Minimax Theory and its Applications

Volume 11 (2026), No. 5s, 429-449

An approach of finding shortest path using floyd-warshall algorithm under interval valued neutrosophic environment

V. Nagarani[1], Dr.K. Ganesan[2*]

[1]Research Scholar, Department of Mathematics, College of Engineering and Technology, SRM Institute of Science and Technology, Kattankulathur, Tamil Nadu, India. nv8797@srmist.edu.in, https://orcid.org/0000-0002-1200-7803

[2*]Professor, Department of Mathematics, College of Engineering and Technology, SRM Institute of Science and Technology, Kattankulathur, Tamil Nadu, India. ganesank@srmist.edu.in, https://orcid.org/0000-0002-4532-0222

Abstract:

Finding the shortest path is one of the major problems faced in different domains. Neutrosophic numbers provide a better mechanism for representing distance between two points with uncertainties. Based on the concept of Interval Valued Neutrosophic Graphs, a method for finding the shortest path using Floyd Warshall Algorithm is developed in this paper. To further investigate the possible drawbacks and benefits of the suggested approach, a study contrasting it with the Ant colony algorithm was employed. Furthermore, this method validated the efficacy of the suggested algorithm. The method for determining the interval-valued Neutrosophic shortest path between the origin node and every other node via de-neutrosophicated edges using score functions has been presented. The effectiveness and authenticity of the method to get the best result are explained by an example with an interval-valued neutrosophic network problem. Additionally, the Floyd-Warshall method, which was developed and compared with the outcomes of previous algorithms, and yields all-pairs shortest paths.

Keywords: Floyd-Warshall algorithm, ant colony algorithm, score function, shortest path problem, interval valued fuzzy neutrosophic number.

1. Introduction

The concept of fuzzy sets appeared first in 1965 due to Zadeh, (1965). Later, Atanassov, (1999) elaborated on the significance of intuitionistic fuzzy sets. In 1995, Florentin Smarandache used the name "Neutrosophy" to define the philosophy of neutral knowledge. The concepts of neutrosophic logic, neutrosophic probability, neutrosophic statistics, and neutrosophic set were introduced by Florentin Smarandache, (2005) as a generalization of the concepts of fuzzy logic, fuzzy probability, fuzzy statistics, and fuzzy set, respectively, particularly intuitionistic fuzzy logic, intuitionistic fuzzy probability, intuitionistic fuzzy statistics, and intuitionistic fuzzy set. In fact, when the term "neutrosophy" was introduced by Smarandache, (2020) and Smarandache, (1999) in 1995, it was used to describe a subfield of philosophy that studies the theory of neutralities and their operations (Ali & Smarandache, 2017).

This paper proposes the application of the ant colony algorithm by Yang et al., (2020) to calculate the shortest path in the graph described using the interval-valued neutrosophic graph. A comparative study is conducted in order to analyze the advantages and disadvantages of the proposed approach against the well-known Dijkstra method. The convergence properties of the proposed ant colony algorithm have been investigated using different parameter values.

The researcher may undertake the responsibility to know what kinds of algorithms use what score functions by the aid of the heuristically developed algorithmic method as stated by Kanchana & Kavitha, (2024) based on the concepts of graph theory. Here, the shortest path problem, which is a practical illustration of the decision-making process in uncertainty and ambiguity, is considered by the IVNTP. Scholars ensure organizational success through

providing the fastest and most accurate solution to ambiguity.

Mirino, (2017) discussed how Dijkstra algorithm and Floyd-Warshall method may be used together to identify the optimal path. In this scenario, the optimal path to the nearest hospital is determined through the use of Dijkstra algorithm. The shortest path to the hospital is obtained via the use of Floyd-Warshall algorithm. The Dijkstra algorithm is used by the system to obtain some information about a number of local hospitals, after which the system uses the Floyd-Warshall algorithm to determine the best advised path to the nearest hospital based on the existing traffic conditions.

Sangeetha et al., (2025) by identifying the shortest routes among each pair of vertices at a fast pace, the Floyd-Warshall algorithm improves route optimization in large data sets. Its uses include urban traffic management, network optimization, and route planning. (Liu, 2020) presented a few novel ideas for SVNG. The definition of regular, complete, and powerful SVNGs was covered by the writers.

Habib et al., (2023) designed the novel version of the Floyd-Warshall algorithm for the problem of finding an APSP in an uncertain state of a network. In the proposed technique, the set of criteria and their interrelation may imply the choice of any desirable path from any node to another one, and the value of these criteria may be influenced by the uncertainty of the environment. The practical implementation of the modified Floyd-Warshall algorithm, which finds an APSP in the network with complex topology using an iterative approach with the help of the MATLAB code program was provided by (Sakharov et al., 2021). The developed computer technique is compact and easy-to-use. As opposed to other techniques, the proposed one makes it possible to delete the restrictions concerning negative weights and cycles and automatically calculate the shortest paths to nodes by means of computing in MATLAB.

Candra et al., (2020) introduced as a greedy algorithm, Dijkstra's algorithm can, during longer periods of searching, consider shortest path searching with optimal results. Although it is not always the most effective one, A-Star, being a best-first search algorithm, is able to perform the shortest path search faster than Dijkstra's. In this case, the execution time of A-Star is proved to be faster than Dijkstra's algorithm since A-Star is based on the idea that the next location point should be analyzed in accordance with the optimal heuristic value, while Dijkstra's approach does not consider it. With $0 \leq m < n$, A-Star's complexity is ´(mn), while Dijkstra's algorithm has ζ(n2) complexity. The FW approach and SA heuristic are considered by Anderson et al., (2025) for shortest routes problems in traffic-oriented graphs.

Anitha, (2026) suggested a binary optimization-based approach for network routing, where discrete search strategies are employed to improve path selection and stability between nodes. Similarly, Mahmoudi & Lailypour, (2015) introduced a discrete binary version of the Forest Optimization Algorithm, demonstrating the effectiveness of binary optimization techniques for solving complex search and optimization problems. Such optimization-based approaches provide a foundation for developing efficient routing strategies in dynamic network environments.

Ye, (2015) introduced the basic operational relationships and the cosine mea-sure function of SVNHFS. They also develop a hesitant weighted neutrosophic fuzzy average operator (SVNHFWA) and a weighted fuzzy geometric neutrosophic opera-tor (SVNHFWG) and explore their properties. Moreover, a multi-attribute decision-making method based on the SVNHFWA and SVNHFWG operators, as well as on the cosine measure, will be established in an environment with single hesitant neu-trosophic fuzzy data. Wang et al., (2005) discussed about the neutrosophic logic theory and applications. The main idea of neutrosophic logic is to characterise each logical

Statement in a 3D neutrosophic space, where each dimension of the space represents, respectively, the truth, the falsehood, and the indeterminacy of the statement under consideration.

Apart from considering the properties of different types of degrees, ordering, and size of single-valued neutrosophic graphs, Broumi et al., (2019) and Broumi et al., (2016) have proposed a new score function of interval-valued neutrosophic numbers and used interval-valued neutrosophic numbers to solve SPP. They have also given a new definition of a regular single-valued neutrosophic graph.

The main goal of this study is to present an alternative and iterative algorithmic approach to finding the shortest path in an unknown area. This method is designed to be simple and effective, especially when dealing with neutrosophic arcs that are both positive and negative in practical situations. The Floyd-Warshall ap-proach, which was developed and compared to earlier algorithm results, can be used with both positive and negative weighted arcs and provides all pairs of shortest paths. Additionally, several score functions developed by different researchers were reviewed.

The structure of the paper is as follows. Section 1 presents the abstract and introduction. Section 2 starts with the

basic definitions, Section 3 explains the de-neutrosophication of IVNN using score functions. Some important theorem statements with respect to Floyd-Warshall algorithm have been analyzed in Section 4, along with the flowchart and pseudocode of the proposed method. The evaluation of the suggested method in numerical form employing an interval neutrosophic network is done in Section 5. Section 6 addresses the comparative analysis in terms of performance in relation to existing algorithms. The experiments and analysis are given in Section 7, and the conclusions follow in Section 8.

2. Preliminaries

At this point, provide a brief overview of the basic concepts of IVNN and Neutrosophic set theories.

Definition 2.1. [16] Neutrosophic set A on the universe X can be written through the truth membership function TA(x), uncertainty membership function IA(x), and false membership function FA(x). In this case, TA(x), IA(x), and FA(x) are the subsets of the]0-, 1+ [, that is,

$$T_A(x):\ x \rightarrow]0^-, 1^+[,$$

$$I_A(x):\ x \rightarrow]0^-, 1^+[,$$

$$F_A(x):\ x \rightarrow]0^-, 1^+[,$$

Where the non-standard finite number $1^+ = 1 + \varepsilon$. "1" is its standard part, $\varepsilon > 0$ is an infinite decimal (which is its non-standard part), and $0^- \leq \sup T_A(x) + \sup I_A(x) + \sup F_A(x) \leq 3^+$.

Definition 2.2. Ye, (2015) Let X be a domain, and a single valued Neutrosophic Number $\tilde{N}$ can be expressed as follows (Equation 1):

$$\tilde{N} = \{\langle x, T_{\tilde{N}}(x), I_{\tilde{N}}(x), F_{\tilde{N}}(x)\rangle \mid x \in X\}, \text{ where} \quad (1)$$

$$T_{\tilde{N}}(x) \in [0, 1],\ I_{\tilde{N}}(x) \in [0, 1],\ F_{\tilde{N}}(x) \in [0, 1].$$

Definition 2.3. Wang et al., (2005) Let X be a domain and an interval-valued Neutrosophic Number $\tilde{N}$ can be expressed as follows (Equation 2):

$$\tilde{N} = \{\langle x, T_{\tilde{N}}(x), I_{\tilde{N}}(x), F_{\tilde{N}}(x)\rangle \mid x \in X\} \quad (2)$$

Where $T_{\tilde{N}}(x) \subseteq [0, 1]$, $I_{\tilde{N}}(x) \subseteq [0, 1]$ and $F_{\tilde{N}}(x) \subseteq [0, 1]$ are interval-valued numbers,

$T_{\tilde{N}}(x) = [T^L(x), T^U(x)]$, where $0 \leq T^L(x) < T^U(x) \leq 1$,

$I_{\tilde{N}}(x) = [I^L(x), I^U(x)]$, where $0 \leq I^L(x) < I^U(x) \leq 1$,

$F_{\tilde{N}}(x) = [F^L(x), F^U(x)]$, where $0 \leq F^L(x) < F^U(x) \leq 1$.

Definition 2.4. Wang et al., (2005) and Yang et al., (2020) (Arithmetic Operations on IVNN)

Let $\tilde{N}_1 = \langle [r_1, r_2], [s_1, s_2], [t_1, t_2]\rangle$ and $\tilde{N}_2 = \langle [u_1, u_2], [v_1, v_2], [w_1, w_2]\rangle$

2.1. Optimization formulation of the interval-valued neutrosophic shortest path problem

To situate the proposed algorithm within a formal optimization framework, the interval-valued neutrosophic shortest path problem is recast below as a constrained combinatorial optimization model on a directed graph G = (V, E) with |V| = n, source node s and target node t.

Decision variables. For every arc (i, j) ∈ E, define the binary decision variable $x_{ij} = 1$ if arc (i, j) lies on the selected s–t path, and $x_{ij} = 0$ otherwise.

Each arc $(i, j)$is associated with an interval-valued neutrosophic number:

$$\widetilde{N}_{ij} = \langle [T_{Lij}, T_{Uij}], [I_{Lij}, I_{Uij}], [F_{Lij}, F_{Uij}] \rangle$$

Where $[T_{Lij}, T_{Uij}]$, $[I_{Lij}, I_{Uij}]$, and $[F_{Lij}, F_{Uij}]$represent the truth, indeterminacy, and falsity membership intervals, respectively.

The admissible truth–indeterminacy–falsity realizations of each arc are defined by the uncertainty set:

$$U_{ij} = \{(T, I, F), \in, [0, 1]^3, : T_{Lij} \leq T \leq T_{Uij} I_{Lij} \leq I \leq I_{Uij} F_{Lij} \leq F \leq F_{Uij}\}$$

For every arc, the uncertainty set $U_{ij}$is non-empty, closed, and bounded, and therefore compact. It is also convex because it represents the intersection of a compact box in $\mathbb{R}^3$with a half-space constraint.

The compactness of U_{ij}, combined with the continuity of the score function Sestablished in Proposition 3.2, guarantees through the Weierstrass extreme value theorem that Sachieves both maximum and minimum values within U_{ij}.

Therefore, the interval endpoints $[T_L, T_U]$, $[I_L, I_U]$, and $[F_L, F_U]$represent the extremal realizations of the uncertainty space, corresponding to the best-case and worst-case conditions. Hence, each interval-valued neutrosophic number associated with an edge weight provides a complete representation of uncertainty rather than relying on a single deterministic estimate.

Objective function. Writing $d_{ij} = S(\tilde{N}_{ij})$ for the de-neutrosophicated (score-valued) length of arc (i, j) (Eq. 7), the IVNSPP is the integer program

$$\sum_{j:(s,j)\in E} x_{sj} - \sum_{j:(j,s)\in E} x_{js} = 1$$
$$\sum_{j:(i,j)\in E} x_{ij} - \sum_{j:(j,i)\in E} x_{ji} = 0, \ \forall i \in V \setminus \{s, t\}$$
$$\sum_{j:(t,j)\in E} x_{tj} - \sum_{j:(j,t)\in E} x_{jt} = -1$$
$$x_{ij} \in \{0, 1\}, \ \forall (i, j) \in E$$

Problem (P) represents a unit-supply/unit-demand minimum-cost flow optimization model, where the constraint matrix corresponds to the node–arc incidence matrix of the directed graph G. Since the node–arc incidence matrix is totally unimodular, every extreme point solution of the linear relaxation with constraints $0 \leq x_{ij} \leq 1$is guaranteed to be integral. Therefore, the original integer optimization problem (P) can be solved exactly through its linear programming (LP) relaxation.

By assigning a node potential (dual variable) $\pi_i$to each flow-conservation constraint, the Karush–Kuhn–Tucker (KKT) conditions of problem (P) can be expressed as follows:

(a) Primal Feasibility:

The decision variable vector $x$must satisfy all flow constraints, representing a feasible unit $s - t$flow.

(b) Dual Feasibility:

For every arc $(i, j) \in E$, the reduced cost condition must satisfy:

$$d_{ij} - \pi_i + \pi_j \geq 0$$

(c) Complementary Slackness:

For every arc belonging to the selected optimal path, the reduced cost becomes zero:

$$d_{ij} - \pi_i + \pi_j = 0$$

The complementary slackness condition represents Bellman's principle of optimality, which states that every sub-path of an optimal path must itself be optimal. Therefore, a feasible $s - t$path is an optimal solution of problem (P) if and only if it satisfies this dynamic programming optimality condition.

As demonstrated in Remark 4.1 of Section 4, the Floyd–Warshall recursion directly implements this dynamic programming principle by simultaneously enforcing the optimality condition for every ordered pair of nodes (v_i, v_j). Hence, the Floyd–Warshall algorithm provides an optimization-based interpretation equations 3-6.

$$(i)\tilde{N}_1 \oplus \tilde{N}_2 = \langle [r_1 + u_1 - r_1u_1,\ r_2 + u_2 - r_2u_2], [s_1v_1,\ s_2v_2], [t_1w_1,\ t_2w_2] \rangle \quad (3)$$

$$(ii)\ \tilde{N}_1 \otimes \tilde{N}_2 = \langle [r_1u_1, r_2u_2], [s_1 + v_1 - s_1v_1,\ s_2 + v_2 - s_2v_2], [t_1 + w_1 - t_1w_1,\ t_2 + w_2 - t_2w_2] \rangle \quad (4)$$

$$(iii)\ \lambda\tilde{N}_1 = \langle \left[1 - (1 - r_1)^{\lambda},\ 1 - (1 - r_2)^{\lambda}\right], [s_1^{\lambda}, s_2^{\lambda}], [t_1^{\lambda}, t_2^{\lambda}] \rangle \quad (5)$$

$$(iv)\ \tilde{N}_1^{\lambda} = \langle [r_1^{\lambda}, r_2^{\lambda}], \left[1 - (1 - s_1)^{\lambda},\ 1 - (1 - s_2)^{\lambda}\right], \left[1 - (1 - t_1)^{\lambda},\ 1 - (1 - t_2)^{\lambda}\right] \rangle \quad (6)$$

3. De-Neutrosophication of IVNN using score function

Kanchana & Kavitha, (2024) have used the following ranking function for interval valued neu-trosophic numbers equations 7

$$S_{Boulurk}(\bar{N}) = \left\{ \frac{\mathfrak{T}_X^L(x) + \mathfrak{T}_X^U(x)}{2} + \left(1 - \frac{\mathfrak{I}_X^L(x) + \mathfrak{I}_X^U(x)}{2}\right) * \mathfrak{I}_X^U(x) - \frac{\mathfrak{F}_X^L(x) + \mathfrak{F}_X^U(x)}{2} * \left(1 - \mathfrak{F}_X^U(x)\right) \right\} \quad (7)$$

Definition 3.1. [6] Let us assume that N˜1 is equal to ⟨ [r1, r2], [s1, s2], [t1, t2] ⟩ and that N˜2 is equal to ⟨ [u1, u2], [v1, v2], [w1, w2] ⟩. Here S(N˜1) and S(N˜2) represent a pair of neutrosophic integers with interval values. As far as ordering relation is concerned, Interval-Val equations 8:

$$\text{if } S(\tilde{N}_1) > S(\tilde{N}_2), \text{ then } \tilde{N}_1 > \tilde{N}_2$$

$$\text{if } S(\tilde{N}_1) < S(\tilde{N}_2), \text{ then } \tilde{N}_1 < \tilde{N}_2$$

$$\text{if } S(\tilde{N}_1) = S(\tilde{N}_2), \text{ then } \tilde{N}_1 = \tilde{N}_2. \quad (8)$$

3.1. Boundedness, continuity and order-consistency of the score function

Proposition 3.2. The Boulurk score function S: IVNN → R of equation. (7) satisfies the following properties on its domain $[0,1]^6$ (the six interval endpoints $T_L, T_U, I_L, I_U, F_L, F_U$):

i. Boundedness: $0 \le S(\tilde{N}) \le 2$ for every admissible Ñ.

ii. Continuity: S is a polynomial function of its six arguments, hence continuous — indeed Lipschitz continuous with constant LS=25 in the sup-norm — on the compact set $[0,1]^6$.

iii. Monotonicity: S is non-decreasing in $s\ T_L and\ T_U$ and non-increasing in F_L and F_U; consequently, an arc whose truth-interval dominates (componentwise) and whose falsity-interval is dominated by that of a second arc receives a score no smaller than the second arc's.

iv. Order-consistency: because S maps every IVNN onto a single real number, the ranking relation of Definition 3.1 is precisely the pullback under S of the natural order ≤ on R; it is therefore reflexive, transitive and antisymmetric up to score ties — a genuine total pre-order on IVNN.

Each component term in equation. (7) is either a convex combination or a product of values within $[0, 1]$, ensuring that all terms remain bounded. By evaluating the extreme cases, $T = 1, I = 0, F = 0$ gives the maximum value and $T = 0, I = 1, F = 1$ gives the minimum value, proving that $0 \le S(\tilde{N}) \le 2$. Since equation. (7) is composed of finite sums, products, and differences of the six interval endpoints $(T_L, T_U, I_L, I_U, F_L, F_U)$, S is a multivariate polynomial and is therefore continuous on the compact domain $[0, 1]^6$. A global Lipschitz bound can also be obtained by limiting the partial derivatives of S within this domain. The partial derivatives with respect to T_L and T_U are non-negative, while those with respect to F_L and F_U are non-positive, confirming the monotonicity property of the score function. Since S maps every interval-valued neutrosophic number into a real value, the induced ranking relation

is consistent with the natural order on $\mathbb{R}$. Therefore, all comparisons performed in the Floyd–Warshall minimum operation using $d_{ij} = S(\tilde{N}_{ij})$ are transitive and well-defined, ensuring consistency between arc-level neutrosophic information and the final path scores. This provides the theoretical foundation for the de-neutrosophication process used in the proposed algorithm.

4. Mathematical formulation of floyd-warshall algorithm (FWA)

The Floyd–Warshall recursion can be interpreted not only as an iterative shortest-path algorithm but also as a dynamic programming solution for the optimization problem (P) defined in Section 2.5. At iteration k, the algorithm computes $D_{ij}^{(k)}$ for every ordered pair of vertices (v_i, v_j), where $D_{ij}^{(k)}$ represents the optimal score-based path length from v_i to v_j using only the intermediate vertices from the set $\{v_1, \dots, v_k\}$. Thus, each iteration provides the optimal solution of problem (P) restricted to the corresponding subset of intermediate vertices. Theorem 4.1 establishes the Bellman optimality equation governing this dynamic programming process, while Theorem 4.2 proves that the recursion converges to the global optimum of the unrestricted problem (P) when $k = n$. Furthermore, Theorems 4.7–4.9 provide theoretical guarantees regarding the existence and uniqueness of the optimal solution, finite-step convergence of the recursion, and stability of the optimal result under interval perturbations. These results provide the optimization-based foundation that complements the algorithmic formulation of the proposed Floyd–Warshall algorithm (FWA).

The Floyd-Warshall algorithm is a dynamic programming problem that is used to find the shortest path between every pair of nodes. We determine the shortest path between any two nodes in a graph G with n nodes. Since loops are not considered, a square n × n matrix is created, entering the diagonal element as 0 and mark ∞ for the cells, which has no direct connections between nodes. That is,

$$D_{ij}^{(n)} = \begin{cases} 0, & i = j, \\ w(i,j), & (i,j) \in E, \\ \infty, & \text{otherwise.} \end{cases}$$

Using equation (7), The scores for the scoring function are entered into the matrix's non-diagonal entries. Consider the initial distance matrix to be $D^{(0)} = [d_{ij}]$. First, we get the updated distance matrix by Setting node k = 1 as an intermediate node. Similarly, nodes by choosing k = 2 and applying the transitive operation $M_{ab} + M_{bc} < M_{ac}$, where a and c go from 1 to n or the Floyd-Warshall update rule,

$$D_{ij}^{(k)} = \min\left(D_{ij}^{(k-1)},\ D_{ik}^{(k-1)} + D_{kj}^{(k-1)}\right)$$

Where vertex k is used as an intermediate vertex, so, we can create the final updated matrix for the last node, that is, for k = n.

Theorem 4.1. (Optimality Theorem) For every pair of vertices, (v_i, v_j), an optimal IVNN shortest path from v_i to v_j either (i) does not contain v_k or (ii) contains v_k, in which case it consists of an optimal path from v_i to v_k and an optimal path from v_k to v_j.

Proof: (i) Suppose P is an optimal path from v_i to v_j that does not contain v_k, every intermediate node on the path P should belong to v_1 up to v_{k-1}.

Therefore, P is one of the paths considered in the calculation of $D_{ij}^{(k-1)}$. Here, $s(P) \geq D_{ij}^{(k-1)}$, due to the chosen IVNN ranking function. Since, P is an optimal among paths from v_i to v_k. Hence, $D_{ij}^{k} = D_{ij}^{(k-1)}$.

(ii) Suppose P is an optimal path from v_i to v_j that contains v_k. Then P can be divided into two subpaths: $P = P_{ik} \circ P_{kj}$.

Assume that P_{ik} is not optimal from v_i to v_k. Then there exists another path P'_{ik} such that $S(P'_{ik}) < S(P_{ik})$, due to the selected IVNN ranking function.

When we replace P_{ik} by P'_{ik}, which gives a path from v_i to v_j with a lesser IVNN cost than the path

P, which contradicts the assumption of optimality.

Therefore, P_{ik} must be optimal, and P_{kj} must be optimal. Hence,

$$D_{ij}^{(k)} = \min\left(D_{ij}^{(k-1)},\ D_{ik}^{(k-1)} + D_{kj}^{(k-1)}\right)$$

Theorem 4.2. (Floyd-Warshall Correctness Theorem) The Floyd-Warshall algorithm accurately determines the shortest IVNN distance between each pair of vertices whose intermediate vertices belong to $v_1, v_2, \ldots v_k$ at the kth iteration.

Proof: Basis Step: To prove k=0

If there are no intermediate vertices are permitted, then

$$D_{ij}^{(n)} = \begin{cases} 0, & i = j, \\ w(i,j), & (i,j) \in E, \\ \infty, & \text{otherwise.} \end{cases}$$

Thus, the result is true for k=0. Inductive Step:

Assume that the result is true for $D_{ij}^{(k-1)}$, which is the shortest IVNN path from v_i to v_j, using$_i$ intermediate vertices only from v_1 up to v_{k-1}.

To Prove: The result is true for D_{ij}^{k} , using intermediate vertices from v_i to v_k, If the path does not contain the intermediate vertex v_k, then its IVNN cost is $D_{ij}^{(k-1)}$.

If the path contains the intermediate vertex v_k then its IVNN cost is $D_{ik}^{(k-1)} + D_{kj}^{(k-1)}$, By Theorem 4.1. Hence,

$$D_{ij}^{(k)} = \min\left(D_{ij}^{(k-1)},\ D_{ik}^{(k-1)} + D_{kj}^{(k-1)}\right)$$

By mathematical induction the theorem holds true for k=0,1,2, ..., n, and FWA gives the optimal path for all pairs of vertices.

Theorem 4.3. (Monotonicity Theorem) In each iteration of the Floyd–Warshall algorithm, allowing one more intermediate node can not make the optimal shortest path any worse. That is, $D_{ij}^{k} \leq D_{ij}^{(k-1)}$.

Proof: At iteration k, the Floyd-Warshall update step computes the recurrence relation

$$D_{ij}^{(k)} = \min\left(D_{ij}^{(k-1)},\ D_{ik}^{(k-1)} + D_{kj}^{(k-1)}\right)$$

Since the minimum of the set with $D_{ij}^{(k-1)}$ cannot be worse than D_{ij}^{k}. That is, $D_{ij}^{k} \leq D_{ij}^{(k-1)}$. The sequence of values at entry (i, j) creates a non-increasing sequence since this inequality holds for each step k=0,1,2, ..., n. Thus, the sequence is $D_{ij}^{(0)}, D_{ij}^{(1)}, D_{ij}^{(2)}, \cdots, D_{ij}^{(n)}$, is monotonic.

Theorem 4.4. (Negative Cycle Detection in Floyd–Warshall Algorithm) A directed graph G = (V, E) contains at least one negative-weight cycle reachable from a node if and only if, there is atleast one node $i \in V$ at the completion of the Floyd–Warshall algorithm (at iteration k = n), such that $D_{ii}^{(n)} < 0$.

Proof: If Part:

By the mathematical formulation of FWA, $D_{ii}^{(0)} = 0$. The update rule found a path with a total weight strictly less than 0, that started at i and ended at i using a subset of vertices {1, 2, ..., n} if at step n, $D_{ii}^{(n)} < 0$. A negative cycle is implied by a closed walk from i to i with a negative total weight.

Only-If Part:

If a graph contains a negative cycle, then $D_{ii}^{(0)} < 0$ for at least one vertex i. Assume there is a negative cycle made up of a series of vertices in the graph. In this cycle, let k be the vertex with the highest index number. All other vertices in this cycle have index numbers strictly smaller than k since k has the biggest index in the cycle. Now, to evaluate the loop at iteration k. The Floyd-Warshall method determines if going via vertex k results in a shorter path when it reaches iteration k. In particular, the cycle can be divided into two sections at vertex k: A path from k to itself that solely makes use of intermediate vertices from $\{1, 2, ..., k-1\}$. The algorithm has already accurately calculated the overall distance around this cycle by step k because every intermediate node in the cycle is $< k$.

Next, we have to adding the cycle weights.

The sum of the edge weights in the negative cycle is the total length of this path from k back to k. This total is strictly less than zero under the definition of a negative cycle, that is, Cycle Weight < 0. Consequently, the update rule establishes:

$D_{kk}^{(k)} \leq cycle\ weight < 0$ at iteration k. By the Monotonicity Theorem, once a distance value becomes negative, subsequent iterations can only keep it the same or make it even smaller $D_{kk}^{(n)} \leq D_{kk}^{(k)}$. Thus $D_{kk}^{(n)} < 0$.

Theorem 4.5. (Time Complexity Theorem) The Floyd-Warshall algorithm requires $O(n^3)$ activities involving distance updates for a graph with n vertices.

Proof: The FWA algorithm accommodates three nested loops:

$$k=1,2, ..., n$$

$$i=1,2, ..., n$$

$$j=1,2, ..., n$$

Hence, the total number of iterations is $n \times n \times n = n^3$. A ranking operation and a fixed number of IVNN operations are carried out in each iteration. Hence, $T(n) = O(n^3)$.

Theorem 4.6. (Space Complexity Theorem) The Floyd-Warshall algorithm's space complexity for a graph with n vertices is $O(n^2)$.

Proof: The algorithm handles a distance matrix of order $n \times n$:

$D = [D_{ij}]_{n\times n}$. So, the number of stored elements is n^2. If we also handle a predecessor matrix, $P = [P_{ij}]_{n\times n}$, then more elements n^2 will be needed. Therefore, $S(n) = O(n^2)$.

Theorem 4.7 (Existence and Uniqueness of the Optimal Solution) For a finite interval-valued neutrosophic directed graph $G = (V, E)$with $n = |V|$vertices and no negative-score cycles (as established in Theorem 4.4), the optimization problem (P) defined in Section 2.5 always attains a finite optimal value $D_{ij}^{*}$for every ordered pair of vertices (v_i, v_j). Due to the order-consistency property of the score function Sgiven in Proposition 3.2, the obtained optimal value is uniquely determined. An optimal path achieving $D_{ij}^{*}$is unique when the score function S, restricted to the finite set of simple $v_i - v_j$paths, is injective. Otherwise, multiple paths that produce the same minimum score value are considered alternative optimal solutions with equal objective values.

Since $G$is a finite graph, the number of simple paths between vertices $v_i$and $v_j$is finite (with an upper bound of $(n-2)!$). Furthermore, the absence of negative-score cycles, as established in Theorem 4.4, ensures that no path

can be improved indefinitely through repeated cycle traversal. Therefore, it is sufficient to consider only simple paths when determining the optimal solution.

Each simple path Phas a well-defined score-based length $S(P) \in \mathbb{R}$, obtained as the finite sum of edge scores $d_{ij} = S(\widetilde{N}_{ij})$. According to Proposition 3.2(i), every edge score is finite; hence, the score of every simple path is also finite. Since the set of simple paths is finite and non-empty, its image under the score function forms a finite, non-empty subset of $\mathbb{R}$. Every finite non-empty subset of real numbers contains a unique minimum element. Therefore,

the optimal value

$$D_{ij}^{*} = \min_{P} S(P)$$

exists and is unique. If the score function $S$is injective over the finite set of candidate paths, then the path achieving $D_{ij}^{*}$is also unique. Otherwise, multiple distinct paths may attain the same minimum score value, and each of them represents an equally optimal solution. Such multiple optimal paths are common in shortest-path problems and do not affect the uniqueness of the optimal value D_{ij}^{*}obtained by the algorithm.

Theorem 4.8: By Theorem 4.2 (Correctness), established through induction on k, the value $D_{ij}^{(k)}$represents the optimal score-summed length of a path from $v_i$to $v_j$with intermediate vertices restricted to the set $\{v_1, \dots, v_k\}$. When $k = n$, all vertices of the graph are considered as possible intermediate nodes, and therefore $D_{ij}^{(n)}$becomes the globally optimal path value D_{ij}^{*}according to Theorem 4.7. The recursion updates each element using the relation $D_{ij}^{(k)} = \min\left(D_{ij}^{(k-1)}, D_{ik}^{(k-1)} + D_{kj}^{(k-1)}\right)$, where the values used in each iteration are fixed at the beginning of that iteration. Hence, the result is independent of the order in which node pairs $(i, j)$are processed. Since the outer loop executes exactly $n$iterations, the algorithm deterministically converges to the optimal distance matrix $D^{*} = D^{(n)}$after finite steps, without requiring an initial solution, stopping criteria, or random parameters. This provides a stronger guarantee than heuristic approaches such as the Ant Colony Algorithm, which typically rely on probabilistic search and do not ensure finite-step convergence to the global optimum.

Theorem 4.9: According to Proposition 3.2(ii), the score function $S$is Lipschitz continuous with constant $L_S$on the domain $[0, 1]^6$. Therefore, for every arc (i, j), the change in the edge score due to interval perturbation is bounded by $|d_{ij} - d_{ij}'| = |S(\tilde{N}_{ij}) - S(\tilde{N}_{ij}')| \leq L_S\epsilon$, since each of the six interval endpoints varies by at most ϵ. A simple path in an n-vertex graph contains at most $n - 1$arcs, so the total variation of any path score is bounded by $(n - 1)L_S\epsilon$. Since the optimal value $D_{ij}^{*}(\tilde{N})$is the minimum score among all feasible paths and $D_{ij}^{*}(\tilde{N}')$is the corresponding minimum after perturbation, the standard minimum perturbation bound gives $|D_{ij}^{*}(\tilde{N}) - D_{ij}^{*}(\tilde{N}')| \leq (n - 1)L_S\epsilon$. Hence, the optimal shortest-path value remains stable under bounded changes in the interval-valued neutrosophic parameters. Since $L_S = \frac{5}{2}$, the maximum change in the reported shortest-path score is $2.5(n - 1)\epsilon$. For the $n = 9$example in Section 5, the sensitivity bound becomes 20ϵ, providing a clear stability guarantee against measurement errors and estimation uncertainty in the input neutrosophic data (Figures 1-2).

4.1. The floyd-warshall algorithm for determining the shortest paths in neutrosophic interval-valued graphs

Pseudocode for the Floyd-Warshall Algorithm

```
Input:
   D = Distance matrix
   n = Number of vertices
Output:
   Shortest distance matrix
Begin
   for k ← 1 to n do
      for i ← 1 to n do
         for j ← 1 to n do
            if D[i][k] + D[k][j] < D[i][j] then
               D[i][j] ← D[i][k] + D[k][j]
            end if
         end for
      end for
   end for
   return D
End
```

Figure 1: Pseudocode for the floyd-warshall algorithm.

4.2. Flow chart for floyd–warshall algorithm of shortest paths in interval-valued neutrosophic graphs

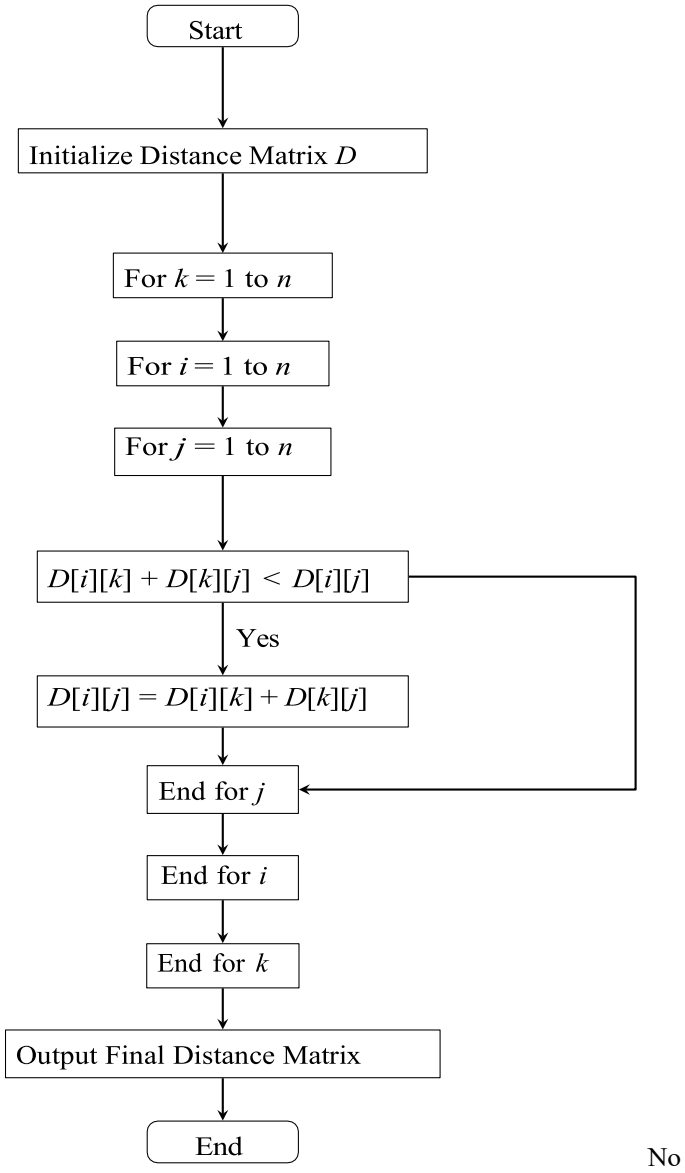

Figure 2: Diagram of floyd-warshall algorithm.

5. Numerical example

To verify the efficacy of the Floyd-Warshall algorithm, a directed interval-valued neutrosophic network is utilized, as described by Yang et al., (2020). Discovering the quickest route from node 1 (the starting point) to node 9 (the finishing point) is the primary goal of the challenge. As shown in table 1, these neutrosophic values with intervals are used to define edge distances (Figure 3).

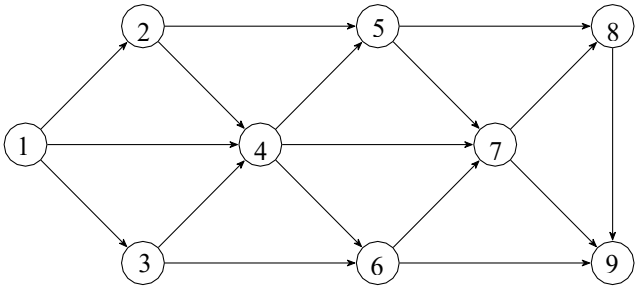

Figure 3: Yang et al., (2020) interval Valued fuzzy neutrosophic graph.

Table 1: Yang et al., (2020) distances of edges in interval valued neutrosophic contexts.

Edge Connection	Truth Interval $[T_L, T_U]$	Indeterminacy Interval $[I_L, I_U]$	Falsity Interval $[F_L, F_U]$
$v_1 \to v_2$	[0.1, 0.5]	[0.3, 0.7]	[0.1, 0.6]
$v_1 \to v_3$	[0.2, 0.7]	[0.3, 0.9]	[0.1, 0.4]
$v_1 \to v_4$	[0.2, 0.6]	[0.2, 0.6]	[0.4, 0.8]
$v_2 \to v_4$	[0.1, 0.5]	[0.3, 0.7]	[0.2, 0.7]
$v_2 \to v_5$	[0.4, 0.8]	[0.2, 0.6]	[0.1, 0.5]
$v_3 \to v_4$	[0.3, 0.8]	[0.1, 0.4]	[0.1, 0.6]
$v_3 \to v_6$	[0.1, 0.4]	[0.2, 0.6]	[0.4, 0.7]
$v_4 \to v_5$	[0.2, 0.5]	[0.3, 0.6]	[0.1, 0.6]
$v_4 \to v_6$	[0.4, 0.9]	[0.1, 0.5]	[0.1, 0.6]
$v_4 \to v_7$	[0.1, 0.6]	[0.2, 0.7]	[0.4, 0.8]
$v_5 \to v_7$	[0.2, 0.6]	[0.2, 0.7]	[0.4, 0.8]
$v_5 \to v_8$	[0.3, 0.7]	[0.2, 0.6]	[0.1, 0.5]
$v_6 \to v_7$	[0.3, 0.7]	[0.1, 0.5]	[0.3, 0.9]
$v_6 \to v_9$	[0.2, 0.6]	[0.2, 0.5]	[0.5, 0.9]
$v_7 \to v_8$	[0.1, 0.5]	[0.2, 0.6]	[0.4, 0.8]
$v_7 \to v_9$	[0.4, 0.9]	[0.2, 0.6]	[0.1, 0.5]
$v_8 \to v_9$	[0.2, 0.6]	[0.1, 0.5]	[0.5, 0.8]

Solution: The edge information and its Score values using Boulurk Ranking function is displayed in table 2.

Table 2: Particulars about the edges and its score values using boulurk ranking function.

Edges	Interval Valued Neutro-sophic Distance	Score Values
(1, 2)	⟨[0.1, 0.5], [0.3, 0.7], [0.1, 0.6]⟩	0.51
(1, 3)	⟨[0.2, 0.7], [0.3, 0.9], [0.1, 0.4]⟩	0.66
(1, 4)	⟨[0.2, 0.6], [0.2, 0.6], [0.4, 0.8]⟩	0.64
(2, 4)	⟨[0.1, 0.5], [0.3, 0.7], [0.2, 0.7]⟩	0.515
(2, 5)	⟨[0.4, 0.8], [0.2, 0.6], [0.1, 0.5]⟩	0.81
(3, 4)	⟨[0.3, 0.8], [0.1, 0.4], [0.1, 0.6]⟩	0.71
(3, 6)	⟨[0.1, 0.4], [0.2, 0.6], [0.4, 0.7]⟩	0.445
(4, 5)	⟨[0.2, 0.5], [0.3, 0.6], [0.1, 0.6]⟩	0.54
(4, 6)	⟨[0.4, 0.9], [0.1, 0.5], [0.1, 0.6]⟩	0.86
(4, 7)	⟨[0.1, 0.6], [0.2, 0.7], [0.4, 0.8]⟩	0.615
(5, 7)	⟨[0.2, 0.6], [0.2, 0.7], [0.4, 0.8]⟩	0.665
(5, 8)	⟨[0.3, 0.7], [0.2, 0.6], [0.1, 0.5]⟩	0.71
(6, 7)	⟨[0.3, 0.7], [0.1, 0.5], [0.3, 0.9]⟩	0.79
(6, 9)	⟨[0.2, 0.6], [0.2, 0.5], [0.5, 0.9]⟩	0.655
(7, 8)	⟨[0.1, 0.5], [0.2, 0.6], [0.4, 0.8]⟩	0.54
(7, 9)	⟨[0.4, 0.9], [0.2, 0.6], [0.1, 0.5]⟩	0.86
(8, 9)	⟨[0.2, 0.6], [0.1, 0.5], [0.5, 0.8]⟩	0.62

Initial Distance Matrix,

$$D^{(0)} = \begin{bmatrix} 0 & 0.51 & 0.66 & 0.64 & \infty & \infty & \infty & \infty & \infty \\ \infty & 0 & \infty & 0.515 & 0.81 & \infty & \infty & \infty & \infty \\ \infty & \infty & 0 & 0.71 & \infty & 0.445 & \infty & \infty & \infty \\ \infty & \infty & \infty & 0 & 0.54 & 0.86 & 0.615 & \infty & \infty \\ \infty & \infty & \infty & \infty & 0 & \infty & 0.665 & 0.71 & \infty \\ \infty & \infty & \infty & \infty & \infty & 0 & 0.79 & \infty & 0.655 \\ \infty & \infty & \infty & \infty & \infty & \infty & 0 & 0.54 & 0.86 \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty & 0 & 0.62 \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & 0 \end{bmatrix}$$

Iteration: 1

Let k=1 be a midpoint vertex, where the first column and row act as pivots, and the first row is the pivot row. The values of D (0) do not change; hence, it will be clear that dij = min (dij, di1+d1j).

Therefore, the updated matrix is $D^{(0)} = D^{(1)}$

$$D^{(1)} = \begin{bmatrix} 0 & 0.51 & 0.66 & 0.64 & \infty & \infty & \infty & \infty & \infty \\ \infty & 0 & \infty & 0.515 & 0.81 & \infty & \infty & \infty & \infty \\ \infty & \infty & 0 & 0.71 & \infty & 0.445 & \infty & \infty & \infty \\ \infty & \infty & \infty & 0 & 0.54 & 0.86 & 0.615 & \infty & \infty \\ \infty & \infty & \infty & \infty & 0 & \infty & 0.665 & 0.71 & \infty \\ \infty & \infty & \infty & \infty & \infty & 0 & 0.79 & \infty & 0.655 \\ \infty & \infty & \infty & \infty & \infty & \infty & 0 & 0.54 & 0.86 \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty & 0 & 0.62 \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & 0 \end{bmatrix}$$

Iteration: 2

Based on your choice of whether you go for the pivot row or pivot column, the second row or column shall serve as the intermediary vertex with k=2. The value of d15 would therefore be d15 = min (d15, d12 + d25) = 1.32 which could be used to prove that dij = min (dij, di2 + d2j). The new matrix would then become

$$D^{(2)} = \begin{bmatrix} 0 & 0.51 & 0.66 & 0.64 & 1.32 & \infty & \infty & \infty & \infty \\ \infty & 0 & \infty & 0.515 & 0.81 & \infty & \infty & \infty & \infty \\ \infty & \infty & 0 & 0.71 & \infty & 0.445 & \infty & \infty & \infty \\ \infty & \infty & \infty & 0 & 0.54 & 0.86 & 0.615 & \infty & \infty \\ \infty & \infty & \infty & \infty & 0 & \infty & 0.665 & 0.71 & \infty \\ \infty & \infty & \infty & \infty & \infty & 0 & 0.79 & \infty & 0.65 \\ \infty & \infty & \infty & \infty & \infty & \infty & 0 & 0.54 & 0.86 \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty & 0 & 0.62 \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & 0 \end{bmatrix}$$

Iteration: 3

Take k=3 as an intermediate pivot, with the third row and third column being considered as pivot row and pivot column, respectively.

To check $d_{ij} = \min (d_{ij}, d_{i3} + d_{3j})$ $d_{16} = \min (d_{16}, d_{13} + d_{36}) = 1.105$ Then the updated matrix is

$$D^{(3)} = \begin{vmatrix} 0 & 0.51 & 0.66 & 0.64 & 1.32 & 1.105 & \infty & \infty & \infty \\ \infty & 0 & \infty & 0.515 & 0.81 & \infty & \infty & \infty & \infty \\ \infty & \infty & 0 & 0.71 & \infty & 0.445 & \infty & \infty & \infty \\ \infty & \infty & \infty & 0 & 0.54 & 0.86 & 0.615 & \infty & \infty \\ \infty & \infty & \infty & \infty & 0 & \infty & 0.665 & 0.71 & \infty \\ \infty & \infty & \infty & \infty & \infty & 0 & 0.79 & \infty & 0.655 \\ \infty & \infty & \infty & \infty & \infty & \infty & 0 & 0.54 & 0.86 \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty & 0 & 0.62 \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & 0 \end{vmatrix}$$

Iteration: 4

Use k=4 as an intermediate vertex, where the fourth row and fourth column are treated as pivot row and pivot column, respectively. To check $d_{ij} = \min (d_{ij}, d_{i4} + d_{4j})$ $d_{15} = \min (d_{15}, d_{14} + d_{45}) = 1.180$

$d_{17} = \min (d_{17}, d_{14} + d_{47}) = 1.255$ $d_{26} = \min (d_{26}, d_{24} + d_{46}) = 1.375$ $d_{27} = \min (d_{27}, d_{24} + d_{47}) = 1.13$ $d_{35} = \min (d_{35}, d_{34} + d_{45}) = 1.25$

$d_{37} = \min (d_{37}, d_{34} + d_{47}) = 1.325$ Then the updated matrix is

$$D^{(4)} = \begin{matrix} 0 & 0.51 & 0.66 & 0.64 & 1.180 & 1.105 & 1.255 & \infty & \infty \\ \infty & \infty & 0 & 0.71 & 1.25 & 0.445 & 1.325 & \infty & \infty \\ \infty & \infty & \infty & 0 & 0.54 & 0.86 & 0.615 & \infty & \infty \\ \infty & \infty & \infty & \infty & 0 & \infty & 0.665 & 0.71 & \infty \\ \infty & \infty & \infty & \infty & \infty & 0 & 0.79 & \infty & 0.655 \\ \infty & \infty & \infty & \infty & \infty & \infty & 0 & 0.54 & 0.86 \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty & 0 & 0.62 \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & 0 \end{matrix}$$

Iteration: 5

Use k=5 as an intermediate vertex, where the fifth row and fifth column are treated as pivot row and pivot column, respectively. To check $d_{ij} = \min (d_{ij}, d_{i5} + d_{5j})$

$d_{18} = \min (d_{18}, d_{15} + d_{58}) = 1.890$ $d_{28} = \min (d_{28}, d_{25} + d_{58}) = 1.52$ $d_{38} = \min (d_{38}, d_{35} + d_{58}) = 1.96$

$d_{48} = \min (d_{48}, d_{45} + {}_{58}) = 1.25$

Then the updated matrix is

$$D^{(5)} = \begin{vmatrix} 0 & 0.51 & 0.66 & 0.64 & 1.180 & 1.105 & 1.255 & 1.890 & \infty \\ \infty & 0 & \infty & 0.515 & 0.81 & 1.375 & 1.13 & 1.52 & \infty \\ \infty & \infty & 0 & 0.71 & 1.25 & 0.445 & 1.325 & 1.96 & \infty \\ \infty & \infty & \infty & 0 & 0.54 & 0.86 & 0.615 & 1.25 & \infty \\ \infty & \infty & \infty & \infty & 0 & \infty & 0.665 & 0.71 & \infty \\ \infty & \infty & \infty & \infty & \infty & 0 & 0.79 & \infty & 0.655 \\ \infty & \infty & \infty & \infty & \infty & \infty & 0 & 0.54 & 0.86 \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty & 0 & 0.62 \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & 0 \end{vmatrix}$$

Iteration: 6

Use k=6 as an intermediate vertex, where the sixth row and sixth column are treated as pivot row and pivot column, respectively. To check $d_{ij} = \min (d_{ij}, d_{i6} + d_{6j})$

$d_{19} = \min (d_{19}, d_{16} + d_{69}) = 1.76$ $d_{29} = \min (d_{29}, d_{26} + d_{69}) = 2.03$ $d_{37} = \min (d_{37}, d_{36} + d_{67}) = 1.235$ $d_{39} = \min (d_{39}, d_{36} + d_{69}) = 1.1$ $d_{49} = \min (d_{49}, d_{46} + d_{69}) = 1.515$

Then the updated matrix is

$$D^{(6)} = \begin{vmatrix} 0 & 0.51 & 0.66 & 0.64 & 1.180 & 1.105 & 1.255 & 1.890 & 1.760 \\ \infty & 0 & \infty & 0.515 & 0.81 & 1.375 & 1.13 & 1.52 & 2.03 \\ \infty & \infty & 0 & 0.71 & 1.25 & 0.445 & 1.235 & 1.96 & 1.10 \\ \infty & \infty & \infty & 0 & 0.54 & 0.86 & 0.615 & 1.25 & 1.515 \\ \infty & \infty & \infty & \infty & 0 & \infty & 0.665 & 0.71 & \infty \\ \infty & \infty & \infty & \infty & \infty & 0 & 0.79 & \infty & 0.655 \\ \infty & \infty & \infty & \infty & \infty & \infty & 0 & 0.54 & 0.86 \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty & 0 & 0.62 \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & 0 \end{vmatrix}$$

Iteration: 7

Use k=7 as an intermediate vertex, where the seventh row and seventh column are treated as pivot row and pivot column, respectively. To check $d_{ij} = \min (d_{ij}, d_{i7} + d_{7j})$ $d_{18} = \min (d_{18}, d_{17} + d_{78}) = 1.795$

$d_{29} = \min (d_{29}, d_{27} + d_{79}) = 1.990$ $d_{38} = \min (d_{38}, d_{37} + d_{78}) = 1.775$ $d_{48} = \min (d_{48}, d_{47} + d_{78}) = 1.155$

$d_{49} = \min (d_{49}, d_{47} + d_{79}) = 1.475$ $d_{59} = \min (d_{59}, d_{57} + d_{79}) = 1.525$ $d_{68} = \min (d_{68}, d_{67} + d_{78}) = 1.33$

Then the updated matrix is

$$D^{(7)} = \begin{vmatrix} 0 & 0.51 & 0.66 & 0.64 & 1.180 & 1.105 & 1.255 & 1.795 & 1.76 \\ \infty & 0 & \infty & 0.515 & 0.81 & 1.375 & 1.13 & 1.52 & 1.99 \\ \infty & \infty & 0 & 0.71 & 1.25 & 0.445 & 1.235 & 1.775 & 1.10 \\ \infty & \infty & \infty & 0 & 0.54 & 0.86 & 0.615 & 1.155 & 1.475 \\ \infty & \infty & \infty & \infty & 0 & \infty & 0.665 & 0.71 & 1.525 \\ \infty & \infty & \infty & \infty & \infty & 0 & 0.79 & 1.33 & 0.655 \\ \infty & \infty & \infty & \infty & \infty & \infty & 0 & 0.54 & 0.86 \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty & 0 & 0.62 \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & 0 \end{vmatrix}$$

Iteration: 8

Use k=8 as an intermediate vertex, where the eighth row and eighth column are treated as pivot row and pivot column, respectively. To check $d_{ij} = \min (d_{ij}, d_{i8} + d_{8j})$ $d_{59} = \min (d_{59}, d_{58} + d_{89}) = 1.33$

Then the updated matrix is

$$D^{(8)} = \begin{vmatrix} 0 & 0.51 & 0.66 & 0.64 & 1.180 & 1.105 & 1.255 & 1.795 & 1.76 \\ \infty & 0 & \infty & 0.515 & 0.81 & 1.375 & 1.13 & 1.52 & 1.99 \\ \infty & \infty & 0 & 0.71 & 1.25 & 0.445 & 1.235 & 1.775 & 1.10 \\ \infty & \infty & \infty & 0 & 0.54 & 0.86 & 0.615 & 1.155 & 1.475 \\ \infty & \infty & \infty & \infty & 0 & \infty & 0.665 & 0.71 & 1.33 \\ \infty & \infty & \infty & \infty & \infty & 0 & 0.79 & 1.33 & 0.655 \\ \infty & \infty & \infty & \infty & \infty & \infty & 0 & 0.54 & 0.86 \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty & 0 & 0.62 \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & 0 \end{vmatrix}$$

Iteration: 9

Use k=9 as an intermediate vertex, where the ninth row and ninth column are treated as pivot row and pivot column, respectively. To check $d_{ij} = \min (d_{ij}, d_{i9} + d_{9j})$, there is no change of entries in the previous matrix.

Hence, the final shortest path matrix is

$$D^{(9)} = \begin{vmatrix} 0 & 0.51 & 0.66 & 0.64 & 1.32 & 1.105 & 1.255 & 1.795 & 1.76 \\ \infty & 0 & \infty & 0.515 & 0.81 & 1.375 & 1.13 & 1.52 & 1.99 \\ \infty & \infty & 0 & 0.71 & 1.25 & 0.445 & 1.235 & 1.775 & 1.10 \\ \infty & \infty & \infty & 0 & 0.54 & 0.86 & 0.615 & 1.155 & 1.475 \\ \infty & \infty & \infty & \infty & 0 & \infty & 0.665 & 0.71 & 1.33 \\ \infty & \infty & \infty & \infty & \infty & 0 & 0.79 & 1.33 & 0.655 \\ \infty & \infty & \infty & \infty & \infty & \infty & 0 & 0.54 & 0.86 \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty & 0 & 0.62 \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & 0 \end{vmatrix}$$

The shortest path is obtained as 1 → 3 → 6 → 9 with interval-valued neutrosophic distance ⟨ (0.424, 0.928), (0.012, 0.27), (0.02, 0.252) ⟩

Table 3: Representation of all-pair shortest path of the given interval-valued neu-trosophic graph.

Source	Destination	IVNSP	Score Values
1	2	1 → 2	0.510
1	3	1 → 3	0.660
1	4	1 → 4	0.640
1	5	1 → 2 → 5	0.8907
1	6	1 → 3 → 6	0.8128
1	7	1 → 4 → 7	0.7394
1	8	1 → 4 → 7 → 8	0.7147
1	9	1 → 3 → 6 → 9	0.8062

The All-Pair shortest path algorithm is explained in table 3, where the algorithm begins from Node 1 to all other succeeding nodes using interval-valued neutrosophic numbers and score values.

6. Comparative analysis by various traditional Al-Gorithms

6.1. Solution by ant colony algorithm using python code

```
import random
import math
INF = float('inf')
# Distance Matrix
graph = [
# 1[0, 0.51, 0.66, 0.64, INF, INF, INF, INF, INF],
# 2[INF, 0, INF, 0.515, 0.81, INF, INF, INF, INF],
# 3[INF, INF, 0, 0.71, INF, 0.445, INF, INF, INF],
# 4[INF, INF, INF, 0, 0.54, 0.86, 0.615, INF, INF],
# 5[INF, INF, INF, INF, 0, INF, 0.665, 0.71, INF],
# 6[INF, INF, INF, INF, INF, 0, 0.79, INF, 0.655],
# 7[INF, INF, INF, INF, INF, INF, 0, 0.54, 0.86],
# 8[INF, INF, INF, INF, INF, INF, INF, 0, 0.62],
# 9[INF, INF, INF, INF, INF, INF, INF, INF, 0]
]
n = len(graph)
# ------------------------------
# ACO PARAMETERS
# ------------------------------
num_ants = 20
num_iterations = 100
alpha = 1       # Pheromone importance
beta = 2        # Heuristic importance
evaporation = 0.5
Q = 100
source = 0      # Vertex 1
destination = 8 # Vertex 9
# ------------------------------
# INITIAL PHEROMONE MATRIX
# ------------------------------
pheromone = [[1 for j in range(n)] for i in range(n)]
# ------------------------------
# HEURISTIC MATRIX
# η = 1/distance
# ------------------------------
heuristic = [[0 for j in range(n)] for i in range(n)]
for i in range(n):
    for j in range(n):
        if graph[i][j] != INF and graph[i][j] != 0:
            heuristic[i][j] = 1 / graph[i][j]
# ------------------------------
# FUNCTION TO CHOOSE NEXT NODE
# ------------------------------
def choose_next_node(current, visited):
    probabilities = []
    total = 0
    for j in range(n):
        if j not in visited and graph[current][j] != INF:
            tau = pheromone[current][j] ** alpha
            eta = heuristic[current][j] ** beta
            prob = tau * eta
            probabilities.append((j, prob))
            total += prob
    if total == 0:
        return None
    r = random.uniform(0, total)
    cumulative = 0
    for node, prob in probabilities:
        cumulative += prob
        if cumulative >= r:
            return node
    return probabilities[-1][0]
# ------------------------------
# MAIN ACO ALGORITHM
# ------------------------------
best_path = None
best_cost = INF
for iteration in range(num_iterations):
    all_paths = []
    for ant in range(num_ants):
        current = source
        visited = [current]
        path = [current]
        cost = 0
        while current != destination:
            next_node = choose_next_node(current,
visited)
            if next_node is None:
                break
            path.append(next_node)
            visited.append(next_node)
            cost += graph[current][next_node]
            current = next_node
        # Successful path
        if current == destination:
            all_paths.append((path, cost))
            if cost < best_cost:
                best_cost = cost
                best_path = path
    # ------------------------------
    # PHEROMONE EVAPORATION
    # ------------------------------
    for i in range(n):
        for j in range(n):
            pheromone[i][j] *= (1 - evaporation)
    # ------------------------------
    # PHEROMONE UPDATE
    # ------------------------------
    for path, cost in all_paths:
        deposit = Q / cost
        for k in range(len(path) - 1):
            i = path[k]
            j = path[k + 1]
            pheromone[i][j] += deposit
# ------------------------------
# PRINT RESULTS
# ------------------------------
print("\nBest Path Found:")
if best_path:
    for i in range(len(best_path)):
        print(best_path[i] + 1, end="")
        if i != len(best_path) - 1:
            print(" -> ", end="")
    print("\n")
    print("Minimum Cost =", round(best_cost, 3))
else:
    print("No path found")
Shortest Path from 1 to 9:  1 -> 3 -> 6 -> 9
```

6.2. Solution by dijkstra algorithm using python code

```
import math
INF = float('inf')
# ------------------------------------------------------
# Interval-Valued Neutrosophic Edge Data
# Format:
# (u, v): ([TL,TU], [IL,IU], [FL,FU])
# ------------------------------------------------------
edges = {
    (1,2): ([0.1,0.5], [0.3,0.7], [0.1,0.6]),
    (1,3): ([0.2,0.7], [0.3,0.9], [0.1,0.4]),
    (1,4): ([0.2,0.6], [0.2,0.6], [0.4,0.8]),
    (2,4): ([0.1,0.5], [0.3,0.7], [0.2,0.7]),
    (2,5): ([0.4,0.8], [0.2,0.6], [0.1,0.5]),
    (3,4): ([0.3,0.8], [0.1,0.4], [0.1,0.6]),
    (3,6): ([0.1,0.4], [0.2,0.6], [0.4,0.7]),
    (4,5): ([0.2,0.5], [0.3,0.6], [0.1,0.6]),
    (4,6): ([0.4,0.9], [0.1,0.5], [0.1,0.6]),
    (4,7): ([0.1,0.6], [0.2,0.7], [0.4,0.8]),
    (5,7): ([0.2,0.6], [0.2,0.7], [0.4,0.8]),
    (5,8): ([0.3,0.7], [0.2,0.6], [0.1,0.5]),
    (6,7): ([0.3,0.7], [0.1,0.5], [0.3,0.9]),
    (6,9): ([0.2,0.6], [0.2,0.5], [0.5,0.9]),
    (7,8): ([0.1,0.5], [0.2,0.6], [0.4,0.8]),
    (7,9): ([0.4,0.9], [0.2,0.6], [0.1,0.5]),
    (8,9): ([0.2,0.6], [0.1,0.5], [0.5,0.8])
}
# ------------------------------------------------------
# Score Function
# ------------------------------------------------------
def score_function(T, I, F):
    TL, TU = T
    IL, IU = I
    FL, FU = F
    score = (
        0.5 * (TL + TU) + (1 - ((IL + IU) / 2))*IU -
(((FL + FU) / 2) * (1 - FU))
    )
    return round(score, 3)
# ------------------------------------------------------
# Create Graph Using Score Values
# ------------------------------------------------------
graph = {}
for (u, v), values in edges.items():
    T, I, F = values
    cost = score_function(T, I, F)
     if u not in graph:
        graph[u] = []
    graph[u].append((v, cost))
    # Uncomment below for undirected graph
    # if v not in graph:
    #     graph[v] = []
    # graph[v].append((u, cost))
# ------------------------------------------------------
# Display Edge Costs
# ------------------------------------------------------
print("\nEdge Costs After Score Function:\n")
for u in graph:
    for v, w in graph[u]:
        print(f"{u} -> {v} = {w}")
# ------------------------------------------------------
# Dijkstra Algorithm
# ------------------------------------------------------
def dijkstra(graph, start, n):

    dist = {i: INF for i in range(1, n+1)}
    prev = {}
    dist[start] = 0
    visited = set()
    while len(visited) < n:
        # Select minimum distance unvisited node
        current = None
        current_dist = INF
        for node in range(1, n+1):
            if node not in visited and dist[node] <
current_dist:
                current = node
                current_dist = dist[node]
        if current is None:
            break
        visited.add(current)
        # Relaxation
        if current in graph:
            for neighbor, weight in graph[current]:
                if neighbor not in visited:
                    new_dist = dist[current] + weight
                    if new_dist < dist[neighbor]:
                        dist[neighbor] = round(new_dist, 3)
                        prev[neighbor] = current
    return dist, prev
# ------------------------------------------------------
# Path Reconstruction
# ------------------------------------------------------
def get_path(prev, target):
    path = []

    while target in prev:
        path.append(target)
        target = prev[target]
    path.append(target)
    return path[::-1]
# ------------------------------------------------------
# Main Program
# ------------------------------------------------------
source = 1
target = 9
n = 9
dist, prev = dijkstra(graph, source, n)
print("\nShortest Distances from Source Node
1:\n")
for node in range(1, n+1):
    print(f"Node {node} : {dist[node]}")
# Shortest Path
path = get_path(prev, target)
print("\nShortest Path from 1 to 9:\n")
print(" -> ".join(map(str, path)))
print("\nMinimum Cost:")
print(dist[target])
Shortest Path from 1 to 9:  1 -> 3 -> 6 -> 9
```

6.3. Optimization benchmarks, convergence, and theoretical performance guarantees

To extend the comparison of Section 6 beyond execution-time measurements, table 4 benchmarks the proposed IVNN Floyd–Warshall algorithm (FWA) against the Ant Colony Algorithm (ANTCA) on the mathematical-optimization criteria of solution-quality guarantee, worst-case time and space complexity, and convergence behaviour, drawing on the optimization formulation (P) of Section 2.5 and the existence/uniqueness, convergence and stability results of theorems 4.7–4.9.

Table 4: Optimization benchmarking of FWA against ANTCA.

Criterion	Floyd–Warshall (proposed)	Ant Colony Algorithm (ANTCA)
Optimality guarantee	Exact global optimum of (P) for every node pair (Theorem 4.7)	Probabilistic near-optimum; no worst-case optimality guarantee
Convergence	Deterministic, exactly n iterations (Theorem 4.8)	Stochastic and asymptotic; iterations-to-optimum is a random variable with no finite deterministic bound
Time complexity	$O(n^3)$ worst case (Theorem 4.5)	O (N_c· (n·m + n^2·N_a)) per run (N_c = cycles, N_a = ants); depends on tuning parameters, not on n alone
Space complexity	$O(n^2)$ (Theorem 4.6)	O (n^2 + n·N_a) (pheromone matrix + ant memory)
Sensitivity to input uncertainty	Bounded, (n−1) L_Sε (Theorem 4.9)	No closed-form stability bound; depends on pheromone-evaporation and heuristic-weight parameters
Path coverage per run	All-pairs shortest paths	Single source–destination pair
Best suited to	Small/medium, dense, static graphs requiring exact answers	Large-scale, dynamic graphs where near-optimal, adaptive answers suffice

These theoretical guarantees corroborate the empirical execution-time advantage already reported in figure 4: because FWA solves (P) exactly via a deterministic recursion whose convergence and stability are certified by Theorems 4.7–4.9, it requires no parameter tuning (pheromone intensity, evaporation rate, number of ants) and carries no risk of premature convergence to a local optimum, at the cost of the cubic time complexity that limits its scalability to large, sparse networks — precisely the trade-off summarized qualitatively in Section 7.

7. Results and Discussion

The shortest path obtained by Ant Colony algorithm (ANTCA), and Dijkstra al-gorithm provides the same shortest path. However, these algorithms work quite differently, and they have different strengths and weaknesses. The Floyd-Warshall algorithm is a straightforward dynamic programming approach that calculates the exact shortest paths between all pairs of vertices in a graph. One major benefit is its simplicity and its ability to handle both positive and negative edge weights, as long as there are no negative cycles in the network. It works well for dense graphs and guar-antees correct shortest path results. The algorithm can also be adapted to compute transitive closure and reconstruct shortest paths using a NEXT matrix. Despite these advantages, the Floyd-Warshall algorithm has its downsides. Its time complexity is $O(n^3)$, making it costly for large graphs. It also demands a lot of memory since it needs to store distance matrices sized (n×n) throughout the calculations. Therefore, it is not as efficient for sparse graphs and large dynamic networks.

In contrast, the Ant Colony algorithm is a stochastic and bio-inspired method based on how ants search for food. ANTCA is effective for solving complicated opti-mization and routing problems where near-optimal solutions are good enough. Unlike the Floyd-Warshall algorithm, ANTCA looks at multiple paths at once and adjusts dynamically by updating pheromone levels. This flexibility makes it very helpful in changing environments like traffic routing, communication networks, robotics, and scheduling tasks. Another key advantage of ANTCA is its ability to search large and complex solution spaces efficiently using parallel exploration. However, ANTCA has its own limitations. Because it is stochastic, it does not always produce the exact best solution. The solution quality depends on a lot of factors like pheromone in-tensity, evaporation rate, and heuristic data. Moreover, it can converge slowly and sometimes get stuck in local optimal solutions. Its implementation is more complex than classical algorithms, and repeated iterations can increase computation time.

The execution time of the numerical example with 9 nodes, as discussed above is calculated using Python. It was compared with the floyd-warshall and ant colony algorithm under the same network conditions. From Figure 5, we see that the execu-tion time of FWA is shorter than that of the ANTCA. Therefore, the proposed FWA is feasible and reasonable.

Execution Time = 0.002 seconds (FWA) Execution Time = 0.035 seconds (ANTCA)

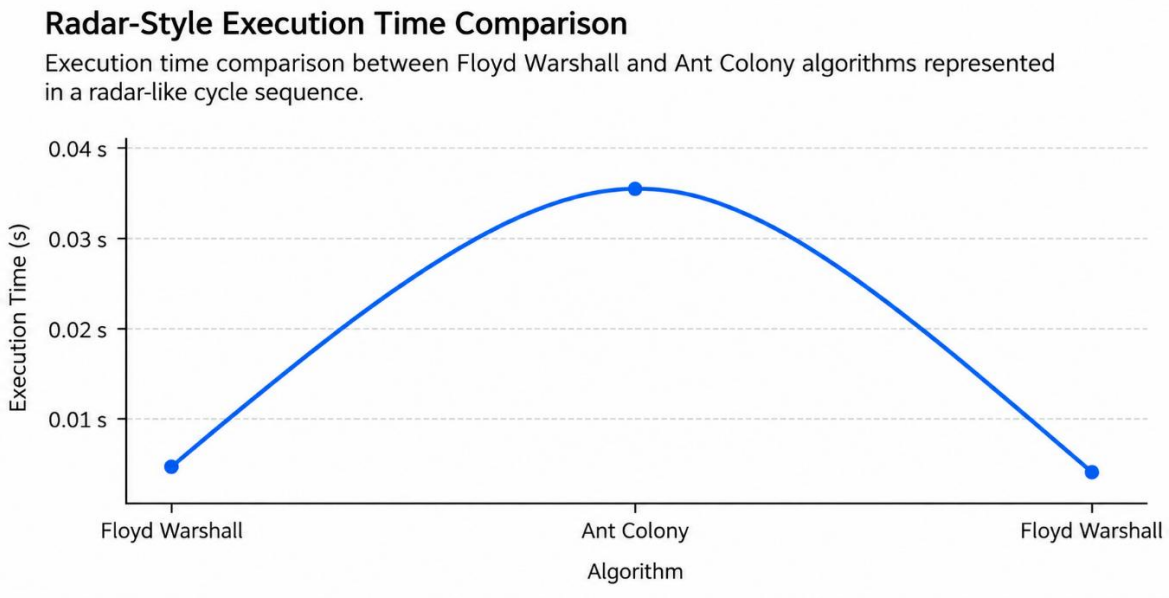

Figure 4: Execution time comparison of FWA with ANTCA.

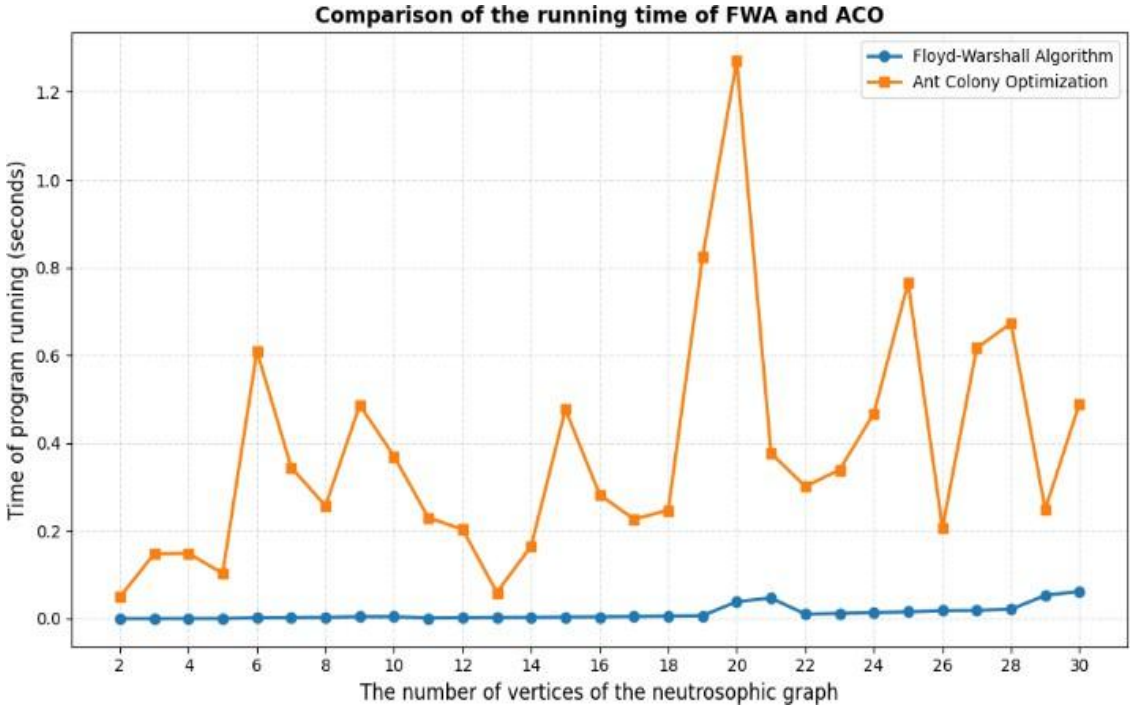

Figure 5: Comparision of the timeline of FWA and ACO.

In figure 5, the horizontal axis shows the number of vertices of the neutrosophic graph, and the vertical axis shows the running time (in seconds) of the Floyd-Warshall and Ant Colony Optimization algorithm.

8. Conclusion

In this research, the suggested Floyd-Warshall algorithm is employed to determine the shortest path and optimal cost for the example presented. The derived IVNSP and IVNMC results are then compared with established methods, including Dijkstra, and Ant Colony Algorithms. Upon comparison, the proposed Floyd-Warshall algorithm demonstrates a lower execution time, leading to a decrease in computational complex-ity, which enhances its efficiency for applications in telecommunication systems, traffic networks, computer networks, internet routing, airline operations, logistics manage-ment, and shortest path optimization. ANTCA only finds the shortest path between a specific origin and destination, such as source node and destination node, whereas Floyd-Warshall calculates the shortest path between every single pair of nodes at the same time.

In a post-disaster scenario, this is very useful because rescue teams often need to coordinate routes between several supply hubs and multiple disaster zones simultaneously. ANTCA is a metaheuristic, which means it provides an approxi-mate optimal path and can sometimes get stuck in local optima; it is a better choice for large-scale and dynamic optimization problems where adaptive and near-optimal solutions are acceptable. Floyd-Warshall is a deterministic dynamic programming algorithm that ensures the exact global optimum based on the score function. Fi-nally, our proposed algorithm, the Floyd-Warshall algorithm works better when exact shortest paths are required in relatively small or dense graphs.

Declaration

Funding: No funding is available for this study.

Data Availability Statement: This manuscript has no associated data.

Conflicts of Interest: The authors declare that they have no conflicts of interest.

References

1. Ali, M., & Smarandache, F. (2017). Complex neutrosophic set. *Neural computing and applications*, *28*(7), 1817-1834. https://doi.org/10.1007/s00521-015-2154-y
2. Anderson, B., Neal, T., Redden, B., & Fazea, Y. (2025, April). Comparative Analysis of Floyd-Warshall and Simulated Annealing for Traffic-Based Shortest Path Problems. In *2025 IEEE Opportunity Research Scholars Symposium (ORSS)* (pp. 1-7). IEEE. https://doi.org/10.1109/ORSS66051.2025.11121623
3. Anitha, J. (2026, March). An Algorithmic Approach to Optimizing Network Routing Using Dijkstra's Algorithm. In *2026 8th International Conference on Intelligent Sustainable Systems (ICISS)* (pp. 765-770). IEEE. https://doi.org/10.1109/ICISS67859.2026.11453799
4. Atanassov, K. T. (1999). Intuitionistic fuzzy sets. In *Intuitionistic fuzzy sets: theory and applications* (pp. 1-137). Heidelberg: Physica-Verlag HD. https://doi.org/10.1007/978-3-7908-1870-3_1
5. Broumi, S., Nagarajan, D., Bakali, A., Talea, M., Smarandache, F., & Lathamaheswari, M. (2019). The shortest path problem in interval valued trapezoidal and triangular neutrosophic environment. *Complex & Intelligent Systems*, *5*(4), 391-402. https://doi.org/10.1007/s40747-019-0092-5
6. Broumi, S., Smarandache, F., Talea, M., & Bakali, A. (2016, July). Single valued neutrosophic graphs: degree, order and size. In *2016 IEEE international conference on fuzzy systems (FUZZ-IEEE)* (pp. 2444-2451). IEEE.
7. Candra, A., Budiman, M. A., & Hartanto, K. (2020, July). Dijkstra's and a-star in finding the shortest path: A tutorial. In *2020 International Conference on Data Science, Artificial Intelligence, and Business Analytics (DATABIA)* (pp. 28-32). IEEE. https://doi.org/10.1109/DATABIA50434.2020.9190342
8. Habib, S., Majeed, A., Akram, M., & Al-Shamiri, M. A. (2023). Floyd-Warshall algorithm based on picture fuzzy information. *Computer Modeling in Engineering & Sciences*, *136*(3), 2873-2894. http://dx.doi.org/10.32604/cmes.2023.026294
9. Kanchana, M., & Kavitha, K. (2024). Heuristic incident edge path algorithm for interval-valued neutrosophic transportation network. *Contemporary Mathematics*, 2016-2036. https://doi.org/10.37256/cm.5220244152
10. Liu, R. (2020). Study on single-valued neutrosophic graph with application in shortest path problem. *CAAI Transactions on Intelligence Technology*, *5*(4), 308-313. https://doi.org/10.1049/trit.2020.0111
11. Mahmoudi, S., & Lailypour, C. (2015). A discrete binary version of the Forest Optimization Algorithm. *International Academic Journal of Innovative Research, 2*(2), 60–73.
12. Mirino, A. E. (2017, October). Best routes selection using Dijkstra and Floyd-Warshall algorithm. In *2017 11th international conference on information & communication technology and system (ICTS)* (pp. 155-158). IEEE. https://doi.org/10.1109/ICTS.2017.8265662
13. Sakharov, V., Chernyi, S., Saburov, S., & Chertkov, A. (2021). Automatization search for the shortest routes in the transport network using the Floyd-Warshell algorithm. *Transportation Research Procedia*, *54*, 1-11. https://doi.org/10.1016/j.trpro.2021.02.041
14. Sangeetha, D. P., Sekar, S., Parvathy, P. R., GaneshBabu, T. R., & Muthulekshmi, M. (2025, February). Optimizing shortest paths in big data using the Floyd-Warshall algorithm. In *2025 International Conference on Intelligent Control, Computing and Communications (IC3)* (pp. 382-387). IEEE. https://doi.org/10.1109/IC363308.2025.10957179
15. Kigarura, M., Okunki, L., & Nbende, P. (2026). Throughput-Calibrated Electromagnetic Compatibility Assessment of Connected-Vehicle Antennas Under Spatially Distributed RF Interference. *National Journal of Antennas and Wireless Communication Systems*, *1*(3), 38-45.
16. Smarandache, F. (2020). *The score, accuracy, and certainty functions determine a total order on the set of neutrosophic triplets (T, I, F)*. Infinite Study.

17. Smarandache, F. A. (1999) unifying field in logics, in Neutrosophy: Neu trosophic Prob-ability, Set and Logic. *Rehoboth, DE, USA*. American Research Press, 7–8.
18. Wang, H., Smarandache, F., Sunderraman, R., & Zhang, Y. Q. (2005). *interval neutrosophic sets and logic: theory and applications in computing: Theory and applications in computing* (Vol. 5). Infinite Study.
19. Yang, L., Li, D., & Tan, R. (2020). Research on the shortest path solution method of interval valued neutrosophic graphs based on the ant colony algorithm. *IEEE Access*, *8*, 88717-88728. https://doi.org/10.1109/ACCESS.2020.2990912
20. Ye, J. (2015). Multiple-attribute decision-making method under a single-valued neutrosophic hesitant fuzzy environment. *Journal of intelligent systems*, *24*(1), 23-36. https://doi.org/10.1515/jisys-2014-0001
21. Beken, K., Caddwine, H., Kech, R., & Mlein, M. (2026). Streaming RISC-V Vector Cochlear Encoder with Adaptive Spike-Rate Control for Multilingual Keyword Intelligence. *Journal of Integrated VLSI and Signal Intelligence*, *1*(3), 40-47.
22. Rai, H. T., Mu, G. W., & Lu, R. Q. (2026). Reliability-Aware Chiplet Floorplanning Using Transient Thermo-Mechanical Analysis and Micro-Bump Lifetime Optimization. *National Journal of VLSI Systems and Integrated Circuit Design*, *1*(3), 42-49.