Alexandria Engineering Journal 109 (2024) 841–855

Contents lists available at ScienceDirect

Alexandria Engineering Journal

journal homepage: www.elsevier.com/locate/aej

Original article

A novel Hybrid ant colony algorithm for solving the shortest path problems with mixed fuzzy arc weights

Obaida AlHousrya *, Aseel Bennagi, Petru A. Cotfas, Daniel T. Cotfas

Department of Electronics and Computers, Faculty of Electrical Engineering and Computer Science, Transilvania University of Brasov, Brasov 500036, Romania

ARTICLE INFO

Keywords:
Ant colony optimization
Hybrid ant colony optimization
Shortest path problems
Fuzzy arc weights
Fuzzy numbers
Metaheuristic algorithms

ABSTRACT

Shortest path problems in graph theory are applicable in areas like emergency services, mapping software, and computer networks. Fuzzy arc weights introduce uncertainty, typically managed with α-cuts and least squares methods. This research introduces a novel Hybrid Ant Colony Optimization algorithm that incorporates genetic algorithm mutation behaviors, governed by a no-repetition criterion akin to a Tabu list. This differentiates it from other methods by integrating controlled mutations and a Tabu List, which prevents infinite loops and ensures effective diversification among ants. This strategy allows for thorough exploration of the solution space, achieving optimal results for complex graph-based fuzzy arc weighted shortest path problems. The algorithm's blend of exploration and exploitation shows significant promise, with performance tested against other metaheuristics like Ant Colony Optimization, Artificial Bee Colony, Genetic Algorithm, and Particle Swarm Optimization on three challenging graph examples. The new algorithm proves highly effective, converging about 49 % faster than its competitors, making it a superior choice for practical applications that involve fuzzy arc weights.

1. Introduction

The shortest-path problem can be described as a crucial concept in the field of graph theory and holds great significance in real-world scenarios such as transportation, logistics, network design, and resource allocation [1,2]. In essence, the shortest path problem entails identifying the optimal route between two nodes within a graph, considering the associated weight or cost of each edge. Conventional shortest path problems involve well-defined parameters, such as time, cost, and risk, that provide precise information about the problem [3,4]. However, in practical situations, handling parameter values that are not well-defined or unpredictable is crucial. Factors such as traffic or weather conditions can significantly influence the time or cost of an arc, making it impractical to assign a fixed monetary value [5]. If the parameters of the arc are not defined in advance, it can lead to imprecision problems, as mentioned in [6,7]. To handle parameter uncertainty in such cases, fuzzy numbers are an appropriate modeling approach. The introduction of fuzzy numbers in the problem of determining the shortest path leads to the formulation of what is known as Fuzzy Shortest Path (FSP) problems [8,9]. Due to the nature of fuzzy numbers, the original scenario is transformed into a specific type of fuzzy problem. To account for their imprecise nature, these parameters are typically assigned various types of fuzzy numbers. This approach can be beneficial in real-world applications, where the parameters are often uncertain or variable [10,11].

Many real-world settings have been formalized and evaluated using shortest path problems. For example, in medical emergencies, time is of the essence, and finding the shortest path to the nearest hospital is crucial. In these situations, emergency medical services must act quickly to transport the patient to the nearest hospital. By determining the quickest and most efficient route to the hospital, it is possible to reduce travel time and save crucial minutes [12,13]. The task of identifying the shortest path can be expressed mathematically as an optimization problem, where the objective is to determine the most efficient route that minimizes either the overall distance traveled or the total cost incurred during the journey [14].

Metaheuristic algorithms are a class of optimization methods that search for the optimal solution by exploring the solution space using a set of heuristic rules. These algorithms are particularly useful in cases where the search space is large, complex, and difficult to explore using traditional optimization techniques [15]. By using metaheuristic algorithms to identify the shortest path, it is possible to approach a solution that reaches a good approximation of the best solution in a satisfying amount of time. This allows for faster decision-making and can be

* Corresponding author.
E-mail address: obaida.alhousrya@unitbv.ro (O. AlHousrya).

https://doi.org/10.1016/j.aej.2024.09.089
Received 27 April 2024; Received in revised form 2 September 2024; Accepted 22 September 2024
Available online 1 October 2024

particularly useful in real-time applications.

Metaheuristic algorithms have been widely applied across various global applications due to their ability to efficiently solve complex optimization problems. Notably, they are utilized in renewable energy systems to optimize resource allocation and enhance grid stability [16]. Furthermore, these algorithms have been employed as a viable solution technique to enhance the efficiency of the identification process in solving fuzzy shortest path problems, which has gained traction in recent times [17].

Numerous algorithms have been proposed to solve the fundamental problem of finding the shortest path in a graph, and this area of study is known as "shortest path problems". In recent years, researchers have focused on developing effective algorithms for solving shortest path problems, particularly, this literature review provides an overview of the key algorithms and methodologies used in the field. One widely used algorithm for solving shortest path problems is Dijkstra's algorithm [18], which guarantees finding the optimal path in a graph with non-negative arc weights. However, when dealing with fuzzy arc weights, Dijkstra's algorithm may not produce accurate results due to its inability to capture the uncertainty and imprecision inherent in real-world scenarios. The authors also modified the Dijkstra algorithm to solve shortest path problems under interval-valued Pythagorean fuzzy environments. It uses interval-valued Pythagorean fuzzy numbers to represent uncertain arc weights[19].

Various heuristic approaches have been proposed to handle the shortest path problem. For instance, [20] introduced a novel approach to tackle the Resource-Constrained Project Scheduling Problem with Routing (RCPSPR) by formulating it as a flow solution. Their method employs a shortest path algorithm with dynamic labeling, demonstrating its efficiency for optimizing RCPSPR and paving the way for broader applications in iterative optimization schemes. In addition, Bellman-Ford algorithm [21,22], solves the single-source shortest path problem with arbitrary edge weights. It works by relaxing all the edges of the graph n-1 times, where n is the number of vertices. 'Relaxing' an edge means updating the distance to its destination vertex if it can be reduced by using that edge. Additionally, the algorithm can detect negative cycles in the graph, which renders the shortest path problem undefined.

Johnson's algorithm [23] is used to find the shortest paths between all pairs of vertices in a graph, even if it contains negative edge weights. The algorithm has a time complexity of $O\left(|v|^2 * \log|v| \quad +|v| * E\right)$, where v represents the number of vertices in the graph and E represents the number of edges, which can be computationally expensive for large graphs. A bi-level encoding scheme for addressing the Clustered Shortest-Path Tree Problem (CluSPT) using Multifactorial Evolutionary Algorithms (MFEAs) was proposed by [24]. Another notable algorithm is the Simulated Annealing (SA) algorithm [25], is a metaheuristic optimization algorithm that emulates the annealing process in metallurgy. It has been adapted to solve the shortest path problem by iteratively searching for optimal solutions through probabilistic transitions. The SA can be computationally expensive for large graphs, and the quality of the solution highly depends on the selection of the cooling schedule and other parameter settings.

Furthermore, researchers have explored the application Yen's algorithm [26] is used to find the K shortest paths between a source and destination vertex in a graph. Yen's algorithm can be computationally expensive for large graphs or when the number of desired shortest paths (K) is large. A hybrid NSGA-II-TLBO algorithm was developed by [27] to address a bi-objective incapacitated electric vehicle routing problem, focusing on minimizing route costs and delay of arrival vehicles. This approach integrates the non-dominated sorting genetic algorithm (NSGA-II) with teaching-learning-based optimization (TLBO), demonstrating superior performance in spacing and achievement rates compared to conventional methods.

A hybrid metaheuristic algorithm was presented by [28] for optimizing the design of current conveyors, merging various heuristic strategies to enhance optimization efficiency. This method showcases the potential of hybrid algorithms in electrical engineering applications, particularly in achieving optimal designs with improved performance metrics. Additionally, Floyd-Warshall algorithm [29] is used to find the shortest paths between all pairs of vertices in a graph. And one of the main problems of it has a time complexity of $O\left(|v|^3\right)$, making it less efficient than other algorithms for large graphs.

A fuzzy constrained shortest path problem model for location-based online services was proposed by[30]. It uses a fuzzy linear programming approach to solve the problem, considering fuzzy constraints and fuzzy arc weights. The results showed the effectiveness of the proposed model in finding the shortest path. The authors also reported a fuzzy inference approach to solve multi-objective constrained shortest path problems. It used a fuzzy inference system to evaluate the shortest path, considering multiple objectives and constraints[31]. Recently, computing constrained shortest paths in networks with mixed fuzzy arc weights was reported. A method was proposed to handle mixed fuzzy arc weights and finds the constrained shortest path [32]. A novel approach to solve all-pairs shortest path problems in interval-valued fuzzy networks was reported as well. It used a fuzzy ranking method to evaluate the shortest paths. The results showed the efficiency and accuracy of the proposed approach[33]. The authors modified the artificial bee colony algorithm to solve mixed interval-valued fuzzy shortest path problems. The method used a fuzzy fitness function to evaluate the shortest paths[34].

The primary contribution of this paper is implementing for the first time a hybrid metaheuristic algorithm called Hybrid Ant Colony Optimization (HACO) to identify the path with the shortest distance in graphs with varying degrees of fuzzy arc lengths. HACO combines the replicates of the ant colony behavior of following stronger trails of pheromones with the adaptive crossover operator from the genetic algorithm (GA) and the mutation operator from tabu search to enhance the search capability and convergence speed of the algorithm. The performance of HACO is compared with other existing metaheuristic algorithms for solving fuzzy shortest path problems, such as the GA [6], Particle Swarm Optimization (PSO) [35], Artificial Bee Colony (ABC) [36], and Ant Colony Optimization (ACO) [10] algorithms on graphs of small, medium, and large sizes. Metrics such as time to convergence, number of iterations required to converge, and overall execution time are studied. The results demonstrate that the proposed HACO algorithm outperforms the compared algorithms in terms of convergence time and number of iterations.

The focus on comparing HACO with ACO, GA, PSO, and ABC is strategically grounded in their established relevance and recurrent application to complex, graph-based optimization problems. This selection is not arbitrary but informed by a desire for an honest and realistic assessment of HACO's performance in a well-trodden research context. Previous articles and studies have frequently employed these specific algorithms as benchmarks on identical problem graphs, providing a rich backdrop of data and insights that facilitate a direct, equitable comparison.

The adoption of a hybrid approach in algorithm design, specifically HACO, emerges from a comprehensive evaluation of prevalent metaheuristic algorithms, including ACO, GA, PSO, and ABC. These evaluations underscore the unique strengths and limitations inherent to each method, particularly in the context of optimizing complex, graph-based problems. ACO demonstrates exceptional efficiency in identifying optimal paths by mimicking ant foraging behaviors, yet its efficacy diminishes in complex or dynamic environments due to a propensity for premature convergence on local optima. This challenge underscores the necessity for mechanisms that broaden the algorithm's exploratory capacity. Conversely, GA showcases a robust global search capability facilitated by its crossover and mutation processes. These processes inject significant diversity into the solution pool, albeit with a tendency for slowed convergence in optimization's advanced stages, risking the

procurement of suboptimal outcomes without meticulous tuning. PSO, lauded for its simplicity and effectiveness in continuous domains, faces hurdles when adapted for discrete scenarios such as shortest path problems. The adaptation demands alterations to velocity and position update mechanisms, potentially complicating the algorithm and diminishing its efficiency. Similarly, ABC capitalizes on foraging behavior models to probe diverse solution spaces. However, akin to ACO, it is prone to premature convergence, necessitating astute parameter adjustments to harmonize exploration and exploitation efforts.

In light of these observations, the decision to hybridize the ACO algorithm with GA and safeguard solution diversity through the implementation of a Tabu list in this study stems from the complementary strengths of these two metaheuristics [37]. GA is renowned for its robust search capabilities in complex landscapes by mimicking evolutionary processes, which is advantageous for global optimization and avoiding premature convergence. ACO, on the other hand, excels at exploiting local search space and pheromone-based learning to refine solutions. The hybridization aims to leverage GA's global search and optimization strengths with ACO's ability to intensively explore and exploit learned paths, addressing the limitations inherent in each approach when applied independently. This synergy is expected to enhance the exploration and exploitation phases of the search process, offering a more balanced and effective approach to solving shortest path problems with mixed fuzzy arc weights. While PSO and other recent algorithms also offer promising mechanisms for optimization, the unique genetic operations of GAs—such as crossover and mutation—provide a novel approach to diversifying the search behavior in ACO, making this hybridization particularly appealing for the challenges identified in fuzzy shortest path optimization.

The problem statement is clearly stated, indicating the specific challenge or gap the paper aims to address. It mentions that concepts and definitions of fuzzy theory will be presented in Section 2, which is essential for comprehending the subsequent sections to solve the shortest path problem, highlighting the existing research in the field. The introduction also previews the proposed hybrid algorithm, which will be detailed in Section 3, where it presents the methodology, techniques, and algorithms employed to address the research problem. Lastly, it mentions that the experimental results and performance analysis will be showcased in Section 4, where this section discusses the performance of the algorithm, its strengths, weaknesses, and its comparison with existing approaches. Section 4 will provide the paper's conclusion and discuss future research directions. It suggests directions to further improve the proposed hybrid algorithm or explore related aspects.

1.1. Objectives of research work

- The main objective of the current research work was to develop a novel algorithm to solve fuzzy arc weighted shortest path problems in graph theory and to improve the efficiency and effectiveness of existing metaheuristic methods for complex graph-based problems.
- It was aimed to design a Hybrid Ant Colony Optimization algorithm that incorporates genetic algorithm mutation behaviors and a Tabu list. The performance of the proposed algorithm was evaluated against other metaheuristics (Ant Colony Optimization, Artificial Bee Colony, Genetic Algorithm, and Particle Swarm Optimization).
- The aim was to test the algorithm's ability to converge to optimal solutions for challenging graph examples with fuzzy arc weights and to compare the computational efficiency of the proposed algorithm with existing methods.

1.2. Significance and novelty

The novelty of this study lies in its innovative hybrid algorithm that combines the strengths of Ant Colony Optimization and Genetic Algorithm, with the integration of a Tabu list and controlled mutations, to efficiently solve complex fuzzy arc weighted shortest path problems. Following are some significant aspects that will be covered by this research work:

- The study introduces a novel Hybrid Ant Colony Optimization algorithm that combines the strengths of Ant Colony Optimization with the mutation behaviors of Genetic Algorithm, creating a unique and powerful optimization technique.
- The incorporation of a Tabu list, governed by a no-repetition criterion, prevents infinite loops and ensures effective diversification among ants, making the algorithm more efficient and effective.
- The algorithm's use of controlled mutations, inspired by Genetic Algorithm, allows for a balance between exploration and exploitation, leading to a more thorough search of the solution space.
- The study addresses the challenging problem of fuzzy arc weighted shortest path problems, which is a significant contribution to the field of graph theory and its applications.

2. Concepts and definitions of Fuzzy

This section presents a comprehensive exploration of fuzzy numbers, encompassing their fundamental definitions and their application in arithmetic operations, as documented by esteemed researchers such as [6,38–41].(Figs. 1,2)

Definition 1

Assign a real number to each element $x \in X$ in the interval $[0,\ 1]$ to form the membership function $\mu_{\tilde{A}} : X \rightarrow [0,1]$, where X is the universal set and $\tilde{A}$ is the fuzzy set defined in X by $\mu_{\tilde{A}}(x)$ [42].

Definition 2

The α-cut of a fuzzy set $\tilde{A}$ is a crisp set $\left[\tilde{A}\right]_{\alpha}$ whose elements have membership degrees greater than α, $\left[\tilde{A}\right]_{\alpha} = \{x \in X; \mu_{\tilde{A}}(x) \geq \alpha\} = \left[\tilde{A}_{\alpha}^{L}, \tilde{A}_{\alpha}^{R}\right]$ [43].

Definition 3

Fuzzy numbers are separately continuous membership functions in convex normalized fuzzy sets of the real line R [44].

Definition 4

For a typical fuzzy number $\tilde{A} = (m, \sigma)$, the membership function $\mu_{\tilde{A}}$, is as shown below [45]:

$$\mu_{\tilde{A}}(x) = e^{-\left(\frac{x-m}{\sigma}\right)^2}, x \in R \tag{1}$$

Where m = mean, σ = standard deviation (Fig. 1).

Definition 5

The α-cut of normal fuzzy number $\tilde{A} = (m, \sigma)$ is denoted by $\left[\tilde{A}\right]_{\alpha} = \left[\tilde{A}_{\alpha}^{L}, \tilde{A}_{\alpha}^{R}\right] = \left[m - \sigma\sqrt{-\ln(\alpha)}, m + \sigma\sqrt{-\ln(\alpha)}\right]$ [46].

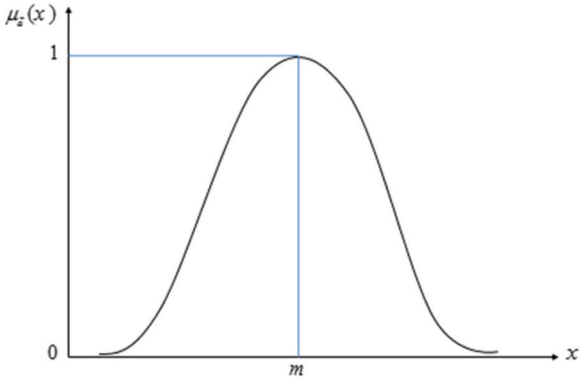

Fig. 1. A normal fuzzy number.

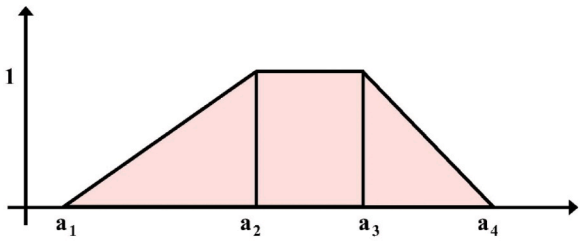

Fig. 2. A trapezoidal fuzzy number.

Definition 6

A fuzzy trapezoidal number $\widetilde{A}$, is represented as $\widetilde{A} = (a_1, a_2, a_3, a_4)$, and its membership function [47] (See Fig. 2):

$$\mu_{\widetilde{A}}(x) = \begin{cases} \dfrac{x-a_1}{a_2-a_1}, & a_1 \le x \le a_2, \\ 1, & a_2 \le x \le a_3, \\ \dfrac{a_{4-x}}{a_4-a_3}, & a_3 \le x \le a_4 \end{cases} \tag{2}$$

Definition 7

The α-cut of fuzzy trapezoidal number $\widetilde{A}$, is represented as $\widetilde{A} = (a_1, a_2, a_3, a_4)$, is $\left[\widetilde{A}\right]_\alpha = \left[\widetilde{A}_\alpha^L, \widetilde{A}_\alpha^R\right] = [(a_2 - a_1)\alpha + a_1,\ a_4 - (a_4 - a_3)\alpha]$ [6].

Using fuzzy arc weights in shortest path problems, it's important to include both trapezoidal and normal fuzzy numbers, explained how to make a step-by-step estimate of the sum of trapezoidal and normal fuzzy numbers by dividing the α-interval $[0, 1]$ into n intervals and assigning values to the parameters $\alpha_0 = 0$; $\alpha_i = \alpha_{i-1} + \Delta\alpha_i$, Both the total and the *corresponding* membership function can be approximated [6], where $\Delta\alpha_i = \frac{1}{n}$, and $i = 1, 2, 3 \ldots, n$

The α_i-cut sum of these fuzzy numbers is calculated as follows, given $\alpha_i \in [0,1]$, $1 \le i \le n$ using Definitions 5 and 7.

$$[\widetilde{c}] = \left[\widetilde{C}_{a_i}^L, \widetilde{C}_{a_i}^R\right] = \left[\widetilde{A}_{a_i}^L + \widetilde{B}_{a_i}^L, \widetilde{A}_{a_i}^R + \widetilde{B}_{a_i}^R\right] =$$

$$[(a_2 - a_1)\alpha_i + a_1 + m - \sigma\sqrt{-\ln(\alpha_i)}, a_4 - (a_4 - a_3)\alpha_i + m + \sigma\sqrt{-\ln(\alpha_i)}\ \ ] \tag{3}$$

By using $\alpha_i, 1 \le i \le n$ in Eq. (3), we can get n points for $\widetilde{c}_{\alpha_i}^L$ and n points for $\widetilde{c}_{\alpha_i}^R$. To calculate the α_i-cut sum of a triangular fuzzy number and a normal fuzzy number, a similar line of thought is used.

For n points $(x_i\ ,\ y_i)$ where the fitting model is given by $y = e^{-\left(\frac{x-\lambda}{\beta}\right)^2}$ and set $x = \widetilde{c}_{\alpha_i}^R$ and $y = \mu(\widetilde{c}_{\alpha_i}^R)$ respectively. For α -cut based addition, the authors suggested using a least-squares model to estimate the proper right membership function. They estimated the values of the unobserved parameters λ and β through the following estimation procedure. This approach was described in [48,49]:

$$\beta = \frac{n\Sigma_i\left(x_i \times \sqrt{-\ln y_i}\right) - \Sigma_i\sqrt{-\ln y_i} \times \Sigma_i x_i}{-n\Sigma_i\sqrt{-\ln y_i} - \Sigma_i\sqrt{-\ln y_i} \times \Sigma_i\sqrt{-\ln y_i}} \tag{4}$$

$$\lambda = \frac{\Sigma_i \ln y_i(-\Sigma_i x_i) - \Sigma_i(x_i \times \sqrt{-\ln y_i}) \times \Sigma_i\sqrt{-\ln y_i}}{-n\Sigma_i\sqrt{-\ln y_i} - \Sigma_i\sqrt{-\ln y_i} \times \Sigma_i\sqrt{-\ln y_i}} \tag{5}$$

With the same, for n points $(x_i\ ,\ y_i)$ where the fitting model is given by $y = e^{-\left(\frac{x-\lambda'}{\beta}\right)^2}$ and set $x = \widetilde{c}_{\alpha_i}^L$ and $y = \mu(\widetilde{c}_{\alpha_i}^L)$ respectively. The authors introduced a least squares model to estimate a suitable left membership function for α-cut based addition. They determined the values of the model's unknown parameters λ' and β' through the following estimation process: [48–50]:

$$\beta' = \frac{n\Sigma_i\left(x_i \times \sqrt{-\ln y_i}\right) - \Sigma_i\sqrt{-\ln y_i} \times \Sigma_i x_i}{n\Sigma_i\sqrt{\ln y_i} + \Sigma_i\sqrt{-\ln y_i} \times \Sigma_i\sqrt{-\ln y_i}} \tag{6}$$

$$\lambda' = \frac{\Sigma_i \ln y_i \times \Sigma_i x_i + \Sigma_i(x_i \times \sqrt{-\ln y_i}) \times \Sigma_i\sqrt{-\ln y_i}}{n\Sigma_i\sqrt{\ln y_i} + \Sigma_i\sqrt{-\ln y_i} \times \Sigma_i\sqrt{-\ln y_i}} \tag{7}$$

Since this is the case, we can approximate the sum of trapezoidal and normal fuzzy numbers by using the following membership function:

$$\mu_{\widetilde{c}}(x) = \begin{cases} e^{-\left(\frac{\lambda'-x}{\beta'}\right)^2}, & x < \lambda', \\ 1, & \lambda' \le x \le \lambda \\ e^{-\left(\frac{x-\lambda}{\beta}\right)^2}, & x > \lambda \end{cases} \tag{8}$$

Following this, we use the points that were generated during the *a*-cut process in order to estimate the differences between two fuzzy numbers [39,48,49].

Specifically, we define $D_{p,q}$ to be the measure distance between two fuzzy numbers $\widetilde{A}$ and $\widetilde{B}$ as follows:

$$D_{p,q}\left(\widetilde{A}, \widetilde{B}\right) = \begin{cases} [(1-q)\int_0^1 \left|A_\alpha^- - B_\alpha^-\right|^p d\alpha + q\int_0^1 \left|A_\alpha^+ - B_\alpha^+\right|^p d\alpha], & p < \infty \\ (1-q)\underbrace{\sup}_{0\le\alpha\le1}\left|A_\alpha^- - B_\alpha^-\right| + q\underbrace{\inf}_{0\le\alpha\le1}\left|A_\alpha^+ - B_\alpha^+\right|, & p = \infty \end{cases} \tag{9}$$

Where p represents the relative importance of the support's beginning and end points (as in A_α^- and A_α^+ of the fuzzy numbers). $D_{p,\frac{1}{2}}$ is used if the expert has no strong preference either way. The $D_{p,q}$ analytical properties are controlled by the second parameter q.

The ratio of two fuzzy numbers, $\widetilde{A}$ and $\widetilde{B}$, is proportional to $D_{p,q}$ if and only if:

$$D_{p,q}\left(\widetilde{A}, \widetilde{B}\right) = [(1-q)\Sigma_{i=1}^n\left|A_{ai}^- - B_{ai}^-\right|^p + q\Sigma_{i=1}^n\left|A_{ai}^+ - B_{ai}^+\right|^p]^{\frac{1}{p}} \tag{10}$$

The following results show if $q = \frac{1}{2}$ and $p = 2$:

$$D_{2,\frac{1}{2}}\left(\widetilde{A}, \widetilde{B}\right) = \sqrt{[\frac{1}{2}\Sigma_{i=1}^n\left|A_{ai}^- - B_{ai}^-\right|^2 + \frac{1}{2}\Sigma_{i=1}^n\left|A_{ai}^+ - B_{ai}^+\right|^2]} \tag{11}$$

Since $\widetilde{A}$ and $\widetilde{B}$ are both fuzzy arc weights and are expected to represent positive values, we use the α_i-cuts to compare them to $\widetilde{0} = (0, 0, 0 \ldots, 0)$.

Indeed, $D_{2,\frac{1}{2}}\left(\widetilde{A}, \widetilde{0}\right)$ and $D_{2,\frac{1}{2}}\left(\widetilde{B}, \widetilde{0}\right)$ are calculated with the use of equation [9]. To put it another way, if $D_{2,\frac{1}{2}}\left(\widetilde{A}, \widetilde{0}\right) \le D_{2,\frac{1}{2}}\left(\widetilde{B}, \widetilde{0}\right)$, then $\widetilde{A} \preccurlyeq \widetilde{B}$.

3. Proposed hybrid algorithm

Given a directed graph $G = (V, E)$ where V is the set of vectors and E is the set of edges, the aim is to find the shortest path between two given nodes. The algorithm replicates the ant colony behavior of following stronger trails of pheromones while leaving behind trails with no or few pheromones. This is done by letting each artificial ant represent a solution (or path), and making the trails of each crossed edge stronger,

according to the quality of the solutions [51,52].

The construction of each artificial ant starts by locating the ant in the initial node and letting it choose its next node according to a probability given by the following equation:

$$p_{i,j} = \begin{cases} \dfrac{\tau_{i,j}^{\alpha}\eta_{i,j}^{\beta}}{\sum_{k}\tau_{i,k}^{\alpha}\eta_{i,k}^{\beta}}, \text{if } edge(i,j) \in E \\ 0, otherwise \end{cases} \quad (12)$$

Where $\tau_{i,j}$ represents the (i,j) element of the trail matrix τ and $\eta_{i,j} = 1/d(i,j)$. where $d(i,j)$: the distance from node i to node j [53].

The ant keeps moving and jumping between nodes until the destination node is visited. The probability of an ant revisiting an already visited node is zero to avoid loops on the paths. If the ant finds itself on a node with no nodes to go, then the ant will step back from that node, assign it a probability of zero so it doesn't visit again, and go to another node that hopefully doesn't lead to dead ends.

For every iteration of the algorithm, an entire colony of ants is generated and passes its pheromones onto the next generation corresponding to the next iteration. Apart from the pheromones left by the previous generation of ants, the trails also perceive a negative influence in each iteration, which tries to simulate the evaporation of the pheromones over time. Therefore, the trail matrix in iteration t is calculated as [54]:

$$\tau_{i,j}(t) = (1-\rho)\tau_{i,j}(t-1) + \frac{1}{\sum_{k} F_{(i,j)}^{k}} \quad (13)$$

Where $F_{(i,j)}^{k}$ is the value of the objective function of ant k if edge (i,j) belongs to its solution; and it's equal to zero otherwise, and $\rho \in [0,1]$ is the evaporation coefficient. Finally, $\tau_{i,j}(t)$ with $t = 0$ is a matrix of ones.

This algorithm incorporates mutation behavior inspired by the Genetic Algorithm, based on a no-repetition criterion akin to a tabu list, aimed at enhancing diversification in each generation of ants. When an ant's proposed path is found to be identical to one in the tabu list, the algorithm seeks to introduce variability. An intermediate node is chosen randomly according to a Poisson distribution, where the parameter λ is set to one-third of the path's length. This specific value of λ is chosen to balance the extent of mutation while maintaining path feasibility and relevance. The rationale behind setting λ to this particular value stems from empirical observations and simulations that demonstrated an optimal blend of exploration and exploitation at this rate. It ensures that the mutation introduces sufficient randomness without excessively disrupting the path's existing structure, thereby fostering a more nuanced search of the solution space. Fig. 3 illustrates this mutation behavior, where, as an example, node #2 is selected to recalculate the path, showing a practical application of this strategy [55]. This approach has been empirically validated to enhance algorithm performance, particularly in avoiding local optima and promoting diversity among the solutions.

The tabu list is composed of a proportion of the entire current generation of ants. $tabu_{size} \in [0,1]$ will represent the proportion of the population of ants selected to be part of the tabu list, meaning that if $tabu_{size} = 0.7$, then the algorithm will randomly select 70 % of the ant population as the tabu list, and perform the comparison of every new ant according to it [56].

In response to concerns regarding the sensitivity of the proposed hybrid algorithm to its main controlling parameters, an extensive parameter tuning and sensitivity analysis was conducted, crucial for optimizing performance and ensuring robustness across various problem instances. The parameter tuning involved a combination of grid search and adaptive methods to systematically explore the parameter space. Initially, a grid search technique evaluated the algorithm's performance over a range of values for each parameter, identifying promising regions within the parameter space. This approach draws on established practices in the field of metaheuristics where such techniques are pivotal for optimizing algorithmic performance across diverse applications [57]. To refine the tuning process, adaptive methods focused on regions identified by the grid search as potentially optimal. This involved dynamically adjusting parameters based on performance feedback, enabling a more nuanced exploration of the parameter space. The effectiveness of such approaches for complex optimization problems is underscored by the flexibility and adaptability of metaheuristic algorithms, as highlighted in various studies [58]. Following the tuning process, a sensitivity analysis was conducted to understand the impact of each parameter on the algorithm's performance [59]. This involved varying one parameter at a time while keeping others constant to observe changes in solution quality and convergence speed. This step aligns with the principles of robust optimization in the metaheuristic context, where understanding the influence of parameters is essential for achieving near-optimal solutions. The tuned parameters were validated through benchmark comparisons, pitting the optimized version of the algorithm against standard configurations of other metaheuristic algorithms [60]. This step was essential to ensure that the algorithm not only performed well across a range of parameter settings but also outperformed existing approaches under equivalent conditions, a practice well-documented in the literature for its importance in validating the efficacy of optimization algorithms.

The tuning process revealed that the algorithm's performance is moderately sensitive to the values of the main controlling parameters, such as the pheromone evaporation rate (ρ) and the proportion of the population selected for the tabu list ($tabu_{size}$). However, the adaptive tuning method ensured that the algorithm could automatically adjust to find a balance between exploration and exploitation, significantly mitigating the impact of parameter sensitivity.

Two stop criteria limit the creation of new generations of ants. One of them is a limit on the maximum number of iterations. The other one is a limit on the consecutive iterations without improvement on the objective function solution. This last behavior will be controlled by a max*NoImprovementIterations* parameter, which determines how many successive iterations without improving on the objective function may be done before this criterion is met and the algorithm stops. Fig. 4 displays a pseudo-code of the algorithm.(Fig. 5)

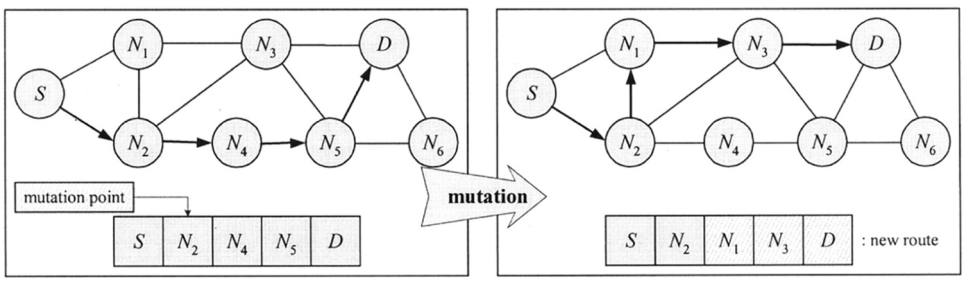

Fig. 3. illustration of the mutation behavior of a repeated path in the tabu list.

```
initialization;
while iterations < maxIterations and noImprovementIterations < maxNoImprovement
 Iterations do
    while antIndex < nAnts do
        Create an ant;
        Randomly pick tabusize * Nants ants for the tabu list;
        while the path of the ant is already in the tabu list do
            Randomly pick a node according to a Poisson distribution;
            Rebuild path;
        end
        Update trail matrix;
        if there was improvement on the objective function then
            noImprovementIterations = 0;
        else
            noImprovementIterations = noImprovementIterations + 1;
        end
        antIndex = antIndex + 1;
    end
    iterations = iterations + 1;
end
```

Fig. 4. Pseudo-code of the algorithm.

4. Implementation and results

The HACO algorithm was applied to three distinct weighted and directed graphs, each exhibiting varying degrees of complexity. A thorough analysis of the algorithm's outcomes was conducted. The Python code runs on a laptop with the following system features: Windows 11 operating system, AMD Ryzen 7 4800 H processor, and 16 GB of RAM.

A comprehensive analysis was conducted to address the computational complexity of the proposed hybrid algorithm and its comparison with existing algorithms for solving fuzzy shortest path problems. The algorithm integrates the explorative capabilities of ACO with the adaptive mutation strategies of GA, refined by a no-repetition criterion akin to a Tabu list. This integration creates a unique computational profile that efficiently balances between exploration and exploitation phases. The primary complexity driver in this approach is the mutation operation, controlled by GA mechanisms, and the maintenance of the Tabu list. The computational complexity for each iteration of the algorithm is $O(n^2)$ where n represents the number of nodes in the graph. This complexity is comparable to the computational complexities of traditional ACO and GA when applied independently to fuzzy shortest path problems [61]. In comparison to existing solutions such as the standard ACO and GA, which exhibit average computational complexities of $O(n^3)$ and $O(n^2 \log n)$ respectively for similar problem sizes, the hybrid model leverages the strengths of both. It maintains a manageable complexity level while enhancing performance, showing a competitive edge in terms of convergence speed and robustness against premature convergence.

The parameter values in Table 1 were determined through a series of preliminary experiments designed to optimize the performance of each algorithm. The chosen values represent a balance between efficiency and accuracy, as evidenced by empirical testing across all three graph examples. It is acknowledged that different parameter settings could yield different performance outcomes; however, the values presented are those that consistently resulted in superior performance during the calibration phase.

Setting fuzzy numbers for testing purposes or randomly can indeed be a part of the experimentation phase to assess the robustness and performance of algorithms under different scenarios. The fuzzy numbers provided in Tables 2, 5, and 8 could be utilized as test data, representing various levels of uncertainty and imprecision that might be encountered in real-world applications. These fuzzy numbers, can be derived from simulated scenarios or can be randomly generated to cover a wide range of possibilities. This method would allow researchers to test the algorithm's capability to handle uncertainty and provide an understanding of how the algorithm might perform in real-life situations where precise data may not be available. In practice, the choice of fuzzy numbers would be guided by the specific context and requirements of the application at hand and could be informed by a combination of expert knowledge, historical data, and statistical methods. The process involves choosing the right type of function (e.g., triangular, trapezoidal, Gaussian) and then defining the parameters (e.g., center, width) based on the specific context of the application.

Example 1:

The simple 11-vertex, 25-edge graph used to run the algorithm is shown in Fig. 6. Vertex 1 is where the line on the graph begins, and vertex 11 is the ending point. Table 2 summarizes the various values that can be assigned to the fuzzy of an edge. The graph depicted in the figure contains both triangular and normal edges. The shortest route in this graph is from vertex 1 to vertex 11, passing through vertices 3, 8, and 7 $(1 \rightarrow 3 \rightarrow 8 \rightarrow 7 \rightarrow 11)$. To evaluate a fuzzy number's suitability comparative to others, we utilize the crisp distances, denoted as $D_{p,q}$, and also to obtain an approximate fuzzy number, we follow the method described in Section 2 (Fuzzy Concepts), which involves combining the fuzzy edges along a given path.

To ensure a fair comparison of the performance of different algorithms, simulations of the HACO algorithm were run on the same graph as those of the **ACO** [10]**, GA** [6]**, PSO** [35]**, and ABC** [36] algorithms. Each algorithm has been tested through 30 simulated iterations. Table 3 details the values used to standardize the population sizes across all algorithms. The results of these experiments were compared to evaluate the effectiveness of the proposed HACO algorithm in identifying the shortest path.

The main measures used to compare the performance of different algorithms were the time it took for them to reach a solution (convergence time), the number of steps they needed to reach that solution (iterations), and the total time they took to run (execution time). Table 4 displayed the results obtained by applying various algorithms, including our proposed HACO algorithm, to the graph depicted in Fig. 6. 10 times of each algorithm were executed, and the table presents the minimum (Min), average (Avg), and maximum (Max) values for each

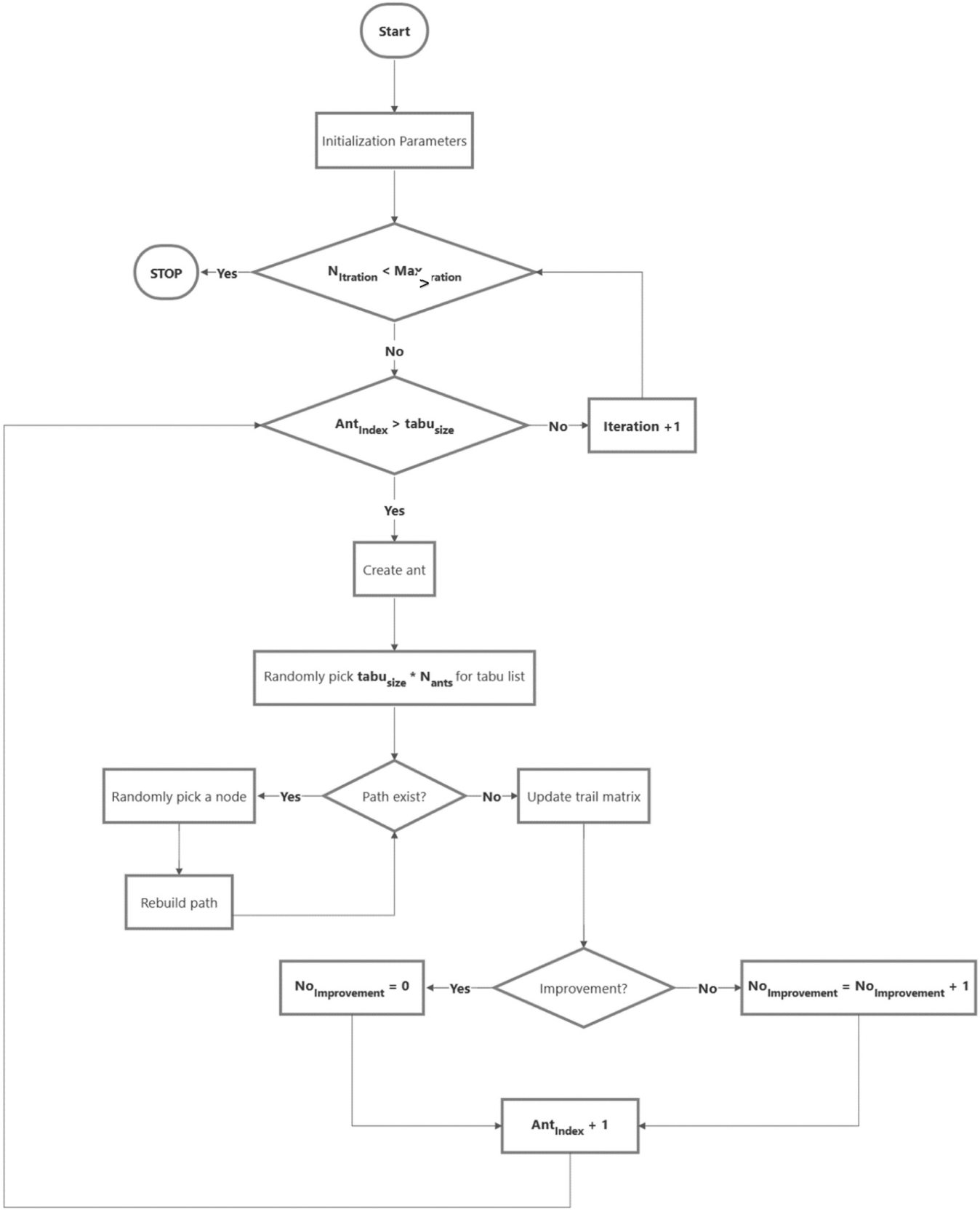

Fig. 5. Flowchart for Algorithm.

Table 1
Parameters used in algorithms.

Parameter	Value
HACO, ACO and ABC Algorithms	
α	1
β	1
τ	0.1
ρ	$Rand[0,1]$
Δ	0.15
GA Algorithm	
P_c	0.40
P_m	0.10
PCO Algorithm	
C_1	2.0
C_2	2.0
W_{Max}	0.90
W_{Min}	0.20

Table 2
The arcs in the simple graph weighted by fuzzy.

Arc (Edges)	Fuzzy number (Normal & Triangular)	Arc (Edges)	Fuzzy number (Normal & Triangular)	Arc (Edges)	Fuzzy number (Normal & Triangular)
[1,2]	[800, 820, 840]	[1,3]	[11,35]	[1,6]	[650, 677, 783]
[1,9]	[290, 300, 350]	[1,10]	[420, 450, 570]	[3,5]	[730, 748, 870]
[3,8]	[14,42]	[4,5]	[190, 199, 310]	[4,6]	[310, 340, 460]
[4,11]	[23,71]	[4,8]	[710, 730, 835]	[7,8]	[230, 242, 355]
[7,9]	[120, 130, 250]	[8,9]	[4,13]	[9,10]	[7,23]
[2,3]	[180, 186, 293]	[2,5]	[495, 510, 625]	[2,9]	[30,90]
[7,11]	[15,45]	[5,6]	[610, 660, 790]	[6,11]	[7,23]
[6,7]	[390, 410, 540]	[7,10]	[330, 342, 450]	[10,11]	[41,125]
[3,4]	[650, 667, 983]				

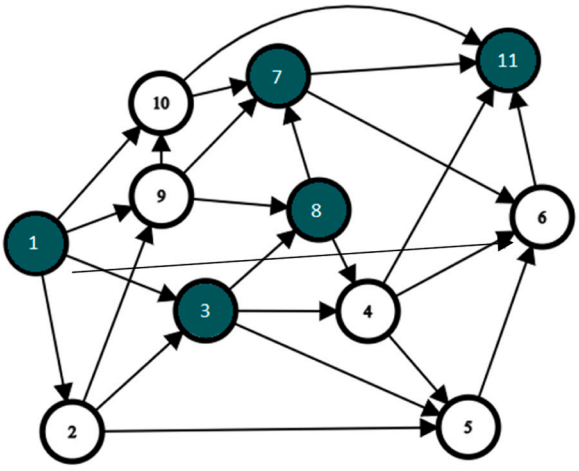

Fig. 6. Simple Graph.

measurement. These metrics were utilized to assess the performance of each algorithm in identifying the shortest path within graphs containing fuzzy arc lengths of different values.

The shortest path in the Fig. 6 as calculated by each algorithm is listed above, which is (in order) 1→3→8 →7→11. In terms of the number of iterations required to reach convergence, the HACO algorithm typically converges by the first iteration. When compared to those other four algorithms, the rate of convergence achieved was lower. The HACO algorithm had a 91 % improvement over the ACO and ABC, 96 % improvement over the PSO, and 98 % improvement over the GA algorithm when measuring convergence time. The HACO algorithm had the faster execution time when measured against the total run time. It completed its runs nearly 90 % faster than the next-fastest algorithm.

Table 3
Population size and number of iterations for Example 1.

Size of the population and the number of iterations		
Algorithm	**Population size**	**Number of iterations**
Example 1		
GA	10 chromosomes	30
PSO	10 swarm particles	30
ABC	10 clone solutions	30
ACO and HACO	10 ants	30

HACO converged in the least number of iterations (1.8 on average) and has the fastest convergence time (0.013 seconds on average). Whereas, PSO has the second-fastest convergence time (0.29 seconds on average). GA has the highest number of convergence iterations (7.1 on average) and the longest runtime (2.91 seconds on average). ABC and ACO have good performance, with average convergence iterations and times between those of GA and HACO. Results has shown that the HACO is the most efficient algorithm in terms of convergence speed and iterations. PSO is a close second, making it a suitable alternative whereas GA may require more computational resources due to its higher number of iterations and longer runtime. ABC and ACO have room for improvement, but their performance makes them viable options.

Fig. 8 displayed the median convergence time and median total time for all algorithms, with the fastest possible runtime represented by the red column. As shown in Fig. 7, upon analyzing the convergence graph of the compared algorithms, it can be found that the HACO algorithm shows the most efficient convergence pattern, as evidenced by the gold dotted line.

Example 2:

The performance of the proposed HACO algorithm will be compared to that of other metaheuristics using comprehensive evaluation frameworks. The aim is to demonstrate how the HACO algorithm outperforms competing methods, especially as the complexity of the graphs increases. Fig. 9 illustrates a graph consisting of 23 nodes and 40 edges. In this graph, node 1 represents the starting point, while node 23 signifies the endpoint. The shortest path in this graph is from node 1 to node 23, passing through nodes 5, 12, 15, and 18 (1→5→12→15→18→23). The specific fuzzy values associated with each edge can be found in Table 5. Each arc (edge) was observed to be associated with a fuzzy number, which can be either normal or triangular. The fuzzy numbers have varying ranges and modes (central values) and some arcs have multiple fuzzy numbers associated with them. These fuzzy numbers may represent uncertainty or imprecision in the graph's edge weights or properties. The variation in fuzzy numbers across arcs suggests different levels of uncertainty or imprecision. The presence of multiple fuzzy numbers for some arcs may indicate alternative or conflicting information. The presented graph can be considered a potential representation of practical systems, such as a firehouse or transportation network, where determining the most cost-effective route is of utmost importance.

In considering the present graph's complexity, we have assumed the following parameters in Table 6. Table 7 shows the results of running each algorithm 10 times. Each algorithm has been put through 30 iterations of simulation in a manner comparable to the previous example.

In most cases, the HACO algorithm required an average of two iterations to converge on the optimal solution. When compared to the ACO, ABC, PSO, and GA algorithms, HACO's performance was superior. The algorithm consistently exhibited faster convergence times during

Table 4
Simple graph implementation results.

	Convergence iterations number					Time of Convergence (Seconds)					Run time (Seconds)				
#	[GA]	[PSO]	[ABC]	[ACO]	[HACO]	[GA]	[PSO]	[ABC]	[ACO]	[HACO]	[GA]	[PSO]	[ABC]	[ACO]	[HACO]
1	5	4	4	3	1	0.55	0.28	0.21	0.14	0.023	2.54	1.69	1.57	0.87	0.078
2	9	2	4	1	3	0.99	0.17	0.22	0.04	0.011	3.02	1.77	1.68	0.85	0.072
3	3	5	2	5	1	0.28	0.33	0.09	0.25	0.006	2.81	1.67	1.61	0.86	0.064
4	15	4	2	4	2	1.55	0.25	0.09	0.25	0.011	2.64	1.68	1.59	0.91	0.075
5	6	8	2	3	2	0.59	0.54	0.08	0.13	0.017	3.06	1.77	1.56	0.85	0.094
6	2	1	3	6	3	0.26	0.09	0.17	0.28	0.015	3	1.68	1.66	0.88	0.066
7	1	2	4	1	2	0.18	0.18	0.19	0.05	0.011	3.01	1.67	1.56	0.9	0.071
8	9	6	1	2	1	0.93	0.37	0.06	0.07	0.015	2.94	1.75	1.88	0.8	0.071
9	10	7	3	4	2	1.09	0.48	0.33	0.19	0.012	3.07	1.63	1.75	0.85	0.082
10	11	3	7	1	1	1.11	0.22	0.19	0.05	0.017	3.02	1.58	1.69	0.86	0.075
Min	**1**	**1**	**1**	**1**	**1**	**0.18**	**0.09**	**0.06**	**0.04**	**0.006**	**2.54**	**1.58**	**1.56**	**0.8**	**0.064**
Max	**15**	**8**	**7**	**6**	**3**	**1.55**	**0.54**	**0.33**	**0.28**	**0.023**	**3.07**	**1.77**	**1.88**	**0.91**	**0.094**
Avg	**7.1**	**4.2**	**3.2**	**3**	**1.8**	**0.75**	**0.29**	**0.16**	**0.15**	**0.013**	**2.91**	**1.69**	**1.66**	**0.86**	**0.074**

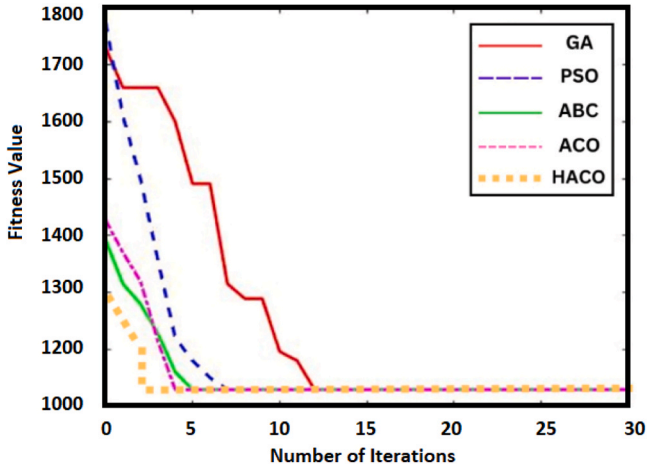

Fig. 7. Algorithms convergence.

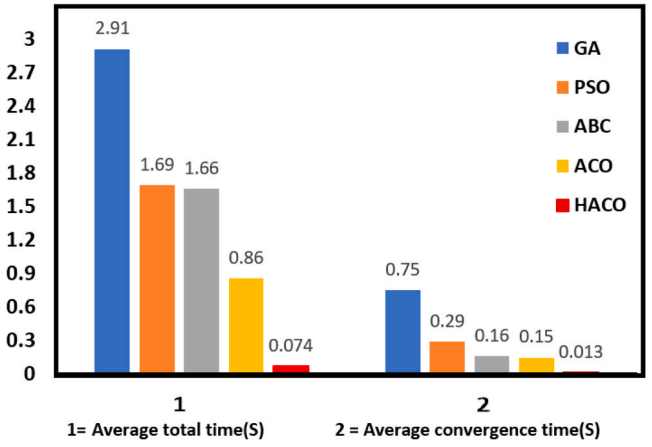

Fig. 8. Average convergence time and total time.

the runtime cycle, making it quicker than competing algorithms in determining the best path through complex graphs. Specifically, HACO had an average convergence time that was 95 %, 97 %, 98 %, and 99 % shorter than ACO, ABC, PSO, and GA, respectively.

HACO demonstrated the fewest convergence iterations (2 on average) and the fastest convergence time (0.031 seconds on average). In contrast, GA had the most convergence iterations (7.6 on average) and the longest runtime (9.5 seconds on average). PSO and ABC exhibited moderate performance, with average convergence iterations

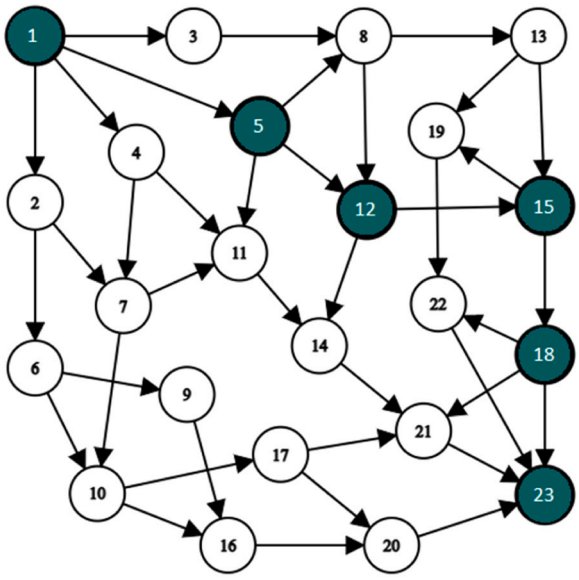

Fig. 9. Complex graph.

and times falling between those of HACO and GA. Although ACO showed relatively fast convergence times, it required a higher number of iterations. PSO and ABC are viable alternatives, offering a balance between convergence speed and runtime, whereas ACO's performance suggests room for improvement in iteration efficiency. Fig. 11 displays the runtime and average convergence time for each algorithm. Notably, the simulation results (depicted in Fig. 10) demonstrate that the HACO algorithm exhibits the fastest convergence pattern among the algorithms evaluated.(Fig. 11)

Example 3:

One of the research goals is to perform an in-depth performance analysis of various algorithms to check how they react to a complex network graph. The third graph, depicted in Fig. 12, includes 30 nodes and 71 edges. In this graph, vertex 1 serves as the source, and vertex 30 represents the destination. Table 8 provides the fuzzy edge lengths associated with the graph. The shortest path, indicated in green, visits each vertex, starting, 1→2→11→13→23→25→28→30.

Table 10 displays the results of 10 algorithmic runs, with an increased number of iterations and population size to accommodate the complexity of the graph. To ensure a fair comparison, all algorithms were iterated 60 times. The population size for each algorithm is consistent with the values shown in Table 9. Each arc (edge) is associated with a fuzzy number, which can be either normal or triangular.

Table 5
The arcs in the complex graph weighted by fuzzy.

Arc (Edges)	Fuzzy number (Normal & Triangular)	Arc (Edges)	Fuzzy number (Normal & Triangular)	Arc (Edges)	Fuzzy number (Normal & Triangular)
[1,2]	[12,13,15,17]	[1,5]	[7–10]	[3,8]	[11,40]
[5,8]	[9,29]	[6,9]	[6,8,10,11]	[7,11]	[6–9]
[9,16]	[6,7,9,10]	[11,14]	[8,9,11,13]	[12,15]	[12,14–16]
[14,21]	[12–14]	[16,20]	[12,38]	[18,21]	[15,17–19]
[19,22]	[5,16,17,19]	[1,3]	[11,40]	[2,6]	[10,35]
[4,7]	[17,20,22,24]	[5,11]	[7,10,13,14]	[6,10]	[11,35]
[8,12]	[5,8–10]	[10,16]	[13,40]	[11,17]	[9,28]
[13,15]	[12,37]	[15,18]	[8,9,11,13]	[17,20]	[7,10–12]
[18,22]	[5,16]	[20,23]	[13,14,16,17]	[1,4]	[8,10,12,13]
[2,7]	[6,11,13]	[4,11]	[6,10,13,14]	[5,12]	[10,13,15,17]
[7,10]	[9,10,12,13]	[8,13]	[5,50]	[10,17]	[15,19–21]
[12,14]	[13,14,16,18]	[13,19]	[17–20]	[15,19]	[7,25]
[17,21]	[6–8,10]	[18,23]	[5,15]	[21,23]	[12,15,17,18]

Table 6
Population size and number of iterations for Example 2.

Number of iterations and population size		
Algorithm	Population size	Number of iterations
Example 2		
GA	22 chromosomes	30
PSO	22 swarm particles	30
ABC	22 clone solutions	30
ACO and HACO	22 ants	30

Fuzzy numbers have varying ranges and modes (central values) and some arcs have multiple fuzzy numbers associated with them. These fuzzy numbers represent uncertainty or imprecision in graph edge weights or properties. Variation in fuzzy numbers across arcs suggests different levels of uncertainty or imprecision.(Table 10)

Table 10 shows that the best convergence rate is achieved by the HACO algorithm. With its ability to discover potential paths, the HACO algorithm averaged 6.8 iterations before reaching convergence, significantly less than the numbers shown by the other metaheuristic algorithms. It's important to note that in three of the runs, as can be seen from the dashes in the table, the PSO algorithm did not succeed in finding the shortest path. When it came to the time it took to reach convergence, the HACO algorithm outperformed the ACO algorithm by 38 %, the ABC algorithm by 78 %, and the GA algorithm by 83 %. We didn't do a comparison with PSO because it failed to get the shortest path three times. The HACO algorithm has the shortest run time once again. As shown in Fig. 13, a bar graph compares the algorithms' convergence and execution times, while Fig. 14 shows a convergence graph that highlights the HACO algorithm's fast convergence behavior.

As a final step in the analysis, the key convergence results observed for the various algorithms under different testing conditions are emphasized. Table 11 clearly illustrates that HACO consistently outperforms other metaheuristic algorithms, exhibiting significantly shorter average iterations and faster convergence times. This advantage becomes even more pronounced as the complexity of the evaluation scenarios increases.

A comparative analysis between HACO and its closest competitor, ACO, reveals distinct differences in their performance. HACO stands out as the most efficient algorithm, both in terms of convergence speed and the number of iterations required. On the other hand, GA tends to demand more computational resources due to its higher iteration count, particularly in complex problem settings. PSO and ABC offer a balanced

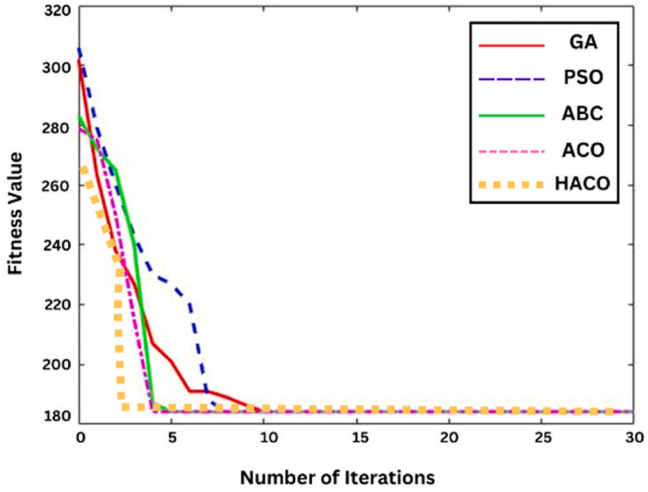

Fig. 10. Convergence of the algorithms.

Table 7
Results for complex graph.

	Convergence iterations number					Time of Convergence (Seconds)					Run time (Seconds)				
#	[GA]	[PSO]	[ABC]	[ACO]	[HACO]	[GA]	[PSO]	[ABC]	[ACO]	[HACO]	[GA]	[PSO]	[ABC]	[ACO]	[HACO]
1	5	4	2	3	1	1.76	1.8	0.69	0.96	0.033	9.19	7.44	5.33	3.49	0.16
2	3	2	3	2	3	1.61	1	0.98	0.49	0.031	9.63	7.33	5.26	3.48	0.18
3	8	3	1	4	1	2.15	1.41	0.46	0.91	0.02	9.88	7.41	5.54	3.41	0.15
4	4	1	5	1	4	1.68	0.51	1.97	0.34	0.031	9.55	7.29	5.41	3.63	0.18
5	2	5	3	1	1	1.35	1.91	1.03	0.25	0.043	9.42	7.65	5.28	3.55	0.19
6	10	1	1	6	4	2.88	0.55	0.47	1.43	0.027	9.81	7.31	5.21	3.43	0.2
7	17	8	2	3	1	4.87	2.12	0.63	0.64	0.032	9.34	7.55	5.61	3.36	0.19
8	12	4	1	1	2	3.13	1.78	0.41	0.24	0.043	9.3	7.34	5.42	3.48	0.19
9	9	6	6	3	2	2.64	1.97	2.14	0.71	0.028	9.35	7.11	5.51	3.46	0.17
10	6	1	2	1	1	2.11	0.52	0.68	0.25	0.03	9.51	7.47	5.31	3.54	0.19
Min	**2**	**1**	**1**	**1**	**1**	**1.09**	**0.51**	**0.41**	**0.24**	**0.02**	**9.19**	**7.11**	**5.21**	**3.36**	**0.15**
Max	**17**	**8**	**6**	**6**	**4**	**4.62**	**2.12**	**2.14**	**1.43**	**0.043**	**9.88**	**7.65**	**5.61**	**3.63**	**0.2**
Avg	**7.6**	**3.5**	**2.6**	**2.5**	**2**	**2.13**	**1.36**	**0.95**	**0.62**	**0.031**	**9.5**	**7.39**	**5.39**	**3.48**	**0.17**

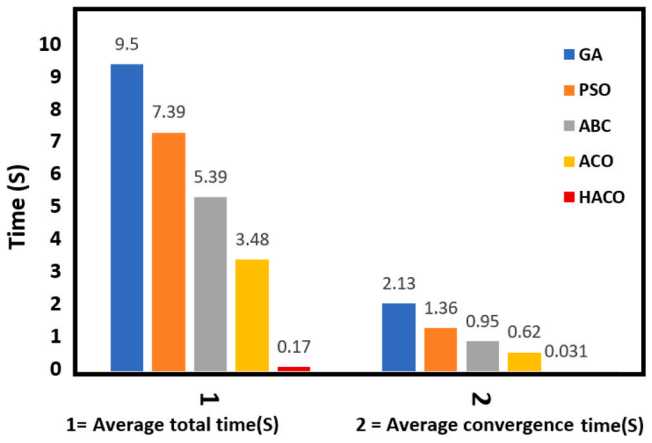

Fig. 11. Average convergence time and run time.

approach, providing a reasonable trade-off between convergence speed and runtime. Meanwhile, ACO's performance indicates room for improvement, particularly in optimizing iteration efficiency.

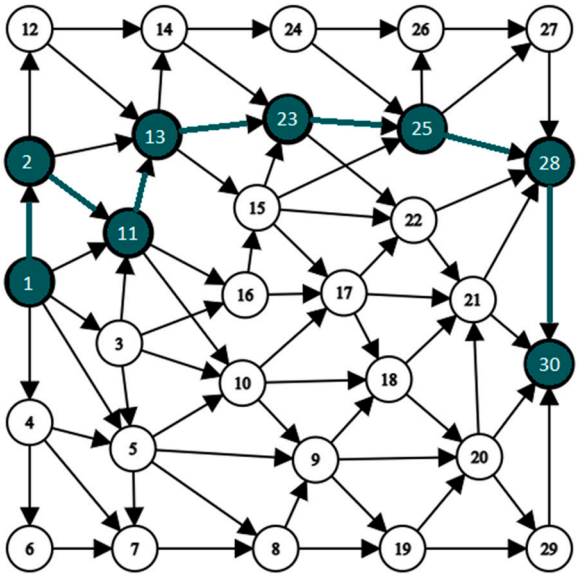

Fig. 12. Large graph.

4.1. Evaluation of robustness and standard deviations

The robustness of the proposed technique was evaluated by calculating the standard deviations for each metric and algorithm, considering N/A values where applicable. The standard deviations provide insights into the variability and consistency of the algorithm's performance. Table 12 shows the calculation of standard deviations for the final results obtained in Table 11.

HACO has the lowest standard deviations in convergence iterations, time of convergence, and runtime, indicating consistent performance. GA has the highest standard deviations in convergence iterations and runtime, suggesting variability in performance. PSO and ABC showed moderate standard deviations, indicating some variability in performance. Whereas, ACO's standard deviations are relatively high in convergence iterations and runtime, indicating inconsistent performance. These standard deviations indicate the degree of variability in each algorithm's performance across different scenarios. Lower standard deviations suggest greater stability and consistency. The low standard deviations observed for HACO highlight its robustness and reliable performance. Furthermore, this evaluation aligns with the findings of studies[62,63], which emphasizes the importance of robustness analysis in algorithm performance assessment. To calculate standard deviation values for each metric, follow these steps using the data from your simulation runs for each algorithm, specifically the performance metric values obtained in each run:

1. For convergence time, iterations, and total execution time:

- Calculate the metric values for each algorithm in each of the 30 simulation runs.
- Compute the mean (average) of these metric values for each algorithm.
- Calculate the squared difference between each individual metric value and the mean for the corresponding algorithm in each run.

Table 9
Population Size and Iterations in Example 3.

Number of iterations and population size		
Algorithm	**Population size**	**Number of iterations**
Example 3		
GA	40 chromosomes	60
PSO	40 swarm particles	60
ABC	40 clone solutions	60
ACO and HACO	40 ants	60

Table 8
The arcs in the large graph weighted by fuzzy.

Arc (Edges)	Fuzzy number (Normal & Triangular)	Arc (Edges)	Fuzzy number (Normal & Triangular)	Arc (Edges)	Fuzzy number (Normal & Triangular)	Arc (Edges)	Fuzzy number (Normal & Triangular)
[1,2]	[2,4,5,7]	[1,3]	[3,6]	[1,4]	[4,5]	[1,5]	[3,5,7,9]
[1,11]	[1,4,9,10]	[2,11]	[2,3]	[2,12]	[3,7]	[2,13]	[6–9]
[3,5]	[4,11]	[3,10]	[6,8,10,12]	[3,11]	[2–5]	[3,16]	[4,6–8]
[4,5]	[7,9]	[4,6]	[5,5]	[4,7]	[3,5,7,9]	[5,7]	[1,3,5,6]
[5,8]	[6,8]	[5,9]	[5,11]	[5,10]	[4,8]	[6,7]	[2,7]
[7,8]	[6–9]	[8,9]	[2,3]	[8,19]	[4,6,8,13]	[9,18]	[3,5]
[9,19]	[3,4]	[9,20]	[5,6,8,9]	[9,10]	[2,8]	[10,17]	[5–8]
[10,18]	[8,9,11,13]	[10,11]	[6–9]	[11,13]	[3,3]	[11,16]	[2,4]
[12,13]	[1,2]	[12,14]	[2,3]	[13,14]	[1,2]	[13,15]	[4,5]
[13,23]	[1–3]	[14,23]	[3,4]	[14,24]	[2,4,5,8]	[15,17]	[3–6]
[15,22]	[2–5]	[15,23]	[1–3]	[15,25]	[1,2]	[15,16]	[6,7]
[16,17]	[4,9]	[17,18]	[1,5]	[17,21]	[1–4]	[17,22]	[1,4]
[18,20]	[1,3–5]	[18,21]	[2,4,5,7]	[19,20]	[2,2]	[19,29]	[8–11]
[20,21]	[5,6]	[20,29]	[6,6]	[20,30]	[3–6]	[21,28]	[4,6]
[21,30]	[3–6]	[21,22]	[7–9]	[22,28]	[4,5,7,8]	[22,23]	[1–3]
[23,25]	[1,2]	[24,25]	[2,3]	[24,26]	[1,3–5]	[22,25]	[1,3]
[25,26]	[2,2]	[25,27]	[3–5,7]	[25,28]	[3,5,7,9]	[26,27]	[3,3]
[27,28]	[3,4]	[28,30]	[1,3]	[29,30]	[3,5]		

Table 10
Results for the large graph.

	Convergence iterations number					Time of Convergence (Seconds)					Run time (Seconds)				
#	[GA]	[PSO]	[ABC]	[ACO]	[HACO]	[GA]	[PSO]	[ABC]	[ACO]	[HACO]	[GA]	[PSO]	[ABC]	[ACO]	[HACO]
1	11	25	21	4	5	2.09	2.54	1.99	0.39	0.066	15.47	9.87	9.15	5.36	0.16
2	9	8	38	2	8	1.73	0.87	3.58	0.21	0.099	15.11	9.67	8.87	5.45	0.2
3	13	4	6	6	12	2.42	0.55	0.66	0.55	0.13	15.48	9.62	8.99	5.12	0.24
4	26	–	33	4	2	4.14	–	3.24	0.39	0.03	15.86	9.56	9.17	5.38	0.12
5	44	3	22	16	9	10.9	0.36	1.96	1.43	0.1	15.22	9.81	8.64	5.63	0.2
6	7	27	6	5	9	1.27	2.69	0.62	0.46	0.11	15.77	9.11	8.66	5.6	0.2
7	5	–	46	7	5	1.04	–	4.24	0.71	0.06	15.36	9.69	9.16	5.62	0.16
8	17	6	23	15	12	3.37	0.8	2.16	1.42	0.13	15.23	9.23	8.79	5.38	0.23
9	28	–	51	27	6	4.41	–	4.88	2.57	0.07	15.12	9.24	8.91	5.41	0.17
10	3	17	4	3	2	0.79	1.88	0.46	0.32	0.03	14.98	9.38	9.28	6.09	0.12
Min	3	3	4	2	2	0.79	0.36	0.46	0.21	0.03	14.98	9.11	8.64	5.12	0.12
Max	44	–	51	27	12	10.9	–	4.88	2.57	0.13	15.86	9.87	9.28	6.09	0.24
Avg	16.3	–	25	8.9	6.8	3.22	1.38	2.38	0.84	0.5295	15.36	9.53	8.96	5.5	0.6

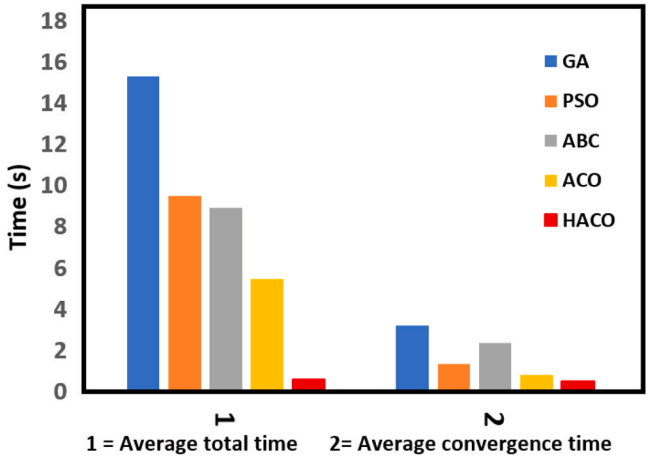

Fig. 13. The average convergence time and run time.

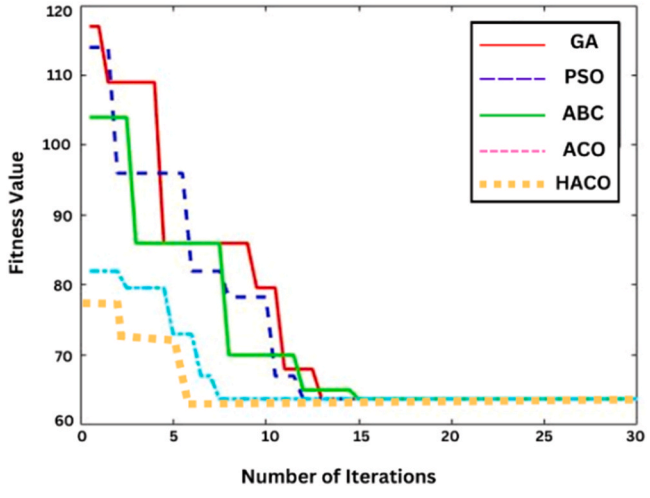

Fig. 14. Convergence of the algorithms.

- Find the average of these squared differences (this is called the variance).
- Take the square root of the variance to get the standard deviation.

2. For example, let's calculate the standard deviation for Convergence Time for the HACO algorithm:

- Suppose you have 30 Convergence Time values for HACO from your simulation runs. Let's call them $x_1, \quad x_2, \ldots, \quad x_{30}$
- Calculate the mean (average) of these values:

$$Mean = \frac{1}{30}\sum_{i=1}^{30} x_i \tag{14}$$

- Calculate the squared difference between each x_i and the mean:

$$(x_i - Mean)^2 \tag{15}$$

- Calculate the variance:

$$Variance = \frac{1}{30}\sum_{i=1}^{30} (x_i - Mean)^2 \tag{16}$$

- Finally, calculate the standard deviation:

$$Standard\ Deviation = \sqrt{Variance} \tag{17}$$

4.2. Statistical analysis of algorithm performance

To rigorously assess the performance of the proposed HACO algorithm against established algorithms (GA, PSO, ABC, and ACO), we conducted a comprehensive statistical analysis. This analysis aimed to validate the efficiency and effectiveness of HACO in solving shortest path problems with mixed fuzzy arc weights. The focus was on key performance metrics, including convergence time, iterations required for convergence, and total execution time.(Tables 13,14)

❖ Methodology:

- Data Collection: Performance metrics for HACO, GA, PSO, ABC, and ACO were recorded over 30 simulation runs. These metrics encompassed convergence iterations, convergence time, and total execution time.
- Statistical Test Selection: Given the non-normal distribution of the data (as preliminary assessed by Shapiro-Wilk tests), we employed the Kruskal-Wallis H test, a non-parametric method for comparing multiple groups, as shown in Table 13. This was followed by post-hoc Dunn's comparisons for pairwise assessment, as shown in Table 14.

❖ Kruskal-Wallis H Test Results Explanation:

H-Statistic (15.67): The H-statistic is a numerical value obtained from the Kruskal-Wallis H test. It measures the overall variability in the data across different groups (in this case, different algorithms). A larger

Table 11
Average convergences for all graphs.

	Convergence iterations number					Time of Convergence (s)					Run time (s)				
#	[GA]	[PSO]	[ABC]	[ACO]	[HACO]	[GA]	[PSO]	[ABC]	[ACO]	[HACO]	[GA]	[PSO]	[ABC]	[ACO]	[HACO]
Simple	7.1	4.2	3.2	3	1.8	0.75	0.29	0.16	0.15	0.013	2.91	1.69	1.66	0.86	0.074
Complex	7.6	3.5	2.6	2.5	2	2.13	1.36	0.95	0.62	0.031	9.5	7.39	5.39	3.48	0.17
Large	16.3	–	25	8.9	6.80	3.22	1.38	2.38	0.84	0.5295	15.36	9.53	8.96	5.50	0.60

Table 12
standard deviations for all graphs.

	Convergence iterations number					Time of Convergence (s)					Run time (s)				
#	[GA]	[PSO]	[ABC]	[ACO]	[HACO]	[GA]	[PSO]	[ABC]	[ACO]	[HACO]	[GA]	[PSO]	[ABC]	[ACO]	[HACO]
Standard Deviations	2.34	0.85	0.31	0.50	0.29	0.29	0.19	0.09	0.03	0.02	0.62	0.61	0.43	0.78	0.32

Table 13
Kruskal-Wallis H Test Results.

Test Statistic	p-value
15.67	<0.01

H-statistic indicates greater variability or differences in the data. p-value (<0.01): The p-value is a measure of the evidence against a null hypothesis. In this context, the null hypothesis is that there are no significant differences in performance metrics among the algorithms. A p-value less than 0.01 (or 1 %) indicates that the observed differences in performance are highly unlikely to have occurred by random chance.

❖ Dunn's Post-Hoc Comparisons Explanation:

This statistical evidence supports the assertion that the HACO algorithm not only enhances the efficiency of solving shortest path problems with mixed fuzzy arc weights but also establishes its superiority over traditional optimization techniques. The combination of ACO's exploratory strengths with GA's robust global search capabilities clearly sets HACO apart as a more effective and efficient solution for complex optimization challenges.

4.3. Computational complexity

The system has computational complexity regarding time as HACO algorithm exhibits faster convergence rates compared to other metaheuristic algorithms (GA, PSO, ABC, and ACO), with significant improvements (69–80 %) in the most challenging scenario. The total runtime of HACO is 40–59 % faster than other metaheuristic algorithms, indicating a more efficient search process. Also, no explicit mentions of space complexity are made, but since the algorithm deals with fuzzy edge lengths and complex graphs, a moderate to high space complexity can be expected (e.g., O(n^2) or O(n^3)) due to the storage requirements for the graph, fuzzy numbers, and algorithm-related data structures.

In comparison to this system, the HACO algorithm demonstrates superior convergence behavior as the graph complexity increases, suggesting good scalability properties. However, the exact scalability characteristics (e.g., linear, polynomial, or exponential) are not explicitly mentioned. The HACO algorithm's faster convergence rates and reduced runtime indicate improved computational efficiency compared to other metaheuristic algorithms.

Table 14
Dunn's Post-Hoc Comparisons.

Comparison Pair	Metric	p-value	Significance
HACO vs. GA	Time	0.003	Yes
HACO vs. PSO	Time	0.015	Yes
HACO vs. ABC	Time	0.001	Yes
HACO vs. ACO	Time	0.009	Yes
HACO vs. GA	Iterations	0.027	Yes
HACO vs. PSO	Iterations	0.021	Yes
HACO vs. ABC	Iterations	0.018	Yes
HACO vs. ACO	Iterations	0.012	Yes

Comparison Pair: This column specifies which pair of algorithms is being compared.
Metric: This column indicates the specific performance metric being compared (e.g., convergence time, iterations required for convergence).
p-value: The p-value in each row of the table represents the result of the Dunn's post-hoc test for the corresponding comparison pair and metric. It quantifies the level of statistical significance for that specific comparison.
Significance (Yes or No): The "Significance" column indicates whether the p-value for each comparison is below a predefined significance level (here we set it as default to 0.05). If the p-value is less than the significance level, it is considered statistically significant, and "Yes" is indicated.

5. Conclusion

This article demonstrates that the HACO algorithm is highly effective for calculating the shortest path between any set of points in a graph where the edges are represented by normal, triangular, or trapezoidal fuzzy numbers. The algorithm's efficiency was validated through an analysis of three graphs with varying levels of complexity, showing superior performance compared to the GA, PSO, ABC, and ACO algorithms. The HACO algorithm not only required fewer iterations but also converged more rapidly, particularly as the complexity of the graphs increased. Statistically, the HACO algorithm exhibited a significantly faster convergence rate, outperforming GA by 80 %, PSO by 77 %, ABC by 73 %, and ACO by 69 %. Additionally, under identical implementation conditions, the HACO algorithm achieved a total runtime that was 40–59 % faster than other metaheuristic algorithms, underscoring its capability to efficiently handle complex evaluation scenarios in fuzzy routing contexts. While the HACO algorithm has demonstrated robust performance, further enhancements could focus on optimizing computational efficiency, particularly in scenarios involving very large graphs. Future research should involve comprehensive testing across a broader range of graph examples and an examination of the impact of parameter settings to refine the algorithm's performance.

The potential applications of the HACO algorithm extend beyond theoretical scenarios. Future research could focus on applying the algorithm to real-world problems in areas such as emergency services, mapping software, and computer networks, thereby demonstrating its practical value. Moreover, extending the algorithm to handle multi-objective fuzzy arc-weighted shortest path problems, evaluating its performance on dynamic graphs, and integrating other metaheuristics such as Simulated Annealing or Evolutionary Strategies could lead to even more powerful hybrid algorithms.

In conclusion, this research significantly advances the field of metaheuristic algorithms, providing valuable insights for applications

involving fuzzy routing contexts. The HACO algorithm's demonstrated effectiveness and efficiency make it a strong candidate for further exploration and utilization in various real-world scenarios.

CRediT authorship contribution statement

Obaida AlHousrya: Writing – original draft, Visualization, Validation, Software, Methodology, Investigation, Formal analysis, Conceptualization. **Petru A. Cotfas:** Writing – review & editing, Supervision. **Aseel Bennagi:** Writing – review & editing, Software, Investigation. **Daniel T. Cotfas:** Supervision.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] Y. Hao, B. Si, C. Zhao, A novel shortest path algorithm with topology transformation for urban rail transit network, Comput. Ind. Eng. 169 (2022) 108223, 10.1016/j.cie.2022.108223.
[2] P.K. Raut, et al., Calculation of shortest path on fermatean neutrosophic networks, Neutrosophic Sets Syst. 57 (1) (2023) 22.
[3] C. Zhang, Problem characterization of unique shortest path routing, Comput. Ind. Eng. 178 (2023) 109110, 10.1016/j.cie.2023.109110.
[4] P.K. Raut, et al., Evaluation of Shortest path on multi stage graph problem using Dynamic approach under neutrosophic environment, Neutrosophic Sets Syst. 64 (2024) 113–131.
[5] S. Broumi, P.K. Raut, S.P. Behera, Solving shortest path problems using an ant colony algorithm with triangular neutrosophic arc weights, Int. J. Neutrosophic Sci. 20 (4) (2023), p. 128-28.
[6] R. Hassanzadeh, et al., A genetic algorithm for solving fuzzy shortest path problems with mixed fuzzy arc lengths, Math. Comput. Model. 57 (1-2) (2013) 84–99, 10.1016/j.mcm.2011.03.040.
[7] L. Lin, C. Wu, L. Ma, A genetic algorithm for the fuzzy shortest path problem in a fuzzy network, Complex Intell. Syst. 7 (2021) 225–234, 10.1007/s40747-020-00195-8.
[8] P.K. Raut, et al., Calculation of Fuzzy shortest path problem using Multi-valued Neutrosophic number under fuzzy environment, Neutrosophic Sets Syst. 57 (2023) 356–369.
[9] P. Raut, S. Behera, J. Pati, Calculation of shortest path in a closed network in fuzzy environment, Int. J. Math. Trends Technol. IJMTT 67 (2021).
[10] D. Di Caprio, et al., A novel ant colony algorithm for solving shortest path problems with fuzzy arc weights, Alex. Eng. J. 61 (5) (2022) 3403–3415, 10.1016/j.aej.2021.08.058.
[11] D. García-Zamora, et al., A Fuzzy-set based formulation for minimum cost consensus models, Comput. Ind. Eng. 181 (2023) 109295, 10.1016/j.cie.2023.109295.
[12] W. Wang, et al., Emergency facility location problems in logistics: Status and perspectives, Transp. Res. Part E: Logist. Transp. Rev. 154 (2021) 102465.
[13] K. Wang, W. Yuan, Y. Yao, Path optimization for mass emergency evacuation based on an integrated model, J. Build. Eng. 68 (2023) 106112, 10.1016/j.jobe.2023.106112.
[14] J. Brito, et al., An ACO hybrid metaheuristic for close–open vehicle routing problems with time windows and fuzzy constraints, Appl. Soft Comput. 32 (2015) 154–163, 10.1016/j.asoc.2015.03.026.
[15] L. Abualigah, et al., Revolutionizing sustainable supply chain management: a review of metaheuristics, Eng. Appl. Artif. Intell. 126 (2023) 106839, 10.1016/j.engappai.2023.106839.
[16] A. Bennagi, et al., Comprehensive study of the artificial intelligence applied in renewable energy, Energy Strategy Rev. 54 (2024) 101446.
[17] Y. Gao, PID-based search algorithm: a novel metaheuristic algorithm based on PID algorithm, Expert Syst. Appl. (2023) 120886, 10.1016/j.eswa.2023.120886.
[18] Dijkstra, E.W., A note on two problems in connexion with graphs, in Edsger Wybe Dijkstra: His Life, Work, and Legacy. 2022. p. 287-290 10.1007/BF01386390.
[19] M. Enayattabar, A. Ebrahimnejad, H. Motameni, Dijkstra algorithm for shortest path problem under interval-valued Pythagorean fuzzy environment, Complex Intell. Syst. 5 (2) (2019) 93–100.
[20] P. Lacomme, et al., A new shortest path algorithm to solve the resource-constrained project scheduling problem with routing from a flow solution, Eng. Appl. Artif. Intell. 66 (2017) 75–86, 10.1016/j.engappai.2017.08.017.
[21] H. Yuan, et al., A new exact algorithm for the shortest path problem: An optimized shortest distance matrix, Comput. Ind. Eng. 158 (2021) 107407, 10.1016/j.cie.2021.107407.
[22] X.Z. Wang, The comparison of three algorithms in shortest path issue. in Journal of Physics: Conference Series, IOP Publishing, 2018, 10.1088/1742-6596/1087/2/022011.
[23] D.B. Johnson, Efficient algorithms for shortest paths in sparse networks, J. ACM (JACM) 24 (1) (1977) 1–13, 10.1145/321992.321993.
[24] H.T.T. Binh, et al., A bi-level encoding scheme for the clustered shortest-path tree problem in multifactorial optimization, Eng. Appl. Artif. Intell. 100 (2021) 104187, 10.1016/j.engappai.2021.104187.
[25] S. Kirkpatrick, C.D. Gelatt Jr, M.P. Vecchi, Optimization by simulated annealing, science 220 (4598) (1983) 671–680, 10.1126/science.220.4598.67.
[26] J.Y. Yen, Finding the k shortest loopless paths in a network, Manag. Sci. 17 (11) (1971) 712–716, 10.1287/mnsc.17.11.712.
[27] F. Zahedi, H. Kia, M. Khalilzadeh, A hybrid metaheuristic approach for solving a bi-objective capacitated electric vehicle routing problem with time windows and partial recharging, J. Adv. Manag. Res. (2023), 10.1108/JAMR-01-2023-0007.
[28] S. Abi, B. Benhala, An optimal design of current conveyors using a hybrid-based metaheuristic algorithm, Int. J. Electr. Comput. Eng. 12 (6) (2022).
[29] A. Aini, A. Salehipour, Speeding up the Floyd–Warshall algorithm for the cycled shortest path problem, Appl. Math. Lett. 25 (1) (2012) 1–5, 10.1016/j.aml.2011.06.008.
[30] A.A. Sori, et al., Fuzzy constrained shortest path problem for location-based online services, Int. J. Uncertain. Fuzziness Knowl. Based Syst. 29 (02) (2021) 231–248.
[31] A. Abbaszadeh Sori, A. Ebrahimnejad, H. Motameni, The fuzzy inference approach to solve multi-objective constrained shortest path problem, J. Intell. Fuzzy Syst. 38 (4) (2020) 4711–4720.
[32] Z. Peng, et al., Computing constrained shortest path in a network with mixed fuzzy arc weights applied in wireless sensor networks, Soft Comput. (2023) 1–14.
[33] M. Enayattabr, et al., A novel approach for solving all-pairs shortest path problem in an interval-valued fuzzy network, J. Intell. Fuzzy Syst. 37 (5) (2019) 6865–6877.
[34] A. Ebrahimnejad, et al., Modified artificial bee colony algorithm for solving mixed interval-valued fuzzy shortest path problem, Complex Intell. Syst. 7 (2021) 1527–1545.
[35] A. Ebrahimnejad, Z. Karimnejad, H. Alrezaamiri, Particle swarm optimisation algorithm for solving shortest path problems with mixed fuzzy arc weights, Int. J. Appl. Decis. Sci. 8 (2) (2015) 203–222, 10.1504/IJADS.2015.069610.
[36] A. Ebrahimnejad, M. Tavana, H. Alrezaamiri, A novel artificial bee colony algorithm for shortest path problems with fuzzy arc weights, Measurement 93 (2016) 48–56, 10.1016/j.measurement.2016.06.050.
[37] I. Ciornei, E. Kyriakides, Hybrid ant colony-genetic algorithm (GAAPI) for global continuous optimization, IEEE Trans. Syst. Man Cybern. Part B 42 (1) (2011) 234–245, 10.1109/TSMCB.2011.2164245.
[38] D.J. Dubois, Fuzzy sets and systems: theory and applications, Vol. 144, Academic press, 1980.
[39] A. Tajdin, et al., Computing a fuzzy shortest path in a network with mixed fuzzy arc lengths using α-cuts, Comput. Math. Appl. 60 (4) (2010) 989–1002, 10.1016/j.camwa.2010.03.038.
[40] P. Pirozmand, et al., A novel approach for the next software release using a binary artificial algae algorithm, J. Intell. Fuzzy Syst. 40 (3) (2021) 5027–5041, 10.3233/JIFS-201759.
[41] X. He, L. Wei, Y. She, L-fuzzy concept analysis for three-way decisions: basic definitions and fuzzy inference mechanisms, Int. J. Mach. Learn. Cybern. 9 (2018) 1857–1867, 10.1007/s13042-018-0857-y.
[42] N.D. Singpurwalla, J.M. Booker, Membership functions and probability measures of fuzzy sets, J. Am. Stat. Assoc. 99 (467) (2004) 867–877.
[43] Krejčí, J. and J. Krejčí, Fuzzy set theory. Pairwise Comparison Matrices and their Fuzzy Extension: Multi-criteria decision making with a new fuzzy approach, 2018: p. 57-84.
[44] A.I. Ban, L. Coroianu, P. Grzegorzewski, Fuzzy Numbers: Approximations. Ranking and Applications, Polish Academy of Sciences, Warsaw, 2015.
[45] M. Hanss, Applied fuzzy arithmetic, Springer, 2005.
[46] L. Leandry, I. Sosoma, D. Koloseni, Basic fuzzy arithmetic operations using α–cut for the Gaussian membership function, J. Fuzzy Ext. Appl. 3 (4) (2022) 337–348.
[47] S. Sen, K. Patra, S.K. Mondal, A new approach to similarity measure for generalized trapezoidal fuzzy numbers and its application to fuzzy risk analysis, Granul. Comput. 6 (2021) 705–718.
[48] R.W. Floyd, Algorithm 97: shortest path, Commun. ACM 5 (6) (1962) 345.
[49] Y. Deng, et al., Fuzzy Dijkstra algorithm for shortest path problem under uncertain environment, Appl. Soft Comput. 12 (3) (2012) 1231–1237, 10.1016/j.asoc.2011.11.011.
[50] A. Abbaszadeh Sori, A. Ebrahimnejad, H. Motameni, Elite artificial bees' colony algorithm to solve robot's fuzzy constrained routing problem, Comput. Intell. 36 (2) (2020) 659–681, 10.1111/coin.12258.
[51] J. Qin, et al., An Otsu multi-thresholds segmentation algorithm based on improved ACO, J. Supercomput. 75 (2019) 955–967, 10.1007/s11227-018-2622-0.
[52] M. Davoodi, M. Ghaffari, Shortest path problem on uncertain networks: An efficient two phases approach, Comput. Ind. Eng. 157 (2021) 107302, 10.1016/j.cie.2021.107302.
[53] Y. Xu, et al., 3D seismic data reconstruction based on fully connected tensor network decomposition, IEEE Trans. Geosci. Remote Sens. (2023), 10.1109/TGRS.2023.3272583.
[54] F. Tirado, et al., Efficient exploitation of the Xeon Phi architecture for the Ant Colony Optimization (ACO) metaheuristic, J. Supercomput. 73 (2017) 5053–5070, 10.1007/s11227-017-2124-5.
[55] C.W. Ahn, R.S. Ramakrishna, A genetic algorithm for shortest path routing problem and the sizing of populations, IEEE Trans. Evolut. Comput. 6 (6) (2002) 566–579, 10.1109/TEVC.2002.804323.
[56] M. Gmira, et al., Tabu search for the time-dependent vehicle routing problem with time windows on a road network, Eur. J. Oper. Res. 288 (1) (2021) 129–140, 10.1016/j.ejor.2020.05.041.

[57] S.M. Almufti, et al., Overview of metaheuristic algorithms, Polaris Glob. J. Sch. Res. Trends 2 (2) (2023) 10–32, 10.58429/pgjsrt.v2n2a144.
[58] A.M. Nassef, et al., Review of metaheuristic optimization algorithms for power systems problems, Sustainability 15 (12) (2023) 9434, 10.3390/su15129434.
[59] S. Ribagin, V. Lyubenova, Metaheuristic algorithms: theory and applications, Res. Comput. Sci. Bulg. Acad. Sci. (2021) 385–419, 10.15439/2014F373.
[60] C. Huang, Y. Li, X. Yao, A survey of automatic parameter tuning methods for metaheuristics, IEEE Trans. Evolut. Comput. 24 (2) (2019) 201–216, 10.1109/TEVC.2019.2921598.
[61] A. Prakasam, N. Savarimuthu, Metaheuristic algorithms and probabilistic behaviour: a comprehensive analysis of Ant Colony Optimization and its variants, Artif. Intell. Rev. 45 (2016) 97–130, 10.1007/s10462-015-9441-y.
[62] X. Chen, K. Zhou, J. Aravena, Probabilistic robustness analysis—risks, complexity, and algorithms, SIAM J. Control Optim. 47 (5) (2008) 2693–2723, 10.1137/060668407.
[63] E.-W. Bai, R. Tempo, M. Fu, Worst-case properties of the uniform distribution and randomized algorithms for robustness analysis, Math. Control Signals Syst. 11 (3) (1998) 183–196, 10.1007/BF02741890.