Neutrosophic Sets and Systems

Volume 25 Article 11

1-1-2019

Novel Open Source Python Neutrosophic Package

Haitham A. El-Ghareeb

University of New Mexico

Novel Open Source Python Neutrosophic Package

Haitham A. El-Ghareeb[1,*]

[1]Information Systems Department, Faculty of Computers and Information Sciences, Mansoura University, 35116, Egypt

[*]Correspondence: Author (helghareeb@mans.edu.eg)

Abstract: Neutrosophic sets has gained wide popularity and acceptance in both academia and industry. Different fields have successfully adopted and utilized neutrosophic sets. However, there is no open-source implementation that provides the basic neutrosophic concepts. Open-source is a global movement that enables developers to share their source-code, as researchers do share their research ideas and results. Presented Open-source python neutrosophic package is the first of its kind. It utilizes object oriented concepts. It is based on one of the most popular multi-paradigm programming languages that is widely used in different academic and industry fields and activities; namely python. Presented package tends to dissolve the barriers and enable both researchers and developers to adopt neutrosophic sets and theory in research and applications. In this paper, the first Open-source, Object-Oriented, Python based Neutrosophic package is presented. This paper intensively scanned neutrosophic sets research attempting to reach the most widely accepted neutrosophic proofs, and then transforming them into source-code. Presented package implements most of the basic neutrosophic concepts. Presented neutrosophic package presents four different classes: Single Valued Neutrosophic Number, Single Valued Neutrosophic Sets, Interval Valued Neutrosophic Number, and Interval Valued Neutrosophic Sets. Presented package source code, test cases, usage examples, and updated documentation can be found online at `https://www.github.com/helghareeb/neutrosophic`. Presented neutrosophic package can be easily integrated into research and applications. This is an ongoing work and research, as neutrosophic theory is largely expanding, and there are lots of features to cover.

Keywords: Single Valued Neutrosophic Number, Neutrosophic Sets, Open Source, Python

1 Neutrosophic Theory

Neutrosophic sets have been introduced to the literature by Smarandache to handle incomplete, indeterminate, and inconsistent information [18]. In neutrosophic sets, indeterminacy is quantified explicitly through a new parameter I. Truth-membership (T), indeterminacy membership (I) and falsity-membership (F) are three independent parameters that are used to define a Neutrosophic Number.

$$\dot{N} = \left\{ < x; T_{\dot{N}}(x), I_{\dot{N}}(x), F_{\dot{N}}(x) >, x \in X \right\}$$

$$x \in X, \quad T_{\dot{N}}(x), I_{\dot{N}}(x), F_{\dot{N}}(x) \in [0,1] \tag{1.1}$$

This paper presents the first novel open source implementation of a Neutrosophic Package in Python programming language. Proposed implementation aims to facilitate Neutrosophic sets utilization in different Python based applications. Python programming language has been chosen exclusively for different reasons. Python is a high-level programming language, that is efficient, supports high-level data structures, and is highly utilized in different academic and industry disciplines; including big data analytics, artificial intelligence, and machine learning. There are no reasons that prevents porting the proposed Neutrosophic package implementation to other programming languages. Porting presented package is a step to take in the near future. However, Python will be the main focus of this research paper.

Presented characteristics and operations are implemented in the presented Open-Source Python Package can be found at `https://www.github.com/helghareeb/neutrosophic`. Presented Neutrosophic Package development covers the following neutrosophic objects:

- Single Valued Neutrorosphic Number (SVNN)
- Interval Valued Neutrosophic Number (IVNN)
- Single Valued Neutrosophic Sets (SVNS)
- Interval Valued Neutrosophic Sets (IVNS)

Rest of the paper goes as follows:

Section 2 presents a literature review on the most recent areas of applications that utilize neutrosophy theory and neutrosophic sets. Neutrosophic sets has been widely accepted among different disciplines, and the need for an open source neutrosophic package implementation has become a necessity.

Section 3 presents the core design methodology and concepts around the presented novel neutrosophic package. Presented package is object oriented based, that supports open-source concepts, and utilizes some Python magic to enhance the performance and functionality.

In the following sections, an introduction to the neutrosophic theory of the presented section is highlighted, followed by the equation that implements the presented operation. Implementation of the presented equation is presented immediately after the equation, so the reader can follow each section of the code and what it is actually responsible for. The complete source code is available at `https://www.github.com/helghareeb/neutrosophic`. This paper avoids mathematical proofs of the implemented equations, and includes external references that include the proofs of the implemented equations and calculations.

Section 4 presents the basic element and the most widely used neutrosophic number, that is the single valued neutrosophic number (SVNN). SVNNs operations and their implementation are presented in this section.

Section 5 introduces the Single Valued Neutrosophic Sets (SVNS). SVNS consists of multiple SVNNs. Aggregation operations are presented in this section. Implementation details simplified the calculation, and hopefully will act as an enabler for researchers in academia and developers in industry as a guidance and concrete implementation on how to adopt neutrosophic sets in real world applications.

Section 6 highlights one of the most important concepts in neutrosophic theory; that is Interval Valued Neutrosophic Numbers (IVNNs). Though IVNNs are really important in describing real world cases, they are tough to implement because they lack the crisp mathematical characteristics presented in SVNNs. This section presents a simplified way to convert mathematical concepts into concrete implementations.

Section 7 presents the Interval Valued Neutrosophic Sets (IVNSs). IVNS is an importnat neutrosophic concept that must be supported in neutrosophic packages. Averaging function of IVNNs is presented.

Section 8 concludes the paper, highlighting the main features and advantages of the presented open source neutrosophic package, and presents the future work. This is an ongoing work that needs continous development, and will grow exponentially as neutrosophic theory, sets, and systems keeps in growing and gaining wide popularity and acceptance. Paper ends with references.

2 Neutrosophic Sets in Applications and Disciplines

Neutrosophic has been widely adopted in important areas. – Here we need to include important references for important areas of neutrosophic applications, specially areas where our package can be utilized.

2.1 MADM and MCDM

Utilizing neutrosophic sets in Multi Attributed Decision Making (MADM) and Multi Criteria Decision Making (MCDM) has gained wide acceptance in research and academia.

Analytic Hierarchy Process (AHP) in neutrosophic environment has gained polpularity and achieved success in different cases. In some realistic situations, the decision makers might be unable to assign deterministic evaluation values to the comparison judgments due to limited knowledge or the differences of individual judgments in group decision making. To overcome these challenges, neutrosophic set theory to have been utilized to handle the AHP, where each pair-wise comparison judgment is represented as a triangular neutrosophic number (TNN). [3] presents such a utilization and applies it to a real life example based on expert opinions from Zagazig University, Egypt. The problem is solved to show the effectiveness of the proposed neutrosophic-AHP decision making model.

An Extension of Neutrosophic AHP–SWOT Analysis for Strategic Planning and Decision-Making is presented in [1]. Every organization seeks to set strategies for its development and growth and to do this, it must take into account the factors that affect its success or failure. The most widely used technique in strategic planning is SWOT analysis. SWOT examines strengths (S), weaknesses (W), opportunities (O) and threats (T), to select and implement the best strategy to achieve organizational goals. The chosen strategy should harness the advantages of strengths and opportunities, handle weaknesses, and avoid or mitigate threats. SWOT analysis does not quantify factors (i.e., strengths, weaknesses, opportunities and threats) and it fails to rank available alternatives. To overcome this drawback, [1] integrated it with the analytic hierarchy process (AHP). The AHP is able to determine both quantitative and the qualitative elements by weighting and ranking them via comparison matrices. Due to the vague and inconsistent information that exists in the real world. The proposed model have been applied in a neutrosophic environment in a real case study of Starbucks Company to validate the model.

A Hybrid Neutrosophic Group ANP-TOPSIS Framework for Supplier Selection Problems is presented in [2]. One of the most significant competitive strategies for organizations is sustainable supply chain management (SSCM). The vital part in the administration of a sustainable supply chain is the sustainable supplier selection, which is a multi-criteria decision-making issue, including many conflicting criteria. The valuation and selection of sustainable suppliers are difficult problems due to vague, inconsistent and imprecise knowledge of decision makers. In the literature on supply chain management for measuring green performance, the requirement for methodological analysis of how sustainable variables affect each other, and how to consider vague, imprecise and inconsistent knowledge, is still unresolved. [2] provides an incorporated multi-criteria decision-making procedure for sustainable supplier selection problems (SSSPs). An integrated framework is presented via interval-valued neutrosophic sets to deal with vague, imprecise and inconsistent information that exists usually in real world. The analytic network process (ANP) is employed to calculate weights of selected criteria by considering their interdependencies. For ranking alternatives and avoiding additional comparisons of analytic network processes, the technique for order preference by similarity to ideal solution (TOPSIS) is used. The proposed framework is turned to account for analyzing and selecting the optimal supplier. An actual case study of a dairy company in Egypt is examined within the proposed framework. Comparison with other existing methods is implemented to confirm the effectiveness and efficiency of the proposed approach.

An Integrated Neutrosophic-TOPSIS Approach and its Application to Personnel Selection as a New Trend in Brain Processing and Analysis is presented in [17]. Personnel selection is a critical obstacle that influences the success of enterprise. The complexity of personnel selection is to determine efficiently the proper applicant to fulfill enterprise requirements. The decision makers do their best to match enterprise requirements with the most suitable applicant. Unfortunately, the numerous criteria, alternatives, and goals make the process of choosing among several applicants very complex and confusing to decision makers. The environment of decision making is a MCDM surrounded by inconsistency and uncertainty. [17] contributes to support personnel selection process by integrating neutrosophic Analytical Hierarchy Process (AHP) with Technique for Order Preference by Similarity to an Ideal Solution (TOPSIS) to illustrate an ideal solution among different alternatives. A case study on smart village Cairo Egypt is developed based on decision maker's judgments recommendations. The proposed study applies neutrosophic analytical hierarchy process and TOPSIS to enhance the traditional methods of personnel selection to achieve the ideal solutions. By reaching to the ideal solutions, the smart village will enhance the resource management for attaining the goals to be a success enterprise. The proposed method demonstrates a great impact on the personnel selection process rather than the traditional decision making methods.

Neutrosophic AHP can be used to help decision makers to estimate the influential factors of IoT in enterprises. A study that combines AHP methods with neutrosophic techniques to estimate the influential factors for a successful enterprise is presented in [16].

2.2 Control Systems

The indeterminacy of parameters in actual control systems is inherent property because some parameters in actual control systems are changeable rather than constants in some cases, such as manufacturing tolerances, aging of main components, and environmental changes, which present an uncertain threat to actual control systems. Therefore, these indeterminate parameters can affect the control behavior and performance. [25] develops a new neutrosophic design method that introduces neutrosophic state space models and the neutrosophic controllability and observability in indeterminate linear systems. Then, establishes a neutrosophic state feedback design method for achieving a desired closed-loop state equation or a desired control ratio for single-input single-output (SISO) neutrosophic linear systems.

2.3 Image Processing and Segmentation

Segmentation is considered as an important step in image processing and computer vision applications, which divides an input image into various non-overlapping homogenous regions and helps to interpret the image more conveniently. [11] presents an efficient image segmentation algorithm using neutrosophic graph cut (NGC). An image is presented in neutrosophic set, and an indeterminacy filter is constructed using the indeterminacy value of the input image, which is defined by combining the spatial information and intensity information. The indeterminacy filter reduces the indeterminacy of the spatial and intensity information. A graph is defined on the image and the weight for each pixel is represented using the value after indeterminacy filtering. The segmentation results are obtained using a maximum-flow algorithm on the graph.

Medical field touches everyone's life, and it has benefited a lot from neutrosophic. Fully automated algorithm for image segmentation in medical field is presented in [19]. Such algorithm segments fluid-associated (fluid-filled) and cyst regions in optical coherence tomography (OCT) retina images of subjects with diabetic macular edema. The OCT image is segmented using a novel neutrosophic transformation and a graph-based shortest path method. An image g is transformed into three sets: T (true), I (indeterminate) that represents noise, and F (false). Fully automatic and accurate breast lesion segmentation that utilizes a novel phase feature to improve the image quality, and a novel neutrosophic clustering approach to detect the accurate lesion boundary is presented in [23].

An efficient scheme for unsupervised colour-texture image segmentation using neutrosophic set (NS) and non-subsampled contourlet transform (NSCT) is presented in [13]. First, the image colour and texture information are extracted via CIE Luv colour space model and NSCT, respectively. Then, the extracted colour and texture information are transformed into the NS domain efficiently by the authors' proposed approach. In the NS-based image segmentation, the indeterminacy assessment of the images in the NS domain is notified by the entropy concept. The lower quantity of indeterminacy in the NS domain, the higher confidence and easier segmentation could be achieved. Therefore, to achieve a better segmentation result, an appropriate indeterminacy reduction operation is proposed. Finally, the K -means clustering algorithm is applied to perform the image segmentation in which the cluster number K is determined by the cluster validity analysis.

2.4 Pattern Recognition and Machine Learning

Data clustering, or cluster analysis, is an important research area in pattern recognition and machine learning which helps the understanding of a data structure for further applications. The clustering procedure is generally handled by partitioning the data into different clusters where similarity inside clusters and the dissimilarity between different clusters are high.

New clustering algorithm, neutrosophic c-means (NCM) for uncertain data clustering, which is inspired from fuzzy c-means and the neutrosophic set framework. To derive such a structure, a novel suitable objective function is defined and minimized, and the clustering problem is formulated as a constrained minimization problem, whose solution depends on the objective function in [12].

The work presented in [12] has been extended by [5] via presenting a new clustering algorithm that is called Kernel Neutrosophic c-Means(KNCM), that has been evaluated through extensive experiments.

3 Proposed Novel Neutrosophic Package Design

Attempting informally to categorize neutrosophic research based implementations, it is clear from scanning the neutrosophic literature that spreadsheets are the most widely used tool. Though spreadsheets is an excellent tool for certain types of problems; such as Multi Criteria Decision Making (MCDM), it is not suitable for automated software systems and machine learning based solutions. The need for an open source neutrosophic package to be utilized via different programming languages is clear.

An attempt to utilize Object Oriented Programming to build a neutrosophic package was presented in [21] and [20], and an attempt to use it in e-Learning systems has been presented in [22]. This attempt suffered from different shortages, including not only:

- It was not even a close start to implementing the basic operations of any neutrosophic family subset
- It was not an open source based attempt, so its source code was never made available to neutrosophic research community to be tested, verified, validated, and utilized
- Though that solution utilized Object Oriented Concepts in CSharp, Programming was used only to calculate values to be exported to Microsoft Excel that was used for plotting the graphs. This approach suffers a lot when developers attempt to integrate it within software solutions.
- Presented OOP CSharp based package has not been tested, verified, or used in real world situations
- There is no clear documentation and illustration of the design of that package. When combined with the lack of software source code, utilizing such a package is almost impossible
- It is clear that it was a very early immature attempt that never made its way through implementation and adoption in neutrosophic community or real world examples and projects. There has been no updates or research papers related to this package since then

In this section, key decisions about the presented novel neutrosophic open source package are discussed. Those key decisions reflects the philosophy of the package author, and shapes the current state, and the future state of the neutrosophic package. Those decisions include:

- Programming Language used
- Open Source Choice (Source Code Availability)
- Neutrosophic Package Licensing
- Packaging choices and alternatives
- Programming Methodology

3.1 Python

Python is considered a multilanguage model, where a high-level language is used to interface libraries and software packages written in low-level languages. In a high-level scientific computing environment, this type of interoperability with software packages written in low-level languages (e.g., Fortran, C, or C++) is an important requirement. Python excels at this type of integration, and as a result, Python has become an interface for setting up and controlling computations that use code written in low-level programming languages for time-consuming number crunching. This is an important reason for why Python is a popular language for numerical computing. The multilanguage model enables rapid code development in a high-level language while retaining most of the performance of low-level languages. [14]

Table 1: MIT License Permissions, Limitations, and Conditions

Permissions	**Limitations**	**Conditions**
✓Commercial use	✗Liability	License and copyright notice
✓Modification	✗Warranty	
✓Distribution		
✓Private use		

3.2 Open Source

Open Source software is a form of intellectual gratification with an intrinsic utility similar to that of scientific discovery [8]. Emerging as it does from the university and research environment, the movement adopts the motivations of scientific research, transferring them into the production of technologies that have a potential commercial value. The process of scientific discovery involves the sharing of results, just as the dictates of the Open Source movement involve sharing source code. Sharing results enables researchers both to improve their results through feedback from other members of the scientific community and to gain recognition and hence prestige for their work. The same thing happens when source code is shared: other members of the group provide feedback that helps to perfect it, while the fact that the results are clearly visible to everyone confers a degree of prestige which expands in proportion to the size of the community.

3.3 Neutrosophic Package Licensing

Presented Neutrosophic Package is licensed under the MIT License. Table 1 presents the MIT License Permissions, Limitations, and Conditions. MIT License is a short and simple permissive license with conditions only requiring preservation of copyright and license notices. Licensed works, modifications, and larger works may be distributed under different terms and without source code. License and copyright notice condition means that a copy of the license and copyright notice must be included with the software. License can be found at `https://github.com/helghareeb/neutrosophic/blob/master/LICENSE`

3.4 Software Packaging

Effective reuse depends not only on finding and reusing components, but also on the ways those components are combined [24]. Software engineering provides a number of diverse styles for organizing software systems. These styles, or architectures, show how to compose systems from components; different styles expect different kinds of component packaging and different kinds of interactions between the components. Unfortunately, these styles and packaging distinctions are often implicit; as a consequence, components with appropriate functionality may fail to work together. Different packaging techniques have been presented in both academia and industry. Though there is no single agreed on packaging methodology; due to differences in features and programming languages, different guidelines are available to achieve successful packaging process. Presented Python Neutrosophic package will be packaged and shipped as a standard Python package.

3.5 Object Oriented Programming

Object oriented programming departs from conventional programming by emphasizing the relationship between consumers and suppliers of codes rather then the relationship between a programmer and code [10].

Main Object Oriented Concepts include [6]

- Inheritance: a mechanism by which object implementations can be organized to share descriptions
- Object: both a data carrier and executes actions. Object is something that has state, behavior, and identity
- Class: set of objects described by the same declaration and is the basic element of Object Oriented modeling
- Encapsulation: There are three primary conceptualizations of encapsulation in the literature.
  - First conceptualization: a process used to package data with the functions that act on the data
  - Second conceptualization: hides the details of the object's implementation so that clients access the object only via its defined external interface
  - Third conceptualization: information about an object, how that information is processed, kept strictly together, and separated from everything else
- Method: involves accessing, setting, or manipulating the object's data
- Message Passing: Message is merely a procedure call from one function to another. Message passing makes a request to one of object's methods
- Polymorphism: There are different conceptualization
  - First conceptualization: ability to hide different implementations behind a common interface
  - Second conceptualization: ability of different objects to respond to the same message and invoke different responses
  - Third conceptualization: ability of different classes to contain different methods of the same name, which appear to behave the same way in a given context; yet different objects can respond to the same message with their own behavior
  - Fourth conceptualization: refers to late binding or dynamic binding
  - Fifth conceptualization: ability of different classes to respond to the same message and each implement the method appropriately
- Abstraction:
  - First conceptualization: mechanism that allows representing a complex reality in terms of a simplified model so that irrelevant details can be suppressed in order to enhance understanding
  - Second conceptualization: the act of removing certain distinctions between objects so that we can see commonalities
  - Third conceptualization: the act of creating classes to simplify aspects of reality using distinctions inherent to the problem

4 Single Valued Neutrosophic Number

4.1 Constructor

Section 1 highlighted that each neutrosophic number consists of three elements: truth, indeterminacy, and false values. Listing 1 highlights the constructor function (the function that gets invoked automatically) when instantiating a new object instance from the Single Valued Neutrosophic Class. New Single Valued Neutrosophic Number validates the values of T, I, F to satisfy 1.1.

Listing 1: SVNN - Constructor

```
class SingleValuedNeutrosophicNumber:

    def __init__(self, id, truth, indeterminacy, falsehood):
        """Initialize neutrosophic element
        :truth:
        :indeterminacy:
        :falsehood:"""
        assert id is not None, 'provide id for element to be initialized'
        assert 0 <= truth <= 1, 'invalid truth value'
        assert 0 <= indeterminacy <= 1, 'invalid indeterminacy value'
        assert 0 <= falsehood <= 1, 'invalid falsehood value'
        assert 0 <= truth + falsehood + indeterminacy <= 3, 'invalid combined sum
            ↪ values'
        self._id = id
        self._truth = truth
        self._indeterminacy = indeterminacy
        self._falsehood = falsehood
```

4.2 SVNN Operations

Single Valued Nuetrosophic Numbers arithmetic operations are defined in [15] as follows: Let two single-valued neutrosophic numbers be

$$x = \langle T_x, I_x, F_x \rangle$$

$$y = \langle T_y, I_y, F_y \rangle$$

4.2.1 SVNN Complement

Calculating SVNN Complement is based on the Equation 4.1 and implemented in Listing 2

$$x^c = \langle F_x, 1 - I_x, T_x \rangle \tag{4.1}$$

Listing 2: SVNN - Complement

```
    def complement(self):
        """
        :return: SVNN object with the new TIF values
        """
        return SingleValuedNeutrosophicNumber (f'{self._id}_complement', self.
            ↪ _falsehood, 1 − self._indeterminacy, self._truth)
```

4.2.2 SVNN is subset of

Identifying either an SVNN is a subset of another SVNN is determined based on the Equation 4.2 and implemented in Listing 3. This method returns a `bool` value type with either `True` or `False`, if the current SVNN object is a subset of another SVNN.

$$x \subseteq y \iff T_x \leq T_y, I_x \geq I_y, F_x \geq F_y \tag{4.2}$$

Listing 3: SVNN - is subset of

```
        def is_subset_of(self, svnn):
                """Check if SVNN is a subset of another SVNN
                :param svnn: Single Value Neutrosohpic Number to compare with
                :return: True or False
                """
                if self._truth <= svnn._truth and self._indeterminacy >= svnn.
                    ↪ _indeterminacy and self._falsehood >= svnn._falsehood:
                        return True
                return False
```

4.2.3 SVNN Equal

Comparing two SVNN to detect if they are equal or not is calculated based on the Equation 4.3 and implemented in Listing 4. One of the advantages of Python magic is utilized in this function via implementing it as `__eq__` which gives the neutrosophic package capability of comparing two SVNN numbers via the equal sign operator.

$$x = y \iff x \subseteq y, y \subseteq x \tag{4.3}$$

Listing 4: SVNN - Equal

```
        def __eq__(self, svnn):
                if self.is_subset_of(svnn) and svnn.is_subset_of(self):
                        return True
                return False
```

4.2.4 SVNN Add

Two SVNNs can be added using the Equation 4.4 and implemented in Listing 5. Added SVNNs return a new SVNN. Using Python magic by implementing the add functionality through `__add__` enables us to utilize the plus operator $\oplus$ operator on SVNNs.

$$x \oplus y = \langle T_x + T_y - T_x T_y, I_x I_y, F_x F_y \rangle \tag{4.4}$$

Listing 5: SVNN Add

```
  def __add__(self, svnn):
    return svnn(f'{self._id} + {svnn._id}', (self._truth + svnn._truth) - (self.
        ↪ _truth * svnn._truth), self._indeterminacy * svnn._indeterminacy, self.
        ↪ _falsehood * svnn._falsehood)
```

4.2.5 SVNN Multiply by SVNN

Two SVNNs can be multiplied by the Equation 4.5 and implemented in Listing 6. Using Python magic by implementing the multiply functionality through `__mul__` enables us to utilize the multiply operator $\otimes$ on SVNNs.

$$x \otimes y = \langle T_x T_y, I_x + I_y - I_x I_y, F_x + F_y - F_x F_y \rangle \tag{4.5}$$

Listing 6: SVNN Multiply by SVNN

```
    def __mul__(self, svnn):
        return svnn(f'{self._id} * {svnn._id}', self._truth * svnn._truth, svnn
          ↪ ._indeterminacy - (self._indeterminacy * svnn._indeterminacy), (
          ↪ self._falsehood + svnn._falsehood) - (self._falsehood * svnn.
          ↪ _falsehood))
```

4.2.6 SVNN Multiply by Alpha

SVNN can be multiplied by constant (alpha) using the Equation 4.6 and implemented in Listing 7. Multiplying by constant is an important operation that is very useful in scaling, that is crucial for computer graphics and image processing, among other fields.

$$\alpha x = \langle 1 - (1 - T_x)^\alpha, I_x^\alpha, F_x^\alpha \rangle \longleftarrow \alpha > 0 \tag{4.6}$$

Listing 7: Multiply by Number

```
    def multiply_by_alpha(self, alpha):
        assert alpha > 0, 'Alpha must be larger than zero'
        return svnn(f'{self._id}_multiplied_by_{alpha}', 1 - pow(1 - self.
          ↪ _truth), alpha, pow(self._indeterminacy, alpha), pow(self.
          ↪ _falsehood, alpha))
```

4.2.7 SVNN Score

Calculating SVNN Score is important for MCDM. Listing 8 presents the implementation of the SVNN Score function calculated in Equation 4.7

$$E(x) = \frac{(2 + T_x - I_x - F_x)}{3}, E(x) \in [0, 1] \tag{4.7}$$

Listing 8: SVNN Score

```
    def score(self):
        return ( 2 + self._truth - self._indeterminacy - self._falsehood ) /
          ↪ 3
```

4.2.8 SVNN Accuracy

SVNN Accuracy also plays an important rule in MCDM. Examples include, not only: rule engines. Listing 9 highlights the Python code that calculates SVNN Accuracy presented in Equation 4.8

$$H(x) = T_x - F_x, H(x) \in [-1, 1] \tag{4.8}$$

Listing 9: SVNN Accuracy

```
    def accuracy(self):
        return self._truth - self._falsehood
```

4.2.9 SVNN Ranking

The ranking method is based on both the score values of E(x) and E(y) [15] and the accuracy degrees of H(x) and H(y) has the following relations depicted in Equations 4.9, 4.10, 4.11 and implemented in Listing 10 as follows:

$$\text{if } E(x) > E(y) \text{ then } x \succ y \tag{4.9}$$

$$\text{if } E(x) = E(y) \text{ and } H(x) > H(y) \text{ then } x \succ y \tag{4.10}$$

$$\text{if } E(x) = E(y) \text{ and } H(x) > H(y) \text{ then } x = y \tag{4.11}$$

Listing 10: SVNN Rank

```
    def ranking_compared_to(self, svnn):
        """
        :param svnn:
        :return: -1 -> Not Applicable, 0 -> equal ranking, 1 -> higher ranking
        """
        if self.score() > svnn.score():
                return 1
            if self.score() == svnn.score() and self.accuracy() > svnn.accuracy()
              ↪ :
                return 1
            if self.score() == svnn.score() and self.accuracy() == svnn.accuracy
              ↪ ():
                return 0
            return -1
```

4.2.10 SVNN Deneutrosophication / Score Function

Deneutrosophication can be defined as mapping a Single Valued Neutrosophic Number into a crisp output and is calculated in [7] as

$$\psi = 1 - \sqrt{\frac{(1-T_x)^2 + I_x^2 + F_x^2}{3}} \tag{4.12}$$

Listing 11 presents the Python code required to implement the Equation 4.12

Listing 11: SVNN Deneutrosophy

```
    def deneutrosophy(self):
        from math import sqrt, pow
        return 1 - (sqrt (((pow(1 - self._truth),2) + pow(self._indeterminacy
          ↪ ,2) + pow(self._falsehood,2)) / 3))
```

4.3 SVNN Helper Methods

Those are additional methods required for coding, debugging, and documentation purposes. SVNN additional methods are listed in Listing 12

Listing 12: SVNN - Helper Methods

```
  def __str__(self):
      return f'ID: {self._id} - Truth: {self._truth} - Indeterminacy: {self.
        ↪ _indeterminacy} - Falsehood: {self._falsehood}'

  def __repr__(self):
      return f'ID: {self._id} - Truth: {self._truth} - Indeterminacy: {self.
        ↪ _indeterminacy} - Falsehood: {self._falsehood}'
```

5 Single Valued Neutrosophic Sets

The implementation in Listing 13 represents thinking of a Single Valued Neutrosophic Set (SVNS) as a Set of Single Valued Neutrosophic Numbers (SVNNs). Utilizing Object Oriented Concepts in proposed neutrosophic package, SVNN is presented as a class, and SVNS is presented as another class, and there is an association relationship between them. This justifies importing SVNN within SVNS class.

Listing 13: SVNS - Constructor

```
from svnn import SingleValuedNeutrosophicNumber

class SVNSet:
"""This class has association relationship with SVNN
"""

        def __init__(self):
                # List of SVNNs
                self._items = []
                # Index variable – used for iteration over SVNNs in SVNS
                self.__idx = -1
```

5.1 SVNS Hybrid Arithmetic Operators

where $\wedge$ is the t-norm, and $\vee$is the t-conorm. Hybrid arithmetic and geometric aggregation operators are defined in [15] as follows

- Single Valued Neutrosophic Number Weighted Arithmetic Average (SVNNWAA)
- Single Valued Neutrosophic Number Weighted Geometric Average (SVNNWGA)
- Single Valued Neutrosophic Number Ordered Weighted Arithmetic Average (SVNNOWAA)
- Single Valued Neutrosophic Number Ordered Weighted Geometric Average (SVNNOWGA)

5.1.1 Single Valued Neutrosophic Number Weighted Arithmetic Average (SVNNWAA)

Listing 14 presents the Python code that calculates SVNWAA as presented in Equation 5.1

$$SVNNWAA(z_1, z_2, \ldots, z_n) = \sum_{j=1}^{n} w_j z_j = \langle 1 - \prod_{j=1}^{n}(1 - T_j)^{w_j}, \prod_{j=1}^{n}(U_j)^{w_j}, \prod_{j=1}^{n}(V_j)^{w_j}\rangle \quad (5.1)$$

Listing 14: SVNNWAA

```
def weighted_arithmetic_average(self, weights):
        """
        single–valued neutrosophic number weighted arithmetic average (SVNNWAA)
        weights: List of weights of each item – list length must be equal to the length of the items
        For more information: Google weighted arithmetic average
        or watch https://www.youtube.com/watch?reload=9&v=IuuBU6fwtNo
        :return: Three values: T, U, V
        """
        assert len(weights) == len(self._items), 'Weights List Length Does Not
            ↪ Match Collection SVNN Items'
        weights_sum = 0.0
        for weight in weights:
            weights_sum += weight
        assert weights_sum == 1, 'Weight\'s sum does not equal 1'
        truth_total = 1.0
        indetermenacy_total = 1.0
        falsehood_total = 1.0
        for item, weight in zip(self._items, weights):
            truth_total *= pow(1 - item._truth, weight)
            indetermenacy_total *= pow(item._indeterminacy, weight)
            falsehood_total *= pow(item._falsehood, weight)
        return 1 - truth_total, indetermenacy_total, falsehood_total
```

5.1.2 Single Valued Neutrosophic Number Weighted Geometric Average (SVNNWGA)

Listing 15 implements Equation 5.2.

$$SVNNWGA(z_1, z_2, \ldots, z_n) = \prod_{j=1}^{n} z_j^{w_j} = \langle \prod_{j=1}^{n} (T_j)^{w_j}, 1 - \prod_{j=1}^{n} (1 - U_j)^{w_j}, 1 - \prod_{j=1}^{n} (1 - V_j)^{w_j} \rangle \quad (5.2)$$

Listing 15: SVNNWGA

```
    def weighted_geometric_average(self, weights):
        """single-valued neutrosophic number weighted geometric average
        """
        weights_sum = 0.0
        for weight in weights:
            weights_sum += weight
        assert weights_sum == 1, 'Weight\'s sum does not equal 1'
        truth_total = 1.0
        indetermenacy_total = 1.0
        falsehood_total = 1.0
        weights.sort()
        for item, weight in zip(self._items, weights):
            truth_total *= pow(item._truth, weight)
            indetermenacy_total *= pow(1 - item._indeterminacy, weight)
            falsehood_total *= pow(1 - item._falsehood, weight)
        return truth_total, 1 - indetermenacy_total, 1 - falsehood_total
```

5.2 SVNS Geometric Aggregation Operators

5.2.1 Single Valued Neutrosophic Number Ordered Weighted Arithmetic Average (SVNNOWAA)

Listing 16 presents Python implementation of Equation 5.3.

$$SVNNOWAA(z_1, z_2, \ldots, z_n) = \sum_{j=1}^{n} \zeta_j z_{p(j)} = \langle 1 - \prod_{j=1}^{n} (1 - T_{p(j)})^{\zeta_j}, \prod_{j=1}^{n} (U_{p(j)})^{\zeta_j}, \prod_{j=1}^{n} (V_{p(j)})^{\zeta_j} \rangle \quad (5.3)$$

Listing 16: SVNNOWAA

```
def ordered_weighted_arithmetic_average(self, weights, ordered_by_position =
    ↪ False):
        assert len(weights) == len(self._items), 'Weights List Length Does Not
            ↪  Match Collection SVNN Items'
        weights_sum = 0.0
        for weight in weights:
                weights_sum += weight
        assert weights_sum == 1, 'Weight\'s sum does not equal 1'
        truth_total = 1.0
        indetermenacy_total = 1.0
        falsehood_total = 1.0
        for item, weight in zip(self._items, weights):
                truth_total *= pow(1 - item._truth, weight)
                indetermenacy_total *= pow(item._indeterminacy, weight)
                falsehood_total *= pow(item._falsehood, weight)
        return 1 - truth_total, indetermenacy_total, falsehood_total
```

5.2.2 Single Valued Neutrosophic Number Ordered Weighted Geometric Average (SVNNOWGA)

Listing 17 depicts the implementation of Equation 5.4. Python provides efficient ways that helps in building such complicated calculations. Here, Weights are sorted to be used for the calculation. Python utilizes efficient builtin methods, for example like the one presented for sorting.

$$SVNNOWGA(z_1, z_2, \ldots, z_n) = \prod_{j=1}^{n} z_{p(j)}^{\zeta_j} = \langle \prod_{j=1}^{n} (T_{p(j)})^{\zeta_j}, 1 - \prod_{j=1}^{n} (1 - U_{p(j)})^{\zeta_j}, 1 - \prod_{j=1}^{n} (1 - V_{p(j)})^{\zeta_j} \rangle \quad (5.4)$$

Listing 17: SVNNOWGA

```
def ordered_weighted_geometric_average(self, weights):
        """single-valued neutrosophic number weighted geometric average
        """
        weights_sum = 0.0
        for weight in weights:
            weights_sum += weight
        assert weights_sum == 1, 'Weight\'s sum does not equal 1'
        truth_total = 1.0
        indetermenacy_total = 1.0
        falsehood_total = 1.0
        # The following line is the main difference
        weights.sort()
        for item, weight in zip(self._items, weights):
            truth_total *= pow(item._truth, weight)
            indetermenacy_total *= pow(1 - item._indeterminacy, weight)
            falsehood_total *= pow(1 - item._falsehood, weight)
        return truth_total, 1 - indetermenacy_total, 1 - falsehood_total
```

5.3 SVNS Helper Methods

Additional helper methods are needed for supporting basic SVNS operations, such as

5.3.1 Add SVNN

Supports adding SVNN to SVNS, as depicted in Listing 18

Listing 18: SVNS - Add SVNN

```
    def add_svnn(self, svnn):
    # TODO: Prevent Duplication
        self._items.append(svnn)
```

5.3.2 Delete SVNN

Supports removing SVNN from SVNS, as implemented in Listing 19

Listing 19: SVNS - Delete SVNN

```
    def delete_svnn(self, svnn):
        #TODO: Notify user about Exception handling
        self._items.remove(svnn)
```

5.3.3 Retrieve All SVNNs

Retrieving a list of all SVNNs in SVNS is a crucial task. Returned list is an iterable one that can be used for further processing. Listing 20 presents such functionality implementation.

Listing 20: SVNS - Retrieve All SVNNs

```
    def get_all_svnns(self):
        return self._items
```

5.3.4 Count All SVNNs in SVNS

Counting all SVNNs in SVNS is a primitive task. Listing 21 presents the code to implement such a functionality. Using Python magic, via utilizing the `__len__` enables us to use the `len()` function syntax over SVNN object.

Listing 21: SVNS - Count All SVNNs

```
    def __len__(self):
        return len(self._items)
```

5.3.5 SVNS: is_empty

Though checking either SVNS is empty or not can be achieved via `__len__()` function, it is important to enable proposed neutrosophic package to check `is_empty()` in conditionals. Listing 22 depicts such functionality.

Listing 22: SVNS - is_empty

```
    def is_empty(self):
        if len(self) == 0:
            return True
        return False
```

5.3.6 SVNS - Iteration

Iteration is a general term for taking each item of something, one after another. While using a loop for example, going over a group of items is called iteration. An iterable object is an object that has an `__iter__` method which returns an iterator. `__getitem__` method can take sequential indexes starting from zero (and raises an IndexError when the indexes are no longer valid). An iterator is an object with a `__next__` method.

Providing iteration functionality within our proposed neutrosophic package is critical, so later users can either loop, or apply map functionalities over SVNNs within SVNSs. Such characteristic is an important feature for future use cases. Listing 23 depicts how iteration functionality is implemented in SVNS.

Listing 23: SVNS - Iteration

```
    def __iter__(self):
        return self

    def __next__(self):
        self.__idx += 1
        try:
            return self._items[self.__idx]
        except IndexError:
            self.__idx = 0
            raise StopIteration

    def __getitem__(self, id):
        try:
                return self._items[id]
            except IndexError:
                raise StopIteration
```

6 Interval Valued Neutrosophic Number

Given the following definitions, operational laws can be applied as defined in [9]

$$\overset{\bullet}{N}_1 = \left\{ < x : \left[T^L_{\overset{\bullet}{N}_1}, T^U_{\overset{\bullet}{N}_1} \right], \left[I^L_{\overset{\bullet}{N}_1}, I^U_{\overset{\bullet}{N}_1} \right], \left[F^L_{\overset{\bullet}{N}_1}, F^U_{\overset{\bullet}{N}_1} \right] >, x \in X \right\}$$

$$\overset{\bullet}{N}_2 = \left\{ < x : \left[T^L_{\overset{\bullet}{N}_2}, T^U_{\overset{\bullet}{N}_2} \right], \left[I^L_{\overset{\bullet}{N}_2}, I^U_{\overset{\bullet}{N}_2} \right], \left[F^L_{\overset{\bullet}{N}_2}, F^U_{\overset{\bullet}{N}_2} \right] >, x \in X \right\}$$

Listing 24 presents the IVNN Class Declaration and Constructor. In the presented implementation, `t_lower` ↪ and `t_upper` for example represents the following mathematical symbols respectively

$$T^L_{\overset{\bullet}{N}_1}, T^U_{\overset{\bullet}{N}_1}$$

Listing 24: IVNN Class Declaration and Constructor

```
class IVNN:

    def __init__(self, id, t_lower, t_upper, i_lower, i_upper, f_lower, f_upper):

        assert 0 <= t_lower <= 1
        assert 0 <= t_upper <= 1
        assert 0 <= i_lower <= 1
        assert 0 <= i_upper <= 1
        assert 0 <= f_lower <= 1
        assert 0 <= f_upper <= 1

        assert 0 <= t_lower + i_lower + f_lower <= 3

        self._id = id
        self._t_lower = t_lower
        self._t_upper = t_upper
        self._i_lower = i_lower
        self._i_upper = i_upper
        self._f_lower = f_lower
        self._f_upper = f_upper
```

6.1 IVNN Operations

6.1.1 IVNN Complement

The complement of an interval valued neutrosophic number

$$A = \langle [T_A^l, T_A^r], [I_A^l, I_A^r], [F_A^l, F_A^r] \rangle$$

is defined by [4] as

$$A^c = \langle [F_A^l, F_A^r], [I_A^l, I_A^r], [T_A^l, T_A^r] \rangle \tag{6.1}$$

Listing 25: IVNN - Complement

```
    def complement(self):
        return IVNN(f'{self._id}_complement',
        self._f_lower,
        self._f_upper,
        self._i_lower,
        self._i_upper,
        self._t_lower,
        self._t_upper)
```

6.1.2 IVNN Add

Two SVNNs can be added using the Equation 6.2 and implemented in Listing 26. Added IVNNs return a new IVNN. Again, using Python magic by implementing the add functionality through __add__ enables us to utilize the plus operator ⊕ operator on IVNNs.

$$\dot{N}_1 \oplus \dot{N}_2 = \left\langle \left[T^L_{\dot{N}_1} + T^L_{\dot{N}_2} - T^L_{\dot{N}_1} T^L_{\dot{N}_2}, T^U_{\dot{N}_1} + T^U_{\dot{N}_2} - T^U_{\dot{N}_1} T^U_{\dot{N}_2} \right], \left[I^L_{\dot{N}_1} I^L_{\dot{N}_2}, I^U_{\dot{N}_1} I^U_{\dot{N}_2} \right], \left[F^L_{\dot{N}_1} F^L_{\dot{N}_2}, F^U_{\dot{N}_1} F^U_{\dot{N}_2} \right] \right\rangle \tag{6.2}$$

Listing 26: IVNN - Add Two IVNNs

```
    def __add__(self, ivnn):
        return IVNN(f'{self._id} + {ivnn._id}',
        self._t_lower + ivnn._t_lower − self._t_lower ∗ ivnn._t_lower,
        self._t_upper + ivnn._t_upper − self._t_upper ∗ ivnn._t_upper,
        self._i_lower ∗ ivnn._i_lower,
        self._i_upper ∗ ivnn._i_upper,
        self._f_lower ∗ ivnn._f_lower,
        self._f_upper ∗ ivnn._f_upper)
```

6.1.3 IVNN Multiply by IVNN

Two IVNNs can be multiplied by the Equation 6.3 and implemented in Listing 27. Using Python magic by implementing the multiply functionality through `__mul__` enables us to utilize the multiply operator $\otimes$ on IVNNs.

$$\dot{N}_1 \otimes \dot{N}_2 = \left\langle \left[T^L_{\dot{N}_1} T^L_{\dot{N}_2}, T^U_{\dot{N}_1} T^U_{\dot{N}_2} \right], \left[I^L_{\dot{N}_1} + I^L_{\dot{N}_2} - I^L_{\dot{N}_1} I^L_{\dot{N}_2}, I^U_{\dot{N}_1} + I^U_{\dot{N}_2} - I^U_{\dot{N}_1} I^U_{\dot{N}_2} \right], \left[F^L_{\dot{N}_1} + F^L_{\dot{N}_2} - F^L_{\dot{N}_1} F^L_{\dot{N}_2}, F^U_{\dot{N}_1} + F^U_{\dot{N}_2} - F^U_{\dot{N}_1} F^U_{\dot{N}_2} \right] \right\rangle \tag{6.3}$$

Listing 27: IVNN - Multiply Two IVNNs

```
    def __mul__(self, ivnn):
        return IVNN(f'P{self._id} * {ivnn._id}',
        self._t_lower * ivnn._t_lower,
        self._t_upper * ivnn._t_upper,
        self._i_lower + ivnn._i_lower - self._i_lower * ivnn._i_lower,
        self._i_upper + ivnn._i_upper - self._i_upper * ivnn._i_upper,
        self._f_lower + ivnn._f_lower - self._f_lower * ivnn._f_lower,
        self._f_upper + ivnn._f_upper - self._f_upper * ivnn._f_upper)
```

6.1.4 IVNN Multiply by Alpha

IVNN can be multiplied by constant (alpha) using the Equation 6.4 and implemented in Listing 28.

$$\delta \dot{N} = \left\langle \left[1 - \left(1 - T^L_N\right)^\delta, 1 - \left(1 - T^U_N\right)^\delta \right], \left[\left(T^L_N\right)^\delta, \left(T^U_N\right)^\delta \right], \left[\left(F^L_N\right)^\delta, \left(F^U_N\right)^\delta \right] \right\rangle \tag{6.4}$$

Listing 28: IVNN - Multiply by Alpha

```
    def multiply_by(self, alpha):
        return IVNN(f'{alpha} * {self._id}',
        1 - pow((1 - self._t_lower),alpha),
        1 - pow((1 - self._t_upper),alpha),
        pow(self._i_lower, alpha),
        pow(self._i_upper, alpha),
        pow(self._f_lower, alpha),
        pow(self._i_upper, alpha))
```

7 Interval Valued Neutrosophic Sets

7.1 IVNS - Weighted Average

Interval Neutrosophic Number Weighted Average Operator (INNWA) defined by [26] Let

$$A_j = \langle T_{A_j}, I_{A_j}, F_{A_j} \rangle (j = 1, 2, ..., n)$$

be a collection of IVNNs, and let

$$INNWA : INN^n \to INN$$

$$\begin{aligned} & INNWA_w(A_1, A_2, \ldots, A_n) \\ = & \langle[1 - \textstyle\prod_{i=1}^{n}(1 - \inf T_{A_i})^{w_i}, 1 - \prod_{i=1}^{n}(1 - \sup T_{A_i})^{w_i}], \\ & [\textstyle\prod_{i=1}^{n} \inf I_{A_i}^{w_i}, \prod_{i=1}^{n} \sup I_{A_i}^{w_i}], \\ & [\textstyle\prod_{i=1}^{n} \inf F_{A_i}^{w_i}, \prod_{i=1}^{n} \sup F_{A_i}^{w_i}]\rangle, \end{aligned} \tag{7.1}$$

Listing 29 presents Python implementation of 7.1

Listing 29: INNWA

```
    def weighted_average(self, weights):
        """
        :return: IVNN
        """
        weights_sum = 0
        for weight in weights:
            assert 0 <= weight <= 1
            weights_sum += weight
        assert weights_sum == 1

        t_lower_dot_product = 1.0
        t_upper_dot_product = 1.0
        i_lower_dot_product = 1.0
        i_upper_dot_product = 1.0
        f_lower_dot_product = 1.0
        f_upper_dot_product = 1.0

        for ivnn, weight in zip(self._ivnns, weights):
            t_lower_dot_product *= pow(1 - ivnn._t_lower, weight)
            t_upper_dot_product *= pow(1 - ivnn._t_upper, weight)
            i_lower_dot_product *= pow(ivnn._i_lower, weight)
            i_upper_dot_product *= pow(ivnn._i_upper, weight)
            f_lower_dot_product *= pow(ivnn._f_lower, weight)
            f_upper_dot_product *= pow(ivnn._i_upper, weight)

        return IVNN(1 - t_lower_dot_product, 1 - t_upper_dot_product,
          ↪ i_lower_dot_product, i_upper_dot_product, f_lower_dot_product,
          ↪ f_upper_dot_product)
```

7.2 IVNS Helper Methods

Additional IVNS helper method is presented in Listing 30. IVNS is a collection of IVNNs, and thus the method `add_ivnn` is presented.

Listing 30: IVNN - Helper Methods

```
        def add_ivnn(self, ivnn):
                self._ivnns.append(ivnn)
```

8 Conclusion and Future Work

Neutrosophic sets has gained wide popularity and acceptance in different disciplines. This paper presented an Open Source Python Neutrosophic package. Presented package utilizes Object Oriented Design and implementation concepts. Presented package is licensed under MIT License. Licensing was chosen carefully to support and enable both open source and neutrosophic community. Python was chosen for this package as a result of Python's wide applicability in different paradigms, including mainly Big Data Analytics, Machine Learning, and Artificial Intelligence. Presented package is an open source one, so developers and researchers in different disciplines can adopt it effectively. Presented Neutrosophic package is a work on progress, as Neutrosophic sets and theory becomes more popular and gets utilized in different fields. Presented package presented support for: Single Valued Neutrosophic Numbers, Single Valued Neutrosophic Sets, Interval Valued Neutrosophic Numbers, and Interval Valued Neutrosophic Sets. Different operations were presented. Presented package can be found at `https://www.github.com/helghareeb/neutrosophic`.

The main challenge was the multiple definitions and proofs for the same operation, with different calculation methods. Example of such a challenge is the Score Function. There are numerous deneutrosophy functions for the same neutrosophic number, each with its own proof. Future Work includes uploading the presented Neutrosophic package to one of the most widely utilized Python Package servers. Besides, porting the presented neutrosophic package into different Programming Languages. Implementing Different Deneutrosophication / Score Functions, highlighting the differences between them is another step to take. Support of Triangular and Trapezoidal Neutrosophic Numbers is another challenge to tackle.

References

[1] Abdel-Basset, M., Mohamed, M., and Smarandache, F. An extension of neutrosophic ahp–swot analysis for strategic planning and decision-making. *Symmetry 10*, 4 (2018), 116.

[2] Abdel-Basset, M., Mohamed, M., and Smarandache, F. A hybrid neutrosophic group anp-topsis framework for supplier selection problems. *Symmetry 10*, 6 (2018), 226.

[3] Abdel-Basset, M., Mohamed, M., Zhou, Y., and Hezam, I. Multi-criteria group decision making based on neutrosophic analytic hierarchy process. *Journal of Intelligent & Fuzzy Systems 33*, 6 (2017), 4055–4066.

[4] Aiwu, Z., Jianguo, D., and Hongjun, G. Interval valued neutrosophic sets and multi-attribute decision-making based on generalized weighted aggregation operator. *Journal of Intelligent & Fuzzy Systems 29*, 6 (2015), 2697–2706.

[5] Akbulut, Y., Şengür, A., Guo, Y., and Polat, K. Kncm: Kernel neutrosophic c-means clustering. *Applied Soft Computing 52* (2017), 714 – 724.

[6] ARMSTRONG, D. J. The quarks of object-oriented development. *Communications of the ACM 49*, 2 (2006), 123–128.

[7] AWANG, A., GHANI, A. T. A., ABDULLAH, L., AND AHMAD, M. F. A dematel method with single valued neutrosophic set (svns) in identifying the key contribution factors of setiu wetland's coastal erosion. In *AIP Conference Proceedings* (2018), vol. 1974, AIP Publishing, p. 020011.

[8] BONACCORSI, A., AND ROSSI, C. Why open source software can succeed. *Research policy 32*, 7 (2003), 1243–1258.

[9] BROUMI, S., NAGARAJAN, D., BAKALI, A., TALEA, M., SMARANDACHE, F., AND LATHAMAHESWARI, M. The shortest path problem in interval valued trapezoidal and triangular neutrosophic environment. *Complex & Intelligent Systems* (Feb 2019).

[10] COX, B. J. Object-oriented programming: an evolutionary approach.

[11] GUO, Y., AKBULUT, Y., ŞENGÜR, A., XIA, R., AND SMARANDACHE, F. An efficient image segmentation algorithm using neutrosophic graph cut. *Symmetry 9*, 9 (2017).

[12] GUO, Y., AND SENGUR, A. Ncm: Neutrosophic c-means clustering algorithm. *Pattern Recognition 48*, 8 (2015), 2710 – 2724.

[13] HESHMATI, A., GHOLAMI, M., AND RASHNO, A. Scheme for unsupervised colour–texture image segmentation using neutrosophic set and non-subsampled contourlet transform. *IET Image Processing 10*, 6 (2016), 464–473.

[14] JOHANSSON, R. Introduction to computing with python. In *Numerical Python*. Springer, 2019, pp. 1–41.

[15] LU, Z., AND YE, J. Single-valued neutrosophic hybrid arithmetic and geometric aggregation operators and their decision-making method. *Information 8*, 3 (2017).

[16] NABEEH, N. A., ABDEL-BASSET, M., EL-GHAREEB, H. A., AND ABOELFETOUH, A. Neutrosophic multi-criteria decision making approach for iot-based enterprises. *IEEE Access* (2019), 1–1.

[17] NABEEH, N. A., SMARANDACHE, F., ABDEL-BASSET, M., EL-GHAREEB, H. A., AND ABOELFETOUH, A. An integrated neutrosophic-topsis approach and its application to personnel selection: A new trend in brain processing and analysis. *IEEE Access 7* (2019), 29734–29744.

[18] OTAY, İ., AND KAHRAMAN, C. *A State-of-the-Art Review of Neutrosophic Sets and Theory*. Springer International Publishing, Cham, 2019, pp. 3–24.

[19] RASHNO, A., KOOZEKANANI, D. D., DRAYNA, P. M., NAZARI, B., SADRI, S., RABBANI, H., AND PARHI, K. K. Fully automated segmentation of fluid/cyst regions in optical coherence tomography images with diabetic macular edema using neutrosophic sets and graph algorithms. *IEEE Transactions on Biomedical Engineering 65*, 5 (May 2018), 989–1001.

[20] SALAMA, A., ABD EL FATTAH, A., EL-GHAREEB, H., AND M MANIE, A. Design and implementation of neutrosophic data operations using object oriented programming. *International Journal of Computer Application 4* (10 2014), 163–175.

[21] SALAMA, A., EL-GHAREEB, H. A., MANIE, A. M., AND SMARANDACHE, F. Introduction to develop some software programs for dealing with neutrosophic sets. *Neutrosophic Sets and Systems* (2014), 51.

[22] SALAMA, A., MANIE, A., AND LOTFY, M. Utilizing neutrosophic set in social network analysis e-learning systems. *International Journal of Information Science and Intelligent System 3* (2014), 61–72.

[23] SHAN, J., CHENG, H., AND WANG, Y. A novel segmentation method for breast ultrasound images based on neutrosophic l-means clustering. *Medical physics 39*, 9 (2012), 5669–5682.

[24] SHAW, M. Architectural issues in software reuse: It's not just the functionality, it's the packaging. *SIGSOFT Softw. Eng. Notes 20*, SI (Aug. 1995), 3–6.

[25] YE, J., AND CUI, W. Neutrosophic state feedback design method for siso neutrosophic linear systems. *Cognitive Systems Research 52* (2018), 1056 – 1065.

[26] ZHANG, H.-Y., WANG, J.-Q., AND CHEN, X.-H. Interval neutrosophic sets and their application in multicriteria decision making problems. *The Scientific World Journal 2014* (2014).

Received: December 15, 2018. Accepted: March 13, 2019.