Parabola Volume 59, Issue 3 (2023)

Flow-shop scheduling with max-plus algebra

Ricky Wijaya[1], **Jonathan Hoseana**[2], and **Iwan Sugiarto**[3]

1 Introduction

Consider a project, call it Project A, consisting of three jobs: Job 1, Job 2 and Job 3, each of which must be processed at two machines: Machine 1 and Machine 2. Job 1 must be processed at Machine 1 for 3 minutes, before being processed at Machine 2 for 2 minutes. Job 2 must be processed at Machine 1 for 3 minutes, before being processed at Machine 2 for 3 minutes. Job 3 must be processed at Machine 1 for 1 minute, before being processed at Machine 2 for 4 minutes. See Table 1. Assume that each machine can only process a single job at a time, but simultaneous processings of different jobs at different machines are possible. If the three jobs have to be processed in order, then what is the total time required to finish the project?

Machine	Job 1	Job 2	Job 3
Machine 1	3	3	1
Machine 2	2	3	4

Table 1: The processing duration, in minutes, of each job at each machine for Project A.

To answer this question, one could construct a time diagram, as in Figure 1, which shows the job being processed at each machine at any given time t (in minutes). The symbols M_1 and M_2 represent Machine 1 and Machine 2, respectively, while the symbols J_1, J_2, and J_3 represent Job 1, Job 2, and Job 3, respectively. We can see that the last process in the project (i.e., the processing of Job 3 at Machine 2) finishes at $t = 13$. Therefore, the total time required to finish the project is 13 minutes.

Next, what if the three jobs do not have to be processed in order? Can we find an order in which these jobs can be processed such that the project finishes in shorter time?

Since this project consists only of three jobs, there are only $3! = 6$ possible orders, allowing exhaustive checking. The first possible order – 1, 2, 3 – is considered in Figure 1 and gives a total time of 13 minutes. The other five possible orders are

$$1,3,2\,, \qquad 2,1,3\,, \qquad 2,3,1\,, \qquad 3,1,2\,, \qquad \text{and} \qquad 3,2,1\,. \tag{1}$$

[1]Ricky Wijaya is an undergraduate student at the Department of Mathematics, Parahyangan Catholic University, Bandung, Indonesia. (`rickywijaya2311@gmail.com`)

[2]Jonathan Hoseana is the Head of the Centre for Mathematics and Society, Department of Mathematics, Parahyangan Catholic University, Bandung, Indonesia. (`j.hoseana@unpar.ac.id`)

[3]Iwan Sugiarto is a researcher at the Centre for Mathematics and Society, Department of Mathematics, Parahyangan Catholic University, Bandung, Indonesia. (`iwans@unpar.ac.id`)

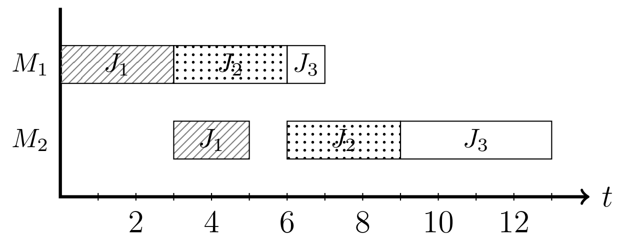

Figure 1: A time diagram for Project A, showing the job being processed at each machine at any given time t, in the case of the three jobs being processed in order.

Time diagrams corresponding to these orders are presented in Figure 2. From the diagrams, we can see that these orders give total times of 12 minutes, 12 minutes, 12 minutes, 10 minutes, and 10 minutes, respectively. Thus, to finish the project in the shortest possible time, the three jobs must be processed in the order

$$3, 1, 2 \qquad \text{or} \qquad 3, 2, 1.$$

Using either of these orders, the project finishes in only 10 minutes.

A natural question now arises. For a project consisting of n jobs to be processed at m machines, is there a way to determine an order in which the jobs should be processed such that the project finishes in the shortest possible time, without exhaustive checking? This is the so-called *flow-shop scheduling problem*. The term *flow-shop* entails the condition that each job is processed at the machines in the same order, indicated by the numbers by which the machines are labelled [2, Sec. 3.3]. Without this condition, the problem is referred to as the *job-shop scheduling problem* [2, Sec. 3.3]. An order of jobs minimising the project's finish time is referred to as an *optimal* order.

In this article, we discuss methods to solve the flow-shop scheduling problem, which involve modelling the problem using matrices over the *max-plus algebra*. We intend to provide not a new method but an exposition of methods already existing in the literature, applying them in a number of illustrative examples. First, we shall deal with two-machine projects, discussing the so-called *JG algorithm* of Bouquard, et al. [1, Sec. 3], a generalised version of an algorithm of Johnson [6], which can be used to find an optimal scheduling of all two-machine projects (Sections 2–4). The basic idea is to represent each job as a matrix, naturally called a *job matrix*, so that the problem of finding an optimal job order becomes that of finding an order in which these job matrices should be multiplied to obtain a result which is minimum with respect to a certain order. Through a computer experiment, we compare the efficiency of the JG algorithm to that of exhaustive checking (Section 4). Subsequently, we deal with the multi-machine case, discussing methods which reduce certain multi-machine projects into two-machine projects [2, 8], which can then be scheduled using the JG algorithm (Section 5). To finish, we provide a brief remark and suggest further readings on general algorithms applicable to the multi-machine case.

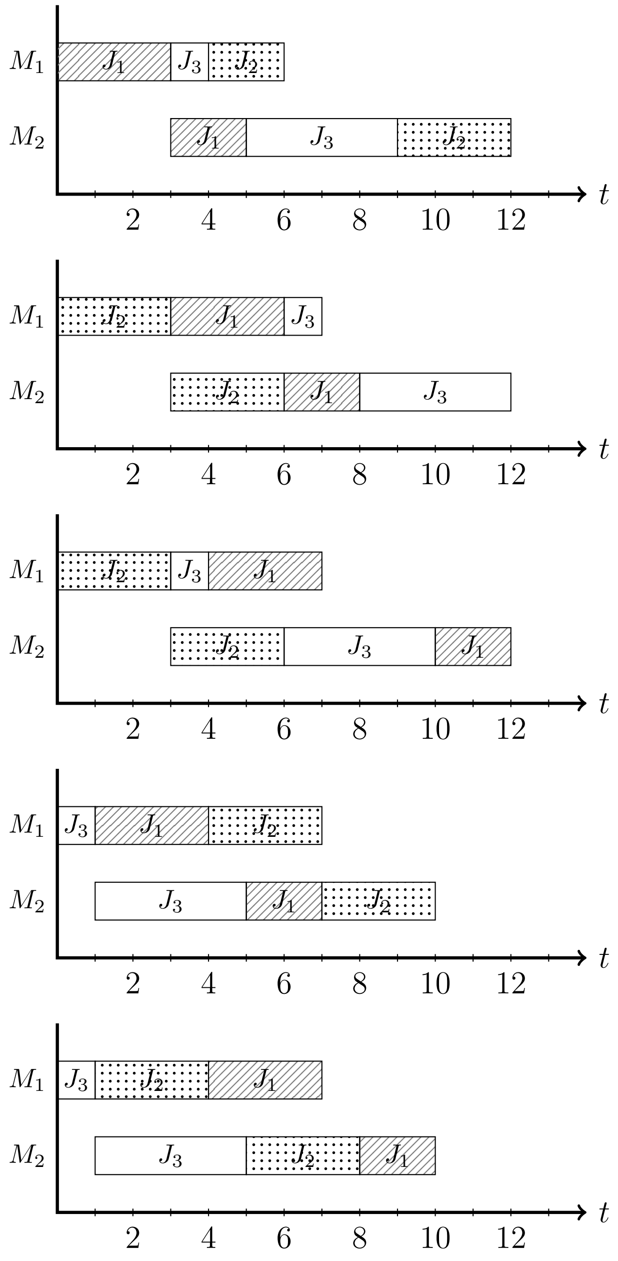

Figure 2: Time diagrams for Project A, showing the job being processed at each machine at any given time t, using the job orders given by (1).

2 The two-machine case

Consider a project consisting of n jobs, Job 1, ..., Job n, which must be processed at two machines: Machine 1 and Machine 2. Each job must be processed at Machine 1 before being processed at Machine 2. For every $i \in \{1, \ldots, n\}$ and $j \in \{1, 2\}$, let $p_{i,j}$ be the processing duration of Job i at Machine j. We aim to establish an optimal processing order of these n jobs.

An order of these n jobs can be represented as a *permutation* of the set $\{1, \ldots, n\}$, that is, a function $\sigma : \{1, \ldots, n\} \to \{1, \ldots, n\}$ which is bijective; that is, both one-to-one and onto [3, p. 41]. Let us agree that σ is such that for every $i \in \{1, \ldots, n\}$, the i-th job to be processed is Job $\sigma(i)$. If σ represents an optimal processing order, then we shall refer to σ as an *optimal permutation*.

Suppose that the process begins at time $t = 0$. For each $i \in \{1, \ldots, n\}$ and $j \in \{1, 2\}$, let $t = F_j(i)$ be the finish time of the i-th job to be processed, Job $\sigma(i)$, at Machine j. Thus, the entire project finishes at $t = F_2(n)$. Our goal, therefore, is to minimise $F_2(n)$.

Since the processing of the i-th job at Machine 1 can only begin once the processing of the $(i-1)$-th job at Machine 1 has finished, we have

$$F_1(i) = p_{\sigma(i),1} + F_1(i-1)\,.$$

Moreover, since the processing of the i-th job at Machine 2 can only begin once both the processing of the i-th job at Machine 1 and the processing of the $(i-1)$-th job at Machine 2 has finished, we have that

$$\begin{aligned} F_2(i) &= p_{\sigma(i),2} + \max\{F_1(i),\ F_2(i-1)\} \\ &= \max\left\{p_{\sigma(i),2} + F_1(i),\ p_{\sigma(i),2} + F_2(i-1)\right\} \\ &= \max\left\{p_{\sigma(i),2} + \left(p_{\sigma(i),1} + F_1(i-1)\right),\ p_{\sigma(i),2} + F_2(i-1)\right\}. \end{aligned}$$

Therefore, we obtain the system of recursions

$$\begin{cases} F_1(i) = p_{\sigma(i),1} + F_1(i-1)\,, \\ F_2(i) = \max\left\{\left(p_{\sigma(i),2} + p_{\sigma(i),1}\right) + F_1(i-1),\ p_{\sigma(i),2} + F_2(i-1)\right\}, \end{cases} \tag{2}$$

for which we define the initial values

$$F_1(0) = F_2(0) = 0\,. \tag{3}$$

Note that the right-hand sides of the recursions in (2) involves exactly two operations: maximum and addition. The idea of *max-plus algebra* is to resymbolise these operations, conventionally as $\oplus$ and $\otimes$, so that the system of equations (2) can be written in a tidier form, which subsequently allows us to obtain the desired scheduling via matrix-based techniques.

3 The max-plus algebra

Let $\varepsilon := -\infty$ and $e := 0$. The *max-plus algebra* is the set $\mathbb{R}_{\max} := \mathbb{R} \cup \{\varepsilon\}$ equipped with the addition $\oplus$ and multiplication $\otimes$ defined by

$$\begin{aligned} & x \oplus y := \max\{x, y\} \\ \text{and} \qquad & x \otimes y := x + y \end{aligned}$$

for every $x, y \in \mathbb{R}_{\max}$ [5, p. 13]. The adjoined element ε is intended to satisfy

$$x \oplus \varepsilon = \varepsilon \oplus x = x \qquad \text{and} \qquad x \otimes \varepsilon = \varepsilon \otimes x = \varepsilon$$

for every $x \in \mathbb{R}_{\max}$, the former making ε the identity for the operation $\oplus$.

One verifies that the addition $\oplus$ is associative, commutative, and has an identity:

P1 for all $x, y, z \in \mathbb{R}_{\max}$, we have $(x \oplus y) \oplus z = x \oplus (y \oplus z)$;

P2 for all $x, y \in \mathbb{R}_{\max}$, we have $x \oplus y = y \oplus x$;

P3 there exists $\varepsilon \in \mathbb{R}_{\max}$ such that, for each $x \in \mathbb{R}_{\max}$, we have $x \oplus \varepsilon = \varepsilon \oplus x = x$;

that the multiplication $\otimes$ is associative:

P4 for all $x, y, z \in \mathbb{R}_{\max}$, we have $(x \otimes y) \otimes z = x \otimes (y \otimes z)$;

and that the multiplication $\otimes$ is two-sided distributive over the addition $\oplus$:

P5 for all $x, y, z \in \mathbb{R}_{\max}$, $x \otimes (y \oplus z) = (x \otimes y) \oplus (x \otimes z)$ and $(y \oplus z) \otimes x = (y \otimes x) \oplus (z \otimes x)$.

Thus, the max-plus algebra satisfies all axioms of a *ring* except the existence of inverses with respect to $\oplus$. For this reason, the max-plus algebra is said to be a *semiring* [5, p. 15]. In fact, the max-plus algebra is a *commutative unital semiring,* i.e., a semiring in which the multiplication $\otimes$ is commutative and has an identity:

P6 for all $x, y \in \mathbb{R}_{\max}$, we have $x \otimes y = y \otimes x$;

P7 there exists $e \in \mathbb{R}_{\max}$ such that, for every $x \in \mathbb{R}_{\max}$, we have $x \otimes e = e \otimes x = x$.

The associativity of both $\oplus$ and $\otimes$ implies that there is no ambiguity in the expressions $x \oplus y \oplus z$ and $x \otimes y \otimes z$.

The algebra of matrices over the max-plus algebra is developed via natural definitions [5, p. 17–20]. We denote by $\mathbb{R}_{\max}^{2\times 2}$ the set of all 2×2 matrices whose entries belong to $\mathbb{R}_{\max}$. For every $A = \left(a_{i,j}\right)_{2\times 2}$ and $B = \left(b_{i,j}\right)_{2\times 2}$ in $\mathbb{R}_{\max}^{2\times 2}$, we define

$$\begin{aligned} & A \oplus B := \left(a_{i,j} \oplus b_{i,j}\right)_{2\times 2} \\ \text{and} \qquad & A \otimes B := \left(c_{i,j}\right)_{2\times 2}, \end{aligned}$$

where

$$c_{i,j} := \left(a_{i,1} \otimes b_{1,j}\right) \oplus \left(a_{i,2} \otimes b_{2,j}\right) \oplus \cdots \oplus \left(a_{i,m} \otimes b_{m,j}\right) .$$

Thus, for the matrices

$$A = \begin{pmatrix} 5 & -2 \\ 10 & e \end{pmatrix} \qquad \text{and} \qquad B = \begin{pmatrix} e & \varepsilon \\ 1 & -8 \end{pmatrix} \tag{4}$$

we have

$$\begin{aligned} A \otimes B &= \begin{pmatrix} (5 \otimes e) \oplus ((-2) \otimes 1) & (5 \otimes \varepsilon) \oplus ((-2) \otimes (-8)) \\ (10 \otimes e) \oplus (e \otimes 1) & (10 \otimes \varepsilon) \oplus (e \otimes (-8)) \end{pmatrix} \\ &= \begin{pmatrix} 5 \oplus (-1) & \varepsilon \oplus (-10) \\ 10 \oplus 1 & \varepsilon \oplus (-8) \end{pmatrix} = \begin{pmatrix} 5 & -10 \\ 10 & -8 \end{pmatrix} \end{aligned}$$

and

$$\begin{aligned} B \otimes A &= \begin{pmatrix} (e \otimes 5) \oplus (\varepsilon \otimes 10) & (e \otimes (-2)) \oplus (\varepsilon \otimes e) \\ (1 \times 5) \oplus ((-8) \otimes 10) & (1 \otimes (-2)) \oplus ((-8) \otimes e) \end{pmatrix} \\ &= \begin{pmatrix} 5 \oplus \varepsilon & (-2) \oplus \varepsilon \\ 6 \oplus (-2) & (-1) \oplus (-8) \end{pmatrix} = \begin{pmatrix} 5 & -2 \\ 4 & -9 \end{pmatrix}, \end{aligned}$$

which shows that multiplication in $\mathbb{R}_{\max}^{2\times2}$ is not commutative. However, it is straightforward to show that

P8 for all $A, B, C \in \mathbb{R}_{\max}^{2\times2}$, we have $(A \oplus B) \oplus C = A \oplus (B \oplus C)$;

P9 for all $A, B, C \in \mathbb{R}_{\max}^{2\times2}$, we have $(A \otimes B) \otimes C = A \otimes (B \otimes C)$;

P10 for all $A, B, C \in \mathbb{R}_{\max}^{2\times2}$, we have
$A \otimes (B \oplus C) = (A \otimes B) \oplus (A \otimes C)$ and $(A \oplus B) \otimes C = (A \otimes C) \oplus (B \otimes C)$,

where **P8** and **P9** imply that the expressions $A \oplus B \oplus C$ and $A \otimes B \otimes C$ are unambiguous [5, p. 19].

Next, let $\mathbb{T}^{2\times2} \subseteq \mathbb{R}_{\max}^{2\times2}$ be the subset containing the 2×2 *lower-triangular* matrices, i.e., those whose $(1, 2)$-entry is ε [5, p. 20]. Thus, for the matrices A and B in (4) we have $A \notin \mathbb{T}^{2\times2}$ but $B \in \mathbb{T}^{2\times2}$. The set $\mathbb{T}^{2\times2}$ is closed under matrix multiplication:

P11 for all $A, B \in \mathbb{T}^{2\times2}$, we have $A \otimes B \in \mathbb{T}^{2\times2}$.

Following [1, Sec. 3], for each $A = \left(a_{i,j}\right)_{2\times2}$ and $B = \left(b_{i,j}\right)_{2\times2}$ in $\mathbb{T}^{2\times2}$, let us define the sentence $A \leqslant B$ to mean

$$a_{1,1} \leqslant b_{1,1}\,, \qquad a_{2,1} \leqslant b_{2,1} \qquad \text{and} \qquad a_{2,2} \leqslant b_{2,2}\,,$$

and the sentence $A < B$ to mean the same with $<$ replacing $\leqslant$. It is not difficult to show that

P12 for all $A, B, C \in \mathbb{T}^{2\times2}$, if $A \leqslant B$, then $A \otimes C \leqslant B \otimes C$ and $C \otimes A \leqslant C \otimes B$, and if $A < B$, then $A \otimes C < B \otimes C$ and $C \otimes A < C \otimes B$.

Finally, for all $A, B \in \mathbb{T}^{2\times2}$, we define the sentence $A \trianglelefteq B$ to mean

$$A \otimes B \leqslant B \otimes A$$

and the sentence $A \triangleleft B$ to mean the same with $<$ replacing $\leqslant$ [1, Sec. 3]. It is not difficult to show that

P13 for all $A, B \in \mathbb{T}^{2\times2}$, we have either $A \triangleleft B$ or $B \trianglelefteq A$.

4 Solving the two-machine case

The resymbolisation of maximum and addition as $\oplus$ and $\otimes$ allows us to rewrite the two-machine flow-shop system (2) in the tidier form

$$\begin{cases} F_1(i) = p_{\sigma(i),1} \otimes F_1(i-1)\,, \\ F_2(i) = \left[\left(p_{\sigma(i),2} \otimes p_{\sigma(i),1}\right) \otimes F_1(i-1)\right] \oplus \left[p_{\sigma(i),2} \otimes F_2(i-1)\right]\,, \end{cases}$$

and subsequently in the matricial form

$$\begin{pmatrix} F_1(i) \\ F_2(i) \end{pmatrix} = \begin{pmatrix} p_{\sigma(i),1} & \varepsilon \\ p_{\sigma(i),2} \otimes p_{\sigma(i),1} & p_{\sigma(i),2} \end{pmatrix} \otimes \begin{pmatrix} F_1(i-1) \\ F_2(i-1) \end{pmatrix} = \mathbf{J}(\sigma(i)) \otimes \begin{pmatrix} F_1(i-1) \\ F_2(i-1) \end{pmatrix}, \quad (5)$$

where

$$\mathbf{J}(\sigma(i)) := \begin{pmatrix} p_{\sigma(i),1} & \varepsilon \\ p_{\sigma(i),2} \otimes p_{\sigma(i),1} & p_{\sigma(i),2} \end{pmatrix} \in \mathbb{T}^{2\times 2}$$

is referred to the i-th *job matrix* [1, Sec. 3]. Since (5) holds for every $i \in \{1,\ldots,n\}$, we have

$$\begin{aligned} \begin{pmatrix} F_1(n) \\ F_2(n) \end{pmatrix} &= \mathbf{J}(\sigma(n)) \otimes \begin{pmatrix} F_1(n-1) \\ F_2(n-1) \end{pmatrix} \\ &= \mathbf{J}(\sigma(n)) \otimes \mathbf{J}(\sigma(n-1)) \otimes \begin{pmatrix} F_1(n-2) \\ F_2(n-2) \end{pmatrix} \\ &= \cdots \\ &= \mathbf{J}(\sigma(n)) \otimes \mathbf{J}(\sigma(n-1)) \otimes \cdots \otimes \mathbf{J}(\sigma(1)) \otimes \begin{pmatrix} F_1(0) \\ F_2(0) \end{pmatrix}, \end{aligned} \quad (6)$$

where $F_1(0) = F_2(0) = e$ by (3). An optimal permutation $\sigma : \{1,\ldots,n\} \to \{1,\ldots,n\}$ is thus that for which the entries of the matrix

$$\mathbf{J}(\sigma) := \mathbf{J}(\sigma(n)) \otimes \mathbf{J}(\sigma(n-1)) \otimes \cdots \otimes \mathbf{J}(\sigma(1)) \in \mathbb{T}^{2\times 2}$$

are as small as possible, i.e., that for which

$$\mathbf{J}(\sigma) \leqslant \mathbf{J}(\tau) \quad (7)$$

for every permutation $\tau : \{1,\ldots,n\} \to \{1,\ldots,n\}$.

Now, for an optimal permutation $\sigma : \{1,\ldots,n\} \to \{1,\ldots,n\}$, suppose for a contradiction that there exists $i \in \{1,\ldots,n-1\}$ such that $\mathbf{J}(\sigma(i)) \lhd \mathbf{J}(\sigma(i+1))$. This means that

$$\mathbf{J}(\sigma(i)) \otimes \mathbf{J}(\sigma(i+1)) < \mathbf{J}(\sigma(i+1)) \otimes \mathbf{J}(\sigma(i))\,.$$

Defining a permutation $\tau : \{1,\ldots,n\} \to \{1,\ldots,n\}$ by

$$\tau(\ell) = \begin{cases} \sigma(\ell), & \text{if } \ell \notin \{i, i+1\}; \\ \sigma(i+1), & \text{if } \ell = i; \\ \sigma(i), & \text{if } \ell = i+1 \end{cases}$$

for every $\ell \in \{1,\dots,n\}$, we have, by **P12**, that

$$\begin{aligned}\mathbf{J}(\tau) &= \mathbf{J}(\sigma(n)) \otimes \cdots \otimes \mathbf{J}(\sigma(i)) \otimes \mathbf{J}(\sigma(i+1)) \otimes \cdots \otimes \mathbf{J}(\sigma(1)) \\ &< \mathbf{J}(\sigma(n)) \otimes \cdots \otimes \mathbf{J}(\sigma(i+1)) \otimes \mathbf{J}(\sigma(i)) \otimes \cdots \otimes \mathbf{J}(\sigma(1)) = \mathbf{J}(\sigma),\end{aligned}$$

contradicting (7). This proves that if $\sigma : \{1,\dots,n\} \to \{1,\dots,n\}$ is an optimal permutation, then for every $i \in \{1,\dots,n-1\}$ we have $\mathbf{J}(\sigma(i+1)) \trianglelefteqslant \mathbf{J}(\sigma(i))$. As a consequence, if $\sigma : \{1,\dots,n\} \to \{1,\dots,n\}$ is an optimal permutation, then for every $i,j \in \{1,\dots,n\}$ with $j > i$, we have $\mathbf{J}(\sigma(j)) \trianglelefteqslant \mathbf{J}(\sigma(i))$.

Conversely, suppose a permutation $\sigma : \{1,\dots,n\} \to \{1,\dots,n\}$ is such that for every $i,j \in \{1,\dots,n\}$ with $j > i$ we have $\mathbf{J}(\sigma(j)) \trianglelefteqslant \mathbf{J}(\sigma(i))$. Let us show that σ is an optimal permutation. Let $\tau_1 : \{1,\dots,n\} \to \{1,\dots,n\}$ be an optimal permutation.

(a) If $\tau_1 = \sigma$, then we are done.

(b) If $\tau_1 \neq \sigma$, then there exist two consecutive terms of the sequence $(\tau_1(n),\dots,\tau_1(1))$, say $\tau_1(k+1)$ and $\tau_1(k)$, such that $\tau_1(k)$ precedes $\tau_1(k+1)$ in the sequence $(\sigma(n),\dots,\sigma(1))$. That is, there exists $k \in \{1,\dots,n-1\}$ such that

$$\tau_1(k) = \sigma(j) \qquad \text{and} \qquad \tau_1(k+1) = \sigma(i)$$

for some $i,j \in \{1,\dots,n\}$ with $j > i$. By the assumed property of σ, this implies that $\mathbf{J}(\tau_1(k)) \trianglelefteqslant \mathbf{J}(\tau_1(k+1))$, i.e.,

$$\mathbf{J}(\tau_1(k)) \otimes \mathbf{J}(\tau_1(k+1)) \leqslant \mathbf{J}(\tau_1(k+1)) \otimes \mathbf{J}(\tau_1(k)).$$

Defining a permutation $\tau_2 : \{1,\dots,n\} \to \{1,\dots,n\}$ by

$$\tau_2(\ell) = \begin{cases} \tau_1(\ell), & \text{if } \ell \notin \{k,k+1\}; \\ \tau_1(k+1), & \text{if } \ell = k; \\ \tau_1(k), & \text{if } \ell = k+1 \end{cases}$$

for every $\ell \in \{1,\dots,n\}$, we have, by **P12**, that

$$\begin{aligned}\mathbf{J}(\tau_1) &= \mathbf{J}(\tau_1(n)) \otimes \cdots \otimes \mathbf{J}(\tau_1(k+1)) \otimes \mathbf{J}(\tau_1(k)) \otimes \cdots \otimes \mathbf{J}(\tau_1(1)) \\ &\geqslant \mathbf{J}(\tau_1(n)) \otimes \cdots \otimes \mathbf{J}(\tau_1(k)) \otimes \mathbf{J}(\tau_1(k+1)) \otimes \cdots \otimes \mathbf{J}(\tau_1(1)) = \mathbf{J}(\tau_2).\end{aligned}$$

Repeating the above with τ_2 replacing τ_1 gives a permutation τ_3 with $\mathbf{J}(\tau_2) \geqslant \mathbf{J}(\tau_3)$. Repeating again and continuing, one obtains a sequence $(\tau_1, \tau_2, \dots)$ of permutations for which there exist $t \in \mathbb{N}$ such that $\tau_t = \sigma$, and hence

$$\mathbf{J}(\tau_1) \geqslant \cdots \geqslant \mathbf{J}(\tau_t) = \mathbf{J}(\sigma),$$

implying that σ is an optimal permutation. We have therefore proved the following proposition, which is a biconditional version of [1, Lemma 2].

Proposition 1. [1, Lemma 2]
A permutation $\sigma : \{1, \dots, n\} \to \{1, \dots, n\}$ is optimal if and only if for every $i, j \in \{1, \dots, n\}$ with $j > i$ we have $\mathbf{J}(\sigma(j)) \trianglelefteqslant \mathbf{J}(\sigma(i))$.

It is possible to show that a permutation σ with the property stated in Proposition 1 always exists and can be constructed using the so-called *JG algorithm* [1, Lemma 1].

Proposition 2 (The JG algorithm)**.** [1, Lemma 1]
Let $\sigma : \{1, \dots, n\} \to \{1, \dots, n\}$ be a permutation defined as follows.

1. *Let $\mathcal{U} := \{i \in \{1, \dots, n\} : p_{i,1} \leqslant p_{i,2}\}$.*
2. *Let $\mathcal{V} := \{i \in \{1, \dots, n\} : p_{i,1} > p_{i,2}\}$.*
3. *Let $\mathcal{U}^S$ be the sequence obtained by sorting the elements $i \in \mathcal{U}$ by non-decreasing $p_{i,1}$, and for elements i having the same $p_{i,1}$, by non-increasing $p_{i,2}$.*
4. *Let $\mathcal{V}^S$ be the sequence obtained by sorting the elements $i \in \mathcal{V}$ by non-increasing $p_{i,2}$, and for elements i having the same $p_{i,1}$, by non-decreasing $p_{i,1}$.*
5. *The concatenation of $\mathcal{U}^S$ and $\mathcal{V}^S$ is the sequence $(\sigma(i))_{i=1}^n$.*

Then, for every $i, j \in \{1, \dots, n\}$ with $j > i$ we have $\mathbf{J}(\sigma(j)) \trianglelefteqslant \mathbf{J}(\sigma(i))$.

Remark 3. *The* concatenation *of two finite sequences $(a_1, \dots, a_k)$ and $(b_1, \dots, b_\ell)$ is the sequence $(a_1, \dots, a_k, b_1, \dots, b_\ell)$* [7, p. 135].

For example, let us apply the JG algorithm to our Project A in Section 1, for which the processing durations are given in Table 1. Using the $p_{i,j}$-notation to denote these durations, we obtain Table 2. Since

$$p_{1,1} > p_{1,2}\,, \qquad p_{2,1} \leqslant p_{2,2} \qquad \text{and} \qquad p_{3,1} \leqslant p_{3,2}\,,$$

we have

$$\mathcal{U} = \{2, 3\} \qquad \text{and} \qquad \mathcal{V} = \{1\}\,.$$

Ordering the elements of these sets as guided by the algorithm, we obtain the sequences

$$\mathcal{U}^S = (3, 2) \qquad \text{and} \qquad \mathcal{V}^S = (1)\,.$$

Concatenating these sequences gives $(3, 2, 1)$. Therefore, the permutation $\sigma : \{1, 2, 3\} \to \{1, 2, 3\}$ given by

$$\sigma(1) = 3, \qquad \sigma(2) = 2 \qquad \text{and} \qquad \sigma(3) = 1$$

is an optimal permutation. According to this permutation, the first, second, and third jobs to be processed are, respectively, Job 3, Job 2, and Job 1. In order words, the JG algorithm gives the last possible order in the list (1), which was indeed found to be optimal via exhaustive checking.

Machine	Job 1	Job 2	Job 3
Machine 1	$p_{1,1} = 3$	$p_{2,1} = 3$	$p_{3,1} = 1$
Machine 2	$p_{1,2} = 2$	$p_{2,2} = 3$	$p_{3,2} = 4$

Table 2: The processing duration $p_{i,j}$, in minutes, of Job i at Machine j for Project A.

Machine	Job 1	Job 2	Job 3	Job 4	Job 5	Job 6	Job 7	Job 8	Job 9	Job 10
Machine 1	2	7	22	4	14	1	19	19	12	18
Machine 2	5	9	10	20	2	10	13	8	29	14

Table 3: The processing duration, in minutes, of Job i at Machine j, used in our computer experiment.

Finally, the total time required to finish the project using the job order given by σ can be computed, either by constructing the associated time diagram (the last one in Figure 2), or by writing down the job matrices

$$\begin{aligned}
\mathbf{J}(\sigma(1)) = \mathbf{J}(3) &= \begin{pmatrix} 1 & \varepsilon \\ 5 & 4 \end{pmatrix}, \\
\mathbf{J}(\sigma(2)) = \mathbf{J}(2) &= \begin{pmatrix} 3 & \varepsilon \\ 6 & 3 \end{pmatrix}, \\
\mathbf{J}(\sigma(3)) = \mathbf{J}(1) &= \begin{pmatrix} 3 & \varepsilon \\ 5 & 2 \end{pmatrix}
\end{aligned}$$

and substituting these into (6) to obtain

$$\begin{aligned}
\begin{pmatrix} F_1(3) \\ F_2(3) \end{pmatrix} &= \mathbf{J}(\sigma(3)) \otimes \mathbf{J}(\sigma(2)) \otimes \mathbf{J}(\sigma(1)) \otimes \begin{pmatrix} F_1(0) \\ F_2(0) \end{pmatrix} \\
&= \begin{pmatrix} 3 & \varepsilon \\ 5 & 2 \end{pmatrix} \otimes \begin{pmatrix} 3 & \varepsilon \\ 6 & 3 \end{pmatrix} \otimes \begin{pmatrix} 1 & \varepsilon \\ 5 & 4 \end{pmatrix} \otimes \begin{pmatrix} e \\ e \end{pmatrix} = \begin{pmatrix} 7 \\ 10 \end{pmatrix}.
\end{aligned}$$

Therefore, using the job order given by σ, the project finishes in $F_2(3) = 10$ minutes.

To observe the efficiency of the JG algorithm, we carried out a computer experiment[4] to compare the running times of the JG algorithm and an algorithm implementing exhaustive check, using the processing durations presented in Table 3, each of which is chosen at random from the set $\{1, \ldots, 30\}$.

For each $N \in \{1, \ldots, 10\}$, we ran each of the two algorithms five times to schedule the two-machine flow-shop project involving the jobs J_1, ..., J_N in Table 3, recording the average running times (in seconds) as $t_{N,2}$ and $T_{N,2}$, for the JG and exhaustive check algorithms, respectively. These average running times are plotted in Figure 3. We can

[4]This experiment was carried out using ASUS ROG Strix 5 GL503GE with Intel Core i7-8750H processor and 8 GB RAM.

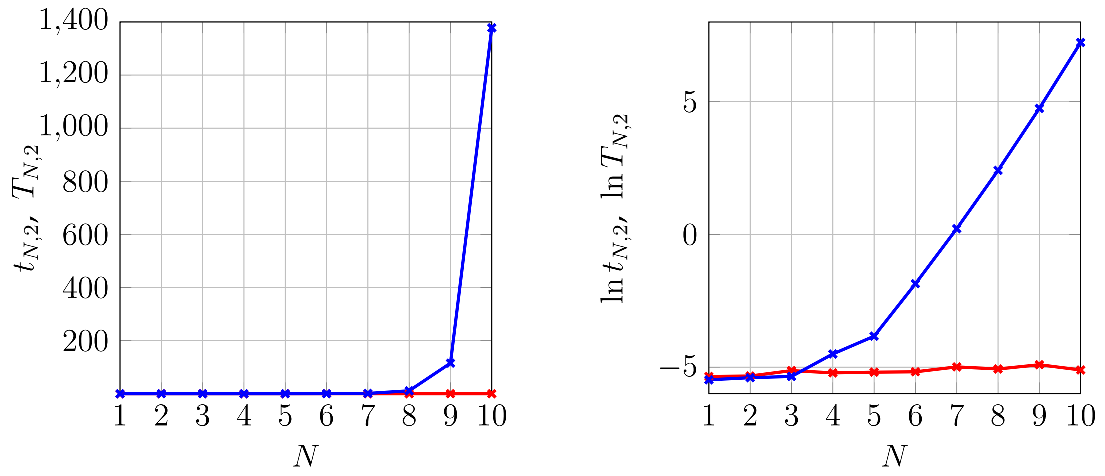

Figure 3: A plot of $t_{N,2}$ (red) and $T_{N,2}$ (blue) as functions of N (left), and a plot of $\ln t_{N,2}$ (red) dan $\ln T_{N,2}$ (blue) as functions of N (right).

Machine	Job 1	Job 2	Job 3
Machine 1	5	3	4
Machine 2	1	4	2
Machine 3	4	4	5

Table 4: The processing duration, in minutes, of each job at each machine for Project B.

see that the exhaustive-check algorithm is faster than the JG algorithm only in the cases where there are only very few jobs: $N \in \{1, 2, 3\}$. As the number of jobs N increases, the JG algorithm becomes significantly faster.

5 The multi-machine case

Let us now say a few words on the three-machine case, and subsequently on the general m-machine case. Consider a flow-shop project, call it Project B, with three jobs, Job 1, Job 2 and Job 3, each of which are to be processed at three machines, Machine 1, Machine 2 and Machine 3, where the processing times are given in Table 4. The time diagram in Figure 4 shows that processing these three jobs in order requires a total time of 21 minutes.

If the jobs do not have to be processed in order, we can certainly perform exhaustive checking, to obtain that an optimal job order is 3, 2, 1, which gives a total time of 19 minutes. However, we can also try implementing a classic mathematical problem-solving strategy: reducing the problem into one which we already know how to solve [9, p. 149–150]. Specifically, we try reduce the three-machine project in Table 4 into a two-machine project, by defining a new Machine 1′ as the composition of Ma-

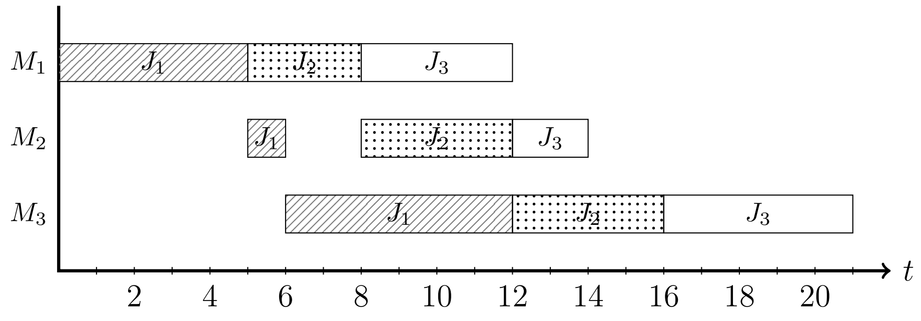

Figure 4: A time diagram for Project B, showing the job being processed at each machine at any given time t, in the case of the three jobs being processed in order.

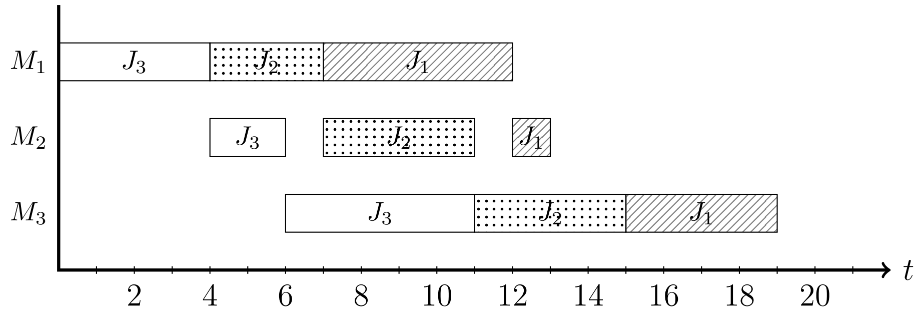

Figure 5: A time diagram for Project B, showing the job being processed at each machine at any given time t, using the job order 3, 2, 1.

chine 1 and Machine 2, and a new Machine 2′ as the composition of Machine 2 and Machine 3. At each composite machine, we define the processing duration of each job to be the product[5] of the job's processing durations at the composed machines. This reduces the three-machine project in Table 4 into the two-machine project in Table 5. Applying the JG algorithm to this two-machine project gives the optimal job order mentioned above: 3, 2, 1. The corresponding total time then can be determined by constructing the associated time diagram (Figure 5), i.e., 19 minutes.[6]

Machine	Job 1	Job 2	Job 3
Machine 1′ (Composition of Machines 1 and 2)	$5 \otimes 1 = 6$	$3 \otimes 4 = 7$	$4 \otimes 2 = 6$
Machine 2′ (Composition of Machines 2 and 3)	$1 \otimes 4 = 5$	$4 \otimes 4 = 8$	$2 \otimes 5 = 7$

Table 5: The processing duration, in minutes, of each job at each composite machine for Project B.

[5]With respect to the operation $\otimes$, i.e., the sum.

[6]It is also possible to determine the total time using job matrices as in the two-machine case. In the three-machine case, the job matrices are of size 3×3 [1, Subsec. 4.5].

Machine	Job 1	Job 2	Job 3
Machine 1	3	3	4
Machine 2	5	2	4
Machine 3	5	4	5

Table 6: The processing duration, in minutes, of each job at each machine for Project C.

Machine	Job 1	Job 2	Job 3
Machine $1'$ (Composition of Machines 1 and 2)	$3 \otimes 5 = 8$	$3 \otimes 2 = 5$	$4 \otimes 4 = 8$
Machine $2'$ (Composition of Machines 2 and 3)	$5 \otimes 5 = 10$	$2 \otimes 4 = 6$	$4 \otimes 5 = 9$

Table 7: The processing duration, in minutes, of each job at each composite machine for Project C.

Is this reduction method successful for all three-machine projects? To answer this, consider another three-machine project, call it Project C, given by Table 6. Let us implement the same idea, namely, defining a new Machine $1'$ as the composition of Machine 1 and Machine 2, and a new Machine $2'$ as the composition of Machine 2 and Machine 3. This gives the two-machine project in Table 7. Applying the JG algorithm to this project gives the job order 2, 1, 3, with a total time of 20 minutes. However, by exhaustive checking, one finds that the order 1, 2, 3 gives a shorter total time of 19 minutes.

Therefore, the reduction method is not always successful. Under what conditions is it guaranteed to be successful? A sufficient condition is given by the following proposition [2, Theorem 1 in sec. 8.3].

Proposition 4. [2, Theorem 1 in Sec. 8.3]
Consider a three-machine flow-shop project, with $p_{i,j}$ denoting the processing duration of Job i at Machine j, for every $i \in \{1, \ldots, n\}$ and $j \in \{1, 2, 3\}$. If either

$$\min\{p_{1,1}, \ldots, p_{n,1}\} \geqslant \max\{p_{1,2}, \ldots, p_{n,2}\} \tag{8}$$

$$\text{or} \qquad \min\{p_{1,3}, \ldots, p_{n,3}\} \geqslant \max\{p_{1,2}, \ldots, p_{n,2}\}, \tag{9}$$

then the JG algorithm applied to the flow-shop project consisting of the two composite machines

Machine $1'$, defined as the composition of Machine 1 and Machine 2, and

Machine $2'$, defined as the composition of Machine 2 and Machine 3,

where, at each composite machine, the processing duration of each job is defined to be the product of the job's processing durations at the composed machines, produces a job order which is optimal for the original three-machine project.

Notice that the condition (8) means that every processing duration in Machine 1 must be longer than every processing duration in Machine 2, while the condition (9) means that every processing duration in Machine 3 must be longer than every processing duration in Machine 2. Notice also that Project B (Table 4) satisfies (9), whereas Project C (Table 6) satisfies neither (8) nor (9).

Conveniently, Proposition 4 is generalisable to the m-machine case, for all $m \geqslant 3$, providing a sufficient condition for which an m-machine project can be scheduled by reduction to a two-machine project involving Machine 1′, defined as the composition of Machine 1, …, Machine $m-1$, and Machine 2′, defined as the composition of Machine 2, …, Machine m [8, Sec. 10.3].

Proposition 5. [8, Sec. 10.3]
Consider an m-machine flow-shop project, with $p_{i,j}$ denoting the processing duration of Job i at Machine j, for every $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, m\}$. If either

$$\begin{array}{lll} & \min\{p_{1,1}, \ldots, p_{n,1}\} \geqslant \max\{p_{1,k}, \ldots, p_{n,k}\} \quad \textit{for every } k \in \{2, \ldots, m-1\} & (10) \\ \textit{or} & \min\{p_{1,m}, \ldots, p_{n,m}\} \geqslant \max\{p_{1,k}, \ldots, p_{n,k}\} \quad \textit{for every } k \in \{2, \ldots, m-1\}, & (11) \end{array}$$

then the JG algorithm applied to the flow-shop project consisting of the two composite machines

Machine 1′, defined as the composition of Machine 1, …, Machine $m-1$, and

Machine 2′, defined as the composition of Machine 2, …, Machine m,

where, at each composite machine, the processing duration of each job is defined to be the product of the job's processing durations at the composed machines, produces a job order which is optimal for the original m-machine project.

For example, consider a four-machine flow-shop project, call it Project D, involving Job 1, Job 2, Job 3, and Machine 1, Machine 2, Machine 3, Machine 4, where the processing times are given in Table 8.

Machine	Job 1	Job 2	Job 3
Machine 1	5	3	4
Machine 2	1	4	2
Machine 3	3	4	4
Machine 4	5	4	6

Table 8: The processing duration, in minutes, of each job at each machine for Project D.

Notice that the project given by Table 8 satisfies the condition (11), namely,

$$\begin{array}{ll} & \min\{p_{1,4}, p_{2,4}, p_{3,4}\} \geqslant \max\{p_{1,2}, p_{2,2}, p_{3,2}\} \\ \text{and} & \min\{p_{1,4}, p_{2,4}, p_{3,4}\} \geqslant \max\{p_{1,3}, p_{2,3}, p_{3,3}\}. \end{array}$$

Machine	Job 1	Job 2	Job 3
Machine 1′ (Composition of Machines 1–3)	$5 \otimes 1 \otimes 3 = 9$	$3 \otimes 4 \otimes 4 = 11$	$4 \otimes 2 \otimes 4 = 10$
Machine 2′ (Composition of Machines 2–4)	$1 \otimes 3 \otimes 5 = 9$	$4 \otimes 4 \otimes 4 = 12$	$2 \otimes 4 \otimes 6 = 12$

Table 9: The processing duration, in minutes, of each job at each composite machine for Project D.

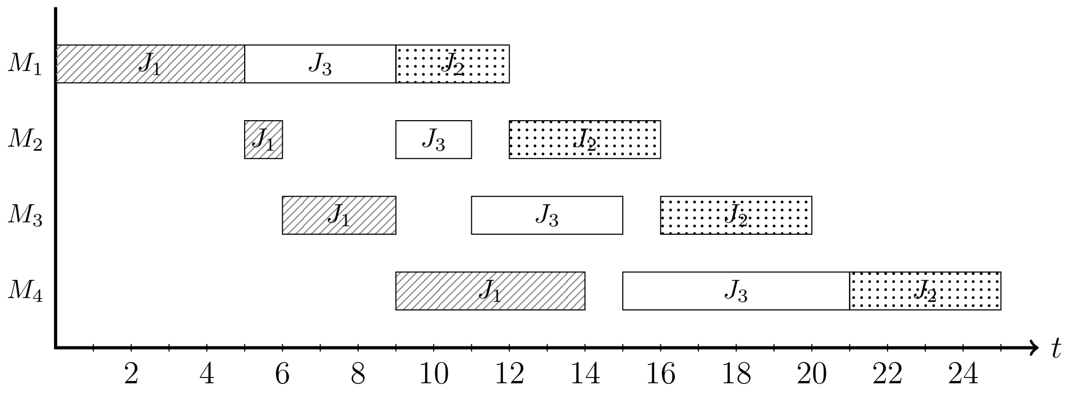

Figure 6: A time diagram for Project D, showing the job being processed at each machine at any given time t, using the job order 1, 3, 2.

Therefore, an optimal job order can be determined by Proposition 5. By defining Machine 1′ as the composition of Machines 1–3, and Machine 2′ as the composition of Machines 2–4, we obtain the two-machine project given by Table 9.

The JG algorithm then gives the optimal job order 1, 3, 2. Constructing the associated time diagram (Figure 6), one finds that this order gives a total time of 25 minutes.

Unlike the JG algorithm (Proposition 2) which applies to all two-machine projects, the method for three and more machines provided by Propositions 4 and 5 apply only under specific sufficient conditions. Indeed, even the scheduling of three-machine flow-shop projects is known to belong to a class of hard problems referred to as *NP* problems: problems for which no polynomial-time solution algorithm has been found. In fact, it is an *NP-complete* problem [1, Proposition 7]: a problem for which the finding of a polynomial-time solution algorithm implies the finding of polynomial-time solution algorithms for all NP problems. For three-machine projects, several alternative methods which apply under weaker sufficient conditions are discussed in [2, Chap. 8], while a general algorithm — called a *branch-and-bound* algorithm — is discussed in [1, Sec. 4]. For m-machine projects, a general algorithm is discussed in [4].

Acknowledgements

This article is a condensed version of the undergraduate thesis written by the first author, supervised by the second and third authors. The first author would like to thank his examiners, Liem Chin and Agus Sukmana, for their valuable comments.

References

[1] J.-L. Bouquard, C. Lenté and J.-C. Billaut, Application of an optimization problem in max-plus algebra to scheduling problems, *Discrete Applied Mathematics* **154** (2006), 2064–2079.

[2] R. Bellman, A.O. Esogbue and I. Nabeshima, *Mathematical Aspects of Scheduling and Applications*, Pergamon Press, Oxford, 1982.

[3] J.B. Fraleigh and N.E. Brand, *A First Course in Abstract Algebra*, 8th edition, Pearson, New Jersey, 2021.

[4] R.D. Smith and R.A. Dudek, A general algorithm for solution of the n-job, M-machine sequencing problem of the flow shop, *Operations Research* **15** (1967), 71–82.

[5] B. Heidergott, G.J. Olsder and J. van der Woude, *Max Plus at Work: Modeling and Analysis of Synchronized Systems: A Course on Max-Plus Algebra and Its Applications*, Princeton University Press, Princeton, 2006.

[6] S.M. Johnson, Optimal two- and three-stage production schedule with setup times included, *Naval Research Logistics Quarterly* **1** (1954), 61–68.

[7] R. Johnsonbaugh, *Discrete Mathematics*, 8th edition, Pearson, New Jersey, 2018.

[8] H.S. Kasana and K.D. Kumar, *Introductory Operations Research: Theory and Applications*, Springer, Berlin, 2004.

[9] R. Nickerson, *Mathematical Reasoning*, Psychology Press, New York, 2010.