A MATLAB Toolbox for Fuzzy Relational Calculus in Variety of Fuzzy Algebras

Ketty Peeva [*,†] and Zlatko Zahariev [†]

Posted Date: 14 August 2025

doi: 10.20944/preprints202508.1072.v1

Keywords: fuzzy relational calculus; fuzzy linear systems of equations; fuzzy linear systems of inequalities; finite fuzzy machines; optimization of linear objective function; software for fuzzy relational calculus

 doi:10.20944/preprints202508.1072.v1

Article

A MATLAB Toolbox for Fuzzy Relational Calculus in Variety of Fuzzy Algebras

Ketty Peeva [*,†] **and Zlatko Zahariev** [†]

Faculty of Applied Mathematics and Informatics, Technical University of Sofia, 8 Kliment Ohridski Blvd, 1000 Sofia, Bulgaria
* Correspondence: kgp@tu-sofia.bg
† These authors contributed equally to this work.

Abstract: The paper presents a comprehensive MATLAB software package designed to model and solve a broad class of problems in fuzzy relational calculus: inverse problem for fuzzy linear systems of equations and fuzzy linear systems of inequalities; behaviour, reduction and minimization for finite fuzzy machines; optimization of linear objective function under fuzzy linear system of equations or inequalities constraints. The software supports a variety of fuzzy algebras, such as max – min, min – max, max –product, Goguen, Gödel, and Łukasiewicz, offering tools for direct and inverse problem resolution. The implementation is based on well-established theoretical results in fuzzy logic and fuzzy algebraic structures, providing a robust foundation for exact and efficient computation. We begin by outlying the mathematical background and the principles behind the implemented algorithms. The development history of the package is briefly reviewed to highlight the motivations and design choices. A selection of examples is presented to illustrate the use of the software in different problem settings. The package is freely available and has been downloaded over 2100 times from MATLAB File Exchange, indicating the interest in applying fuzzy relational methods in computational environments.

Keywords: fuzzy relational calculus; fuzzy linear systems of equations; fuzzy linear systems of inequalities; finite fuzzy machines; optimization of linear objective function; software for fuzzy relational calculus

1. Introduction

We present theory and software for fuzzy relational calculus (FRC) with applications in variety of areas. Attention is paid on the inverse problem resolution and obtaining complete solution set for fuzzy linear system of equations (FLSEs) and fuzzy linear system of inequalities (FLSIs). Applications are given for finite fuzzy machines – behaviour, reduction and minimization, as well as for optimization of linear objective function under FLSEs or FLSIs constraints.

Chronologically the main problem - how to solve FLSEs arose when we investigated finite fuzzy machines: in order to find finite behaviour matrix, to reduce or minimize states, we needed a solution method for FLSEs. This motivated our interest to FLSEs.

Publications for FLSEs demonstrate long period of investigations for discovering methods and procedures to solve them. Traditional linear algebra methods [1] cannot be used here – operations in fuzzy algebras are different from classical addition and multiplication.

A short review of the main results and approaches follows. Fuzzy relational equations (FREs) were first proposed in connection with medical diagnosis by Sanchez [2,3]. In [4] (1976) he gave formulas how to compute the maximal solution in the case of max – min composition (Theorem 5) and the minimal solution in the case of min – max composition (Theorem 6).

After Sanchez results attention was paid to the lower solutions of max – min FLSEs, the complete solution set and the solution procedures. Higashi and Klir in [5] (1984) improved the results from Czogała et. all [6] (1982) and derived several schemes for solving finite FREs. assuming that the coefficients on the right-hind side of FLSEs are ordered decreasingly. The procedures are tremendous. Analogical approach was given by Miyakoshi and Shimbo [7] (1986), the results are valid for FREs with

 doi:10.20944/preprints202508.1072.v1

triangular norms. Pappis and Sugeno [8] (1985) proposed analytical method for obtaining the lower solutions, marking the essential coefficients for the solution and introducing the dominance condition (that helps to exclude from investigation some non-essential paths); nevertheless, the set of potential lower solutions has to be investigated in order to find valid one. Pappis and Adamopoulos [9–11], proposed procedures for inverse problem resolution in ZBasic for matrices with dimensions up to 10x10. Peeva [12] (1992) proposed analytical method and algorithms to solve FLSEs, introducing three semantic types of coefficients (G and E as essential and S as non-essential) for the potential solutions. She also marked that the time complexity of the algorithm for finding lower solutions is exponential, later proved by Chen and Wang [13] (2002).

After these early years the interest was paid to other compositions: min $-$ max (Peeva and Kyosev [15]); max $-$product (Bourke and Fisher [16], Di Nola et. all [17], Loetamonphong and Fang [18], Markovskii [19]); sup $-t-$norm (Bartl and Belohlávek [20], Feng Sun [21], Nosková and Perfilieva [22]); inf $-$residuum (Perfilieva and Nosková [23]), etc.

There exist many other valuable publications and generalizations for the inverse problem resolution: see Bartl as well as Bartl and co-authors [24–26]; De Baets [27]; Di Nola et. all [17]; Li and Fang [28]; Molai and Khorram [29]; Shieh [30]; Shivanian [31]; Wu and Guu [32].

Finding analytical method for the lower solutions is a long and arduous period [33], not to mention the even more difficult task of a software. The procedures are tremendous. The time complexity of the problem is exponential [13]. Various approaches were proposed to find solutions (extremal and the set of all solutions):

- analytical methods based on algebraic-logical approach [5,7,8,14,15,17,24,27,28], with applications in various subjects [15] and with the first software [34] by Kyosev;
- characteristic matrix and decomposition [7,8];
- covering and binding variables [19,35–37];
- partitions and irreducible paths [5];
- solution based matrix [13], etc.

In this paper attention is paid on software description (based on our unified approach and exact methods) for solving FLSEs in some $BL-$algebras: in Gödel algebra, in Goguen algebra and in Łukasiewicz algebra.

In Section 2 we give the basic notions for the exposition - $BL-$algebra, Gödel algebra, Goguen algebra and Łukasiewicz algebra, $t-$ and $s-$ norms, compositions of fuzzy relations and variety of products for finite fuzzy matrices.

Section 3 presents the theory and exact method for solving FLSEs or FLSIs in some $BL-$algebras. A short overview is given of the chronological development of the fundamental ideas.

Applications for finite fuzzy machines and for linear optimization problems are subject of Section 4. They concern solving behaviour, reduction and minimization problems for finite fuzzy machines and solving linear optimization problems subject to FLSEs of FLSIs constraints.

The exact algorithms currently used in the software [38] are explained in details in Section 5, together with their computational complexity.

Section 6 presents the software from [38] itself and describes each of its modules and capabilities.

All algorithms and software functions are supported by examples provided in Appendix A.

2. Basic Notions

In this section the notions for classical algebra, orders and lattices are given according to [1,39], for $BL-$algebra – according to [40], for fuzziness - according to [15].

Partial order relation on a partially ordered set (poset) P is denoted by the symbol $\leq$. By a *greatest element* of a poset P we mean an element $b \in P$ such that $x \leq b$ for all $x \in P$. The *least element* of P is defined dually.

 doi:10.20944/preprints202508.1072.v1

2.1. $BL-$algebra

$BL-$algebra is the algebraic structure:

$$BL = \langle L, \vee, \wedge, *, \rightarrow, 0, 1\rangle,$$

where $\vee, \wedge, *, \rightarrow$ are binary operations, $0, 1$ are constants and:

1. $L = \langle L, \vee, \wedge, 0, 1\rangle$ is a lattice with universal bounds 0 and 1;
2. $L = \langle L, *, 1\rangle$ is a commutative semigroup;
3. $*$ and $\rightarrow$ establish an adjoint couple:

$$z \leq (x \rightarrow y) \Leftrightarrow x * z \leq y, \forall x, y, z \in L.$$

4. for all $x, y \in L$

$$x * (x \rightarrow y) = x \wedge y \quad \text{and} \quad (x \rightarrow y) \vee (y \rightarrow x) = 1.$$

We suppose in next exposition that $L = [0, 1]$ and $x, y \in [0, 1]$.
The following algebras are examples for $BL-$algebras:

1. Gödel algebra

$$BL_G = \langle [0, 1], \vee, \wedge, \rightarrow_G, 0, 1\rangle,$$

where operations are

(a) *Maximum* or s_3-conorm:

$$\max\{x, y\} = x \vee y. \tag{1}$$

(b) *Minimum* or t_3-norm:

$$\min\{x, y\} = x \wedge y. \tag{2}$$

(c) The residuum $\rightarrow_G$ is

$$x \rightarrow_G y = \begin{cases} 1 \text{ if } & x \leq y \\ y \text{ if } & x > y \end{cases}. \tag{3}$$

(d) A supplementary operation is useful

$$x \varepsilon y = \begin{cases} y, & \text{if } x < y \\ 0, & \text{if } x \geq y \end{cases}. \tag{4}$$

2. Product (Goguen) algebra

$$BL_P = \langle [0, 1], \vee, \wedge, \circ, \rightarrow_P, 0, 1\rangle,$$

where max and min are as (1) and (2), respectively, $\circ$ is the conventional real number multiplication (the t_2- norm, i. e, $t_2(x, y) = xy$) and the residuum $\rightarrow_P$ is

$$x \rightarrow_P y = \begin{cases} 1 \text{ if } & x \leq y \\ \frac{y}{x} \text{ if } & x > y \end{cases}. \tag{5}$$

Here the supplementary useful operation is:

$$x \gamma y = \begin{cases} 0 \text{ if } & x \geq y \\ \frac{y-x}{1-x} \text{ if } & x < y \end{cases}. \tag{6}$$

3. Łukasiewicz algebra

$$BL_L = \langle [0, 1], \vee, \wedge, \otimes, \rightarrow_L, 0, 1\rangle,$$

where max and min are as (1) and (2), respectively, and

 doi:10.20944/preprints202508.1072.v1

(a) $x \otimes y = 0 \vee (x + y - 1) \equiv t_1(x, y)$.

(b) The residuum $\rightarrow_L$ is

$$x \rightarrow_L y = 1 \wedge (1 - x + y). \tag{7}$$

(c) A supplementary operation is useful

$$x \delta y = 0 \vee (y - x). \tag{8}$$

These operations, as well as the operations in Table 1 are realized in [38], see Sections 5, 6, and Apendix A for details and working examples.

These three couples are also described as $t-$norms and $s-$norms (or $t-$conorms):

Table 1. t-norms and s-norms.

t-norm	name	expression	s-norm	name	expression	
t_3	minimum, Gödel t-norm	$t_3(x, y) = \min\{x, y\}$	s_3	maximum, Gödel t-conorm	$s_3(x, y) = \max\{x, y\}$	
t_2	Algebraic product	$t_2(x, y) = xy$	s_2	Probabilistic sum	$s_2(x, y) = x + y - xy$	
t_1	Łukasiewicz t-norm	$t_1(x, y) = \max\{x + y - 1,\ 0\}$	s_1	Bounded sum	$s_1(x, y) = \min\{x + y, 1\}$	

2.2. Compositions of Fuzzy Relations and Fuzzy Matrix Products

Let X and Y be crisp sets and $F(X \times Y)$ denote all fuzzy sets over $X \times Y$. A *fuzzy relation* $R \in F(X \times Y)$ is defined as a fuzzy subset of the Cartesian product $X \times Y$,

$$R = \{ ((x, y),\ \mu_R(x, y)) \mid (x, y) \in X \times Y,\ \mu_R : X \times Y \rightarrow [0, 1] \}.$$

The *inverse* (or *transpose*) $R^{-1} \in F(Y \times X)$ of $R \in F(X \times Y)$ is defined as

$$R^{-1}(y, x) = R(x, y) \quad \text{for all pairs} \quad (y, x) \in Y \times X.$$

The finite fuzzy matrix (FFM) $A = (a_{ij})_{m \times n}$, with $a_{ij} \in [0, 1]$ for each $i, j, 1 \leq i \leq m, 1 \leq j \leq n$, is called a *membership matrix*.

Any fuzzy relation $R \in F(X \times Y)$ over finite support $X \times Y$ is representable by FFM, written for convenience with the same letter $R = (r_{ij})_{m \times n}$, where $r_{ij} = \mu_R(x_i, y_j)$ for any $(x_i, y_j) \in X \times Y$.

The matrix representation of the inverse fuzzy relation is $R^{-1} = (r_{ji})$, called *transpose* or *inverse* of the given matrix. In this sense $R^{-1} = R^t$.

Two FFMs are called *conformable*, if the number of the columns in the first FFM equals the number of the rows in the second FFM: the matrices $A = (a_{ij})_{m \times p}$ and $B = (b_{ij})_{p \times n}$ are conformable and their product $C = (c_{ij})_{m \times n} = AB$, in this order, makes sense.

We consider some products of FFMs in $BL-$algebras. Since in references special attention is paid on $max - min$, $min - max$ and $max - product$ compositions, they are specially listed in next Definition 1.

Definition 1. *Let $A = (a_{ij})_{m \times p}$ and $B = (b_{ij})_{p \times n}$ be conformable FFMs.*

 doi:10.20944/preprints202508.1072.v1

The FFM $C = (c_{ij})_{m\times n}$, *where*

i) $C = A \bullet B$, *is called the* $\max - \min$ *product of* A *and* B *if*

$$c_{ij} = \bigvee_{k=1}^{p} (a_{ik} \wedge b_{kj}) \ \ when \ \ 1 \le i \le m, \ 1 \le j \le n.$$

ii) $C = A \circ B$, *is called the* $\min - \max$ *product of* A *and* B *if*

$$c_{ij} = \bigwedge_{k=1}^{p} (a_{ik} \vee b_{kj}) \ \ when \ \ 1 \le i \le m, \ 1 \le j \le n.$$

iii) $C = A \odot B$, *is called the* $\max -$*product (the product '.' is the conventional real number multiplication) of* A *and* B *if*

$$c_{ij} = \bigvee_{k=1}^{p} (a_{ik}.b_{kj}) \ \ when \ \ 1 \le i \le m, \ 1 \le j \le n.$$

iv) $C = A \ \rightarrow_G \ B$, *is called the* $\min - \rightarrow_G$ *product of* A *and* B *if*

$$c_{ij} = \bigwedge_{k=1}^{p} (a_{ik} \ \rightarrow_G \ b_{kj}) \ \ when \ \ 1 \le i \le m, \ 1 \le j \le n.$$

v) $C = A \ \rightarrow_P \ B$, *is called the* $\min - \rightarrow_P$ *product of* A *and* B *if*

$$c_{ij} = \bigwedge_{k=1}^{p} (a_{ik} \rightarrow_P \ b_{kj}) \ \ when \ \ 1 \le i \le m, \ 1 \le j \le n.$$

vi) $C = A \, \varepsilon \, B$, *is called the* $\max -\varepsilon$ *product of* A *and* B *if*

$$c_{ij} = \bigvee_{k=1}^{p} (a_{ik} \, \varepsilon \, b_{kj}) \ \ when \ \ 1 \le i \le m, \ 1 \le j \le n.$$

For simplicity and compactness of exposition we write $A \, B$ for any of the matrix products introduced in Definition 1.

Definition 2. *Let* $A = (\mu_{ij}^A)_{m\times p}$ *and* $B = (\mu_{ij}^B)_{p\times n}$ *be conformable FFMs. The matrix* $C = (\mu_{ij}^C)_{m\times n}$ *is called* $s - t$ ***product*** *of* A *and* B *in* $BL-$*algebra, written* $C = A *_{BL} B$, *if for each* $i, \ j, 1 \le i \le m, \ \ 1 \le j \le n$ *it holds:*

$$\mu_{ij}^C = s_r \, {}_{k=1}^{p} \Big(t_r(\mu_{ik}^A, \ \mu_{kj}^B) \Big), \tag{9}$$

where s_r *and* t_r *are* $s-$*norm and* $t-$*norm respectively for* $r = 1, \ 2, \ 3$, *see Table 1.*

Definition 3. *Let* $A = (\mu_{ij}^A)_{m\times p}$ *and* $B = (\mu_{ij}^B)_{p\times n}$ *be conformable FFMs. The matrix* $C = (\mu_{ij}^C)_{m\times n}$ *is called* $t- \rightarrow_{BL}$ *product of* A *and* B *in* $BL-$*algebra, written* $C = A \rightarrow_{BL} B$, *if for each* $i, \ j, 1 \le i \le m, \ \ 1 \le j \le n$ *it holds:*

$$\mu_{ij}^C = t_3 \, {}_{k=1}^{p} \ \Big((\mu_{ik}^A \stackrel{r}{\rightarrow}_{BL} \ \mu_{kj}^B) \Big), \tag{10}$$

where t_3 *is the* $t-$*norm (see Table 1) and* $\stackrel{r}{\rightarrow}_{BL} = \delta$ *for* $r = 1$, *see (8);* $\stackrel{r}{\rightarrow}_{BL} = \gamma$ *for* $r = 2$, *see (6);* $\stackrel{r}{\rightarrow}_{BL} = \varepsilon$ *for* $r = 3$, *see (4).*

2.3. Direct and inverse problem resolution

Let $A = (a_{ij})_{m\times p}$ and $B = (b_{ij})_{p\times n}$ be conformable FFMs. Computing their product according to Definition 1, 2 or 3, respectively, is called ***direct problem resolution***.

 doi:10.20944/preprints202508.1072.v1

Let Let $A = (a_{ij})_{m\times p}$ and $C = (c_{ij})_{m\times n}$ be given FFMs. Computing the matrix $B = (b_{ij})_{p\times n}$, such that $AB = C$ or $A *_{BL} B = C$ or $A \to_{BL} B = C$ is called ***inverse problem resolution***.

Details for algorithm and software on this subject are given in subsections 5.1 and 6.2 with working examples in Appendix A.1.

3. FLSEs and FLSIs in Some $BL-$Algebras

We began our investigations before 1992 [12] with $\max - \min$, i.e. $s_3 - t_3$-norm FLSEs, given with long form and short form description:

$$\left|\begin{array}{ccccccc} (a_{11} \wedge & x_1) & \vee\cdots\vee & (a_{1n} \wedge & x_n) & = & b_1 \\ \cdots & & \cdots & & \cdots & \cdots & \cdots \\ (a_{m1} \wedge & x_1) & \vee\cdots\vee & (a_{mn} \wedge & x_n) & = & b_m \end{array}\right. \quad \leftrightarrow \quad A \bullet X = B. \tag{11}$$

Here $A = (a_{ij})_{m\times n}$ stands for the matrix of coefficients, $X = (x_j)_{n\times 1}$ – for the matrix of unknowns, $B = (b_i)_{m\times 1}$ is the right-hand side of the system, $a_{ij}, b_i, x_j \in [0, 1]$ for $i = 1, ..., m$ and $j = 1, ..., n$.

The next step was to investigate $\min - \max$ FLSEs $A \circ X = B$ and $\max - product$ FLSEs $A \odot X = B$. The main author's results for all these FLSEs are published in [41].

The theoretical background - how to solve (11), is implemented as universal scheme in theory, algorithms and software for all the other kind of compositions for the FLSEs and FLSIs ($s_3 - t_3$, $s_3 - t_2$, $s_3 - t_1$ as well as $t_3 - s_3$, $t_3 - s_2$, $t_3 - s_1$), presented in the next subsection.

3.1. $s_3 - t$-norm FLSEs in $BL-$algebras

The natural extension of the theory for $\max - \min$ FLSEs (11) leads to the unified approach for FLSEs when the composition is $s_3 - t$:

$$\left|\begin{array}{l} t_r(a_{11}, x_1) \vee t_r(a_{12}, x_2) \vee \ldots \vee t_r(a_{1n}, x_n) = b_1 \\ t_r(a_{21}, x_1) \vee t_r(a_{22}, x_2) \vee \ldots \vee t_r(a_{2n}, x_n) = b_2 \\ \ldots \\ t_r(a_{m1}, x_1) \vee t_r(a_{m2}, x_2) \vee \ldots \vee t_r(a_{mn}, x_n) = b_m \end{array}\right. \quad \leftrightarrow \quad A *_{BL} X = B. \tag{12}$$

where $A = (a_{ij})_{m\times n}$ is the matrix of coefficients, $B = (b_i)_{m\times 1}$ holds for the right-hand side vector, $X = (x_j)_{1\times n}$ is the vector of unknowns; for the indices we suppose $i = 1, ..., m$, $j = 1, ..., n$; $a_{ij}, b_i \in [0, 1]$ are given, $x_j \in [0, 1]$ marks the unknowns in the system and t_r ($r = 1, 2, 3$) is according to Table 1.

The approach is also valid for FLSEs $A \to_{BL} X = B$.

Solving (12) for the unknown matrix X is called ***inverse problem resolution for FLSEs*** $A *_{BL} X = B$.

In the next exposition we suppose that the matrices A, X and B are as follows: $A = (a_{ij})_{m\times n}$, $X = (x_j)_{1\times n}$ and $B = (b_i)_{m\times 1}$.

3.2. Solutions

For FFMs $X_{p\times n}$ and $Y_{p\times n}$ the inequality

$$X \leq Y$$

means $\mu_{ij}(x) \leq \mu_{ij}(y)$ for each $i = 1, \ldots, p$, $j = 1, \ldots, n$.

Definition 4. *For the FLSEs (12) $A *_{BL} X = B$:*

*i) $X^0_{n\times 1}$ is called a **point solution** of $A *_{BL} X = B$ if $A *_{BL} X^0 = B$ holds.*

*ii) The set of all solutions of (12) is called **complete solution set** and is denoted by $\mathbb{X}$. If $\mathbb{X} \neq \emptyset$ then (12) is called **consistent**, otherwise it is called **inconsistent**.*

*iii) A solution $\hat{X} \in \mathbb{X}$ is called an **upper** or **maximal** solution of (12) if for any $X \in \mathbb{X}$ the relation $\hat{X} \leq X$ implies $X = \hat{X}$. When the upper solution is unique, it is called the **greatest** or **maximum solution**.*

 doi:10.20944/preprints202508.1072.v1

*iv) A solution $\check{X} \in \mathbb{X}$ is called a **lower** or **minimal solution** of (12) if for any $X \in \mathbb{X}$ the relation $X \leq \check{X}$ implies $X = \check{X}$. When the lower solution is unique, it is called the **least** or **minimum solution**.*

*v) $(X_1, \ldots, X_n)$ with $X_j \subseteq [0, 1]$ for each j, $1 \leq j \leq n$, is called an **interval solution** if any $X^0 = (x_j^0)_{n \times 1}$ belongs to $\mathbb{X}$ when $x_j^0 \in X_j$ for each j, $1 \leq j \leq n$.*

*vi) Any interval solution, whose interval bounds are bounded by a lower solution from the left and by the greatest solution from the right, is called **maximal interval solution**.*

3.3. Solvability and greatest solution

We give the analytical expression for the greatest solution of FLSEs (12) and a criterion for its consistency.

Theorem 1. *[14] Let A and B be FFMs and let $\mathbb{X}$ be the complete solution set of $A *_{BL} X = B$. Then:*

i) $\mathbb{X} \neq \emptyset \quad \Leftrightarrow \quad \hat{X}_{BL} = A^t \rightarrow_{BL} B \in \mathbb{X}$;

ii) If the FLSEs (12) is consistent, then $\hat{X}_{BL} = A^t \rightarrow_{BL} B$ is its greatest solution.

iii) There exists polynomial time algorithm for computing $A^t \rightarrow_{BL} B$.

Corollary 1. *[14] The following statements are valid for FLSEs $A *_{BL} X = B$:*

i) The FLSEs (12) is consistent iff $B = A *_{BL} (A^t \rightarrow_{BL} B)$ holds;

ii) There exists polynomial time algorithm for establishing consistency of the FLSEs (12) and for computing its greatest solution $\hat{X}_{BL} = A^t \rightarrow_{BL} B$.

In fact, Theorem 1, item ii), gives the analytical expression for the greatest solution of the equation (12). It is the core of the algorithms and software presented in [38] for establishing consistency of FLSEs and finding its extremal solution: maximum for the equation (12) and minimum for the dual case.

First we present finding the greatest solution of FLSEs (12) $A *_{BL} X = B$, a criterion for its consistency and and how to find the lower solutions, based on [42].

3.4. Greatest Solution

According to Theorem 1 any solvable max $-t$-norm FLSEs has unique greatest solution. In order to find all solutions of the consistent system, it is necessary to find its greatest solution and all of its minimal solutions. Finding the greatest solution is relatively simple task which can be used as a criteria for establishing consistency of the system. Finding all minimal solutions is reasonable for consistent system.

3.4.1. Classical Approach

The approach how to solve (12) is based on Theorem 1 and Corollary 1.

If (12) is consistent, its greatest solution is $\widehat{X} = (\widehat{x}_j) = A^t \rightarrow_{BL} B$.

Using this fact, an appropriate algorithm for checking consistency of the system and for finding its greatest solution can be obtained. Its computational complexity is $O(m.n^2)$. Nevertheless that it is simple, it is too hard for such a task.

3.4.2. More Efficient Approach

Here we propose a simpler way to answer both questions for (12) – consistency and the greatest solution (simultaneously computing the greatest solution and establishing consistency of (12)). Extending the ideas of [12] and [15], we work with four types of coefficients (S, E, G and H) and a boolean vector (*IND*).

In the FLSEs (12):

 doi:10.20944/preprints202508.1072.v1

- If $r = 3$ i.e. operation is minimum ($t_3(x, y) = \min\{x, y\}$):
  - a_{ij} is called *S-type coefficient* if $a_{ij} < b_i$.
  - a_{ij} is called *E-type coefficient* if $a_{ij} = b_i$.
  - a_{ij} is called *G-type coefficient* if $a_{ij} > b_i$.
  - a_{ij} is called *H-type coefficient* if $a_{ij} \geq b_i$.
- If $r = 2$ i.e. operation is the algebraic product ($t_2(x, y) = xy$):
  - a_{ij} is called *S-type coefficient* if $a_{ij} < b_i$.
  - a_{ij} is called *E-type coefficient* if $a_{ij} = b_i = 0$.
  - a_{ij} is called *G-type coefficient* if $a_{ij} \geq b_i > 0$.
  - a_{ij} is called *H-type coefficient* if $a_{ij} \geq b_i$.
- If $r = 1$ i.e. operation is the Łukasiewicz $t-$norm ($t_1(x, y) = \max\{x + y - 1, 0\}$):
  - a_{ij} is called *S-type coefficient* if $a_{ij} - 1 > b_i$.
  - a_{ij} is called *E-type coefficient* if $a_{ij} = b_i = 0$.
  - a_{ij} is called *G-type coefficient* if $a_{ij} \geq b_i > 0$.
  - a_{ij} is called *H-type coefficient* if $a_{ij} - 1 \leq b_i$.

The next Algorithm 1 uses the fact that it is possible to find the value of the unknown $\widehat{x}_j$ only by the j^{th} column of the matrix A. For every $i = 1, ..., m$ and depending on the operation, we consider the following cases:

- When the operation is t_3
  - If a_{ij} is E-type coefficient then the i-th equation can be satisfied by $a_{ij} \wedge x_j$ when $x_j \geq b_i$ because $a_{ij} \wedge x_j = b_i \wedge x_j = b_i$.
  - If a_{ij} is G-type coefficient then the i-th equation can be satisfied by $a_{ij} \wedge x_j$ only when $x_j = b_i$ because $a_{ij} \wedge x_j = a_{ij} \wedge b_i = b_i$.
  - If a_{ij} is S-type coefficient then the i-th equation cannot be satisfied by $a_{ij} \wedge x_j$ for any $x_j \in [0, 1]$.
- When the operation is t_2
  - If a_{ij} is E-type coefficient then the i-th equation can be satisfied by $a_{ij}x_j$ when $x_j \in [0, 1]$ because $a_{ij}x_j = 0x_j = b_i = 0$.
  - If a_{ij} is G-type coefficient then the i-th equation can be satisfied by $a_{ij}x_j$ only when $x_j = b_i/a_{ij}$ because $a_{ij}x_j = a_{ij}(b_i/a_{ij}) = b_i$.
  - If a_{ij} is S-type coefficient then the i-th equation cannot be satisfied by $a_{ij}x_j$ for any $x_j \in [0, 1]$.
- When the operation is t_1
  - If a_{ij} is E-type coefficient then the i-th equation can be satisfied by $a_{ij} \otimes x_j$ when $x_j \in [0, 1]$ because $a_{ij} \otimes x_j = \max\{0 + x_j - 1,\ 0\} = \max\{x_j - 1,\ 0\} = b_i = 0$.
  - If a_{ij} is H-type coefficient then the i-th equation can be satisfied by $a_{ij} \otimes x_j$ only when $x_j = 1 - a_{ij} + b_i$ because $a_{ij} \otimes x_j = a_{ij} + 1 - a_{ij} + b_i - 1 = b_i$.
  - If a_{ij} is S-type coefficient then the i-th equation cannot be satisfied by $a_{ij} \otimes x_j$ for any $x_j \in [0, 1]$.

  Hence, S-type coefficients are not interesting because they do not lead to solution.
  For the purposes of the next theorem, $\widehat{b}_j$ is introduced as follows:
- If the operation is t_3:

$$\widehat{b}_j = \begin{cases} \min_{i=1}^{m} \{b_i\}, \text{ for all } i \text{ such that } a_{ij} > b_i \\ 1 \text{ otherwise} \end{cases} \tag{13}$$

- If the operation is t_2:

$$\widehat{b}_j = \begin{cases} \min_{i=1}^{m} \{b_i/a_{ij}\}, \text{ for all } i \text{ such that } a_{ij} > b_i \\ 1 \text{ otherwise} \end{cases} \tag{14}$$

 doi:10.20944/preprints202508.1072.v1

- If the operation is t_1:

$$\widehat{b}_j = \begin{cases} \min_{i=1}^{m} \{1 - a_{ij} + b_i\}, \text{for all } i \text{ such that } a_{ij} - 1 < b_i \\ 1 \text{ otherwise} \end{cases} \tag{15}$$

Theorem 2. *[42] The FLSEs (12) is consistent iff $\widehat{X} = (\widehat{b}_j)$ is its solution.*

Corollary 2. *[42] In a consistent FLSEs (12):*

- *choosing $x_j > \widehat{b}_j$ for at least one $j = 1, ..., n$ makes the system inconsistent;*
- *for every $j = 1, ..., n$, the greatest admissible value for x_j is $\widehat{b}_j$;*
- *its greatest solution is $\widehat{X} = (\widehat{x}_j) = (\widehat{b}_j)$, $j = 1, ..., n$.*

In general, Theorem 2 and its Corollary show that instead of calculating $\widehat{X} = A^t \rightarrow_{BL} B$ we can implement a faster way to obtain $\widehat{X} = (\widehat{x}_j) = (\widehat{b}_j)$ (as realized in Algorithm 1).

$\widehat{X}$ is only the eventual greatest solution of the FLSEs (12) and it can be obtained for any FLSEs, even if the system is inconsistent, so the eventual solution should be checked in order to confirm that it is a solution of (12). Explicit checking for the eventual solution will increase the computational complexity of the algorithm. To avoid this in the next presented Algorithm 1 this is done by extraction of the coefficients of the potential greatest solution. For every $(\widehat{x}_j) \in \widehat{X}$ we check which equations of (12) are satisfied (hold in the boolean vector IND). If at the end of the algorithm all the equations of FLSEs (12) are satisfied (i.e. all the coefficients in IND are set to $TRUE$) this means that the system is consistent and the computed solution is its greatest solution, otherwise the FLSEs (12) is inconsistent.

Algorithm 1 Greatest solution and consistency of FLSEs (12).

Step 1: Initialize the vector $\widehat{X} = (\widehat{x}_j)$ with $\widehat{x}_j = 1$ for $j = 1, ..., n$.
Step 2: Initialize a boolean vector IND with $IND_i = FALSE$ for $i = 1, ..., m$. This vector is used to mark equations that are satisfied by the eventual greatest solution.
Step 3: For each column $j = 1, ..., n$ in A: walk successively through all coefficients $a_{ij}, i = 1, ..., m$ seeking the smallest G-type coefficient.
Step 4: If a_{ij} is E-type coefficient, according to (13), (14) or (15) respectively (depending on the operation), it means that the i^{th} equation in the system can be solved through this coefficient, but $\widehat{b}_j$ still should be found. Correct IND_i to $TRUE$.
Step 5: For the smallest H-type (which is not E-type) coefficient correct IND_i to $TRUE$. All other H-type (which are not E-type) coefficients are now insignificant, as they are not leading to a solution. In $\widehat{X}$ correct the value for $\widehat{x}_j = \widehat{b}_j$.
Step 6: Check if all components of IND are set to $TRUE$.
Step 7: If $IND_i = FALSE$ for some i the system $A *_{BL} X = B$ is inconsistent.
Step 8: If $IND_i = TRUE$ for all $i = 1, ..., m$ the system $A *_{BL} X = B$ is consistent and its greatest solution is stored in $\widehat{X}$.

Theorem 2 and its Corollary provide that if (12) is consistent, $\widehat{X}$ computed by Algorithm 1 is its greatest solution.

Algorithm 1 uses IND vector to check which equations are satisfied by the eventual $\widehat{X}$. At the end of the algorithm if all components in IND are $TRUE$ then $\widehat{X}$ is the greatest solution of the system, otherwise the system is inconsistent.

3.5. Lower Solutions

Each equation in (12) can be satisfied only by the terms with H-type coefficients. Also, the minimal value for every component in the solution is either the value of the corresponding $\hat{b}$ or 0. Along these lines, the hearth of the presented here algorithm is to find H-type components a_{ij} in A and to give to X_{low_j} either the value of the corresponding $\hat{b}$ when the coefficient contributes to solve the system or 0 when it doesn't.

 doi:10.20944/preprints202508.1072.v1

Using this, the set of candidates for solutions can be obtained. All candidate solutions are of three different types:

- Lower solution;
- Non-lower solution;
- Not solution at all.

The aim of the next Algorithm 2 is to extract all lower solutions and to skip the second and third types (non-lower solution or not solution at all). In order to extract all lower solutions a method, based on the idea of the dominance matrix in combination with list manipulation techniques is developed here.

3.5.1. Domination

For the purposes of presented here Algorithm 2, the definition for domination is given.

Definition 5. *Let a_l and a_k be the l^{th} and the k^{th} equations, respectively, in (12) and $b_l \geq b_k$. Equation a_l is called dominant to a_k and equation a_k is called dominated by a_l, if for each $j = 1, ..., n$ it holds: if a_{lj} is H-type coefficient then a_{kj} is also H-type coefficient.*

3.5.2. Extracting Lower Solutions

Lower solutions are extracted by removing from A the dominated rows. A new matrix is produced and marked with $\widetilde{A} = (a_{\widetilde{i}\widetilde{j}})$ where $\widetilde{i} = 1, ..., \widetilde{m}$, $\widetilde{m} < m$ for obvious reasons. It preserves all the needed information from A to obtain the solutions.

Extraction introduced here is based on the following recursive principle. If in the j^{th} column of $\widetilde{A}$ there are one or more rows $(\widetilde{i}^*)$ such that coefficients $a_{\widetilde{i}^*\widetilde{j}}$ are H-type then x_j should be taken equal to the smallest $b_{\widetilde{i}^*}$ and all rows $\widetilde{i}^*$ should be removed from $\widetilde{A}$. The same procedure is repeated for $(j+1)^{th}$ column of the reduced $\widetilde{A}$. "Backtracking" based algorithm using this principle is presented next:

Algorithm 2 Extract the lower solutions from $\widetilde{A}$.

Step 1: Initialize solution vector $X_{low_0}(j) = 0,\ j = 1, ..., n$.

Step 2: Initialize a vector $rows(\widetilde{i}),\ i = 1, ..., \widetilde{m}$ which holds all consecutive row numbers in $\widetilde{A}$. This vector is used as a stopping condition for the recursion. Initially it holds all the rows in $\widetilde{A}$. On every step some of the rows there are removed. When *rows* is empty the algorithm exits from the current recursive branch.

Step 3: Initialize *sols* to be the empty set of vectors, which is supposed to be the set of all minimal solutions for current problem.

Step 4: Check if $rows = \emptyset$. If so, add X_{low_0} to *sols* and exit current step.

Step 5: Fix $\widetilde{i}$ equal to the first element in *rows*, then for every $j = 1, ..., n$ such that $a_{\widetilde{i}\widetilde{j}}$ is H-type coefficient

Step 6: Create a copy of X_{low_0} and update its j^{th} coefficient to be equal to $b_{\widetilde{i}}$. Create a copy of *rows*.

Step 7: For all $\widetilde{k}$ in *rows* if $a_{\widetilde{k}\widetilde{j}}$ is a H-type coefficient, remove $\widetilde{k}$ from the copy of *rows*.

Step 8: Go to step 4 with copied in this step *rows* and X_{low_0}, i.e. start new recursive branch with reduced *rows* and changed X_{low_0}.

3.6. General algorithm

The next algorithm is based on the above given Algorithms 1 and 2.

Algorithm 3 Solving $A *_{BL} X = B$.

Step 1: Obtain input data for the matrices A and B.

Step 2: Obtain the greatest solution for the system and check it for consistency (Algorithm 1).

Step 3: If the system is inconsistent - Exit.

Step 4: Obtain the matrix $\widetilde{A}$.

Step 5: Obtain all minimal solutions from $\widetilde{A}$ and B, Algorithm 2.

 doi:10.20944/preprints202508.1072.v1

Algorithm 2 (Step 5) is the slowest part of the Algorithm 3. In general this algorithm has its best and worst cases and this is the most important improvement according to algebraic-logical approach [41] (from the time complexity point of view). Algorithm 2 is going to have the same time complexity as the algorithm presented in [14] only in its worst case.

Concerning the FLSEs $A \rightarrow_{BL} X = B$ - this is the analogical case and we do not describe the theory here, but software is also available in [38].

More details for algorithms and software for all of the main problems in this section are given in subsections 5.2 and 6.3 with working examples in Apendix A.2.

Software for inverse problem resolution is provided in [34] (only for $\max - \min$ and $\min - \max$ FLSEs) and in [38] for all other cases.

3.7. $s_3 - t$-Norm FLSIs in $BL-$Algebras

For the FLSIs $A *_{BL} X \leq B$, $A *_{BL} X \geq B$, $A \rightarrow_{BL} X \leq B$, $A \rightarrow_{BL} X \geq B$, we implement analogical approach as described for the FLSEs $A *_{BL} = B$.

For algorithms and software for FLSIs see also subsections 5.4, 5.5 and 6.3 with working examples in Apendix A.2.

The algorithms implemented in the software [38] are described in Section 5. They are the result of more than a decade of continuous refinement [14,15]. Earlier versions were built on the analytical methods [12,14,15], aimed at solving FLSEs, especially for the max-min composition.

3.8. Short Chronological Overview

The historical core of these methods was the algebraic-logical approach [12,14,15,41] which is still the main basis of all the currently used algorithms. This approach introduced a structured methodology involving potential extremal solutions, consistency checks via index vectors, and coefficient classification. The coefficients in the matrix of the coefficients A were categorized into three semantic types : *S-type*, *E-type*, and *G-type* coefficients [12,15]. A later refinement introduced *H-type* coefficients [41], which acted as a lightweight abstraction to further streamline certain matrix operations without changing the semantic interpretation of the system.

Such classifications made it possible to construct *associated matrix* [12] that was used to generate solution intervals and identify dominating paths. The *dominance relation* [41] was fundamental for reducing complexity by pruning branches that could not lead to solutions.

The next fundamental step is the construction of so-called *help matrix* [15], used to guide the recursive generation of candidate solutions. In some algorithmic versions, these structures were complemented by list-based procedures, where possible solutions and operations were represented and manipulated as lists rather than fixed-size matrices. These list-based algorithms enabled greater flexibility in traversing the solutions, but turn out to be even more complex from implementational point of view. Although the computational complexity of this problem is exponential [13], various simplification heuristics were developed to make the process more tractable in practice.

Over time, these methods evolved and expanded to support a wide variety of compositions in BL-algebras, including not only max–min but also min - max, max-product, and other BL systems such as those using the Goguen or Łukasiewicz implications. [42–44]. New versions of the algorithms integrated modular procedures that preserved the core logical structure while allowing extensibility to different algebras.

It is worth mentioning that the current implementation still employs a structure referred to as the *help matrix*, although its construction and role have significantly changed. The current algorithmic flow is explained in Section 5.

The evolution of these methods and their application to broader classes of fuzzy systems is documented in a series of publications [42–44]. These works include concrete examples and software implementations. Although the internal architecture of the software has shifted toward more unified and abstract representations (as detailed in Section 6), many of the core ideas — such as the use

 doi:10.20944/preprints202508.1072.v1

of coefficient classification, list-based construction, and filtering - continue to influence the current implementation.

4. Some Applications

4.1. Linear Dependency/Independency

Let $A(1) = (a_{i1})_{m\times 1}$, $A(2) = (a_{i2})_{m\times 1}$, ..., $A(n) = (a_{in})_{m\times 1}$ be fuzzy column-vectors with elements $a_{ij} \in [0,1]$ for $i = 1,...,m$, $j = 1,...,n$.

The fuzzy column-vector $B = (b_j)_{m\times 1}$ is called **linear combination** of $A(1), A(2), \ldots, A(n)$ if there exist $X = (x_j)_{1\times n}$ such that $A *_{BL} X = B$. Checking whether $B_{m\times 1}$ is a linear combination of $A(1), \ldots, A(n)$ requires to solve the system (12)

$$A *_{BL} X = B$$

for the unknown X, if $A_{m\times n}$ has as columns $A(1), A(2), \ldots, A(n)$.

If the FLSEs $A *_{BL} X = B$ is consistent the vectors $A(1), A(2), \ldots, A(n), B$ are ***linearly dependent***. When the FLSEs $A *_{BL} X = B$ is inconsistent, the vectors $A(1), A(2), \ldots, A(n), B$ are ***linearly independent***.

We implement this for instance in subsection 4.2 for finite fuzzy machines when computing the finite behaviour matrix, as well as for minimization of states.

The algorithms in subsections 5.6 and 5.7 are for establishing linear combination as well as linear dependency/independency. Software is described in subsection 6.2. Examples are given in Appendix A.1.

4.2. Finite Fuzzy Machines

We present finite fuzzy machines (FFMachs) with emphasis on the complete behavior matrix, reduction and minimization problems. For solving all these problems we implement the software from [38],

FFMachs were introduced by Santos, see [45] – [47]. Here the presentation is based on our results as published in [15,48].

For a finite set C we denote by $|C|$ its cardinality.

Definition 6. *A finite fuzzy machine is a quadruple*

$$\mathcal{A} = (X, Q, Y, \mathcal{M}),$$

where:

i) X, Q, Y are nonempty finite sets of input letters, states and output letters, respectively.

ii) $\mathcal{M}$ is the set of transition-output matrices of $\mathcal{A}$, that determines its stepwise behavior. Each matrix $M(x|y) = (m_{qq'}(x|y)) \in \mathcal{M}$ is a square matrix of order $|Q|$ and $x \in X$, $y \in Y$, $q, q' \in Q$, $m_{qq'}(x|y) \in [0,1]$.

In Definition 6 ii) we mark by $(x|y)$ the pair $(x,y) \in X \times Y$ to emphasize that $x \in X$ is the input letter when $y \in Y$ is the output response letter. The same stipulation is used in next exposition for input-output pair of words of the same length.

We regard $m_{qq'}(x|y)$ as the degree of membership for the FFMach to enter state $q' \in Q$ and produce output $y \in Y$ if the present state is $q \in Q$ and the input is $x \in X$.

4.2.1. Extended input-output behavior

While $\mathcal{M}$ is the set of transition-output matrices that describes operating of $\mathcal{A}$ for exactly one step, in this subsection we will be interested in matrices that describe operating of $\mathcal{A}$ for input-output pairs of words.

 doi:10.20944/preprints202508.1072.v1

The *free monoid of the words* over the finite set X is denoted by X^* with the *empty word* e as the identity element. If $X \neq \emptyset$ then X^* is countably infinite. The *length of the word* u is denoted by $|\,u\,|$. By definition $|\,e\,| = 0$. Obviously $|\,u\,| \in \mathbb{N}$ for each $u \neq e$, $\mathbb{N} = \{1, 2, \ldots\}$ stands for the set of natural numbers.

For $u \in X^*$ and $v \in Y^*$, if $|\,u\,| = |\,v\,|$ we write $(u|v) \in (X|Y)^*$ to distinguish it from the case $(u, v) \in X^* \times Y^*$. We denote by $(X\,|Y)^*$ the set of all input-output pairs of words of the same length:

$$(X\,|Y)^* = \{\,(u|v)\,|\,u \in X^*,\ v \in Y^*,\ |\,u\,| = |\,v\,|\,\}.$$

Definition 7. *Let $\mathcal{A} = (X, Q, Y, \mathcal{M})$ be FFMach.*

For any $(u|v) \in (X|Y)^*$ the extended input-output behavior of $\mathcal{A}$ upon the law of composition $*$ is determined by the square matrix $M(u|v)$ of order $|\,Q\,|$:

$$M(u\,|\,v) = \begin{cases} M(x_1|y_1) * \ldots * M(x_k|y_k), \\ \qquad if \quad (u|v) = (x_1 \ldots x_k|y_1 \ldots y_k),\ k \geq 1 \\ \\ E, \quad if \quad (u|v) = (e|e) \end{cases}, \tag{16}$$

where $M(x_1\,|\,y_1) * \ldots * M(x_k\,|\,y_k)$, for $* = \max - \min$, $\min - \max$ or $\max -$ products and $E = (e_{ij})$ is the square matrix of order $|Q|$ with elements e_{ij} determined as follows:

- *if the composition is* $\max - \min$ *or* $\max -$ *product,*

$$e_{ij} = \begin{cases} 1, & if\ i = j \\ 0, & if\ i \neq j \end{cases}.$$

- *if the composition is* $\min - \max$,

$$e_{ij} = \begin{cases} 0, & if\ i = j \\ 1, & if\ i \neq j \end{cases}.$$

Each element $m_{qq'}(u|v)$ in $M(u|v)$ is the degree of membership that $\mathcal{A}$ will enter state $q' \in Q$ and produce output word $v \in Y^*$ under the input word $u \in X^*$ beginning at state $q \in Q$, after $|u| = |v|$ consecutive steps.

Definition 7 describes the extended input-output behavior of $\max - \min$, $\min - \max$ and $\max -$product FFMachs.

4.2.2. Complete input-output behavior

When we consider FFMach as a 'black box' we are not interested in the next state q'. This is the essence of the input-output behavior of $\mathcal{A}$, determined by column-matrices $T(u|v)_{|Q|\times 1}$ as follows:

$$T(u|v)_{|\,Q\,|\times 1} = \big(t_q(u|v)\big) = \begin{cases} M(u|v) * E, \text{ if } (u|v) \neq (e|e); \\ E, \text{ if } (u|v) = (e|e), \end{cases} \tag{17}$$

where E is the $|\,Q\,| \times 1$ column-matrix with all elements equal to 1 if the composition is $\max - \min$ or $\max -$product and with all elements equal to 0 if the composition is $\min - \max$.

Each element $t_q(u|v)$ of $T(u|v)$ in (17) determines the operation of $\mathcal{A}$ under the input word u beginning at state q and producing the output word v after $|u| = |v|$ consecutive steps.

For instance, if the FFMach is $\max - \min$ or $\max -$product, the element

$$t_q(u|v) = \bigvee_{q' \in Q} (m_{qq'}(u|v))$$

gives the way of achieving maximal degree of membership under the input word u, beginning at state q and producing the output word v.

 doi:10.20944/preprints202508.1072.v1

We denote by $T_{\mathcal{A}}$ the *complete behavior matrix* of $\mathcal{A}$. It is semi-infinite matrix with $n = |Q|$ rows and with columns $T(u|v)$, $(u|v) \in (X|Y)^*$, computed by (17) and ordered according to the lexicographical order in $(X|Y)^*$, see Table 2.

Table 2. $T_{\mathcal{A}}$ – initial fragment.

	$T(e\|e)$	$T(x_1\|y_1)$	...	$T(u\|v)$	...
q_1	$t_{q_1}(e\|e)$	$t_{q_1}(x_1\|y_1)$	...	$t_{q_1}(u\|v)$	...
q_2	$t_{q_2}(e\|e)$	$t_{q_2}(x_1\|y_1)$	...	$t_{q_2}(u\|v)$	...
...	...	...	...	...	...
q_n	$t_{q_n}(e\|e)$	$t_{q_n}(x_1\|y_1)$	...	$t_{q_n}(u\|v)$	...
length l	$l = 0$	$l = 1$	...	$l = \|\, u \,\|$	...

Let us mention that $t_q(e|e) = 1$ for $i = 1, ..., n$ if FFMach is $\max - \min$ or $\max -$product, $t_q(e|e) = 0$ for $i = 1, ..., n$ if FFMach is $\min - \max$.

4.3. Behavior Matrix

For any FFMach $\mathcal{A}$ its complete input-output behavior matrix $T_{\mathcal{A}}$ is semi-infinite – it has finite number of rows (equal to the number of states in Q) and infinite number of columns. Since $T_{\mathcal{A}}$ is semi-infinite, one can not solve traditional problems – equivalence of states, reduction of states, minimization of states, because all of them explore $T_{\mathcal{A}}$.

The matrix $B_{\mathcal{A}}$ obtained by omitting all columns from $T_{\mathcal{A}}$ that are linear combination of the previous columns is called **behavior matrix** of $\mathcal{A}$.

We may compute the behaviour matrix for FFMach, but the time complexity of the algorithm is exponential. Algorithm 15 in subsection 5.8 extracts the finite behaviour matrix $B_{\mathcal{A}}$ from the complete behavior matrix $T_{\mathcal{A}}$, it implements linear dependency/independency from subsection 4.1.

4.4. Equivalence of States, Reduction, Minimization

$B_{\mathcal{A}}$ captures all the properties of $T_{\mathcal{A}}$ and provides solving equivalence, reduction and minimization problems for FFMachs.

For the FFMach $\mathcal{A} = (X, Q, Y, \mathcal{M})$ the states $q_i \in Q$ and $q_j \in Q$ are called **equivalent** if the input-output behavior of $\mathcal{A}$ when beginning with initial state q_i is the same as its input-output behavior when beginning with initial state q_j. It means that the $i-$th and $j-$th rows are identical in $T_{\mathcal{A}}$ and in $B_{\mathcal{A}}$. Since $T_{\mathcal{A}}$ is semi-infinite, we can not derive equivalence of states from it. In order to solve this problem we first have to extract $B_{\mathcal{A}}$ from $T_{\mathcal{A}}$ and then to find identical rows in $B_{\mathcal{A}}$. Then we left only one of those identical rows.

FFMach $\mathcal{A} = (X, Q, Y, \mathcal{M})$ is in **reduced form** if there does not exist equivalent states in Q.

A FFMach $\mathcal{A} = (X, Q, Y, \mathcal{M})$ is not in **minimal form** if the input-output behavior of $\mathcal{A}$ when it begins with initial state q_i is the same as its input-output behavior when it begins with initial distribution over Q, isolating the state q_i. Formally it means that the $i-$th row of $T_{\mathcal{A}}$ ($B_{\mathcal{A}}$, respectively) is a linear combination of the other rows.

Software from [38] implements this idea: if the $i-$th row of $B_{\mathcal{A}}$ is a linear combination of the other rows of $B_{\mathcal{A}}$, then the FFMach $\mathcal{A}$ is not in minimal form.

Algorithms and software for obtaining the behaviour matrix, as well as for solving reduction and minimization problems are given in subsections 5.8 and 6.4 with working examples in Appendix A.3.

4.5. Linear Optimization

We present optimization problem when the linear objective function is

$$Z = \sum_{j=1}^{n} c_j x_j, \quad c_j \in \mathbb{R}, \; 0 \leq x_j \leq 1, \; 1 \leq j \leq n, \tag{18}$$

 doi:10.20944/preprints202508.1072.v1

with traditional addition and multiplication; $c = (c_1, ..., c_n)$ is the weight (cost) vector and the function (18) is subject to FLSEs or FLSIs as constraint

$$A *_{BL} X = B, or A *_{BL} X \leq B, or A *_{BL} X \geq B. \tag{19}$$

The aim is to minimize or maximize Z subject to one of the constraints in (19).

The linear optimization problem was investigated first for $\max - \min$ composition by Fang and Li [49], Guu and Wu [50], Peeva [41]; for $\max -$product composition by Loetamphong and Fang [51], Peeva and co-authors [52]; for $BL-$algebras by Peeva and Petrov [53]. If the objective function becomes $Z = \max_{j=1}^{n} \min(c_j, x_j)$ with $c_j \in [0, 1]$ the so called latticized linear programming problem was studied by Wang et al [54].

Since the solution set of (19) is non-convex, traditional linear programming methods cannot be applied. In order to find the optimal solution for (18), the problem is converted into a $0-1$ integer programming problem. Fang and Li in [49] solved this $0-1$ integer programming problem by branch and bound method with jump-tracking technique. Improvements of this method are proposed in [55] by providing fixed initial upper bound for the branch and bound part and with update of this initial upper bound. Conventional investigations [49], [50], [51], [55] begin with various attempts to find some feasible solutions, then to find an arbitrary solution of the optimization problem subject to constraint from (19) and proceed with it in the solution set for search of minimal solutions. The procedures are clumsy, the main obstacle is how to solve (19).

The algorithms and software from [38] proceeds for solving optimization problems, among them are:

i) Minimize the linear objective function (18), subject to FLSEs constraints, when the composition is $\max - \min$, $\max -$product or $\max -$Łukasiewicz;

ii) Maximize the linear objective function (18), subject to FLSEs, when the composition is $\max - \min$, $\max -$product or $\max -$Łukasiewicz;

iii) Minimize or maximize the linear objective function (18), subject to FLSEs constraints, when the composition is $\min - \max$.

For clarity of exposition we present the ideas for $\max - \min$ and $\max -$product case.

4.6. Minimize the Linear Objective Function, Subject to FLSEs Constraints, When the Composition is $\max - \min$ *or* $\max -$*Product Case.*

We present how to minimize the linear objective function (18), subject to constraints: a fuzzy linear system of equations with $\max - \min$ or $\max -$ product composition

$$A \circ X = B, \tag{20}$$

where $A = (a_{ij})_{m \times n}$ with $a_{ij} \in [0, 1]$ stands for the matrix of coefficients, $X = (x_i)_{n \times 1}$, $x_i \in [0, 1]$ stands for the matrix of unknowns, $B = (b_i)_{m \times 1}$, $b_i \in [0, 1]$ is the right-hand side of the system, the $\max - \min$ or $\max -$ product composition is written as $\circ$ and $c = (c_1, ..., c_m)$ is the weight (cost) vector.

We apply the software from [38] for computing the greatest solution and all minimal solutions of the consistent system $A \circ X = B$. Then we solve the linear optimization problem, first decomposing Z into two vectors $Z' = (c'_1, c'_2, ..., c'_n)$ and $Z'' = (c''_1, c''_2, ..., c''_n)$, such that $Z = Z' + Z''$ and:

$$c_i = c'_i + c''_i, \quad \text{for each} \quad i = 1, ..., n,$$

$$c'_i = \begin{cases} c_i, & \text{if } c_i \geq 0, \\ 0, & \text{if } c_i < 0 \end{cases}, \tag{21}$$

 doi:10.20944/preprints202508.1072.v1

$$c_i'' = \begin{cases} 0, & \text{if } c_i \geq 0, \\ c_i, & \text{if } c_i < 0 \end{cases}. \tag{22}$$

Now the original problem: to minimize Z subject of constraints (20), is split into two problems, namely to minimize both

$$Z' = \sum_{i=1}^{m} c_i' x_i \tag{23}$$

and

$$Z'' = \sum_{i=1}^{m} c_i'' x_i \tag{24}$$

with constraints (20).

Z takes its minimum when Z' takes its minimum and Z'' takes its maximum. Hence for the problem (24) the optimal solution is $\hat{X} = (\hat{x_1}, ..., \hat{x_n}) = X_{gr}$, and for the problem (23) the optimal solution $\check{X} = (\check{x_1}, ..., \check{x_n})$ is among the minimal solutions of the system (20). In this case the optimal solution of the problem is $X^* = (x_1^*, ..., x_n^*)$, where

$$x_i^* = \begin{cases} \hat{x_i}, & \text{if } c_i < 0 \\ \check{x_i}, & \text{if } c_i \geq 0 \end{cases}. \tag{25}$$

4.7. Maximize the Linear Objective Function, Subject to Constraints $A \circ X = B$, When the Composition is $\max - \min$ *or* $\max -$*Product*

In order to maximize the linear objective function Z, we again split it, but now for (24) the optimal solution is among the minimal solutions of the system (20), for (23) the optimal solution is X_{gr}. In this case the optimal solution is $X^* = (x_1^*, ..., x_n^*)$, where

$$x_i^* = \begin{cases} \check{x_i}, & \text{if } c_i < 0 \\ \hat{x_i}, & \text{if } c_i \geq 0 \end{cases}. \tag{26}$$

4.8. Algorithm for Finding Optimal Solutions

We propose the algorithm for finding optimal solutions if the linear objective function is Z and the composition is $\max - \min$ or $\max -$product for the constraints (20).

In any of these cases the optimal value is

$$Z^* = \sum_{i=1}^{n} c_i x_i^* \tag{27}$$

The algorithm for finding optimal solutions is based on the above results.

Algorithm 4 Algorithm for finding optimal solutions

Step 1: Enter the matrices $A_{m\times n}$, $B_{m\times 1}$ and the cost vector $C_{1\times n}$.
Step 2: Implement the software from [38] to establish consistency of the system (20). If the system is inconsistent - Exit.
Step 3: Implement the software from [38] to compute X_{gr} and all minimal solutions of (20).
Step 4: If finding $Z_{\max}$ go to Step 6.
Step 5: For finding $Z_{\min}$ compute x_i^*, $i = 1, ..., n$ according to (25). Go to Step 7.
Step 6: For finding $Z_{\max}$ compute x_i^*, $i = 1, ..., n$ according to (26).
Step 7: Compute the optimal value according to (27).

Algorithms and software for the above described linear optimization problems are given in subsections 5.9 and 6.5 with working examples in Appendix A.4.

 doi:10.20944/preprints202508.1072.v1

5. Algorithms

In this section, we present algorithms that have been developed over the years and are used in the software [38]. These algorithms cover fuzzy matrix compositions, inverse problem resolution, linear dependence and independence, fuzzy optimization, and finite fuzzy machines. They are designed to support a variety of fuzzy algebras, including max-min, min-max, max-product, and all the others defined in Definitions 1 - 3.

However, in order to keep the exposition focused and accessible, we restrict attention in this chapter to the max-min case. This is justified not only by its historical role in the development of fuzzy relational methods, but also by the fact that it forms the basis of many practical applications. Other compositions introduced in Definitions 1 - 3 will be demonstrated in the following Chapter 6.

The inverse problem for fuzzy linear systems is particularly challenging when computing the complete solution set. The algorithms presented here incorporate structural optimizations to address this challenge while preserving the mathematical properties of the underlying systems.

5.1. Fuzzy Matrix Compositions

Direct problem in FRC means fuzzy matrix composition. According to Definitions 1 - 3, fuzzy matrix composition is defined by applying two fuzzy operations. Similarly to conventional matrix multiplication, the general procedure computes each entry c_{ij} of the resulting matrix C from the row i of matrix A and column j of matrix B.

Algorithm 5 Generic Fuzzy Matrix Composition

Step 1: Input fuzzy matrices A, B, and the pair of fuzzy operations
Step 2: For each entry c_{ij} in the resulting matrix C:
Step 3: For each index k:
Step 4: Apply the first operation to a_{ik}, b_{kj}
Step 5: Aggregate all intermediate results using the second operation
Step 6: Output matrix C as the composition between A and B

Complexity: Algorithm 5 computes each entry c_{ij} by evaluating k pairs and aggregating the result, leading to a total of mnk basic operations. This gives a complexity of $O(mnk)$.

This algorithm serves as a general template for all supported compositions. Each specific case, such as max-min, min-max, max-product, or compositions based on fuzzy implications is implemented as an alias function that calls this general routine with the corresponding pair of fuzzy operations.

5.2. Inverse Problem Resolution for Fuzzy Linear Systems of Equations

The inverse problem in FRC consists of solving an FLSEs of the form $A\,X = B$ or $A *_{BL} X = B$ or $A \rightarrow_{BL} X = B$ where A and B are known fuzzy matrices, X is the unknown, and the composition is defined as in Definitions 1 - 3. Solving this problem means finding the complete solution set $\mathbb{X}$ under the selected fuzzy composition.

While direct composition is computationally straightforward, solving the inverse problem, especially for obtaining the complete solution set, is significantly more complex. In what follows, we focus on the classical case of max-min composition, where most theoretical and algorithmic developments are concentrated.

Over the years, several different algorithms have been developed and implemented for solving the inverse problem in Fuzzy Calculus Core. These earlier methods, based on the algebraic-logical framework and analytical constructions, are described in detail in Section 3. The current version of the software incorporates an optimized algorithm that builds on these foundational techniques.

The method described below represents the latest and most efficient version currently implemented in the software. It is designed to compute the greatest solution and recursively generate all minimal solutions through domination and recursive search.

 doi:10.20944/preprints202508.1072.v1

Algorithm Overview

The algorithm for solving FLSEs $A \bullet X = B$ under max-min composition is divided into the following stages:

- **Stage 1: Initializations**
- **Stage 2: Find the greatest solution**
- **Stage 3: Domination**
- **Stage 4: Recursive extraction of all lower solutions**

Stage 1: Initializations

Algorithm 6 Stage 1: Initializations

Step 1: Enter fuzzy matrix A and fuzzy vector B
Step 2: Initialize help matrix H with the same size as A, filled with 0
Step 3: **for** each a_{ij} in A **do**
Step 4: **if** $a_{ij} \geq b_i$ **then**
Step 5: Set $h_{ij} = b_i$
Step 6: **end if**
Step 7: **end for**

Complexity: Algorithm 6 has complexity $O(mn)$, where m and n are the number of rows and columns, respectively, of the input matrix A.

Stage 2: Find the greatest solution

Earlier algorithms for computing the greatest solution relied on slow residuum-based expressions or complicated algorithms based on S, E, G, and H elements, presented in Section 3.4.

The current implementation avoids implication operators entirely. Instead, it computes the greatest solution directly from the help matrix using column operations. The consistency of the system is checked during the same pass, marking all covered rows without additional matrix traversal. This leads to a significant improvement in performance and simplicity.

Algorithm 7 Stage 2: Find the greatest solution

Step 1: Initialize vector X with ones representing the greatest solution
Step 2: Initialize vector IND with zeros to mark system consistency
Step 3: **for** each j in X **do**
Step 4: Set x_j to the minimum of all h_{ij} such that $h_{ij} = b_i$ and $a_{ij} \neq b_i$
Step 5: Mark all IND_i with 1 where $h_{ij} = b_i$
Step 6: Set $h_{ij} = 0$ for all i where $h_{ij} > x_j$
Step 7: **end for**
Step 8: **if** there exists any 0 in IND **then**
Step 9: System is inconsistent.
Step 10: **else**
Step 11: System is consistent, its greatest solution is X
Step 12: **end if**

Complexity: Algorithm 7 requires at most mn comparisons in its worst-case scenario, giving $O(mn)$ complexity. This algorithm has better scenarios, where the solution can be found early for each variable, resulting in the best-case performance of $O(n)$.

Stage 3: Domination

Definition 8. *[15] Let $h_l = (h_{lj})$ and $h_k = (h_{kj})$ be the $l-$th and the $k-$th rows, respectively, in the help matrix H. If for each j, $1 \leq j \leq n$, $h_{lj} \leq h_{kj}$, then:*

- *h_l is said to be a dominant row to h_k in H*

 doi:10.20944/preprints202508.1072.v1

- *h_k is redundant row and can be removed from H*

In earlier implementations, dominance was analyzed using a separate dominance matrix. In the current version, this step is evaluated directly over the help matrix.

Algorithm 8 Stage 3: Domination

```
Step 1:  for each row i in H do
Step 2:      for each row j ≠ i in H do
Step 3:          if row j dominates row i then
Step 4:              Mark row i for deletion
Step 5:              Break
Step 6:          else if row i dominates row j then
Step 7:              Mark row j for deletion
Step 8:          end if
Step 9:      end for
Step 10: end for
Step 11: Remove all marked rows from H
```

Complexity: Algorithm 8 is giving a worst-case complexity of $O(m^2 n)$. In favorable cases where domination is detected early, many comparisons are skipped via early termination, resulting in best-case performance of $O(mn)$.

Stage 4: Recursive extraction of all lower solutions

The current implementation uses a recursive procedure that incrementally builds each solution from valid components in the help matrix. At every step, the algorithm maintains a set of uncovered rows and explores only those elements that contribute to the current potential solution. Branches that cannot lead to a complete solution are skipped early. This method avoids full enumeration of all combinations, and provides a conceptually simpler and more memory-efficient procedure.

Algorithm 9 Stage 4: Recursive extraction of all lower solutions

```
Step 1:  Initialize empty set of solutions 𝕏
Step 2:  Initialize candidate solution X with zeros
Step 3:  Initialize marking vector M with zeros, meaning no rows are initially marked
Step 4:  Start recursion with the first unmarked row i = 1
Step 5:  for each column j such that h_ij > 0 do
Step 6:      Set x_j = h_ij
Step 7:      Mark all rows k such that h_kj > 0 by setting m_k = 1
Step 8:      if all rows are marked then
Step 9:          if X is not dominated by any other Y ∈ 𝕏 then
Step 10:             Remove all Y ∈ 𝕏 such that X ≤ Y
Step 11:             Add X to 𝕏
Step 12:         end if
Step 13:     else
Step 14:         Let i′ be the next unmarked row in M
Step 15:         Repeat this loop recursively with updated X, M, and i′
Step 16:     end if
Step 17: end for
```

 doi:10.20944/preprints202508.1072.v1

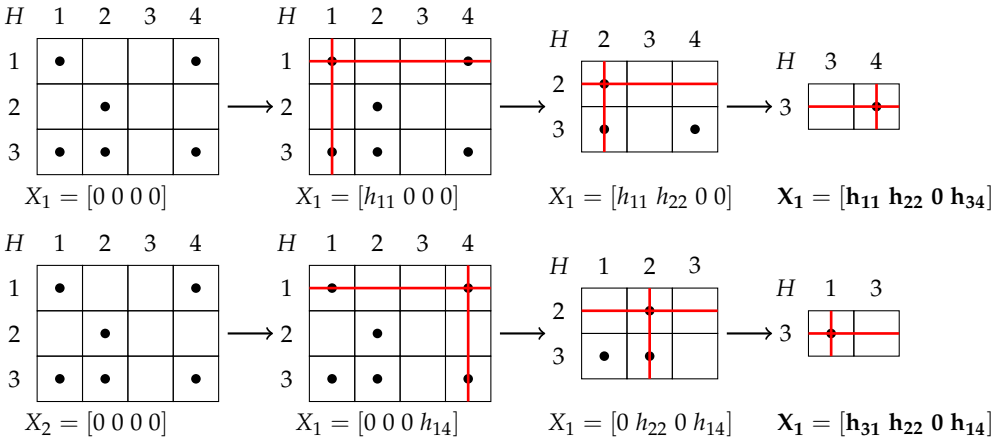

Figure 1. Visual representation of Algorithm 9

Complexity: The time complexity of the recursive extraction of all lower solutions from H is exponential, but in this case, it depends on the number of non-zero elements in the help matrix, which itself reflects the number of possible lower solutions. More precisely, in the worst case, the number of recursive paths can grow to $O(d^m)$, where d is the average number of non-zero entries per row, and m is the number of rows in H which may be smaller than in A, due to the domination step. In practice, however, the algorithm is highly efficient due to early pruning of incomplete branches and the reduced size of H after domination.

5.3. Complete Algorithm for Solving $A \bullet X = B$

The previous procedures are combined into a complete algorithm for solving FLSEs $A \bullet X = B$.

Algorithm 10 Solving the inverse problem for $A \bullet X = B$

Step 1: Initialize the help matrix H using Algorithm 6
Step 2: Compute the greatest solution and check consistency using Algorithm 7
Step 3: **if** the system is consistent **then**
Step 4: Perform row domination on H using Algorithm 8
Step 5: Extract all lower solutions using Algorithm 9
Step 6: **end if**

Complexity: The total complexity is dominated by the recursive enumeration of all lower solutions (Algorithm 9), which is exponential in the number of rows in the help matrix. All preceding steps—help matrix construction, consistency check, and row domination—run in $O(mn)$ or $O(m^2n)$ time.

The presented algorithm can be extended to other types of systems and compositions. Similar procedures are implemented in [38] for all composition operators defined in Definition 1, illustrating the potential for both algorithmic and theoretical generalizations of this approach.

5.4. Inverse Problem Resolution for Fuzzy Linear Systems of $\geq$ Inequalities

While FLSEs of the form $A \bullet X = B$ have been widely studied, FLSIs such as $A \bullet X \geq B$ have received comparatively little attention in the literature. To the best of our knowledge, no other general-purpose software tools for solving such systems exist.

The algorithms presented in the previous section for solving FLSEs can be extended to handle systems of inequalities with only minor modifications. In this section, we focus on the case $A \bullet X \geq B$, where the greatest solution is trivial if it exists, but lower solutions must still be extracted recursively. We cannot assume that the corresponding system of equations has a solution, there are cases where the inequality system is consistent, even though the equation system is not. We highlight the key differences in structure and interpretation compared to the case of equations.

 doi:10.20944/preprints202508.1072.v1

Algorithm 11 Solving the inverse problem for $A \bullet X \geq B$

Step 1: Use Algorithm 6 to find the help matrix H
Step 2: Initialize vector IND with zeros
Step 3: Mark all IND_i with 1 where $h_{ij} = b_i$
Step 4: **if** any element of IND is still zero **then**
Step 5: The system is inconsistent
Step 6: **else**
Step 7: Set X with ones for the greatest solution
Step 8: Apply Algorithm 8 to eliminate dominated rows from H
Step 9: Use Algorithm 9 to compute all lower solutions
Step 10: **end if**

Complexity: The computational complexity is identical to that of the algorithms described in Section 5.2.

5.5. Inverse Problem Resolution for Fuzzy Linear Systems of $\leq$ Inequalities

FLSIs of the form $A \bullet X \leq B$ are structurally simpler than equations or $\geq$ inequalities. A trivial lower solution X of all zeros always exists, and no recursive extraction is required. The problem reduces to finding the greatest solution that satisfies all inequality constraints. This is performed using the same method as in the equality case, with minor simplifications.

Algorithm 12 Solving the inverse problem for $A \bullet X \leq B$

Step 1: Use Algorithm 6 to find the help matrix H
Step 2: Initialize vector IND with zeros
Step 3: Mark all IND_i with 1 where $h_{ij} = b_i$
Step 4: **if** any element of IND is still zero **then**
Step 5: The system is inconsistent
Step 6: **else**
Step 7: Use Algorithm 7 to find the greatest solution X
Step 8: Set X zeros as the trivial lower solution
Step 9: **end if**

Complexity: The complexity is significantly lower than in Section 6.2. No domination or recursion is required. We are left with just the greatest solution, which is computed in $O(mn)$ time.

5.6. Fuzzy Linear Combination of Vectors

A fuzzy vector B is said to be a linear combination of the columns of a matrix A under a fixed composition (e.g. max-min) if there exists a fuzzy vector X such that $A \bullet X = B$. This is equivalent to solving the inverse problem.

To determine whether such a combination exists, we reuse the algorithms for computing the greatest solution described in Section 5.2. Since we are only interested in the existence of a solution, it is sufficient to validate that the system is consistent i.e. there is no need to extract all lower solutions.

Algorithm 13 Check if B is a fuzzy linear combination of the columns of A

Step 1: Enter fuzzy matrix A and vector B
Step 2: Check if the system $A \bullet X = B$ is consistent according to Algorithm 7
Step 3: **if** the system is consistent **then**
Step 4: B is a fuzzy linear combination of the columns in A
Step 5: **else**
Step 6: B is not a fuzzy linear combination of the columns in A
Step 7: **end if**

Complexity: The complexity is the same as the consistency check in Algorithm 7.

 doi:10.20944/preprints202508.1072.v1

5.7. Fuzzy Linear Independence

A set of fuzzy vectors $\{A_1, A_2, \ldots, A_n\}$ is said to be linearly independent if no vector in the set can be expressed as a fuzzy linear combination of the others vectors under a fixed composition (e.g. max-min).

Our implementation tests linear independence by iteratively checking, each column A_i in a fuzzy matrix A, for a linear combination of the remaining columns. This is done using the procedure from Algorithm 13.

Algorithm 14 Check if the columns of A are fuzzy linearly independent

Step 1: Enter fuzzy matrix A
Step 2: Initialize empty list D to hold dependent columns
Step 3: **for** each column A_i in A **do**
Step 4: Let A_i^* be the matrix A without column A_i
Step 5: Check if A_i is a fuzzy linear combination of A_i^* using Algorithm 13
Step 6: **if** A_i is a linear combination of the other columns **then**
Step 7: Add index i to list D
Step 8: **end if**
Step 9: **end for**
Step 10: **if** D is empty **then**
Step 11: The columns of A are fuzzy linearly independent
Step 12: **else**
Step 13: D holds the list of all dependent columns
Step 14: **end if**

Complexity: The algorithm performs n consistency checks, each with complexity $O(mn)$. Therefore, the total complexity is $O(mn^2)$.

5.8. Finite Fuzzy Machines

This subsection focuses on computing the behavior of max-min FFMach, see subsection 4.2. The behavior matrix is obtained by composing all reachable transitions iteratively until stabilization. The process terminates automatically when further compositions no longer change the result.

Here we present the algorithms used in [38] for computing the behavior matrix, and for two types of postprocessing: reduction and minimization presented in Section 4.2.

Algorithm 15 Compute behavior matrix B of a FFMach

Step 1: Enter the initial set of fuzzy transition matrices $\mathcal{M} = \{M_1, \ldots, M_k\}$
Step 2: Initialize the behavior matrix B with a single column of ones
Step 3: **repeat**
Step 4: Let n be the current number of columns in B
Step 5: **for** $j = 1$ to n **do**
Step 6: **for** each $M_i \in \mathcal{M}$ **do**
Step 7: Compute $b = M_i \bullet B(:, j)$
Step 8: **if** b is not a linear combination of the columns in B **then**
Step 9: Append b as a new column to B
Step 10: **end if**
Step 11: **end for**
Step 12: **end for**
Step 13: **until** no new columns are added to B

Complexity: Let N denote the final number of behavioral vectors in the matrix B. At each iteration, up to kn new candidates are generated, where k is the number of fuzzy transition matrices and n is the current number of columns in B. Each new vector is tested for fuzzy linear combination against the existing ones, using Algorithm 13, which has complexity $O(mn)$ for vectors of length m. Assuming

 doi:10.20944/preprints202508.1072.v1

no vector is rejected as dependent, the total number of candidates grows quadratically and each test becomes increasingly more expensive. The resulting worst-case complexity is approximately $O(kmN^3)$.

Algorithm 16 Reduce the FFMach

Step 1: Enter the behavior matrix B
Step 2: Identify all duplicate columns in the transposed matrix B^T
Step 3: Remove the corresponding rows from the original matrix B

Complexity: The software [38] uses MATLAB's built-in `unique` function, which is based on lexicographical sorting. The overall complexity is approximately $O(m \log m)$, where m is the number of rows.

Algorithm 17 Minimize states of the FFMach

Step 1: Enter the behavior matrix B
Step 2: Use Algorithm 14 to identify all dependent columns in the transposed matrix B^t
Step 3: Remove the corresponding rows from the original matrix B

Complexity: The algorithm internally applies Algorithm 14, and therefore has the same complexity.

5.9. Fuzzy Optimization

Let us remind (see Section 4.5) that the linear objective function

$$Z = \sum_{j=1}^{n} c_j x_j$$

is subject to fuzzy constraints of the form

$$A \bullet X = B, \quad A \bullet X \leq B, \quad \text{or} \quad A \bullet X \geq B.$$

The algorithm splits the objective function into two parts: Z', corresponding to components with $c_i \geq 0$, and Z'', corresponding to components with $c_i < 0$. When minimizing Z, the variables in Z' are selected among all lower solutions, while the variables in Z'' are taken from the greatest solution of the constraint system. When maximizing - variables in Z' are taken from the greatest solution, and those in Z'' are selected among the lower solutions. The result is a combined vector X formed by selecting components from the appropriate solutions.

Since any value between a lower and an upper solution is itself a solution of the fuzzy system (see Section 3.2), the obtained X is also a valid solution of the system.

Algorithm 18 Fuzzy optimization under max–min FLSE constraints

Step 1: Enter fuzzy system $A \bullet X \star B$, where $\star \in \{=, \leq, \geq\}$
Step 2: Enter objective weights C and optimization goal: `minimize` or `maximize`
Step 3: Solve the constraint system using Algorithm 10, 12, or 11, depending on $\star$
Step 4: Split the objective function into Z' (for $c_j \geq 0$) and Z'' (for $c_j < 0$)
Step 5: **if** goal is `minimize` **then**
Step 6: Take the corresponding components for Z'' from the greatest solution
Step 7: Evaluate $Z = Z' + Z''$ for all lower solutions
Step 8: For Z', select the solution that minimizes Z
Step 9: **else** if goal is `maximize`
Step 10: Take the corresponding components for Z' from the greatest solution
Step 11: Evaluate $Z = Z' + Z''$ for all lower solutions
Step 12: For Z'', select the solution that maximizes Z
Step 13: **end if**

 doi:10.20944/preprints202508.1072.v1

Complexity: The algorithm internally applies Algorithm 10, 12, or 11, depending on the constraint type, and inherits their respective complexity.

6. Software

This chapter presents the software implementation of the Fuzzy Calculus Core package [38]. The software provides tools for working with fuzzy matrices, solving fuzzy relational equations and inequalities, performing fuzzy optimization, and reduction and minimization of finite fuzzy machines.

The architecture of the system reflects the algorithmic structure introduced in Chapter 5. Each algorithm described there corresponds to a specific implementation in the respective module. Examples illustrating the use of each functionality are provided in Appendix A.

6.1. Architecture

The software is structured as a modular MATLAB package with a clear separation between low-level data representation and high-level problem-solving routines. Its architecture reflects the logical layering of the algorithms described in Chapter 5, and supports reuse, extension, and composition.

At the core of the system lies the `fuzzyMatrix` module, which implements data structures and operations for fuzzy matrices and vectors. On top of this base layer are modules for solving fuzzy linear systems of equations and inequalities (`fuzzySystem`), for solving optimization problems (`fuzzyOptimizationProblem`), and for finding behavior, minimizing and reducing fuzzy finite machines (`fuzzyMachine`).

An overview of the module dependencies is presented in Figure 2.

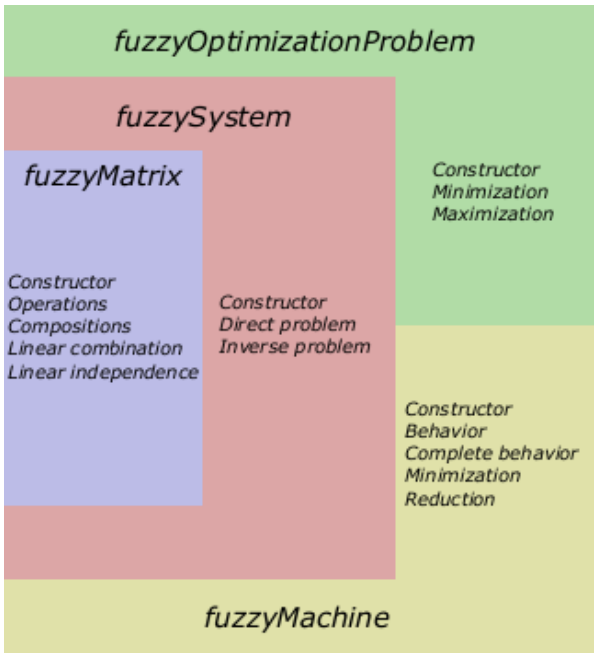

Figure 2. Architecture of the Fuzzy Calculus Core software.

The system supports a variety of fuzzy matrix compositions, as defined in Definitions 1–3. These are:

- Max-min
- Min-max
- Max-product
- Max-ϵ
- Łukasiewicz composition
- Min-$\rightarrow_G$ (Gödel implication)
- Min-$\rightarrow_P$ (Product implication)

 doi:10.20944/preprints202508.1072.v1

- Min-$\to_L$ (Łukasiewicz implication)

All compositions are supported uniformly across the system: they can be used not only for direct matrix operations, but also in solving both the direct and the inverse problems, as well as in fuzzy optimization and fuzzy finite machines.

6.2. The `fuzzyMatrix` Module

The `fuzzyMatrix` module provides the core data structure for representing fuzzy matrices. It is implemented as a MATLAB class that extends the built-in `double` type, enabling native support for arithmetic operations while enforcing fuzzy semantics.

A `fuzzyMatrix` object can be initialized in one of the following ways:

- Without arguments, creating an empty fuzzy matrix.
- With a single argument, which must be a real matrix with values in the $[0, 1]$ interval.
- With two scalar arguments, interpreted as dimensions, which creates a zero-filled matrix of the corresponding size.

Internally, all matrix elements are stored as standard numerical arrays. The class overrides basic operations such as addition and subtraction to ensure that results remain within the fuzzy domain.

In addition to standard elementwise operations and transposition, the module implements a wide range of fuzzy matrix compositions:

- `maxmin`, `minmax`, `maxprod`
- `minalpha`, `maxepsilon`, `mindiamond`
- `godel`, `goguen`, `lukasiewicz`, `maxlukasiewicz`
- `minprobabilistic`, `minbounded`, `maxdelta`, `maxgama`

Each of these is implemented through the general-purpose method `fcompose`, which composes two fuzzy matrices using a pair of operations.

The module also provides methods for fuzzy linear combination (`is_lincomb`) and fuzzy linear independence (`is_linindep`), allowing users to test whether a given vector can be expressed as a fuzzy combination of others under a specified composition.

Examples illustrating the use of this module can be found in Appendix A.1.

6.3. The `fuzzySystem` Module

The `fuzzySystem` module provides a high-level abstraction for solving fuzzy relational problems of the form $=$, $\leq$, and $\geq$. Both the direct and the inverse problems are supported, under all compositions listed in Section 6.1.

The module acts as a front-end interface to the lower-level algorithms described in Chapter 5. A `fuzzySystem` object is constructed by specifying the composition type and the equality/inequality type, together with two `fuzzyMatrix` instances. It delegates the computation to the appropriate method depending on the composition.

For direct problems (computing B), the module internally calls the `fcompose` method from the `fuzzyMatrix` class. For inverse problems the underlying logic follows Algorithms 10, 11 and 12, depending on the selected system type.

The main interface function `solve_inverse` returns a structure with standardized fields, which include:

- `gr` – a set of upper solutions, which contains either the greatest solution or all upper solutions, depending on the composition
- `low` – a set of lower solutions, which contains either the lowest solution or all lower solutions;
- `exist` – a boolean flag indicating whether the system is compatible or incompatible.

In certain situations, it is sufficient to check whether a system is consistent or to obtain a single representative solution. In such cases, the algorithm computes only the greatest (resp. lowest) solution, avoiding the extraction of the full solution set. This mode is significantly more efficient and is based

 doi:10.20944/preprints202508.1072.v1

on Algorithm 7, which also serves as a consistency check. When full enumeration is needed, setting `full=true` triggers the recursive Algorithm 10 form Section 5.

All input matrices are instances of `fuzzyMatrix`, and all computed results are returned in the same class.

Examples of usage are provided in Appendix A.2.

6.4. The *fuzzyMachine* Module

The `fuzzyMachine` module implements operations for finding behavior, reduction and minimization of FFMach. A FFMach is defined by a set of fuzzy matrices, one for each input/output pair of words. These matrices are stored in a cell array and passed to the constructor. The user specifies the composition type, the maximum word length to consider, and whether to return the full behavior matrix, a reduced or a minimized one.

The main method `find_behavior()` applies the matrices to compute the machine's behavior. If postprocessing is enabled, the resulting behavior matrix is reduced or minimized by eliminating redundant columns or states. The minimization procedure removes states whose behavior vectors can be expressed as fuzzy linear combinations of others.

All compositions listed in Section 6.1 are supported. The final behavior matrix is stored internally and can be accessed or exported for further analysis.

Examples are provided in Appendix A.3.

6.5. The *fuzzyOptimizationProblem* Module

The `fuzzyOptimizationProblem` class provides a unified interface for solving fuzzy linear optimization problems of the form described in Section 4.5. A problem instance is created by specifying:

- an objective weight (cost) vector C;
- a system of constraints defined by a `fuzzySystem` object and supporting all of the `fuzzySystem` options such as compositions, and inequalities

Internally, the functions first solve the inverse problem for the `fuzzySystem` object to obtain the complete solutions set for the system of constraints. Then, it either evaluates the objective function on all solutions, according on the optimization goal.

The methods `minimize()` and `maximize()` fill the `object_solution` and the `object_value` properties of the `fuzzyOptimizationProblem` object containing respectively, the optimal fuzzy vector `x` and the objective value `z` for the optimization problem. All properties of the system of constrains are held in the `fuzzySystem` object (e.g., consistency status or complete solutions set).

Examples of usage are provided in Appendix A.4.

7. Conclusions

In 1977 Peeva finished her PhD Thesis on categories of stochastic machines. The main attention was paid on computing behavior, establishing equivalence of states and equivalence of stochastic machines, as well as on reduction and minimization. All these problems were solved for stochastic machines, using linear algebra: they require solving linear system of equations with traditional algebraic operations, establishing linear dependence or linear independence of vectors, Noetherian property. After 1980 she was interested in similar problems, but for finite fuzzy machines. Nevertheless that stochastic machine seems to be similar to fuzzy machine, the main obstacle was that linear algebra and fuzzy algebras are completely unrelated. In order to investigate fuzzy machines, two types of algebras, called max $-$ min algebra and max $-$product algebra have been developed. The role played by these algebras in the theory of max $-$ min and max $-$product FFMachs is supposed to be the same as that played by linear algebra in the theory of stochastic machines. But these algebras propose tremendous manipulations for solving problems for FFMachs. Supplementary, there are several issues in [56], that are either incomplete or confused. This motivated the author to develop theory, algorithms and software for FFMachs. Obviously developing theory and software for solving problems for

 doi:10.20944/preprints202508.1072.v1

FFMachs requires first to develop theory and software for FRC and then to implement it for FFMachs. At this time these problems were open and she intended first to develop method, algorithm and software for solving FLSEs when the composition is max − min, then to implement it to FFMachs. In 1980 she had no idea whether the solution exists and how long it will take her to solve the problems. Just now, after 40 years, we have positive answers to all these questions.

She worked on this subject with great pleasure during the years with her bachelor, master and PhD students, now Prof. Dr. Yordan Kyosev, Assoc. Prof. Dr. Zlatko Zahariev, Dobromir Petrov and Ivailo Atanassov to whom she would like to express deep and heartfelt gratitude.

Zlatko Zahariev joined this work in 2005 as a PhD student under the supervision of Prof. DSc. Ketty Peeva. His main focus was on developing algorithmic versions the methods, creating their software implementation, and contributing to the ongoing effort toward designing unified algorithms for solving the problems described in this paper. He considers himself fortunate to have the opportunity to work under Prof. DSc. Peeva's guidance, and is deeply grateful to her for the years of meaningful and fulfilling scientific collaboration.

The authors thank the journal team for the special offer to write this article.

Appendix A. Examples

Appendix A.1. Examples for the `fuzzyMatrix` module

This section provides usage examples for the `fuzzyMatrix` module, including initialization, basic operations, and several types of fuzzy matrix composition.

Max-min composition of two fuzzy matrices

```
>> A = fuzzyMatrix(rand(3))
A =
  3×3 fuzzyMatrix:
  double data:
    0.9649    0.9572    0.1419
    0.1576    0.4854    0.4218
    0.9706    0.8003    0.9157

>> B = fuzzyMatrix(rand(3))
B =
  3×3 fuzzyMatrix:
  double data:
    0.7922    0.0357    0.6787
    0.9595    0.8491    0.7577
    0.6557    0.9340    0.7431

>> C = maxmin(A,B)
C =
  3×3 fuzzyMatrix:
  double data:
    0.9572    0.8491    0.7577
    0.4854    0.4854    0.4854
    0.8003    0.9157    0.7577
```

Max-ε composition of two fuzzy matrices

```
>> C = maxepsilon(A,B)
C =
  3×3 fuzzyMatrix:
  double data:
    0.9595    0.9340    0.7431
```

 doi:10.20944/preprints202508.1072.v1


```
    0.9595    0.9340    0.7577
    0.9595    0.9340         0
```

Test for fuzzy linear combination with min-max composition

```
>> coeffs = fuzzyMatrix(rand(3,1))
coeffs =
  3×1 fuzzyMatrix:
  double data:
    0.0344
    0.4387
    0.3816

>> X = minmax(A, coeffs)
X =
  3×1 fuzzyMatrix:
  double data:
    0.3816
    0.1576
    0.8003

>> is_lincomb('minmax', A, X)
ans =
  3×1 fuzzyMatrix:
  double data:
         0
         0
    0.3816
```

Test for fuzzy linear dependence with min-max composition

```
>> A_extended = fuzzyMatrix([double(A) double(X)])
A_extended =
  3×4 fuzzyMatrix:
  double data:
    0.9649    0.9572    0.1419    0.3816
    0.1576    0.4854    0.4218    0.1576
    0.9706    0.8003    0.9157    0.8003

>> is_linindep(A_extended, 'minmax', true)
ans =
     4
```

Appendix A.2. Examples for the `fuzzySystem` module

This section provides example uses of the `fuzzySystem` class, illustrating how to construct and solve fuzzy relational systems under various compositions and inequality types.

Solving A X = B using $\odot$ composition [52]

```
>> A = fuzzyMatrix([0.00, 0.20, 0.05, 0.00, 0.40, 0.00;
                    0.10, 0.60, 0.30, 0.00, 0.20, 0.20;
                    0.80, 0.48, 0.24, 0.48, 0.00, 0.00;
                    0.30, 0.00, 0.00, 0.40, 0.80, 0.15;
                    0.00, 0.00, 0.12, 0.20, 0.48, 0.10;
                    0.50, 0.30, 0.00, 0.10, 0.60, 0.00]);
```

 doi:10.20944/preprints202508.1072.v1


```
>> B = fuzzyMatrix([0.10; 0.30; 0.24; 0.20; 0.12; 0.15]);

>> S = fuzzySystem('maxprod', A, B, [], true)
S =
  fuzzySystem with properties:
    composition: 'maxprod'
              a: [6×6 fuzzyMatrix]
              b: [6×1 fuzzyMatrix]
              x: [0×0 fuzzyMatrix]
           full: 1
   inequalities: 0

>> S.solve_inverse();

>> S.x
ans =
  struct with fields:
      rows: 6
      cols: 6
      help: [4×6 fuzzyMatrix]
        gr: [6×1 fuzzyMatrix]
       ind: [6×1 double]
     exist: 1
 dominated: [6 3]
 help_rows: 4
       low: [6×3 fuzzyMatrix]

>> S.x.exist
ans =
  logical
    1

>> S.x.gr
ans =
  6×1 fuzzyMatrix:
  double data:
    0.3000
    0.5000
    1.0000
    0.5000
    0.2500
    1.0000

>> S.x.low
ans =
  6×3 fuzzyMatrix:
  double data:
         0         0         0
    0.5000    0.5000         0
    1.0000         0    1.0000
    0.5000         0         0
         0    0.2500    0.2500
         0         0         0
```

 doi:10.20944/preprints202508.1072.v1

Fast consistency check for the same system

```
>> S.full = false;

>> S.solve_inverse();

>> S.x
ans =
  6×1 fuzzyMatrix:
  double data:
    0.3000
    0.5000
    1.0000
    0.5000
    0.2500
    1.0000
```

Solving A X $\leq$ B using $\odot$ composition and the same A and B

```
>> S.inequalities = -1;

>> S.solve_inverse();

>> S.x
ans =
  struct with fields:
    rows: 6
    cols: 6
    help: [6×6 fuzzyMatrix]
      gr: [6×1 fuzzyMatrix]
     ind: [6×1 double]
   exist: 1
     low: [6×1 fuzzyMatrix]

>> S.x.exist
ans =
  logical
    1

>> S.x.gr
ans =
  6×1 fuzzyMatrix:
  double data:
    0.3000
    0.5000
    1.0000
    0.5000
    0.2500
    1.0000

>> S.x.low
ans =
  6×1 fuzzyMatrix:
  double data:
     0
     0
     0
```

 doi:10.20944/preprints202508.1072.v1


```
     0
     0
     0
```

Solving A X ≥ B using ⊙ composition and the same A and B

```
>> S.inequalities = 1;

>> S.solve_inverse();

>> S.x
ans =
  struct with fields:
    rows: 6
    cols: 6
    help: [4×6 fuzzyMatrix]
      gr: [6×1 fuzzyMatrix]
     ind: [6×1 double]
   exist: 1
ominated: [6 3]
elp_rows: 4
     low: [6×3 fuzzyMatrix]

>> S.x.exist
ans =
  logical
    1

>> S.x.gr
ans =
  6×1 fuzzyMatrix:
  double data:
    1
    1
    1
    1
    1
    1

> S.x.low
ans =
  6×3 fuzzyMatrix:
  double data:
         0         0         0
    0.5000    0.5000         0
    1.0000         0    1.0000
    0.5000         0         0
         0    0.2500    0.2500
         0         0         0
```

Solving A X = B using Łukasiewicz composition

```
>> A = fuzzyMatrix([0.8, 0.1, 0.7, 0.9;
                    0.9, 0.7, 0.2, 0.8;
                    0.2, 0.8, 0.9, 0.7;
                    0.3, 0.1, 0.0, 0.9]);
```

 doi:10.20944/preprints202508.1072.v1


```
>> B = fuzzyMatrix([0.5; 0.6; 0.7; 0.0]);

>> S = fuzzySystem('maxlukasiewicz', A, B, [], true)
S =
  fuzzySystem with properties:
    composition: 'maxlukasiewicz'
              a: [4×4 fuzzyMatrix]
              b: [4×1 fuzzyMatrix]
              x: [0×0 fuzzyMatrix]
           full: 1
   inequalities: 0

>> S.solve_inverse();

>> S.x
ans =
  struct with fields:
      rows: 4
      cols: 4
      help: [3×4 fuzzyMatrix]
        gr: [4×1 fuzzyMatrix]
       ind: [4×1 double]
     exist: 1
 dominated: 4
 help_rows: 3
       low: [4×3 fuzzyMatrix]

>> S.x.exist
ans =
  logical
    1

>> S.x.gr
ans =
  4×1 fuzzyMatrix:
  double data:
    0.7000
    0.9000
    0.8000
    0.1000

>> S.x.low
ans =
  4×3 fuzzyMatrix:
    double data:
      0.7000    0.7000         0
      0.9000         0    0.9000
      0         0.8000    0.8000
      0         0         0
```

 doi:10.20944/preprints202508.1072.v1

Appendix A.3. Examples for the `fuzzyMachine` module

Initialize a fuzzy finite machine using • composition [48]

```
>> m1=fuzzyMatrix([0 0.6 0.5; 0.6 0.1 0.5; 0.2 0.1 0.2]);
>> m2=fuzzyMatrix([0 0 0; 0 0 0; 0.2 0.1 0.1])
>> m3=fuzzyMatrix([0.4 0.2 0.1; 0.3 0.4 0.1; 0 0 0])
>> m4=fuzzyMatrix([0 0.3 0.2; 0.3 0.1 0.2; 0.1 0 0.1])

>> m = fuzzyMachine({m1,m2,m3,m4}, 'maxmin', 'none', 2, true)
m =
  fuzzyMachine with properties:
        initial_set: {
          [3×3 fuzzyMatrix]  [3×3 fuzzyMatrix]
          [3×3 fuzzyMatrix]  [3×3 fuzzyMatrix]
        }
        composition: 'maxmin'
               norm: 'max'
            conorm: 'min'
        postprocess: 'none'
        word_length: 2
               full: 1
    behavior_matrix: []
            letters: 3
```

Find full behavior for letters with length 2

```
>> m.word_length = 2; m.full = true; m.postprocess = 'none';

>> m.find_behavior

>> m.behavior_matrix
ans =
  3×21 fuzzyMatrix:
  double data:
  Columns 1 through 6
    1.0000    0.6000         0    0.4000    0.3000    0.6000
    1.0000    0.6000         0    0.4000    0.3000    0.6000
    1.0000    0.2000    0.2000         0    0.1000    0.2000

  Columns 7 through 12
    0.2000    0.4000    0.3000         0         0         0
    0.2000    0.4000    0.3000         0         0         0
    0.2000    0.2000    0.2000    0.2000    0.1000    0.2000

  Columns 13 through 18
         0    0.4000    0.1000    0.4000    0.3000    0.3000
         0    0.4000    0.1000    0.4000    0.3000    0.3000
    0.2000         0         0         0         0    0.1000

  Columns 19 through 21
    0.2000    0.3000    0.3000
    0.2000    0.3000    0.3000
    0.1000    0.1000    0.1000
```

 doi:10.20944/preprints202508.1072.v1


```
Find minimized behavior for letters with arbitrary length
>> m.word_length = -1; m.full = false; m.postprocess = 'minimize';

>> m.find_behavior

>> m.behavior_matrix
ans =
  2×4 fuzzyMatrix:
  double data:
    1.0000    0.6000         0    0.4000
    1.0000    0.6000         0    0.4000
    1.0000    0.2000    0.2000         0
```

```
Find reduced behavior for letters with arbitrary length
>> m.word_length = -1; m.full = false; m.postprocess = 'reduce';

>> m.find_behavior

>> m.behavior_matrix
ans =
  2×4 fuzzyMatrix:
  double data:
    1.0000    0.6000         0    0.4000
    1.0000    0.2000    0.2000         0
```

Appendix A.4. Examples for the `fuzzyOptimizationProblem` module

This section provides examples demonstrating the use of the `fuzzyOptimizationProblem` class to define, minimize and maximize fuzzy optimization problems.

```
Define a fuzzy optimization problem with ⊙ composition constraints [52]
>> A = fuzzyMatrix([0.00, 0.20, 0.05, 0.00, 0.40, 0.00;
                    0.10, 0.60, 0.30, 0.00, 0.20, 0.20;
                    0.80, 0.48, 0.24, 0.48, 0.00, 0.00;
                    0.30, 0.00, 0.00, 0.40, 0.80, 0.15;
                    0.00, 0.00, 0.12, 0.20, 0.48, 0.10;
                    0.50, 0.30, 0.00, 0.10, 0.60, 0.00]);

>> B = fuzzyMatrix([0.10; 0.30; 0.24; 0.20; 0.12; 0.15]);

>> S = fuzzySystem('maxprod', A, B, [], true);

>> O = fuzzyOptimizationProblem([5, -4, 8, 2, -3, 7], S)
O =
  fuzzyOptimizationProblem with properties:
            object: [5 -4 8 2 -3 7]
       constraints: [1×1 fuzzySystem]
    object_solution: []
       object_value: []
```

```
Minimization
>> O.minimize()
ans =
  fuzzyOptimizationProblem with properties:
```

 doi:10.20944/preprints202508.1072.v1


```
             object: [5 -4 8 2 -3 7]
        constraints: [1×1 fuzzySystem]
    object_solution: [6×1 double]
       object_value: -2.7500

>> O.object_solution
ans =
         0
    0.5000
         0
         0
    0.2500
         0

>> O.object_value
ans =
   -2.7500
```

Maximization

```
>> O.maximize()
ans =
  fuzzyOptimizationProblem with properties:
             object: [5 -4 8 2 -3 7]
        constraints: [1×1 fuzzySystem]
    object_solution: [6×1 double]
       object_value: 16.7500

>> O.object_solution
ans =
    0.3000
         0
    1.0000
    0.5000
    0.2500
    1.0000

>> O.object_value
ans =
   16.7500
```

Author Contributions: Conceptualization, K.P. and Z.Z.; methodology, K.P. and Z.Z.; software, Z.Z.; validation, K.P. and Z.Z.; formal analysis, K.P. and Z.Z.; investigation, K.P.; resources, K.P.; data curation, K.P. and Z.Z.; writing—original draft preparation, K.P. and Z.Z.; writing—review and editing, K.P. and Z.Z.; visualization, K.P. and Z.Z.; supervision, K.P. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding

Abbreviations

The following abbreviations are used in this manuscript:

 doi:10.20944/preprints202508.1072.v1

FRC	fuzzy relational calculus
FFM	finite fuzzy matrix
FRE	fuzzy relation equation
FLSEs	fuzzy linear system of equations
FLSIs	fuzzy linear system of inequalities
FFMach	Finite fuzzy machine

References

1. MacLane, S.; Birkhoff G. *Algebra*. Macmillan, New York, **1979**.
2. Sanchez E. *Equations de Relations Floves*. Thèse Biologie Humaine, **1972**, Marseile, France.
3. Sanchez E. Solution in Composite Fuzzy Relation Equations: Application to Medical Diagnosis in Brouwerian Logic. *Fuzzy Automata and Decision Processes* **1977**, Elsevier North-Holland, INC pp. 221-234,
4. Sanchez, E. Resolution of composite fuzzy relation equations. *Information and Control* **1976**, vol. 30, pp. 38–48.
5. Higashi M.; Klir G.J. Resolution of finite fuzzy relation equations. *FSS* **1984**, vol. 13 (1), pp. 65–82.
6. Czogała, E.; Drewniak, J.; Pedricz, W. Fuzzy relation equations on a finite set. *FSS* **1982**, vol. 7(1), pp. 89-101.
7. Miyakoshi M.; Shimbo M. Lower solutions of systems of fuzzy equations. *FSS* **1986**, vol. 19 pp. 37–46.
8. Pappis C.P.; Sugeno M. Fuzzy relational equations and the inverse problem. *FSS* **1985**, vol. 15, pp. 79–90.
9. Adomopoulos G.; Pappis, C. Some results on the resolution of fuzzy relation equations. *FSS* **1993**, vol.60 (1), pp. 83–88.
10. Pappis C.; Adamopoulos G. A software routine to solve the generalized inverse problem of fuzzy systems. *FSS* **1992**, vol. 47, pp. 319-322.
11. Pappis C.; Adamopoulos G. A computer algorithm for the solution of the inverse problem of fuzzy systems. *FSS* **1991**, vol. 39, pp. 279–290.
12. Peeva K. Fuzzy linear systems. *FSS* **1992**, vol. 49, pp.339–355.
13. Chen, L.; Wang, P. Fuzzy relational equations (I): The General and Specialized Solving Algorithms. *Soft Computing* **2002**, vol. 6, pp. 428–435.
14. Peeva K. Universal algorithm for solving fuzzy relational equations. *Italian Journal of Pure and Applied Mathematics* **2006**, vol. 19, pp. 9–20.
15. Peeva K.; Kyosev Y. *Fuzzy relational calculus – theory, applications and software (with CD-ROM)*, Advances in Fuzzy Systems – Applications and Theory, vol. 22, World Scientific Publishing Company, **2004**.
16. Bourke, M.M.; Fisher, D.G. Solution algorithms for fuzzy relational equations with max–product composition. *FSS* **1998**, vol. 94, pp. 61–69.
17. Di Nola A.; Pedrycz W.; Sessa S.; Sanchez E. *Fuzzy Relation Equations and Their Application to Knowledge Engineering*. Kluwer Academic Press, Dordrecht/Boston/London, **1989**.
18. Loetamonphong J.; Fang S.-C. An efficient solution procedure for fuzzy relational equations with max–product composition. *IEEE Transactions on Fuzzy Systems* **1999**, vol. 7 (4), pp.441–445.
19. Markovskii, A.V. On the relation between equations with max–product composition and the covering problem. *FSS*, **2005**, vol. 153 (2), pp. 261-273.
20. Bartl, E.; Belohlávek, R. Sup-t-norm and inf-residuum are a single type of relational equations. *Int. J. Gen. Syst.* **2011**, vol. 40(6) pp. 599-609.
21. Feng Sun. Conditions for the existence of the least solution and minimal solutions to fuzzy relation equations over complete Brouwerian lattices. *Information Sciences*, **2012**, vol. 205, pp. 86-92.
22. Nosková L.; Perfilieva I. System of fuzzy relation equations with sup-* composition in semi-linear spaces: minimal solutions. *2007 IEEE International Fuzzy Systems Conference*, London, UK, **2007**, pp. 1-6, doi: 10.1109/FUZZY.2007.4295592.
23. Perfilieva I.; Nosková L. System of fuzzy relation equations with inf-$\rightarrow$ composition: Complete set of solutions. *FSS* **2008**, vol 159 (17), pp. 2256-2271.
24. Bartl, E. Minimal solutions of generalized fuzzy relational equations: Probabilistic algorithm based on greedy approach. *FSS* **2015**, vol. 260, pp. 25-42.
25. Bartl, E.; Klir, G. J. Fuzzy relational equations in general framework. *Int. J. Gen. Syst.* **2014**, vol. 43(1), pp. 1-18.
26. Medina J.; Turunen E.; Bartl E.; Juan Carlos Díaz-Moreno. Minimal Solutions of Fuzzy Relation Equations with General Operators on the Unit Interval. *IPMU* **2014** (3), pp. 81-90.

 doi:10.20944/preprints202508.1072.v1

27. De Baets, B. Analytical solution methods for fuzzy relational equations. In: *Fundamentals of Fuzzy Sets, The Handbooks of Fuzzy Sets Series*, vol. 1, D. Dubois, H. Prade (Eds.), Kluwer Academic Publishers **2000**, pp. 291–340.
28. Li P.; Fang S.-C. A survey on fuzzy relational equations. Part I: Classification and solvability. *Fuzzy Optimization and Decision Making* **2009**, vol. 8, pp. 179–229.
29. Molai, A.A.; Khorram E. An algorithm for solving fuzzy relation equations with max-T composition operator. *Information Sciences* **2008**, vol. 178(5), pp.1293-1308.
30. Shieh B.-S. New resolution of finite fuzzy relation equations with max-min composition. *International Journal of Uncertainty Fuzziness Knowledge Based Systems* **2008**, vol 16 (1), pp. 19–33.
31. Shivanian E. An algorithm for finding solutions of fuzzy relation equations with max-Lukasiewicz composition, *Mathware & Soft Computing*, **2010** vol. 17, pp. 15-26.
32. Wu Y.-K.; Guu S.-M. An efficient procedure for solving a fuzzy relational equation with max-Archimedean t-norm composition. *IEEE Transactions on Fuzzy Systems* **2008**, vol. 16 (1), pp. 73–84.
33. Bartl, E.; Belohlávek, R. Hardness of Solving Relational Equations. *IEEE Trans. Fuzzy Syst.* **2015**, vol. 23(6), pp. 2435-2438.
34. http://www.mathworks.com/matlabcentral/fleexchange/6214-fuzzy-relational-calculus-toolbox-rel-1-01.
35. Bartl, E.; Trnecka, M. Covering of minimal solutions to fuzzy relational equations. *International Journal of General Systems* **2021**, vol. 50 (2), pp. 117–138.
36. Lin J.-L. On the relation between fuzzy max-Archimedean t-norm relational equations and the covering problem. *FSS* **2009**, vol. 160 (16), pp. 2328–2344.
37. Lin J.-L.; Wu Y.-K.; Guu S.-M. On fuzzy relational equations and the covering problem. *Information Sciences*, **2011**, Vol. 181, (14), pp 2951-2963.
38. http://www.mathworks.com/matlabcentral, fuzzy-calculus-core-fc2ore (2010).
39. Grätzer, G. *General Lattice Theory*, Akademie-Verlag, Berlin, **1978**.
40. Hàjek, P. *Metamathematics of Fuzzy Logic*, Kluer, Dordecht, **1998**.
41. Peeva, K. Resolution of Fuzzy Relational Equations – Method, Algorithm and Software with Applications. *Inf. Sci.* **2013**, vol 234, pp. 44 - 63.
42. K. Peeva, G. Zaharieva, Zl. Zahariev, *Resolution of max–t-norm fuzzy linear system of equations in BL-algebras*, AIP Conference Proceedings, Vol. 1789, 060005, 2016. https://doi.org/10.1063/1.4968497
43. Zl. Zahariev, G. Zaharieva, K. Peeva, *Fuzzy relational equations – Min-Goguen implication*, AIP Conference Proceedings, Vol. 2505, 120004, 2022. https://doi.org/10.1063/5.0103030
44. Zl. Zahariev, *Fuzzy Relational Equations And Inequalities – Min-Łukasiewicz Implication*, AIP Conference Proceedings, Vol. 2939, 030011, 2023. https://doi.org/10.1063/5.0179435
45. Santos, E. S. Maximin automata *Information and Control*, **1968**, vol. 13 pp. 363-377.
46. Santos,E. S. Maximin, minimax and composite sequential mashines. *J. Math. Anal. Appl.*, **1968**, vol. 24, pp. 246-259.
47. Santos E. S.; Wee, W. G. General formulation of sequential machines. *Information and Control*, **1968**, vol. 12 (1), pp. 5-10.
48. Peeva, K; Zahariev, Zl. Computing behavior of finite fuzzy machines – Algorithm and its application to reduction and minimization. *Information Sciences*, **2008**, vol. 178, pp. 4152-4165.
49. Fang S.-G; Li G. Solving Fuzzy Relation Equations with Linear Objective Function. *FSS*, **1999** , vol. 103, pp. 107–113.
50. Guu S. M; Wu Y-K. Minimizing a linear objective function with fuzzy relation equation constraints. *Fuzzy Optimization and Decision Making*, **2002**, vol. 4 (1), pp.347-360.
51. Loetamonphong J.; Fang S.-C. (2001) Optimization of fuzzy relation equations with max-product composition. *FSS*, **2001**, vol. 118 (3), pp. 509–517.
52. Peeva K.; Zahariev Zl.; Atanasov I. Optimization of linear objective function under max-product fuzzy relational constraints. In *Proceedings of the 9th WSEAS International Conference on FUZZY SYSTEMS (FS'08) – Advantest Topics on Fuzzy Systems*, **2008**, Book series: Artificial Intelligence Series – WSEAS Sofia, Bulgaria, May 2-4, 2008, ISBN: 978-960-6766-56-5.
53. Peeva, K.; Petrov, D. Optimization of Linear Objective Function under Fuzzy Equation Constraint in BL–Algebras – Theory, Algorithm and Software. In: *Intelligent Systems: From Theory to Practice. Studies in Computational Intelligence*, Sgurev, V., Hadjiski, M., Kacprzyk, J. (eds), **2010** vol 299. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-13428-9-20.

 doi:10.20944/preprints202508.1072.v1

54. Wang P.Z.; Zhang D.Z.; Sanchez E.; Lee E.S. Latticized linear programming and fuzzy relation inequalities. *Math. Anal. Appl.*, **1991**, vol. 159 (1), pp. 72-87.
55. Wu, Y-K; Guu S. M.; Liu Y.-C. An Accelerated Approach for Solving Fuzzy Relation Equations with a Linear Objective Function. |textitIEEE Transactions on Fuzzy Systems, **2002**, vol. 10(4), pp. 552–558.
56. Santos, E. S. On reduction of maxi-min machines. *J. Math. Anal. Appl.*, **1972** vol. 40 pp. 60-78.