Utilizing Python for Neutrosophic Theory: A Study of Neutrosophic Crisp Sets and Topological Spaces

Kokilavani V[1], Tharani K[2*]
[1]Associate Professor and Head, Department of Mathematics, Kongunadu Arts and Science College, G.N.Mills, Coimbatore-641 029, Tamil Nadu,India, Email: vanikasc@yahoo.co.in
[2*]PG and Research Department of Mathematics, Kongunadu Arts and Science College, G.N.Mills, Coimbatore-641029, TamilNadu, India, Email: tharanitopo23@gmail.com
*Corresponding Author

Received: 17.07.2024 Revised: 18.08.2024 Accepted: 20.09.2024

ABSTRACT
Python programming is a fundamental component of modern research that greatly reduces the workload for human resources. Researchers all throughout the world find that encoding their data into code gives them rapid computational power. Python's efficiency and adaptability are vital for enhancing research activities in mathematics, especially in areas like topology. In a world characterized by indeterminacy, traditional crisp sets, with their rigid boundaries of truth and falsehood, fail to accurately reflect reality. Consequently, neutrosophic theory has emerged in contemporary research as an alternative way to represent the real world. In this paper, finding the Neutrosophic Crisp Set(NCS), Neutrosophic Crisp Topological space(NCTS) utilizing Python programming.

Keywords: NCS, NCTS, NCOS, NCCS

1. INTRODUCTION

Several mathematical techniques have been created for illustrating and resolving issues in our day-to-day life. The idea of the fuzzy set(FS), first presented by Zadeh (1965)[1], has drawn a lot of attention for problems including imprecision, ambiguity, and uncertainty because of its capacity to approximate human utilize vocabulary to aid in perception and thought. Later, a number of hypotheses were presented in various forms with the goal of resolving the impreciseness issue. Recently, the neutrosophic theory was put out as a better solution since intuitionistic fuzzy sets(IFS's) and FS's were unable to handle information indeterminacy, and FS's and fuzzy logic couldn't convey false membership information. This is because the information is uncertain and imprecise, usually including against and neutral information.

A recent development in philosophy is the field of neutrosophic, which examines the nature, origin, and extent of neutralizes as well as their relationships between various additional spectra. Finding an idea's three sides: truth, falsity and indeterminacy and combining or reversing them is the first step in neutralizing it. Neutrosophic set(NS) is a generalization of FS and IFS. IFS was established by Atanassov [2] in 1983. Coker[3] introduced the notion of intuitionistic fuzzy topological space(IFTS). Florentin Smarandache et.al.,[4][5][6] presented the NS. In that, they introduced the neutrosophic components T, I, and F which represent the membership, indeterminacy and non-membership values respectively, where $]{-0}, 1+[$ is the non- standard unit interval. The notion of neutrosophic topological spaces was first proposed by A. A Salama and S. A. Alblowi[7]. A. A. Salama et. al.,[8] propose the idea of NCTS. M. Vivek Prabu and M. Rahini[9]using the Python program to design the efficiently handle topologies with varying set sizes. In this article, we have constructed a Python program to examine the properties of Neutrosophic crisp sets in Neutrosophic crisp topological spaces. By reducing the manpower required for complex and repetitive calculations and, more importantly, obtaining the results of complex problems almost instantly, our Python program enhances the efficiency and accuracy of examining Neutrosophic crisp sets in Neutrosophic crisp topological spaces.

2. Preliminaries

Definition 2.1. [10] The for loop in Python is iterator-based. It goes through the elements in any ordered sequence list, i.e., string, lists, tuples, the keys of dictionary. A value is assigned to a loop variable at each iteration step.

```
Syntax:
    for x in y:
        Block 1
    else:                   #optional
        Block 2             #executed only when the loop exits
```

Fig.1 The for loop Syntax

Definition 2.2. [10] An if elif else statement is a control flow structure in programming that allows you to execute different code blocks based on multiple conditions.
Structure:
if: Checks the first condition.
elif: Checks additional conditions if the previous ones are False.
else: Executes if none of the previous conditions are True.

```
Syntax:
    if expression1:
        statment1
    elif expression2:
        statement2
    elif expression3:
        statement3
    else expression4:
        statement4
```

Fig.2 if elif else Statement Syntax

Definition 2.3. [10] The while statement is used when you have a piece of code and you want to repeat if 'n' number of times or forever. With while loop, we have to give a conditional statement that tells the interpreter when the loop will halt.

```
Syntax:
    while condition:
        block 1
    else:               #optional
        statement
```

Fig.3 While Statement Syntax

Definition 2.4. [10] The break statement exits from the loop and transfers the execution from the loop to the statement that is immediately following the loop.

Definition 2.5 [10] A Neutrosophic Crisp Set (NCS) is a mathematical concept it allows for elements to have degrees of truth, indeterminacy and falsity. In theoretical terms, a Neutrosophic Crisp Set is defined as follows:
A NCS A can be represented as $\langle A_1, A_2, A_3 \rangle$ where A1, A2, and A3 are the subsets of the universal set X.

(i) Type 1 NCS: A_1 is disjoint with both A_2 and A_3. Furthermore A_2, is disjoint with A_3.

(ii) Type 2 NCS: This type is an extension of Type 1 with an additionally requires that theunion of A_1, A_2, and A_3 equals the universal set X.

(iii) Type 3 NCS: The subsets A_1, A_2, and A_3 are all disjoint and their union equals the universal set X.

Definition 2.6 [10] Let X represent a non empty set and the NCS's A and B be expressed as $A = \langle A_1, A_2, A_3 \rangle, B = \langle B_1, B_2, B_3 \rangle$, then subset A of B implies either one of the following forms,
(i) A_1 is subset of B_1, A_2 is subset of B_2 & B_3 is subset of A_3.
(ii) A_1 is subset of B_1, B_2 is subset of A_2 & B_3 is subset of A_3.

Definition 2.7 [10] Let X represent a non empty set and the NCS's A and B be expressed as $A = \langle A_1, A_2, A_3 \rangle, B = \langle B_1, B_2, B_3 \rangle$, then the intersection of A and B can be obtained by any one of the following forms,
(i) A_1 intersect B_1, A_2 intersect B_2 & A_3 union B_3.
(ii) A_1 intersect B_1, A_2 union B_2 & A_3 union B_3.

Definition 2.8 [10] Let X represent a non empty set and the NCS's A and B be expressed as $A = \langle A_1, A_2, A_3 \rangle$, $B = \langle B_1, B_2, B_3 \rangle$, then the union of A and B can be obtained by any one of the following forms,
(i) A_1 union B_1, A_2 union B_2 & A_3 intersect B_3.
(ii) A_1 union B_1, A_2 intersect B_2 & A_3 intersect B_3.

Definition 2.9 [10] Let $A = \langle A_1, A_2, A_3 \rangle$ a NCS on X, then the complement of A (in short, A^c)may be characterized in three distinct ways:
(i) $A^c = \langle A_1^c, A_2^c, A_3^c \rangle$
(ii) $A^c = \langle A_3, A_2, A_1 \rangle$
(iii) $A^c = \langle A_3, A_2^c, A_1 \rangle$

Definition 2.10 [10] A NCTS on a non empty set X is a family (Γ) of neutrosophic crisp subsets ∈ X which satisfy below:
(i) $\phi_N, X_N \in \Gamma$, where ϕ_N can be in any one of the appropriates forms $\langle \phi, \phi, X \rangle$ or $\langle \phi, X, X \rangle$ or $\langle \phi, X, \phi \rangle$ or $\langle \phi, \phi, \phi \rangle$ and X_N can be in any one of the appropriates forms $\langle X, \phi, \phi \rangle$ or $\langle X, X, \phi \rangle$ or $\langle X, \phi, X \rangle$ or $\langle X, X, X \rangle$
(ii) $A_1 \cap A_2 \in \Gamma$ for any A_1, $A_2 \in \Gamma$.
(iii) $\cup A_j \in \Gamma$ for any arbitrary family $\{A_j : j \in J\} \subseteq (X, \Gamma)$ is known as NCTS and the objects in Γ is namely neutrosophic crisp open set (NCOS) and complement of NCOS is neutrosophic crisp closed set (NCCS).

3. Computation of Neutrosophic Crisp Topological Space

Algorithm 1 Process of finding subsets and combination of subsets

---

DISPLAY: "Enter the elements of the set (e.g., {a,b}):"
READ: user_input
SET: user_set to parse_input(user_input)
SET: input_sets to set(user_set)
SET: result to powerset(user_set)
DISPLAY: "Subsets of the given **SET:**"
for subset in result **do**
    DISPLAY: subset or '[ ]' if subset is empty
end for
SET: set size to **3**
SET: sets to generate_sets(result, set_size)
DISPLAY: "Combinations of subsets:"
for s in sets **do**
    DISPLAY: s
end for

Algorithm 2 Process for Computing Neutrosophic Crisp Set

```
DEFINE: value as an empty list
for i from 0 to LENGTH of sets -1 do
    if N1 = = EMPTY SET AND N2 = = EMPTY SET AND N3 = = EMPTY
    SET then
        DISPLAY: "type 1 satisfied sets", sets[i]
        APPEND: sets[i] to value
    end if
    if N1 = = EMPTY SET AND N2 = = EMPTY SET AND N3 = = EMPTY
    SET AND N4 = = input sets then
        DISPLAY: "type 2 satisfied sets", sets[i]
        APPEND: sets[i] to value
    end if
    if N5 = = EMPTY SET AND N4 = = input sets then
        DISPLAY: "type 3 satisfied sets", sets[i]
        APPEND: sets[i] to value
    end if
end for
```

Algorithm 3 Process for Choosing the Empty and Whole neutrosophic crisp set

```
DEFINE: X_N as (user_set, user_set, EMPTY LIST)
DEFINE: ϕ_N as (EMPTY LIST, EMPTY LIST, user_set)
DEFINE: X_N1 as (user_set, EMPTY LIST, EMPTY LIST)
DEFINE: X_N2 as (user_set, EMPTY LIST, user_set)
DEFINE: ϕ_N1 as (EMPTY LIST, user_set, user_set)
DEFINE: ϕ_N2 as (EMPTY LIST, user_set, EMPTY LIST)
DISPLAY: "Choose one whole and one empty neutrosophic crisp set:"
DISPLAY: "Whole Neutrosophic Crisp Sets:"
DISPLAY: "1. X_N:", X_N
DISPLAY: "2. X_N1:", X_N1
DISPLAY: "3. X_N2:", X_N2
READ: whole_choice
SET: whole_crisp_set to corresponding set based on whole_choice
DISPLAY: "Empty Neutrosophic Crisp Sets:"
DISPLAY: "1. ϕ_N : ", ϕ_N
DISPLAY: "2. ϕ_N1 : ", ϕ_N1
DISPLAY: "3. ϕ_N2 : ", ϕ_N2
READ: empty_choice
SET: empty_crisp_set to corresponding set based on empty_choice
REMOVE: unchosen sets from value list
```

Algorithm 4 Process for Computing the Tau(First and Second Condition)

```
DISPLAY: "For example: [([], [], ['a']),([], [], ['b'])]"
while TRUE do
    DISPLAY: "Enter the Tau (whole and empty neutrosophic crisp sets should
    not be included):"
    READ: tau_input
    SET: tau_set to EVALUATE tau_input
    if tau_set contains duplicates then
        DISPLAY: "Invalid Tau. Duplicate subsets are not allowed."
    else if tau_set contains whole crisp set or empty crisp set then
        DISPLAY: "Invalid Tau. The whole or empty neutrosophic crisp set
```

```
        should not be included."
    else if any subset in tau_set is not in value then
        DISPLAY: "Invalid Tau. Please make sure all subsets are valid and from
        the value list."
    else
        SET: tau_list to [empty crisp set, whole crisp set] + tau set
        DISPLAY: "Tau list with chosen whole and empty neutrosophic crisp
        sets:"
        for tau in tau list do
            DISPLAY: tau
        end for
        DISPLAY: "First condition is satisfied"
        BREAK
    end if
end while
DISPLAY: "Choose the type of union:"
DISPLAY: "1. Union Type 1: [A1 ∪ B1, A2 ∪ B2, A3 ∩ B3]"
DISPLAY: "2. Union Type 2: [A1 ∪ B1, A2 ∩ B2, A3 ∩ B3]"
READ: union_type
SET: union_result to True
for i from 0 to LENGTH of Tau - 1 do
    for j from 0 to LENGTH of Tau - 1 do
        if union of subsets(Tau[i], Tau[j], union type) NOT IN Tau then
            SET: union result to False
        BREAK
        end if
    end for
    if union result is False then
        BREAK
    end if
end for
```

Algorithm 5 Algorithm 5 Process for Computing the Tau(Third Condition and Complement)

```
if union_result then
    DISPLAY: "Second condition is satisfied"
    DISPLAY: "Choose the type of intersection:"
    DISPLAY: "1. Intersection Type 1: [A1 ∩ B1, A2 ∩ B2, A3 ∪ B3]"
    DISPLAY: "2. Intersection Type 2: [A1 ∩ B1, A2 ∪ B2, A3 ∪ B3]"
    READ: intersection_type
    SET: intersection_result to True
    for i from 0 to LENGTH of Tau - 1 do
        for j from 0 to LENGTH of Tau - 1 do
            if intersection_of_subsets(Tau[i], Tau[j], intersection_type) NOT IN Tau then
                SET: intersection result to False
                BREAK
            end if
        end for
        if intersection result is False then
            BREAK
        end if
    end for
    if intersection_result then
        DISPLAY: "Third condition is satisfied"
        DISPLAY: "Choose the type of complement:"
        DISPLAY: "1. Complement Type 1: ([complement of A1],
                                  [complement of A2], [complement of A3])"
        DISPLAY: "2. Complement Type 2: ([A3], [A2], [A1])"
```

DISPLAY: "3. Complement Type 3: ([A3], [complement of A2], [A1])"
READ: complement_type
SET: Tau_complement to EMPTY LIST
for subset in Tau **do**
    ADD: find_complement(subset, complement_type) to Tau_complement
end for
DISPLAY: "Complement of Tau:"
DISPLAY: Tau complement
else
    DISPLAY: "Third condition is Not Satisfied"
end if
else
    DISPLAY: "Second condition is Not Satisfied"
end if

Coding:

```
def generate_sets(elements, set_size):
    return tuple(product(elements, repeat=set_size))

# Get user input for the set
user_input = input("Enter the elements of the set (e.g., {a,b}): ")
# Parse the input string to get the set elements
user_set = parse_input(user_input)
input_sets = set(user_set)

# Generate all subsets of the user-defined set
result = powerset(user_set)

# Print all subsets of the given set
print("Subsets of the given set:")
for subset in result:
    print(subset if subset else '[]')

# Generate combinations of subsets
set_size = 3
sets = tuple(generate_sets(result, set_size))

# Print the generated sets
print("Combinations of subsets:")
for s in sets:
    print(tuple(s))
```

Fig. 4 Coding for Finding the Subsets and Combinations of the Subset

In Figure 4, We defined the set as X={a,b} and obtain the subsets of X. Then the combination of ['a'],['b'],['a','b'],[ ] expressed in the form of triple subset.

```
    # Type1 if A1 ∩ A2 = ϕ, A1 ∩ A3 = ϕ and A2 ∩ A3 = ϕ
    if N1 == set() and N2 == set() and N3 == set():
        type1 = sets[i]
        print('type 1 satisfied sets', type1)
        if type1 not in value:
            value.append(type1)
    # Type2 if A1 ∩ A2 = ϕ, A1 ∩ A3 = ϕ and A2 ∩ A3 = ϕ and A1 ∪ A2 ∪ A3 = X_N
    if N1 == set() and N2 == set() and N3 == set() and N4 == input_sets:
        type2 = sets[i]
        print('type 2 satisfied sets', type2)
        if type2 not in value:
            value.append(type2)
    # Type3 if A1 ∩ A2 ∩ A3 = ϕ and A1 ∪ A2 ∪ A3 = X_N
    if N5 == set() and N4 == input_sets:
        type3 = sets[i]
        print('type 3 satisfied sets', type3)
        if type3 not in value:
            value.append(type3)
```

Fig. 5 Coding for Computing Neutrosophic Crisp Set

In Figure 5, A NCS of Type1 if $A_1 \cap A_2 = \phi$, $A_1 \cap A_3 = \phi$ and $A_2 \cap A_3 = \phi$, NCS of Type2 if $A_1 \cap A_2 = \phi$, $A_1 \cap A_3 = \phi$ and $A_2 \cap A_3 = \phi$ and $A_1 \cup A_2 \cup A_3 = X$ and NCS of Type3 if $A_1 \cap A_2 \cap A_3 = \phi$ and $A_1 \cup A_2 \cup A_3 = X$

are defined.

```
# Choose one whole and one empty crisp set
print("Choose one whole and one empty crisp set:")
print("Whole Crisp Sets:")
print("1. X_N: ", X_N)          #X_N = (user_set, user_set, []) [X_N = ⟨X, X, ϕ⟩]
print("2. X_N1: ", X_N1)        #X_N1 = (user_set,[],[]) [X_N1 = ⟨X, ϕ, ϕ⟩]
print("3. X_N2: ", X_N2)        #X_N2 = (user_set,[],user_set) [X_N2 = ⟨X, ϕ, X⟩]
whole_choice = int(input("Enter the index of the whole crisp set: "))
whole_crisp_set = [X_N, X_N1, X_N2][whole_choice - 1]

print("\nEmpty Crisp Sets:")
print("1. ϕ_N: ", ϕ_N)       # ϕ_N = ([], [], user_set) [ϕ_N = ⟨ϕ, ϕ, X⟩]
print("2. ϕ_N1: ", ϕ_N1)     # ϕ_N1 = ([], user_set, user_set) [ϕ_N1 = ⟨ϕ, X, X⟩]
print("3. ϕ_N2: ", ϕ_N2)     # ϕ_N2 = ([], user_set, []) [ϕ_N2 = ⟨ϕ, X, ϕ⟩]
empty_choice = int(input("Enter the index of the empty crisp set: "))
empty_crisp_set = [ϕ_N, ϕ_N1, ϕ_N2][empty_choice - 1]

unchosen_sets = [set for set in [X_N, X_N1, X_N2,ϕ_N, ϕ_N1,ϕ_N2]
                 if set != whole_crisp_set and set != empty_crisp_set]
for unchosen_set in unchosen_sets:
    value.remove(unchosen_set)
# print("original crispset", len(value))
v=[]
for val in value:
    x +=1
    print(x,':',val)
    v.append(val)
```

Fig. 6 Coding for Choosing the Empty and Whole Neutrosophic crisp set

In Figure 6, The user is prompted to choose one whole and one empty neutrosophic crisp set from the provided options. The user's choices are recorded, and the selected sets are identified. A list of all possible sets is then compared to the chosen sets to determine which ones were not selected. These not chosen sets are removed from the value list. After updating the value list, a counter is initialized, and the function iterates through the remaining sets

```
# Prompt the user to enter Tau
print("For example: [(['a'], ['b'], [])]")
while True:
    tau_input = input("Enter the Tau(whole and empty neutrosophic crisp sets should not be included):")
    tau_set = eval(tau_input)  # Safely evaluate the input string to a tuple
    Tau=[]
    # Check for repeated subsets
    if len(set(map(lambda x: tuple(map(tuple, x)), tau_set))) < len(tau_set):
        print("Invalid Tau . Duplicate subsets are not allowed.")
    elif any(subset in tau_set for subset in [whole_crisp_set, empty_crisp_set]):
        print("Invalid Tau . The whole or empty neutrosphic crisp set should not be included.")
    elif not all(subset in value for subset in tau_set):
        print("Invalid Tau . Please make sure all subsets are valid and from the value list.")
    else:
        # Insert whole and empty neutrosophic crisp sets at the beginning of the Tau set
        tau_list = [empty_crisp_set, whole_crisp_set] + list(tau_set)
        Tau.extend(tau_list)
        print("Tau list with chosen whole and empty neutrosophic crisp sets:")
        for tau in tau_list:
            print(tau)
        print("First Axiom is Satisfied")
        print(tau_list)
        break  # Exit the loop once a valid Tau is entered
```

.**Fig. 7** Coding for User to enter Tau

```
print("Choose the type of union:")
print("1. Union Type 1: [A1 ∪ B1, A2 ∪ B2, A3 ∩ B3]")
print("2. Union Type 2: [A1 ∪ B1, A2 ∩ B2, A3 ∩ B3]")
union_type = int(input("Enter the index of the union type: "))
# Compute the union of all subsets in Tau based on the chosen union type
union_result = all(union_of_subsets(Tau[i], Tau[j], union_type) in Tau
                   for i in range(len(Tau)) for j in range(len(Tau)))
# Check if the union of all subsets is present in Tau
if union_result:
    print("Second Condition is Satisfied")
    # Proceed to intersection condition
    # Prompt the user to choose the intersection type
    # Validate type choices
    print("\nChoose the type of intersection:")
    print("1. Intersection Type 1: [A1 ∩ B1, A2 ∩ B2, A3 ∪ B3]")
    print("2. Intersection Type 2: [A1 ∩ B1, A2 ∪ B2, A3 ∪ B3]")
    intersection_type = int(input("Enter the index of the intersection type: "))
    # Compute the intersection of all subsets in Tau based on the chosen intersection type
    intersection_result = all(intersection_of_subsets(Tau[i], Tau[j], intersection_type) in Tau
        for i in range(len(Tau)) for j in range(len(Tau)))

    # Check if the intersection of all subsets is present in Tau
    if intersection_result:
        print("Third Condition is Satisfied")
        #Proceed to complement of Tau
        Tau_complement=[]
        # Prompt the user to choose the complement type
        print("Choose the type of complement:")
        print("1. Complement Type 1: ([complement of A1], [complement of A2], [complement of A3])")
        print("2. Complement Type 2: ([A3], [A2], [A1])")
        print("3. Complement Type 3: ([A3], [complement of A2], [A1])")
        complement_type = int(input("Enter the index of the complement type: "))
        # Compute the complement of Tau based on the chosen type
        tau_complement = [find_complement(subset, complement_type) for subset in Tau]
        Tau_complement.extend(tau_complement)
        # Print the complement of Tau
        print("Complement of Tau:")
        print(Tau_complement)
    else:
        print("Third Condition is Not Satisfied")
else:
    print("Second Condition is Not Satisfied")
```

Fig. 8 Coding for Computing the Tau

In Figure 7 and 8, The program proceeds to compute the union of all subsets in the list Tau based on the selected union type. If it is satisfied, the program moves to the intersection condition. Next, the program computes the intersection of all subsets in Tau based on the selected intersection type. If it is satisfied, the program proceeds to compute the complement of Tau.

Working Process

The process begin with Users by inputting elements for a set. The power set is generated, showing all possible subsets, which are then displayed. Combinations of three subsets are created. Each combination is evaluated to check if it meets specific type conditions (Type 1, Type 2, Type 3), and satisfied sets are identified and printed. Users then select a whole and an empty neutrosophic crisp set from predefined options. Next, they input a Tau, ensuring it doesn't include the chosen whole and empty neutrosophic crisp sets and has no duplicates. The Tau list is formed, and the first axiom is checked and satisfied. Users then select union and intersection types, and the conditions are checked and satisfied accordingly. Lastly, users select a complement type, and the complement of Tau is calculated and displayed.

Output

```
Enter the elements of the set (e.g., {a,b}): {a,b}
Subsets of the given set:
['a', 'b']
['a']
['b']
[]
Combinations of subsets:
(['a', 'b'], ['a', 'b'], ['a', 'b'])
(['a', 'b'], ['a', 'b'], ['a'])
(['a', 'b'], ['a', 'b'], ['b'])
(['a', 'b'], ['a', 'b'], [])
(['a', 'b'], ['a'], ['a', 'b'])
(['a', 'b'], ['a'], ['a'])
(['a', 'b'], ['a'], ['b'])
(['a', 'b'], ['a'], [])
(['a', 'b'], ['b'], ['a', 'b'])
(['a', 'b'], ['b'], ['a'])
(['a', 'b'], ['b'], ['b'])
(['a', 'b'], ['b'], [])
(['a', 'b'], [], ['a', 'b'])
(['a', 'b'], [], ['a'])
(['a', 'b'], [], ['b'])
(['a', 'b'], [], [])
(['a'], ['a', 'b'], ['a', 'b'])
(['a'], ['a', 'b'], ['a'])
(['a'], ['a', 'b'], ['b'])
(['a'], ['a', 'b'], [])
(['a'], ['a'], ['a', 'b'])
(['a'], ['a'], ['a'])
(['a'], ['a'], ['b'])
(['a'], ['a'], [])
(['a'], ['b'], ['a', 'b'])
(['a'], ['b'], ['a'])
(['a'], ['b'], ['b'])
(['a'], ['b'], [])
(['a'], [], ['a', 'b'])
(['a'], [], ['a'])
(['a'], [], ['b'])
(['a'], [], [])
(['b'], ['a', 'b'], ['a', 'b'])
(['b'], ['a', 'b'], ['a'])
(['b'], ['a', 'b'], ['b'])
(['b'], ['a', 'b'], [])
(['b'], ['a'], ['a', 'b'])
(['b'], ['a'], ['a'])
(['b'], ['a'], ['b'])
(['b'], ['a'], [])
(['b'], ['b'], ['a', 'b'])
(['b'], ['b'], ['a'])
(['b'], ['b'], ['b'])
(['b'], ['b'], [])
(['b'], [], ['a', 'b'])
(['b'], [], ['a'])
(['b'], [], ['b'])
(['b'], [], [])
([], ['a', 'b'], ['a', 'b'])
([], ['a', 'b'], ['a'])
([], ['a', 'b'], ['b'])
([], ['a', 'b'], [])
([], ['a'], ['a', 'b'])
([], ['a'], ['a'])
([], ['a'], ['b'])
([], ['a'], [])
([], ['b'], ['a', 'b'])
([], ['b'], ['a'])
([], ['b'], ['b'])
([], ['b'], [])
([], [], ['a', 'b'])
([], [], ['a'])
([], [], ['b'])
([], [], [])
```

Fig. 9 Output of Subsets and Combinations of the Subset

```
type 3 satisfied sets (['a', 'b'], ['a', 'b'], [])
type 3 satisfied sets (['a', 'b'], ['a'], ['b'])
type 3 satisfied sets (['a', 'b'], ['a'], [])
type 3 satisfied sets (['a', 'b'], ['b'], ['a'])
type 3 satisfied sets (['a', 'b'], ['b'], [])
type 3 satisfied sets (['a', 'b'], [], ['a', 'b'])
type 3 satisfied sets (['a', 'b'], [], ['a'])
type 3 satisfied sets (['a', 'b'], [], ['b'])
type 1 satisfied sets (['a', 'b'], [], [])
type 2 satisfied sets (['a', 'b'], [], [])
type 3 satisfied sets (['a', 'b'], [], [])
type 3 satisfied sets (['a'], ['a', 'b'], ['b'])
type 3 satisfied sets (['a'], ['a', 'b'], [])
type 3 satisfied sets (['a'], ['a'], ['b'])
type 3 satisfied sets (['a'], ['b'], ['a', 'b'])
type 3 satisfied sets (['a'], ['b'], ['a'])
type 3 satisfied sets (['a'], ['b'], ['b'])
type 1 satisfied sets (['a'], ['b'], [])
type 2 satisfied sets (['a'], ['b'], [])
type 3 satisfied sets (['a'], ['b'], [])
type 3 satisfied sets (['a'], [], ['a', 'b'])
type 1 satisfied sets (['a'], [], ['b'])
type 2 satisfied sets (['a'], [], ['b'])
type 3 satisfied sets (['a'], [], ['b'])
type 1 satisfied sets (['a'], [], [])
type 3 satisfied sets (['b'], ['a', 'b'], ['a'])
type 3 satisfied sets (['b'], ['a', 'b'], [])
type 3 satisfied sets (['b'], ['a'], ['a', 'b'])
type 3 satisfied sets (['b'], ['a'], ['a'])
type 3 satisfied sets (['b'], ['a'], ['b'])
type 1 satisfied sets (['b'], ['a'], [])
type 2 satisfied sets (['b'], ['a'], [])
type 3 satisfied sets (['b'], ['a'], [])
type 3 satisfied sets (['b'], ['b'], ['a'])
type 3 satisfied sets (['b'], [], ['a', 'b'])
type 1 satisfied sets (['b'], [], ['a'])
type 2 satisfied sets (['b'], [], ['a'])
type 3 satisfied sets (['b'], [], ['a'])
type 1 satisfied sets (['b'], [], [])
type 3 satisfied sets ([], ['a', 'b'], ['a', 'b'])
type 3 satisfied sets ([], ['a', 'b'], ['a'])
type 3 satisfied sets ([], ['a', 'b'], ['b'])
type 1 satisfied sets ([], ['a', 'b'], [])
type 2 satisfied sets ([], ['a', 'b'], [])
type 3 satisfied sets ([], ['a', 'b'], [])
type 3 satisfied sets ([], ['a'], ['a', 'b'])
type 1 satisfied sets ([], ['a'], ['b'])
type 2 satisfied sets ([], ['a'], ['b'])
type 3 satisfied sets ([], ['a'], ['b'])
type 1 satisfied sets ([], ['a'], [])
type 3 satisfied sets ([], ['b'], ['a', 'b'])
type 1 satisfied sets ([], ['b'], ['a'])
type 2 satisfied sets ([], ['b'], ['a'])
type 3 satisfied sets ([], ['b'], ['a'])
type 1 satisfied sets ([], ['b'], [])
type 1 satisfied sets ([], [], ['a', 'b'])
type 2 satisfied sets ([], [], ['a', 'b'])
type 3 satisfied sets ([], [], ['a', 'b'])
type 1 satisfied sets ([], [], ['a'])
type 1 satisfied sets ([], [], ['b'])
```

Fig. 10 Output of Satisfied the Type1, Type2 and Type3 of Neutrosophic Crisp Set

```
Choose one whole and one empty crisp set:
Whole Crisp Sets:
1. X_N:  (['a', 'b'], ['a', 'b'], [])
2. X_N1:  (['a', 'b'], [], [])
3. X_N2:  (['a', 'b'], [], ['a', 'b'])
Enter the index of the whole crisp set: 1

Empty Crisp Sets:
1. ϕ_N:  ([], [], ['a', 'b'])
2. ϕ_N1:  ([], ['a', 'b'], ['a', 'b'])
3. ϕ_N2:  ([], ['a', 'b'], [])
Enter the index of the empty crisp set: 1
1 : (['a', 'b'], ['a', 'b'], [])
2 : (['a', 'b'], ['a'], ['b'])
3 : (['a', 'b'], ['a'], [])
4 : (['a', 'b'], ['b'], ['a'])
5 : (['a', 'b'], ['b'], [])
6 : (['a', 'b'], [], ['a'])
7 : (['a', 'b'], [], ['b'])
8 : (['a'], ['a', 'b'], ['b'])
9 : (['a'], ['a', 'b'], [])
10 : (['a'], ['a'], ['b'])
11 : (['a'], ['b'], ['a', 'b'])
12 : (['a'], ['b'], ['a'])
13 : (['a'], ['b'], ['b'])
14 : (['a'], ['b'], [])
15 : (['a'], [], ['a', 'b'])
16 : (['a'], [], ['b'])
17 : (['a'], [], [])
18 : (['b'], ['a', 'b'], ['a'])
19 : (['b'], ['a', 'b'], [])
20 : (['b'], ['a'], ['a', 'b'])
21 : (['b'], ['a'], ['a'])
22 : (['b'], ['a'], ['b'])
23 : (['b'], ['a'], [])
24 : (['b'], ['b'], ['a'])
25 : (['b'], [], ['a', 'b'])
26 : (['b'], [], ['a'])
27 : (['b'], [], [])
28 : ([], ['a', 'b'], ['a'])
29 : ([], ['a', 'b'], ['b'])
30 : ([], ['a'], ['a', 'b'])
31 : ([], ['a'], ['b'])
32 : ([], ['a'], [])
33 : ([], ['b'], ['a', 'b'])
34 : ([], ['b'], ['a'])
35 : ([], ['b'], [])
36 : ([], [], ['a', 'b'])
37 : ([], [], ['a'])
38 : ([], [], ['b'])
```

Fig. 11 Output of Finalized the Neutrosophic Crisp Sets

```
For example: [([], [], ['a']),([], [], ['b'])]
Enter the Tau set (whole and empty sets should not be included): [(['a'],['b'],[])]
Tau list with chosen whole and empty crisp sets:
([], [], ['a', 'b'])
(['a', 'b'], ['a', 'b'], [])
(['a'], ['b'], [])
First Axiom is Satisfied
[([], [], ['a', 'b']), (['a', 'b'], ['a', 'b'], []), (['a'], ['b'], [])]
Choose the type of union:
1. Union Type 1: [A1 ∪ B1, A2 ∪ B2, A3 ∩ B3]
2. Union Type 2: [A1 ∪ B1, A2 ∩ B2, A3 ∩ B3]
Enter the index of the union type: 1
Union Condition is Satisfied

Choose the type of intersection:
1. Intersection Type 1: [A1 ∩ B1, A2 ∩ B2, A3 ∪ B3]
2. Intersection Type 2: [A1 ∩ B1, A2 ∪ B2, A3 ∪ B3]
Enter the index of the intersection type: 1
Intersection Condition is Satisfied
Choose the type of complement:
1. Complement Type 1: ([complement of A1], [complement of A2], [complement of A3])
2. Complement Type 2: ([A3], [A2], [A1])
3. Complement Type 3: ([A3], [complement of A2], [A1])
Enter the index of the complement type: 1
Complement of Tau:
[(['a', 'b'], ['a', 'b'], []), ([], [], ['a', 'b']), (['b'], ['a'], ['a', 'b'])]
```

Fig. 12 Output of Computing the Tau (Satisfying the three axioms)

CONCLUSION

In this article, the development of Python program marks a significant advancement in the study of Neutrosophic crisp sets within Neutrosophic crisp topological spaces. Our program not only minimizes the manpower required but also delivers results with remark able speed and accuracy. This tool enhances the efficiency of research and practical applications, the way for more effective analysis and utilization of Neutrosophic crisp sets in various scientific and mathematical fields.

Data Availability

I have not used any external data source forth is manuscript.

Conflict of interest

The authors declare that they have no conflict of interest.

REFERENCES

[1] Zadeh, L.: Fuzzy sets inform and control. Applied Science Periodical 8(4), 338–353 (2006)

[2] Atanassov,K.T.:Intuitionisticfuzzysets.FuzzySetsandSystems20(1),87–96 (1986) https://doi.org/10.1016/S0165-0114(86)80034-3

[3] C¸oker,D.: An introduction to intuitionistic fuzzy topological spaces. Fuzzy Sets and Systems 88(1), 81–89(1997)https://doi.org/10.1016/S0165-0114(96)

[4] 00076-0

[5] Smarandache,F.: First international conference on neutrosophy, neutrosophic logic, set, probability and statistics. Florentin Smarandache 4 (2001)

[6] Smarandache, F.: A unifying field in logics: Neutrosophic logic, neutrosophy, neutrosophic set, neutrosophic probability. In: Philosophy, pp. 1–141. American Research Press, Rehoboth [N.M.] (1998)

[7] Al-Omeri,W., Smarandache,F.: New neutrosophic sets via neutrosophic topological spaces. Infinite Study2016, 1–15 (2016)

[8] Salama, A., Alblowi, S.: Neutrosophic set and neutrosophic topological spaces. IOSR Journal of Mathematics 3, 31–35 (2012)

[9] Salama, A.A., Smarandache, F.: Neutrosophic Crisp Set Theory. Educational Publishers, Columbus, OH, USA (2015)

[10] Prabhu, M.V., Rahini, M.: Developing a topology generator using python program. In: AIP Conference Proceedings, vol. 2649 (2023). AIP Publishing

[11] Balagurusamy, E.: Introduction to computing & problem solving python. first edition, published by McGraw Hill Education (India) Private Ltd (2016)

[12] Salama, A., Smarandache, F., Kroumov, V.: Neutrosophic crisp sets& neutrosophic crisp topological spaces. Infinite Study (2014)