Годишник на секция "Информатика"
Съюз на учените в България
Том 6, 2013, 1–16

Annual of "Informatics" Section
Union of Scientists in Bulgaria
Volume 6, 2013, 1–16

SOLVING MAX-MIN FUZZY LINEAR SYSTEMS OF EQUATIONS. ALGORITHM AND SOFTWARE

Zlatko Zahariev

Faculty of Applied Mathematics and Informatics
Technical University of Sofia
Sofia 1000, P. O. Box 384
e-mail: zlatko@tu-sofia.bg

Abstract: An algorithm for solving max-min fuzzy linear systems of equations is presented in this paper. The algorithm is based on the theory as presented in [10], but introduces improvements. Also a new data type is proposed to be used in software implementations. Analysis of the computational and memory complexity is given.
Keywords: Fuzzy linear systems of equations, Max-min composition, Algorithm, Complexity.

1. Introduction

This paper is focused on solving *fuzzy linear systems of equations*:

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

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 indixes we suppose $i = 1, ..., m, j = 1, ..., n$

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

$$A \bullet X = B \tag{2}$$

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.

These systems are investigated in [1], [3]- [11] and [14] from various points of view.

Despite that nowadays the problem appears as almost classical, there are still only few relevant algorithms to solve it [1], [4], [6]- [11], [16] and [19] and even less software implementations [11], [20], [22].

The algorithm presented here is based on the results given in [10]. Still a lot of principal improvements are made here. In order to help the reader, all improvements are presented below as well as all the parts from the original algorithm, which are relevant to this article.

At first glance the presented algorithm looks similar to the algorithms presented in [16] and [19] but instead of using *generators* [16] or *coverings* [19] it uses *domination* and *list operations* – that improves the algebraic-logical approach from [10].

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 domination matrix form [10] is substituted by a system of H-positioning vectors (Section 4) and the algebraic-logical approach is substituted by a list operations (Section 3.3) which highly improves the algorithm robustness. New approach of finding the greatest solution of (1) is also presented in Section 3.2.2 as well as other small improvements pointed among the lines.

The algorithms proposed in [10], [16] and [19] are heavy and obscure while the algorithm presented here is robust and lucid.

The paper is divided in seven sections. Next section gives some basic notions. In Section 3 the theoretical background for solving the system (1) is given. In Section 4 a specific data type is presented. The purpose of this data type is to make the algorithm more relevant for software implementations. Examples are given in Section 5. Analysis of the computational complexity and memory complexity of the algorithm is presented in Section 6. Some conclusions can be found in Section 7.

The terminology for fuzzy sets is according to [5], for fuzzy equations – as in [2], [3] and [10], for algorithms, computational complexity and memory complexity – as in [15] and [17].

2. 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 $\to_G$ between a and b is defined as

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

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

$$c_{ij} = \vee_{k=1}^{p}(\wedge(a_{ik}, b_{kj})) \tag{6}$$

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

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

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

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

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

Next notions are according to [10]:

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 $A \bullet X^0 = B$ 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 $X_{low}^0 \in \mathbb{X}^0$ is called *lower solution* of $A \bullet X = B$ if for any $X^0 \in \mathbb{X}^0$ the inequality $X^0 \leq X_{low}^0$ implies $X^0 = X_{low}^0$. A solution $X_u^0 \in \mathbb{X}^0$ is called *upper solution* of $A \bullet X = B$ if for any $X^0 \in \mathbb{X}^0$ the inequality $X_u^0 \leq X^0$ implies $X^0 = X_u^0$. If the upper solution is unique, it is called *greatest* (or *maximum*) solution. The n-tuple $(X_1, ..., X_n)$ with $X_j \subseteq [0,1]$ is called *interval solution* of the system $A \bullet X = B$ 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 $A \bullet X = B$ whose components (interval bounds) are determined by a lower solution from the left and by the greatest solution from the right, is called *maximal interval solution* of $A \bullet X = B$.

3. Simplifications

3.1. Normalization

The system (1), is called *normalized* [10] or *correctly ordered* with respect to the right hand side if $b_1 \geq b_2 \geq ... \geq b_m$. The algorithm in [10] as well as the algorithm here, presumes that the system is normalized. If not, it needs to be reordered. To decrease the computational complexity of the algorithm instead of reordering (1), a mapping vector $map(b)$ for B is used for indexing B

and rows in A. $map(b)$ marks the permutation of equations in (1). Using $map(b)$ is equivalent to reorder equations in (1) and thus without loss of generality, in what follows the system is supposed to be in normalized form.

3.2. Greatest solution

It is well known [14], that any solvable max-min fuzzy linear system of equations has unique greatest solution. In order to find all solutions of the solvable system, it is necessary to find both its greatest solution and all of its minimal solutions. Finding the greatest solution is relatively simple task often used as a criteria for establishing solvability of the system [10], [11], [16]. Finding all minimal solutions is reasonable only when the greatest solution exists.

In 3.2.1 and the beginning of 3.2.2 we follow the terminology and we remind the results from [10] and [11].

3.2.1. Classical approach

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

Theorem 1. [14] Let A and B be given matrices and $X_\bullet$ be the set of all matrices such that $A \bullet X = B$ when $X \in X_\bullet$. Then

- $X_\bullet \neq \emptyset$ iff $A^t \rightarrow_G B \in X_\bullet$.
- If $X_\bullet \neq \emptyset$ then $A^t \rightarrow_G B$ is the greatest element in $X_\bullet$.

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

Using this fact, an appropriate algorithm for checking consistency of the system and for finding its greatest solution is obtained [11], [16]. Its computational complexity is $O(m.n^2)$. Nevertheless that it is simple, it is too hard for such a task.

3.2.2. More efficient approach

Here is proposed a simpler way to answer both questions, simultaneously computing the greatest solution and establishing consistency of (1). Instead of using Theorem 1, we work with four types of coefficients (S, E, G and H) and a boolean vector (*IND*).

In the system (1):

- 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 $a_{ij} \geq b_i$.

The algorithm uses the fact that it is possible to find the value of the unknown $\widehat{x}_j$ if only the j^{th} column of the matrix A is considered. For every $i = 1, ..., m$, there are three cases:

- If a_{ij} is E-type coefficient then the i-th equation can be satisfied by $a_{ij} \wedge x_j < b_i$ 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 $a_{ij} \wedge x_j < b_i$ 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 $a_{ij} \wedge x_j < b_i$ 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:

$$\widehat{b}_j = \begin{cases} \min_{i=1}^{m} \{b_i\}, \text{ for all } i \text{ such that } a_{ij} > b_i \\ 1 \text{ otherwise} \end{cases} \tag{8}$$

In other words, for each $j = 1, ..., n$, $\widehat{b}_j$ is equal to the lowest coefficient in $B = (b_i)$, $i = 1, ..., m$, such that $a_{ij} > b_i$. If there is no such coefficient then $\widehat{b}_j = 1$.

Theorem 2. [4] A system $A \bullet X = B$ is solvable iff $\widehat{X} = (\widehat{b}_j)$ is its solution.

Corollary 1. In a solvable system (1), choosing $x_j > \widehat{b}_j$ for at least one $j = 1, ..., n$ makes the system unsolvable.
Proof. Suppose $\widehat{b}_j = b_k \neq 1$. Let we choose $x_j > b_k$. This means that the left-hand side of the k^{th} equation is greater than b_k and this proves the theorem. $\square$

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

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

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

In general, Theorem 3 and its corollaries shows that instead of calculating $\widehat{X} = A^t \rightarrow_G B$ we can use faster algorithm to obtain $\widehat{X} = (\widehat{x}_j) = (\widehat{b}_j)$ (presented further in the paper).

$\widehat{X}$ is only the eventual greatest 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 extraction of the coefficients of the potential greatest solution. For every obtained coefficient $(\widehat{x}_j) \in \widehat{X}$ 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 greatest solution of (1), otherwise the system (1) is unsolvable.

Algorithm 1 Greatest solution of (1).

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 store equations what are satisfied by the eventual greatest solution.

Step 3. For every column $j = 1, ..., n$ in A, walk successively through all coefficients $a_{ij}, i = 1, ..., m$ in the j^{th} column of A, until the first G-type coefficient is found. Here it is important that the matrix A is normalized [10].

(a) If a_{ij} is E-type coefficient 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 $TRUE$.

(b) If a_{ij} is G-type coefficient (i.e. first G-type for the current column is found) correct IND_i to $TRUE$. As A is normalized, the fact that a_{ij} is G-type coefficient means that $\widehat{b}_j = b_i$ and so $\widehat{x}_j = \widehat{b}_j = b_i$. In $\widehat{X}$ correct the value for $\widehat{x}_j$ to $\widehat{b}_j$.

In the current column, check if there exists other i such that a_{ij} is H type coefficient and $b_i = \widehat{b}_j$. If yes, correct the value for IND_i to $TRUE$. As the matrix A is normalized, this check can be done by continue walk successively through the coefficients in the current column until $b_i \neq \widehat{b}_j$.

Go to the next j.

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

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

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

Step 5. Exit.

Theorem 2 and its corollaries prove that if the system is solvable, $\widehat{X}$ computed by this algorithm is its greatest 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 [1], [4], [7], [8], [10], [11], [16], [18]) consistency of the system is established substituting $\widehat{X} = A^t \rightarrow_G B$ for X in (2): when $A \bullet (A^t \rightarrow_G B) = B$ holds the system is solvable, otherwise it is unsolvable. With Algorithm 1 $\widehat{X}$ can be obtained in more efficient way thus improving the time complexity. In addition there is no need to substitute $\widehat{X}$ in order to establish consistency of the system.

IND vector proposed first in [9] here is used for a similar purpose. The algorithm uses this 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 unsolvable.

Analysis of the computational complexity for Algorithm 1 is given in Section 6. In general it is between $O(n \log n) + O(m + n)$ and $O(n \log n) + O(m.n)$.

3.3. Lower solutions

The algorithm presented here proposes improvements of the theory from [10] in order to provide lower computational complexity and higher robustness.

It is important that every equation in the system (1) can be satisfied only through terms with H-type coefficients. Also, the minimal value for every component in the solution is either the value of the corresponding component in the vector B or 0 according to [9](Corollary 4). 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 b_i when the coefficient contributes to solve the system or 0 when it doesn't.

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

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

The task of the algorithm is to extract all lower solutions and to skip the second and third types (i.e. not lower solutions). In order to extract all lower solutions a new method, based on the idea of the *dominance matrix* [10] in combination with list manipulation techniques is developed here. A set of vectors (discussed in Section 4) is constructed and used to eliminate non-perspective H-type elements. After obtaining this set of vectors a new technique is used to extract all lower solutions from it.

3.3.1. 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 $j = 1, ..., n$ it holds: a_{lj} is H-type coefficient $\Rightarrow a_{kj}$ is also H-type coefficient.

3.3.2. Extracting lower solutions

Algebraic-logical approach is used up to now for finding all lower solutions [4], [6], [8], [10], algorithms are with exponential memory and time complexity [1]. 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 [11], [16].

Lower solutions are extracted by removing from A (or from $map(B)$) 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 coefficients $a_{\widetilde{i}^*j}$ 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 that 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 go to step 7.

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

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

(b) For all $\widetilde{k}$ in $rows$ if $a_{\widetilde{k}j}$ is a H-type coefficient, remove $\widetilde{k}$ from the copy of $rows$.

(c) Go to step 5 with copied in this step $rows$ and X_{low_0}, i.e. start new recursive branch with reduced $rows$ and changed X_{low_0}.

Step 6. Exit.

As this is a recursive algorithm the best explanations for it can be done with an example. A suitable example is given in Section 5 (Example 1).

3.4. Algorithms overview

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

Algorithm 3 Solving $A \bullet X = B$.

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

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

Step 3. If the system is unsolvable go to step 6.

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

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

Step 6. Exit.

Algorithm 3.3.2 (Step 5) is the slowest part of the Algorithm 3.4. Detailed complexity analysis can be found in Section 6. 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 3.3.2 is going to have the same time complexity as the algorithm presented in [10] only in its worst case.

Nevertheless Algorithm 3 is simpler than the ones proposed in [10], [11] (see Section 7), in next Section 4 another slight improvement is proposed.

4. H-positioning vectors

Definition 2. The vector H_i which elements are the numbers $j \in \{1, ..., n\}$ (ordered ascending) in the i^{th} row of A such that a_{ij} is H-type coefficient is called a *H-positioning vector*.

In general H-positioning vectors can be used as a replacement of the matrix $\widetilde{A}$. The most significant value of using H-positioning vectors is reducing the size $\widetilde{A}$ and thus reducing the memory and the computational complexity of Algorithm 3.3.2. The complexity for obtaining the set of H-positioning vectors is the same as the complexity for obtaining $\widetilde{A}$, but as the H-positioning vectors hold only the indexes of the H-type coefficient (and $\widetilde{A}$ holds the entire rows), the size of the set of the H-positioning vectors is reduced with as much elements as the number of S-type coefficient in the system (1). Another advantage of using H-positioning vectors is that they are more suitable for computer implementations of the algorithm.

Next is a revised version of Definition 1 with the same meaning but instead of rows in matrix, it uses H-positioning vectors.

Definition 3. Let H_l and H_k be two H-positioning vectors for (1) and $b_l \geq b_k$. If each number h_l (component of H_l) is a component of H_k, then H_l is called dominant vector to H_k and H_k is called dominated vector by H_l.

Obviously all dominated vectors in A correspond to dominated rows and so, the first step for obtaining the lower solutions of (1) is to obtain the set of all non-dominated H-positioning vectors ($\widetilde{H}$) according to Definition 3. After that the algorithm is almost the same as Algorithm 3.3.2. The only difference is a substitution of the matrix $\widetilde{A}$ with a $\widetilde{H}$ and the according changes on the indexes. Because of that and to keep this paper compact such an algorithm is not presented here.

5. Examples

In the first example we illustrate presented algorithms. The second example shows solving a bigger problem with a software based on Algorithm 4 and developed by the author. Execution time and a comparison with the software presented in [11] are given.

5.1. Example 1

Solve

$$\begin{pmatrix} 0.7 & 0 & 0.9 & 0.4 & 0.7 \\ 0.3 & 0.9 & 0.9 & 0.4 & 0.2 \\ 0.6 & 0.1 & 0 & 0.8 & 0.5 \\ 0.8 & 0.7 & 0.4 & 0.2 & 0.7 \\ 0.4 & 0.1 & 0.2 & 0.5 & 0.1 \end{pmatrix} \bullet \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} = \begin{pmatrix} 0.9 \\ 0.9 \\ 0.7 \\ 0.7 \\ 0.5 \end{pmatrix}$$

5.1.1. Finding the greatest solution (Algorithm 1)

Step 1. Initialize $\widehat{X} = (1.0\ \ 1.0\ \ 1.0\ \ 1.0\ \ 1.0)'$.

Step 2. Initialize $IND = (ind_i) = FALSE$ for each $i = 1, ..., m$.

Step 3. In the first column of the matrix A, a_{41} is the only G-type coefficient and $b_4 = 0.7 \Rightarrow \widehat{b}_1 = 0.7 \Rightarrow \widehat{x}_1 = 0.7$.

There is no E-type coefficient in this column, so only $IND_4 = TRUE$

Step 4. In the second column there is no G-type coefficient so $\widehat{b}_2 = 1 \Rightarrow \widehat{x}_2 = 1$.

Two E-type coefficients are located in this column: a_{22} and a_{42}. This means that $IND_2 = TRUE$. IND_4 is already $TRUE$ and there is no need to change it.

Step 5. In the third column there is no G-type coefficient so $\widehat{b}_3 = 1 \Rightarrow \widehat{x}_3 = 1$.

Two E-type coefficients are located in this column: a_{13} and a_{23}. This means that $IND_1 = TRUE$. IND_2 is already $TRUE$ and there is no need to change it.

Step 6. In the fourth column of the matrix A the only G-type coefficient is a_{34} and $b_3 = 0.7 \Rightarrow \widehat{b}_4 = 0.7 \Rightarrow \widehat{x}_4 = 0.7$. Also $a_{54} = \widehat{b}_4$ is E-type coefficient, so $IND_3 = TRUE$ and

$IND_5 = TRUE$. All elements in IND are set to $TRUE$ on this step. This means that no more checks for E-type coefficients should be performed and the system is solvable.

Step 7. In the fifth column there is no G-type coefficient, so $\widehat{x}_5 = 1$.

With this the greatest solution of the system is obtained: $\widehat{X} = (0.7 \quad 1 \quad 1 \quad 0.7 \quad 1)'$. Also, Theorem 2 and its corollaries prove that this is the greatest solution and no more verifications are needed.

5.1.2. Finding all lower solutions using H-positioning vectors (Algorithm 4)

Step 1. Form the set of H-positioning vectors

$$H = \{(3), (2 \;\; 3), (4), (1 \;\; 2 \;\; 5), (4)\}.$$

From Definition 3, the second vector is dominated by the first and the fifth is dominated by the third. As all dominated vectors are redundant they should be removed from the set. New initial set of H-positioning vectors is obtained:

$$H = \{(3), (4), (1 \;\; 2 \;\; 5)\}.$$

Step 2. Start with initial vector $X_{low_1} = (0 \;\; 0 \;\; 0 \;\; 0 \;\; 0)'$. Iterate through the first H-positioning vector (corresponding to the second equation with $b_2 = 0.9$ in the system). Since its only value is 3, we have $X_{low_1}(3) = b_2 = 0.9$ so $X_{low_1} = (0 \;\; 0 \;\; 0.9 \;\; 0 \;\; 0)'$.

Step 3. Iterate through the second H-positioning vector (corresponding to the third equation with $b_3 = 0.7$). Its only value is 4, $X_{low_1}(4) = b_3 = 0.7$. $X_{low_1} = (0 \;\; 0 \;\; 0.9 \;\; 0.7 \;\; 0)'$.

Step 4. From the third H positioning vector with $b_4 = 0.7$ all three solutions are obtained

$X_{low_1} = (0.7 \;\; 0.0 \;\; 0.9 \;\; 0.7 \;\; 0.0)'$

$X_{low_2} = (0.0 \;\; 0.7 \;\; 0.9 \;\; 0.7 \;\; 0.0)'$

$X_{low_1} = (0.0 \;\; 0.0 \;\; 0.9 \;\; 0.7 \;\; 0.7)'$

5.2. Example 2

Solve

$$\begin{pmatrix} 0.9 & 0.9 & 0.0 & 0.9 & 0.0 & 0.0 & 0.0 & 0.9 \\ 0.8 & 0.8 & 0.0 & 0.0 & 0.8 & 0.0 & 0.8 & 0.0 \\ 0.7 & 0.7 & 0.0 & 0.0 & 0.0 & 0.7 & 0.0 & 0.0 \\ 0.6 & 0.6 & 0.6 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.5 & 0.0 & 0.5 & 0.0 & 0.0 & 0.0 & 0.5 & 0.5 \\ 0.4 & 0.0 & 0.4 & 0.4 & 0.4 & 0.0 & 0.0 & 0.0 \\ 0.3 & 0.0 & 0.0 & 0.3 & 0.0 & 0.3 & 0.3 & 0.0 \\ 0.2 & 0.0 & 0.0 & 0.0 & 0.2 & 0.2 & 0.0 & 0.2 \end{pmatrix} \bullet \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \end{pmatrix} = \begin{pmatrix} 0.90 \\ 0.80 \\ 0.70 \\ 0.60 \\ 0.50 \\ 0.40 \\ 0.30 \\ 0.20 \end{pmatrix}$$

Although this example is not much bigger than the first one, it is much more hard, because it has 92 lower solutions. Because of this, only MATLAB session of solving this example with the developed by the author software is demonstrated here and comparison with the software from [11] is made. 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 [23].

5.2.1. 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 'maxmin' composition, empty matrix X and option for finding all lower solutions set as 'true'

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

4. Solve the system

   ```
   >> S.solve_inverse()
   ans =
     fuzzySystem handle
     Properties:
        composition: 'maxmin'
                  a: [8x8 fuzzyMatrix]
                  b: [8x1 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 greater solution ($x.gr$) and 92 lower solution ($x.low$)

   ```
   >> S.x
   ans =
   ```

```
       rows: 8
       cols: 8
       help: [8x8 fuzzyMatrix]
         gr: [8x1 fuzzyMatrix]
        ind: [8x1 double]
      exist: 1
  dominated: []
  help_rows: 8
        low: [8x92 fuzzyMatrix]
```

From the example it can be seen that this system has one greatest solution and 92 lower solutions. Because lack of space all the solution are not listed here. They can be obtained running the example with the software form [23].

5.2.2. Execution time comparison

On a test computer this example was solved with the presented above software in about 0.1 seconds, which is very impressive result. For comparison solving the same example on the same computer with the software from [11] took about 315 seconds.

6. Computational and memory complexity

In this section the terminology for computational complexity and for memory complexity is according to [15] and [17].

The problem has exponential computational complexity [1]. It actually depends on the the number of the lower solutions and if the system has a solution at all. Every step in this algorithm has its best and worst case.

Computational complexity for reordering the equations of the system is $O(m \log m)$ with memory complexity of $O(n)$. This is because not the entire system is reordered, but only its right hand side (vector $B_{m\times 1}$). Used algorithm is *binary tree sort*. For comparison, software package from [11] reorders both, matrix A and vector B, this leads to $O(n.m^2)$ computational and memory complexly.

Computational complexity for obtaining $\widehat{X}$ depends first of all on the consistency of the system. The worst case is when the system is unsolvable. In this case the algorithm needs to iterate through all the elements in the matrix A, so the time complexity is $O(m.n)$. Of course in this case, there is no sense to obtain any lower solutions and in general this is the fastest case for solving the problem. There are some best cases to obtain greatest solution if the system is solvable. All of them are with the same complexity. For instance one best case is when a system has G-type coefficients among entire first row and E-type coefficients among entire first column (a_{11} can be either E-type or G-type coefficient). In that case computational 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 lower solutions. Its computational complexity hardly depends on the number of non-dominated H-positioning vectors. From Definition 3 it is easy to see that for a system in n unknowns and m equations, the maximal number of non-dominated H-positioning vectors is less than or equal to $2^n - 1$. The components of these H-positioning vectors should be iterated with *backtracking* based Algorithm 4 with exponential time complexity.

Obtaining the set of all H-positioning vectors and checking for domination has complexity from a lower class and for this reason it is not discussed. But an extracted solution can be not lower 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 lower solutions.

7. Conclusion

Presented algorithm is a simple and straightforward way to solve fuzzy max-min systems of linear equations. It gives better and reasonable approach. The tremendous procedure based on the algebraic-logical properties from [10], [11] here is replaced with list manipulations. The software, created by the author, 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 [20].

Based on the presented here algorithm and software, a new software package is created by the author. The package is called Fuzzy Calculus Core (FC^{2}ore) [21], [23]. 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: $max - min$ and $min - max$ according to the theory in [11], $max - product$ according to the theory in [12], Gödel, Goguen and Łukasiewicz according to the theory in [13].

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 [23].

References

[1] Chen, L., P. Wang, Fuzzy relational equations (I): The general and specialized solving algorithm, *Soft Computing*, Vol. 6, 2002, 428–435.

[2] De Baets, B., 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.), Kluwer Academic Publishers, Vol. 1,2000, 291–340.

[3] Di Nola, A., W. Pedrycz, S. Sessa, E. Sanchez, *Fuzzy Relation Equations and Their Application to Knowledge Engineering*, Kluwer Academic Press, Dordrecht, 1989.

[4] Higashi, M., G. Klir, Resolution of finite fuzzy relation equations, *Fuzzy Sets and Systems*, Vol. 13, 1984, No. 1, 65–82.

[5] Klir, G., B. Yuan, *Fuzzy Sets and Fuzzy logic: Theory and Applications*, Prentice-Hall, PTR, Englewood Cliffs, NJ, 1995.

[6] Miyakoshi, M., M. Shimbo, Lower solutions of systems of fuzzy equations, *Fuzzy Sets and Systems*, Vol. 19, 1986, 37–46.

[7] Pappis, C., G. Adamopoulos, A computer algorithm for the solution of the inverse problem of fuzzy systems, *Fuzzy Sets and Systems*, Vol. 39, 1991 279–290.

[8] Pappis, C., M. Sugeno, Fuzzy relation equations and the inverse problem, *Fuzzy Sets and Systems*, Vol. 15, 1985, 79–90.

[9] Peeva, K., Fuzzy linear systems, *Fuzzy Sets and Systems*, Vol. 49, 1992, 339–355.

[10] Peeva, K., Universal algorithm for solving fuzzy relational equations, *Italian Journal of Pure and Applied Mathematics*, Vol. 19, 2006, 9–20.

[11] Peeva, K., 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.

[12] Peeva, K., Y. Kyosev, Algorithm for Solving Max-product Fuzzy Relational Equations, *Soft Computing*, Vol. 11, 2007, No. 7, 593–605.

[13] Perfilieva, I., L. Nosková, System of fuzzy relation equations with $inf - \rightarrow$ composition: Complete set of solutions, *Fuzzy Sets and Systems*, Vol. 159, 2008, 2256–2271.

[14] Sanchez, E., Resolution of composite fuzzy relation equations, *Information and Control*, Vol. 30, 1976, 38–48.

[15] Sedgewick, R., *Algorithms in C, Third Edition*, Pearson Education, Inc., publishing as Addison Wesley Longman, 1998.

[16] Shieh, B., New resolution of finite fuzzy relation equations with max-min composition, *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, Vol. 16, 2008, No. 1, 19–33.

[17] Wirth, N. *Algorithms + Data Structures = Programs*, Prentice-Hall, Inc., Englewood Cliffs, 1975.

[18] Wu, Y., S. Guu, An Efficient Procedure for Solving a Fuzzy Relational Equation With Max-Archimedean t-Norm Composition, *IEEE Transactions on Fuzzy Systems*, Vol. 16, 2008, No. 1, 73–84.

[19] Yeh, C. On the minimal solutions of max–min fuzzy relational equations, *Fuzzy Sets and Systems*, Vol. 159, 2008, 23–39.

[20] Zahariev, Z., Solving Max-min Relational Equations. Software and Applications, *Proc. of International Conference "Applications of Mathematics in Engineering and Economics (AMEE'08)"*, Vol. 1067, G. Venkov, R. Kovatcheva, V. Pasheva (eds.), American Institute of Physics, 2008, 516–523.

[21] Zahariev, Z., Software package and API in MATLAB for working with fuzzy algebras, *Proc of International Conference "Applications of Mathematics in Engineering and Economics (AMEE'09)"*, vol. 1184, G. Venkov, R. Kovatcheva, V. Pasheva (eds.), American Institute of Physics, 2009, 434–350.

[22] `http://www.mathworks.com/matlabcentral/fileexchange/6214-fuzzy-relational-calculus-toolbox-rel-1-01`

[23] `http://www.mathworks.com/matlabcentral/fileexchange/27046-fuzzy-calculus-core-fc2ore`