CAUCHY – Jurnal Matematika Murni dan Aplikasi
Volume 11 (2) (2026), Pages 2138–2159
p-ISSN: 2086-0382; e-ISSN: 2477-3344

Determining Logistics Route Capacity Using Maximal Solutions of Two-Sided Max-Min Linear System with Python Implementation

Zakia Nur Ramadhani Putri*, Ari Suparwanto, and Sutopo

Department of Mathematics, Faculty of Mathematics and Natural Sciences, Universitas Gadjah Mada, Indonesia

Abstract

Max-min algebra, equipped with maximum and minimum operations, provides a natural framework for modeling systems with bottleneck constraints, such as logistics and transportation networks. An algorithm for solving two-sided linear systems of the form $A \otimes x = B \otimes x$ in max-min algebra has been proposed, for which the maximal solution can be constructed explicitly for the $1 \times n$ case under three conditions—unconstrained, with a given upper bound, and with a given lower bound—and generalized to the $m \times n$ case, terminating after at most m iterations and producing a unique maximal solution with complexity $O(m^2 n)$. However, no implementation has been made available, which limits its use for larger, practically sized problems: the algorithm may require up to m computational cycles, so the computational burden grows with the matrix size—particularly the number of rows—and manual calculation becomes more prone to error. This paper addresses this gap by providing a Python implementation of the algorithm, along with a numerical application to a logistics distribution network. The Python implementation makes the algorithm accessible to practitioners, and the logistics example demonstrates how the model can be used to determine optimal road capacities. This work provides a practical computational tool for solving capacity planning problems in logistics and transportation.

Keywords: max-min algebra; two-sided linear system; maximal solution; algorithm; logistics network.

1. Introduction

Max-min algebra is set $\bar{\mathbb{R}} = \mathbb{R} \cup \{-\infty, +\infty\}$ equipped with two binary operations maximum ($\oplus$) and minimum ($\otimes$). This structure forms a commutative idempotent semiring and provides a natural mathematical framework for modeling systems that involve bottleneck constraints, where the overall capacity of a system is determined by its smallest component. Such problems arise in various fields, including logistics and transportation networks. In these contexts, the minimum operation captures the bottleneck principle, the capacity of a route consisting of multiple segments is limited by the segment with the smallest capacity, while the maximum operation represents the optimal choice among available alternatives. The theoretical foundation of max-min algebra and its applications have been extensively studied since the pioneering work of Cunningham-Green and Raymond on minimax algebra [1], and later developed by Gondran and Minoux in the context of dioids and semirings [2].

*Corresponding author. E-mail: zakiaputri119101@gmail.com

Received: June 26, 2026 Revised: August 21, 2026 Accepted: September 15, 2026
DOI: https://doi.org/10.18860/cauchy.v11i2.44534

The study of linear systems in max-min algebra has received much attention in recent decades. Sanchez [3] pioneered the resolution of composite fuzzy relation equations, which are closely related to max-min systems, and later applied these results to medical diagnosis [4]. Santos [5] investigated maximin automata, which indirectly use maximum and minimum as composition operations. Butkovic et al. [6] studied strong linear independence and regularity of matrices in bottleneck algebra, providing foundational results for matrix theory in this structure. Fiedler et al. [7] provided a comprehensive treatment of one-sided systems of the form $A \otimes x = b$, including the discussion of the greatest subsolution and various applications in production scheduling and transportation networks. For two-sided systems of the form $A \otimes x = B \otimes x$, Gavalec and Zimmermann [8] proposed a polynomial algorithm for solving such systems, establishing the theoretical feasibility of obtaining solutions in polynomial time. Subsequently, Krbálek and Pozdílková [9] presented a simpler and more efficient iterative method, which forms the basis of the algorithm examined in this paper.

More recently, researchers have explored several extensions of max-min linear systems, such as optimization with two-sided constraints [10, 11], robustness of interval matrices [12, 13], and methods for max-plus and tropical algebras [14]. Mufid et al. [15] developed a method for max-min ω systems, which generalizes both max-plus and min-plus systems. While these works focus on specific extensions, the basic algorithm for finding maximal solutions of two-sided max-min systems is still an important building block for applications.

As noted above, Gavalec and Zimmermann [8] proposed a polynomial algorithm for two-sided max-min systems, and Krbálek and Pozdílková [9] later proposed the simpler iterative algorithm adopted in this paper; both establish theoretical properties but neither provides a computational implementation. The more recent extensions discussed above [10, 11, 14] likewise remain purely theoretical. To the best of our knowledge, no publicly available implementation exists for computing maximal solutions of two-sided max-min systems, which limits their practical adoption in applied settings such as logistics and transportation planning, especially as problem size grows and the number of computational cycles required by the algorithm increases.

This paper addresses this gap. Its main contribution is a Python implementation of the algorithm proposed by [9], which makes the method accessible for larger, practically sized networks. To demonstrate its practical use, we present a numerical case study on a logistics distribution network connecting three industrial areas to a distribution center through three transit points, showing how the maximal solution yields balanced two-way road capacities. In presenting the theoretical background needed for the implementation, we also establish a new uniqueness result for the maximal solution produced by the algorithm. The rest of this paper is organized as follows. Section 2 presents the mathematical setting and methodology. Section 3 contains the main results, including the derivation of the maximal solution for the $1 \times n$ and $m \times n$ cases, Algorithm 1 with its proof of correctness, and a numerical example. Section 4 concludes the paper. The Python implementation of Algorithm 1 is provided in Listing 1 (Appendix A).

2. Methods

This study adopts a theoretical–computational framework. The theoretical part establishes the mathematical background and the correctness of the algorithm, while the computational part provides a Python implementation and a numerical example to demonstrate the practical applicability of the model. The study is conducted within the framework of max-min algebra, defined as the structure $(\overline{\mathbb{R}}, \oplus, \otimes)$, where $\overline{\mathbb{R}} = \mathbb{R} \cup \{-\infty, +\infty\}$, equipped with two binary operations $a \oplus b = \max(a, b)$ and $a \otimes b = \min(a, b)$, for any $a, b \in \overline{\mathbb{R}}$. The operation $\oplus$ is called maximum, and $\otimes$ is called minimum. The structure $(\overline{\mathbb{R}}, \oplus, \otimes)$ forms a commutative idempotent semiring, with $\varepsilon = -\infty$ as the neutral element for $\oplus$ and $\varepsilon' = +\infty$ as the identity element for $\otimes$. The concept of semiring follows from the works of Baccelli et al. [16], Gondran and Minoux [2], and Subiono [17]. We consider matrices $A, B \in \overline{\mathbb{R}}^{m \times n}$ and vectors $x \in \overline{\mathbb{R}}^{n}$. The matrix-vector

product $A \otimes x$ is defined componentwise as [18]:

$$(A \otimes x)_i = \bigoplus_{j=1}^{n} (a_{ij} \otimes x_j) = \max_{j=1,\dots,n} \big(\min(a_{ij}, x_j)\big), \quad i = 1, \dots, m.$$

The relation "$\leq$" defined on $\overline{\mathbb{R}}$ by $a \leq b \iff a \oplus b = b$ is a partial order on $\overline{\mathbb{R}}$. Moreover, "$\leq$" is a total order on $\overline{\mathbb{R}}$ [18]. The relation "$\leq$" is then extended componentwise to vectors and matrices. The problem is the two-sided system $A \otimes x = B \otimes x$, and the goal is to determine its maximal solution. A solution $\bar{x}$ of $A \otimes x = B \otimes x$ is called maximal if, for every solution $\tilde{x}$ of the same system, $\bar{x} \leq \tilde{x}$ implies $\tilde{x}{=}\bar{x}$ [16].

The algorithm used in this paper was proposed by Krbálek and Pozdílková [9]. It is an iterative method that finds the maximal solution of $A \otimes x = B \otimes x$. Starting from an upper bound, the algorithm repeatedly reduces the solution vector until a solution is found. The algorithm terminates after at most m iterations and produces a unique maximal solution. The full algorithm, along with its proof of correctness, is presented in Section 3.

To make the algorithm usable in practice, we implement the algorithm in Python 3.9. The implementation utilizes the NumPy library for efficient array manipulation, which is widely available in the Python ecosystem. The code reads matrices A and B as input and iteratively computes the maximal solution following the algorithm described in Section 3. The full implementation is provided in Appendix A.

To show how the model can be applied, we construct a numerical example from a logistics distribution network. It involves three industrial areas and three transit points, where the system $A \otimes x = B \otimes x$ is used to determine the optimal capacities of access roads from the transit points to the main distribution center. The data used in this example are simulated and serve only for illustrative purposes; they are not based on actual field measurements. The aim of this example is to show the applicability of the model, rather than to serve as a real-world case study.

3. Results and Discussion

This section presents the theoretical foundations needed for the implementation, followed by a numerical application. We begin with the $1 \times n$ case, then generalize to $m \times n$ matrices. The theoretical results are then used to develop the Python implementation and the logistics application. For the $1 \times n$ case, we consider three scenarios: without constraints, with a lower bound, and with an upper bound.

3.1. Maximal Solutions of $A \otimes x = B \otimes x$ for $1 \times n$ Matrices A, B

This subsection discusses the maximal solution of the system

$$A \otimes x = B \otimes x \tag{1}$$

where A, B are of size $1 \times n$, and $x \in \overline{\mathbb{R}}^n$ without constraints, with an upper bound, and with a lower bound. System (1) can be written as

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \end{pmatrix} \otimes \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1n} \end{pmatrix} \otimes \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$$

or equivalently,

$$(a_{11} \otimes x_1) \oplus (a_{12} \otimes x_2) \oplus \cdots \oplus (a_{1n} \otimes x_n) = (b_{11} \otimes x_1) \oplus (b_{12} \otimes x_2) \oplus \cdots \oplus (b_{1n} \otimes x_n).$$

In other words, we seek the maximal vector x satisfying

$$\bigoplus_{j=1}^{n} (a_{1j} \otimes x_j) = \bigoplus_{j=1}^{n} (b_{1j} \otimes x_j).$$

3.1.1. Maximal Solution of $A \otimes x = B \otimes x$ for $1 \times n$ Matrices A, B with $x \in \overline{\mathbb{R}}^n$ Without Constraints

This subsection discusses the maximal solution $\bar{x} = (\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n)$ of System (1) where A, B are of size $1 \times n$ and $\bar{x}$ is given without any constraints. Let $N = \{1, 2, \ldots, n\}$, and define

$$a = \max_{j=1,\ldots,n}(a_{11}, \ldots, a_{1n}) = \bigoplus_{j=1}^{n} a_{1j}, \qquad b = \max_{j=1,\ldots,n}(b_{11}, \ldots, b_{1n}) = \bigoplus_{j=1}^{n} b_{1j}.$$

There are three possibilities for the values of a and b: $a = b$, $a < b$, or $a > b$.

Lemma 1 and Lemma 2 discuss the maximal solution for the cases $a = b$ and $a > b$, respectively. The case $a < b$ is analogous to the case $a > b$.

Lemma 1. *If $a = b$, then $\bar{x} = (\infty, \infty, \ldots, \infty)^T$ is the maximal solution of the linear system $A \otimes x = B \otimes x$.*

Proof. Substituting $\bar{x}$ into $A \otimes x = B \otimes x$, we obtain

$$A \otimes \bar{x} = \bigoplus_{j=1}^{n}(a_{1j} \otimes \infty) = \bigoplus_{j=1}^{n} a_{1j} = a \quad \text{and} \quad B \otimes \bar{x} = \bigoplus_{j=1}^{n}(b_{1j} \otimes \infty) = \bigoplus_{j=1}^{n} b_{1j} = b.$$

Since $a = b$, we have $A \otimes \bar{x} = B \otimes \bar{x}$, which means $\bar{x}$ is a solution of $A \otimes x = B \otimes x$. Clearly, $\bar{x}$ is the maximal solution. □

Lemma 2. *Let $N_1 = \{j \in N \mid a_{1j} > b\}$. If $a > b$, then $\bar{x} = (\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n)^T$ is the maximal solution of $A \otimes x = B \otimes x$, where*

$$\bar{x}_j = \begin{cases} b, & j \in N_1, \\ \infty, & \text{otherwise}, \end{cases}$$

for $j = 1, 2, \ldots, n$.

Proof. Given $N_1 = \{j \in N \mid a_{1j} > b\}$. Since $a > b$, we have $\bigoplus_{j=1}^{n} a_{1j} > b$, so there exists $j \in N$ such that $a_{1j} > b$; hence $N_1 \neq \emptyset$. Substituting $\bar{x}$ into $A \otimes x = B \otimes x$, we obtain

$$B \otimes \bar{x} = \bigoplus_{j=1}^{n}(b_{1j} \otimes \bar{x}_j) = \left(\bigoplus_{j \in N_1}(b_{1j} \otimes b)\right) \oplus \left(\bigoplus_{j \in N \setminus N_1}(b_{1j} \otimes \infty)\right).$$

Since $b_{1j} \otimes b = \min(b_{1j}, b) \leq b$ for every $j \in N_1$, we have $\bigoplus_{j \in N_1}(b_{1j} \otimes b) \leq b$. Moreover, since $b = \bigoplus_{j=1}^{n} b_{1j}$, we obtain $\bigoplus_{j \in N \setminus N_1}(b_{1j} \otimes \infty) = \bigoplus_{j \in N \setminus N_1} b_{1j} \leq \bigoplus_{j=1}^{n} b_{1j} = b$. Hence,

$$B \otimes \bar{x} \leq b. \tag{2}$$

On the other hand, there exists $k \in N$ such that $b_{1k} = b$. Since $\bar{x}_k = b$ if $k \in N_1$ and $\bar{x}_k = \infty$ otherwise, in either case $\bar{x}_k \geq b$, so $b_{1k} \otimes \bar{x}_k = min(b, \bar{x}_k) = b$. Hence,

$$B \otimes \bar{x} = \bigoplus_{j=1}^{n}(b_{1j} \otimes \bar{x}_j) \geq b_{1k} \otimes \bar{x}_k = b \otimes \bar{x}_k = b. \tag{3}$$

From Eqs. (2) and (3), we obtain $B \otimes \bar{x} = b$. Next,

$$A \otimes \bar{x} = \bigoplus_{j=1}^{n}(a_{1j} \otimes \bar{x}_j) = \left(\bigoplus_{j \in N_1}(a_{1j} \otimes b)\right) \oplus \left(\bigoplus_{j \in N \setminus N_1}(a_{1j} \otimes \infty)\right).$$

For $j \in N_1$, we have $a_{1j} > b$, so $a_{1j} \otimes b = \min(a_{1j}, b) = b$; since $N_1 \neq \emptyset$, it follows that $\bigoplus_{j\in N_1}(a_{1j} \otimes b) = b$. For $j \in N \setminus N_1$, we have $a_{1j} \leq b$ by the definition of N_1, so $a_{1j} \otimes \infty = a_{1j} \leq b$, and thus $\bigoplus_{j\in N\setminus N_1}(a_{1j} \otimes \infty) \leq b$. Therefore

$$A \otimes \bar{x} = b \oplus \left(\bigoplus_{j\in N\setminus N_1} a_{1j} \right) = b.$$

Thus, $A \otimes \bar{x} = B \otimes \bar{x}$, so $\bar{x} = (\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n)^T$ is a solution of $A \otimes x = B \otimes x$. We now show that $\bar{x}$ is the maximal solution by contradiction. Let x' be a solution. Suppose $\bar{x}$ is not the maximal solution. Then there exists $k \in N$ such that $\bar{x}_k < x'_k$. Clearly $\bar{x}_k \neq \infty$, which means $k \in N_1$ and $a_{1k} > b = \bar{x}_k$. Since $b = \bar{x}_k < x'_k$, we have

$$A \otimes x' = \bigoplus_{j=1}^{n}(a_{1j} \otimes x'_j) \geq a_{1k} \otimes x'_k > b.$$

On the other hand,

$$B \otimes x' = \bigoplus_{j=1}^{n}(b_{1j} \otimes x'_j) = \left(\bigoplus_{j\in N_1} (b_{1j} \otimes b) \right) \oplus \left(\bigoplus_{j\in N\setminus N_1} (b_{1j} \otimes \infty) \right) = \bigoplus_{j=1}^{n} b_{1j} \leq b.$$

Hence $A \otimes x' \neq B \otimes x'$, so x' is not a solution. This contradiction shows that $\bar{x}$ must be the maximal solution. □

3.1.2. Maximal Solution of $A \otimes x = B \otimes x$ for $1 \times n$ Matrices A, B with $x \in \overline{\mathbb{R}}^n$ Given a Lower Bound

This subsection discusses the maximal solution $\bar{x} = (\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n)$ of System (1) where A, B are of size $1 \times n$ with the constraint

$$l = (l_1, l_2, \ldots, l_n)^T \leq (\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n)^T = \bar{x}.$$

Lemma 3. *Let $\bar{x} = (\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n)^T$ be the maximal solution of $A \otimes x = B \otimes x$ without constraints.*

1. *The maximal solution of $A \otimes x = B \otimes x$ with lower bound $l = (l_1, l_2, \ldots, l_n)^T$ is $\bar{x}$ if $l \leq \bar{x}$.*
2. *The system $A \otimes x = B \otimes x$ with lower bound $l = (l_1, l_2, \ldots, l_n)^T$ has no solution if $l \not\leq \bar{x}$.*

Proof. Let $\bar{x}$ be the maximal solution of $A \otimes x = B \otimes x$ without constraints.

1. If $l \leq \bar{x}$, then $\bar{x}$ is a solution of $A \otimes x = B \otimes x$ with lower bound l. It remains to show that $\bar{x}$ is the maximal solution. Let x' be any solution satisfying $l \leq x'$. Suppose $\bar{x} \leq x'$, since x' is a solution without constraints and $\bar{x}$ is the maximal solution without constraints, we must have $x' \leq \bar{x}$. Thus $x' = \bar{x}$, which means $\bar{x}$ is the maximal solution of $A \otimes x = B \otimes x$ with lower bound l.
2. For $l \not\leq \bar{x}$, we prove by contradiction. Since $l \not\leq \bar{x}$, there exists $k \in N$ such that $\bar{x}_k \leq l_k$. Suppose there exists a solution x' satisfying $l \leq x'$. Then x' is also a solution of $A \otimes x = B \otimes x$ without constraints. Since $\bar{x}$ is the maximal solution without constraints, we must have $x' \leq \bar{x}$, so $x'_k \leq \bar{x}_k$ for each $k \in N$. Consequently, $l \leq x' \leq \bar{x}$, which implies $l \leq \bar{x}$. This contradicts $l \not\leq \bar{x}$. Therefore, $A \otimes x = B \otimes x$ with lower bound l has no solution. □

3.1.3. Maximal Solution of $A \otimes x = B \otimes x$ for $1 \times n$ Matrices A, B with $x \in \overline{\mathbb{R}}^n$ Given an Upper Bound

This subsection discusses the maximal solution $\bar{x} = (\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n)$ of System (1) where $A = (a_{1j}), B = (b_{1j})$ are of size $1 \times n$ with the constraint

$$\bar{x} = (\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n)^T \leq u = (u_1, u_2, \ldots, u_n)^T.$$

Let $a = A \otimes u$ and $b = B \otimes u$. There are three possibilities for a and b: $a = b$, $a < b$, or $a > b$. Lemma 4 and Lemma 5 discuss the maximal solution for the cases $a = b$ and $a > b$, respectively. The case $a < b$ is analogous to the case $a > b$.

Lemma 4. *If $a = b$, then $u = (u_1, u_2, \ldots, u_n)^T$ is the maximal solution of $A \otimes x = B \otimes x$ with upper bound u.*

Proof. Substituting u into $A \otimes x = B \otimes x$, we obtain

$$A \otimes u = \bigoplus_{j=1}^{n} (a_{1j} \otimes u_j) = a \qquad \text{and} \qquad B \otimes u = \bigoplus_{j=1}^{n} (b_{1j} \otimes u_j) = b.$$

Since $a = b$, u is a solution of $A \otimes x = B \otimes x$, and because u is the upper bound, it is the maximal solution. □

Lemma 5. *Let $N_2 = \{j \in N \mid a_{1j} \otimes u_j > b\}$. If $a > b$, then the maximal solution of $A \otimes x = B \otimes x$ with upper bound u is $\bar{x} = (\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n)^T$, where*

$$\bar{x}_j := \begin{cases} b & j \in N_2, \\ u_j & \text{otherwise}, \end{cases}$$

for $j = 1, \ldots, n$.

Proof. Since $a > b$, we have $\bigoplus_{j=1}^{n} a_{1j} \otimes u_j > b$, which means there exists $j \in N$ such that $a_{1j} \otimes u_j > b$. Hence $N_2 \neq \emptyset$. We consider $\bar{x}_j$ in two cases, for $j \in N_2$, then $\bar{x}_j = b < a_{1j} \otimes u_j \leq u_j$, and for $j \in N \setminus N_2$, then $\bar{x}_j = u_j \leq u_j$. Thus, $\bar{x}_j \leq u_j$ for all $j \in N$. Substituting $\bar{x}$ into $A \otimes x = B \otimes x$, we obtain

$$B \otimes \bar{x} = \bigoplus_{j=1}^{n} (b_{1j} \otimes \bar{x}_j) \leq \bigoplus_{j=1}^{n} (b_{1j} \otimes u_j) = b. \tag{4}$$

On the other hand, since $b = \bigoplus_{j \in N} (b_{1j} \otimes u_j)$, there exists $k \in N$ such that $b_{1k} \otimes u_k = b$. We obtain

$$B \otimes \bar{x} = \bigoplus_{j=1}^{n} (b_{1j} \otimes \bar{x}_j) \geq b_{1k} \otimes \bar{x}_k = b. \tag{5}$$

From Eqs. (4) and (5), we get $B \otimes \bar{x} = b$. Next, since $\bigoplus_{j \in N_2} (a_{1j} \otimes b) = b$ and $\bigoplus_{j \in N \setminus N_2} (a_{1j} \otimes u_j) \leq b$, then we obtain

$$A \otimes \bar{x} = \bigoplus_{j=1}^{n} (a_{1j} \otimes \bar{x}_j) = \left(\bigoplus_{j \in N_2} (a_{1j} \otimes b) \right) \oplus \left(\bigoplus_{j \in N \setminus N_2} (a_{1j} \otimes u_j) \right) = b.$$

We have $A \otimes \bar{x} = B \otimes \bar{x}$, so $\bar{x}$ is a solution. It remains to show that $\bar{x}$ is the maximal solution. We prove by contradiction. Let x' be a solution. Suppose $\bar{x}$ is not the maximal solution. Then there exists $k \in N$ such that $\bar{x}_k < x'_k$. Since u is the upper bound, we must

have $x'_k \leq u_k$, so $\bar{x}_k \neq u_k$. Thus $k \in N_2$, which implies $a_{1k} \otimes u_k > b = \bar{x}_k$, so $a_{1k} > b$ and $b = \bar{x}_k < x'_k$. Substituting x' into the system, we obtain

$$A \otimes x' = \bigoplus_{j=1}^{n} (a_{1j} \otimes x'_j) \geq a_{1k} \otimes x'_k > b.$$

On the other hand,

$$B \otimes x' = \bigoplus_{j=1}^{n} (b_{1j} \otimes x'_j) \leq \bigoplus_{j=1}^{n} (b_{1j} \otimes u_j) = b.$$

Thus $A \otimes x' \neq B \otimes x'$. Hence x' is not a solution. Therefore, $\bar{x}$ must be the maximal solution. □

The following example illustrates the solution of $A \otimes x = B \otimes x$ for $A, B \in \overline{\mathbb{R}}^{1\times n}$ with the constraint $x \leq u$.

Example 1. Consider the system $A \otimes x = B \otimes x$ with $x \leq u$, where $A = \begin{pmatrix} 10 & 8 & 10 \end{pmatrix}$, $B = \begin{pmatrix} 9 & 9 & 9 \end{pmatrix}$, and $u = \begin{pmatrix} 11 & 11 & 11 \end{pmatrix}^T$.
We compute

$$a = A \otimes u = \begin{pmatrix} 10 & 8 & 10 \end{pmatrix} \otimes \begin{pmatrix} 11 & 11 & 11 \end{pmatrix}^T = 10, \quad \textit{and}$$
$$b = B \otimes u = \begin{pmatrix} 9 & 9 & 9 \end{pmatrix} \otimes \begin{pmatrix} 11 & 11 & 11 \end{pmatrix}^T = 9.$$

Since $a > b$, by Lemma 5 we obtain $N_2 = \{j \in N \mid a_{1j} \otimes u_j > 9\} = \{1, 3\}$. Thus,

$$\bar{x}_j = \begin{cases} 9, & j \in \{1, 3\}, \\ u_j, & j = 2, \end{cases}$$

so $\bar{x} = (9, 11, 9)^T$ is a solution of $A \otimes x = B \otimes x$.

3.1.4. Interval Solution of $A \otimes x = B \otimes x$ for $1 \times n$ Matrices A, B with $x \in \overline{\mathbb{R}}^n$ Given an Upper Bound

Lemma 6 discusses the interval solution of $A \otimes x = B \otimes x$ with the constraint $\bar{x} \leq u$.

Definition 1. An interval vector $\hat{x}$ is called an interval solution of $A \otimes x = B \otimes x$ with upper bound u if for every $\bar{x} \in \hat{x}$, we have $A \otimes \bar{x} = B \otimes \bar{x}$.

Lemma 6. *Let $a > b$ and $N_2 = \{j \in N \mid a_{1j} \otimes u_j > b\}$. The interval vector $\hat{x}$ defined by*

$$\hat{x}_j := \begin{cases} [b, b] & j \in N_2, \\ [b, u_j] & \textit{otherwise}, \end{cases}$$

for $j = 1, 2, \ldots, n$ is an interval solution of $A \otimes x = B \otimes x$ with upper bound u.

Proof. Note that $b \leq \hat{x}_j \leq u_j$ for each $j \in N$. Since for $j \in N_2$, $a_{1j} > b$ implies $a_{1j} \otimes b = b$

and for $j \notin N_2$, $a_{1j} \otimes \hat{x}_j \leq a_{1j} \otimes u_j \leq b$, by direct substitution we have

$$A \otimes \hat{x} = \bigoplus_{j=1}^{n} (a_{1j} \otimes \hat{x}_j) = \left(\bigoplus_{j \in N_2} (a_{1j} \otimes b) \right) \oplus \left(\bigoplus_{j \in N \setminus N_2} (a_{1j} \otimes \hat{x}_j) \right) = b.$$

Next, we have $B \otimes \hat{x} \leq B \otimes u = b$. On the other hand, since $\hat{x}_j \geq b$, there exists k such that $b_{1k} \otimes u_k = b$, giving $B \otimes \hat{x} \geq b$. Hence, $B \otimes \hat{x} = b$ and $A \otimes \hat{x} = B \otimes \hat{x}$. So, $\hat{x}$ is a solution. □

3.2. Maximal Solution of $A \otimes x = B \otimes x$ for $m \times n$ Matrices A, B

In this subsection, we discuss the maximal solution of System (1) where A, B are of size $m \times n$. The maximal solution for $m \times n$ matrices is constructed by repeatedly applying Lemma 5 to each row, since the system $A \otimes x = B \otimes x$ consists of m equations of size $1 \times n$. Several adjustments are needed:

1. Identify rows where $a_i(x) \neq b_i(x)$.
2. Swap rows where $a_i(x) < b_i(x)$.
3. Choose the smallest right-hand side to ensure the condition for N_2 holds for all rows.

Based on these adjustments, the procedure to determine the maximal solution for $m \times n$ matrices is described as follows.

Let $M = \{1, \ldots, m\}$ and $N = \{1, \ldots, n\}$. Given the system $A \otimes x = B \otimes x$ with $A, B \in \overline{\mathbb{R}}^{m \times n}$, we begin by initialize $k = 0$, $u = (\infty, \infty, \ldots, \infty)^T$, and $\bar{x}^0 = u$.

Step 1. Set $k = k + 1$. If any equation in the system has not yet satisfied equality, swap the left and right sides so that

$$a_i(\bar{x}^{k-1}) \geq b_i(\bar{x}^{k-1})$$

holds for every $i \in M$.

Step 2. Determine

$$M_k = \{r \in M \mid a_r(\bar{x}^{k-1}) \neq b_r(\bar{x}^{k-1})\}$$

and

$$I_k = \{i \in M_k \mid b_i(\bar{x}^{k-1}) = \min_{r \in M_k} b_r(\bar{x}^{k-1})\}.$$

Then choose an index i_k such that $i_k \in I_k$.

Step 3. Determine the maximal solution $\bar{x}^k$ for row i_k with $\bar{x}^{k-1}$ as the upper bound. The solution is given by Lemma 5:

$$\bar{x}_j^k := \begin{cases} b_k & \text{if } j \in J_k, \\ \bar{x}_j^{k-1} & \text{otherwise,} \end{cases}$$

for $j = 1, 2, \ldots, n$, where

$$b_k = b_{i_k}(\bar{x}^{k-1}) = \bigoplus_{j=1}^{n} (b_{i_k j} \otimes \bar{x}_j^{k-1}),$$

and

$$J_k = \{j \in N \mid a_{i_k j} \otimes \bar{x}_j^{k-1} > b_k\}.$$

Step 4. Check whether $\bar{x}^k$ is a solution of $A \otimes x = B \otimes x$. If yes, $\bar{x}^k$ is the maximal solution and the process stops. Otherwise, return to Step 1 and repeat until a maximal solution is obtained.

The procedure above is formalized in Algorithm 1 below.

Algorithm 1: Determination of the maximal solution of $A \otimes x = B \otimes x$

Input : Matrices $A, B \in \overline{\mathbb{R}}^{m \times n}$
Output : Maximal solution $\bar{x} \in \overline{\mathbb{R}}^{n}$

$k \leftarrow 0$
$u \leftarrow (\infty, \infty, \ldots, \infty)^T$
$\bar{x}^0 \leftarrow u$
while *True* **do**
  $k \leftarrow k + 1$
  for $i \leftarrow 1$ **to** m **do**
    if $a_i(\bar{x}^{k-1}) < b_i(\bar{x}^{k-1})$ **then**
      swap a_i and b_i for row i `// Step 1: swap rows`
  $M_k \leftarrow \{r \in M \mid a_r(\bar{x}^{k-1}) \neq b_r(\bar{x}^{k-1})\}$ `// Step 2a: identify unsatisfied rows`
  if $M_k = \emptyset$ **then**
    return $\bar{x}^{k-1}$ `// solution found`
  $I_k \leftarrow \{i \in M_k \mid b_i(\bar{x}^{k-1}) = \min_{r \in M_k} b_r(\bar{x}^{k-1})\}$ `// Step 2b: select row with min RHS`
  choose $i_k \in I_k$
  $b_k \leftarrow b_{i_k}(\bar{x}^{k-1}) = \bigoplus_{j=1}^{n} (b_{i_k j} \otimes \bar{x}_j^{k-1})$ `// Step 3a: compute` b_k
  $J_k \leftarrow \{j \in N \mid a_{i_k j} \otimes \bar{x}_j^{k-1} > b_k\}$ `// Step 3b: compute` J_k
  for $j \leftarrow 1$ **to** n **do**
    if $j \in J_k$ **then**
      $\bar{x}_j^k \leftarrow b_k$ `// Step 3c: update solution vector`
    else
      $\bar{x}_j^k \leftarrow \bar{x}_j^{k-1}$
  if $\bar{x}^k$ *is a solution of* $A \otimes x = B \otimes x$ **then**
    return $\bar{x}^k$ `// Step 4: check solution`
return $\bar{x}^k$

To prove the validity of Algorithm 1, we first present Lemma 7 through Lemma 10, which describe the monotonicity and stability of the iterative sequences $\bar{x}^k$ and b_k generated by Algorithm 1. These lemmas, together with Lemma 11 and Lemma 12, establish that Algorithm 1 produces a solution and that this solution is maximal, within at most m cycles.

Lemma 7 discusses the interval solution at each cycle.

Lemma 7. *Let $\bar{x}^k$ be the solution for row i_k in Step 3 of cycle k. Then $\bar{x}^k$ can be replaced by any value in the interval $\hat{x}^k$ defined by*

$$\hat{x}_j^k := \begin{cases} [b_k, b_k], & j \in J_k, \\ [b_k, \bar{x}_j^{k-1}], & \text{otherwise}, \end{cases}$$

for $j = 1, 2, \ldots, n$, and we have

$$a_{i_k}(\hat{x}^k) = b_{i_k}(\hat{x}^k).$$

Proof. Given that $\bar{x}^k$ is the maximal solution obtained in Step 3. From Steps 1 and 2, we have $a_{i_k}(\bar{x}^{k-1}) > b_{i_k}(\bar{x}^{k-1})$. By Lemma 6, setting $a = a_{i_k}(\bar{x}^{k-1})$, $b = b_{i_k}(\bar{x}^{k-1}) = b_k$, and

$N_2 = J_k$, we obtain that

$$\hat{x}_j^k := \begin{cases} [b_k, b_k], & j \in J_k, \\ [b_k, \bar{x}_j^{k-1}], & \text{otherwise}, \end{cases}$$

for $j = 1, \ldots, n$ is a solution of

$$\begin{pmatrix} a_{i_k 1} & a_{i_k 2} & \cdots & a_{i_k n} \end{pmatrix} \otimes \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_{i_k 1} & b_{i_k 2} & \cdots & b_{i_k n} \end{pmatrix} \otimes \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}.$$

Thus, $a_{i_k}(\hat{x}^k) = b_{i_k}(\hat{x}^k)$. □

Lemma 8 shows that as the cycle increases, the solution vector becomes smaller or remains stable.

Lemma 8. *Let $\bar{x}^k$ be defined as in Step 3 of cycle k, and let $\bar{x}^{k-1}$ be from cycle $(k-1)$. Then*

$$\bar{x}^k \leq \bar{x}^{k-1}.$$

Proof. Recall that

$$\bar{x}_j^k := \begin{cases} b_k & \text{if } j \in J_k, \\ \bar{x}_j^{k-1} & \text{otherwise.} \end{cases}$$

For $j \in J_k$, we have $\bar{x}_j^k = b_k < a_{i_k j} \otimes \bar{x}_j^{k-1} \leq \bar{x}_j^{k-1}$, hence $\bar{x}_j^k < \bar{x}_j^{k-1}$. For $j \notin J_k$, we have $\bar{x}_j^k = \bar{x}_j^{k-1}$. Therefore $\bar{x}_j^k \leq \bar{x}_j^{k-1}$ for all j, so $\bar{x}^k \leq \bar{x}^{k-1}$. □

Lemma 9 proves that as the cycle increases, the minimum value on the right-hand side remains stable or increases. Let S_1 be the set of rows that have not yet satisfied the equation in cycle k, and S_2 the set of rows that have already satisfied it:

$$S_1 = \{s \in M \setminus I_k \mid b_s(\bar{x}^{k-1}) > b_k\}, \qquad S_2 = \{s \in M \setminus I_k \mid b_s(\bar{x}^{k-1}) \leq b_k\}.$$

These sets will be used in the proofs of Lemma 9 and Lemma 11.

Lemma 9. *Let b_k be defined as in Step 3 of cycle k with upper bound $\bar{x}^{k-1}$, and let b_{k+1} be defined in cycle $k+1$ with upper bound $\bar{x}^k$. Then*

$$b_{k+1} \geq b_k.$$

Proof. By definition, $b_k = \min_{r \in M_k} b_r(\bar{x}^{k-1})$ and $a_r(\bar{x}^{k-1}) > b_r(\bar{x}^{k-1})$. From Step 3, $\bar{x}_j^k = b_k$ for $j \in J_k$ and $\bar{x}_j^k = \bar{x}_j^{k-1}$ otherwise, with $b_k \leq \bar{x}_j^{k-1}$. Hence, $b_k \leq \bar{x}_j^k$ for all j. For any $s \in S_1$, we have $b_s(\bar{x}^{k-1}) > b_k$. Thus there exists j^* such that $b_{sj^*} \otimes \bar{x}_{j^*}^{k-1} > b_k$, implying $b_{sj^*} > b_k$. Since $\bar{x}_j^k \geq b_k$, we obtain $\min(b_{sj^*}, \bar{x}_{j^*}^k) \geq \min(b_{sj^*}, b_k) = b_k$. Therefore $b_s(\bar{x}^k) \geq b_k$ for every $s \in S_1$. Now $b_{k+1} = \min_{s \in S_1} b_s(\bar{x}^k) \geq b_k$, completing the proof. □

Using Lemma 8 and Lemma 9, we now show in Lemma 10 that once a variable is decreased, its value does not change in subsequent cycles.

Lemma 10. *For $j \in J_k$, once $\bar{x}_j^k$ is determined in iteration k, its value remains unchanged for all cycles l with $l > k$ and $l \leq K$.*

Proof. Since $j \in J_k$, we have $\bar{x}_j^k = b_k$. Take any $l > k$ and suppose, for contradiction, that $\bar{x}_j^l \neq \bar{x}_j^k$. Then by Step 3, $j \in J_l$ and $\bar{x}_j^l = b_l$. By Lemma 9, $b_l \geq b_k$. If $b_l > b_k$, then $\bar{x}_j^l = b_l > b_k = \bar{x}_j^k$, contradicting Lemma 8 which states $\bar{x}_j^l \leq \bar{x}_j^k$. If $b_l = b_k$, then $\bar{x}_j^l = b_l = b_k = \bar{x}_j^k$, contradicting the assumption $\bar{x}_j^l \neq \bar{x}_j^k$. Hence $\bar{x}_j^l = \bar{x}_j^k$. □

Lemma 11 states that any row that has already satisfied the equation remains satisfied in subsequent cycles.

Lemma 11. *Let $\bar{x}^k$ be defined as in Step 3. Then for every $s \in S_2$,*

$$a_s(\bar{x}^k) = b_s(\bar{x}^k).$$

Proof. Recall that $S_2 = \{s \in M \setminus I_k \mid b_s(\bar{x}^{k-1}) \leq b_k\}$. Let $s \in S_2$ be arbitrary. If $s \in M_k$, then by definition of I_k, $b_s(\bar{x}^{k-1}) > b_k$, which contradicts $s \in S_2$. Thus $s \notin M_k$, which implies row s was already satisfied in cycle $(k-1)$, i.e.,

$$a_s(\bar{x}^{k-1}) = b_s(\bar{x}^{k-1}) \leq b_k.$$

Consequently, for every $j \in \{1, 2, \ldots, n\}$, we have

$$a_{sj} \otimes \bar{x}_j^{k-1} \leq b_k \tag{6}$$

and

$$b_{sj} \otimes \bar{x}_j^{k-1} \leq b_k. \tag{7}$$

Now consider the update to $\bar{x}^k$. By Step 3, the components of $\bar{x}^k$ are given by

$$\bar{x}_j^k = \begin{cases} b_k, & \text{if } j \in J_k, \\ \bar{x}_j^{k-1}, & \text{otherwise.} \end{cases}$$

We consider the components $j \in \{1, 2, \ldots, n\}$ in two cases. For $j \notin J_k$, we have $\bar{x}_j^k = \bar{x}_j^{k-1}$. Therefore,

$$a_{sj} \otimes \bar{x}_j^k = a_{sj} \otimes \bar{x}_j^{k-1} \quad \text{and} \quad b_{sj} \otimes \bar{x}_j^k = b_{sj} \otimes \bar{x}_j^{k-1}.$$

For $j \in J_k$, we have $\bar{x}_j^k = b_k$. By Lemma 8, we have $\bar{x}^k \leq \bar{x}^{k-1}$, so $b_k = \bar{x}_j^k < \bar{x}_j^{k-1}$ for all $j \in J_k$. From Eq. (6), we obtain $a_{sj} \leq b_k$. Consequently, since $a_{sj} \leq b_k < \bar{x}_j^{k-1}$, we obtain

$$a_{sj} \otimes \bar{x}_j^k = a_{sj} \otimes b_k = a_{sj} = a_{sj} \otimes \bar{x}_j^{k-1}.$$

By an identical argument, Eq. (7) yields $b_{sj} \leq b_k$, thus

$$b_{sj} \otimes \bar{x}_j^k = b_{sj} \otimes b_k = b_{sj} = b_{sj} \otimes \bar{x}_j^{k-1}.$$

Since $a_{sj} \otimes \bar{x}_j^k = a_{sj} \otimes \bar{x}_j^{k-1}$ and $b_{sj} \otimes \bar{x}_j^k = b_{sj} \otimes \bar{x}_j^{k-1}$ for all j, taking the maximum over all j directly yields

$$a_s(\bar{x}^k) = \bigoplus_{j=1}^{n} (a_{sj} \otimes \bar{x}_j^k) = \bigoplus_{j=1}^{n} (a_{sj} \otimes \bar{x}_j^{k-1}) = a_s(\bar{x}^{k-1})$$

and

$$b_s(\bar{x}^k) = \bigoplus_{j=1}^{n} (b_{sj} \otimes \bar{x}_j^k) = \bigoplus_{j=1}^{n} (b_{sj} \otimes \bar{x}_j^{k-1}) = b_s(\bar{x}^{k-1}).$$

Since $a_s(\bar{x}^{k-1}) = b_s(\bar{x}^{k-1})$, we obtain $a_s(\bar{x}^k) = b_s(\bar{x}^k)$. Hence, row s remains satisfied in cycle k. □

Lemma 12 shows that every solution of $A \otimes x = B \otimes x$ is bounded above by $\bar{x}^k$ for any cycle k.

Lemma 12. *Let $\tilde{x}$ be any solution of $A \otimes x = B \otimes x$. Then $\tilde{x}$ does not exceed $\bar{x}^k$ from Step 3 for any cycle k.*

Proof. Let $\tilde{x}$ be any solution. We prove $\tilde{x} \not> \bar{x}^k$ by induction. For $k = 1$, by Lemma 5, $\bar{x}^1$ is the maximal solution for row i_1 with upper bound $\bar{x}^0$. Since $\tilde{x}$ is also a solution, we must have $\tilde{x} \not> \bar{x}^1$. Assume the statement holds for cycle $k-1$. Suppose $\tilde{x} > \bar{x}^k$, then there exists $j \in N$ such that $\tilde{x}_j > \bar{x}_j^k$. If $j \notin J_k$, then $\bar{x}_j^k = \bar{x}_j^{k-1}$, so $\tilde{x}_j > \bar{x}_j^{k-1}$, contradicting the induction hypothesis. If $j \in J_k$, then $\bar{x}_j^k = b_{i_k}(\bar{x}^{k-1})$, so $\tilde{x}_j > b_{i_k}(\bar{x}^{k-1})$. By definition of J_k, $a_{i_k j} \otimes \bar{x}_j^{k-1} > b_{i_k}(\bar{x}^{k-1})$, which implies $a_{i_k j} > b_{i_k}(\bar{x}^{k-1})$. Consequently,

$$a_{i_k}(\tilde{x}) = \bigoplus_{j=1}^{n} a_{i_k j} \otimes \tilde{x}_j > b_{i_k}(\bar{x}^{k-1}).$$

Since $\tilde{x}$ is a solution, $b_{i_k}(\tilde{x}) = a_{i_k}(\tilde{x}) > b_{i_k}(\bar{x}^{k-1})$, so

$$\bigoplus_{j=1}^{n} b_{i_k j} \otimes \tilde{x}_j > \bigoplus_{j=1}^{n} b_{i_k j} \otimes \bar{x}_j^{k-1}.$$

Thus there exists $l \in N$ such that $b_{i_k l} \otimes \tilde{x}_l > b_{i_k l} \otimes \bar{x}_l^{k-1}$, implying $\tilde{x}_l > \bar{x}_l^{k-1}$. This again contradicts the induction hypothesis. Hence $\tilde{x} \not> \bar{x}^k$, completing the induction. □

Algorithm 1 produces a maximal solution for the linear system $A \otimes x = B \otimes x$, as proved in Theorem 1.

Theorem 1. *The maximal solution $\bar{x}$ of the system $A \otimes x = B \otimes x$ with $A, B \in \overline{\mathbb{R}}^{m \times n}$ is given by Algorithm 1, with $\bar{x} = \bar{x}^K$, where $K \leq m$.*

Proof. We first show that Algorithm 1 terminates after at most m cycles. At each cycle k, the pivot row i_k selected in Step 2 satisfies, by Lemma 7, $a_{i_k}(\bar{x}^k) = b_{i_k}(\bar{x}^k)$ once Step 3 of that cycle is completed. By Lemma 11, once a row is satisfied it remains satisfied in every subsequent cycle. Since each row of the system is selected as pivot at most once, the set M_k of unsatisfied rows loses at least one member at every cycle; hence $M_k = \emptyset$ after at most m cycles, so $K \leq m$. By Lemma 11, $\bar{x}^K$ is a solution of $A \otimes x = B \otimes x$. By Lemma 12, no solution exceeds $\bar{x}^k$ for any cycle k; in particular, no solution exceeds $\bar{x}^K$. Hence, $\bar{x}^K$ is a maximal solution of $A \otimes x = B \otimes x$. □

In one cycle, Algorithm 1 computes $\bar{x}^k$ as a solution of $A \otimes x = B \otimes x$. Since there are m rows and each row involves operations on n columns, the number of operations per cycle is $m \times n$. Thus, each cycle has complexity $O(mn)$. By Theorem 1, $K \leq m$, so the maximum number of cycles required is m. Therefore, the total complexity is $m \times O(mn) = O(m^2 n)$.

For the $m \times n$ case, the same result as Lemma 3 holds. That is, if $\bar{x}$ is the maximal solution without constraints, then $\bar{x}$ is the maximal solution with lower bound l if $l \leq \bar{x}$ and it has no solution if $l \not\leq \bar{x}$. The proof is analogous to the $1 \times n$ case.

From Subsections 3.1 and 3.2, the system $A \otimes x = B \otimes x$ always has a solution when there are no constraints or when an upper bound $x \leq u$ is given. For $1 \times n$ without constraints, Lemma 2 explicitly constructs a solution. For $1 \times n$ with an upper bound, Lemma 5 provides a direct formula. For $m \times n$, Algorithm 1 converges by Lemma 10 and produces a solution (Theorem 1).

For a lower bound $l \leq x$, the existence of the maximal solution depends on l (Lemma 3). The uniqueness of the maximal solution is given in Theorem 2.

Theorem 2. *The maximal solution of $A \otimes x = B \otimes x$ produced by Algorithm 1 is unique.*

Proof. Let $\bar{x}, \bar{y}$ be two maximal solutions. Define $\bar{z} = \max(\bar{x}, \bar{y})$. Using distributivity,

$$A \otimes \bar{z} = (A \otimes \bar{x}) \oplus (A \otimes \bar{y}) = (B \otimes \bar{x}) \oplus (B \otimes \bar{y}) = B \otimes \bar{z},$$

so $\bar{z}$ is a solution. Since $\bar{x} \leq \bar{z}$ and $\bar{x}$ is maximal, $\bar{x} = \bar{z}$. Similarly, $\bar{y} = \bar{z}$. Hence, $\bar{x} = \bar{y}$. □

3.3. Application to Logistics Distribution Network Using Python

This section presents an application of maximal solutions of two-sided linear systems in max-min algebra to the planning of logistics distribution infrastructure.

Consider a transportation system connecting three industrial areas (A, B, and C) as the origin locations of goods to one main distribution center (T). To reach the distribution center T, each transport vehicle must pass through one of three transit points or toll gates (X, Y, and Z). The roads connecting the industrial areas to the toll gates are existing networks, while the roads connecting the toll gates to the distribution center are special access roads whose capacities are to be determined. The capacities of the existing roads, for both outward and return flows, may have different values. The distribution network scheme is shown in Figure 1.

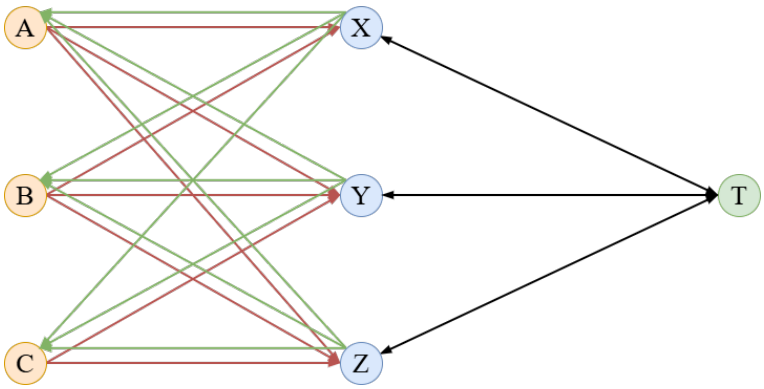

Fig. 1: Schematic diagram of the logistics distribution network

The road capacities (in passenger car units per hour, smp/hour) for each segment are given in Table 1 below.

Table 1: Forward and return road capacity

Forward road capacity			Return road capacity		
Symbol	**Route**	**Capacity (smp/hour)**	**Symbol**	**Route**	**Capacity (smp/hour)**
a_{11}	$A \to X$	1500	b_{11}	$X \to A$	1000
a_{12}	$A \to Y$	800	b_{12}	$Y \to A$	900
a_{13}	$A \to Z$	1200	b_{13}	$Z \to A$	1300
a_{21}	$B \to X$	900	b_{21}	$X \to B$	800
a_{22}	$B \to Y$	1600	b_{22}	$Y \to B$	1100
a_{23}	$B \to Z$	1100	b_{23}	$Z \to B$	1200
a_{31}	$C \to X$	1300	b_{31}	$X \to C$	1400
a_{32}	$C \to Y$	700	b_{32}	$Y \to C$	1000
a_{33}	$C \to Z$	1700	b_{33}	$Z \to C$	1500

We aim to determine the maximal capacity of the special access roads from each toll gate

to the distribution center, namely $x = (x_1, x_2, x_3)$, where x_1, x_2, x_3 respectively denote the two-way road capacities from X, Y, Z to T in smp/hour. The purpose of determining these capacities is to ensure that the incoming and outgoing vehicle flows remain balanced. With such balance achieved, logistics circulation at the distribution center can be supported and the risk of congestion caused by directional capacity imbalance can be mitigated. Based on Table 1, the maximal road capacities for each area toward the distribution center are as follows.

1. Area A
   (a) Forward capacity $= \max(\min(a_{11}, x_1), \min(a_{12}, x_2), \min(a_{13}, x_3))$
   (b) Return capacity $= \max(\min(b_{11}, x_1), \min(b_{12}, x_2), \min(b_{13}, x_3))$
2. Area B
   (a) Forward capacity $= \max(\min(a_{21}, x_1), \min(a_{22}, x_2), \min(a_{23}, x_3))$
   (b) Return capacity $= \max(\min(b_{21}, x_1), \min(b_{22}, x_2), \min(b_{23}, x_3))$
3. Area C
   (a) Forward capacity $= \max(\min(a_{31}, x_1), \min(a_{32}, x_2), \min(a_{33}, x_3))$
   (b) Return capacity $= \max(\min(b_{31}, x_1), \min(b_{32}, x_2), \min(b_{33}, x_3))$

To avoid vehicle congestion at the distribution center T, the forward and return capacities must be balanced for each area. Thus, we obtain the following two-sided linear system:

$$\begin{aligned}
\max(\min(a_{11}, x_1), \min(a_{12}, x_2), \min(a_{13}, x_3)) &= \max(\min(b_{11}, x_1), \min(b_{12}, x_2), \min(b_{13}, x_3)) \\
\max(\min(a_{21}, x_1), \min(a_{22}, x_2), \min(a_{23}, x_3)) &= \max(\min(b_{21}, x_1), \min(b_{22}, x_2), \min(b_{23}, x_3)) \\
\max(\min(a_{31}, x_1), \min(a_{32}, x_2), \min(a_{33}, x_3)) &= \max(\min(b_{31}, x_1), \min(b_{32}, x_2), \min(b_{33}, x_3))
\end{aligned}$$

or equivalently, $A \otimes x = B \otimes x$ with

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}, \quad B = \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{pmatrix}, \quad \text{and} \quad x = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}.$$

Applying Algorithm 1 to the system above, we trace the computation step by step. We initialize $k = 0$, $u = (\infty, \infty, \infty)^T$, and $\bar{x}^0 = u$.

Cycle $k = 1$**.** Since $a_i(\bar{x}^0) \geq b_i(\bar{x}^0)$ for every row ($a_1 = 1500 \geq b_1 = 1300$, $a_2 = 1600 \geq b_2 = 1200$, $a_3 = 1700 \geq b_3 = 1500$), no row is swapped. We obtain $M_1 = \{1, 2, 3\}$, and since $b_1 = 1300$, $b_2 = 1200$, $b_3 = 1500$, the minimum is attained at row 2, so $I_1 = \{2\}$ and $i_1 = 2$. We compute

$$b_1 = b_{i_1}(\bar{x}^0) = \max\big(\min(900, \infty), \min(1600, \infty), \min(1100, \infty)\big) = 1200,$$

and $J_1 = \{j \in N \mid a_{2j} \otimes \bar{x}^0_j > 1200\} = \{2\}$, since $a_{22} = 1600 > 1200$ while $a_{21} = 900$ and $a_{23} = 1100$ do not exceed 1200. Updating accordingly gives $\bar{x}^1 = (\infty, 1200, \infty)^T$.

Cycle $k = 2$**.** With $\bar{x}^1$, row 2 now satisfies $a_2(\bar{x}^1) = b_2(\bar{x}^1) = 1200$ and is excluded. Rows 1 and 3 remain unswapped, so $M_2 = \{1, 3\}$. Since $b_1(\bar{x}^1) = 1300 < b_3(\bar{x}^1) = 1500$, we have $I_2 = \{1\}$ and $i_2 = 1$. We compute

$$b_2 = b_{i_2}(\bar{x}^1) = \max\big(\min(1000, \infty), \min(900, 1200), \min(1300, \infty)\big) = 1300,$$

and $J_2 = \{j \in N \mid a_{1j} \otimes \bar{x}^1_j > 1300\} = \{1\}$, since $a_{11} \otimes \bar{x}^1_1 = \min(1500, \infty) = 1500 > 1300$, while $a_{12} \otimes \bar{x}^1_2 = \min(800, 1200) = 800$ and $a_{13} \otimes \bar{x}^1_3 = \min(1200, \infty) = 1200$ do not exceed 1300. Updating gives $\bar{x}^2 = (1300, 1200, \infty)^T$.

Cycle $k = 3$**.** With $\bar{x}^2$, rows 1 and 2 already satisfy equality and are excluded, leaving $M_3 = \{3\}$, so $I_3 = \{3\}$ and $i_3 = 3$. We compute

$$b_3 = b_{i_3}(\bar{x}^2) = \max\big(\min(1400, 1300), \min(1000, 1200), \min(1500, \infty)\big) = 1500,$$

and $J_3 = \{j \in N \mid a_{3j} \otimes \bar{x}_j^2 > 1500\} = \{3\}$, since $a_{33} \otimes \bar{x}_3^2 = \min(1700, \infty) = 1700 > 1500$, while $a_{31} \otimes \bar{x}_1^2 = \min(1300, 1300) = 1300$ and $a_{32} \otimes \bar{x}_2^2 = \min(700, 1200) = 700$ do not exceed 1500. Updating gives $\bar{x}^3 = (1300, 1200, 1500)^T$.

Checking $\bar{x}^3$ against all three rows confirms equality throughout:

1. Area A:
   $\max(\min(1500, 1300), \min(800, 1200), \min(1200, 1500)) = 1300$, and
   $\max(\min(1000, 1300), \min(900, 1200), \min(1300, 1500)) = 1300$;
2. Area B:
   $\max(\min(900, 1300), \min(1600, 1200), \min(1100, 1500)) = 1200$, and
   $\max(\min(800, 1300), \min(1100, 1200), \min(1200, 1500)) = 1200$;
3. Area C:
   $\max(\min(1300, 1300), \min(700, 1200), \min(1700, 1500)) = 1500$, and
   $\max(\min(1400, 1300), \min(1000, 1200), \min(1500, 1500)) = 1500$.

Since all rows are satisfied, the algorithm terminates after $K = 3$ cycles, consistent with $K \leq m = 3$ as established in Theorem 1, yielding the maximal solution $x = (1300, 1200, 1500)^T$.

This means that the special access roads from toll gates X, Y, and Z to the distribution center T should be built with capacities of 1300 smp/hour, 1200 smp/hour, and 1500 smp/hour, respectively. With these capacities, the forward and return flows for each industrial area are balanced. Consequently, the incoming and outgoing vehicle flows at each toll gate do not structurally exceed one another, meaning the capacity design does not inherently favor one direction over the other.

It should also be noted that the balance criterion used here captures only the bottleneck-capacity relationship between forward and return flows; it does not account for other operational factors relevant to real-world logistics planning, such as time-varying demand, travel time, transportation cost, queueing behavior at the toll gates, or physical and budgetary constraints on road construction. The numerical example therefore illustrates how the model can be applied to determine balanced road capacities, rather than serving as a complete operational recommendation for logistics infrastructure planning.

This maximality also provides a natural basis for comparison with alternative capacity assignments. Since $x = (1300, 1200, 1500)^T$ is the maximal solution of the system (Theorem 1), any smaller capacity vector satisfying the same balance condition would under-utilize the available road capacity, while any larger capacity vector would necessarily violate the balance condition for at least one industrial area. Moreover, by Theorem 2, this maximal solution is unique, so no other capacity assignment can achieve the same balance at a higher overall capacity. The proposed solution is therefore never dominated, in terms of capacity, by any other feasible balanced assignment.

In addition to this worked example, the implementation was further validated against several edge cases — a row already satisfied at the initial upper bound, a tie among rows for the minimum right-hand side (I_k) and the degenerate single-row ($m = 1$) case — all of which reduce correctly to the theoretical results of Sections 3.1 and 3.2; the corresponding test cases are provided in Appendix B.

4. Conclusion

This paper provides a Python implementation of the algorithm for determining the maximal solution of two-sided linear systems in max-min algebra, along with a numerical application to a logistics distribution network. The theoretical results show that for the $1 \times n$ case, the maximal solution can be explicitly constructed under three scenarios: without constraints, with a given upper bound, and with a given lower bound. These results are then generalized to the $m \times n$ case, where the algorithm terminates after at most m iterations and produces a

unique maximal solution with complexity $O(m^2n)$. The implementation makes the algorithm accessible to practitioners, and the logistics example demonstrates how the model can be used to determine road capacities. While the example uses simulated data, the approach is applicable to capacity planning problems in logistics and transportation. Future work may include applying the algorithm to real-world data or extending the implementation to other programming languages.

CRediT Authorship Contribution Statement

Zakia Nur Ramadhani Putri: Methodology, Formal Analysis, Software, Validation, Investigation, Writing – Original Draft Preparation, Visualization. **Ari Suparwanto:** Conceptualization, Supervision, Writing – Review & Editing. **Sutopo:** Conceptualization, Supervision, Writing – Review & Editing.

Declaration of Generative AI and AI-assisted technologies

During the preparation of this manuscript, the authors used ChatGPT (OpenAI) for writing assistance, language refinement, and proofreading. The AI tool was employed to assist in structuring the manuscript, improving sentence flow, and checking grammatical consistency. The authors have thoroughly reviewed and edited the AI-generated suggestions and take full responsibility for the final content of the publication.

Declaration of Competing Interest

The authors declare no competing interests.

Funding and Acknowledgments

This research received no external funding. The authors thank the Department of Mathematics, Universitas Gadjah Mada, for providing the necessary facilities and support during this study.

Data and Code Availability

The Python source code implementing Algorithm 1 is provided in Appendix A and Appendix B of this manuscript. The numerical example in Section 3 uses simulated data constructed solely for illustrative purposes, and all relevant data are included within the text. The code is also available from the corresponding author upon reasonable request.

References

[1] Cuninghame-Green and A. Raymond. *Minimax algebra*. Springer Science & Business Media, 1979. DOI: 10.1007/978-3-642-48708-8.

[2] M. Gondran and M. Minoux. *Graphs, dioids and semirings: new models and algorithms*. Springer, 2008. DOI: 10.1007/978-0-387-75450-5.

[3] E. Sanchez. "Resolution of composite fuzzy relation equations". In: *Information and Control* 30.1 (1976), pp. 38–48. DOI: 10.1016/S0019-9958(76)90446-0.

[4] E. Sanchez. "Inverses of fuzzy relations. Application to possibility distributions and medical diagnosis". In: *Fuzzy Sets and Systems* 2.1 (1979), pp. 75–86.

[5] E. S. Santos. "Maximin automata". In: *Information and Control* 13.4 (1968), pp. 363–377. DOI: 10.1016/S0019-9958(68)90864-4.

[6] P. Butkovič, K. Cechlárová, and P. Szabó. "Strong linear independence in bottleneck algebra". In: *Linear Algebra and its Applications* 94 (1987), pp. 133–155. DOI: 10.1016/0024-3795(87)90085-1.

[7] M. Fiedler, J. Nedoma, J. Ramík, J. Rohn, and K. Zimmermann. *Linear optimization problems with inexact data*. Springer Science & Business Media, 2006. DOI: 10.1007/0-387-32698-7.

[8] M. Gavalec and K. Zimmermann. "Solving systems of two–sided (max, min)–linear equations". In: *Kybernetika* 46.3 (2010), pp. 405–414. http://eudml.org/doc/33986.

[9] P. Krbálek and A. Pozdílková. "Maximal solutions of two–sided linear systems in max–min algebra". In: *Kybernetika* 46.3 (2010), pp. 501–512. http://eudml.org/doc/33988.

[10] H. Myšková and J. Plavka. "Optimizing the max-min function with a constraint on a two-sided linear system". In: *AIMS Mathematics* 9.4 (2024), pp. 7791–7809. DOI: 10.3934/math.2024378.

[11] P. Li. "Linear optimization over the approximate solutions of a system of max-min equations". In: *Fuzzy Sets and Systems* 485 (2024), p. 108946. DOI: 10.1016/j.fss.2024.108946.

[12] H. Myšková and J. Plavka. "AE and EA versions of X-robustness for interval circulant matrices in max–min algebra". In: *Linear and Multilinear Algebra* 71.15 (2022), pp. 2441–2461. DOI: 10.1080/03081087.2022.2105783.

[13] J. Plavka. "Strong X-robustness of interval max-min matrices". In: *Kybernetika* 58.1 (2022), pp. 50–68. DOI: 10.14736/kyb-2022-1-0050.

[14] Y. Ooga, Y. Nishida, and Y. Watanabe. "On max-plus two-sided linear systems whose solution sets are min-plus linear". In: *Linear Algebra and its Applications* 694 (2024), pp. 283–306. DOI: 10.1016/j.laa.2024.04.014.

[15] M. S. Mufid, E. Patel, and S. Sergeev. "Solving linear equations over maxmin-ω systems". In: *Linear Algebra and its Applications* 679 (2023), pp. 1–32. DOI: 10.1016/j.laa.2023.10.012.

[16] F. Baccelli, G. Cohen, G. J. Olsder, and J. P. Quadrat. *Synchronization and linearity*. Vol. 1. Wiley New York, 1992.

[17] Subiono. *Aljabar Min-Max Plus dan Terapannya*. Surabaya: Jurusan Matematika, Institut Teknologi Sepuluh Nopember, 2015.

[18] A. Rudhito. "Aljabar Max-Min Interval". In: *Prosiding Seminar Nasional Penelitian, Pendidikan, dan Penerapan MIPA*. Yogyakarta, 2013.

A. Python Implementation of Algorithm 1

```python
import numpy as np

# Function to print a separator line
def print_separator(title=""):
    print(f"\n{'='*50}")
    if title:
        print(f"{title}")
        print(f"{'='*50}")


def solve_two_sided_max_min(A, B, m, n, verbose=True):
    # ---- input validation ----
    if len(A) != m or len(B) != m:
        raise ValueError(
            f"A and B must each have {m} rows, but A has {len(A)} rows "
            f"and B has {len(B)} rows."
        )
    for i, row in enumerate(A):
        if len(row) != n:
            raise ValueError(f"Row {i + 1} of A has {len(row)} columns, "
                expected {n}.")
    for i, row in enumerate(B):
        if len(row) != n:
            raise ValueError(f"Row {i + 1} of B has {len(row)} columns, "
                expected {n}.")

    # Work on copies so the caller's A, B are never modified by the
    # row-swapping in Step 1.
    A = [row[:] for row in A]
    B = [row[:] for row in B]

    k = 0
    x = [float('inf')] * n

    if verbose:
        print_separator("START ALGORITHM")

    while k < m:
        k += 1

        # ========== Step 1 (swap sides) ==========
        for i in range(m):
            a_i = max(min(A[i][j], x[j]) for j in range(n))
            b_i = max(min(B[i][j], x[j]) for j in range(n))
            if a_i < b_i:
                A[i], B[i] = B[i], A[i]

        # ========== Step 2 (M_k and I_k) ==========
        rhs = []
        lhs = []
        M_k = []
        I_k = []

        for i in range(m):
            a_i = max(min(A[i][j], x[j]) for j in range(n))
            b_i = max(min(B[i][j], x[j]) for j in range(n))
```

```
55
56              if a_i != b_i:
57                  M_k.append(i)
58
59              rhs.append(b_i)
60              lhs.append(a_i)
61
62          if verbose:
63              print_separator(f"ITERATION␣{k}")
64              print(f"A␣=␣{np.array(A)}")
65              print(f"B␣=␣{np.array(B)}")
66              print(f"lhs␣(a_i)␣=␣{lhs}")
67              print(f"rhs␣(b_i)␣=␣{rhs}")
68              print(f"M_{k}␣=␣{[i␣+␣1␣for␣i␣in␣M_k]}")
69
70          # Check if M_k is empty (solution found)
71          if not M_k:
72              if verbose:
73                  print("SOLUTION␣FOUND!")
74                  print(f"x␣=␣{np.array(x)}")
75              return x, k
76
77          # Compute b_min and I_k (rows tied at the minimum are all kept
78          # in I_k; we simply pick the first one as i_k)
79          b_min = min(rhs[i] for i in M_k)
80          for i in M_k:
81              if rhs[i] == b_min:
82                  I_k.append(i)
83
84          i_k = I_k[0]
85          b_k = b_min
86
87          if verbose:
88              print(f"I_{k}␣=␣{[i␣+␣1␣for␣i␣in␣I_k]}")
89              print(f"b_{k}␣=␣{b_k}")
90              print(f"i_{k}␣=␣{i_k␣+␣1}")
91
92          # ========== Step 3 (J_k and update x) ==========
93          J_k = []
94          for j in range(n):
95              if min(A[i_k][j], x[j]) > b_k:
96                  J_k.append(j)
97
98          for j in J_k:
99              x[j] = b_k
100
101         if verbose:
102             print(f"J_{k}␣=␣{[j␣+␣1␣for␣j␣in␣J_k]}")
103             print(f"x␣=␣{np.array(x)}")
104
105         # ========== Step 4 (check solution) ==========
106         ai_new = []
107         bi_new = []
108         for i in range(m):
109             a_i = max(min(A[i][j], x[j]) for j in range(n))
110             b_i = max(min(B[i][j], x[j]) for j in range(n))
111             ai_new.append(a_i)
112             bi_new.append(b_i)
```

```
113
114         if verbose:
115             print(f"Solution␣check:␣ai␣=␣{ai_new},␣bi␣=␣{bi_new}")
116
117         if all(ai_new[i] == bi_new[i] for i in range(m)):
118             if verbose:
119                 print_separator("SOLUTION␣FOUND")
120                 print(f"x␣=␣{np.array(x)}")
121             return x, k
122
123     return x, k
124
125
126 def read_matrix(name, m, n):
127     print(f"Enter␣elements␣of␣matrix␣{name}")
128     return [[float(input(f"{name.lower()}_{i␣+␣1},{j␣+␣1}␣=␣"))
129               for j in range(n)] for i in range(m)]
130
131
132 if __name__ == "__main__":
133     print_separator("TWO-SIDED␣MAX-MIN␣LINEAR␣SYSTEM␣SOLVER")
134     print("Enter␣the␣number␣of␣rows␣and␣columns")
135     m = int(input("Number␣of␣rows␣␣␣␣:␣"))
136     n = int(input("Number␣of␣columns␣:␣"))
137
138     print(f"\nEnter␣elements␣of␣matrix␣A␣({m}␣x␣{n})")
139     A = read_matrix("A", m, n)
140     print(f"\nEnter␣elements␣of␣matrix␣B␣({m}␣x␣{n})")
141     B = read_matrix("B", m, n)
142
143     x, k = solve_two_sided_max_min(A, B, m, n, verbose=True)
144
145     print_separator("RESULT")
146     print(f"x␣=␣{x}")
147     print(f"Number␣of␣cycles␣used:␣K␣=␣{k}␣(K␣<=␣m␣=␣{m})")
```

Listing 1: Python implementation of Algorithm 1

B. Test Cases and Validation Results

Listing 2 presents the companion test suite used to validate the implementation in Appendix A. It reuses `solve_two_sided_max_min` from Appendix A (via a standard Python `import`, so both files must reside in the same directory) and covers the logistics example from Section 3.3 together with the edge cases and input-validation check discussed therein.

```
1  from two_sided_max_min_solver import solve_two_sided_max_min,
       print_separator
2
3
4  def verify_solution(A, B, x, m, n):
5      for i in range(m):
6          a_i = max(min(A[i][j], x[j]) for j in range(n))
7          b_i = max(min(B[i][j], x[j]) for j in range(n))
8          if a_i != b_i:
9              return False, i + 1, a_i, b_i
10     return True, None, None, None
11
12
```

```
def run_test(title, A, B, m, n, expected=None, verbose=True):
    print_separator(title)
    x, k = solve_two_sided_max_min(A, B, m, n, verbose=verbose)
    print(f"\nResult: x = {x}, number of cycles K = {k} (K <= m = {m})")
    if expected is not None:
        status = "MATCH" if x == expected else "MISMATCH"
        print(f"Expected: {expected} -> {status}")

    # Validation test: re-substitute x independently of the solver's
    # own Step 4 check, to confirm it really is a solution.
    ok, bad_row, a_val, b_val = verify_solution(A, B, x, m, n)
    if ok:
        print("Validation test: PASS (A (x) x = B (x) x holds for every
            row)")
    else:
        print(f"Validation test: FAIL at row {bad_row} (a={a_val}, b={
            b_val})")

    return x, k


if __name__ == "__main__":
    # Test 1: Main logistics example, Section 3.3 of the paper.
    A1 = [[1500, 800, 1200],
          [900, 1600, 1100],
          [1300, 700, 1700]]
    B1 = [[1000, 900, 1300],
          [800, 1100, 1200],
          [1400, 1000, 1500]]
    run_test("TEST 1: Logistics example (Section 3.3)",
             A1, B1, 3, 3, expected=[1300, 1200, 1500])

    # Edge Case 1: a row is already satisfied from the very first cycle.
    A2 = [[10, 6, 8],
          [5, 5, 5],
          [9, 7, 12]]
    B2 = [[7, 9, 6],
          [5, 5, 5],
          [11, 8, 6]]
    run_test("EDGE CASE 1: a row already satisfied at the start",
             A2, B2, 3, 3)

    # Edge Case 2: a tie in I_k (more than one minimum row).
    A3 = [[10, 8, 9],
          [12, 7, 10],
          [9, 9, 11]]
    B3 = [[6, 9, 8],
          [6, 8, 9],
          [7, 10, 9]]
    run_test("EDGE CASE 2: tie in I_k (more than one minimum row)",
             A3, B3, 3, 3)

    # Edge Case 3: m = 1 (reduces to the 1xn case)
    A4 = [[10, 8, 10]]
    B4 = [[9, 9, 9]]
    run_test("EDGE CASE 3: m = 1 (reduces to the 1xn case)",
             A4, B4, 1, 3, expected=[9, float('inf'), 9])

```

```
    print_separator("ALL␣TESTS␣COMPLETE")

    # Demo: input validation catches a malformed matrix.
    print_separator("DEMO:␣input␣validation")
    A_bad = [[1500, 800, 1200],
             [900, 1600],          # <- wrong number of columns on
                purpose
             [1300, 700, 1700]]
    B_bad = [[1000, 900, 1300],
             [800, 1100, 1200],
             [1400, 1000, 1500]]
    try:
        solve_two_sided_max_min(A_bad, B_bad, 3, 3, verbose=False)
        print("Unexpected:␣no␣error␣was␣raised.")
    except ValueError as e:
        print(f"Validation␣correctly␣caught␣the␣problem:␣{e}")
```

Listing 2: Test cases and validation checks for the implementation in Appendix A

Table 2 summarizes the results obtained by running Listing 2.

Table 2: Summary of test case and validation results

Test	Size ($m \times n$)	Expected x	Result x	K	Validation
Test 1 (Subsection 3.3)	3×3	$(1300, 1200, 1500)$	$(1300, 1200, 1500)$	3	PASS
Edge Case 1 (row satisfied at start)	3×3	$(9, \infty, 9)$	$(9, \infty, 9)$	2	PASS
Edge Case 2 (tie in I_k)	3×3	$(9, 9, 9)$	$(9, 9, 9)$	3	PASS
Edge Case 3 ($m = 1$)	1×3	$(9, \infty, 9)$	$(9, \infty, 9)$	1	PASS
Malformed input (row size mismatch)	$3 \times 3^*$	—	`ValueError` raised	—	PASS

*Row 2 of A deliberately given only 2 values instead of 3, to test the input validation in Appendix A.