RESEARCH ARTICLE | SEPTEMBER 06 2022

Fuzzy relational equations - Min-Goguen implication

Zlatko Zahariev; Galina Zaharieva; Ketty Peeva

AIP Conference Proceedings 2505, 120004 (2022)
https://doi.org/10.1063/5.0103030

Fuzzy Relational Equations - Min-Goguen Implication

Zlatko Zahariev,[a] Galina Zaharieva,[b] and Ketty Peeva[c]

Faculty of Applied Mathematics, Technical University of Sofia, Sofia 1000, Bulgaria

[a]*Corresponding author: zlatko@tu-sofia.bg*
[b]*Electronic mail: gpantaleeva@gmail.com*
[c]*Electronic mail: kgp@tu-sofia.bg*

Abstract. An algorithm for solving Min-Goguen fuzzy linear systems of equations is presented in this paper. Solving linear systems of equations is subject of great scientific interest. The authors have developed fast an efficient algorithms over several algebras. Here we present, relatively simple, fast and efficient algorithm to solve fuzzy linear systems of equations for Min-Goguen algebra, logically backed by previously developed by the authors algorithms in max-min and min-max algebras.

INTRODUCTION

Inverse problem resolution for fuzzy linear systems is subject of great scientific interest. The main results are published in [1], [2] and [3]. They demonstrate long and difficult period of investigations for discovering analytical methods and procedures to determine complete solution set, as well as to develop software for computing solutions, see [3] and [4]. The first and most essential are Sanchez results [5] for the greatest solution of fuzzy relational equations with max-min and min-max composition. Sanchez gives formulas that permit to determine the potential greatest solution in any of these cases, often used as solvability criteria. Universal algorithm and software for solving max-min and min-max fuzzy relational equations is proposed in [3], [4], and [6]. The relationship with the covering problem is subject of [7], where two methods for solving such fuzzy linear systems (algebraic and with table decomposition) are discussed and an algorithm is proposed, realizing table decomposition method.

This paper proposes methodology, unified method and algorithm for inverse problem resolution for Min-Goguen Implication fuzzy linear systems of equations. The algorithm is backed by a logic similar to the presented in [8] and [9], but adapted to a not investigated there operation.

It is focused on solving *fuzzy linear systems of equations*:

$$\left| \begin{array}{l} (a_{11} \rightarrow_{\odot} x_1) \wedge (a_{12} \rightarrow_{\odot} x_2) \wedge \ldots \wedge (a_{1n} \rightarrow_{\odot} x_n) = b_1 \\ (a_{21} \rightarrow_{\odot} x_1) \wedge (a_{22} \rightarrow_{\odot} x_2) \wedge \ldots \wedge (a_{2n} \rightarrow_{\odot} x_n) = b_2 \\ \ldots \\ (a_{m1} \rightarrow_{\odot} x_1) \wedge (a_{m2} \rightarrow_{\odot} x_2) \wedge \ldots \wedge (a_{mn} \rightarrow_{\odot} x_n) = b_m \end{array} \right. \tag{1}$$

where $a_{ij}, b_i \in [0,1]$, are given and $x_j \in [0,1]$ marks the unknowns in the system. In this paper for the indices we suppose $i = 1,...,m,\ j = 1,...,n$

The system (1) will be presented in matrix form:

$$A \rightarrow_{\odot} X = B \tag{2}$$

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 and $X = (x_j)_{1\times n}$ is the vector of unknowns.

The algorithm presented here partially follows the logical structure used in [10] but as the used algebra is different, a lot of significant adjustments are presented in order to implement that logic for the presented here use-case. The algorithm is somehow based on the algorithms presented in [10] but instead of using *generators* [11] or *coverings* [12] it uses *domination* and *list operations* – that improves the algebraic-logical approach from [10], and make it more efficient.

In general the algorithm keeps the idea for domination and the algebraic-logical approach from [10] but highly improves their realization. To achieve this, the algebraic-logical approach is substituted by a highly improved and simplified procedure presented further. New approach for finding the lowest solution of (1) is also presented further as well as other small improvements pointed among the lines.

The paper is divided in six sections. Next section gives some basic notions. Then, the theoretical background for solving the system (1) is given. Then the algorithms are presented. Examples are given after this. Analysis of the computational complexity and memory complexity of the algorithm as well as some conclusions can be found in the last two sections.

The terminology for fuzzy sets is according to [13], for fuzzy equations – as in [1], and [3], for algorithms, computational complexity and memory complexity – as in [14] and [15].

BASIC NOTIONS

Let $a, b \in [0,1]$.

Operation $\vee$ between a and b is defined as

$$a \vee b = max(a,b) \tag{3}$$

Operation $\wedge$ between a and b is defined as

$$a \wedge b = min(a,b) \tag{4}$$

Operation $\odot$ between a and b is defined as the conventional real numbers multiplication.

Operation $\rightarrow_{\odot}$ between a and b is defined as

$$a \rightarrow_{\odot} b = \begin{cases} 1, \text{ if } a \leq b \\ \frac{b}{a}, \text{ if } a > b \end{cases} \tag{5}$$

A matrix $A = (a_{ij})_{m\times n}$ with $a_{ij} \in [0,1]$ for each $i = 1,...,m$, $j = 1,...,n$ is called *membership matrix*. In what follows *'matrix'* is used instead of *'membership matrix'*.

Let the matrices $A = (a_{ij})_{m\times p}$ and $B = (b_{ij})_{p\times n}$ be given.

The matrix $C_{m\times n} = (c_{ij}) = A \rightarrow_{\odot} B$ is called *min*$-\rightarrow_{\odot}$ *product* of A and B if

$$c_{ij} = \wedge_{k=1}^{p}(a_{ik} \rightarrow_{\odot} b_{kj}) \tag{6}$$

for each $i = 1,...,m$, $j = 1,...,n$.

The matrix $C_{m\times n} = (c_{ij}) = A \odot B$ is called *max*$-\odot$ *product* of A and B if

$$c_{ij} = \vee_{k=1}^{p}(a_{ik} \odot b_{kj}) \tag{7}$$

for each $i = 1,...,m$, $j = 1,...,n$.

For $X = (x_j)_{1\times n}$ and $Y = (y_j)_{1\times n}$ the inequality $X \geq Y$ holds iff $x_j \geq y_j$ for each $j = 1,...,n$.

Next notions are according to [4]

A vector $X^0 = (x_j^0)_{1\times n}$ with $x_j^0 \in [0,1]$, $j = 1,...,n$, is called *solution* of the system (2) if $A \rightarrow_{\odot} X^0 = B$ holds. The set of all solutions of (2) is called *complete solution set* and it is denoted by $\mathbb{X}^0$. If $\mathbb{X}^0 \neq \emptyset$ then the system is called *solvable* (or *consistent*), otherwise it is called *unsolvable* (or *inconsistent*).

A solution $X_u^0 \in \mathbb{X}^0$ is called *upper solution* of $A \rightarrow_{\odot} X = B$ if for any $X^0 \in \mathbb{X}^0$ the inequality $X_u^0 \leq X^0$ implies $X^0 = X_u^0$. A solution $X_{low}^0 \in \mathbb{X}^0$ is called *lower solution* of $A \rightarrow_{\odot} X = B$ if for any $X^0 \in \mathbb{X}^0$ the inequality $X^0 \leq X_{low}^0$ implies $X^0 = X_{low}^0$. If the lower solution is unique, it is called *lowest* (or *minimum*) solution. The *n*-tuple $(X_1,...,X_n)$ with $X_j \subseteq [0,1]$ is called *interval solution* of the system $A \rightarrow_{\odot} X = B$ if any $X^0 = (x_j^0)_{n\times 1}$ with $x_j^0 \in X_j$ for each $j = 1,...,n$ implies $X^0 = (x_j^0)_{n\times 1} \in \mathbb{X}^0$. Any interval solution of $A \rightarrow_{\odot} X = B$ whose components (interval bounds) are determined by the lowest solution from the left and by an upper solution from the right, is called *minimal interval solution* of $A \rightarrow_{\odot} X = B$.

SIMPLIFICATIONS

Lowest solution

It is well known [1], that any solvable *min*$-\rightarrow_{\odot}$ fuzzy linear system of equations has unique lowest solution. In order to find all solutions of the solvable system, it is necessary to find both its lowest solution and all of its upper solutions. Finding the lowest solution is relatively simple task often used as a criteria for establishing solvability of the system [10]. Finding all upper solutions is reasonable only when the lowest solution exists.

Next we follow the terminology and we remind the results from [10].

Classical approach

The traditional approach to solve (1) is based on the following theorem:

Theorem 1. [10] Let A and B be given matrices and $X_{\rightarrow_\odot}$ be the set of all matrices such that $A \rightarrow_\odot X = B$ when $X \in X_{\rightarrow_\odot}$. Then

- $X_{\rightarrow_\odot} \neq \emptyset$ iff $A^t \odot B \in X_{\rightarrow_\odot}$.
- If $X_{\rightarrow_\odot} \neq \emptyset$ then $A^t \odot B$ is the lowest element in $X_{\rightarrow_\odot}$. □

If the system (1) is solvable, its lowest solution is given by $\check{X} = (\check{x}_j) = A^t \odot B$.

Using this fact, an appropriate algorithm for checking consistency of the system and for finding its lowest solution is obtained [10]. Its computational complexity is $O(m.n^2)$. Nevertheless that it is simple, it is too hard for such a task.

More efficient approach

Here we propose a simpler way to answer both questions, simultaneously computing the lowest solution and establishing consistency of (1).

The algorithm uses the fact that it is possible to find the value of the unknown $\check{x}_j$ if only the j^{th} column of the matrix A is considered. For every $i = 1,...,m$, all the a_{ij} coefficients provide a way to satisfy the corresponding i^{th} equation with $a_{ij} \rightarrow_\odot x_j = b_i$ when $x_j = a_{ij} \odot b_i$

For the purposes of the next theorem, $\check{b}_j$ is introduced as follows:

$$\check{b}_j = \max_{i=1}^{m} \{a_{ij} \odot b_i\} \tag{8}$$

In other words, for each $j = 1,...,n$, $\check{b}_j$ is equal to the highest value of $a_{ij} \odot b_i$, $i = 1,...,m$.

Theorem 2. [10] A system $A \rightarrow_\odot X = B$ is solvable iff $\check{X} = (\check{b}_j)$ is its solution. □

Corollary 1. In a solvable system (1), choosing $x_j < \check{b}_j$ for at least one $j = 1,...,n$ makes the system unsolvable.

Proof. Suppose $\check{b}_j = a_{kj} \odot b_k$. Let we choose $x_j < a_{kj} \odot b_k$. This will make at least one $(a_{kj} \rightarrow_\odot x_j)$ part of the k^{th} equation lower then b_k and because we take the minimum of all those $(a_{kj} \rightarrow_\odot x_j)$ parts, that means that the whole left-hand side of the k^{th} equation will become lower than b_k and this proves the corollary. □

Corollary 2. In a solvable system (1), for every $j = 1,...,n$, the lowest admissible value for x_j is $\check{b}_j$. □

Corollary 3. If the system (1) is solvable its lowest solution is $\check{X} = (\check{x}_j) = (\check{b}_j)$, $j = 1,...,n$. □

Corollary 4. $\check{X} = (\check{x}_j) = (\check{b}_j)$ and $\check{X} = A^t \odot B$ are equivalent. □

In general, Theorem 2 and its corollaries shows that instead of calculating $\check{X} = A^t \odot B$ we can use faster algorithm to obtain $\check{X} = (\check{x}_j) = (\check{b}_j)$ (presented further in the paper).

$\check{X}$ is only the eventual lowest solution of the system (1), because it can be obtained for any system (1), even if the system is unsolvable, so the eventual solution should be checked in order to confirm that it is solution of (1). Explicit checking for the eventual solution will increase the computations complexity of the algorithm. To avoid this in the next presented algorithm this is done during the generation of the coefficients of the potential lowest solution. For every obtained coefficient $(\check{x}_j) \in \check{X}$ there is check, which equations of (1) can be satisfied with it (hold in the boolean vector *IND*). If in the end of the algorithm all the equations of (1) are satisfied (i.e. all the coefficients in *IND* are set to $TRUE$) this means that the obtained solution is the lowest solution of (1), otherwise the system (1) is unsolvable.

Algorithm 1 Lowest solution of (1).

1. Initialize the vector $\check{X} = (\check{x}_j)$ with $\check{x}_j = 0$ for $j = 1,...,n$.
2. Initialize a boolean vector *IND* with $IND_i = FALSE$ for $i = 1,...,m$. This vector is used to store equations what are satisfied by the eventual lowest solution.
3. For every column $j = 1,...,n$ in A:
   (a) Find $\check{b}_j = \max_{i=1}^{m} \{a_{ij} \odot b_i\}$
   (b) Correct the value for $\check{x}_j$ to $\check{b}_j$.

(c) Find all i in the current j, such that $a_{ij} \odot b_i = \check{b}_j$ and correct the corresponding IND_i to $TRUE$

4. Check if all components of IND are set to $TRUE$.

   (a) If $IND_i = FALSE$ for some i the system $A \rightarrow_{\odot} X = B$ is unsolvable.

   (b) If $IND_i = TRUE$ for all $i = 1, ..., m$ the system $A \rightarrow_{\odot} X = B$ is solvable and its lowest solution is $\check{X}$.

5. Exit.

Theorem 2 and its corollaries prove that if the system is solvable, $\check{X}$ computed by this algorithm is its lowest solution. This is the first difference between published up to now results and the result presented in this paper. In existing up to now algorithms (see [10]) consistency of the system is established substituting $\check{X} = A^t \odot B$ for X in (2): when $A \rightarrow_{\odot} (A^t \odot B) = B$ holds, the system is solvable, otherwise it is unsolvable. With Algorithm 1 $\check{X}$ can be obtained in more efficient way thus improving the time complexity. In addition there is no need to substitute $\check{X}$ in order to establish consistency of the system.

IND vector proposed first in [16] here is used for a similar purpose. The algorithm uses this vector to check which equations are satisfied by the eventual $\check{X}$. At the end of the algorithm if all components in IND are $TRUE$ then $\check{X}$ is the lowest solution of the system, otherwise the system is unsolvable.

Analysis of the computational complexity for Algorithm 1 is given further. In general it is $O(m.n)$.

Upper solutions

It is important that every term (a_{ij}) in the system such that $a_{ij} < \check{b}_j$ cannot contribute to satisfy its according equation, so it should not be considered when extracting upper solutions. Also, the value for every component in the solution is either the value of the corresponding component in the vector $\check{b}_j$ or some bigger value. Along these lines, the hearth of the presented here algorithm is to find components a_{ij} in A and to give to X_{u_j} either the value of the corresponding $\check{b}_j$ or 1.

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

- Upper solution;
- Non-upper solution;
- Not solution at all.

The task of the algorithm is to extract all upper solutions and to skip the second and third types (i.e. not upper solutions). In order to extract all upper solutions a new method, based on the idea of the *dominance matrix* [3] in combination with a new technique is used to extract all lower solutions.

Domination

For the purposes of presented here algorithm, a modified version of the definition for domination is given. Original definition can be found in [10].

Definition 1. Let a_l and a_k be the l^{th} and the k^{th} equations, respectively, in (1) and $b_l \geq b_k$. a_l is called dominant to a_k and a_k is called dominated by a_l, if for each $j = 1, ..., n$ such that $a_{ij} < \check{b}_j$ it holds: $a_{lj} \odot b_l \leq a_{kj} \odot b_k$.

Extracting upper solutions

Algebraic-logical approach is used up to now for finding all upper solutions [10], algorithms are with exponential memory and time complexity [17]. The new algorithm, proposed here, also has exponential complexity but with a lower degree. Thereby it needs less number of steps from the other algorithms to obtain all lower solutions, for a problems with size $< \infty$. Also, it reduces part of the operations needed on every step (for instance *absorption*) and realizes faster approach on the other operations and thereby it is more efficient among now available algorithms.

Upper solutions are extracted by removing from A the dominated rows. A new matrix is produced and marked with $\widetilde{A} = (a_{\widetilde{i}j})$ 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 $\widetilde{a_{ij}} \odot b_{\widetilde{i}^*} = \check{b}_j$, x_j should be taken equal to the biggest $\widetilde{a_{ij}} \odot b_{\widetilde{i}^*}$ and all rows such that $\widetilde{a_{ij}} \odot b_{\widetilde{i}^*} < x_j$ 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 that principle is presented next:

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

1. Initialize solution vector $X_{u_0}(j) = 1,\ j = 1,...,n$.
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.
3. Initialize *sols* to be the empty set of vectors, which is supposed to be the set of all maximal solutions for current problem.
4. Check if $rows = \emptyset$. If so, check if in *sols* there is another vector $> X_{u_0}$, if no such a vector - add X_{u_0} to *sols*. Remove from *sols* all the vectors $< X_{u_0}$ and go to step 6.
5. Fix $\widetilde{i}$ equal to the first element in *rows*, then for every $j = 1,...,n$
   (a) Create a copy of X_{u_0} and update its j^{th} coefficient to be equal to $\widetilde{a_{ij}} \odot b_{\widetilde{i}^*}$ (if $\widetilde{a_{ij}} > 0$). Create a copy of *rows*.
   (b) For all $\widetilde{k}$ in *rows* if j^{th} coefficient of $X_{u_0} < \widetilde{a_{ij}} \odot b_{\widetilde{i}^*}$, remove $\widetilde{k}$ from the copy of *rows*.
   (c) Go to step 5 with copied in this step *rows* and X_{u_0}, i.e. start new recursive branch with reduced *rows* and changed X_{u_0}.
6. Exit.

As this is a recursive algorithm the best explanations for it can be done with an example. A suitable example is given further in Example 1.

Algorithms overview

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

Algorithm 3 Solving $A \rightarrow_{\odot} X = B$.

1. Obtain input data for the matrices A and B.
2. Obtain lowest solution for the system and check it for consistency (Algorithm 1).
3. If the system is unsolvable go to step 6.
4. Obtain the matrix $\widetilde{A}$.
5. Obtain all maximal solutions from $\widetilde{A}$ and B (Algorithm 2).
6. Exit.

Step 5 is the slowest part of the Algorithm 3. Detailed complexity analysis can be found further. In general, this algorithm has its best and worst cases and this is the most important improvement according to algebraic-logical approach from [10] (from the time complexity point of view). Algorithm 2 is going to have the same time complexity as the algorithm presented in [10] only in its worst case.

EXAMPLES

In the first example we illustrate presented algorithms. The second example shows solving a problem with a software based on Algorithm 3 and developed by Zl. Zahariev. Execution time is also given.

Example 1

Solve

$$\begin{pmatrix} 0 & 0.1 & 0.8 & 0.3 & 0 & 0.5 \\ 0.2 & 0.6 & 0.48 & 0 & 0 & 0.3 \\ 0.05 & 0.3 & 0.24 & 0 & 0.12 & 0 \\ 0 & 0 & 0.48 & 0.4 & 0.2 & 0.1 \\ 0.4 & 0.2 & 0 & 0.8 & 0.48 & 0.6 \\ 0.1 & 0.3 & 0.24 & 0.2 & 0.12 & 0.15 \end{pmatrix} \to_{\odot} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{pmatrix} = \begin{pmatrix} 0.3 \\ 0.5 \\ 1 \\ 0.5 \\ 0.25 \\ 1 \end{pmatrix}$$

Finding the lowest solution (Algorithm 1)

1. Initialize $\check{X} = (1.0\ \ 1.0\ \ 1.0\ \ 1.0\ \ 1.0)'$.
2. Initialize $IND = (ind_i) = FALSE$ for each $i = 1,...,m$.
3. Calculate $\check{x}_1 = \check{b}_1 = \max_{i=1}^{m}\{a_{i1} \odot b_i\}$:

$$\check{x}_1 = \check{b}_1 = \max_{i=1}^{m} \begin{pmatrix} a_{11} \odot b_1 \\ a_{21} \odot b_2 \\ a_{31} \odot b_3 \\ a_{41} \odot b_4 \\ a_{51} \odot b_5 \\ a_{61} \odot b_6 \end{pmatrix} = \max_{i=1}^{m} \begin{pmatrix} 0 \odot 0.3 \\ 0.2 \odot 0.5 \\ 0.05 \odot 1 \\ 0 \odot 0.5 \\ 0.4 \odot 0.25 \\ 0.1 \odot 1 \end{pmatrix} = \max_{i=1}^{m} \begin{pmatrix} 0 \\ 0.1 \\ 0.05 \\ 0 \\ 0.1 \\ 0.1 \end{pmatrix} = 0.1$$

   $a_{ij} \odot b_i = 0.1$ for $i = (2,5,6)$, so $IND_2 = TRUE$, $IND_5 = TRUE$, $IND_6 = TRUE$

4. Calculate $\check{x}_2 = \check{b}_2 = \max_{i=1}^{m}\{a_{i2} \odot b_i\}$:

$$\check{x}_2 = \check{b}_2 = \max_{i=1}^{m} \begin{pmatrix} a_{12} \odot b_1 \\ a_{22} \odot b_2 \\ a_{32} \odot b_3 \\ a_{42} \odot b_4 \\ a_{52} \odot b_5 \\ a_{62} \odot b_6 \end{pmatrix} = \max_{i=1}^{m} \begin{pmatrix} 0.1 \odot 0.3 \\ 0.6 \odot 0.5 \\ 0.3 \odot 1 \\ 0 \odot 0.5 \\ 0.2 \odot 0.25 \\ 0.3 \odot 1 \end{pmatrix} = \max_{i=1}^{m} \begin{pmatrix} 0.03 \\ 0.3 \\ 0.3 \\ 0 \\ 0.05 \\ 0.3 \end{pmatrix} = 0.3$$

   $a_{ij} \odot b_i = 0.3$ for $i = (2,3,6)$, IND_2 and IND_6 are already $TRUE$, so $IND_3 = TRUE$

5. Calculate $\check{x}_3 = \check{b}_3 = \max_{i=1}^{m}\{a_{i3} \odot b_i\}$:

$$\check{x}_3 = \check{b}_3 = \max_{i=1}^{m} \begin{pmatrix} a_{13} \odot b_1 \\ a_{23} \odot b_2 \\ a_{33} \odot b_3 \\ a_{43} \odot b_4 \\ a_{53} \odot b_5 \\ a_{63} \odot b_6 \end{pmatrix} = \max_{i=1}^{m} \begin{pmatrix} 0.8 \odot 0.3 \\ 0.48 \odot 0.5 \\ 0.24 \odot 1 \\ 0.48 \odot 0.5 \\ 0 \odot 0.25 \\ 0.24 \odot 1 \end{pmatrix} = \max_{i=1}^{m} \begin{pmatrix} 0.24 \\ 0.24 \\ 0.24 \\ 0.24 \\ 0 \\ 0.24 \end{pmatrix} = 0.24$$

   $a_{ij} \odot b_i = 0.24$ for $i = (1,2,3,4,6)$, IND_2, IND_3 and IND_6 are already $TRUE$, so $IND_1 = TRUE$, and $IND_4 = TRUE$. With this all the components of IND are now $TRUE$, so we can stop checking it.

6. Calculate $\check{x}_4 = \check{b_4} = \max_{i=1}^{m}\{a_{i4} \odot b_i\}$:

$$\check{x}_4 = \check{b_4} = \max_{i=1}^{m} \begin{pmatrix} a_{14} \odot b_1 \\ a_{24} \odot b_2 \\ a_{34} \odot b_3 \\ a_{44} \odot b_4 \\ a_{54} \odot b_5 \\ a_{64} \odot b_6 \end{pmatrix} = \max_{i=1}^{m} \begin{pmatrix} 0.3 \odot 0.3 \\ 0 \odot 0.5 \\ 0 \odot 1 \\ 0.4 \odot 0.5 \\ 0.8 \odot 0.25 \\ 0.2 \odot 1 \end{pmatrix} = \max_{i=1}^{m} \begin{pmatrix} 0.09 \\ 0 \\ 0 \\ 0.2 \\ 0.2 \\ 0.2 \end{pmatrix} = 0.2$$

7. Calculate $\check{x}_5 = \check{b_5} = \max_{i=1}^{m}\{a_{i5} \odot b_i\}$:

$$\check{x}_5 = \check{b_5} = \max_{i=1}^{m} \begin{pmatrix} a_{15} \odot b_1 \\ a_{25} \odot b_2 \\ a_{35} \odot b_3 \\ a_{45} \odot b_4 \\ a_{55} \odot b_5 \\ a_{65} \odot b_6 \end{pmatrix} = \max_{i=1}^{m} \begin{pmatrix} 0 \odot 0.3 \\ 0 \odot 0.5 \\ 0.12 \odot 1 \\ 0.2 \odot 0.5 \\ 0.48 \odot 0.25 \\ 0.12 \odot 1 \end{pmatrix} = \max_{i=1}^{m} \begin{pmatrix} 0 \\ 0 \\ 0.12 \\ 0.1 \\ 0.12 \\ 0.12 \end{pmatrix} = 0.12$$

8. Calculate $\check{x}_6 = \check{b_6} = \max_{i=1}^{m}\{a_{i6} \odot b_i\}$:

$$\check{x}_6 = \check{b_6} = \max_{i=1}^{m} \begin{pmatrix} a_{16} \odot b_1 \\ a_{26} \odot b_2 \\ a_{36} \odot b_3 \\ a_{46} \odot b_4 \\ a_{56} \odot b_5 \\ a_{66} \odot b_6 \end{pmatrix} = \max_{i=1}^{m} \begin{pmatrix} 0.5 \odot 0.3 \\ 0.3 \odot 0.5 \\ 0 \odot 1 \\ 0.1 \odot 0.5 \\ 0.6 \odot 0.25 \\ 0.15 \odot 1 \end{pmatrix} = \max_{i=1}^{m} \begin{pmatrix} 0.15 \\ 0.15 \\ 0 \\ 0.05 \\ 0.15 \\ 0.15 \end{pmatrix} = 0.15$$

With this the lowest solution of the system is obtained: $\check{X} = (0.1\ \ 0.3\ \ 0.24\ \ 0.2\ \ 0.12\ \ 0.15)'$. Also, Theorem 2 and its corollaries prove that this is the lowest solution and no more verification is needed.

Finding all upper solutions

1. For the matrix A

   From Definition 1, the second vector is dominated by the first, the third vector is dominated by the sixth and the sixth vector is dominated by the first. As all dominated vectors are redundant they should be removed from the set.

$$\tilde{A} = \begin{pmatrix} 0 & 0.1 & 0.8 & 0.3 & 0 & 0.5 \\ 0 & 0 & 0.48 & 0.4 & 0.2 & 0.1 \\ 0.4 & 0.2 & 0 & 0.8 & 0.48 & 0.6 \end{pmatrix}$$

2. Start with initial vector $X_{u_1} = (1\ \ 1\ \ 1\ \ 1\ \ 1)'$.

   Iterate through the first row of $\tilde{A}$. The first coefficient i from this row where $\tilde{a_{1j}} \odot b_j = \check{b}_j$ is $i = 3$. We update $X_{u_1}(3) = \tilde{a_{13}} \odot b_1 = 0.24$. As $\tilde{a_{23}} \odot b_2 = \tilde{a_{13}} \odot b_1$ we skip the second row from $\tilde{A}$.

   On the third row the first coefficient where $\tilde{a_{3j}} \odot b_j = \check{b}_j$ is 1, so we update $X_{u_1}(1) = \tilde{a_{31}} \odot b_3 = 0.1$. With this we acquire the first upper solution:

   $X_{u_1} = (0.1\ \ 1\ \ 0.24\ \ 1\ \ 1\ \ 1)'$

3. Step back in the "backtracking", we continue iterating through the third row. There are three more qualified coefficients - $\tilde{a_{34}}$, $\tilde{a_{35}}$, and $\tilde{a_{36}}$. From them we extract the next three upper solutions:

   $X_{u_2} = (1\ \ 1\ \ 0.24\ \ 0.2\ \ 1\ \ 1)'$

   $X_{u_3} = (1\ \ 1\ \ 0.24\ \ 1\ \ 0.12\ \ 1)'$

   $X_{u_4} = (1\ \ 1\ \ 0.24\ \ 1\ \ 1\ \ 0.15)'$

4. Step back in the "backtracking", we are back to continue iterating through the first row. There is one more qualified coefficient - $\tilde{a_{16}}$. From if we update $X_{u_5}(6) = \tilde{a_{16}} \odot b_1 = 0.15$. Next we go through the second row of $\tilde{A}$, and we will entirely skip the third row.

   From the second row we have two qualified coefficients - $\tilde{a_{23}}$ and $\tilde{a_{24}}$. From the first one we receive (1 1 0.24 1 1 0.15)', which is already extracted as a solution, so we skip it and from the second coefficient we extract the last upper solution for this example:

   $X_{u_5} = (1\ \ 1\ \ 1\ \ 0.2\ \ 1\ \ 0.15)'$

Example 2

Solve the same system:

$$\begin{pmatrix} 0 & 0.1 & 0.8 & 0.3 & 0 & 0.5 \\ 0.2 & 0.6 & 0.48 & 0 & 0 & 0.3 \\ 0.05 & 0.3 & 0.24 & 0 & 0.12 & 0 \\ 0 & 0 & 0.48 & 0.4 & 0.2 & 0.1 \\ 0.4 & 0.2 & 0 & 0.8 & 0.48 & 0.6 \\ 0.1 & 0.3 & 0.24 & 0.2 & 0.12 & 0.15 \end{pmatrix} \rightarrow_{\odot} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{pmatrix} = \begin{pmatrix} 0.3 \\ 0.5 \\ 1 \\ 0.5 \\ 0.25 \\ 1 \end{pmatrix}$$

using developed by Zl. Zahariev software. This also will demonstrate the efficiency of the presented here algorithm. The software used in this example as well as instructions can be found in [18].

Solving the problem

1. Input matrix A and B

2. Declare A and B as fuzzy matrices. *FuzzyMatrix* is a MATLAB class holding some crucial operations for a fuzzy matrices ([22])

```
>> A=fuzzyMatrix(A); B=fuzzyMatrix(B);
```

3. Create the system object with 'goguen' composition, empty matrix X and option for finding all upper solutions set as 'true'

```
>> S = fuzzySystem('goguen',A,B,
                   fuzzyMatrix(), true);
```

4. Solve the system

```
>> S.solve_inverse()
ans =
  fuzzySystem handle
  Properties:
     composition: 'goguen'
               a: [6x6 fuzzyMatrix]
               b: [6x1 fuzzyMatrix]
               x: [1x1 struct]
            full: 1
    inequalities: 0
  Methods, Events, Superclasses
```

5. The member variable x is a stricture which holds all the solutions of the system. We can inspect it and see that among the other information it holds one lower solution ($x.low$) and 5 upper solution ($x.gr$)

```
>> S.x
ans =
  struct with fields:
         rows: 6
         cols: 6
         help: [3?6 fuzzyMatrix]
          low: [6?1 fuzzyMatrix]
          ind: [6?1 double]
        exist: 1
    dominated: [2 3 6]
    help_rows: 3
           gr: [6?5 fuzzyMatrix]

>> S.x.low
ans =
  6?1 fuzzyMatrix:
  double data:
    0.1000
    0.3000
    0.2400
    0.2000
    0.1200
    0.1500

>> S.x.gr
ans =
  6?5 fuzzyMatrix:
  double data:
    0.1000    1.0000    1.0000    1.0000    1.0000
    1.0000    1.0000    1.0000    1.0000    1.0000
    0.2400    0.2400    0.2400    0.2400    1.0000
    1.0000    0.2000    1.0000    1.0000    0.2000
    1.0000    1.0000    0.1200    1.0000    1.0000
    1.0000    1.0000    1.0000    0.1500    0.1500
```

Execution time

On a test computer this example was solved with the presented above software in about 0.001 seconds.

COMPUTATIONAL AND MEMORY COMPLEXITY

In this section the terminology for computational complexity and for memory complexity is according to [14].

The problem has exponential computational complexity [17]. It actually depends on the the number of the lower solutions and if the system has a solution at all.

For obtaining $\check{X}$ the algorithm needs to iterate through all the elements in the matrix A, so the time complexity is $O(m.n)$. Since there is need to keep information about the greatest solution, as well as IND vector, memory complexity for this part of the problem is $O(m+n)$.

The most complicated part of the problem is to obtain the upper solutions. Its computational complexity hardly depends on the number of non-dominated vectors. From Definition 1 it is easy to see that for a system in n unknowns and m equations, the maximal number of non-dominated vectors is less than or equal to $2^n - 1$. The components of these vectors should be iterated with *backtracking* based Algorithm 4 with exponential time complexity.

Checking for domination has complexity from a lower class and for this reason it is not discussed. But an extracted solution can be not upper or can be duplicated, and therefore it should be checked against all other extracted solution. This operation has the same computational complexity as the extraction of all upper solutions.

CONCLUSION

Presented algorithm is a simple and straightforward way to solve fuzzy $\rightarrow_{\odot}$ systems of linear equations. It gives fast and reasonable approach. The software, created by Zl. Zahariev, implements Algorithm 4. This software is about only 120 lines of MATLAB code, which just demonstrates algorithm simplicity. It is free distributed under BSD license agreements. Comparison with other software packages can be found in [19].

Based on the presented here algorithm and software, a new software package is created by Zl. Zahariev. The package is called Fuzzy Calculus Core (FC^{2}ore) [18], [20]. It supports operations with fuzzy matrices and solves fuzzy linear systems of equations and inequalities in various algebras. Up to now six types of systems can be solved with FC^{2}ore according to the used algebra: $max - min$, $min - max$, $max - product$, Gödel, Goguen and Łukasiewicz.

It is important that all six types of systems are solved with algorithms very close to the algorithm presented in this paper.

This software also proposes solving fuzzy optimization problems as well as obtaining, reducing and minimizing complete behavior matrix for fuzzy machines in all mentioned above algebras. It is also supposed that this software will be useful for fuzzy reasoning, pattern recognition and in graph theory for finding irredundant coverings.

FC^{2}ore is free distributed under BSD license agreements [18].

ACKNOWLEDGMENTS

The authors would like to thank the Research and Development Sector at the Technical University – Sofia for the financial support.

REFERENCES

1. B. De Baets, Analytical solution methods for fuzzy relational equations, in the series: Fundamentals of Fuzzy Sets, The Handbooks of Fuzzy Sets Series, D. Dubois and H. Prade (eds.), Vol. 1, Kluwer Academic Publishers, 2000, pp. 291-340.
2. A. Di Nola, W. Pedrycz, S. Sessa and E. Sanchez, Fuzzy Relation Equations and Their Application to Knowledge Engineering, Kluwer Academic Press, Dordrecht, 1989.
3. K. Peeva and Y. Kyosev, Fuzzy Relational Calculus-Theory, Applications and Software (with CD-ROM), In the series Advances in Fuzzy Systems - Applications and Theory, Vol. 22, World Scientific Publishing Company, 2004. Software downloadable from http://www.mathworks.com/matlabcentral/fileexchange/loadFile.do?objectId=6214
4. K. Peeva, Universal algorithm for solving fuzzy relational equations, Italian Journal of Pure and Applied Mathematics 19, 2006, pp. 9-20.
5. E. Sanchez, Resolution of composite fuzzy relation equations, Information and Control, 30, 1976, pp. 38-48.
6. K. Peeva, Fuzzy linear systems – Theory and applications in Artificial Intelligence areas. DSc Thesis, Sofia, 2002, pp 239 (in Bulgarian)
7. A. Markovskii, On the relation between equations with max-product composition and the covering problem, Fuzzy Sets Systems, 153, 2005, pp. 261–273
8. K. Peeva, M. Durcheva, Constructing solution set of fuzzy linear systems of equations in product algebra. Comptes Rendus de L'Academie Bulgare des Sciences, 68, 2015, 165-174.
9. K. Peeva, G. Zaharieva, and Z. Zahariev, Resolution of max-t-norm fuzzy linear system of equations in BL-algebras, AIP Conference Proceedings vol. 1789, 060005, 2016, doi: 10.1063/1.4968497
10. K. Peeva, Resolution of Fuzzy Relational Equations – Method, Algorithm and Software with Applications, Information Sciences, 234 (Special Issue), 2013, 44-63, doi 10.1016/j.ins.2011.04.011, ISSN 0020-0255.
11. B. Shieh, New resolution of finite fuzzy relation equations with max-min composition, International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems, 16(1), (2008), pp. 19-33.
12. C. Yeh, On the minimal solutions of max–min fuzzy relational equations, Fuzzy Sets and Systems, 159, 2008, pp. 23-39.
13. G. Klir, U. Clair, B. Yuan, Fuzzy set theory foundations and applications. Prentice Hall, Englewood Cliffs, 1997.
14. A. Aho, J. Hopcroft, J, Ullman, The design and analysis of computer algorithms. Addison-Wesley, London, 1976.
15. M. Garey, D. Johnson, Computers and intractability. Aguideto the theory of NP-completeness. Freeman San Francisco, 1979.
16. K. Peeva, Fuzzy linear systems, Fuzzy Sets and Systems, 49, 1992, pp. 339-355.
17. L. Chen, P. Wang, Fuzzy relational equations (I): the general and specialized solving algorithm, Soft Computing 6, 2002, pp. 428-435.

18. http://www.mathworks.com/matlabcentral/fileexchange/27046-fuzzy-calculus-core-fc2ore
19. Z. Zahariev, Solving Max-min Relational Equations. Software and Applications, in International Conference "Applications of Mathematics in Engineering and Economics (AMEE'08)", AIP Conference Proceedings, vol. 1067, G. Venkov, R. Kovatcheva, V. Pasheva (eds.), American Institute of Physics, 2008, 516-523.
20. Z. Zahariev, Software package and API in MATLAB for working with fuzzy algebras, in International Conference "Applications of Mathematics in Engineering and Economics (AMEE'09)", AIP Conference Proceedings, vol. 1184, G. Venkov, R. Kovatcheva, V. Pasheva (eds.), American Institute of Physics, ISBN 978-0-7354-0750-9, 2009, 434-350.