










# NCMPy: A Modelling Software for Neutrosophic Cognitive Maps based on Python Package

Ilanthenral Kandasamy<sup>1,\*</sup> , Divakar Arumugam<sup>1</sup> , Aryan Rathore<sup>1</sup> , Ateeth Arun<sup>1</sup> , Manan Jain<sup>1</sup> , Vasantha .W.B<sup>1</sup> , and Florentin Smarandache<sup>2</sup> 

<sup>1</sup> School of Computer Science and Engineering (SCOPE), Vellore Institute of Technology, Vellore, 632014, Tamil Nadu, India.

Emails: ilanthenral.k@vit.ac.in; divakara.s2019@vitstudent.ac.in; aryan.2019@vitstudent.ac.in; ateeth.arun2019@vitstudent.ac.in; manan.jain2019@vitstudent.ac.in; vasantha.wb@vit.ac.in.

<sup>2</sup> University of New Mexico, Mathematics, Physics, and Natural Sciences Division 705 Gurley Ave., Gallup, NM 87301, USA; smarand@unm.edu

\* Correspondence: ilanthenral.k@vit.ac.in

**Abstract:** Cognitive maps are a vital tool that can be used for knowledge representation and reasoning. Fuzzy Cognitive Maps (FCMs) are popular soft computing techniques used to model large and complex systems, and they can aid in explainable artificial intelligence (AI). FCMs, however, cannot model the indeterminacy that arises in a system due to various uncertainties. Neutrosophic Cognitive Maps (NCMs), upgraded FCMs that could model indeterminacy, were introduced to address this issue. NCMs are a generalization of FCMs, a field of cognitive science firmly based on neural networks. NCMs have been used to solve a wide range of problems. NCMs were introduced in 2002, and even after 20 years, NCMs do not have any supportive software, package, toolbox, or visualization software like FCMs. The main reason for the absence of dedicated software is due to the indeterminacy concept 'I' and how it has to be handled. This paper presents the dedicated Python package created for handling the functioning of NCMs. The modelling software presented in this paper aids in visualizing the NCMs as a signed digraph with indeterminacy that is a directed signed neutrosophic graph. This package implements a sample case study using NCMs.

**Keywords:** Neutrosophy; Neutrosophic Cognitive Maps; Python Package; Visualization of NCMs.

## 1. Introduction

Fuzzy theory is a branch of mathematics that deals with vagueness and uncertainty in decision-making [1]. Fuzzy sets and logic model complex problems involving imprecise terms or partial truths. It has many applications in engineering fields, the healthcare sector, economics, and social science, which pertain to real-world problems. Fuzzy logic and its models have many applications in various fields, such as engineering, artificial intelligence, medicine, economics, and social problems. It can help model multifaceted problems that involve human knowledge, preferences, or emotions [2].

A fuzzy cognitive map (FCM) [3] represents a mental landscape within which the connections between the nodes (e.g., events, concepts, resources, or attributes) are used to compute the "strength of impact" of these elements. FCMs are signed fuzzy digraphs introduced by Bart Kosko [3].

FCMs have been used to analyze several socio-economic, healthcare, and decision-making problems. The applications and extensions of FCMs are vast and widely researched; a few are presented here.

In [4], the authors explore using FCMs as an agency for collective decision-making and how FCMs can capture the cognitive models and group beliefs of different stakeholders. FCMs were used as a learning assessment tool in [5] to understand the planning of children by stimulating cognitive function.

In [6], the Multi-Agent Genetic Algorithm (MAGA) is proposed to optimize convergence error for learning FCMs. A GIS-dependent crisis management tool to predict earthquakes in Tehran using FCMs was proposed in [7].

An evolutionary algorithm, known as IBMTEA-FCM, was proposed in [8] for learning large-scale FCMs. A qualitative analytical method using FCMs to specify causal-effect links between the interdependent SDGs by considering the long-term effect of COVID-19 was presented in [9].

In [10], the PRescriptiVe FCM (PRV-FCM) was introduced, based on FCMs and metaheuristic algorithms, to develop prescriptive models. In [11], three federated learning approaches were combined with FCMs for mortality prediction and treatment prescription in severe dengue cases.

Iran's population's health was analyzed [12] using FCMs. Since the concept of health is a complex and comprehensive system, other sector policies profoundly affect health. Borrero-Domínguez and Escobar-Rodríguez [13] proposed a decision support system for crowdfunding using FCMs. The various extensions of FCMs have been systemically reviewed by [14].

Several software packages are available for modelling FCMs. One such software, Mental Modeler, helps build FCMs intuitively and easily. After creating the models, decreasing or increasing the model's elements allows us to examine various change tactics. FCMexpert is a software tool for FCM-based scenario analysis and pattern classification presented in [15]. Over 10 FCM extensions were handled by supporting interoperability in the FCM extensions in [16].

Python packages are also available for modelling FCMs. FCMpy [17] is a recently introduced open-source package for building and analyzing FCMs. Notably, FCMpy allows simulating system behaviour using qualitative data to create fuzzy causal weights, applying ML algorithms to modify the FCMs matrix to aid in classification, and executing scenario examination by simulating theoretical interventions.

The package also helps apply ML algorithms (e.g., nonlinear and active Hebbian learning, deterministic learning, and genetic algorithms) to adjust the FCM weight matrix.

Neutrosophy is a branch of philosophy investigating neutralities' origin, nature, and scope and their interactions. Florentin Smarandache introduced neutrosophy in the 1990s [18]. Neutrosophy regards a proposition, hypothesis, concept, event, or entity depending on the modelling. Neutrosophy is the basis of the neutrosophic set, logic, probability, and statistics. Indeterminacy ("I") is a concept in neutrosophy that measures the degree of neutrality or uncertainty of a proposition, event, theory, entity, or concept.

Neutrosophic Cognitive Maps (NCMs) are an extension of FCMs that can handle indeterminate relationships between two concepts, obtaining more significant and sensitive results. It was introduced in [19] to analyze diverse social issues. NCMs have been modelled considerably on the AI focus to mimic the thinking-humanly approach. Here, it is unsupervised data and has a limited set of features.

Over the past two decades, many investigators have utilized NCMs to analyze diverse problems like situation analysis [20], pest analysis [21], transgressions against people experiencing homelessness [22], and imaginative play in children [23]. FCMs and NCMs on COVID variants were compared in [24]. SWOT analysis and NCMs were combined to analyze organic farming in India [25]. Al-Subhi et al. [26] proposed triangular NCMs and used them in multistage decision-making with a use case of evaluation. NCMs and cloud data were used in [27] to detect violence, and several datasets were used. NCMs and FCMs were compared in this analysis, and it clearly states that NCMs are better at handling indeterminacy than FCMs.

Dynamic NCMs [28], enhanced cuckoo search, and ensemble classifiers were presented for acquiring the profile of gene expression and differentiating between the individuals affected by rheumatoid arthritis and possible control subjects. Bhutani et al. [29] proposed a technique combining pest analysis based on fuzzy and neutrosophic logic to analyze the food industry.

In [30], the various factors impacting the paper-packaging industry are analyzed to provide a notional representation using NCMs since sustainable supply chains can be attained with repeated product use and recycling.

In [31], NCMs were used to analyze the various causes and effects that lead to violent behaviour. NCMs were used to determine the elements that enable proper decision-making to confirm a precise diagnosis of conversion disorder [32]. The various factors that affect homeless people were analyzed using NCMs in [33]. A neutrosophic sociogram-based NCM approach was introduced in [34]. FCMs and NCMs have been used in health care to analyze dengue fever [35].

NCMs have been applied in various domains such as health, agriculture, engineering, social problems, business, law, environment, and medicine. The substantial advantage of NCMs over other cognitive maps is their capacity to capture data realistically and consistently represent expert opinions, making them a valuable tool for decision-making strategies where there is an advanced degree of indeterminacy or uncertainty. NCMs can be constructed either by a data-driven approach or experts' opinions.

NCMs have not been integrated with machine-learning algorithms like FCMs. So, little research combines various machine learning algorithms that have been utilized in adjusting weights in an NCM, like in FCM. Despite the various applications of NCMs, there is no dedicated software or Python package for NCMs. This paper presents a dedicated software and Python package for constructing, analyzing, and visualizing NCMs.

The dedicated modules of our modelling software function in the following way:

- i. Generating the neutrosophic graph and the related connection matrix using expert opinion
  - From linguistic terms.
  - From edge weights.
  - From a file as input from the user.
- ii. Visualizing the NCMs as a neutrosophic digraph.
- iii. Simulating the NCMs using various state vectors.
- iv. Analysis of various case scenarios for given NCMs.

The paper is organized as follows: Section 2 recalls the workings of NCMs and their construction. Section 3 provides the software-based visualization and simulation of NCM for a case study. The conclusions and results are presented in the last section, together with suggestions for future research.

## 2. Working of Neutrosophic Cognitive Maps (NCMs)

When data is unsupervised and the association between two concepts is indeterminate, the indeterminacy can be captured by using neutrosophy. [19] introduced the concept of indeterminacy in FCMs, called NCMs. The basic properties of NCM are recalled to make this section self-contained.

NCM is a digraph with concepts as nodes and their causal relationships as edges. These concepts can be events, strategies, or policies as nodes of the graph and relationships as edges, where each concept is mathematically represented as a neutrosophic vector from the neutrosophic vector space. Every node, in its vector form, is represented by  $(x_1, \dots, x_n)$ ;  $x_i \in \{0, 1, I\}$ , where 0 is off state, 1 is on state and  $I$  is the indeterminate state.

Consider two nodes  $N_a$  and  $N_b$  of the NCM; their relationship is given by the edge  $e_{ab}$ . Every weighted edge  $e_{ab}$  is from  $\{-1, 0, 1, I\}$ , where 0 means no impact, positive value means increase in  $N_a$  implies increase in  $N_b$ , similarly decrease in  $N_a$  implies decreases in  $N_b$ . If  $e_{ab}$  takes a negative value like  $-1$ , it implies that an increase in  $N_a$  implies a decrease in  $N_b$ , or similarly, a decrease in  $N_a$  implies an increase in  $N_b$ . The edge weight is assigned a value  $I$  if the effect from  $N_a$  to  $N_b$  can not be determined. The edge weights of simple NCMs are from  $\{-1, 0, 1, I\}$ . The NCM's adjacency matrix is denoted by  $N(E) = (e_{ab})$ ,  $e_{ab} \in \{-1, 0, 1, I\}$ .

NCMs are expert opinions constructed based on data obtained from the expert, where they identify the factors or concepts relevant to the domain and associated causal relationships in terms of numbers and indeterminacy or linguistic terms.

Generally, the neutrosophic connection matrix is obtained directly from the digraph of the expert's opinion. Here, we have introduced the concept of dealing with neutrosophic linguistic terms using the following logic. Algorithm 1 provides neutrosophic edge weights from the linguistic terms.

---

**Algorithm 1: Generate NCM Matrix**


---

```

Data: matrix ← Empty Matrix; row ← Empty List; n ← No of nodes
Result: NCM Matrix;
i, j ← 0 while i ≠ n do
  string ← ""
  while j ≠ n do
    term ← input linguistic term
    if i = j then
      element ← 0
    else
      if term = "-VH" then
        element ← random(-5, -2.75)
      else if term = "-H" then
        element ← random(-2.75, -2)
      else if term = "-M" then
        element ← random(-2, -1.5)
      else if term = "-L" then
        element ← random(-1.5, -1)
      else if term = "-VL" then
        element ← random(-1, 0)
      else if term = "NC" then
        element ← 0
      else if term = "+VH" then
        element ← random(2.75, 5)
      else if term = "+H" then
        element ← random(2, 2.75)
      else if term = "+M" then
        element ← random(1.5, 2)
      else if term = "+L" then
        element ← random(1, 1.5)
      else if term = "+VL" then
        element ← random(0, 1)
      else if term = "NaN" then
        element ← I
    string ← string + element + ','
    j ← j + 1
  for x ∈ temp do
    if x = "I" then
      continue
  row ← row.append(temp)
  i ← i + 1
i ← 0
while i ≠ n do
  matrix ← matrix.row_insert(i, row[i])
  i ← i + 1
return matrix

```

---

In the case of linguistic terms, these edges can be considered as negative very high (-VH), negative high (-H), negative medium (-M), negative low (-L), and negative very low (-VL). It implies a negative influence when the decrease of the influence of the node  $N_a$  results in the increase of the influence of the node  $N_b$  or the increase of  $N_a$  results in the decrease of  $N_b$ . Similarly, these edges can be considered as positive very high (+VH), positive high (+H), positive medium (+M), positive low (+L), and positive very low (+VL). It implies a positive influence when the decrease of  $N_a$  results in the decrease of  $N_b$  or the increase of  $N_a$  results in the increase of  $N_b$ . If the linguistic term is no causality, 0 is used; if it is indeterminate, "NaN" (not a number) is used. These linguistic terms are converted to numerical values using simple assignments as follows:

- If *term* = "-VH" then the element gets a random value from [-5, -2.75).
- If *term* = "-H", then the element gets a random value from [-2.75, -2).
- If *term* = "-M", then the element gets a random value from [-2, -1.5).

- If  $term = "-L"$ , then the element gets a random value from  $[-1.5, -1]$ .
- If  $term = "-VL"$ , then the element gets a random value from  $[-1, 0]$ .
- If  $term = "-NC"$  then the element gets 0.
- If  $term = "+VH"$  then the element gets a random value from  $(2.75, 5]$ .
- If  $term = "+H"$ , then the element gets a random value from  $(2, 2.75]$ .
- If  $term = "+M"$ , then the element gets a random value from  $(1.5, 2]$ .
- If  $term = "+L"$ , then the element gets a random value from  $(1, 1.5]$ .
- If  $term = "+VL"$ , then the element gets a random value from  $(0, 1]$ .
- If  $term = "NaN"$  then the element gets  $I$ .

The values generated by the algorithm range from  $[-5, 5]$ ; since the non-indeterminate edge weights of NCM are from  $[-1, 1]$ , the edge weights are normalized before the NCM/neutrosophic digraph is constructed.

The expert opinion obtained can be used to generate the neutrosophic adjacency matrix of the NCMs. Generation can be done by using linguistic terms. According to the linguistic terms, the neutrosophic matrix can be obtained using random values between the ranges.

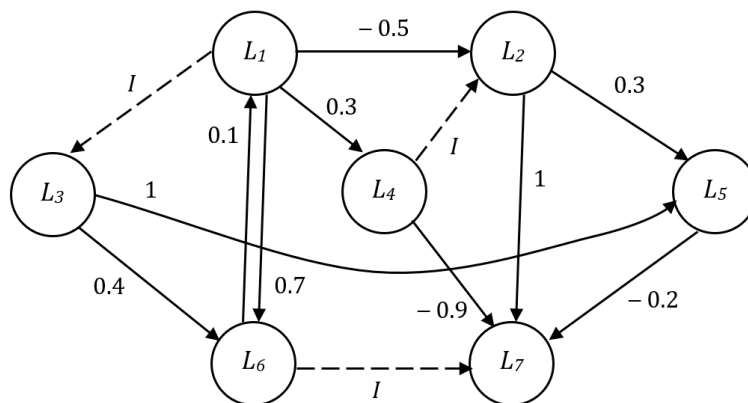
**Example 1:** Consider the graph of the NCM, using the seven concepts (or nodes or attributes)  $L_1, L_2, \dots, L_7$  and the expert opinion is obtained in terms of linguistic terms, using Algorithm 1, the weights for the edges are provided.

For illustration, assuming that the connection from  $L_2$  to  $L_7$  has a very high positive influence, and the connection between from  $L_1$  to  $L_3$  is indeterminate, and the connection from  $L_4$  to  $L_7$  is a very high negative influence. The edge weights generated by the algorithm using the linguistic terms are tabulated in the Table 1:

**Table 1.** The edge weights assigned by the algorithm.

	$L_1$	$L_2$	$L_3$	$L_4$	$L_5$	$L_6$	$L_7$
$L_1$	0	-2.5	I	1.5	0	3.5	0
$L_2$	0	0	0	0	1.5	0	5
$L_3$	0	0	0	0	5	2	0
$L_4$	0	I	0	0	0	0	-4.5
$L_5$	0	0	0	0	0	0	-1
$L_6$	0.5	0	0	0	0	0	I
$L_7$	0	0	0	0	0	0	0

The NCM will be given in Figure 1. Each edge is weighted and directed. The dashed lines are used to represent indeterminate edges. The weight of each edge is given in the graph.



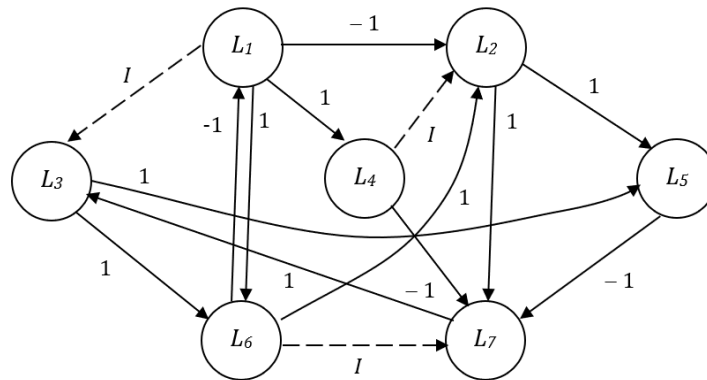
**Figure 1.** An illustration of the neutrosophic directed graph.

It is then normalized to weigh between  $[-1, 1]$  and indeterminacy  $I$ . The connection matrix for the neutrosophic directed graph is given in Eq. (1).

$$E_{normalized} = \begin{pmatrix} 0 & -0.5 & I & 0.3 & 0 & 0.7 & 0 \\ 0 & 0 & 0 & 0 & 0.3 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0.4 & 0 \\ 0 & I & 0 & 0 & 0 & 0 & -0.9 \\ 0 & 0 & 0 & 0 & 0 & 0 & -0.2 \\ 0.1 & 0 & 0 & 0 & 0 & 0 & I \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \quad (1)$$

The edge weights can also be obtained from the expert. The edges of simple NCMs are from  $\{-1, 0, 1, I\}$ .

**Example 2:** Consider a simple NCM given by an expert with seven concepts  $L_1, L_2, \dots, L_7$  as the nodes of the directed neutrosophic graph. The expert opinion is obtained in terms of edge weights.



**Figure 2.** Neutrosophic directed graph for simple NCM.

The edge weights are from  $\{-1, 0, 1, I\}$ , and the indeterminate edges are represented by dotted lines. The NCM's connection matrix is denoted by  $N(E) = (e_{ab})$ , where  $e_{ab} \in \{0, 1, -1, I\}$ .

$$N(E) = \begin{pmatrix} 0 & -1 & I & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & I & 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 \\ 1 & 1 & 0 & 0 & 0 & 0 & I \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix} \quad (2)$$

The notion of state vector, dynamical system and its functioning are described. The neutrosophic state vector  $S = (s_1, \dots, s_n); s_i \in \{0, 1, I\}$ ; where 0 indicates the off state, 1 is in the on state, and  $I$  indicates the indeterminate state. Let  $\overline{L_1L_2}, \overline{L_2L_3}, \overline{L_3L_4}, \dots, \overline{L_nL_1}$  be the NCM's directed edges. Given the edges creating a cycle, the NCM becomes cyclic; otherwise, it is acyclic. Consider  $\overline{L_1L_2}, \overline{L_2L_3}, \dots, \overline{L_{n-1}L_n}$  to be cyclic, if node  $L_a$  is on the influence will flow via the existing cycle and cause  $L_a$  to be on again. This state of equilibrium of the dynamical system is called as a hidden pattern.

Various state vectors with different nodes in on state are considered to activate the system Consider the NCM with feedback given in Figure 2; its neutrosophic adjacency matrix is  $N(E)$  given in Eq. (2).

The state vector  $S_1 = (1, 0, 0, , 0)$  where  $L_1$  is in on state is considered. The data needs to be transformed by  $N(E)$ , so we multiply  $S_1$  by  $N(E)$ .

The state vector multiplied by the neutrosophic matrix  $N(E)$  is given in Algorithm 3. In the given algorithm,  $S$  is a 1-dimensional matrix (row vector) and the neutrosophic adjacency matrix  $N(E)$  is a 2-dimensional matrix denoted by  $B$ . The resultant vector Res of the multiplication of  $S \times N(E)$  is returned.

An example illustrates this: Consider the graph in Figure 2 and its related connection matrix. Take the state vector  $S_1 = (1\ 0\ 0\ 0\ 0\ 0\ 0)$ . The state vector  $S_1$  is multiplied with the neutrosophic adjacency matrix  $N(E)$ .

$$S_1 \times N(E) = (1\ 0\ 0\ 0\ 0\ 0\ 0) \times \begin{pmatrix} 0 & -1 & I & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & I & 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 \\ 1 & 1 & 0 & 0 & 0 & 0 & I \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{3}$$

$$= (0\ -1\ I\ 1\ 0\ 1\ 0) \tag{4}$$

---

**Algorithm 2: Multiply**

---

**Data:** 1D Matrix  $S$  and  $B \leftarrow$  2D Neutrosophic Adjacency Matrix  $N(E)$   
**Result:** Resultant Matrix  
 $Res \leftarrow$  1D Matrix  
 $r \leftarrow S.size$   
 $R \leftarrow B.size$   
 $i, j \leftarrow 0$   
**while**  $i \neq r$  **do**  
    **while**  $j \neq R$  **do**  
         $Res[i] \leftarrow Res[i] + (S[i] * B[j][i])$   
         $j \leftarrow j + 1$   
     $i \leftarrow i + 1$   
**return**  $Res$

---

After obtaining the resultant vector Res, it must undergo the threshold and update operations. By the definition of NCM, the threshold and update operation is denoted by the  $\hookrightarrow$  symbol.

It is important to note here that working with  $I$  (indeterminate) needs to be done carefully. As by the definition of  $I$

$$I \times I = I^2 = I \tag{5}$$

Any power of  $I$  gets mapped to  $I$ , as shown in Algorithm 3.

---

**Algorithm 3: UpdateIPower**

---

**Data:**  $V \leftarrow$  state vector  
**Result:**  $V$  state vector after updating  $I$ 's  
 $i \leftarrow 0$   
**for**  $i \in V$  &  $j \in b$  **do**  
    **if**  $i = I^2$  **then**  $i \leftarrow I$   
**return**  $V$

---

Similarly, during the updating and threshold operation, any constant into  $I$  is also mapped into  $I$ , as shown in Eq. (6)

$$n \times I = I \tag{6}$$

The resultant vector from the multiplication of the state vector with  $N(E)$  is thresholded and updated.

Let  $X = S_1 N(E) = (s_1, s_2, \dots, s_n)$  is thresholded by replacing  $s_i$  accordingly to the Eq. (7).

$$s_i = \begin{cases} 1 & \text{if } s_i > t \\ 0 & \text{if } s_i < t \text{ ( } t \text{ is a suitable positive integer)} \\ I & \text{if } s_i \text{ not an integer} \end{cases} \tag{7}$$

The resultant  $X$  is updated to ensure that the state considered on in the initial state vector  $S_1$  is on in the resulting vector. Here, it is updated to make the concept  $N_1$  as 1 in the resulting vector.

The algorithm for thresholding and updating is given in Algorithm 4.

---

**Algorithm 4: ThresholdandUpdate**

---

**Data:**  $X \leftarrow$  resultant vector and  $t$  threshold value, Initial On state

**Result:** Thresholded and updated resultant vector

$X \leftarrow$  updateIPower( $X$ )

$len \leftarrow X.size$

$i \leftarrow 0$

**while**  $i \neq len$  **do**

$temp\_expr \leftarrow X[i]$

**if**  $temp\_expr = J$  **then**

$temp\_expr \leftarrow 0$

**if**  $temp\_expr \geq t$  **then**

$X[i] \leftarrow t$

**else if**  $temp\_expr = 0$  **then**

**if**  $X[i] \neq 0$  **then**

$X[i] \leftarrow J$

**else**

$X[i] \leftarrow 0$

$i \leftarrow i + 1$

$X[state - 1] \leftarrow 1$

**return**  $X$

---

Here, the Algorithm 4 illustrates where only one node is taken in the on state in the initial state vector.

The thresholding and updating operation is mathematically denoted by  $\hookrightarrow$ . For example, consider the resultant vector of  $S_1 \times N(E)$ , the thresholding and updating result in  $S_2$ .

$$S_1 \times N(E) = (0 \ -1 \ 1 \ 1 \ 0 \ 1 \ 0) \hookrightarrow (1 \ 0 \ 1 \ 1 \ 0 \ 1 \ 0) = S_2 \tag{8}$$

Here, during the threshold operation,  $-1$  is made 0, and during the update operation, the very first state is made on again.

Generally, in any NCM, more than one concept/ node can be considered in the on state. This will deal with the combined effect of both states being on. The proposed model can handle more than one node in the on state, and it can work to analyse the effect of a combination of various nodes. The multiplication of resultant vectors with the neutrosophic adjacency matrix  $N(E)$  will continue until the resultant vector yields a fixed point or limit cycle.

If the NCM settles to a neutrosophic state vector repeating in the form

$$S_1 \hookrightarrow S_2 \hookrightarrow S_j \dots \hookrightarrow S_i \hookrightarrow S_j,$$

Then the dynamic system's equilibrium is called NCM's limit cycle. Suppose,  $S_1 N(E) \hookrightarrow S_2$  (where  $\hookrightarrow$  denotes the resultant vector of  $S_1 N(E)$  which is thresholded and updated) and for  $S_2 N(E)$  we repeat the same procedure until we attain the fixed point / limit cycle.



Two vectors are compared in Algorithm 5.

---

**Algorithm 5: Compare**

---

**Data:**  $a, b$  - 1D vectors  
**Result:** Boolean value  
 $i, j \leftarrow 0$   
**for**  $i \in a$  &  $j \in b$  **do**  
    **if**  $i \neq j$  **then**  
        **return** *false*  
**return** *true*

---

A fixed point or limit cycle is attained when both the vectors under comparison are the same. For example, if we compare  $S_1$  with  $S_2$ , we can see that  $S_1 \neq S_2$ .

$$(1\ 0\ 0\ 0\ 0\ 0) \neq (1\ 0\ 1\ 1\ 0\ 1) = S_2 \tag{9}$$

Since the fixed point nor limit cycle is reached, i.e.,  $S_1 \neq S_2$ , the process is continued.

$$S_2 \times N(E) = (1\ 1\ 1\ 1\ 1\ 1 + I - 1 + I) \hookrightarrow (1\ 1\ 1\ 1\ 1\ 0) = S_3 \tag{10}$$

Since the fixed point still needs to be reached, i.e.,  $S_3 \neq S_2$ , the process is continued.

$$\begin{aligned} S_3 \times N(E) &= (1 + I\ 2I\ 2I\ 1\ 2I\ 1 + I - 1 + 2I) \\ &\hookrightarrow (1\ 1\ 1\ 1\ 1\ 0) = S_4 \end{aligned} \tag{11}$$

Here  $S_3 = S_4$ . The fixed point has been reached.

The Algorithm 6 is used to determine if a limit cycle is reached. It compares with the previous resultant vectors using the previously described in compare Algorithm 5 to check if the limit cycle is reached. In case it is reached, it returns a true or a false.

---

**Algorithm 6: Check\_cycle**

---

**Data:**  $b \leftarrow$  list of vectors  
**Result:** Boolean value  
 $n \leftarrow b.size$   
 $i \leftarrow 0$   
**while**  $i \neq n - 1$  **do**  
     $j \leftarrow i + 1$   
    **while**  $j \neq n$  **do**  
         $v1 \leftarrow b[i]$   
         $v2 \leftarrow b[j]$   
        **if**  $compare(v1, v2) = true$  **then**  
            **return** *true*  
         $j \leftarrow j + 1$   
     $i \leftarrow i + 1$   
**return** *false*

---

To illustrate an example of limit cycle, consider the state vector  $P_1 = (0\ 1\ 0\ 0\ 0\ 0)$ , and the connection matrix  $N(E)$  as given in Eq. (2), that is the related adjacency matrix for the NCM given in Figure 2.

$$\begin{aligned} P_1 \times N(E) &= (0\ 0\ 0\ 0\ 1\ 0\ 1) \hookrightarrow (0\ 1\ 0\ 0\ 1\ 0\ 1) = P_2 \\ P_2 \times N(E) &= (0\ 0\ 1\ 0\ 1\ 0\ 0) \hookrightarrow (0\ 1\ 1\ 0\ 1\ 0\ 0) = P_3 \\ P_3 \times N(E) &= (0\ 0\ 0\ 0\ 2\ 1\ 0) \hookrightarrow (0\ 1\ 0\ 0\ 1\ 1\ 0) = P_4 \\ P_4 \times N(E) &= (1\ 1\ 0\ 0\ 1\ 0\ 1) \hookrightarrow (1\ 1\ 0\ 0\ 1\ 0\ 1) = P_5 \end{aligned}$$

$$\begin{aligned}
 P_5 \times N(E) &= (0 - 1 \ 2 * I \ 1 \ 1 \ 1 \ 0) \\
 &\hookrightarrow (0 \ 1 \ I \ 1 \ 1 \ 1 \ 0) = P_6 \\
 P_6 \times N(E) &= (1 \ I + 1 \ 0 \ 0 \ I + 1 \ I \ I - 1) \\
 &\hookrightarrow (1 \ 1 \ 0 \ 0 \ 1 \ I \ 0) = P_7 \\
 P_7 \times N(E) &= (I \ I - 1 \ I \ 1 \ 1 \ 1 \ I^2) \\
 &\hookrightarrow (I \ I \ I \ 1 \ 1 \ 1 \ I) = P_8 \\
 P_8 \times N(E) &= (1 \ 1 \ I^2 + I \ I \ I + 1 \ 2 * I \ I - 1) \\
 &\hookrightarrow (1 \ 1 \ I \ I \ 1 \ I \ 0) = P_9 \\
 P_9 \times N(E) &= (I \ I^2 + I - 1 \ I \ I \ I + 1 \ I + 1 \ I^2 - I) \\
 &\hookrightarrow (I \ I \ I \ 1 \ 1 \ 1 \ 0) = P_{10} \\
 P_{10} \times N(E) &= (1 \ 1 \ I^2 \ I \ I + 1 \ 2 * I \ I - 1) \\
 &\hookrightarrow (1 \ 1 \ I \ I \ 1 \ I \ 0) = P_{11} = P_9
 \end{aligned} \tag{12}$$

Since  $P_{11} = P_9$ , the iteration is stopped since a limit cycle has been achieved, enabling the determination of the hidden pattern. The limit cycle is as follows:  $P_8$  gives  $P_9$ ,  $P_9$  gives  $P_{10}$ ,  $P_{10}$  gives  $P_{11}$ ; that is same as  $P_9$ .

The process of multiplication of the resultant vector with the matrix  $N(E)$  is continued until a limit cycle / fixed point is reached.

The Algorithm 7 takes an adjacency/connection matrix and state vector (an integer as input denoting the state to be activated) and an integer denoting the threshold values as parameters.

---

**Algorithm 7:** Iteration

---

**Data:** an Adjacency matrix  $E$ ,  $state \leftarrow$  state to be activated,  $t$  threshold value

**Result:** list of vectors

$len \leftarrow E.size$

$c1 \leftarrow list[len]$  filled with 0

$c1[state - 1] \leftarrow 1$

$start \leftarrow Matrix(c1)$

$flag \leftarrow false$

$vectors \leftarrow$  Empty List

**while**  $flag = false$  **do**

$y \leftarrow multiply(start, E)$

$y \leftarrow thresholdAndUpdate(y, t)$

$vectors \leftarrow vectors.append(y)$

$start \leftarrow y$

$flag \leftarrow check\_cycle(vectors)$

**return**  $vectors$

---

The algorithm 7 continues multiplying the resultant vector with the  $N(E)$  until a fixed point or a limit cycle is reached. It is dependent on several previously described algorithms. Every concept must be made in the active state to capture the hidden pattern and understand the effect of the concept or node on others to analyse the NCM thoroughly. Only in NCMs can we signify that the clout of a node on different nodes can be indeterminate, and this vision needs to be revised in the case of FCMs.

For the given example, the results have been tabulated in Table 2 for various state vectors.

Table 2. Determining the Hidden Pattern.

Input state vector	limit cycle / fixed point	Resultant vector
(1 0 0 0 0 0)	fixed point	(1 1 1 1 1 0)
(0 1 0 0 0 0)	limit cycle	(1 1 1 1 1 0)
		(1 1 1 1 1 0)
		(1 1 1 1 1 0)
		(1 1 1 1 1 0)
(0 0 1 0 0 0)	fixed point	(1 1 1 1 1 0)
(0 0 0 1 0 0)	fixed point	(0 1 0 1 1 0 0)
(0 0 0 0 1 0)	fixed point	(0 0 0 0 1 0 0)
(0 0 0 0 0 1)	fixed point	(1 1 1 1 1 0)
(0 0 0 0 0 0 1)	fixed point	(1 1 1 1 1 1 1)

It is seen that only the active state of concept  $L_2$  results in a limit cycle—the rest results in a fixed point.

The conclusions that can be drawn from the equilibrium state of the dynamic system are as follows: When node  $L_1$  is active, all nodes are either indeterminate or on state. When  $L_2$  is in the on state, it results in a limited cycle, which affects all nodes other than  $L_7$ . Similarly, when nodes  $L_3, L_4, L_6$  or  $L_7$  alone are in the on state, other nodes are either indeterminate or on state. Whereas when node  $L_5$  is in on state, no other node is affected; all of them remain in the off state.

### 3. Description of the modelling package

The overall flow of the modelling software is as given in Figure 3. The first module is for the input module, which can either be linguistic term-based or edge-weight-based.

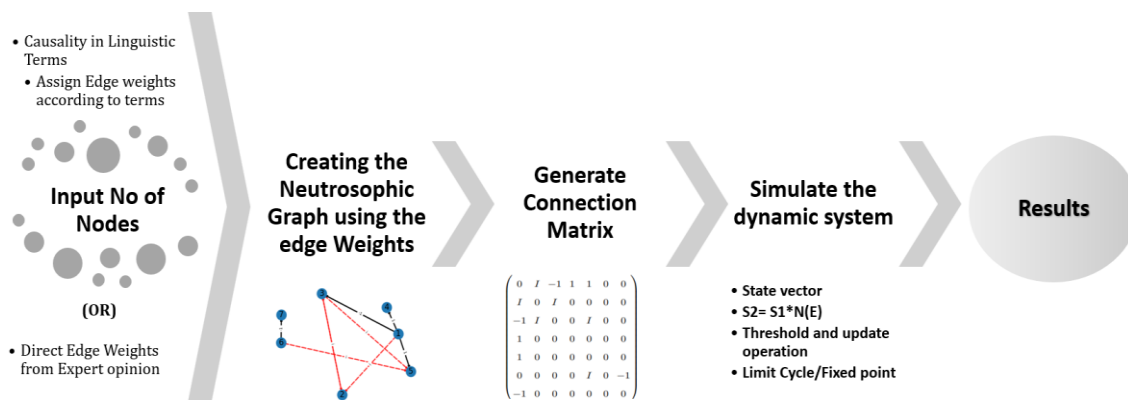


Figure 3. Various modules of NCM modelling software.

1. Input module: There are two methods in which the model can be created using expert opinion. They can enter by either method as described below:
  - *Linguistic terms*: The user can enter the linguistic terms to describe the relationship between two concepts in NCM as very negative or positive. According to the previously discussed algorithm, the edge weights are assigned based on the linguistic term.
  - *Edge weights*: The user can instead directly enter the edge weight to denote the causality between two nodes.
2. Neutrosophic digraph: The neutrosophic digraph is generated and visualised using the edge weights obtained from the user.

3. Connection matrix: The related connection matrix  $N(E)$  is obtained from the neutrosophic digraph of the NCM.
4. Dynamical system: Using various state vectors, the dynamical system is simulated to analyse the effect of the on state of the various nodes.
5. Results: The effect of the on state of various nodes and combination of various states is consolidated.

The NCMPy python package performs a simulation of the dynamic system—a detailed description of the package is given in the next subsection.

### 3.1 Description of the NCMPy Python Package

The flowchart of the NCMPy package is given in the following Figure 4.

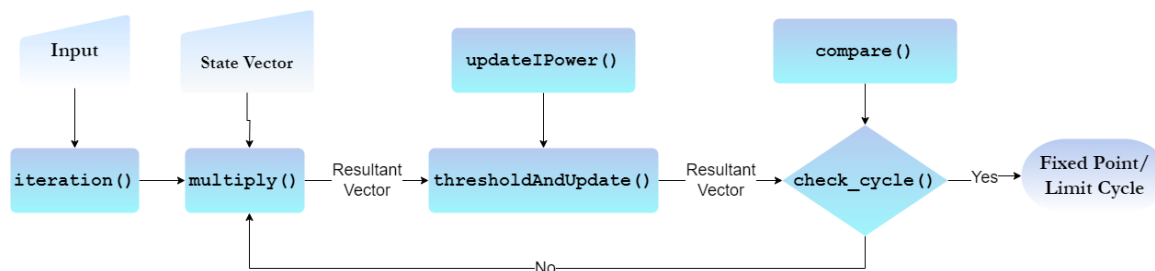


Figure 4. Flowchart of the NCM package.

The SymPy library has been used to handle indeterminate values in modelling the NCMs in this proposed package. In SymPy package, the names E, I, O, S, N, and Q collide with names already defined in the package. Hence,  $I$  can not be used to represent indeterminacy, so  $J$  is used instead of  $I$ . Throughout the coding snippets,  $J$  is used instead of  $I$ .

The various functions/modules are described here.

1. compare(x, y): This function takes two vectors as input and checks for their equality; it is dependent on Algorithm 6. The code snippet is as follows:

```

def compare(x, y):
    res = True
    for k in range(len(x)):
        if x[k] != y[k]:
            return False
    return True
    
```

Assume that compare function takes  $S_7 = (I I I I I I 0)$  and  $S_8 = (I I I I I I 0)$  as  $x$  and  $y$ . In that case, compare(x, y) will return True, in case it was  $S_5 = (1 I I 1 I I I)$  and  $S_6 = (I I I 1 I I 0)$  under consideration, then it would return False.

2. check\_cycle(b): This function is dependent on Algorithm 6; it is used to check if the resultant vector is a fixed point or limit cycle.

```

def check_cycle(b) :
    for i in range(len(b) - 1) :
        for j in range(i + 1 , len(b)) :
            v1 = b[i]
            v2 = b[j]
            if(compare(v1 , v2) == True) :
                return True
    return False
    
```

The fixed point is achieved when the resultant vector is the same as the previous resultant vector, that is  $P_i N(E) \hookrightarrow P_i$ . The limit cycle is achieved when the recently calculated state vector is the same as any one of the previously calculated resultant vector, which results in a cycle.

$$P_i N(E) \hookrightarrow P_{i+1}; P_{i+1} N(E) \hookrightarrow P_{i+2}; \dots P_x N(E) \hookrightarrow P_i;$$

In Example 2, considering the active state  $P_1 = (0 \ 1 \ 0 \ 0 \ 0 \ 0)$  results in a limit cycle.

3. `updateIPower(x)`: This function takes a 1D vector as input and converts the indeterminate quadratic polynomial to a linear polynomial. This function is based on Algorithm 3. This function is used by the threshold and update function.

```
def updateIPower(x):
    for i in range((np.shape(x))[1]):
        if x[i].find(J**2):
            x[i] = x[i].xreplace({J**2: J})
    return x
```

Consider the resultant vector;

$$R_1 = (J \ ** \ 2 \ J \ J \ + \ 1 \ 1 \ 0 \ J \ ** \ 2 \ J)$$

For a sample scenario. The `updateIPower(R_1)` will change this vector  $P_1$  into

$$R_1 = (J \ J \ J \ + \ 1 \ 1 \ 0 \ J \ 2 \ J)$$

4. `thresholdAndUpdate(X,threshold_value,state)` : This function takes a 1D vector and a threshold value as a parameter and updates each vector value according to the defined thresholding operation. Also, the updation operation must see to it that the node which was on in the initial state is on in the next state and so on in the resultant state also; if not, it is set to 1 again.

```
def thresholdAndUpdate(X , threshold_value, state) :
    X=updateIPower(X)
    for i in range((np.shape(X)[1])):
        if(X[i].find(-1)):
            X[i]=0
            temp_expr=X[i].subs(J,0)
            if(temp_expr>=threshold_value):
                X[i]=X[i].subs(X[i],threshold_value)
            elif(temp_expr==0):
                if(X[i].find(J)):
                    X[i]=J
            else:
                X[i]=X[i].subs(X[i],0)
    if isinstance(state, list):
        activated_states = [i for i, x in enumerate(state) if x == 1]
        for s in activated_states:
            X[s] = 1
    else:
        X[state - 1] = 1
    return X
```

$$X_{10} \times N(E) = (I + 1I + 1III + 12 * II - 1) \hookrightarrow (1 \ 1 \ 1 \ 1 \ 1 \ 0) = X_{11}$$

5. `iteration(E,state,threshold_value)` : This function takes an adjacency/connection matrix, an integer denoting the state to be activated and an integer denoting the threshold values as parameters. This function is based on the Algorithm 7 .

```

def iteration(E , state, threshold_value = 1) :
    if isinstance(state, list):
        start = sym.Matrix(state)
    else:
        c1 = np.zeros((np.shape(E)[1]))
        c1[state - 1] = 1
        start = sym.Matrix(c1)

    flag = False
    start = start.T
    vectors = []
    while flag == False :
        y = multiply(start , E)
        # Performing the thresholding operation on output vector y
        y = thresholdAndUpdate(y , threshold_value, state)
        vectors.append(y)
        # Updating start vector to start with new state vector
        start = y
        # Checking for cycle among state vectors
        flag = check_cycle(vectors)
    return vectors

```

Consider Example 2, where the working out is done with active state  $P_1 = (0\ 1\ 0\ 0\ 0\ 0)$ ; it iterates until the limit cycle is achieved.

### 3.2 Modelling NCMs for sample case study

The opening page for the modelling software is given in Figure 5. Here, the user can select the option of working with linguistic terms or directly entering the neutrosophic edge weights given by the expert as shown in Figure 6.



Figure 5. Homepage.

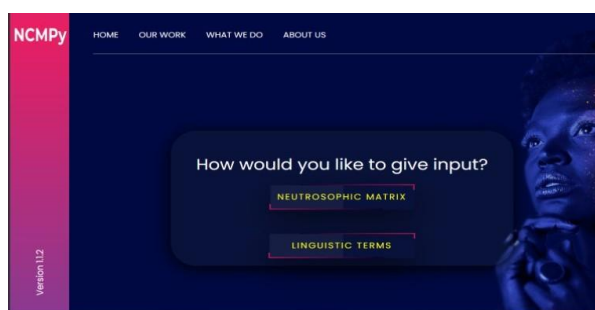


Figure 6. Option selection.

### 3.3 NCMs using Linguistic Terms

The neutrosophic linguistic terms are obtained from the user, as shown in Figure 7.

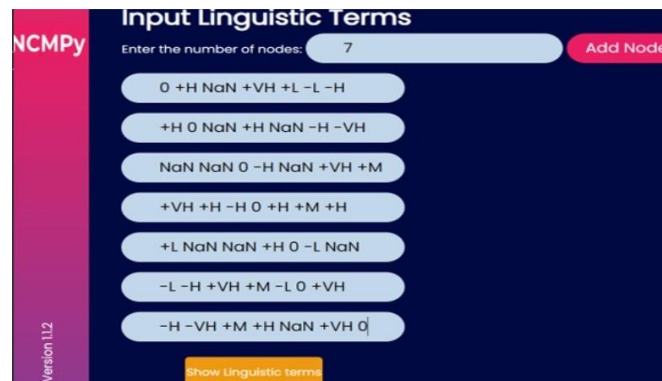


Figure 7. Sample input page for the linguistic terms related to the neutrosophic directed graph.

The NCMpy package runs through the Alogrithm GenerateNCM 2 and creates the necessary edge weights and normalises them as shown in Table 3, providing the neutrosophic bigraphs. The neutrosophic-directed graph of the NCM is given in Figure 8.

Table 3. The edge weights assigned.

	$L_1$	$L_2$	$L_3$	$L_4$	$L_5$	$L_6$	$L_7$
$L_1$	0	4.88	I	3.58	1.91	-2.39	-2.81
$L_2$	4.22	0	I	2.13	I	-2.74	-4.68
$L_3$	I	I	0	-4.77	I	3.23	1.93
$L_4$	2.98	1.58	-2.41	0	2.41	1.81	3.11
$L_5$	1.22	I	I	4.2	0	-2.12	I
$L_6$	-1.20	-4.36	4.03	1.71	-2.40	0	2.85
$L_7$	-2.03	-4.77	1.34	3.89	I	4.44	0

The resultant neutrosophic connection matrix of the graph is

```
[
[0, 0.98, J, 0.72, 0.38, -0.48, -0.56],
[0.84, 0, J, 0.43, J, -0.55, -0.94],
[J, J, 0, -0.95, J, 0.65, 0.39],
[0.60, 0.32, -0.48, 0, 0.48, 0.36, 0.62],
[0.24, J, J, 0.84, 0, -0.42, J],
[-0.24, -0.87, 0.81, 0.34, -0.48, 0, 0.57],
[-0.41, -0.95, 0.27, 0.78, J, 0.89, 0]
]
```

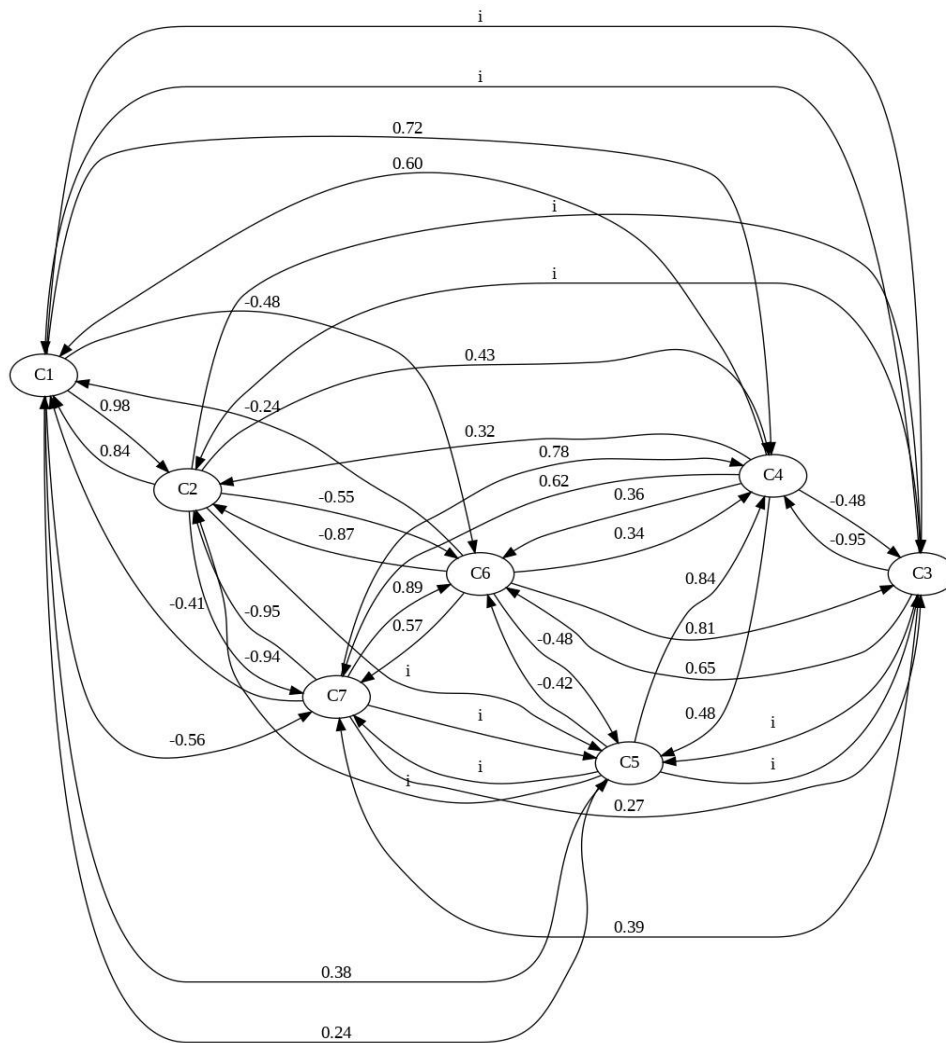


Figure 8. Neutrosophic directed graph.

To show the plotting capacity of the visualizing module, we have taken a matrix with all connections for the sample. The threshold value is obtained from the user. According to the threshold value set by the user, the thresholding and updating of state vectors are done.

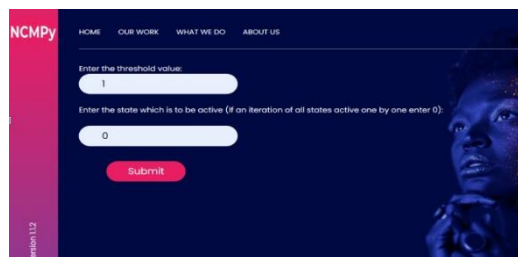


Figure 9. Insert the threshold value and state vector.

The threshold value of 1 was taken here, and the following results were obtained for various state vectors. The working out for the first state vector  $S_1 = (1\ 0\ 0\ 0\ 0\ 0\ 0)$ , where the concept  $C_1$  is on is shown here:



```

FOR ACTIVE STATE 1
Matrix([[0, 0.98, 1.0*J, 0.72, 0.38, -0.48, -0.56]])
Matrix([[0, 0.98, 1.0*J, 0.72, 0.38, -0.48, -0.56]])
Matrix([[0, 0.98, 1.0*J, 0.72, 0.38, -0.48, -0.56]])
Matrix([[0, 0, 1.0*J, 0.72, 0.38, -0.48, -0.56]])
Matrix([[0, 0, J, 0.72, 0.38, -0.48, -0.56]])
Matrix([[0, 0, J, 0, 0.38, -0.48, -0.56]])
Matrix([[0, 0, J, 0, 0, -0.48, -0.56]])
Matrix([[0, 0, J, 0, 0, 0, -0.56]])
After Thresholding and Updating
Matrix([[1, 0, J, 0, 0, 0, 0]])
Matrix([[J**2, J**2 + 0.98, J, 0.72 - 0.95*J, J**2 + 0.38, 0.65*J - 0.48, 0.39*J - 0.56]])
Matrix([[J, J + 0.98, J, 0.72 - 0.95*J, J + 0.38, 0.65*J - 0.48, 0.39*J - 0.56]])
Matrix([[J, J + 0.98, J, 0.72 - 0.95*J, J + 0.38, 0.65*J - 0.48, 0.39*J - 0.56]])
Matrix([[J, 0, J, 0.72 - 0.95*J, J + 0.38, 0.65*J - 0.48, 0.39*J - 0.56]])
Matrix([[J, 0, J, 0.72 - 0.95*J, J + 0.38, 0.65*J - 0.48, 0.39*J - 0.56]])
Matrix([[J, 0, J, 0, J + 0.38, 0.65*J - 0.48, 0.39*J - 0.56]])
Matrix([[J, 0, J, 0, 0, 0.65*J - 0.48, 0.39*J - 0.56]])
Matrix([[J, 0, J, 0, 0, 0, 0.39*J - 0.56]])
After Thresholding and Updating
Matrix([[1, 0, J, 0, 0, 0, 0]])
[Matrix([[1, 0, J, 0, 0, 0, 0]]), Matrix([[1, 0, J, 0, 0, 0, 0]])]

```

Similarly, the working out for each and every state vector is carried out. For state vector  $B_1 = (0\ 1\ 0\ 0\ 0\ 0)$ , where the concept  $C_2$  is on, the resultant vectors will be:

```

FOR ACTIVE STATE 2
Matrix([[0.84, 0, 1.0*J, 0.43, 1.0*J, -0.55, -0.94]])
After Thresholding and Updating
Matrix([[0, 1, J, 0, J, 0, 0]])
Matrix([[J**2 + 0.24*J + 0.84, 2*J**2, J**2 + J, 0.43 - 0.11*J, J**2 + J, 0.23*J - 0.55, J**2 + 0.39*J - 0.94]])
After Thresholding and Updating
Matrix([[0, 1, J, 0, J, 0, 0]])
Resultant vector(s) [Matrix([[0, 1, J, 0, J, 0, 0]]), Matrix([[0, 1, J, 0, J, 0, 0]])]

```

For state vector  $G_1 = (0\ 0\ 1\ 0\ 0\ 0)$ , where the concept  $C_3$  is on, the resultant vectors will be:

```

FOR ACTIVE STATE 3
Matrix([[1.0*J, 1.0*J, 0, -0.95, 1.0*J, 0.65, 0.39]])
After Thresholding and Updating
Matrix([[J, J, 1, 0, J, 0, 0]])
Matrix([[2.1*J, J**2 + 2.0*J, 3*J**2, 2.0*J - 0.95, J**2 + 1.4*J, 0.65 - 1.4*J, J**2 - 1.5*J + 0.39]])
After Thresholding and Updating
Matrix([[J, J, 1, 0, J, 0, 0]])
Resultant vector(s) [Matrix([[J, J, 1, 0, J, 0, 0]]), Matrix([[J, J, 1, 0, J, 0, 0]])]

```

For state vector  $X_1 = (0\ 0\ 0\ 1\ 0\ 0)$ , where the concept  $C_4$  is on, the resultant vectors will be:

```

FOR ACTIVE STATE 4
Matrix([[0.60, 0.32, -0.48, 0, 0.48, 0.36, 0.62]])
After Thresholding and Updating
Matrix([[0, 0, 0, 1, 0, 0, 0]])
Matrix([[0.60, 0.32, -0.48, 0, 0.48, 0.36, 0.62]])
After Thresholding and Updating
Matrix([[0, 0, 0, 1, 0, 0, 0]])
Resultant vector(s) [Matrix([[0, 0, 0, 1, 0, 0, 0]]), Matrix([[0, 0, 0, 1, 0, 0, 0]])]

```

For state vector  $Y_1 = (0\ 0\ 0\ 0\ 1\ 0)$ , where the concept  $C_5$  is on, the resultant vectors will be:

```

FOR ACTIVE STATE 5
Matrix([[0.24, 1.0*J, 1.0*J, 0.84, 0, -0.42, 1.0*J]])
After Thresholding and Updating
Matrix([[0, J, J, 0, 1, 0, J]])
Matrix([[J**2 + 0.43*J + 0.24, J**2 + 0.05*J, J**2 + 1.3*J, 0.26*J + 0.84, 3*J**2, 0.99*J - 0.42, 0.45*J]])
After Thresholding and Updating
Matrix([[0, J, J, 0, 1, 0, J]])
Resultant vector(s) [Matrix([[0, J, J, 0, 1, 0, J]]), Matrix([[0, J, J, 0, 1, 0, J]])]

```

For state vector  $Z_1 = (0\ 0\ 0\ 0\ 0\ 1)$ , where the concept  $C_6$  is on, the resultant vectors will be:

```

FOR ACTIVE STATE 6
Matrix([[-0.24, -0.87, 0.81, 0.34, -0.48, 0, 0.57]])
After Thresholding and Updating
Matrix([[0, 0, 0, 0, 0, 1, 0]])
Matrix([[-0.24, -0.87, 0.81, 0.34, -0.48, 0, 0.57]])
After Thresholding and Updating
Matrix([[0, 0, 0, 0, 0, 1, 0]])
Resultant vector(s) [Matrix([[0, 0, 0, 0, 0, 1, 0]]), Matrix([[0, 0, 0, 0, 0, 1, 0]])]

```

```
FOR ACTIVE STATE 7
Matrix([[ -0.41, -0.95, 0.27, 0.78, 1.0*J, 0.89, 0]])
After Thresholding and Updating
Matrix([[0, 0, 0, 0, J, 0, 1]])
Matrix([[0.24*J - 0.41, J**2 - 0.95, J**2 + 0.27, 0.84*J + 0.78, J, 0.89 - 0.42*J, J**2]])
After Thresholding and Updating
Matrix([[0, 0, 0, 0, J, 0, 1]])
Resultant vector(s) [Matrix([[0, 0, 0, 0, J, 0, 1]]), Matrix([[0, 0, 0, 0, J, 0, 1]])]
```

For state vector  $A_1 = (0\ 0\ 0\ 0\ 0\ 1)$ , where the concept  $C_7$  is on, the resultant vectors will be:

The result vector in each case has been shown. Results regarding the resultant vectors that can be discussed

1. Maximum Influence: The nodes  $C_2, C_3$  and  $C_5$  are the most influential since they do affect many other nodes and turn to an indeterminate state.
2. Least Influential nodes: The nodes  $C_4$  and  $C_6$  are the least influential since they do not affect any other node than itself.

### 3.4 NCMs using Edge Weights

The edge weights are obtained from the expert, as shown in Figure 10.

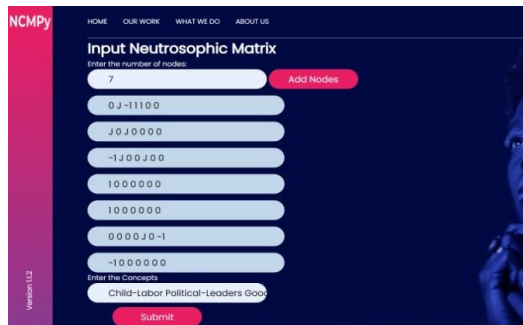


Figure 10. Input screen.

Using the edge weights obtained, the NCM is created. It is the visualization of the same represented as a NCMs digraph as shown in Figure 11.

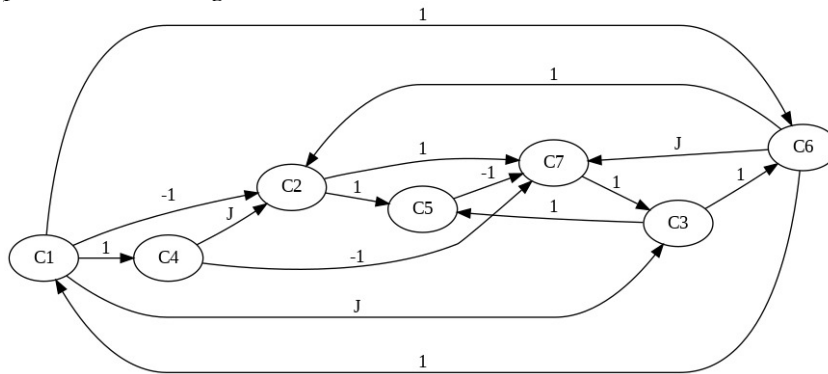


Figure 11. Visualization of the neutrosophic graph.

The related neutrosophic connection matrix  $N(E)$  is given in Eq. (2). The obtained matrix is

$$\begin{bmatrix} [0, -1, 'J', 1, 0, 1, 0], \\ [0, 0, 0, 0, 1, 0, 1], \\ [0, 0, 0, 0, 1, 1, 0], \\ [0, 'J', 0, 0, 0, 0, -1], \\ [0, 0, 0, 0, 0, 0, -1], \\ [1, 1, 0, 0, 0, 0, 'J'], \\ [0, 0, 1, 0, 0, 0, 0] \end{bmatrix}$$

The threshold value of 1 was taken here, and the following results were obtained for various state vectors. The working out for the first state vector  $S_1 = (1\ 0\ 0\ 0\ 0\ 0)$ , where the concept  $C_1$  is on is shown here:

```
FOR ACTIVE STATE 1
Matrix([[0, -1.0000000000000000, 1.0*J, 1.0000000000000000, 0, 1.0000000000000000, 0]])
After Thresholding and Updating Matrix([[1, 0, J, 1, 0, 1, 0]])
Matrix([[1, J, J, 1, J, J + 1, J - 1]])
After Thresholding and Updating Matrix([[1, J, J, 1, J, 1, 0]])
Matrix([[1, J, J, 1, 2*J, J + 1, J - 1]])
After Thresholding and Updating Matrix([[1, J, J, 1, J, 1, 0]])
Resultant vector(s) [Matrix([[1, 0, J, 1, 0, 1, 0]]), Matrix([[1, J, J, 1, J, 1, 0]]), Matrix([[1, J, J, 1, J, 1, 0]])]
```

The resultant vectors for all active states are given in the snippet below:

```
FOR ACTIVE STATE 2
Resultant vector(s) [Matrix([[0, 1, 0, 0, 1, 0, 1]]), Matrix([[0, 1, 1, 0, 1, 0, 0]]), Matrix([[0, 1, 0,
0, 1, 1, 0]]), Matrix([[1, 1, 0, 0, 1, 0, J]]), Matrix([[0, 1, J, 1, 1, 1, 0]]), Matrix([[1, 1, 0, 0, 1,
J, 0]]), Matrix([[J, 1, J, 1, J, 1, 1, J]]), Matrix([[1, 1, J, J, 1, J, 0]]), Matrix([[J, 1, J, 1, 1, 1,
0]]), Matrix([[1, 1, J, J, 1, J, 0]])]
FOR ACTIVE STATE 3
Resultant vector(s) [Matrix([[0, 0, 1, 0, 1, 1, 0]]), Matrix([[1, 1, 1, 0, 1, 1, 0]]), Matrix([[1, 0, 1,
1, 1, J, 1]]), Matrix([[1, J, 1, 1, 1, 1, 0]]), Matrix([[1, J, 1, 1, 1, 1, 0]])]
FOR ACTIVE STATE 4
Resultant vector(s) [Matrix([[0, J, 0, 1, 0, 0, 0]]), Matrix([[0, J, 0, 1, J, 0, 0]]), Matrix([[0, J, 0,
1, J, 0, 0]])]
FOR ACTIVE STATE 5
Resultant vector(s) [Matrix([[0, 0, 0, 0, 1, 0, 0]]), Matrix([[0, 0, 0, 0, 1, 0, 0]])]
FOR ACTIVE STATE 6
Resultant vector(s) [Matrix([[1, 1, 0, 0, 0, 1, J]]), Matrix([[1, 0, J, 1, 1, 1, 1]]), Matrix([[1, J, 1,
1, J, 1, 0]]), Matrix([[1, J, J, 1, 1, 1, 0]]), Matrix([[1, J, J, 1, J, 1, 0]]), Matrix([[1, J, J, 1, J,
1, 0]])]
FOR ACTIVE STATE 7
Resultant vector(s) [Matrix([[0, 0, 1, 0, 0, 0, 1]]), Matrix([[0, 0, 1, 0, 1, 1, 1]]), Matrix([[1, 1, 1,
0, 1, 1, 1]]), Matrix([[1, 0, 1, 1, 1, 1, 1]]), Matrix([[1, J, 1, 1, 1, 1, 1]]), Matrix([[1, J, 1, 1, 1,
1, 1]])]
```

The combination of two concepts being in the on state is also handled by the package done there. Similarly, several other combinations can be done, according to the user's choice. A sample of the combinations of various concepts begins in on state, and the working is given in the snippet.

```
Activating state: [1, 1, 0, 0, 0, 0, 0]
[Matrix([[1, 1, J, 1, 1, 1, 1]]), Matrix([[1, 1, 1, 1, 1, 1, 0]]), Matrix([[1, 1, J, 1, 1, 1, 0]]),
Matrix([[1, 1, J, 1, 1, 1, 0]])]
Activating state: [1, 0, 1, 0, 0, 0, 0]
[Matrix([[1, 0, 1, 1, 1, 1, 0]]), Matrix([[1, J, 1, 1, 1, 1, 0]]), Matrix([[1, J, 1, 1, 1, 1, 0]])]
Activating state: [1, 0, 0, 1, 0, 0, 0]
[Matrix([[1, 0, J, 1, 0, 1, 0]]), Matrix([[1, J, J, 1, J, 1, 0]]), Matrix([[1, J, J, 1, J, 1, 0]])]
Activating state: [1, 0, 0, 0, 1, 0, 0]
[Matrix([[1, 0, J, 1, 1, 1, 0]]), Matrix([[1, J, J, 1, 1, 1, 0]]), Matrix([[1, J, J, 1, 1, 1, 0]])]
Activating state: [1, 0, 0, 0, 0, 1, 0]
[Matrix([[1, 0, J, 1, 0, 1, J]]), Matrix([[1, J, J, 1, J, 1, 0]]), Matrix([[1, J, J, 1, J, 1, 0]])]
Activating state: [1, 0, 0, 0, 0, 0, 1]
[Matrix([[1, 0, 1, 1, 0, 1, 1]]), Matrix([[1, J, 1, 1, 1, 1, 1]]), Matrix([[1, J, 1, 1, 1, 1, 1]])]
Activating state: [0, 1, 1, 0, 0, 0, 0]
[Matrix([[0, 1, 1, 0, 1, 1, 1]]), Matrix([[1, 1, 1, 0, 1, 1, J]]), Matrix([[1, 1, 1, 1, 1, 1, J]]),
Matrix([[1, 1, 1, 1, 1, 1, 0]]), Matrix([[1, 1, 1, 1, 1, 1, 0]])]
```

The complete NCMPy python package will be made available online.

#### 4. Conclusions and discussions

As the world moves towards no code or less coding paradigms, a Python package for NCM will aid and help mathematicians, social scientists, economists, strategists, and other policymakers analyze real-world problems without worrying about the mathematical background or computational complexities of NCM. A visualization tool and a Python package to help in the working of NCM were presented in this paper. The NCMPy package and modelling software provide

the functions essential to studying problems involving indeterminacy, which can be done using NCMs.

This package and modelling tool are open-source, written in Python, straightforward to implement, and provide the required functionality for handling models with indeterminacy.

This tool implementation is a collaboration with the founding and leading experts in the field of NCMs. This tool will facilitate research and enable new researchers and scientists to apply NCMs to their projects that involve indeterminacy. We plan to update our library and constantly welcome all scientific community contributions.

### **Acknowledgments**

The author is grateful to the editorial and reviewers, as well as the correspondent author, who offered assistance in the form of advice, assessment, and checking during the study period.

### **Data availability**

The datasets generated during and/or analyzed during the current study are not publicly available due to the privacy-preserving nature of the data but are available from the corresponding author upon reasonable request.

### **Conflict of interest**

The authors declare that there is no conflict of interest in the research.

### **Ethical approval**

This article does not contain any studies with human participants or animals performed by any of the authors.

### **References**

1. Zadeh, L. A. (1978). Fuzzy sets as a basis for a theory of possibility. *Fuzzy sets and systems*, 1(1), 3-28. [https://doi.org/10.1016/0165-0114\(78\)90029-5](https://doi.org/10.1016/0165-0114(78)90029-5).
2. Felix, G., Nápoles, G., Falcon, R., Froelich, W., Vanhoof, K., & Bello, R. (2019). A review on methods and software for fuzzy cognitive maps. *Artificial intelligence review*, 52, 1707-1737. <https://doi.org/10.1007/s10462-017-9575-1>.
3. Kosko, B. (1986). Fuzzy cognitive maps. *International journal of man-machine studies*, 24(1), 65-75. [https://doi.org/10.1016/S0020-7373\(86\)80040-2](https://doi.org/10.1016/S0020-7373(86)80040-2).
4. Gray, S. A., Zanre, E., & Gray, S. R. (2013). Fuzzy cognitive maps as representations of mental models and group beliefs. In *Fuzzy cognitive maps for applied sciences and engineering: From fundamentals to extensions and learning algorithms* (pp. 29-48). Berlin, Heidelberg: Springer Berlin Heidelberg. [https://doi.org/10.1007/978-3-642-39739-4\\_2](https://doi.org/10.1007/978-3-642-39739-4_2).
5. Barón, H. B., Crespo, R. G., Pascual Espada, J., & Martínez, O. S. (2015). Assessment of learning in environments interactive through fuzzy cognitive maps. *Soft Computing*, 19, 1037-1050. <https://doi.org/10.1007/s00500-014-1313-x>.
6. Yang, Z., & Liu, J. (2020). Learning fuzzy cognitive maps with convergence using a multi-agent genetic algorithm. *Soft Computing*, 24, 4055-4066. <https://doi.org/10.1007/s00500-019-04173-2>.
7. Choukolaei, H. A., Mirani, S. E., Hejazi, M., & Ghasemi, P. (2023). A GIS-based crisis management using fuzzy cognitive mapping: PROMETHEE approach (a potential earthquake in Tehran). *Soft Computing*, 1-22. <https://doi.org/10.1007/s00500-023-09260-z>.
8. Wang, C., Liu, J., Wu, K., & Ying, C. (2021). Learning large-scale fuzzy cognitive maps using an evolutionary many-task algorithm. *Applied Soft Computing*, 108, 107441. <https://doi.org/10.1016/j.asoc.2021.107441>.
9. Ameli, M., Esfandabadi, Z. S., Sadeghi, S., Ranjbari, M., & Zanetti, M. C. (2023). COVID-19 and Sustainable Development Goals (SDGs): Scenario analysis through fuzzy cognitive map modeling. *Gondwana Research*, 114, 138-155. <https://doi.org/10.1016/j.gr.2021.12.014>.

10. Hoyos, W., Aguilar, J., & Toro, M. (2023). PRV-FCM: An extension of fuzzy cognitive maps for prescriptive modeling. *Expert Systems with Applications*, 120729. <https://doi.org/10.1016/j.eswa.2023.120729>.
11. Hoyos, W., Aguilar, J., & Toro, M. (2023). Federated learning approaches for fuzzy cognitive maps to support clinical decision-making in dengue. *Engineering Applications of Artificial Intelligence*, 123, 106371. <https://doi.org/10.1016/j.engappai.2023.106371>.
12. Salehnia, N., Arani, A. A., Olyaeemanesh, A., & Saghdel, H. S. (2022). Analysis of Casual Relationships between Social Determinants of Health in Iran: Using Fuzzy Cognitive Map. *Journal of Research in Health Sciences*, 22(4). <https://doi.org/10.34172/jrhs.2022.101>.
13. Borrero-Domínguez, C., & Escobar-Rodríguez, T. (2023). Decision support systems in crowdfunding: A fuzzy cognitive maps (FCM) approach. *Decision Support Systems*. <https://doi.org/10.1016/j.dss.2023.114000>
14. Schuerkamp, R., & Giabbanelli, P. J. (2023). Extensions of fuzzy cognitive maps: A systematic review. *ACM Computing Surveys*, 56(2), 1-36. <https://doi.org/10.1145/3610771>.
15. Nápoles, G., Espinosa, M. L., Grau, I., & Vanhoof, K. (2018). FCM expert: software tool for scenario analysis and pattern classification based on fuzzy cognitive maps. *International Journal on Artificial Intelligence Tools*, 27(07), 1860010. <https://doi.org/10.1142/S0218213018600102>.
16. Schuerkamp, R., Giabbanelli, P. J., & Daclin, N. (2023, May). Facilitating the Interoperability and Reuse of Extensions of Fuzzy Cognitive Maps. In *2023 Annual Modeling and Simulation Conference (ANNSIM)* (pp. 708-719). IEEE.
17. Mkhitarian, S., Giabbanelli, P., Wozniak, M. K., Nápoles, G., De Vries, N., & Crutzen, R. (2022). FCMpy: a python module for constructing and analyzing fuzzy cognitive maps. *PeerJ Computer Science*, 8, e1078. <https://doi.org/10.7717/peerj-cs.1078>.
18. Smarandache, F. (2000). Neutrosophy. arXiv preprint math/0010099.
19. Kandasamy, W. V., & Smarandache, F. (2003). Fuzzy cognitive maps and neutrosophic cognitive maps. *Infinite Study*.
20. Zafar, A., & Wajid, M. A. (2019). Neutrosophic cognitive maps for situation analysis. *Neutrosophic Sets and Systems*, 29-78.
21. Alava, R. P., Murillo, J. M., Zambrano, R. B., Vélez, M. I. Z., & Vazquez, M. L. (2018). PEST Analysis Based on Neutrosophic Cognitive Maps: A Case Study for Food Industry. *Neutrosophic Sets and Systems*, 21, 84-92.
22. Palacios, A. J. P., Ricardo, J. E., Piza, I. A. C., & Herrería, M. E. E. (2021). Phenomenological hermeneutical method and neutrosophic cognitive maps in the causal analysis of transgressions against the homeless. *Neutrosophic sets and systems*, 44, 147-156.
23. Vasantha, W. B., Kandasamy, I., Devvrat, V., & Ghildiyal, S. (2019). Study of imaginative play in children using neutrosophic cognitive maps model. *Neutrosophic Sets and Systems*, 30, 241-252.
24. Murugesan, R., Parthiban, Y., Devi, R., Mohan, K. R., & Kumarave, S. K. (2023). A Comparative Study of Fuzzy Cognitive Maps and Neutrosophic Cognitive Maps on Covid Variants. *Neutrosophic Sets and Systems*, 55(1), 20.
25. Obbineni, J., Kandasamy, I., Vasantha, W. B., & Smarandache, F. (2023). Combining SWOT analysis and neutrosophic cognitive maps for multi-criteria decision making: a case study of organic agriculture in India. *Soft Computing*, 1-22. <https://doi.org/10.1007/s00500-023-08097-w>.
26. Al-subhi, S. H., Papageorgiou, E. I., Pérez, P. P., Mahdi, G. S. S., & Acuña, L. A. (2021). Triangular neutrosophic cognitive map for multistage sequential decision-making problems. *International Journal of Fuzzy Systems*, 23, 657-679. <https://doi.org/10.1007/s40815-020-01014-5>.
27. Wajid, M. S., Terashima-Marin, H., Paul Rad, P. N., & Wajid, M. A. (2022). Violence detection approach based on cloud data and Neutrosophic cognitive maps. *Journal of Cloud Computing*, 11(1), 1-18. <https://doi.org/10.1186/s13677-022-00369-4>.
28. Chithra, B., & Nedunchezian, R. (2022). Dynamic neutrosophic cognitive map with improved cuckoo search algorithm (DNCM-ICSA) and ensemble classifier for rheumatoid arthritis (RA) disease. *Journal of King Saud University-Computer and Information Sciences*, 34(6), 3236-3246. <https://doi.org/10.1016/j.jksuci.2020.06.011>.
29. Bhutani, K., Gaur, S., Panwar, P., & Garg, S. (2023, February). A Neutrosophic Cognitive Maps Approach for Pestle Analysis in Food Industry. In *International Conference On Emerging Trends In Expert*

- Applications & Security (pp. 349-360). Singapore: Springer Nature Singapore. [https://doi.org/10.1007/978-981-99-1909-3\\_30](https://doi.org/10.1007/978-981-99-1909-3_30).
30. Karadayi-Usta, S. (2022). The Role of the Paper Packaging Industry in the Circular Economy: The Causal Relationship Analysis via Neutrosophic Cognitive Maps. In Handbook of Research on Advances and Applications of Fuzzy Sets and Logic (pp. 605-618). IGI Global. <https://doi.org/10.4018/978-1-7998-7979-4.ch027>.
  31. Rojas, H. E., Nelson, F. S., & Cabrita Marina, M. (2022). Neutrosophic Cognitive Maps for Violence Cause Analysis. International Journal of Neutrosophic Science (IJNS), 19(1).
  32. Rodríguez, J. O. C., Veloz, Á. P. M., & Andi, S. T. T. (2022). Neutrosophic Cognitive Maps for the Analysis of the Factors in the proper Diagnosis of Conversion Disorder. Neutrosophic Sets and Systems, 52(1), 36.
  33. Raúl, D. C., Fabricio, T. T., & Elizabeth, D. P. F. (2022). Analyzing the impact of social factors on homelessness with neutrosophic cognitive maps. International Journal of Neutrosophic Science (IJNS), 19(1).
  34. Priya, R., & Martin, N. (2022). Neutrosophic Sociogram Approach to Neutrosophic Cognitive Maps in Swift Language. Neutrosophic Sets and Systems, 49, 111-144.
  35. Kumaravel, S. K., Murugesan, R., Nagaram, N. B., Rasappan, S., & Yamini, G. (2023). Applications of fuzzy cognitive maps and neutrosophic cognitive maps on analysis of dengue fever. Fuzzy Logic Applications in Computer Science and Mathematics, 249-265. <https://doi.org/10.1002/9781394175130.ch16>.

**Received:** 19 Sep 2023, **Revised:** 22 Oct 2023,

**Accepted:** 25 Nov 2023, **Available online:** 08 Dec 2023.



© 2024 by the authors. Submitted for possible open access publication under the terms and conditions of the Creative Commons Attribution (CC BY) license (<http://creativecommons.org/licenses/by/4.0/>).