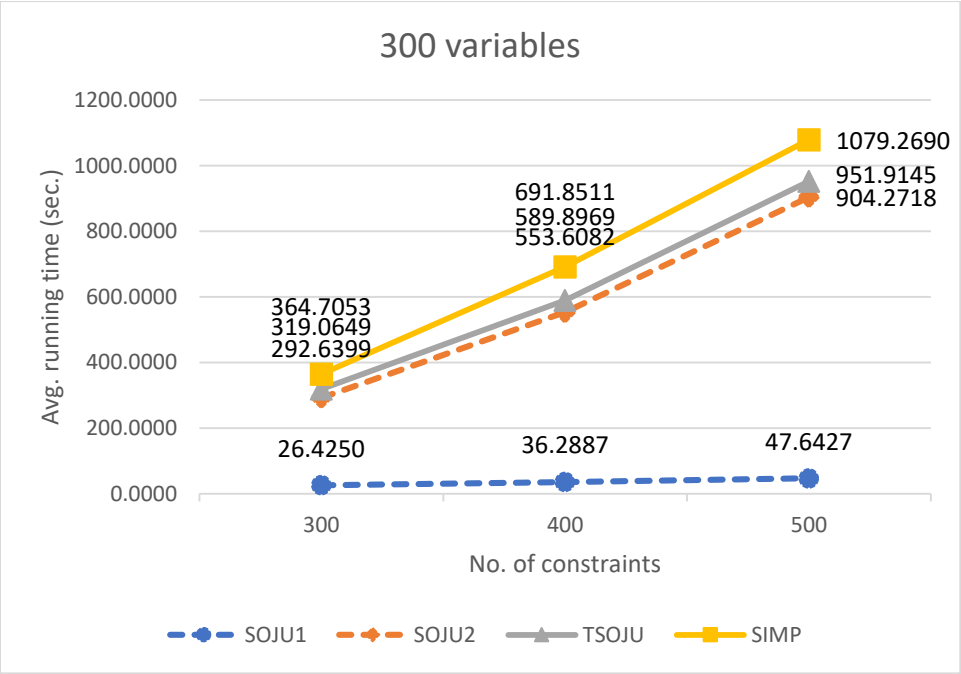


a) Medium number of constraints

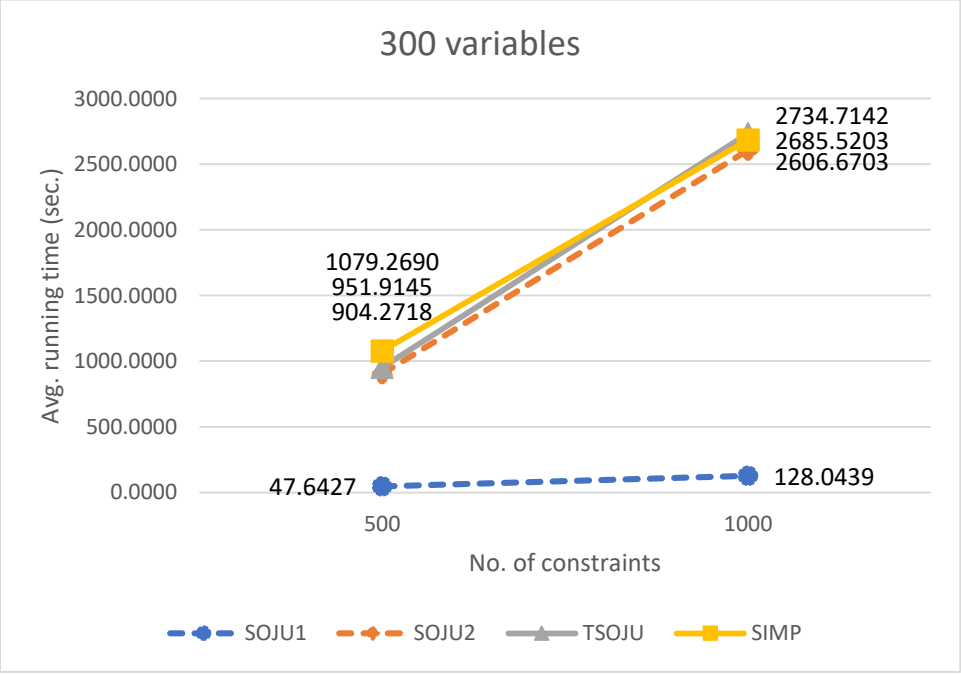


b) Large number of constraints

Figure 4.26: The average number of iterations by SIMP and SOJU with 200 variables of Test I.

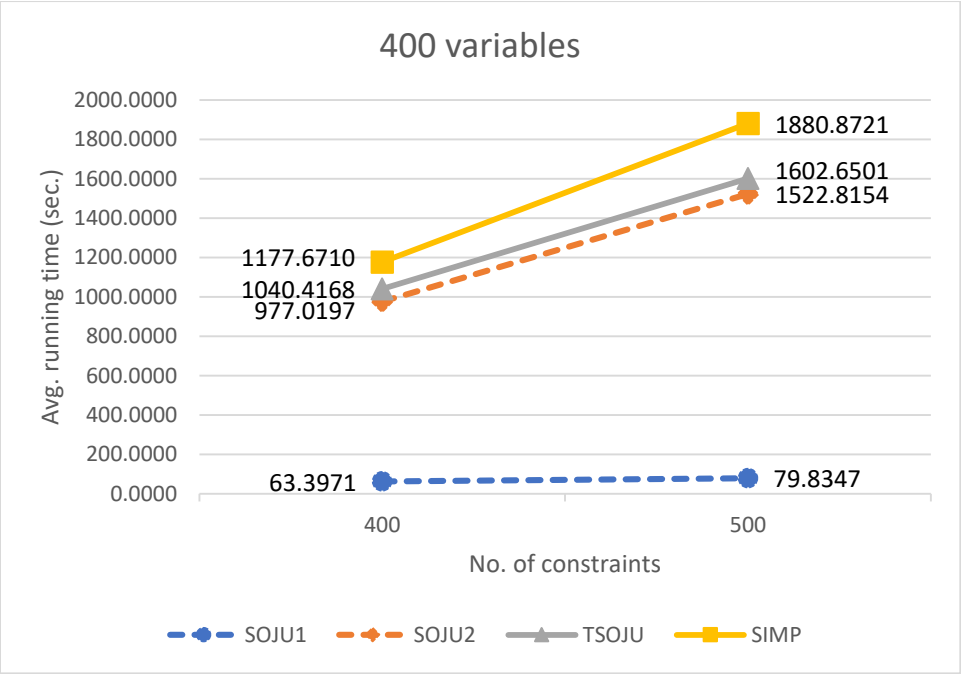


a) Medium number of constraints

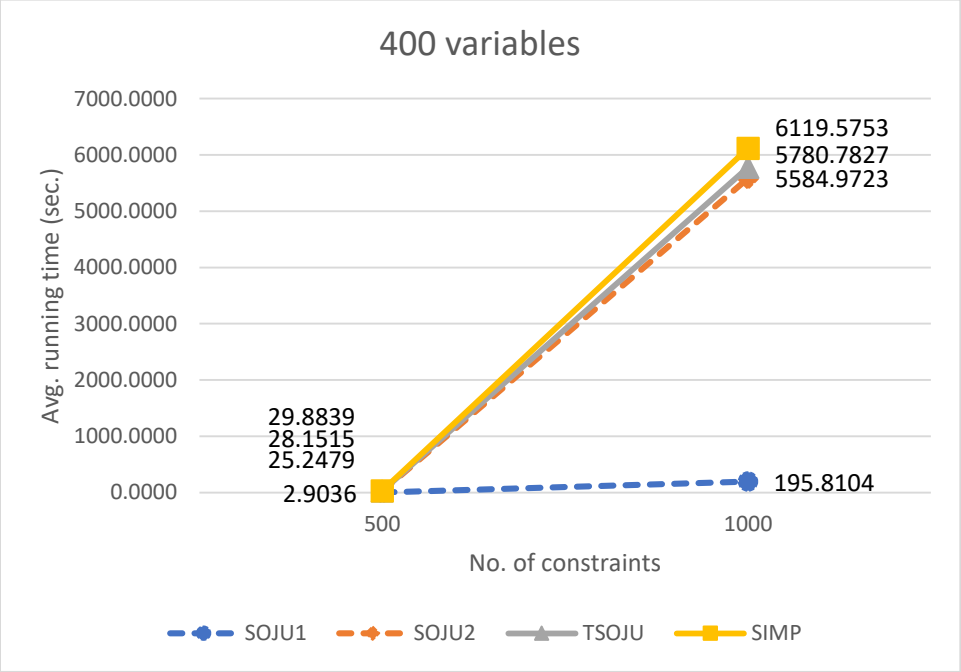


b) Large number of constraints

Figure 4.27: The average number of iterations by SIMP and SOJU with 300 variables of Test I.



a) Medium number of constraints



b) Large number of constraints

Figure 4.28: The average number of iterations by SIMP and SOJU with 400 variables in Test I.

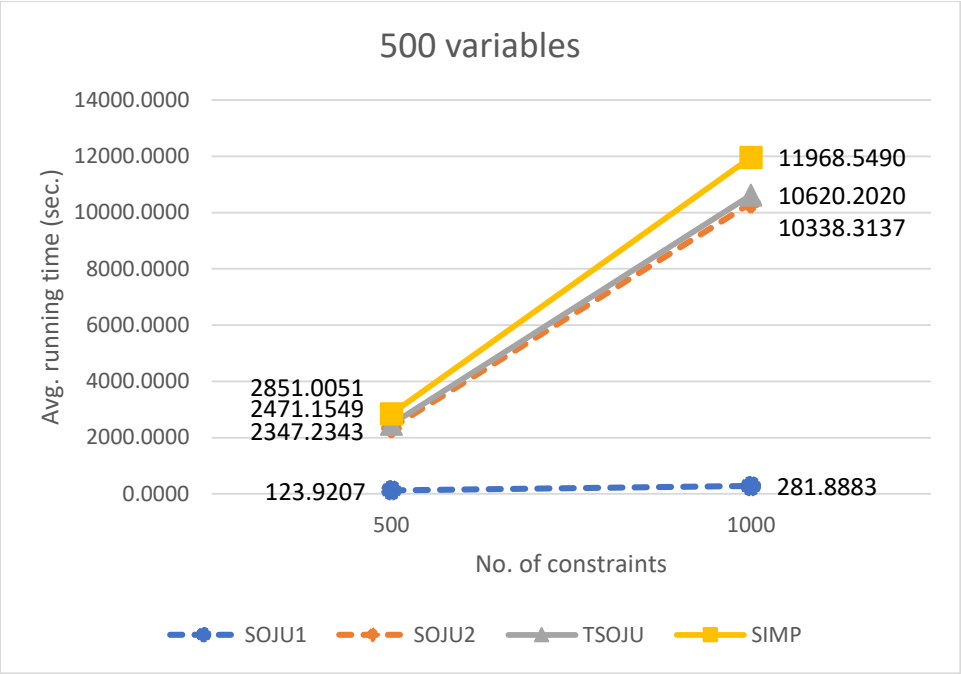


Figure 4.29: The average number of iterations by SIMP and SOJU with 500 variables in Test I.

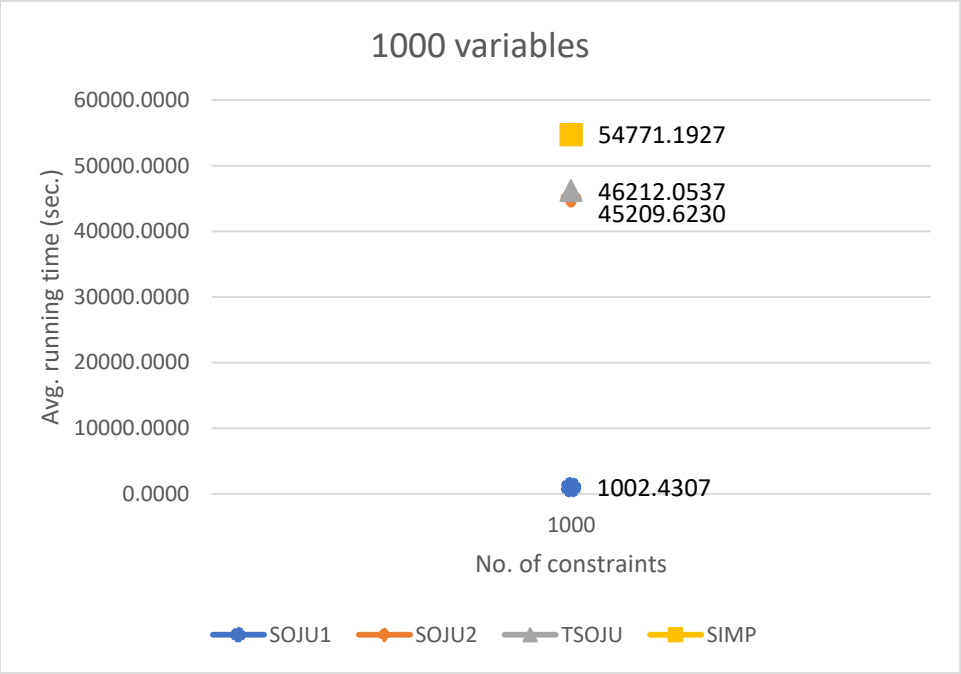
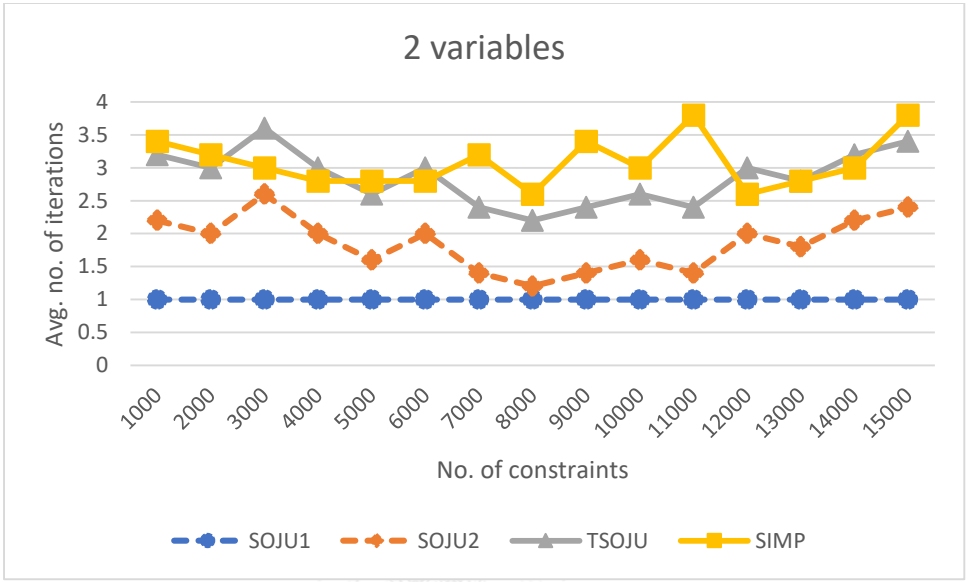


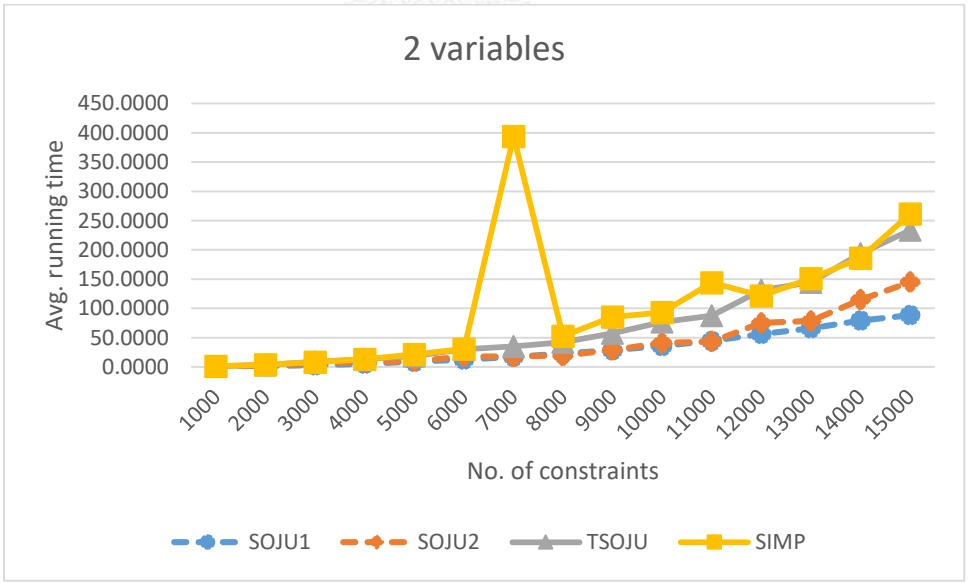
Figure 4.30: The average number of iterations by SIMP and SOJU with 1000 variables in Test I.

4.2.3 Average number of iterations and running time (Test II)

When SOJU was tested on problems with small number of variables and much larger number of constraints, the results were not much different from SIMP in terms of number of iterations and running time. (See Figures 4.31 – 4.36.)

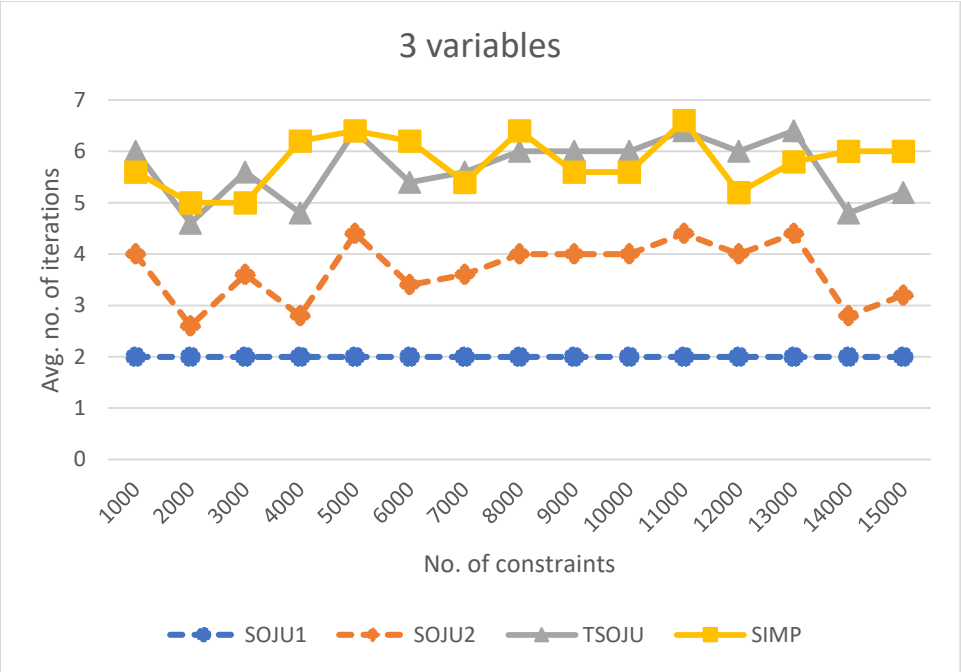


a) Average number of iterations

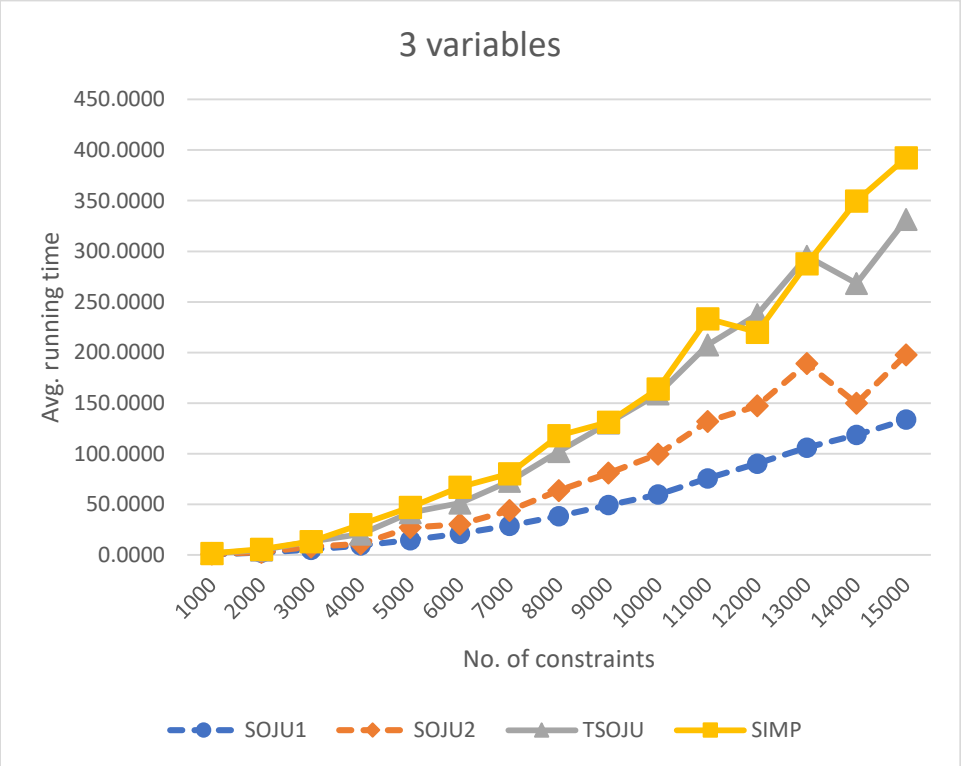


b) Average running time

Figure 4.31: The average number of iterations running time by SIMP and SOJU with 2 variables in Test II.

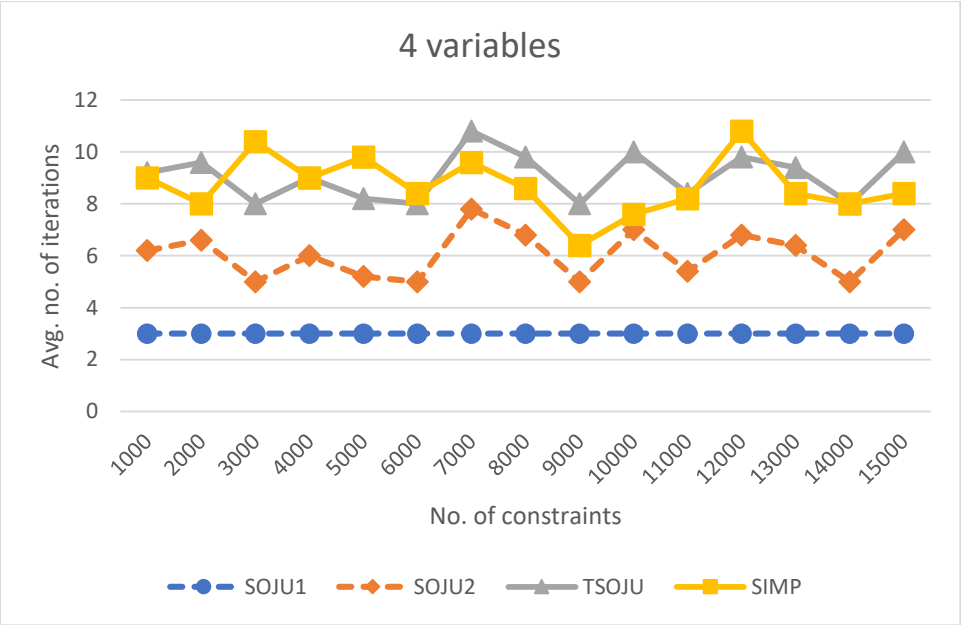


a) Average number of iterations

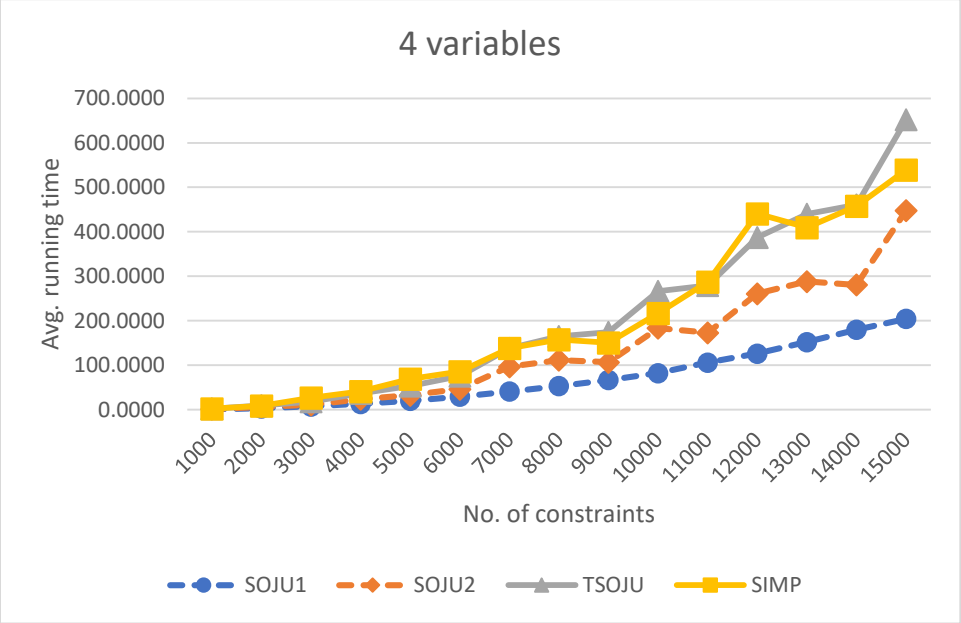


b) Average running time

Figure 4.32: The average number of iterations and running time by SIMP and SOJU with 3 variables in Test II.

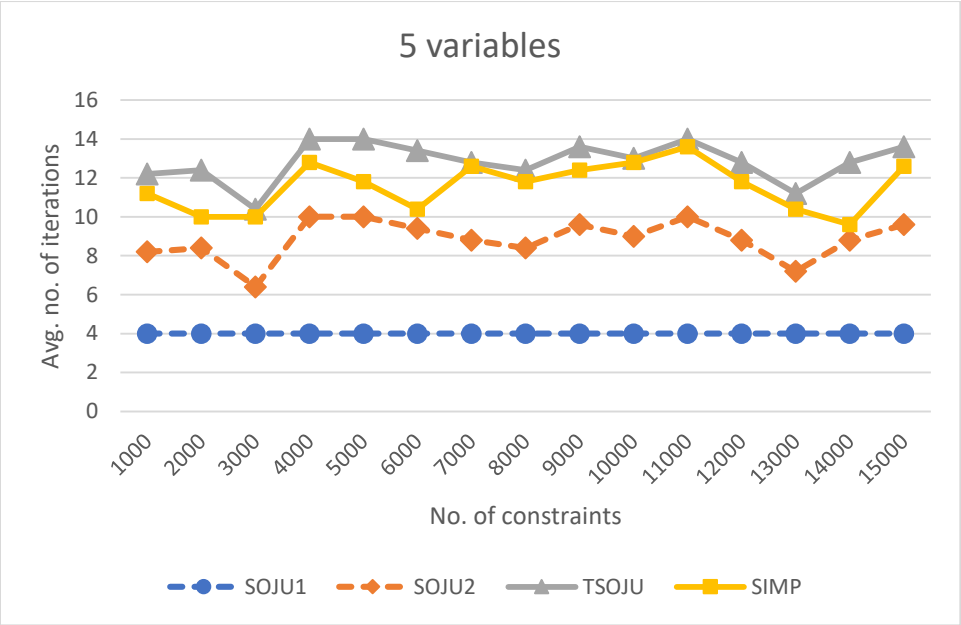


a) Average number of iterations

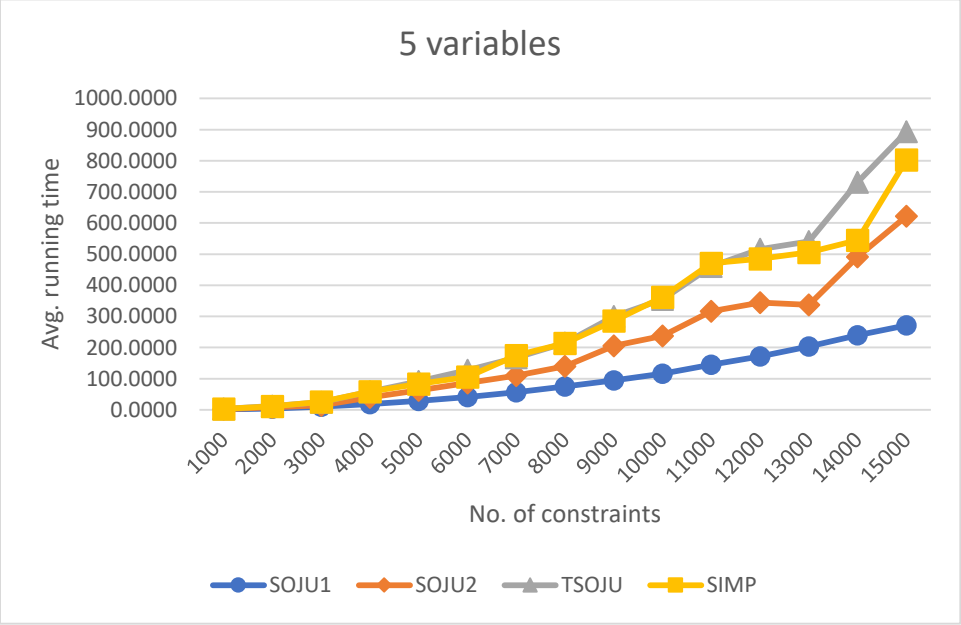


b) Average running time

Figure 4.33: The average number of iterations and running time by SIMP and SOJU with 4 variables in Test II.

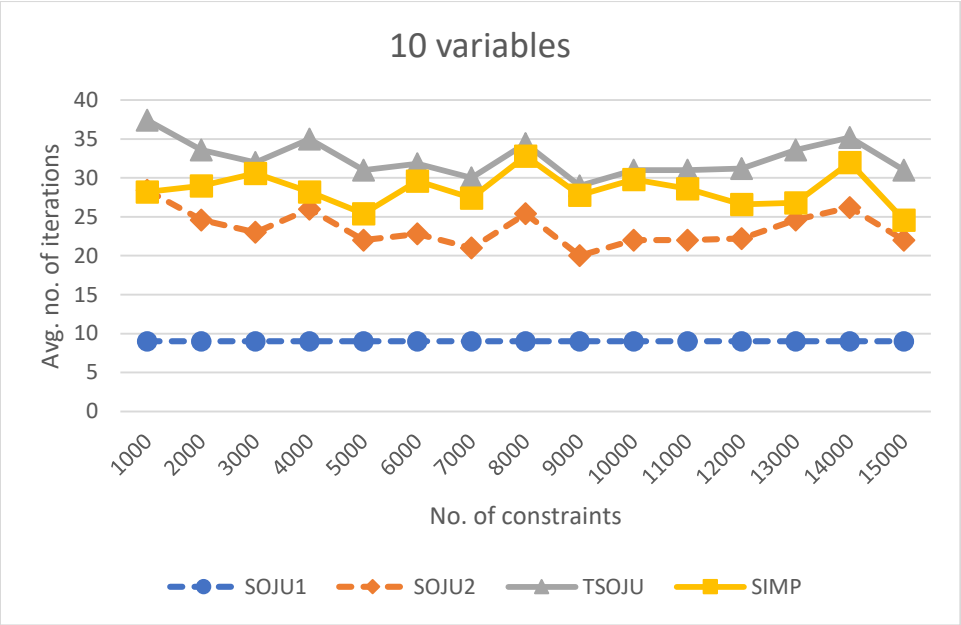


a) Average number of iterations

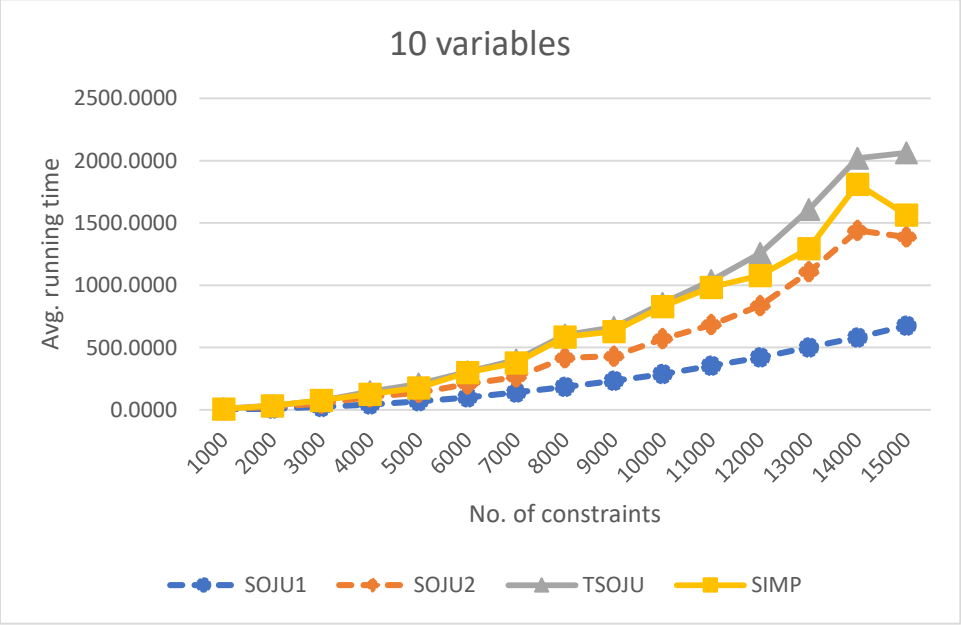


b) Average running time

Figure 4.34: The average number of iterations and running time by SIMP and SOJU with 5 variables in Test II.

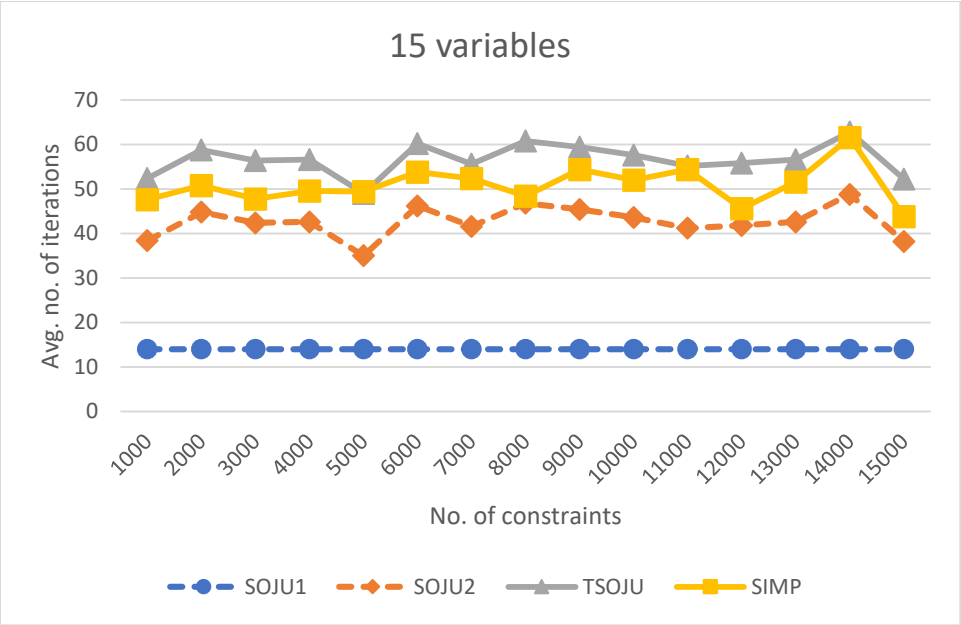


a) Average number of iterations

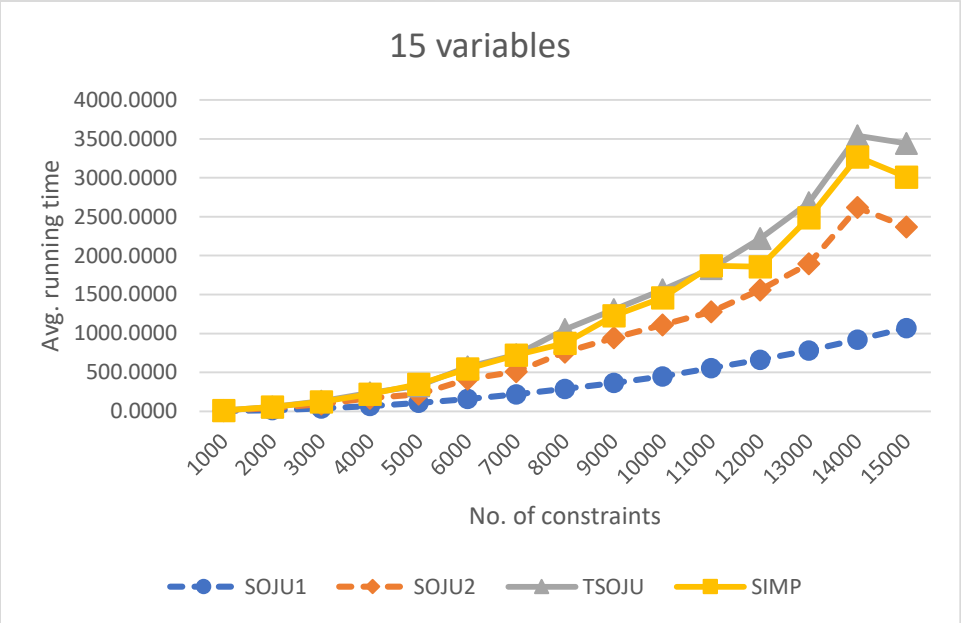


b) Average running time

Figure 4.35: The average number of iterations and running time by SIMP and SOJU with 10 variables in Test II.



a) Average number of iterations



b) Average running time

Figure 4.36: The average number of iterations and running time by SIMP and SOJU with 15 variables in Test II.

CHAPTER 4
CONCLUSION AND SUGGESTION

The simplex method with objective jump was proposed to improve the initial basic feasible solution for solving the linear programming problem. SOJU algorithm was divided into 2 stages. The first stage was the process to find a new initial basic feasible solution called objective jump. At this stage, the process always takes all $n-1$ iterations to pivot out from the jump point and is operated on the only one sub-LP problem from all possible 2^{n-1} sub-problems. Another stage was the regular simplex algorithm to find the optimal solution by starting with the initial basic feasible solution obtained in the first stage.

The algorithm was tested in the randomly generated linear programming problems with two test configurations and the results were shown in terms of the average number of iterations and running time. In most cases, the second stage of SOJU took smaller number of iterations and running time than SIMP, which means the SOJU new initial basis outperformed the traditional basis used in SIMP. However, the first stage SOJU is needed to be considered to fairly compare SOJU and SIMP.

The results show that SOJU performed better than SIMP on problems with 2 to 5 variables and the number of constraints is no more than 1000. Interestingly, SOJU also was faster than SIMP for most cases when $300 \le m \le 1000$ and $40 \le n \le m$ although the number of iterations was larger than SIMP. In other cases, SOJU either had the comparable or worse performance than SIMP.

For the future work, simplex method with objective jump should improve the process in the first stage which is to identify the jump point and corresponding basis by reducing the process to create the objective jump tableau. LU factorization may be used to find the inverse of large matrix. Additionally, the pivot operation after the jump point in the second stage should be improved by selecting other appropriated sub-problems formulation from all possible 2^{n-1} sub-problems. Moreover, another direction can be used instead of the gradient of the objective function. Furthermore, the work could be

extended to the negative gradient of the objective function. Finally, the process of the second stage can be improved by traversing through a better region after the jump point using different pivot rules such as steepest edge and devex.

REFERENCES

[1] Bazaraa, M.S., Jarvis, J.J., and Sherali, H.D. Linear programming and network flows. NJ: A John Wiley & Sons, Inc., Publication, 2010.

[2] Bland, R.G. New finite pivoting rules for the simplex method. Mathematics of Operations Research 2 (May 1977) : 103-107.

[3] Continuum analytics, Download anaconda distribution [online]. 2016. Available from: https://www.continuum.io/downloads [2016, Sep 8].

[4] Dantzig, G.B. Linear programming and extensions. Princeton, The Rand Corporation, 1963.

[5] Forrest, J.J. and Goldfarb, D. Steepest-edge simplex algorithms for linear programming. Mathematical Programming 57 (August 1992) : 341-374.

[6] Harris, P.M.J. Pivot selection methods of the devex LP code. Mathematical Programming 5 (Feburary 1973) : 1-28.

[7] Junior, H.V., and Lins, M.P.E. An improved initial basis for the simplex algorithm. Computers & Operations Research 32 (2005) : 1983-1993.

[8] Karwan, M.H., Lotfi, V., Telgen, J., and Zionts S. Redundancy in mathematical programming. Buffalo, NY: Springer-Verlag, 1983.

[9] Klee, V. and Minty, G.J. How good is the simplex algorithm? Academic Press, NY, 1971.

[10] Lawanyawut, N. Sirirat, C., and Yawila N. Simplex method with the objective jump. Senior Project, Department of Mathematics and Computer Science, Faculty of Science, Chulalongkorn University, 2014.

[11] Pan, P.Q. A largest-distance pivot rule for the simplex algorithm. European Journal of Operational Research 187 (June 2008) : 393-402.

[12] Potraa, F.A. and Wright, S.J. Interior-point methods. Journal of Computational and Applied Mathematics 124 (February 2000) : 281–302.

APPENDICES

APPENDICES

A1: Python source code for SOJU algorithm and numerical experiments

```
import numpy as np
import sympy as sp
import random
from numpy.linalg import inv
import fractions
from copy import copy, deepcopy
import itertools
import time
sp.init_printing()
```

Random LP model

```
def random_LP(ex,m,n):
    m = m
    n = n
    c = np.random.rand(n)*10
    A = np.random.rand(m,n)*20-10
    AA =A.dot(1/c)
    if all(AA[i]<=0 for i in range(m)):
            print("u")
            AA[-1]=-1*AA[-1]
            A[-1]=-1*A[-1]
    x = np.random.rand(n)*10
    # Ax=b guarantee that there is a feasible solution
    b = A.dot(x)
    for i in range(m):
        if b[i] < 0:
            b[i] = -1*b[i]
    return c.tolist(),b.tolist(),A.tolist()
```

Test Example

```
np.random.seed(0)
mList = [1000,2000,3000,4000,5000,6000,7000,8000,9000,10000,11000,120
00,13000,14000,15000]
nList = [2,3,4,5,10,15]
exList = [1,2,3,4,5]
lpex = []
l = 0
for j in nList:
    for i in mList :
        for k in exList :
```

```
            lpex.append(random_LP(k,i,j))
            print(k,i,j,"lpex",l)
            l=l+1
```

SOJU Algorithm

```
def chooseBinedConstr(n,m,c,b,A,report =False):
    t00 = time.time()
    ratio = []
    indratio = []
    for i in range(m):
        S=sum(A[i][j]/c[j] for j in range(n))
        if S > 0:
            R = b[i]/S
            ratio.append(R)
            indratio.append(i)

    q = min(enumerate(ratio), key = lambda ratio:ratio[1])[0]
    if report:
        print("indratio=",indratio)
        print("ratio=",ratio)
        print("q=",q)
    "............................................................"
    ##createObjConstr(n,m,c):
    objConstr = []
    qq = indratio[q]
    objConstr.append(A[qq][:])

    #O(n^2)
    ##objConstr.append(A[0][:])
    objConstr.extend([[] for i in range(n-1)])
    for i in range(n-1):
        objConstr[i+1].extend([0 for i in range(n)])
    for i in range(1,n):
        for j in range(i+1):
            if j < i:
                objConstr[i][j] = 1*c[j]
            else:
                objConstr[i][j]=-1*(i)*c[j]

    if report:
        print("qq=",qq)
        print("objConstr=",np.matrix(objConstr))

    ##Generate B^-1N
```

```
#J,J_inv
J = objConstr
J_inv = inv(np.matrix(J)).tolist()

#Q
Q = []
Q.extend(A)
Q.remove(Q[qq])

#QJ_inv
QJ_inv = (-1*np.matrix(Q)*J_inv).tolist()

#B_invN
B_invN = []
B = J_inv+QJ_inv
B_invN.extend(B)

if report:
    print("J=",J)
    print("J_inv=",J_inv)
    print("Q=",Q)
    print("QJ_inv=",QJ_inv)
    print("B_invN=",B_invN)


##Generate B^-1b_0=>RHS
#J_invb2
J_invb2 = []
for i in range(n):
    J_invb2.append(J_inv[i][0]*b[qq])

#b_1
b_1 = []
b_1.extend(b)
b_1.remove(b_1[qq])

#b1QJ=b_1-QJ_invb2
b1QJ=[]
for i in range(m-1):
    b_2 = b_1[i]+b[qq]*QJ_inv[i][0]
    b1QJ.append(b_2)
#RHS
RHS=J_invb2+b1QJ

if report:
```

```
    print("b_1=",b_1)
    print("J_invb2=",J_invb2)
    print("b1QJ=",b1QJ)
    print("RHS=",RHS)

##create cJ_inv=>reducedcost
##create cB_invb=>obj
cJ_inv=[]
for j in range(n):
    reducedcost = sum(c[i]*J_inv[i][j] for i in range(n))
    cJ_inv.append(reducedcost)

##create obj
obj = sum(c[j]*J_invb2[j] for j in range(n))

if report:
    print("reducedcost=",cJ_inv)
    print("obj=",obj)

"............................................................"

## Transform inputs
cJ_inv.extend([0 for i in range(m+n-1)])

ind = 0
for i in range(m+n-1):
    B_invN[i].extend([0 for j in range(m+n-1)])
    B_invN[i][n+ind] = 1
    ind = ind+1

if report:
    print ("objump=",np.matrix(B_invN))

## Implement objective jump tableau
# Generate the initial tableau and the basic var. index
curTableau = [cJ_inv[:]]
curTableau[0].append(obj)
for i in range(m+n-1):
    temp = B_invN[i][:]
    temp.append(RHS[i])
    curTableau.append(temp)
indB = [n+i for i in range(m+n-1)]
indN = [i for i in range(n)]
indRHS = n+m+n-1
if report:
```

```
        print (indN ," " ,indB)
        print(np.matrix(curTableau))

    "............................................................"
    ##Stage1
    #Code loop
    #iteration = 0
    k=0
    for l in range(n-1):
        # Minimum ratio test
        validValue = []
        validIndex = []
        for i in range(m+n-1-l):
            if curTableau[i+1][k+1] > 0:
                validIndex.append(i)
                validValue.append(curTableau[i+1][-1]/curTableau[i+1]
[k+1])
            else:
                validIndex.append(float("inf"))
                validValue.append(float("inf"))

        if report:
            print ("validValue=",validValue)
            print ("validIndex=",validIndex)

        r = min(enumerate(validValue),key = lambda x: x[1])[0]
        if report:
            print ("indB=",indB)
            print ("r=",r,"indB[r]=",indB[r])

        ## Pivot between r and k
        pivot = curTableau[r+1][k+1]
        if report:
            print ("pivot",pivot)

        #rowr=rowrj/pivot
        curTableau[r+1] = [curTableau[r+1][j]/pivot for j in range(n+
m+n-l)]

        for i in range(m+n-l):
            if i < r+1 or i > r+1:
                curTableau[i] = [curTableau[i][j] - curTableau[i][k+1
]*curTableau[r+1][j] for j in range(n+m+n-l)]

        #delete row,column,index
```

```
        curTableau.remove(curTableau[r+1])
        for i in range(m+n-1-l):
            del curTableau[i][k+1]
        indB.remove(indB[r])
        indN.remove(indN[k+1])

        if report:
            print(indN,indB)
            print(np.matrix(curTableau))
    return (curTableau)
    "............................................................."
def SimplexJump(m,n,c,b,A,dir='max',report = False):
    t0 = time.time()
    startTableau = deepcopy(chooseBinedConstr(n,m,c,b,A))
    print("%s" % (time.time() - t0))
    if report:
        print("startTableau",np.matrix(startTableau))

    iteration = 0
    while True:
        row0 = [startTableau[0][j] for j in range(n+m)]
        k = min(enumerate(row0),key = lambda x: x[1])[0] #k is the in
dex of indN, the original index, list of indN, is indN[k]
        # If the row0[k] is negative, then the optimal solution is re
ached.
        if startTableau[0][k] >= 0:
            #print("The current tableau gives the optimal solution.")
            #print ("Iteration ", iteration)
            #print(iteration)
            #print(startTableau[0][-1])
            return
        # Minimum ratio test
        validValue = []
        validIndex = []
        for i in range(1,m+1):
            if startTableau[i][k] > 0:
                validIndex.append(i-1)
                validValue.append(startTableau[i][-1]/startTableau[i]
[k])
            else:
                validIndex.append(float("inf"))
                validValue.append(float("inf"))
        # Check for unboundedness.
        if [validValue[i] for i in range(m)] == [float("inf") for i i
n range(m)]:
```

```
            print ("This problem has the unbounded feasible region.")
            return
        if report:
            print (validIndex)
            print (validValue)
        # Pivot between r and k
        r = min(enumerate(validValue),key = lambda x: x[1])[0]
        if report:
            print (r,k)
        pivot = startTableau[r+1][k]
        if report:
            print (pivot)
        startTableau[r+1] = [startTableau[r+1][j]/pivot for j in rang
e(n+m+1)]
        startTableau[0] = [startTableau[0][j] - startTableau[0][k]*st
artTableau[r+1][j] for j in range(n+m+1)]
        for i in range(m):
            if i < r or i>r :
                startTableau[i+1] = [startTableau[i+1][j] - startTabl
eau[i+1][k]*startTableau[r+1][j] for j in range(n+m+1)]
        if report:
            print(np.matrix(startTableau))
        iteration = iteration +1
```

A2: Python source code for SIMP algorithm and numerical experiments

```
def Simplex(m,n,c,b,A, dir='max',report = True):
    #print ("---------- Simplex Method ----------\n")
    m = m
    n = n
    c = c[:]
    from copy import copy, deepcopy
    A = deepcopy(A)
    b = b[:]
    ## Transform inputs
    c.extend([0 for i in range(m)])
    ind = 0
    for i in range(m):
        A[i].extend([0 for j in range(m)])
        A[i][n+ind] = 1
        ind = ind+1
    if report:
        print (np.matrix(A))
```

```
    ## Implement Simplex tableau
    # Generate the initial tableau and the basic var. index
    curTableau = [c[:]]
    curTableau[0].append(0)
    for i in range(m):
        temp = A[i][:]
        temp.append(b[i])
        curTableau.append(temp)
    indB = [n+i for i in range(m)]
    indN = [i for i in range(n)]
    indRHS = len(curTableau[0])-1
    if report:
        print (indN ," " ,indB)
        print(np.matrix(curTableau))

    ## Code loop
    iteration = 0
    while True:
        row0 = [curTableau[0][j] for j in indN]
        k = max(enumerate(row0),key = lambda x: x[1])[0] #k is the in
dex of indN, the original index, list of indN, is indN[k]
        # If the row0[k] is negative, then the optimal solution is re
ached.
        if curTableau[0][indN[k]] <= 0:
            #print ("The current tableau gives the optimal solution."
)
            #print ("The optimal objective value is ", -curTableau[0]
[indRHS])
            #print ("Iteration ", iteration)
            print(iteration)
            #print(-curTableau[0][indRHS])
            return

        # Minimum ratio test
        validValue = []
        validIndex = []
        for i in range(1,m+1):
            if curTableau[i][indN[k]] > 0:
                validIndex.append(i-1)
                validValue.append(curTableau[i][indRHS]/curTableau[i]
[indN[k]])
            else:
                validIndex.append(float("inf"))
                validValue.append(float("inf"))
```

```
        # Check for unboundedness.
        if [validValue[i] for i in range(m)] == [float("inf") for i i
n range(m)]:
            print ("This problem has the unbounded feasible region.")
            return
        if report:
            print (validIndex)
            print (validValue)

        # Pivot between r and k
        from fractions import Fraction
        from decimal import Decimal
        r = min(enumerate(validValue),key = lambda x: x[1])[0]
        if report:
            print (indB)
            print (r,indB[r])
        pivot = curTableau[r+1][indN[k]]
        if report:
            print (pivot)
        curTableau[0] = [curTableau[0][j] - curTableau[0][indN[k]]*cu
rTableau[r+1][j]/pivot for j in range(len(curTableau[i]))]
        for i in range(m):
            if i < r:
                curTableau[i+1] = [curTableau[i+1][j] - curTableau[i+
1][indN[k]]*curTableau[r+1][j]/pivot for j in range(len(curTableau[i]
))]
            elif i > r:
                curTableau[i+1] = [curTableau[i+1][j] - curTableau[i+
1][indN[k]]*curTableau[r+1][j]/pivot for j in range(len(curTableau[i]
))]
        curTableau[r+1] = [curTableau[r+1][j]/pivot for j in range(le
n(curTableau[r+1]))]

        if report:
            print (indN , indB)
            print (indN[k] , indB[r])
        indB[r], indN[k] = indN[k], indB[r]

        if report:
            print (indN, indB)
            print(np.matrix(curTableau))
        iteration = iteration +1
```

VITA

Name: Miss Nutcha Yawila

Birthdate: 1/11/1991 Birthplace: Lamphun

Education:

2017: M.Sc. (Applied Mathematics and Computational Science), Chulalongkorn University, Thailand

2014: B.Sc (Mathematics), Chulalongkorn University, Thailand

Publication:

N. Yawila, B. Intiyot, and K. Sinapiromsaran,
"Simplex Method with Objective Jump",
Proceeding of International Conference on
Applied Statistics 2016 (ICAS 2016),
13-15 July 2016, Phuket, Thailand, pp. O-193-O198