CHAPTER 2

Fuzzy Numbers – Their Ranking Methods and Applications

Abstract

This chapter deals with two types of fuzzy numbers viz. Quasi-Gaussian fuzzy numbers and trapezoidal fuzzy numbers and their applications. A ranking method has been proposed for the Quasi-Gaussian fuzzy number called the link preference index and has been applied on the shortest path, minimum spanning tree and Steiner tree problem. The ranking methods available in the literature for the trapezoidal fuzzy numbers have been experimentally analyzed by applying on the constrained shortest path problem to determine the one which is most appropriate for practical applications. Also, an application of the trapezoidal fuzzy numbers has been shown for the wireless sensor networks. A max-min formulation has been presented for the orienteering problem to deal with the fuzzy version where the score values associated with the nodes and time values associated with the edges of the graphs are represented as trapezoidal fuzzy numbers. Further, to tackle large instances, a parallel formulation of the fuzzy orienteering problem algorithm has been suggested.

In section 2.1, the various graphs problems considered and the types of fuzziness that can be induced in a graph have been discussed. Section 2.2 presents the definition of the Quasi-Gaussian fuzzy number and trapezoidal fuzzy number, the formulas used for their mathematical operations and their applications. The max-min formulation and its parallel version for the fuzzy orienteering problem have been stated in section 2.3 and finally the conclusion of this chapter has been specified in section 2.4.

2.1 Introduction

Graphs are used to depict the pair wise relations between the objects from a collection. A graph can be represented as $G = (V, E)$ where V is the set of vertices and E is the set of edges connecting pair of vertices. In this chapter, we consider several graph problems which includes shortest path, minimum spanning tree, steiner tree, constrained shortest path and orienteering problem.

In a network, a path is an alternating sequence of vertices and edges connecting a source node and a destination node. In general, there can be more than one path connecting the source node and the destination node, the shortest path (SPP) is one where the sum of the weights assigned to the edges is minimum. The problem of finding the shortest path is of great importance in graph theory as it finds various real life applications like Communication, Computer Networks, Transport, Routing, Supply Chain Management etc. (Elizabeth & Sujatha, 2011; Hernandes, Lamata, Verdegay, & Yamakami, 2007). An example of finding the shortest path could be to determine the route between City A and City B on a road map with minimum travel time. Here the various connecting cities between City A and City B can be represented as vertices and their connecting road segments can be represented as edges connecting the vertices.

For a given undirected graph $G(V, E)$, many spanning trees are possible that are sub graphs of G connecting each of the vertices present in V without forming cycles. If the considered graph G is a weighted undirected graph, then a Minimum Spanning Tree (MST) can be generated which is one among the several spanning trees possible with the lowest total cost. One example where MST can be applied is that a cable TV company wants to lay out cable connections to a new neighbourhood. Thus, a path is to be determined that can connect the new location to the existing ones with least cost. Several other applications include network designing while laying out the telephone cables, in cluster analysis and in solving problems like travelling salesman problem, Steiner tree problem etc. (Bang, & Kun, 2006).

For a given undirected graph $G = (V, E, w)$ where w is the weight function mapping the set E to a set of non negative numbers and a set S where $S \subseteq V$, the Steiner Tree Problem (STP) is to determine a sub graph of G without cycles that spans S with lowest total cost of its edges (Kou, Markowsky, & Berman, 1981). Therefore, for a given set of objects, determining the shortest interconnect is a combinatorial optimization problem STP named after Jakob Steiner. The STP is a NP-Hard problem with two versions, one is the optimization problem where a minimum weight Steiner tree is to be found and the other is the decision problem where the task is to determine whether a Steiner tree of total weight at most K exists or not. This problem has several practical applications like in network designing or designing the circuit layout, in facility location problem, in the fields of wireless communication, computational biology etc.

Shortest Path Problem is one of the fundamental optimization problems in the field of Computer Science, Telecommunications and Operations Research that is polynomial time solvable. An extension of SPP is Constrained Shortest Path Problem (CSPP) where the shortest path computed is required to fulfill a set of constraints which leads to removal of those links from the graph that violates the constraints and then the Shortest Path algorithm is applied (Chen, Song, & Sahni, 2008; Xiao, Thulasiraman, Xue, & Juttner). Adding one or more constraints to the SPP makes it intractable and CSPP is a well-known NP-Complete Problem (Chen, Song, & Sahni, 2008). Applications of CSPP include computer networks, cable television networks, communication networks, transportation networks and multimedia applications like web broadcasting, video teleconferencing, remote diagnosis etc. (Dou, Zhu, & Wang, 2012; Xiao, Thulasiraman, Xue, & Juttner).

The orienteering problem (OP) is an NP-Hard problem, derived from the game of orienteering where the player is required to start from the initial control point and arrive at the final control point within the specified time limit, at the same time collect the rewards (score) assigned to each of the checkpoints that link the initial control point to the final control point. All those players who arrive at the final control point after the time expires are disqualified and the player reaching the final control point within the allotted time and with the maximum collected score is declared as the winner of the game. This game reflects many real life situations related to the field of logistics, tourism, building telecommunication networks, home delivery systems etc., which can be modelled as OP (Vansteenwegen, Souffriau, & Oudheusden, 2011). There are four types of OP which include the simple orienteering problem, team orienteering problem, orienteering problem with time window and team orienteering problem with time window. The OP can be observed as a combination of two well-known problems, namely the Travelling Salesman Problem (TSP) and the Knapsack Problem (KP) where the objective of maximizing the score is derived from KP and the objective of minimizing the time taken to travel from the initial control point and final control point is similar to the objective of TSP with the difference that in TSP all the vertices connecting the source to the target should be visited once but in OP it is not necessary to visit all the intermediate checkpoints (Vansteenwegen, Souffriau, & Oudheusden, 2011). One real life application of OP is in the tourism industry where tourists come to visit the cities which are rich in culture and traditions and have some historical significance. The aim is to visit as many locations (including palaces, museums, monuments, restaurants etc.) as possible within their duration of stay. Due to the restriction of time it is not possible to visit all the places. Therefore, depending upon the choice and taste of the tourist, the guide can assign priorities to the various locations to be visited and then explore as many of them as possible within the time frame available in order to make their trip as economical and as beneficial as possible (Schilde, Doerner, Hartl, & Kiechle, 2009).

In the real world the weights assigned to the edges, in each of the above stated problems correspond to parameters like cost, time, capacities, demand, distance, energy etc. which are represented as real numbers. However these parameters are not naturally precise and the uncertainty involved cannot be modelled using real numbers. Such a situation can be tackled by introducing the use of fuzzy numbers as they can model the uncertainty in a much better way and help in computing a more practical solution to the problems and provide a quality of service (QoS) assurance in problems like CSPP. In a graph, various types of fuzziness are possible which are given by Blue et al (2002) as:

- Type I: Fuzzy set of crisp graphs.
- Type II: Crisp vertex set and fuzzy edge set.
- Type III: Crisp vertices and edges with fuzzy connectivity.
- Type IV: Fuzzy vertex set and crisp edge set.

- Type V: Crisp graph with fuzzy weights.

Here, Type V graphs are considered where the weights assigned to the edges are represented as fuzzy numbers instead of crisp numbers. There are several types of fuzzy numbers viz. triangular, trapezoidal, exponential, quadratic, Gaussian, Quasi-Gaussian etc. In each of the above stated problems, it is important to rank the fuzzy numbers to determine their ordering as the requirement is to find the shortest interconnect in most of the cases taking into consideration certain constraints. In case of real numbers, there exists a natural ordering which is absent for fuzzy numbers. Therefore, many ranking methods convert the fuzzy numbers to real numbers and then analyze them but in this procedure much of the information is lost. Attempts have been made to determine techniques that can rank the fuzzy numbers with the minimum loss of information. The concept of ranking fuzzy numbers was first suggested by Jain (1976). Since then a lot of methods have been suggested by different researchers. Wang and Kerre (2001a; 2001b) categorized all the methods suggested for ranking the fuzzy numbers into three classes based on fuzzy mean and spread, fuzzy scoring and preference relation.

2.2 Fuzzy Numbers

Unlike Boolean logic where two extreme values i.e., true (0) and false (1) is considered, fuzzy logic is a matter of degree and considers the set of values lying within the interval [0, 1]. If U is the universal set, then a set X denoted as $X = \{(x, \mu_X(x)) : x \in U\}$ is called a fuzzy set where μ_X signifies the membership function. This membership function assigns a weight to every element x which is called the grade of membership and lies between 0 and 1. A fuzzy set X satisfying the following properties is called a fuzzy number (Bansal, 2011):

(a) μ_X is piecewise continuous.

(b) X is convex. $(\mu_X(\alpha x_1 + (1-\alpha)x_2) \geq min(\mu_X(x_1), \mu_X(x_2))$

$\forall\, x_1, x_2 \in \mathbb{R}, \forall\, \alpha \in [0,1])$

(c) X is normal. $(\mu_X(x_0) = 1 , x_0 \in \mathbb{R})$

where, $\mathbb{R}$ is the universal set of real numbers.

A fuzzy number is an extension of the crisp or real number which refers to a set of possible values represented by a membership function, each of which is assigned a weight called its grade of membership between 0 and 1 whereas the crisp number is associated with a single value.

It is used to represent those parameters which are imprecise and for which it is not possible to specify exact values. In these situations, it is appropriate to use a fuzzy number instead of a crisp number as they provide a more practical presentation of the real world. There are several types of fuzzy numbers, each of which can be defined by a membership function.

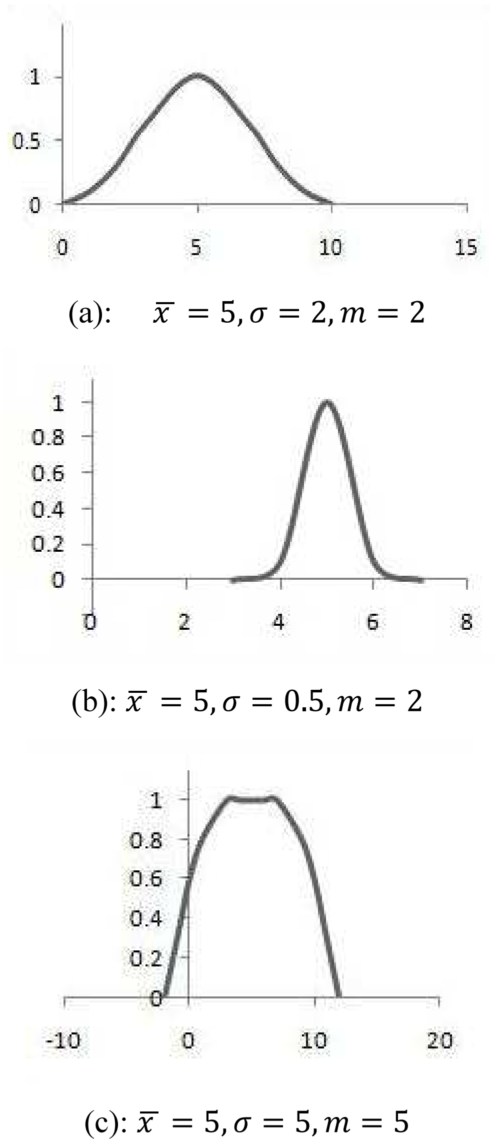
2.2.1 Quasi-Gaussian Fuzzy Number (QGFN)

All of the reported research work is based on piece-wise linear membership functions like Trapezoidal or Triangular (Elizabeth, & Sujatha, 2011). Although these membership functions are easy to analyze, they are unsuitable for applying gradient based parameter optimization methods (Dongrui, 2012). Gaussian membership function is simpler to represent since it requires fewer parameters. It is continuous and differentiable enabling efficient gradient based optimization in cases where actual measurement data is used to identify the parameters of membership functions while using parameter estimation and system identification techniques, continuity and differentiability and fewer parameters are highly desirable (Grauel, & Ludwig, 1999). Kashtiban *et al.* (2008) have used a recursive extended Kalman filter to obtain very good estimates of membership function from observed data. Their method also requires the membership function to be continuous and differentiable leading to a choice of Gaussian functions.

Quasi-Gaussian Fuzzy Number is a variation of Gaussian Fuzzy number. It is highly desirable to have a SPP, MST, STP etc. algorithms that model the uncertainty in link weights with Gaussian function expressed as

$$\mu(x;\ \bar{x},\ \sigma, m) = exp\left[-\frac{1}{2}\left|\frac{x-\bar{x}}{\sigma}\right|^{m}\right]$$

where, $\bar{x}$ is the centre, σ is the width and m is the fuzzification factor. Various shapes come into existence by varying the values of these parameters showing different degrees of fuzziness which are presented in Fig. 2.1. To model the uncertainty in some graph problems, here we use a variation of Gaussian membership function called Quasi-Gaussian membership function.



(a): $\bar{x} = 5, \sigma = 2, m = 2$

(b): $\bar{x} = 5, \sigma = 0.5, m = 2$

(c): $\bar{x} = 5, \sigma = 5, m = 5$

Fig. 2.1: Different shapes obtained by varying the value of fuzzification factor m

(I) Membership Function of QGFN

Fuzzy Numbers find a lot of practical applications and Gaussian fuzzy numbers can be applied in various fields. Hence, it is beneficial to limit the Gaussian fuzzy numbers and derive a new category called "Quasi-Gaussian Fuzzy Number" (QGFN). QGFN is a Gaussian fuzzy number with finite support i.e., the value of x beyond $\bar{x}$ - $3\sigma_l$ and $\bar{x}$ + $3\sigma_r$ is zero where $\bar{x}$ is the modal value, σ_l and σ_r denote the left and right spreads respectively corresponding to the Gaussian Distribution's standard deviation. We use Hanss (2005) notation as follows:

$$p = gfn^{*}\ (\bar{x}, \sigma_l, \sigma_r) \tag{2.1}$$

Here p is a QGFN and the membership function $\mu_p(x)$ is defined as: $\mu_p(x) =$

$$\begin{cases} 0 & for\, x \leq \bar{x} - 3\sigma_l \\ \exp\left[-\frac{(x-\bar{x})^2}{2\sigma_l^2}\right] & for\, \bar{x} - 3\sigma_l < x < \bar{x} \\ & \forall\, x \in R \\ \exp\left[-\frac{(x-\bar{x})^2}{2\sigma_r^2}\right] & for\, \bar{x} \leq x < \bar{x} + 3\sigma_r \\ 0 & for\, x > \bar{x} + 3\sigma_r \end{cases} \tag{2.2}$$

(II) Fuzzy Arithmetic using QGFN

Definition 1: Decomposed Fuzzy Numbers (DFN)

Fuzzy arithmetical operations like addition, subtraction etc. can be performed in several ways using Zadeh's extension principle, LR fuzzy numbers (Dubois, & Prade,1978; Dubois, & Prade 1979), discretized fuzzy numbers (Hanss,1999; Hanss, & Willner, 2000) and decomposed fuzzy numbers (Moore, 1966). Here we use decomposed fuzzy numbers which implements interval arithmetic.

A fuzzy set A can be represented as sequence of α-cuts. An α-cut (A_α) of A is defined as (Hanss, 2005):

$$A_\alpha = \alpha\text{-cut}\,(A) = \{x | \mu_A\,(x) \geq \alpha\} \qquad \text{where } \alpha \in [0,1]$$

To reduce the infinite number of α-cuts and make the decomposed fuzzy numbers usable for practical applications, the infinite set is reduced to a finite one by selecting some discrete values $\alpha_j = \mu_j$ for α (Hanss, 2005).

The finite set is created by dividing the interval [0,1] into k sub-intervals by applying the following formula (Hanss, 2005):-

$$\Delta\mu = \frac{1}{k}$$

Now the discrete values are

$$\mu_j = \frac{j}{k},\ j = 0,1, \dots \dots \dots \dots k.$$

with the properties

$$\mu_0 = 0, \mu_1 = 1,$$

$$\mu_{j+1} = \mu_j + \Delta\mu, \quad j = 0,1, \dots \dots \dots \dots k-1.$$

The parameter k which controls the degree of refinement is called the *decomposition number*. The decomposed form of the fuzzy number p_i that corresponds to the finite number of α-cuts is represented by the set

$$P_i = \left\{X_i^{(0)}, X_i^{(1)}, \dots \dots \dots \dots .., X_i^{(k)}\right\} \text{ of } (k+1)\text{intervals} \qquad (2.3)$$

Where $X_i^{(j)} = \left[a_i^{(j)}, b_i^{(j)}\right] = cut_{\mu_j}(p_i)$,

$$a_i^{(j)} \leq b_i^{(j)}, \qquad j = 1,2, \dots \dots \dots \dots \dots \dots k.$$

These $(k+1)$ intervals are called *intervals of confidence.*

Definition 2: Addition operation using DFN

If two decomposed fuzzy numbers $X_1^{(j)}$ and $X_2^{(j)}$ represented by the interval $\left[a_1^{(j)},\ b_1^{(j)}\right]$ and $\left[a_2^{(j)},\ b_2^{(j)}\right]$ respectively, are to be added to form the result $Z^{(j)}$ shown by the interval $\left[a^{(j)}, b^{(j)}\right]$ then

$$\left[a_1^{(j)}, b_1^{(j)}\right] + \left[a_2^{(j)}, b_2^{(j)}\right] = \left[\underbrace{a_1^{(j)} + a_2^{(j)}}_{a^{(j)}}, \underbrace{b_1^{(j)} + b_2^{(j)}}_{b^{(j)}}\right] \tag{2.4}$$

The concept of interval arithmetic induced by the decomposed fuzzy numbers is preferred over other methods of performing fuzzy arithmetic operations due to its less complex implementation (Hanss, 2005).

Definition 3: Minimum operation using DFN

The technique used for determining the minimum of two decomposed fuzzy numbers using the concept of lattice of fuzzy numbers and α-cuts is preferred over other ranking methods like those discussed in the paper by Ramli and Mohamad (2009) using the Centroid Index that falls under the category of fuzzy scoring techniques. These Centroid Indexes can be used for ranking of triangular and trapezoidal fuzzy numbers as most of it involves calculating the area under the curve which requires less number of calculations in case of triangular and trapezoidal fuzzy numbers and more calculations when Gaussian fuzzy numbers are involved, thereby increasing the complexity of the operations. Another reason for using this method of determining the minimum of two decomposed fuzzy numbers is that the Centroid methods fail to rank fuzzy numbers with the same centre values $(\bar{x})$ but different spread or width values (σ) but in our case this situation can be easily handled and the minimum of two fuzzy numbers can be generated as the concept of α-cuts is used here.

As stated by Klir and Yuan (1997), the set of real numbers R is linearly ordered and a pair of value p and q can be stated as either $p \leq q$ or $q \leq p$. Here, the lattice $(R, \leq)$ can be presented by an operation

$$min(p, q) = \begin{cases} p & if\ p \leq q \\ q & if\ q \leq p \end{cases} \qquad \text{For every } p, q \in R$$

However, this kind of linear ordering cannot be applied on fuzzy numbers and the minimum operation on two fuzzy numbers A, B is expressed as (A, B) .

Let S denote the set of all fuzzy numbers then the triple $\langle S, MIN, MAX\rangle$ is a distributive lattice in which MIN corresponds to *meet* and MAX corresponds to *join* operation (Klir, & Yuan, 1997). It is possible to define partial ordering on the lattice $\langle S, MIN, MAX\rangle$. The decomposed fuzzy numbers are represented in terms of α-cuts and these α-cuts can also be utilized in finding the partial ordering of the fuzzy numbers.

For any $A, B \in S$ and $\alpha \in [0,1]$ if the α-cut of A is expressed as A_α and that of B as B_α where $A_\alpha = [a_1, a_2]$ and $B_\alpha = [b_1, b_2]$. Then

$$MIN(A_\alpha, B_\alpha) = [min(a_1, b_1), min(a_2, b_2)]$$

Similarly, in case of decomposed fuzzy numbers $X_1^{(j)}\ and\ X_2^{(j)}$ represented by the interval $\left[a_1^{(j)}, b_1^{(j)}\right]$ and $\left[a_2^{(j)}, b_2^{(j)}\right]$ respectively, the minimum of the two fuzzy numbers can be represented as $Z_{min}^{(j)}$ stated as interval $\left[a^{(j)}, b^{(j)}\right]$ then

$$Z_{min}^{(j)} = \left[a^{(j)}, b^{(j)}\right] = \left[min\left(a_1^{(j)}, a_2^{(j)}\right), min\left(b_1^{(j)}, b_2^{(j)}\right)\right] \quad (2.5)$$

Definition 4: Link Preference Index (LPI)

The Link Preference Index defines the ranking order where each path length (Z_I) is compared with the fuzzy shortest path length (Z_{min}) which is shown in Fig. 2.2. The path (Z_I) with the highest LPI is preferred to all other paths and is the fuzzy shortest path i.e.,

$$Z_1 < Z_2\ \ iff\ \ \ LPI(Z_{min} < Z_1) > LPI(Z_{min} < Z_2)$$

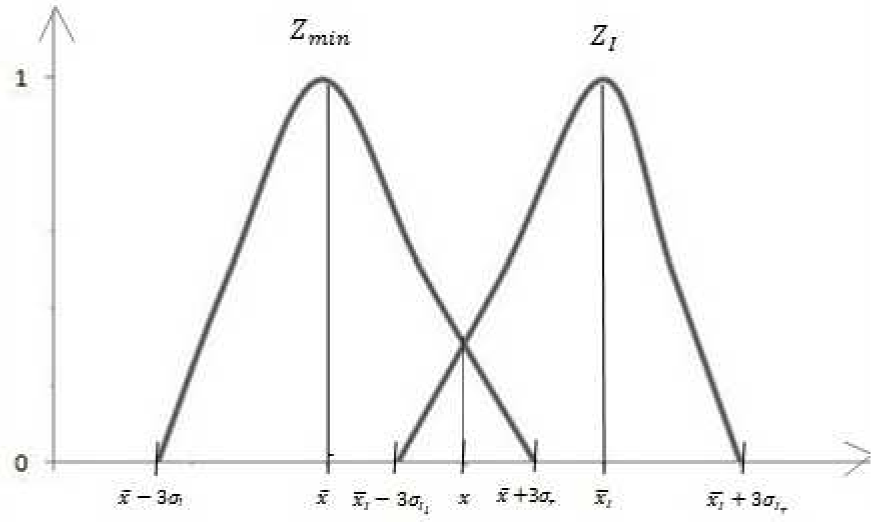


Fig. 2.2: Link Preference Index diagram

The formula for (LPI) using QGFN is derived as follows:-

$Z_{min} = gfn^*(\overline{x}, \sigma_l, \sigma_r)$ $\qquad$ $Z_I = gfn^*(\overline{x_I}, \sigma_{I_l}, \sigma_{I_r})$

Let μ_d be the membership function.

If $\bar{x} \leq x < \bar{x} + 3\sigma_r$

$\mu_d = exp[-(x-\bar{x})^2/2{\sigma_r}^2]$

$$=> \log\frac{1}{\mu_d} = \frac{(x-\bar{x})^2}{2{\sigma_r}^2}$$

$$=> 2{\sigma_r}^2\left(\log\frac{1}{\mu_d}\right) = (x-\bar{x})^2$$

$$=> \sigma_r\sqrt{2\left(\log\frac{1}{\mu_d}\right)} = x - \bar{x}$$

$$=> x = \left[\sigma_r\sqrt{2\left(\log\frac{1}{\mu_d}\right)} + \bar{x}\right] \tag{2.6}$$

If $\bar{x} - 3\sigma_{I_l} < x < \overline{x_I}$

$\mu_d = exp\left[-(x-\overline{x_I})^2/2{\sigma_{I_l}}^2\right]$

$$=> x = \left[\sigma_{I_l}\sqrt{2\left(\log\frac{1}{\mu_d}\right)} + \overline{x_I}\right] \tag{2.7}$$

Equating (2.6) and (2.7)

$$=> \sigma_r\sqrt{2\left(\log\frac{1}{\mu_d}\right)} + \bar{x} = \sigma_{I_l}\sqrt{2\left(\log\frac{1}{\mu_d}\right)} + \overline{x_I}$$

$$=> \bar{x} - \overline{x_I} = \sigma_{I_l}\sqrt{2\left(\log\frac{1}{\mu_d}\right)} - \sigma_r\sqrt{2\left(\log\frac{1}{\mu_d}\right)}$$

$$=> \bar{x} - \overline{x_I} = \sqrt{2\log\frac{1}{\mu_d}}\ \left(\sigma_{I_l} - \sigma_r\right)$$

$$=> \sqrt{2\left(\log\frac{1}{\mu_d}\right)} = \frac{(\bar{x} - \bar{x_I})}{\left(\sigma_{I_l} - \sigma_r\right)}$$

Squaring on both sides,

$$=> 2\left(\log\frac{1}{\mu_d}\right) = \left[(\bar{x} - \bar{x_I})/\left(\sigma_{I_l} - \sigma_r\right)\right]^2$$

$$=> \log\frac{1}{\mu_d} = \frac{1}{2}\left[(\bar{x} - \bar{x_I})/\left(\sigma_{I_l} - \sigma_r\right)\right]^2$$

$$\mu_d = exp\left[-(\bar{x} - \bar{x_I})^2/2\left(\sigma_{I_l} - \sigma_r\right)^2\right], \quad where \quad \sigma_{I_l} \neq \sigma_r \tag{2.8}$$

We take:

$$LPI = exp\left[-(\bar{x} - \bar{x_I})^2/2\left(\sigma_{I_l} - \sigma_r\right)^2\right] \qquad if\ \sigma_{I_l} \neq \sigma_r \tag{2.9}$$

The above stated formula can accurately rank the Quasi-Gaussian fuzzy numbers but cannot tackle situations where the left spread $\left(\sigma_{I_l}\right)$ of Z_I becomes equal to the right spread (σ_r) of Z_{min}. To deal with such a situation, we consider the following three cases:

$Case\ I$: $if\ \ \bar{x_I} - 3\sigma_{I_l} < \bar{x} + 3\sigma_r$

$$=> \bar{x_I} - \bar{x} < 3\left(\sigma_{I_l} + \sigma_r\right)$$

$$=> \bar{x_I} - \bar{x} = 3\varepsilon\left(\sigma_{I_l} + \sigma_r\right) \qquad where\ 0 < \varepsilon < 1$$

$$=> \sigma_{I_l} + \sigma_r = [\bar{x_I} - \bar{x}/3\varepsilon]$$

$Let\ (\bar{x_I} - \bar{x}) = \Delta\bar{x}$

$$=> \sigma_{I_l} + \sigma_r = [\Delta\bar{x}/3\varepsilon] \tag{2.10}$$

Squaring on both sides,

$$=> {\sigma_{I_l}}^2 + {\sigma_r}^2 + 2\sigma_{I_l}\sigma_r = [(\Delta\bar{x})^2/9\varepsilon^2]$$

$$=> {\sigma_{I_l}}^2 + {\sigma_r}^2 = [(\Delta\bar{x})^2/9\varepsilon^2] - 2\sigma_{I_l}\sigma_r \tag{2.11}$$

Substituting (2.11) in (2.9), we get,

$$LPI = exp\left[-(-\Delta\bar{x})^2/2\left(\sigma_{I_l} - \sigma_r\right)^2\right]$$

$$=> LPI = exp\left[-(\Delta\bar{x})^2/2\left({\sigma_{I_l}}^2 + {\sigma_r}^2 - 2\sigma_{I_l}\sigma_r\right)\right]$$

$$=> LPI = exp\left[\frac{-(\Delta\bar{x})^2}{2\left(\frac{(\Delta\bar{x})^2}{9\varepsilon^2} - 2\sigma_{I_l}\sigma_r - 2\sigma_{I_l}\sigma_r\right)}\right]$$

$$=> LPI = exp\left[\frac{-(\Delta\bar{x})^2}{2\left(\frac{(\Delta\bar{x})^2}{9\varepsilon^2} - 4\sigma_{I_l}\sigma_r\right)}\right] \tag{2.12}$$

Similarly we get the following equations for $Case\ II$ and $Case\ III$:

$Case\ II$: $if\ \ \bar{x_I} - 3\sigma_{I_l} = \bar{x} + 3\sigma_r$

$$LPI = exp\left[\frac{-(\Delta\bar{x})^2}{2\left(\frac{(\Delta\bar{x})^2}{9\varepsilon^2} - 4\sigma_{I_l}\sigma_r\right)}\right] \qquad where\ \varepsilon = 1 \tag{2.13}$$

$Case\ III$: $if\ \ \bar{x_I} - 3\sigma_{I_l} > \bar{x} + 3$

$$LPI = exp\left[\frac{-(\Delta\bar{x})^2}{2\left(\frac{(\Delta\bar{x})^2}{9\ \varepsilon^2} - 4\sigma_{I_l}\sigma_r\right)}\right] \qquad where\ \ \varepsilon > 1 \tag{2.14}$$

$$LPI = \begin{cases} & if\ \ \bar{x_I} - 3\sigma_{I_l} < \bar{x} + 3\sigma_r\ \ then\ choose\ \ 0 < \varepsilon < 1 \\ exp\left[\frac{-(\Delta\bar{x})^2}{2\left(\frac{(\Delta\bar{x})^2}{9\varepsilon^2} - 4\sigma_{I_l}\sigma_r\right)}\right] & if\ \ \bar{x_I} - 3\sigma_{I_l} = \bar{x} + 3\sigma_r\ \ then\ choose\ \ \varepsilon = 1 \\ & if\ \ \bar{x_I} - 3\sigma_{I_l} > \bar{x} + 3\sigma_r\ \ then\ choose\ \ \varepsilon > 1 \end{cases} \tag{2.15}$$

(III) Applications of QGFN

(a) Fuzzy Shortest Path Problem (FSPP)

The fuzzy shortest path is computed using link preference index (LPI).

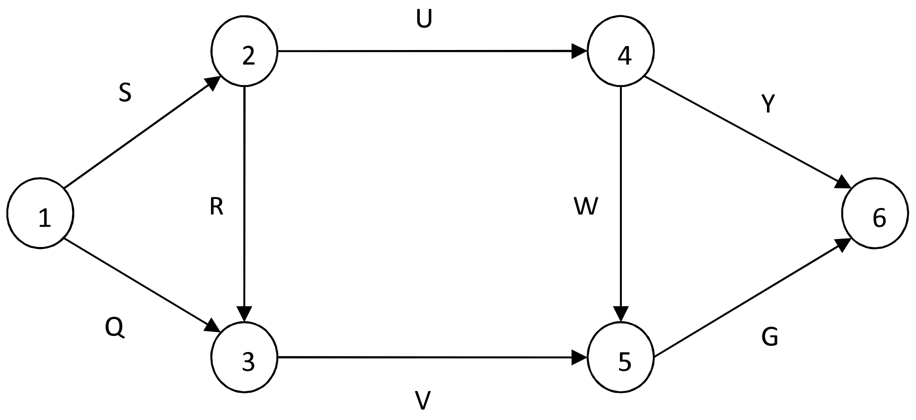


Fig. 2.3: Network for FSPP

The following steps are applied on the network shown in Fig. 2.3.

Step 1: Construct a Network $G = (V, E)$.

The above figure shows a network G with $|V| = 6 \; and \; |E| = 8$ where $V \; and \; E$ are the set of vertices and set of edges respectively. The edge weights in the form of QGFN are as follows:

$S = gfn^*[9,3,2]$ $\qquad$ $Q = gfn^*[20,6,4]$

$R = gfn^*[10,2,2]$ $\qquad$ $U = gfn^*[15,5,5]$

$V = gfn^*[14,4,5]$ $\qquad$ $W = gfn^*[8,1,2]$

$Y = gfn^*[16,3,4]$ $\qquad$ $G = gfn^*[20,6,4]$

Step 2: Find all possible paths (P_I), from source vertex to the destination vertex.

In the considered network G four paths are possible from the source vertex 1 to the destination vertex 6 which are as follows:

$P_1 = S + U + W + G$ $\qquad$ $P_2 = S + U + Y$

$P_3 = S + R + V + G$ $\qquad$ $P_4 = Q + V + G$

Step 3: Convert the weights (QGFN) of the edges present in the path (P_I) to DFN, using Definition 1 of section 2.2.1 (II)

The above stated edge weights are now converted into decomposed fuzzy numbers which are shown below:

Let the value of $K = 2$, and then the following intervals are generated:

$S = \{[0,15]^0, [5.5,11.5]^{0.5}, [9,9]^1\}$

$Q = \{[2,32]^0, [13,25]^{0.5}, [20,20]^1\}$

$R = \{[4,16]^0, [7.5,12.5]^{0.5}, [10,10]^1\}$

$U = \{[0,30]^0, [9,21]^{0.5}, [15,15]^1\}$

$V = \{[2,29]^0, [9,20.5]^{0.5}, [14,14]^1\}$

$W = \{[4,14]^0, [6.5,10.5]^{0.5}, [8,8]^1\}$

$Y = \{[7,28]^0, [12.5,21]^{0.5}, [16,16]^1\}$

$G = \{[1,22]^0, [4.5,13]^{0.5}, [7,7]^1\}$

Step 4: Compute the path length (Z_I) for each of the paths (P_I) found using Definition 2 of section 2.2.1 (II) $(I = 1,2 \dots \dots \dots \dots, n)$.

Let the path lengths be denoted by Z_1, Z_2, Z_3, Z_4 of paths P_1, P_2, P_3, P_4 respectively. As per the Definition 2 of section 2.2.1 (II), the following path lengths are generated:

$Z_1 = \{[5,81]^0, [25.5,56]^{0.5}, [39,39]^1\}$

$Z_2 = \{[7,73]^0, [27,53.5]^{0.5}, [40,40]^1\}$

$Z_3 = \{[7,82]^0, [26.5,57.5]^{0.5}, [40,40]^1\}$

$Z_4 = \{[5,83]^0, [26.5,58.5]^{0.5}, [41,41]^1\}$

Step 5: Compute the fuzzy shortest path length (Z_{min}) using Definition 3 of section 2.2.1 (II).

Using Definition 3 of section 2.2.1 (II), the FSPL (Z_{min}) is calculated considering Z_1, Z_2, Z_3 *and* Z_4.

$Z_{min} = \{[5,73]^0, [25.5,53.5]^{0.5}, [39,39]^1\}$

Step 6: Convert $Z_I \;\; for\; I = 1,2, \dots \dots \dots \dots, n$ and Z_{min} to QGFN.

Now the path lengths $Z_1, Z_2, Z_3, Z_4\, and\, Z_{min}$ are converted back from decomposed fuzzy numbers to QGFN.

$Z_1 = gfn^*[39,11,15]$

$Z_2 = gfn^*[40,11,11]$

$Z_3 = gfn^*[40,10,14]$

$Z_4 = gfn^*[41,13,15]$

$Z_{min} = gfn^*[39,11,12]$

Step 7: Calculate the LPI between Z_{min} and $Z_I \quad for\ I = 1,2, \dots \dots \dots .., n$ using Definition 4 of section 2.2.1 (II). Assign ranks to each of the paths (P_I).

Using Eq. 2.15, we get the following LPI and ranks:

Table 2.1: LPI value and Ranks of different paths

Path	Equation	LPI Value	Rank
$P_1: 1-2-4-5-6$	$LPI(Z_{min} < Z_1)$	1	1
$P_2: 1-2-4-6$	$LPI(Z_{min} < Z_2)$	0.98	2
$P_3: 1-2-3-5-6$	$LPI(Z_{min} < Z_3)$	0.97	3
$P_4: 1-3-5-6$	$LPI(Z_{min} < Z_4)$	0.136	4

Step 8: Identify the fuzzy shortest path as the one with the highest LPI.

The different paths along with their ranks are shown in Fig. 2.4.

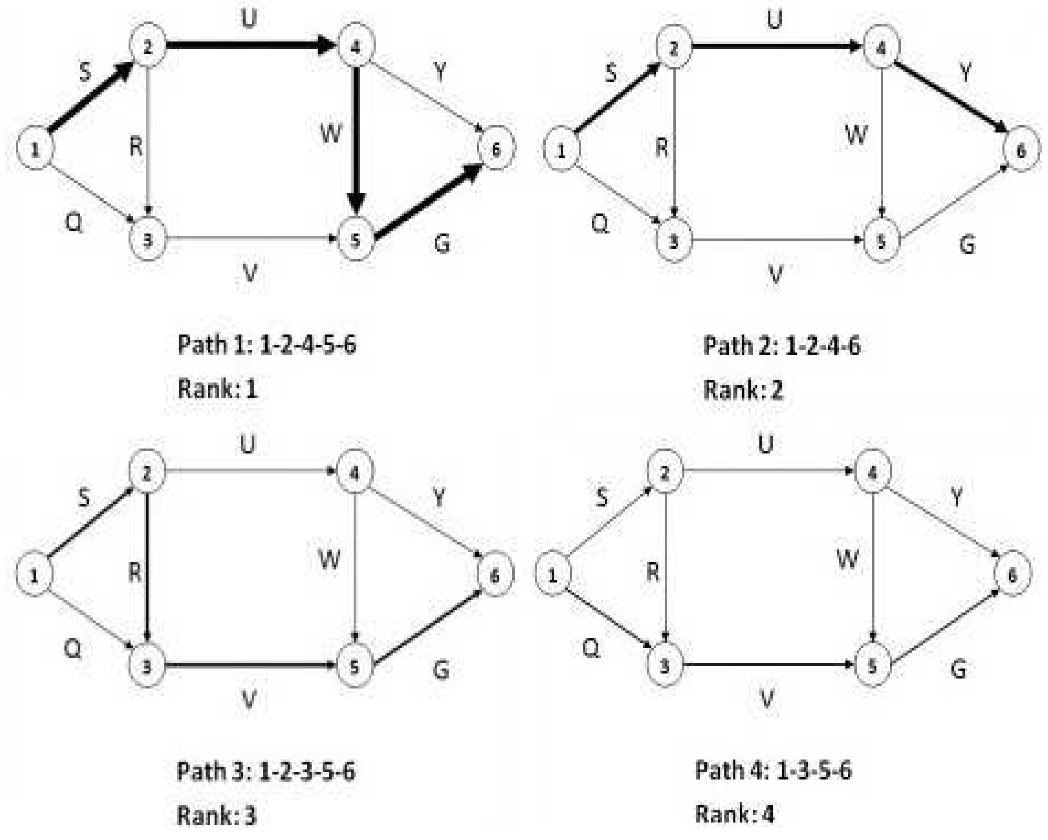


Fig. 2.4: Ranking of paths in the network

The fuzzy shortest path in the network considered is P_1 which has the highest LPI value and hence the greatest rank 1. If all the weight spreads are set to zero, the QGFN is reduced to its crisp value and then it can be easily determined that the shortest path is P_1 which is the same as generated using LPI. The two path lengths Z_2 and Z_3 have the same modal values 40. Hence, using the centroid method, we cannot rank the two path lengths but LPI has the advantage of ranking these two path lengths.

Extended Dijkstra's Algorithm using Fibonacci heap and Quasi-Gaussian Membership Functions

```
Fuzzy_Dijkstra $(G, s, t)$

1   for each vertex $v$ in $G$:
```

```
2        $LPI[v] \leftarrow -\infty$ ;
// Initialization of the array storing the Link Preference Index (LPI).
3    $\boldsymbol{end\ for}$
4    $LPI[s] \leftarrow 0$ ;                  // s is the source vertex.
5    $\boldsymbol{INSERT}\ (Q, v, LPI[v])$ ;
// Initialize the Priority Queue Q and function INSERT ( ) is used to update the queue.
6    $V_T \leftarrow \emptyset$ ;
// $V_T$ is an array storing the sequence of nodes in the shortest path.
7    $\boldsymbol{while}\ u \neq t$:                  // t is the destination vertex.
8        $u \leftarrow$ vertex in $Q$ with greatest $LPI[\,]$ ;
9            $\boldsymbol{if}\ LPI[u] = -\infty$
10               $\boldsymbol{break}$ ;
11           $\boldsymbol{end\ if}$
12       $u^* \leftarrow \boldsymbol{DELETE\_MAX}(Q)$;
// DELETE_MAX ( ) is used to remove the vertex with the greatest LPI value from the priority queue Q.
13       $V_T \leftarrow V_T \cup \{u^*\}$ ;
14           $\boldsymbol{for}$ each neighbour $v$ of $u$:
15               $val \leftarrow \boldsymbol{LPI\_between}(V_T, v)$ ;
// LPI_between ( ) is a function that determines Link Preference Index (LPI) using Eq. (2.15) .
16                   $\boldsymbol{if}\ val > LPI[v]$
17                       $LPI[v] \leftarrow val$ ;
18                       $\boldsymbol{INCREASE}(Q, v, LPI[v])$ ;
// INCREASE ( ) function updates the value of LPI for vertex v in the priority queue Q.
19                   $\boldsymbol{end\ if}$
20               $\boldsymbol{end\ for}$
21   $\boldsymbol{end\ while}$
22   $\boldsymbol{return}\ V_T$              // the sequence of nodes in the shortest path is returned.
23 end Fuzzy_Dijkstra
```

Discussion

The above stated algorithm is an extended version of the Dijkstra's algorithm that generates the shortest path for a source-destination pair. If one requires all the shortest paths emanating from the source, the above algorithm can be rerun using different targets or destination nodes. In steps 1-3, the array maintaining the value of LPI for each vertex is initialized with $-\infty$. In the next step, for the vertex chosen as source the value of LPI is replaced with a zero. In step 5, the ***INSERT ()*** function is used to initialize the priority queue Q with the LPI value of each vertex. In step 6, an array V_T , used to store the sequence of nodes in the shortest path is initialized. From step 7-21, a while loop runs inside which u is a variable that holds the vertex with the maximum value of LPI in Q and ***DELETE_MAX ()*** is a function used to delete that vertex from Q and store it in variable u^* which is then placed in the array V_T. Then in step 14-15, for each neighbour v of vertex u, the new LPI value is calculated using the function $\boldsymbol{LPI\_between}$ (V_T, v) that uses Eq. 2.15. For the sequence of nodes in V_T, first the weights of the edges connecting them is added then that value is compared with the weights of the edges connecting the neighbouring vertices v of u. This new LPI value of vertex v is saved in the variable val. In steps 16-17, we compare to see if the new LPI value of v is greater than the previous one stored in the array LPI, if yes the value LPI $[v]$ is updated and the ***INCREASE ()*** function is used to update the LPI value of vertex v in Q. In step 21 the while loop is ended when $u = t$ where t is the specified destination vertex. Finally, in step 22 the array V_T is returned which stores the shortest path.

When the priority queue Q is implemented as an ordinary array or a linked list, the running time of the algorithm is $O(|V|^2)$. When an adjacency list is used for storing a graph and a Fibonacci heap is used for implementing the priority queue Q, the running time of the algorithm is $O(|E| + |V|\log|V|)$ (Cormen, Leiserson, Rivest, & Stein, 1990). Also, since exponent is a monotonic function of its argument, the algorithm can still work if in step 15 of the above algorithm, exponentiation is avoided and only the argument values are used.

(b) Fuzzy Minimum Spanning Tree Problem (FMST)

The most commonly used algorithms to solve the MST problem are two greedy algorithms that run in polynomial time called Prim's (1957) and Kruskal (1956) algorithm. Here, the extended Prim's algorithm using QGFN and LPI is introduced and an illustrative example is presented to show the application of the algorithm.

```
$\boldsymbol{Fuzzy\_MST\_PRIM(G, w, r)}$

1  $\boldsymbol{for}$ each $u \in V[G]$
2        $\boldsymbol{do}$ $Key[u] \leftarrow -\infty$
// Initialization of the array that stores the LPI value for each node.
3           $\pi[u] \leftarrow NIL$
// Initialization of the array that stores the parent of each node.
4  $\boldsymbol{end\ for}$
5  $Key[r] \leftarrow 0$                 // the value for root is initialized.
6  $Q \leftarrow V[G]$                   // Priority queue is initialized
7  $\boldsymbol{while}$ $Q \neq \emptyset$
8        $\boldsymbol{do}$ $u \leftarrow \boldsymbol{EXTRACT\_MAX}(Q)$
// Function to extract the node with maximum LPI value from Q.
9            $\boldsymbol{for}$ $each\ v \in Adj[u]$
10                $\boldsymbol{do\ if}$ $v \in Q\ and\ \boldsymbol{LPI}(u, v) > Key[v]$
11                    $\boldsymbol{then}$ $\pi[v] \leftarrow u$
12                         $Key[v] \leftarrow \boldsymbol{LPI}(u, v)$
// The new LPI value of the selected nodes are calculated using Eq. (2.15).
13                $\boldsymbol{end\ if}$
14           $\boldsymbol{end\ for}$
15  $\boldsymbol{end\ while}$
```

The above stated algorithm can be implemented using adjacency matrix in $O(|V|^2)$ time, using binary heap and adjacency list in $O(|E|\log|V|)$time and using Fibonacci heap and adjacency list in $O(|E| + |V|\log|V|)$ time.

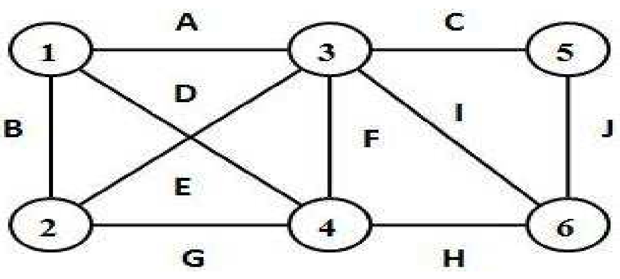


Fig. 2.5: Network for FMST

Step 1: Construct a Network $G = (V, E)$.

Fig. 2.5 shows a network G with $|V| = 6 \; and \; |E| = 10$ where $V \; and \; E$ are the set of vertices and set of edges respectively. The edge weights in the form of QGFN are as follows:

$A = gfn^*[6,2,1]$ $\qquad$ $F = gfn^*[20,5,5]$

$B = gfn^*[10,3,2]$ $\qquad$ $G = gfn^*[8,2,5]$

$C = gfn^*[15,4,5]$ $\qquad$ $H = gfn^*[12,4,4]$

$D = gfn^*[18,1,1]$ $\qquad$ $I = gfn^*[11,3,6]$

$E = gfn^*[5,1,2]$ $\qquad$ $J = gfn^*[22,3,5]$

Step 2: Find the minimum edge weight (Z_{min}) to which all other edge weights will be compared and a LPI value will be calculated by converting the weights (QGFN) to DFN using Definition 1 of section 2.2.1 (II). Let $K = 2$.

$A = \{[0,9]^0, [3.65,7.18]^{0.5}, [6,6]^1\}$

$B = \{[1,16]^0, [6.47,12.35]^{0.5}, [10,10]^1\}$

$C = \{[3,30]^0, [10.29, 20.89]^{0.5}, [15,15]^1\}$

$D = \{[15,21]^0, [16.82,19.18]^{0.5}, [18,18]^1\}$

$E = \{[2,11]^0, [3.82,7.35]^{0.5}, [5,5]^1\}$

$F = \{[5,35]^0, [14.11,25.89]^{0.5}, [20,20]^1\}$

$G = \{[2,23]^0, [5.65,13.89]^{0.5}, [8,8]^1\}$

$H = \{[0,24]^0, [7.29,16.71]^{0.5}, [12,12]^1\}$

$I = \{[2,29]^0, [7.47,18.06]^{0.5}, [11,11]^1\}$

$J = \{[13,37]^0, [18.47,27.89]^{0.5}, [22,22]^1\}$

Step 3: We initialize the two arrays storing the parent of each node and its LPI value as per the $\boldsymbol{Fuzzy\_MST\_PRIM}$ algorithm (as shown in Table 2.2). According to Step 5 of the algorithm let the root be Node 1. Thus, $Key[1] \leftarrow 0$ after which the two arrays store the values as shown in Table 2.3. Let $Q = \{1,2,3,4,5,6\}$

As per the $\boldsymbol{Fuzzy\_MST\_PRIM}$ algorithm the node in Q with the highest Key value is extracted and assigned to u. So, now, $u = 1$.

Next the neighbours of u present in Q are considered and the LPI value between them is calculated using Eq. (2.15).

After considering the neighbours of Node 1, Table 2.4 presents the values stored in the array.

Step 4: Next the node with the greatest key value is selected among the existing nodes in Q and the node extracted is, $u = 3$ and the status of the two arrays after calculation is shown in Table 2.5.

The loop continues till the priority queue Q becomes $\boldsymbol{EMPTY}$. Finally the array denoted by π returns the connections of the FMST as depicted in Table 2.6 and the final Fuzzy Minimum Spanning Tree is shown in Fig. 2.6.

Table 2.2: Initial values of the two arrays

Node	1	2	3	4	5	6
$\pi[u]$	NIL	NIL	NIL	NIL	NIL	NIL
$Key[u]$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$

Table 2.3: Initializing the LPI value of root node

Node	1	2	3	4	5	6
$\pi[u]$	NIL	NIL	NIL	NIL	NIL	NIL
$Key[u]$	0	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$

Table 2.4: Status of the two arrays after the neighbours of root are explored

Node	1	2	3	4	5	6
$\pi[u]$	NIL	1	1	1	NIL	NIL
$Key[u]$	0	0.3800	0.9685	$3.3358e - 86$	$-\infty$	$-\infty$

Table 2.5: Status of the two arrays after exploring the next node in the priority queue Q

Node	1	2	3	4	5	6
$\pi[u]$	NIL	3	1	3	3	3
$Key[u]$	0	1	0.9685	0.154	0.018	0.106

Table 2.6: Final values returning the connections in MST

Node	1	2	3	4	5	6
$\pi[u]$	NIL	3	1	2	3	3
$Key[u]$	0	1	0.9685	0.688	0.018	0.106

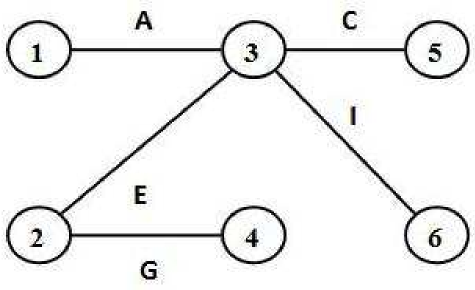


Fig. 2.6: Fuzzy Minimum Spanning Tree for the given network

(c) Fuzzy Steiner Tree Problem (FSTP)

The algorithm for FSTP considers a connected undirected distance graph $G = (V, E, d)$ and a set S where $S \subseteq V$. Here V is the set of vertices, E is the set of edges and d is the distance function which is represented as a QGFN and denotes the distance between a pair of vertices$(v_i, v_j) \in V$. To determine the final Steiner tree, a complete undirected distance sub graph $G_1 = (V_1, E_1, d_1)$ is constructed from the given two inputs $G \, and \, S$ such that now $V_1 = S$ and $d_1(v_i, v_j)$ of the edge $(v_i, v_j) \in E_1$ is equal to the fuzzy shortest path from $v_i \, to \, v_j$ in G.

Input: An undirected distance graph $G = (V, E, d)$ and a set of Steiner points $S \subseteq V$.

Output: A Fuzzy Steiner Tree (T_{FS}) for $G \, and \, S$.

Step 1: Construct the fuzzy complete undirected distance graph $G_1 = (V_1, E_1, d_1)$ from $G \, and \, S$.

Step 2: To construct G_1 consider each vertex $v_i \in S$ and find out the fuzzy shortest path to all other vertices $v_j \in S$ among all possible paths from v_i to $v_j \in G$ using Definition 2, 3 of section 2.2.1 (II) and Eq. (2.15).

Step 3: Find the fuzzy minimal spanning tree $T_1\ of\ G_1$ using the following extended Prim's algorithm for Fuzzy MST (if there are several fuzzy minimum spanning trees, choose any arbitrary one).

Step 4: Construct the sub graph $G_s\ of\ G$ by replacing each edge in T_1 by its corresponding fuzzy shortest path in G (if there are more than one fuzzy shortest paths then choose any arbitrary one).

Step 5: Find the Fuzzy MST $(T_s\ of\ G_s)$ using the $\boldsymbol{Fuzzy\_MST\_PRIM}$ algorithm (if there are several Fuzzy MST then choose any arbitrary one).

Step 6: Construct the Fuzzy Steiner Tree (T_{FS}) from T_s by deleting the edges in T_s, if necessary so that all the leaves in T_{FS} are steiner points and not the terminal nodes.

The above stated heuristic algorithm for the FSTP runs in $O(|S||V|^2)$ time.

An Illustrative Example

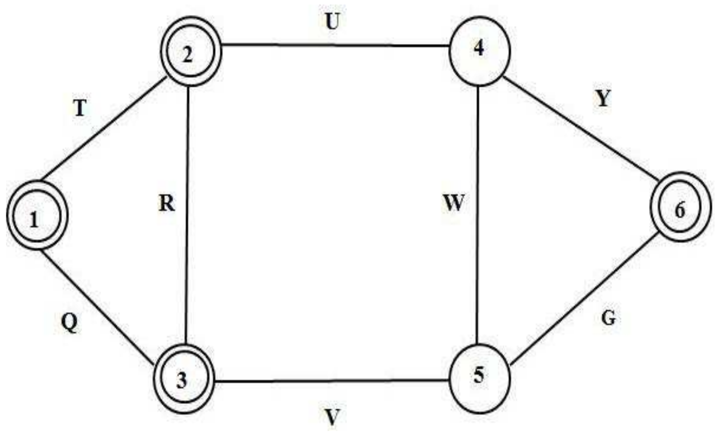


Fig. 2.7: Network for FSTP with Steiner points shown as double circles

Step1: For the given graph G with $|V| = 6$ and $|E| = 8$ and $S = \{1,2,3,6\}$. The value of d for each edge i.e., fuzzy edge weights are as follows:

$T = gfn^*[9,3,2]$	$Q = gfn^*[20,6,4]$
$R = gfn^*[10,2,2]$	$U = gfn^*[15,5,5]$
$V = gfn^*[14,4,5]$	$W = gfn^*[8,1,2]$
$Y = gfn^*[16,3,4]$	$G = gfn^*[7,2,5]$

From the given graph, the fuzzy complete undirected distance graph $G_1 = (V_1, E_1, d_1)$ is constructed following the below stated procedure:

Vertex set $V_1 = S = \{1,2,3,6\}$ and using these vertices, the fuzzy complete graph G_1 is created.

The Edge weights are converted from QGFN to DFN using Definition 1 of section 2.2.1 (II). Let $K = 2$

$$T = \{[0,15]^0, [5.5,11.5]^{0.5}, [9,9]^1\}$$

$$Q = \{[2,32]^0, [13,25]^{0.5}, [20,20]^1\}$$

$$R = \{[4,16]^0, [7.5,12.5]^{0.5}, [10,10]^1\}$$

$$U = \{[0,30]^0, [9,21]^{0.5}, [15,15]^1\}$$

$$V = \{[2,29]^0, [9,20.5]^{0.5}, [14,14]^1\}$$

$$W = \{[4,14]^0, [6.5,10.5]^{0.5}, [8,8]^1\}$$

$$Y = \{[7,28]^0, [12.5,21]^{0.5}, [16,16]^1\}$$

$$G = \{[1,22]^0, [4.5,13]^{0.5}, [7,7]^1\}$$

Step 2: All possible paths between pair of nodes in V_1 are explored and the fuzzy shortest path between them is determined using the given graph G, Definition 2, 3 of section 2.2.1 (II) and Eq. (2.15).

Between Node 1-2 possible paths are:

$(a)\ T = \{[0,15]^0, [5.5,11.5]^{0.5}, [9,9]^1\} \ = gfn^*[9,3,2]$

$(b)\ Q + R = \{[6,48]^0, [20.5,37.5]^{0.5}, [30,30]^1\} = gfn^*[30,8,6]$

$(c)\ Q + V + W + U = \{[8,105]^0, [37.5,77]^{0.5}, [57,57]^1\} = gfn^*[57,16,16]$

$(d)\ Q + V + G + Y + U = \{[12,141]^0, [48,100.5]^{0.5}, [72,72]^1\}$
$= gfn^*[72,20,23]$

Now, $Z_{min} = \{[0,15]^0, [5.5, 11.5]^{0.5}, [9,9]^1\} = gfn^*[9,3,2]$

Hence, the fuzzy shortest path from Node1-2 is T.

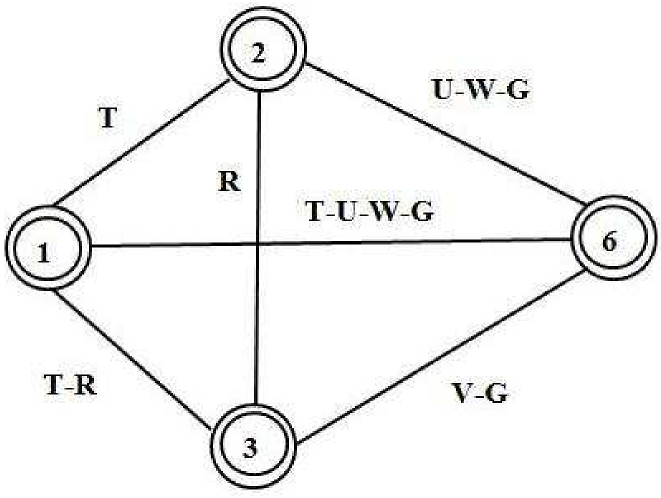


Fig. 2.8: Fuzzy complete undirected distance graph G_1

Similarly, the fuzzy shortest path between all pair of nodes in the vertex set V_1 is determined and the fuzzy complete undirected distance graph G_1 shown in Fig. 2.8 is generated.

Step 3: The minimal spanning tree $T_1\ of\ G_1$ is constructed using extended Prim's algorithm for Fuzzy MST.

$T = \{[0,15]^0, [5.5,11.5]^{0.5}, [9,9]^1\} = gfn^*[9,3,2]$

$T - R = \{[4,31]^0, [13,24]^{0.5}, [19,19]^1\} = gfn^*[19,5,4]$

$T - U - W - G = \{[5,81]^0, [25.5,56]^{0.5}, [39,39]^1\} = gfn^*[39,11,15]$

$R = \{[4,16]^0, [7.5,12.5]^{0.5}, [10,10]^1\} = gfn^*[10,2,2]$

$U - W - G = \{[5,66]^0, [20,44.5]^{0.5}, [30,30]^1\} = gfn^*[30,8,12]$

$V - G = \{[3,51]^0, [13.5,33.5]^{0.5}, [21,21]^1\} = gfn^*[21,6,10]$

Now, $Z_{min} = \{[0,15]^0, [5.5, 11.5]^{0.5}, [9,9]^1\} = gfn^*[9,3,2]$

Following are the status of the two arrays used in the extended Prim's algorithm, after each run of the loop till the priority queue Q becomes EMPTY. Table 2.7 shows the initial status of the two arrays, Table 2.8 shows the status after the root is initialized, Table 2.9 shows the status of the two arrays after the neighbours of the root are explored and Table 2.10 shows the final connections in the fuzzy minimal spanning tree $T_1\ of\ G_1$ and Fig. 2.9 shows the final connections in the Fuzzy Minimal Spanning Tree.

Table 2.7: Initial status of the two arrays

Node	1	2	3	6
$\pi[u]$	NIL	NIL	NIL	NIL
$Key[u]$	$-\infty$	$-\infty$	$-\infty$	$-\infty$

Table 2.8: LPI value for the root is initialized

Node	1	2	3	6
$\pi[u]$	NIL	NIL	NIL	NIL
$Key[u]$	0	$-\infty$	$-\infty$	$-\infty$

Table 2.9: Showing the status of the two arrays after the neighbours of the root are explored

Node	1	2	3	6
$\pi[u]$	NIL	1	1	1
$Key[u]$	0	1	0.111	0.0014

Table 2.10: Showing the final connections in the fuzzy minimal spanning tree $\boldsymbol{T_1 of\ G_1}$

Node	1	2	3	6
$\pi[u]$	NIL	1	2	3
$Key[u]$	0	1	0.9695	0.044

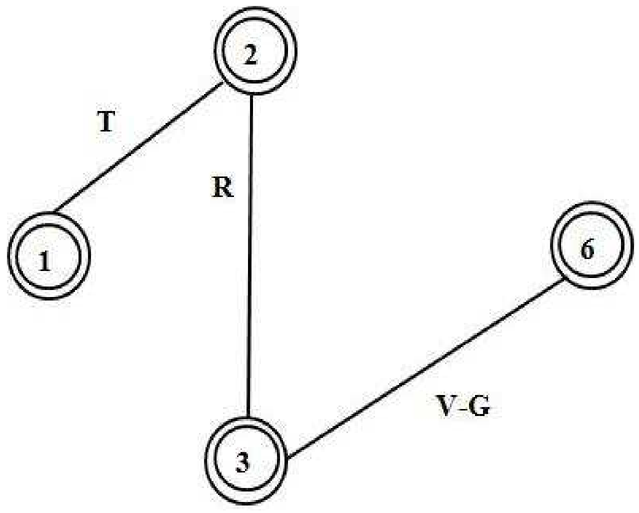


Fig. 2.9: Fuzzy Minimal Spanning Tree $T_1 of\ G_1$

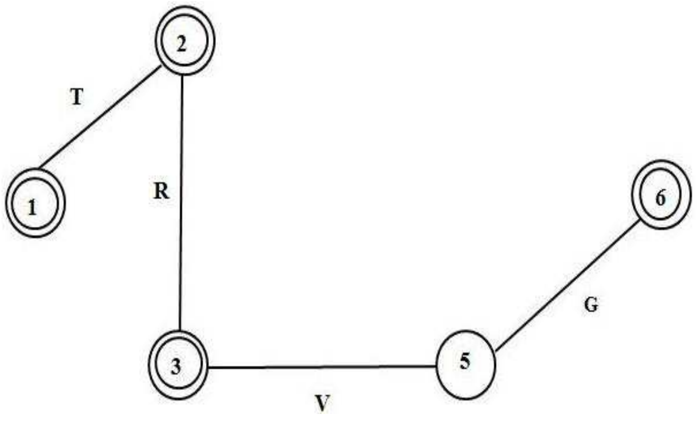


Fig. 2.10: Final Fuzzy Steiner Tree(T_{FS})

Step 4: Now the connections in T_1 are replaced by their corresponding shortest paths in G and we get the G_s and because there are no cycles and no terminal node becomes a leaf node, the fuzzy minimum spanning tree T_s becomes the final fuzzy Steiner tree (T_{FS}) as shown in the Fig. 2.10.

2.2.2 Trapezoidal Fuzzy Number (TFN)

Out of the several types of fuzzy numbers that exist, most of the times trapezoidal fuzzy numbers are used for tackling the imprecise nature of the parameters associated with the real life applications for two important reasons.

Firstly, it is the most generic class of fuzzy numbers and has a linear membership function. Therefore, it is widely used in modelling the uncertainty of the scientific and applied engineering problems. Secondly, its simplicity both in terms of concept and computation increases its use while dealing with problems like CSPP (Bansal, 2011).

(I) Membership Function of TFN

A Trapezoidal Fuzzy Number $A = (a_1, a_2, a_3, a_4)$ can be represented by the following membership function where $a_1 \leq a_2 \leq a_3 \leq a_4$ (Bansal, 2011).

$$\mu_A(x) = \begin{cases} 0, & x < a_1 \\ \frac{x-a_1}{a_2-a_1}, & a_1 < x \leq a_2 \\ 1, & a_2 < x < a_3 \\ \frac{a_4-x}{a_4-a_3}, & a_3 \leq x < a_4 \\ 0, & x > a_4 \end{cases} \qquad (2.16)$$

(II) Fuzzy Arithmetic using TFN

Definition 1: Addition of TFN

In problems where the goal is to determine the shortest interconnect, we need to find out the sum of the weights assigned to the edges and when the weights are fuzzy numbers; the procedure is not the same as that for real numbers. As stated in section 2.2.2 (I), each TFN is represented by a set of four values and if we have two such TFN represented as $= (a_1, a_2, a_3, a_4)$ and $B = (b_1, b_2, b_3, b_4)$, then the formula for their addition is as follows (Bansal, 2011):

$$\begin{aligned} A + B &= (a_1, a_2, a_3, a_4) + (b_1, b_2, b_3, b_4) \\ &= (a_1 + b_1 , a_2 + b_2 , a_3 + b_3 , a_4 + b_4) \end{aligned} \qquad (2.17)$$

Definition 2: Ranking of TFN

As discussed earlier, one important property that exists in case of real numbers is that of natural ordering which is lacking in case of fuzzy numbers. Several methods have been suggested for ranking TFN. In the literature, these methods are usually compared against each other using small handpicked examples. Three recently introduced techniques of ranking TFN are discussed below and their behaviour is compared on a non-trivial constrained fuzzy shortest path problem in subsequent sections.

(a) Circumcenter of Centroids (COC)

Many of the methods proposed use the centroid of the trapezoid for ranking of TFN as it can be considered the balancing point but the technique suggested by Rao and Shanker (2011) considers the COC to be a better reference point as it is determined by joining the centroids of the three sub parts of the trapezoid i.e., two triangle and one rectangle. We get another triangle on joining the three centroids and the circumcenter of this resultant triangle when used as a reference point for ranking of TFN, gives a more accurate result. For details see Fig. 2.12.

According to Rao and Shanker (2011), the following formula can be used to determine the COC of any given TFN. If $A = (a_1, a_2, a_3, a_4)$, then

$$COC(A) = (x, y) = \left(\frac{a_1+2a_2+2a_3+a_4}{6}, \frac{(2a_1+a_2-3a_3)(2a_4+a_3-3a_2)+5}{12}\right) \qquad (2.18)$$

This value of $COC(A)$ is then used to determine the rank of fuzzy number A using the following formula:

$$R(A) = \sqrt{x^2 + y^2} \qquad (2.19)$$

For two given fuzzy numbers $A\ and\ B$, $R(A) > R(B)$ implies $A > B$ and $R(A) < R(B)$ implies $A < B$. In the third case, when $R(A) = R(B)$, we need some additional information to determine the rank of $A\ and\ B$ and for that we use the following procedure:

Index of Optimism: An index was defined by Rao and Shanker (2011) that relates to the decision maker's view point by considering his degree of optimism. If we have a fuzzy number $A = (a_1, a_2, a_3, a_4)$ and $COC(A)$ then the index of optimism can be stated as:

$$I_\delta(A) = \delta y + (1-\delta)x \qquad \text{where } \delta \epsilon [0,1] \tag{2.20}$$

If the value of δ is large, it shows that the decision maker is more optimistic and vice versa.

Index of Modality: Another index called index of modality was proposed by Rao and Shanker (2011) for ranking of fuzzy numbers $A\ and\ B$ which considers the importance of both the central value as well as the extreme values. The index of optimism alone was not sufficient enough for ranking as it considered only the extreme values of COC. The index of modality can be represented as:

$$I_{\delta,\gamma}(A) = \gamma((x+y)/2) + (1-\gamma)I_\delta(A) \qquad \text{where } \gamma \in [0,1] \tag{2.21}$$

Following the index stated above we can tackle the situation where $R(A) = R(B)$ by calculating $I_{\delta,\gamma}(A)$ and $I_{\delta,\gamma}(B)$. If $I_{\delta,\gamma}(A) > I_{\delta,\gamma}(B)$ it implies $A > B$ and vice versa.

(b) Maximizing Set (MAS) and Minimizing Set (MIS)

A concept of total utility (combination of left and right utility values) using MAS and MIS was proposed by Chen (1985) for ranking fuzzy numbers but some shortcomings were discovered in this method like the incapability of ranking those fuzzy numbers which had the same left, right or total utility values. To overcome these limitations, Chou *et al.* (2011) suggested a revised ranking approach in which two left and right utilities were considered instead of just single left and right utility value while calculating the total utility. In addition, the revised approach also considers the degree of optimism of the decision maker and has the capability of differentiating between various types of fuzzy numbers.

In (Chou, Dat, & Yu, 2011), for a TFN $A = (a_1, a_2, a_3, a_4)$, f_A^L and f_A^R are used to represent the left and right membership functions, which according to Eq. 2.16 are as follows:

$$f_A^L(x) = \frac{x-a_1}{a_2-a_1} \quad \text{and} \quad f_A^R(x) = \frac{a_4-x}{a_4-a_3} \tag{2.22}$$

If n trapezoidal fuzzy numbers are considered i.e., $A_i, i = 1,2, \dots \dots n$ with membership function μ_{A_i} then the $MAS(M)\ and\ MIS(G)$ with membership functions $f_M\ and\ f_G$ respectively are represented as follows:

$$f_M(x) = \begin{cases} [(x - x_{min})/(x_{max} - x_{min})]^k, & x_{min} \le x \le x_{max} \\ 0, & otherwise \end{cases} \tag{2.23}$$

$$f_G(x) = \begin{cases} [(x_{max} - x)/(x_{max} - x_{min})]^k, & x_{min} \le x \le x_{max} \\ 0, & otherwise \end{cases} \tag{2.24}$$

Where $x_{min} = \inf S, x_{max} = \sup S, S = \cup_{i=1}^{n} S_i, S_i = \{x/\mu_{A_i}(x) \succ 0\}\ and\ k = 1$

As stated earlier, in this revised approach suggested by Chou *et al.* (2011), pair wise computation is performed and for each A_i, two left utilities and two right utilities are calculated. The formulas for utilities are stated below:

Right Utilities.

$$U_{M_{i1}} = \sup_x \left(f_M(x) \wedge f_{A_i}^R(x)\right), i = 1,2. \tag{2.25}$$

$$U_{G_{i2}} = \sup_x \left(f_G(x) \wedge f_{A_i}^R(x)\right), i = 1,2. \tag{2.26}$$

Left Utilities.

$$U_{G_{i1}} = \sup_x \left(f_G(x) \wedge f_{A_i}^L(x)\right), i = 1,2. \tag{2.27}$$

$$U_{M_{i2}} = \sup_x \left(f_M(x) \wedge f_{A_i}^L(x)\right), i = 1,2. \tag{2.28}$$

The formula for total utility which also considers the decision maker's degree of optimism by using the parameter α is as follows:

$$U_T^\alpha(i) = \left\{ \begin{matrix} \alpha\left[U_{M_{i1}}(i) + 1 - U_{G_{i2}}(i)\right] + \\ (1-\alpha)\left[U_{M_{i2}}(i) + 1 - U_{G_{i1}}(i)\right] \end{matrix} \right\} / 2 \qquad i = 1,2. \tag{2.29}$$

The functions f_M and f_G of $MAS(M)$ and $MIS(G)$ respectively intersects the left membership function f_A^L and the right membership function f_A^R of the fuzzy number A. The points of intersection for a TFN $A_i = (a_{i_1}, a_{i_2}, a_{i_3}, a_{i_4})$ can be denoted as M_{i1}, M_{i2} and G_{i1}, G_{i2} and the following equations can be used to determine their values:

$$x_{M_{i1}} = \frac{a_{i_4}x_{max} - a_{i_3}x_{min}}{(a_{i_4} - a_{i_3}) + (x_{max} - x_{min})} \tag{2.30}$$

$$U_{M_{i1}} = \frac{a_{i_4} - x_{M_i}}{(a_{i_4} - a_{i_3})} = \frac{a_{i_4} - x_{min}}{(a_{i_4} - a_{i_3}) + (x_{max} - x_{min})} \tag{2.31}$$

$$x_{G_{i1}} = \frac{a_{i_2}x_{max} - a_{i_1}x_{min}}{(a_{i_2} - a_{i_1}) + (x_{max} - x_{min})} \tag{2.32}$$

$$U_{G_{i1}} = \frac{x_{G_i} - a_{i_1}}{(a_{i_2} - a_{i_1})} = \frac{x_{max} - a_{i_1}}{(a_{i_2} - a_{i_1}) + (x_{max} - x_{min})} \tag{2.33}$$

Thus, for each fuzzy number A_i, the total utility value is given by:

$$U_T^\alpha(i) = \frac{1}{2}\left(\alpha\left[\frac{a_{i_4} - x_{min}}{(a_{i_4} - a_{i_3}) + (x_{max} - x_{min})} + \frac{a_{i_3} - x_{min}}{(a_{i_3} - a_{i_4}) + (x_{max} - x_{min})}\right] + (1 - \alpha)\left[\frac{a_{i_1} - x_{min}}{(a_{i_1} - a_{i_2}) + (x_{max} - x_{min})} + \frac{a_{i_2} - x_{min}}{(a_{i_2} - a_{i_1}) + (x_{max} - x_{min})}\right]\right), i = 1, 2. \tag{2.34}$$

For two given fuzzy numbers A_1 and A_2, if $U_T^\alpha(A_1) > U_T^\alpha(A_2)$ then $A_1 > A_2$ and vice versa.

(c) Weighting Function (WF)

Another method of ranking fuzzy numbers was proposed by Saeidifar (2011). The interesting fact about this ranking technique is that it uses two things i.e., fuzzy numbers are defuzzified and a weighting function is applied. In this approach a weighted distance measure is defined on fuzzy numbers and then this distance is minimized to obtain the point approximations and the weighted interval of fuzzy numbers.

According to Saeidifar (2011), a few definitions for a fuzzy number A with $[A]_\beta = \left[\underline{a}(\beta), \overline{a}(\beta)\right]$ and a weighting function $f(\beta) = \left(\underline{f}(\beta), \overline{f}(\beta)\right)$ are as follows:

(i) Nearest $\boldsymbol{f} = \left(\underline{\boldsymbol{f}}, \overline{\boldsymbol{f}}\right) -$ weighted interval approximation (NWIA):

$$NWIA_f(A) = \left[\int_0^1 \underline{f}(\beta)\underline{a}(\beta)\, d\beta, \int_0^1 \overline{f}(\beta)\overline{a}(\beta)\, d\beta\right] \tag{2.35}$$

(ii)$\boldsymbol{f} -$ weighted mean:

$$\overline{L_f}(A) = \frac{\int_0^1 \underline{f}(\beta)\underline{a}(\beta) + \overline{f}(\beta)\overline{a}(\beta)}{2} d\beta \tag{2.36}$$

Therefore, for the fuzzy number, the weighting mean $\overline{L_f}(A)$ is denoted as:

$$\overline{L_f}(A) = \frac{1}{2}\int_0^1 \underline{f}(\beta)\underline{a}(\beta)\, d\beta + \overline{f}(\beta)\overline{a}(\beta)\, d\beta$$

$$= \frac{\int_0^1 \underline{f}(\beta)\underline{a}(\beta)\, d\beta + \int_0^1 \overline{f}(\beta)\overline{a}(\beta)\, d\beta}{\int_0^1 \underline{f}(\beta)d(\beta) + \int_0^1 \overline{f}(\beta)d(\beta)} \tag{2.37}$$

If we have two fuzzy numbers $A\ and\ B$ and a WF (f) then the ranking can be determined in the following way:

$A \prec B\ if\ and\ only\ if\ \overline{L_f}(A) < \overline{L_f}(B)$,

$A \sim B\ if\ and\ only\ if\ \overline{L_f}(A) = \overline{L_f}(B)$,

$A \succ B\ if\ and\ only\ if\ \overline{L_f}(A) > \overline{L_f}(B)$

(III) Application of TFN

(a) Constrained Fuzzy Shortest Path Problem (CFSPP)

As stated earlier CSPP is an NP-Complete problem. Techniques suggested to solve this problem reduce the number of different values (cost or delay) by using the process of discretization so that the problem can be reduced to a polynomial time solvable problem (Chen, Song, & Sahni, 2008). The amount of error introduced during the process of discretization determines the effectiveness of these techniques. Taking into consideration two factors, (1) the utilization of limited resources and (2) improving the efficiency of network functions, Chen *et al.* (2008) suggested two methods of discretization, namely randomized discretization and path-delay discretization.

(i) Path Delay Discretization

Here we use the path-delay discretization technique to solve CFSPP. Instead of working with individual link delays as in case of randomized discretization which leads to the problem of error accumulation. Path-delay discretization deals with path delays and discretization is performed using interval partitioning method so the problem of error accumulation is eliminated (Chen, Song, & Sahni, 2008). For a given path P,

$$d'(P) = \left\lfloor \frac{d(P)}{r} \lambda \right\rfloor \tag{2.38}$$

Where $\lfloor X \rfloor = floor(X)$ is the largest integer not greater than X. $d(P)$ denotes the delay of the path P, r is the given delay requirement and the delay requirement is bounded by an integer λ.

(ii) Problem Formulation

Given network is denoted as $G(V, E)$ where V is a set of n nodes and E is a set of m links. Each edge $(u, v) \in E$ has a delay and a cost value associated with it denoted as $d(u, v)$ and $c(u, v)$ respectively. For a path P, the delay and cost of the path are denoted as $d(P)$ and $c(P)$ respectively where

$$d(P) = \sum_{(u,v) \in P} d(u, v) \tag{2.39}$$

$$c(P) = \sum_{(u,v) \in P} c(u, v) \tag{2.40}$$

The length of the longest path in the network is denoted by L. A path P is called a feasible path if $d(P) \leq r$ where r is the given delay requirement and the cheapest feasible path is one that not only satisfies the delay requirement but for which the cost $c(P_{s,t})$ is the minimum among all possible paths connecting the source node s and the destination node t (Chen, Song, & Sahni, 2008).

In case of CFSPP, the cost values $c(u,v)$ are trapezoidal fuzzy numbers. Therefore, the cost of the path $c(P)$ can be determined using Eq. 2.17 and the cheapest possible path can be determined using the ranking methods discussed in Definition 2 of section 2.2.2 (II).

(iii) CFSPP Problem

Given a graph $\boldsymbol{G}(\boldsymbol{V},\boldsymbol{E})$ with a fuzzy cost and a crisp delay associated with each edge, find the cheapest path that satisfies the given delay constraint.

(iv) Proposed Algorithm for CFSPP

```
$\boldsymbol{Get\_abcd}(\ \boldsymbol{alpha}, \boldsymbol{stretch}, \boldsymbol{cost})$
1. $a_1 = cost - stretch$
2. $a_2 = a_1 + alpha$
3. $a_4 = cost + stretch$
4. $a_3 = a_4 - alpha$

$\boldsymbol{Initialize}\ (\boldsymbol{V}, \boldsymbol{s}, \boldsymbol{\lambda})$

1.$\boldsymbol{for}$ $each\ vertex\ \boldsymbol{v} \in \boldsymbol{V}, each\ \boldsymbol{i} \in [\boldsymbol{0}, \dots \dots, \boldsymbol{\lambda}]$
2.  $\boldsymbol{Get\_abcd}(\ \boldsymbol{alpha}, \boldsymbol{stretch}, \boldsymbol{cost})$
3.  $w[v,i] := \infty, \pi[v,i] := NIL, z[v,i] := \infty$
4.  $w[s,0] := 0, z[s,i] := 0$
5. $\boldsymbol{end\ for}$

$\boldsymbol{Relax\_FPDA}(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{i}, \boldsymbol{\lambda})$

1. $i' := floor\left(\frac{z[u,i] + d(u,v)}{r}\lambda\right)$
```

```
2. $\boldsymbol{Get\_abcd}(\ \boldsymbol{alpha}, \boldsymbol{stretch}, \boldsymbol{cost})$
3. $\boldsymbol{if}\ i' \leq\ \lambda\ and\ w[v,i'] > w[u,i] + c(u,v)$

// Compare 1 using Definition 2(a) of Section 2.2.2 (II) and Eq. 2.18, 2.19, 2.20 and 2.21.
// Compare 2 using Definition 2(b) of Section 2.2.2 (II) and Eq. 2.34.
// Compare 3 using Definition 2(c) of Section 2.2.2 (II) and Eq. 2.37.

4.   $w[v,i'] \coloneqq w[u,i] + c(u,v)$
5.   $\pi[v,i'] \coloneqq u$
6.   $z[v,i'] \coloneqq min\{z[v,i'],\ z[u,i] + d(u,v)\}$
7. $\boldsymbol{end\ if}$
```

$\boldsymbol{FPDA\_Dijkstra}(\boldsymbol{G}, \boldsymbol{s}, \boldsymbol{\lambda})$

```
1. $\boldsymbol{Initialize}\ (\boldsymbol{V}, \boldsymbol{s}, \boldsymbol{\lambda})$
2. $\boldsymbol{for}$ i = 0 to $\lambda$
3.    $Q \coloneqq V$
4.    $\boldsymbol{while}\ Q \neq \varphi$
5.       $u \coloneqq Extract\_Min(Q)$
6.       $\boldsymbol{if}\ w[u,i] = \infty$
7.        $\boldsymbol{break}$ $out\ of\ the$ $\boldsymbol{while}$ $loop$
8.       $\boldsymbol{end\ if}$
9.   $Q \coloneqq Q - \{\,u\,\}$
10.  $\boldsymbol{for}\ every\ adjacent\ node\ \boldsymbol{v}\ of\ \boldsymbol{u}$
11.   $\boldsymbol{Relax\_FPDA}(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{i}, \boldsymbol{\lambda})$
12.  $\boldsymbol{end\ for}$
13.     $\boldsymbol{end\ while}$
14. $\boldsymbol{end\ for}$
```

$\boldsymbol{FPDA}(\boldsymbol{G}, \boldsymbol{s})$

```
1. $\lambda \coloneqq \lambda_0$
2. $\boldsymbol{do}$
3. $\lambda \coloneqq 2\lambda$
```

```
4.    $\boldsymbol{FPDA\_Dijkstra(G, s, \lambda)}$
5. $\boldsymbol{while}\ \exists v \in V,\ d(P^v) > (1 + \epsilon)r$
// where $P^v$ is the path with $min\{w[v, i] | \ i \in [0 \ldots\ldots \lambda]\}$
```

The above stated algorithm is an extension of the Path Delay Discretization Algorithm (PDA), presented by Chen *et al.* (2008). The algorithm is capable of handling the fuzzy environment. Here we are dealing with two constraints, namely cost and delay. Each edge of the graph has two values associated with it (cost and delay) and in the fuzzy version of the algorithm we consider one of the parameters i.e., cost as a fuzzy number and the other parameter which is delay is represented as a real number. Both the values are initially generated randomly using $\boldsymbol{rand()}$**.** From these crisp values, the trapezoidal fuzzy number for each cost value is generated using $\boldsymbol{Get\_abcd()}$. $\boldsymbol{alpha}$ and $\boldsymbol{stretch}$ are the two constant values supplied by the user for the formulation of TFN. $w[v, i], \pi[v, i]\ and\ z[v, i]$ are the three two dimensional arrays used which are all initialized by the function $\boldsymbol{Initialize\ (V, s, \lambda)}$. The cost of the cheapest path P connecting $s\ and\ v$ with $d(P^v) = i$ is stored in the array $w[v, i]$, $, \forall\ v \in V\ and\ i \in [0, \ldots\ldots, \lambda]$. $z[v, i]$ is an array that keeps a track of the minimum delay of paths connecting $s\ and\ v$ for which discretized delays are i. $\pi[v, i]$ is another array storing the last link of the path. $\boldsymbol{FPDA\_Dijkstra(G, s, \lambda)}$ evaluates $w[v, i]$ and $\pi[v, i]$ for any target v and any given λ. This function determines the cheapest path among different delay paths, $d(P^v) \epsilon\ [0, \ldots\ldots, \lambda]$. Let the cheapest path be denoted as P^v. The function $\boldsymbol{FPDA(G, s)}$ calls $\boldsymbol{FPDA\_Dijkstra(G, s, \lambda)}$ iteratively with increasing value of λ (as can be observed in the lines 1-4 of the function $\boldsymbol{FPDA(G, s)}$ of CFSPP algorithm) till $d(P^v)$ is less than $(1 + \epsilon)r$ for all $v \in V$.

We use three different methods for ranking of fuzzy numbers with the aim to compare their performances. The experimental analysis along with the result is presented in the next section. The time complexity of the algorithm remains the same as specified in (Chen, Song, & Sahni, 2008), i.e., $O((m + n \log n)L/\varepsilon)$ since the number of arithmetic operations in the fuzzy version increases only by a constant factor.

(v) Experimental Analysis

We implemented CFSPP algorithm in C language and compiled using CodeBlocks for running on an i5 based system running at 3.20 GHz with 3GB RAM. Three versions of the program were written with the three different ranking methods as per Eqs. 2.18, 2.19, 2.20, 2.21, 2.34 and 2.37. These programs were verified by comparison with hand calculations on small graphs. Also, it was verified that for large graphs when the width of fuzzy numbers is set to zero the path obtained agrees with algorithm for the crisp case as given in (Chen, Song, & Sahni, 2008).

Many real networks like the World Wide Web links, biological networks, and social networks are known to be *scale-free*, i.e., the fraction of nodes in the network having a given degree follows a power law. Therefore, we created test cases using power law random graph model. The test graphs were generated using *gengraph-win*. The parameters used for the random graph generation were (*n, alpha, min, max, z*), where *n* is an integer denoting the count of degree number or the number of nodes, *min* is the minimum degree, *max* is the maximum degree and *alpha* is the exponent of the power law distribution which is a random number between 1-2.5. The command "*distrib n alpha min max z*" was used to generate a sample of *n* integers in the range specified by *min-max* from a heavy-tailed distribution of exponent *alpha* and average *z.* The experiments were repeated sufficient number of times with $n = 250$, $alpha = 2.5$, $min = 25$, $max = 75$ *and* $z = 45$ and conclusions were drawn based on the average behaviour. The graph generator *gengraph-win* gave un-weighted power law graphs with random structure. Random weights were assigned to the edges using uniformly distributed random numbers in the range 10 to 100 obtained using the C *rand()* function.

The results are shown in the following figures. In Fig. 2.11(a), the defuzzified cost of the shortest path found is plotted against the delay requirement. It was observed that as we relax the delay requirement, the cost of the shortest path found by the algorithm decreases and finally at very large values of delay requirement, the delay constraint becomes insignificant and the algorithm runs like the classical shortest path algorithm viz. Dijkstra's algorithm giving the same result for both the algorithms. In terms of cost, MAS/MIS and

WF methods provide better results than COC. In Fig. 2.11(b), the path discretization error that comes into existence due to discretization is shown for all the three methods and it was observed that in terms of path discretization error, COC shows better performance than the other two methods. This may be due to the fact that COC is equidistant from each vertex (i.e., centroids of the two triangles and central rectangle) as shown in Fig. 2.12. Fig. 2.11(c) shows how the change in delay requirement, affects the CPU execution time of the three methods and from the graph it is visible that WF is better than the other two techniques in this aspect. We also notice that the CPU execution time generally increases when the delay requirement is relaxed (increased). This is because with relaxed delay constraint, there is an increase in number of feasible paths thereby increasing the size of search space that has to be explored by the algorithm.

As the delay requirement is relaxed, the cost of cheapest path found decreases and for very high delay requirement value, the cost of shortest path reduces and tends to Dijkstra's values as shown by the blue part of the plot in Fig. 2.13. In the same figure, the green and red regions of the surface demonstrate the effect of the proposed algorithm and shows how the cost of shortest path found by the algorithm increases with stricter delay constraint. The behaviour of the proposed algorithm shows a marked deviation from Dijkstra's algorithm when delay constraint is strict (i.e., small), as is clearly seen in Fig. 2.13. The cost obtained by all the three methods in the form of trapezoidal fuzzy numbers with respect to a constant delay constraint of 30 is shown using a box and whisker plot in Fig. 2.14. As visible in Fig. 2.14, the cost obtained by COC method of ranking is much higher than the cost obtained by the other two methods, namely MAS/MIS and WF.

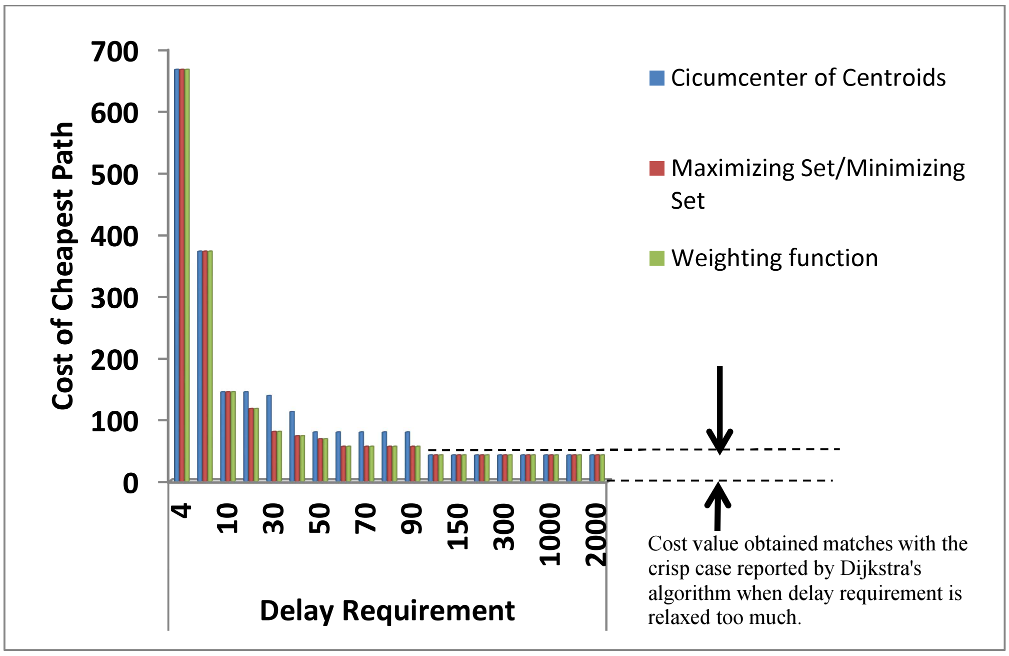


Fig. 2.11(a): Behavior in terms of cost of shortest path shown by different ranking methods by varying the delay requirement on a random graph with 250 nodes generated by gengraph-win

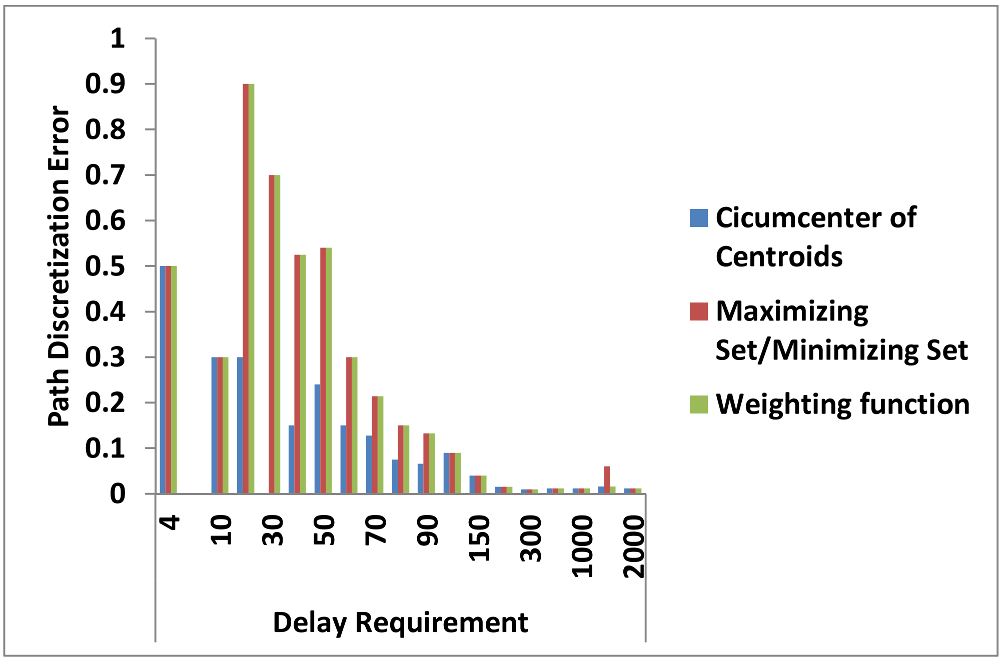


Fig. 2.11(b): Behavior in terms of path discretization error shown by different ranking methods by varying the delay requirement on a random graph with 250 nodes generated by gengraph-win

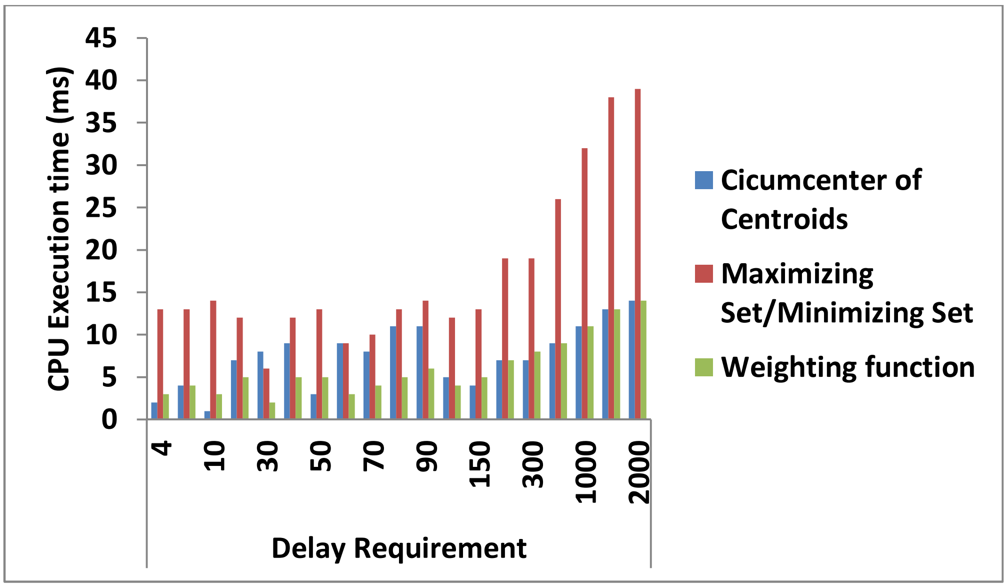


Fig. 2.11(c): Behavior in terms of CPU Execution time shown by different ranking methods by varying the delay requirement on a random graph with 250 nodes generated by gengraph-win

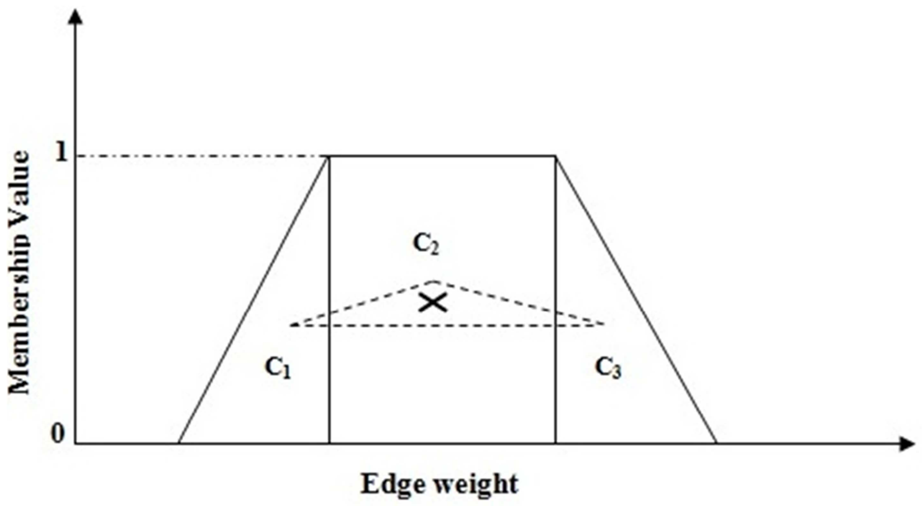


Fig. 2.12: The point considered as Circumcenter of Centroid (COC) is shown by X

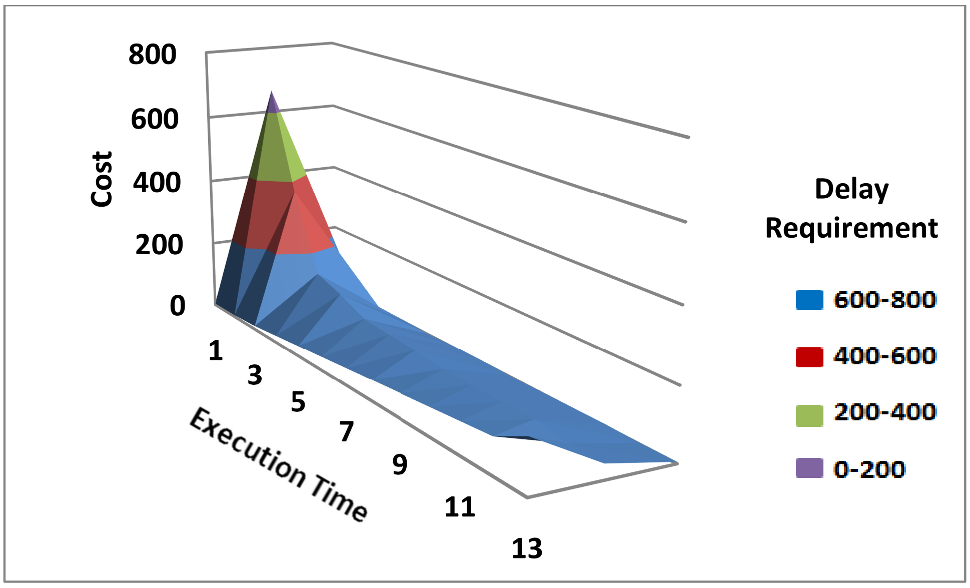


Fig. 2.13: A surface plot with the delay requirement, cost and CPU execution time of WF

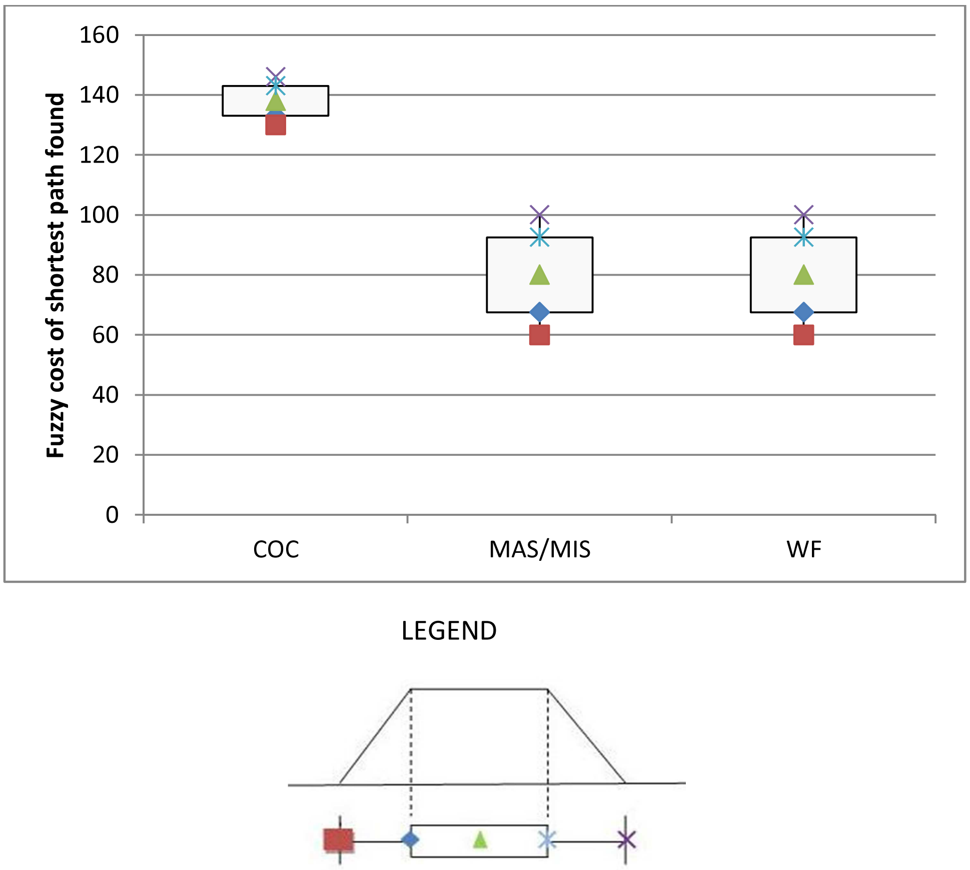


Fig. 2.14: Box and Whisker plot showing the fuzzy cost as a TFN with four parameters at the delay requirement = 30 units

(b) Wireless Sensor Networks

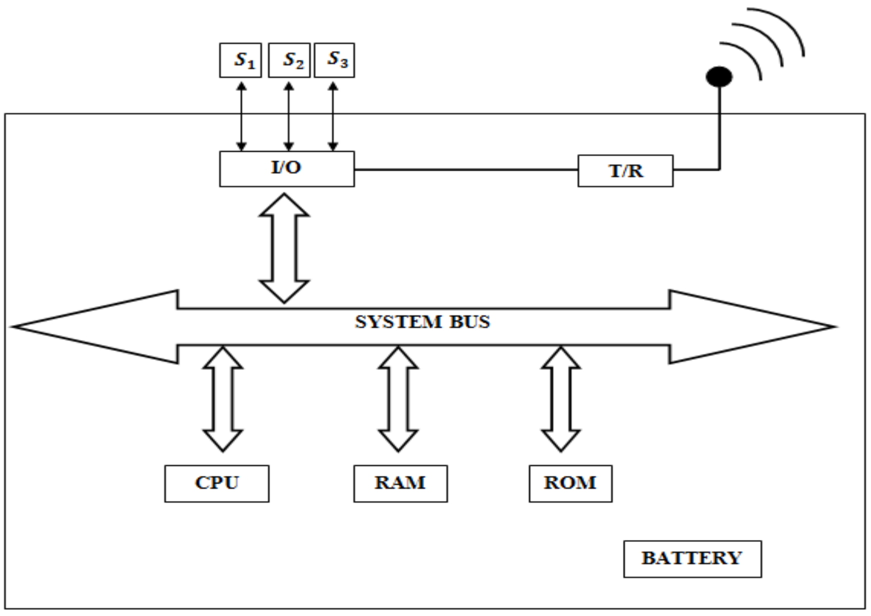
Recent technological advancement has given rise to the necessity of establishing a connection or a relationship between the physical world and the digital world and a wireless sensor network (WSN) plays an important role in creating such a connection. A WSN is a collection of nodes (called motes) interfaced with an array of diverse sensors and possessing the ability to perform limited computations. The sensor nodes are connected through wireless links to communicate the sensor outputs to a designated destination. These sensor networks are capable of fulfilling the demands of the end user by providing precise and reliable information in terms of time and space of the process/system under consideration obtained through sensor measurements. These WSN are ad hoc in nature as they do not have a predefined architecture like the one that exists in case of wired networks (Meguerdichian, Koushanfar, Qu, & Potkonjak, 2001).

Each mote in a WSN has (1) a microprocessor and some memory for processing of signals and scheduling of tasks, (2) an array of sensors which is capable of measuring the changes in the physical environment like the presence of toxic gases, seismic activity, temperature, humidity, pressure etc., (3) a transceiver for transmitting and receiving data and is responsible for forwarding the data to its neighbouring nodes (lying within its radio range). The advantage of WSN is that it is decentralized in nature which makes the network scalable and more robust but because these networks are battery operated, energy is to be conserved to save the motes from dying out. Another limitation of WSN is the communication bandwidth which is restricted to its wireless frequency range (Zhao, & Guibas, 2004). A simplified view of a mote is shown in Fig. 2.15.

A WSN can be represented as a graph where motes and wireless links of WSN correspond to the nodes and edges of the graph respectively. WSNs can be modelled using both a directed as well as an undirected graph. If the radio range of all the nodes is same / different then they are modelled using an undirected / directed graph. A WSN with motes lying in a plane can be signified by a unit disc graph (UDG) in which a circle is used to symbolize a mote (centre of the circle) and its radio range (radius of the circle) as shown in Fig. 2.16. All the nodes within a circle are adjacent to the node at the centre (Shukla, & Sah, 2013).

WSN has a wide range of applications which includes environmental monitoring, battlefield awareness, infrastructure protection, industrial sensing and diagnostics, context-aware computing etc. (Capella, Bonastre, Ors, & Peris, 2013; Lee, & Chung, 2009; Shepherd, Beirne, Lau, Corcoran, & Diamond, 2007; Zhao, & Guibas, 2004; Zhuiykov, 2012) and as stated earlier, WSNs are battery operated, so these networks have a life till the battery can supply power. This leads to the need of determining shortest paths (SP) for the routing of data such that energy can be utilized optimally and the network lifetime can be increased. Another method of preserving the network energy and increasing the throughput is to reduce the energy consumption and at the same time maintain appropriate end to end delay. This additional constraint of delay along with that of energy consumption makes the determination of SP an intractable problem which can be stated as the constrained shortest path problem (CSPP) (Chen, Song, & Sahni, 2008). Considering the fields where WSN finds its applications, we observe that the network parameters involved like energy and delay cannot be assessed precisely since the characteristics of radio links depends upon a number of conditions including weather conditions like temperature and humidity. This gap can be filled by using fuzzy numbers to represent the network parameters and tackle the uncertainty involved.

We use TFN to represent the parameters energy and delay and determine a constrained shortest path in a WSN by using the path delay discretization technique suggested by Chen *et al*. (2008). Though the algorithm presented here is most suitable for the network-flow and QoS aware routing protocols of WSNs, it can still be adapted for other routing protocols available for WSNs like location-based protocols, hierarchical protocols, data-centric protocols etc. with small and appropriate modifications (Akkaya, & Younis, 2005; Xiangning, & Yulin, 2007).



T/R: Radio Transceiver	I/O: Input / Output Interface
S_1, S_2, S_3 : Sensors	CPU: Central Processing Unit
RAM: Random Access Memory	ROM: Read Only Memory

Fig. 2.15: Block diagram of a typical Wireless Sensor node (mote)

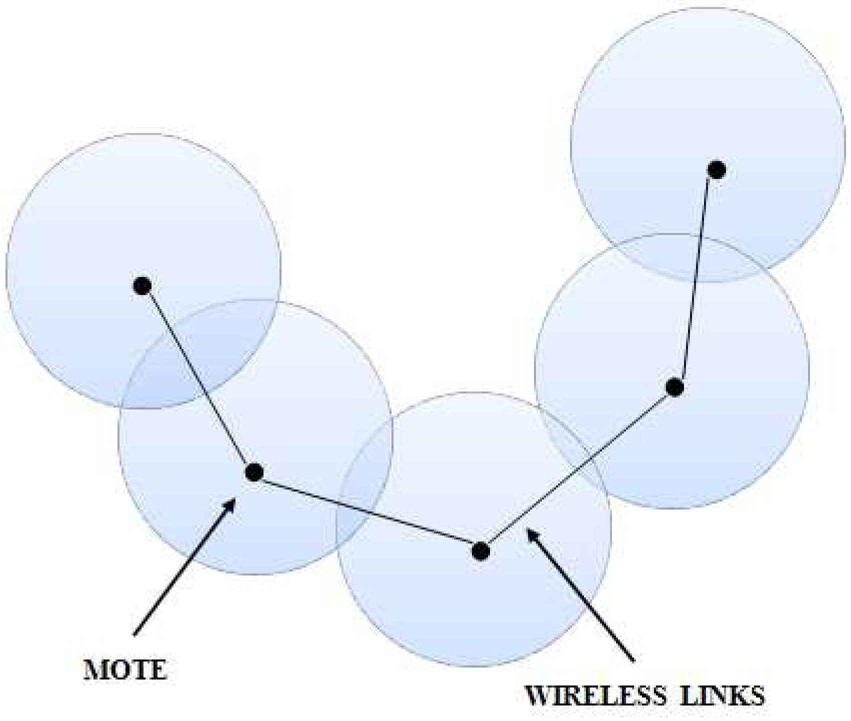


Fig. 2.16: A Wireless Sensor Network (WSN) represented as a Unit Disc Graph (UDG)

(i) Proposed Algorithm

The CFSPP algorithm stated in section 2.2.2 (III) (a) (iv) can be used in WSNs with the difference that the function for creating the TFN accepts "energy" as its input value instead of cost i.e., $(\boldsymbol{alpha}, \boldsymbol{stretch}, \boldsymbol{energy})$ function is used and the structure of TFN is shown in Fig. 2.17.

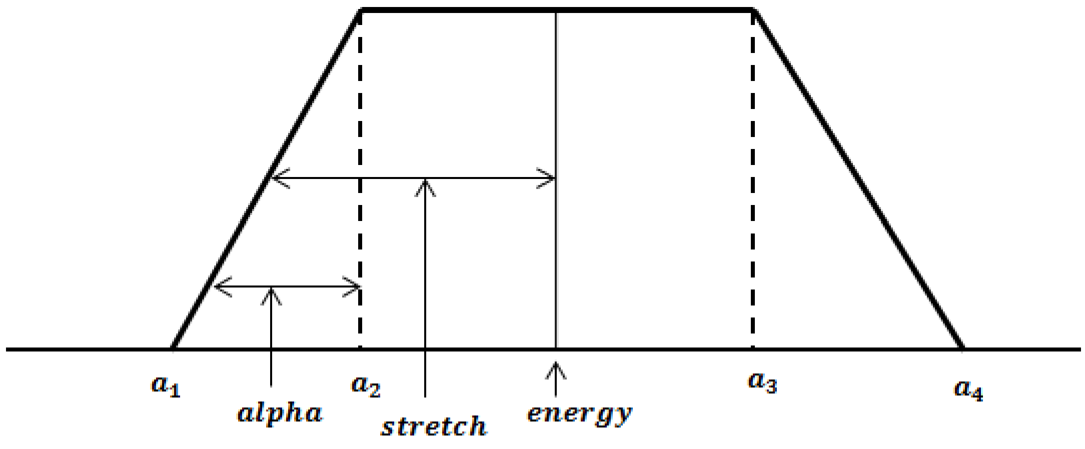
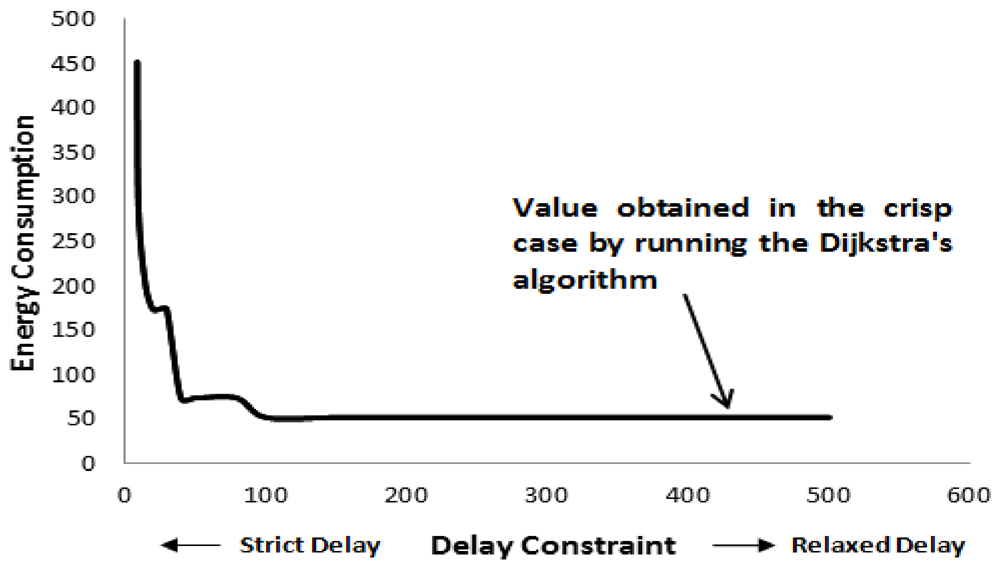


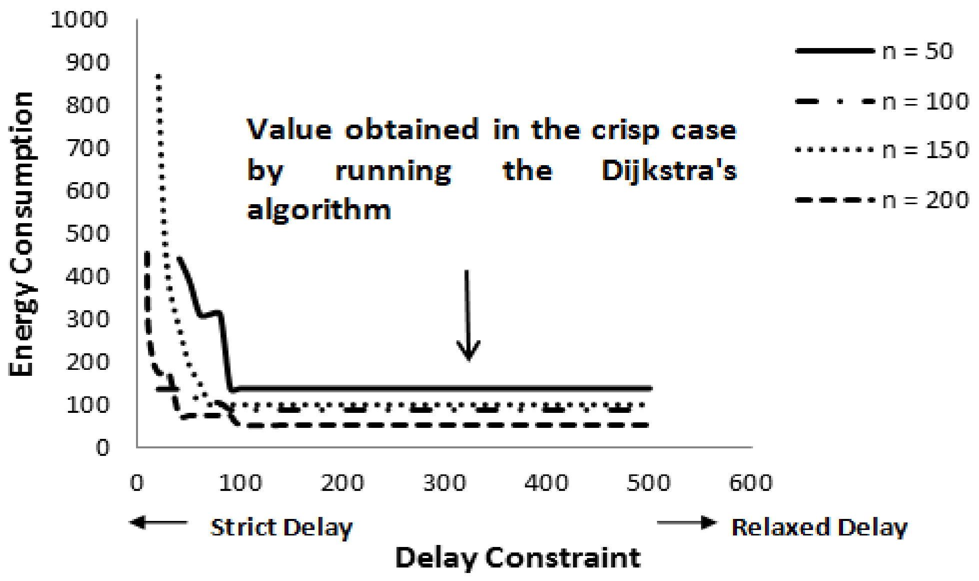
Fig. 2.17: Pictorial representation of trapezoidal fuzzy number

(ii) Experimental Analysis

CFSPP algorithm was implemented in C language and compiled using CodeBlocks. All experiments were performed on an Intel® i7 based system running at 3.40 GHz with 2GB RAM. The test cases were generated using the power law random graph generator *gengraph-win* (http://www-rp.lip6.fr/~latapy/FV/generation.html). This network model is scale free, i.e., the fraction of nodes in the network having a given degree follows a power law (M. Faloutsos, P. Faloutsos, & C. Faloutsos, 1999). Since radio range of motes is small, the probability of nodes with high degree is low. Therefore the power law graph is an appropriate model for WSN. We generated our test cases using *gengraph-win* which gave an un-weighted power law graph having a random structure and then uniformly distributed random numbers ranging from 10 to 100 were assigned to the edges of these graphs. The random edge weights were generated using the C *rand()* function. We applied the CFSPP algorithm on graphs with 50, 100, 150 and 200 nodes and observed the effect of fuzzifying both the parameters viz. energy and delay. It was seen that when the delay constraint value (input) was small, the energy consumption of the path obtained was very high. As the value of delay constraint was increased, the energy consumption of the path obtained kept on decreasing up to a certain point after which the delay constraint was almost insignificant and the energy consumption of the path became constant. At this point the behaviour of CFSPP algorithm was similar to the classical shortest path Dijkstra's algorithm. We verified our results with the one obtained by applying the Dijkstra's algorithm on the same test cases. Fig. 2.18(a) shows the above stated observation for a graph with $n = 200, alpha = 2.5, min = 20, max = 30\ and\ z = 23$. The comparison for all the four networks with number of nodes (n) as 50, 100, 150 and 200 is shown in Fig. 2.18(b). The deviation of energy consumption from the optimum value (i.e., the one obtained using Dijkstra's algorithm) is presented in Table 2.11. As can be observed, for n = 50, no feasible path could be found for strict values of delay constraint i.e., between 9 to 30 as denoted by * in Table 2.11. As the delay constraint value was relaxed i.e., between 40 to 80, feasible paths connecting the source and target were found but with high energy consumption values. This energy consumption value decreased with an increase in the delay constraint value and for higher values of delay constraint i.e., between 90 to 500, the energy consumption value for the obtained feasible path was the optimum and agreed with the value obtained by applying the Dijkstra's algorithm (as shown by value 0 in Table 2.11). A similar behaviour was seen for other network sizes with n = 100, n = 150 and n = 200. Fig. 2.19 shows how the energy consumption value increases with each iteration (as a new edge is added to the path) for two cases i.e., one satisfying the strict and the other satisfying the relaxed delay constraint. In Fig. 2.20, it was observed that as the path is formed, with each iteration an edge is added to form the final path so the path delay value keeps on increasing with each iteration but the final path satisfies the input delay constraint which in our case is taken as 350 and a comparison for four different network sizes (n = 50, n = 100, n = 150, n = 200) is presented.



(a)



(b)

Fig. 2.18: (a) Behaviour of the CFSPP algorithm when applied on a graph with 200 nodes generated using gengraph-win with source (s) = 45 and target (t) = 68, (b) Comparison of the same behaviour for four different network sizes with source (s) = 4 and target (t) = 45

Table 2.11: The energy deviation from unconstrained optimum for increasing delay constraint

Delay Constraint	Energy Deviation From Unconstrained Optimum			
	n = 50	n = 100	n = 150	n = 200
9	*	*	*	400
10	*	*	*	234
15	*	*	*	148
20	*	48	767	122
25	*	48	441	122
30	*	48	283	122
40	304	48	178	22
50	251	48	91	22
60	173	16	51	22
70	173	16	0	22
80	173	16	0	22
90	0	0	0	22
100	0	0	0	0
150	0	0	0	0
200	0	0	0	0
300	0	0	0	0
500	0	0	0	0

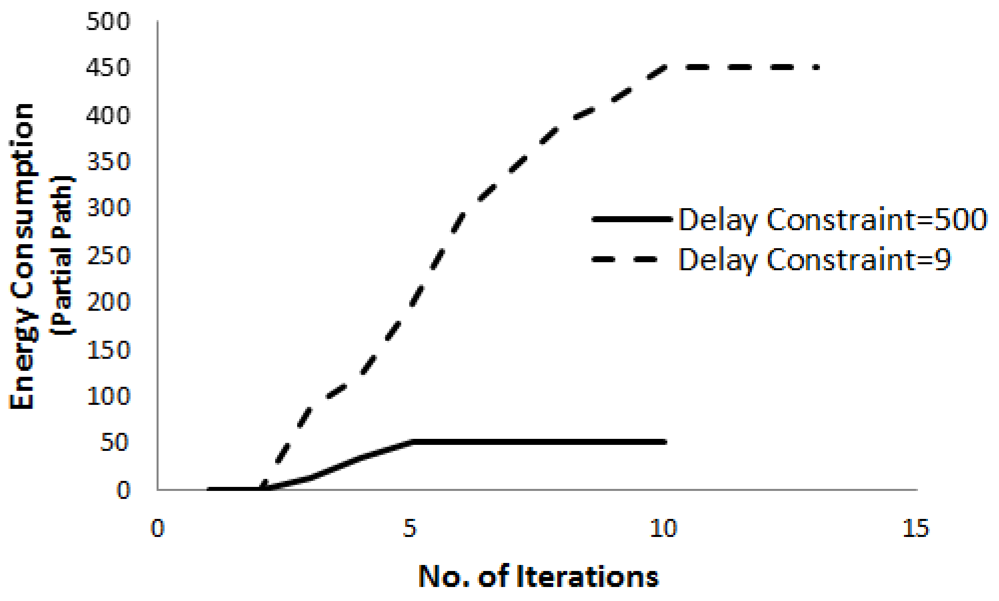


Fig. 2.19: Progress of the CFSPP algorithm in terms of energy consumption with a strict and a relaxed delay constraint when applied on a graph with 200 nodes generated using gengraph-win

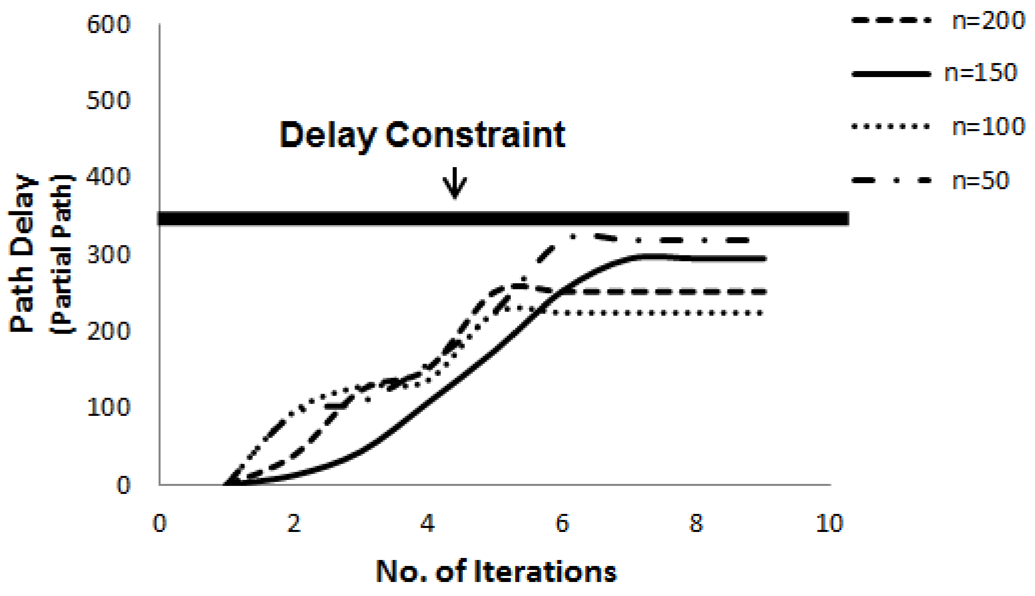


Fig. 2.20: Progress of the CFSPP algorithm in terms of path delay with delay constraint= 350 when applied on a graph with 50, 100, 150, 200 nodes generated using gengraph-win

2.3 Max – Min Formulation for Orienteering Problem

In engineering design or decision making problems, a large number of feasible solutions are available and to choose the solution which is the best one from this set, we need to concentrate on the uncertainty associated with the variables that lead to the optimal solution. Probabilistic concepts can take care of the randomness that arises due to natural fluctuation or natural variations but the uncertainty that comes into existence due to qualitative statements, vague statements, vague nature of the objective, linguistic statements showing the willingness of the decision maker (like the solution is acceptable, low, satisfactory etc.) cannot be addressed through probabilistic concepts. Consequently, we introduce the concept of fuzzy logic in solving the optimization problems. In the crisp definition of optimization problems, we have crisp conditions where solutions violating the constraints or not satisfying the objective function are completely unacceptable but in fuzzy optimization, the concept of degree is introduced. The solution becomes a matter of degree i.e., degree of acceptability or degree of satisfaction is associated with the constraints and the objective functions, and this way we provide a latitude to the acceptability of a solution. This degree of acceptability linked with the objective functions and constraints can be reflected through fuzzy membership functions.

To deal with the situations where several stakeholders vaguely state their preferences as constraints or objective functions using linguistic statements, we convert these statements into fuzzy sets or fuzzy membership functions and then using some technique find out the best "compromise solution". In fuzzy optimization, we do not distinguish between the objective functions and the constraints instead refer to them as fuzzy goals, represented in the form of fuzzy sets defined by their respective membership functions. Therefore, the latitude or uncertainty present in the decision making is tackled through these membership functions. In addition to the fuzzy goals, we can also have crisp constraints modelling the physical conditions or technological feasibility that have to be met in a particular solution.

The whole idea of fuzzy optimization is to allow for latitude in the constraints and flexibility in the objective function. Instead of a 0-1 type solution, we allow for some violation of the original constraints to some degree, set certain limit for the objective function and accept solutions on both sides of the limit to different degrees. The objective function and the set of constraints are converted into fuzzy sets, their associated membership functions are defined and then all the membership functions are combined to determine the fuzzy decision. Through fuzzy optimization, the preferences of the decision maker are quantified and the uncertainty due to vagueness, imprecision etc. which is common in decision making problems are tackled using membership functions (Kaymak, & Sousa, 2001; Loucks, & Beek, 2005; Ramik, 2001).

2.3.1 Basic Definitions

(a) Expected Value of Trapezoidal Fuzzy Number (EV)

To determine the grade of membership of a TFN i.e., to find out the degree to which the TFN satisfies the specified requirement we need to calculate its expected value and then use definition in section 2.2.2 (I). The formula for expected value is as follows (Jimenez, Arenas, Bilbao, & Rodriguez, 2007):

For a given TFN $A = (a_1, a_2, a_3, a_4)$,

$$EV(A) = \frac{1}{4}(a_1 + a_2 + a_3 + a_4) \qquad (2.41)$$

(b) Fuzzy Decision Set (Z)

Fuzzy decision set is a set of elements providing a feasible solution to the stated problem. The fuzzy linear programming problem can have several goals each represented by a membership function and a fuzzy set (F_l) containing the elements along with their grades of membership obtained using the membership function (Jimenez, Arenas, Bilbao, & Rodriguez, 2007).

We define the fuzzy decision as:

$$Z = F_1 \cap F_2 \cap \dots\dots\dots \cap F_l$$

i.e., $$\mu_Z(x) = \mu_{F_1}(x) * \mu_{F_2}(x) * \dots\dots\dots\dots * \mu_{F_l}(x) \qquad (2.42)$$

where $*$ represents a $t-norm$ which can be any operation like $minimum, algebraic\ product, etc.$ In case of OP, $*$ represents $minimum$ operation. To obtain the most desirable solution, we determine the element with the highest membership degree in the fuzzy decision set Z i.e., the value of x that maximizes the membership function of fuzzy decision Z denoted by x^* using the equation stated below:

$$\mu_{Z^*}(x^*) = max\{\mu_Z(x)\} \tag{2.43}$$

Where Z^* denotes the fuzzy set of the most desirable solution.

2.3.2 Problem Definition

The orienteering problem can be represented by a completely connected undirected graph $G(V, E)$ where $V = \{v_1, \dots\dots, v_N\}$ is the set of vertices and E is the set of edges. A score S_i is associated with every vertex $v_i \in V$ and the time taken to traverse each edge $e_{ij} \in E$ is denoted by t_{ij}. The goal here is to determine a path P connecting any subset of V that necessarily includes the start vertex (v_1) and the end vertex (v_N), satisfies the time bound T_{max} and also maximizes the total collected score (Vansteenwegen, Souffriau, & Oudheusden, 2011).

In this chapter, we present the fuzzy orienteering problem (FOP) where the two quantities involved, namely time and score are considered to be fuzzy numbers. The reason to introduce fuzziness into the formulation of OP is that the crisp mathematical formulation is very strict in three ways: (1) the objective function should be either maximized or minimized; (2) none of the constraints should be violated as it leads to an infeasible solution and (3) all constraints are given equal importance. However, these three necessary requirements lead to an unrealistic representation of the real world. By partly relaxing these using fuzzy logic, we can model the physical world in a more realistic manner. Several situations might exist in real life applications that can be easily represented in the fuzzy environment which may include the following (Zimmermann, 2010): The decision maker is not willing to maximize or minimize the objective function; instead he wants to reach some aspiration level like ‘improve the present fuel consumption situation to some extent’ in transportation problems which cannot even be defined or stated in the crisp case. May be the decision maker is willing to accept small violations in the constraints especially when the objective function deals with the aspiration levels and the less than or greater than relation is not required to be followed in the strict mathematical sense. Moreover, the parameters or variables involved may have vagueness which cannot be expressed in the crisp formulation. In the crisp representation, all constraints are of equal importance but may be for the decision maker, different constraints have different importance and small violations of different constraints may be acceptable to different degrees.

In this fuzzy formulation of OP, we have recognized that the parameters time and score are fuzzy in nature and in the fuzzy version we provide latitude to the desired solution by relaxing the constraints to some extent and stating the degree up to which they are feasible. Here we do not distinguish between objective function and constraints instead the two necessary conditions of maximizing the total collected score and following the time bound are represented as two goals which are conflicting as one is to be maximized and the other one is to be minimized. These are represented as linear membership functions and the rest of the constraints are considered to be crisp. The detailed explanation of the fuzzy formulation is presented in the next section.

2.3.3 Fuzzy Formulation of OP

The fuzzy version can be represented in the following way showing by tilde the parameters that have a fuzzy character:

$$\sum_{i=1}^{N-1}\sum_{j=2}^{N}\widetilde{S_i}\, x_{ij} \gtrsim S_{min} \tag{2.44}$$

$$\sum_{j=2}^{N} x_{1j} = 1 \quad , \quad \sum_{i=1}^{N-1} x_{iN} = 1 \tag{2.45}$$

$$\sum_{i=1}^{N-1} x_{ik} \leq 1 \quad \forall\, k = 2, \dots\dots, N-1 \tag{2.46}$$

$$\sum_{j=2}^{N} x_{kj} \leq 1 \quad \forall\, k = 2, \dots\dots, N-1 \tag{2.47}$$

$$\sum_{i=1}^{N-1}\sum_{j=2}^{N}\widetilde{t_{ij}}\, x_{ij} \lesssim T_{max} \tag{2.48}$$

$$2 \leq u_i \leq N \quad \forall\, i = 2, \dots\dots, N \tag{2.49}$$

$$u_i - u_j + 1 \leq (N-1)(1 - x_{ij}) \qquad \forall\, i,j = 2, \dots\dots\dots, N \tag{2.50}$$

$$x_{ij} \in \{0,1\} \quad \forall\, i,j = 1, \dots\dots, N \tag{2.51}$$

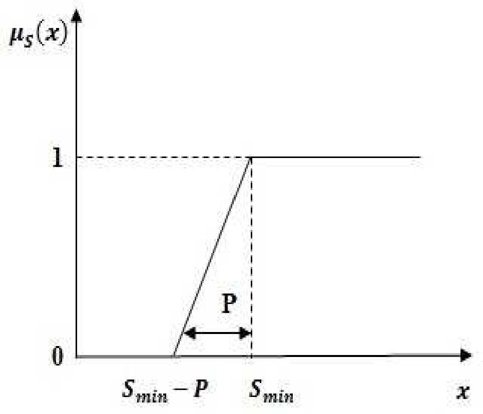
As stated above, the constraints represented by Eqs. 2.44 and 2.48 are considered to be fuzzy and represent the two conflicting goals of maximizing the total collected score and minimizing the time taken to traverse a path respectively. The remaining constraints represented by Eqs. 2.45, 2.46, 2.47, 2.49, 2.50 and 2.51 remain crisp. The variable u_i denotes the position of vertex v_i in the path and if vertex v_j is visited after vertex v_i , then x_{ij} = 1 else x_{ij} = 0. The necessary condition that each path starts in v_1 and ends in v_N is ensured by Eq. 2.45. The constraint that each path remains connected and no vertex is visited more than once in a path is taken care by Eqs. 2.46 - 2.47 and the requirement of eliminating sub tours is implemented by Eqs. 2.49 - 2.50 (Vansteenwegen, Souffriau, & Oudheusden, 2011). The symbols '$\gtrsim$' and '$\lesssim$' are the fuzzy version of '$\geq$' and '$\leq$' representing the 'fuzzy greater than or equal to' and 'fuzzy less than or equal to' respectively. The meaning of these symbols is that the constraints can be violated to some extent and depending on the importance of the constraint, this violation leads to different degree of acceptance (Jarray, 2011). For example, in Eq. 2.44, the symbol '$\gtrsim$' indicates that the total collected score of a particular path should be greater than S_{min} (most desirable case) but those paths which have their total collected score slightly less than S_{min} i.e., up to $S_{min} - P$ are also acceptable but to a lesser degree. Similarly, in Eq. 2.48, the total time taken to traverse a path should be less than T_{max} (most acceptable case) but paths having their total time greater than T_{max} are also acceptable to different degrees up to the limit of $T_{max} + L$ as signified by the symbol '$\lesssim$'. The two fuzzy goals can be represented by their membership functions as shown in Fig. 2.21 and 2.22. The fuzzy decision set $Z\ and\ Z^*$ is depicted in Fig. 2.23. It denotes the best 'compromise solution' which is obtained by the following max-min formulation:

$$Z = F_1 \cap F_2 \tag{2.52}$$

$$\mu_Z(x) = min\{\mu_T(x), \mu_S(x)\} \tag{2.53}$$

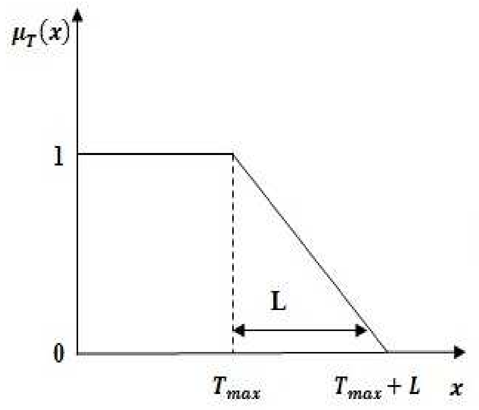
$$\mu_{Z^*}(x^*) = max\{\mu_Z(x)\} = max\{min\{\mu_T(x), \mu_S(x)\}\} \tag{2.54}$$

The intersection of the two fuzzy sets (F_1 *and* F_2) is the minimum value of the two fuzzy sets for each x, which forms the fuzzy decision set Z (as shown by dark straight lines) and the maximum value of this decision set Z forms the other set Z^* which holds the most desirable solution (as shown by the dashed line) of Fig. 2.23.



$$\mu_S(x) = \begin{cases} 0 & if\, x \leq S_{min} - P \\ \dfrac{x - S_{min} + P}{P} & if\, S_{min} - P < x < S_{min} \\ 1 & if\, x \geq S_{min} \end{cases}$$

Fig. 2.21: Membership Function for total collected score of a path



$$\mu_T(x) = \begin{cases} 0 & if\, x \geq T_{max} + L \\ \dfrac{T_{max} - x + L}{L} & if\, T_{max} < x < T_{max} + L \\ 1 & if\, x \leq T_{max} \end{cases}$$

Fig. 2.22: Membership Function for total time taken to traverse a path

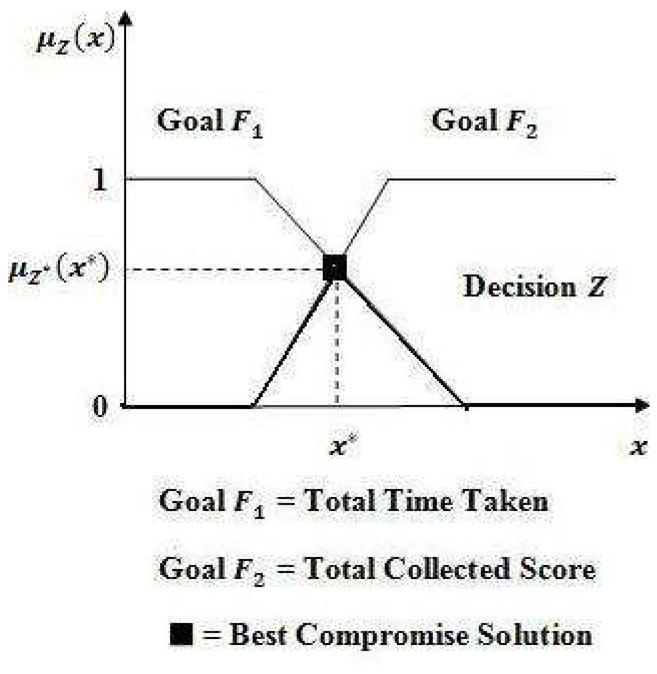

$\mu_Z(x)$
Goal F_1
Goal F_2
1
$\mu_{Z^*}(x^*)$
Decision Z
0
x^*
x
Goal F_1 = Total Time Taken
Goal F_2 = Total Collected Score
■ = Best Compromise Solution

Fig. 2.23: Fuzzy decision set Z *and* Z^*

2.3.4 FOP Algorithm

Following are the steps to determine the path that maximizes the total collected score within the specified time limit for a given $G(V, E)$ with N nodes:

1) Determine all the possible paths (W_m) connecting v_1 and v_N (using Eqs. 2.45, 2.46, 2.47, 2.49, 2.50 and 2.51).

2) For each possible path:

(a) Calculate the total time taken to traverse the path and the total collected score (using Eq. 2.17).

(b) Calculate the expected value of the total time taken to traverse the path and the total collected score (using Eq. 2.41).

(c) Calculate its membership degree for the membership function of time (using Eq. 2.48 and Fig. 2.22). Let it be denoted as fuzzy set F_1.

(d) Calculate its membership degree for the membership function of score (using Eq. 2.44 and Fig. 2.21). Let it be denoted as fuzzy set F_2.

3) Determine the feasible paths denoted by the fuzzy decision set Z obtained using definition in section 2.3.1 (b) and Eq. 2.42.

4) Most desirable path (final solution) is denoted by the fuzzy decision set Z^* obtained using definition in section 2.3.1 (b) and Eq. 2.43.

5) If the fuzzy decision set Z^* contains more than one paths, the total collected score (S_i) for each of the paths in Z^* can be ranked (using definition 2(a) of section 2.2.2 (II) and Eqs. 2.18, 2.19, 2.20 and 2.21) to determine the path that maximizes the total collected score.

2.3.5 Illustrative Example

Consider the following network $G(V, E)$ with total number of nodes $N = 5$.

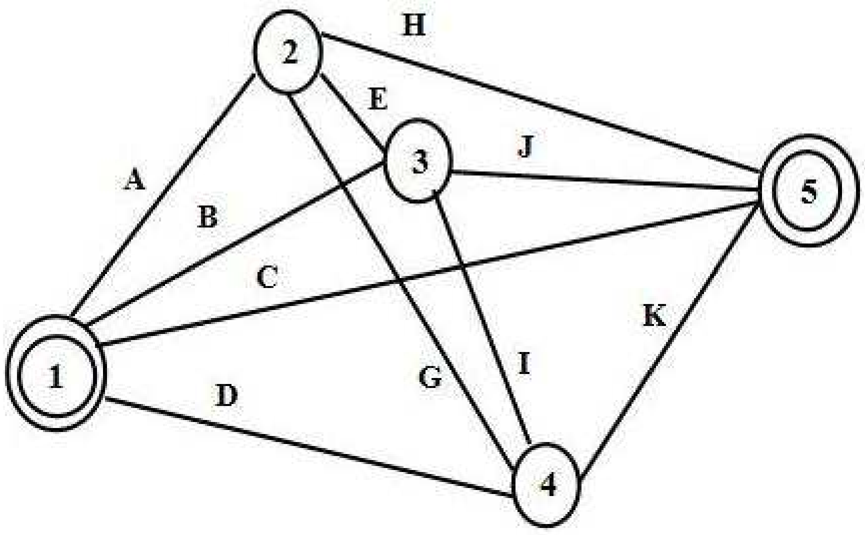


Fig. 2.24: Input Graph $G(V, E)$ with source vertex = 1 and destination vertex = 5

Table 2.12: The value of edge weights (time taken to travel from one node to another)

Edge	Label	Fuzzy Time Values
e_{12}	A	(1,4,6,9)
e_{13}	B	(6,9,13,16)
e_{14}	D	(14,15,21,22)
e_{15}	C	(4,6,8,10)
e_{23}	E	(2,3,3,4)
e_{24}	G	(5,8,8,11)
e_{25}	H	(14,18,22,26)
e_{34}	I	(5,8,12,15)
e_{35}	J	(0,2,10,12)
e_{45}	K	(1,2,2,3)
$T_{max} = 20, L = 15$		

Table 2.13: The value of node weights (score values)

Node	Label	Fuzzy Score Values
v_1	1	(1,2,8,9)
v_2	2	(8,9,11,12)
v_3	3	(3,5,9,11)
v_4	4	(17,20,24,27)
v_5	5	(1,2,4,5)
$S_{min} = 25, P = 13$		

Step 1**:** All possible paths connecting v_1 and v_5 are found that satisfy the Eqs. 2.45, 2.46, 2.47, 2.49, 2.50 and 2.51 and are as follows:

W_1: $1-5$

W_2: $1-2-5$

W_3: $1-3-5$

W_4: $1-4-5$

W_5: $1-2-3-5$

W_6: $1-3-2-5$

W_7: $1-2-4-5$

W_8: $1-4-2-5$

W_9: $1-3-4-5$

W_{10}: $1-4-3-5$

W_{11}: $1-2-3-4-5$

W_{12}: $1-2-4-3-5$

W_{13}: $1-3-4-2-5$

W_{14}: $1-4-3-2-5$

W_{15}: $1-4-2-3-5$

W_{16}: $1-3-2-4-5$

Step 2(a)**:** Table 2.14 shows the total time taken to traverse and total collected score for each of the above stated paths which is calculated using Eq. 2.17.

Step 2(b)**:** Next we calculate the expected value of the total time taken to traverse the path and the total collected score (using Eq. 2.41) for each of the paths W_1 *to* W_{16} as shown in Table 2.15.

Step 2(c) and 2(d)**:** The grade of membership of each possible path from W_1 *to* W_{16} is determined for both the membership functions (time and score) using Eqs. 2.44, 2.48 and Fig. 2.21 and 2.22. The values for the same are shown in Table 2.16.

Hence, the two fuzzy sets F_1 *and* F_2 for the total time taken and total collected score respectively, are as follows:

$$F_1 = \{W_1/1\,, W_2/0.67, W_3/1, W_4/1\,, W_5/1, W_6/0.06, W_7/1, W_8/0, W_9/0.8,$$
$$W_{10}/0.06, W_{11}/1, W_{12}/0.4, W_{13}/0, W_{14}/0, W_{15}/0, P_{16}/0.73\}$$

$$F_2 = \{W_1/0\,, W_2/0.23, W_3/0, W_4/1\,, W_5/0.77, W_6/0.77, W_7/1, W_8/1, W_9/1,$$
$$W_{10}/1, W_{11}/1, W_{12}/1, W_{13}/1\,, W_{14}/1, W_{15}/1, W_{16}/1\}$$

***Step 3*:** Now the fuzzy decision set Z is obtained using definition in section 2.3.1 (b) and Eq. 2.42 to determine all the feasible paths along with the membership grades stating the degree up to which each of the feasible paths is acceptable.

Here, $\mu_Z(x) = min\{\mu_T(x), \mu_S(x)\}\ where\ x = W_1\ to\ W_{16}$ is used calculate Z .

$$Z = \{W_1/0\,, W_2/0.23, W_3/0, W_4/1\,, W_5/0.77, W_6/0.06, W_7/1, W_8/0, W_9/0.8,$$
$$W_{10}/0.06, W_{11}/1, W_{12}/0.4, W_{13}/0\,, W_{14}/0, W_{15}/0, P_{16}/0.73\}$$

***Step 4*:** Finally, the fuzzy decision set Z^* that contains the desirable path is obtained using definition in section 2.3.1 (b) and Eq. 2.43.

$$Z^* = \{W_4/1\,, W_7/1\,, W_{11}/1\}$$

***Step 5*:** As can be observed from Step 4, there are three paths that are desirable as they satisfy all the crisp constraints and the two fuzzy goals. To conclude with the most desirable path among the three available in Z^*, we use the ranking method described using definition 2(a) of section 2.2.2 (II) and Eqs. 2.18, 2.19, 2.20 and 2.21 to rank the score of each path in Z^* and determine the best one. The values for which are shown in Table 2.17.

Table 2.14: The total time taken and total collected score for each of the paths

Path	Total Time Taken To Traverse	Total Collected Score
W_1	(4,6,8,10)	(1,2,8,9)
W_2	(15,22,28,35)	(9,11,19,21)
W_3	(6,11,23,28)	(4,7,17,20)
W_4	(15,17,23,25)	(18,22,32,36)
W_5	(3,9,19,25)	(12,16,28,32)
W_6	(22,30,38,46)	(12,16,28,32)
W_7	(7,14,16,23)	(26,31,43,48)
W_8	(33,41,51,59)	(26,31,43,48)
W_9	(12,19,27,34)	(21,27,41,47)
W_{10}	(19,25,43,49)	(21,27,41,47)
W_{11}	(9,17,23,31)	(29,36,52,59)
W_{12}	(11,22,36,47)	(29,36,52,59)
W_{13}	(30,43,55,68)	(29,36,52,59)
W_{14}	(35,44,58,67)	(29,36,52,59)
W_{15}	(21,28,42,49)	(29,36,52,59)
W_{16}	(14,22,26,34)	(29,36,52,59)

Table 2.15: The expected value of the total time taken to traverse the path and the total collected score for each of the paths

Path	EV(Total Time Taken)	EV(Total Collected Score)
W_1	7	5
W_2	25	15
W_3	17	12
W_4	20	27
W_5	14	22
W_6	34	22
W_7	15	37
W_8	46	37
W_9	23	34
W_{10}	34	34
W_{11}	20	44
W_{12}	29	44
W_{13}	49	44
W_{14}	51	44
W_{15}	35	44
W_{16}	24	44

Table 2.16: The grade of membership of each possible path for both the membership functions of time and score

Path (W_m)	$\mu_T(W_m)$	$\mu_S(W_m)$
W_1	1	0
W_2	0.67	0.23
W_3	1	0
W_4	1	1
W_5	1	0.77
W_6	0.06	0.77
W_7	1	1
W_8	0	1
W_9	0.8	1
W_{10}	0.06	1
W_{11}	1	1
W_{12}	0.4	1
W_{13}	0	1
W_{14}	0	1
W_{15}	0	1
W_{16}	0.73	1

Table 2.17: Showing the ranks of the desirable paths

Path	Score	Rank
W_4	$(18,22,32,36)$	3
W_7	$(26,31,43,48)$	2
$\boldsymbol{W_{11}}$	$(\mathbf{29,36,52,59})$	1

Thus, the most desirable path is W_{11} as it has the highest rank.

2.3.6 Parallel Formulation of FOP

To solve the FOP more efficiently and to yield a better and improved performance when applied on large instances, we present a parallel algorithm for FOP that uses the CREW PRAM (Concurrent Read Exclusive Write Parallel Random Access Machine) model. In a PRAM model, several processors are connected to a common block of shared memory. This shared memory can be accessed by all the processors and these processors communicate with each other by reading from or by writing to the shared memory. In a PRAM model, each processor is like a sequential RAM and can perform the arithmetic operations, assignment, comparison, memory access etc. in unit time. In CREW PRAM, more than one processor can read concurrently from the same cell of the shared memory but cannot write into the same cell concurrently i.e., write operation has to be exclusive (Blelloch, 1996; Horowitz, Sahni, & Rajasekaran, 1999). For each parallel algorithm, the following terms are computed to determine whether it is work-optimal or not (Horowitz, Sahni, & Rajasekaran, 1999):

$$(a)\ speed\ up = \frac{runtime\ of\ the\ best\ known\ sequential\ algorithm}{runtime\ of\ the\ parallel\ algorithm\ for\ a\ p\text{−}processor\ machine} \qquad (2.55)$$

$$(b)\ total\ work\ done = p * runtime\ of\ the\ parallel\ algorithm\ for\ a\ p - processor\ machine \qquad (2.56)$$

$$(c)\ efficiency = \frac{runtime\ of\ the\ best\ known\ sequential\ algorithm}{total\ work\ done} \qquad (2.57)$$

$Any\ parallel\ algorithm\ is\ said\ to\ be\ work\ optimal\ if\ it\ has\ a\ linear$

$speed\ up\ and\ an\ efficiency\ of\ 1.$

If we have a graph with n nodes, then the maximum number of edges possible in the path connecting the source vertex (v_1) and the destination vertex (v_n) is $n-1$. Hence, the total number of possible paths can be calculated using the following formula:

$$Total\ no.of\ paths\ (p) = \sum_{i=1}^{n-1} \frac{(n-2)!}{[n-(i+1)]!} \qquad (2.58)$$

Fig. 2.25 shows the several steps of the proposed parallel algorithm for FOP:

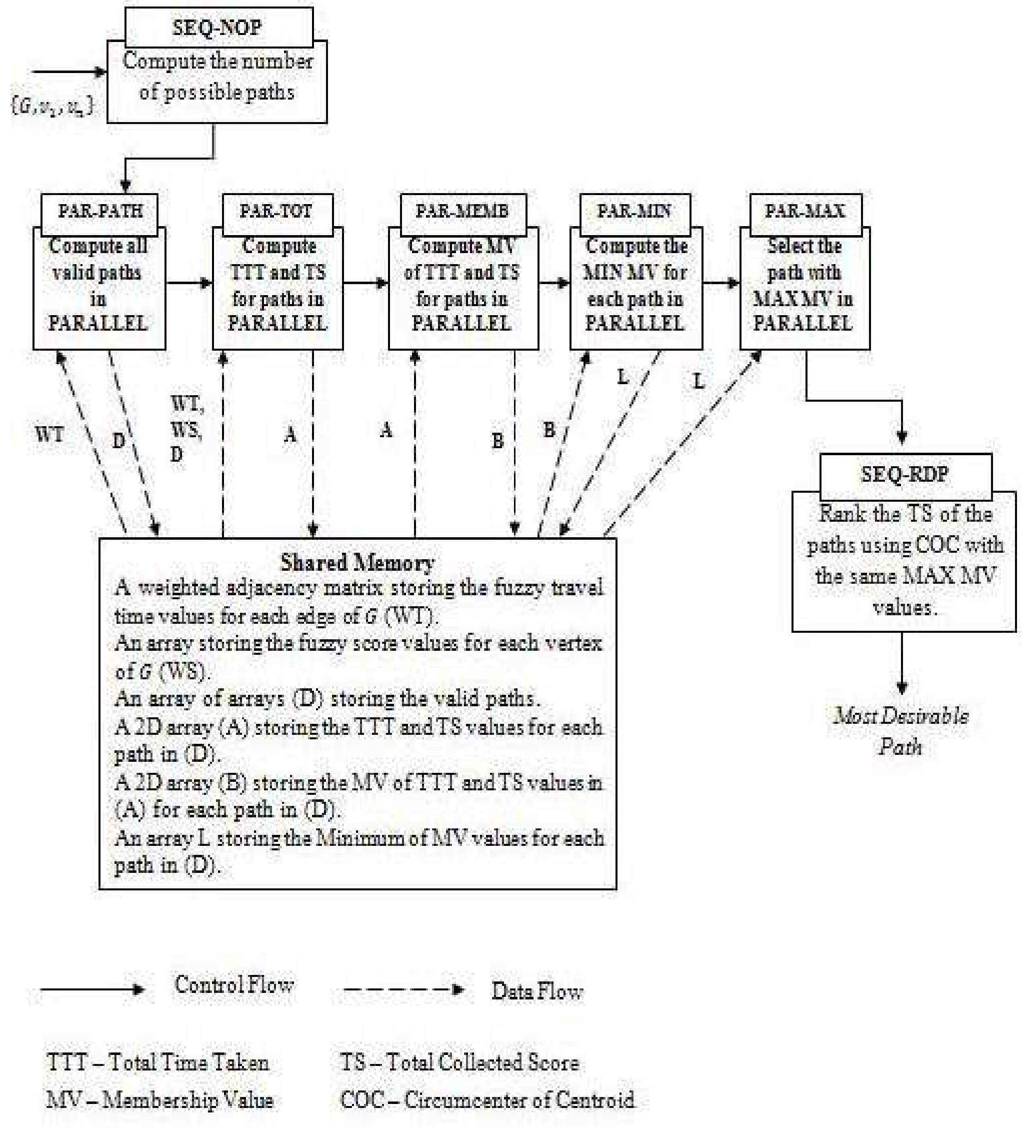

SEQ-NOP
Compute the number of possible paths
$\{G, v_1, v_n\}$
PAR-PATH
Compute all valid paths in PARALLEL
PAR-TOT
Compute TTT and TS for paths in PARALLEL
PAR-MEMB
Compute MV of TTT and TS for paths in PARALLEL
PAR-MIN
Compute the MIN MV for each path in PARALLEL
PAR-MAX
Select the path with MAX MV in PARALLEL
WT
D
WT, WS, D
A
A
B
B
L
L
SEQ-RDP
Rank the TS of the paths using COC with the same MAX MV values.
Most Desirable Path
Shared Memory
A weighted adjacency matrix storing the fuzzy travel time values for each edge of G (WT).
An array storing the fuzzy score values for each vertex of G (WS).
An array of arrays (D) storing the valid paths.
A 2D array (A) storing the TTT and TS values for each path in (D).
A 2D array (B) storing the MV of TTT and TS values in (A) for each path in (D).
An array L storing the Minimum of MV values for each path in (D).
Control Flow
Data Flow
TTT – Total Time Taken
MV – Membership Value
TS – Total Collected Score
COC – Circumcenter of Centroid

Fig. 2.25: Several steps of the parallel formulation of FOP

In the parallel formulation of FOP, the module $\boldsymbol{SEQ-NOP}$ computes the total number of possible paths (p) for a given input graph $\{G, v_1, v_n\}$ using Eq. 2.58 sequentially. We assume the number of processors is equal to the total number of paths (p). This value is utilized by the following parallel modules to output the path in G connecting v_1 and v_n that maximizes the total collected score and minimizes the total travel time. If the module $\boldsymbol{PAR-MAX}$ returns more than one path with the same maximum membership value (MV) then the module $\boldsymbol{SEQ-RDP}$ sequentially ranks the total collected score (TS) of the paths with the same maximum value of MV using the COC method of ranking fuzzy numbers (using definition 2(a) of section 2.2.2 (II) and Eqs. 2.18, 2.19, 2.20 and 2.21). Finally, the path with the highest rank is returned as the *most desirable path.*

The entire operation that is performed in parallel has been divided into modules, namely $\boldsymbol{PAR-PATH}, \boldsymbol{PAR-TOT}, \boldsymbol{PAR-MEMB}, \boldsymbol{PAR-MIN}, \boldsymbol{PAR-MAX}$. The output of one parallel module forms the input for the other parallel module. The module $\boldsymbol{PAR-PATH}$ computes all the valid paths in parallel and is stored in an array of arrays D. This array is then used by the $\boldsymbol{PAR-TOT}$ module to compute the total time taken (TTT) and total collected score (TS) of each valid path in D in parallel. The TTT and TS values generated for each path in D are then stored in a 2D array A. This array A is then taken as input by the next parallel module $\boldsymbol{PAR-MEMB}$. In $\boldsymbol{PAR-MEMB}$, the membership value of TTT and TS of each path is calculated and stored in another 2D array B. The next parallel module $\boldsymbol{PAR-MIN}$ considers array B as input and for each path the minimum membership value is determined out of the two membership values (one for TTT and the other for TS) in parallel. These minimum membership values are then stored in an array L. The last parallel module $\boldsymbol{PAR-MAX}$ generates those paths which have the maximum membership values (can be more than one path) using the array L as input. The pseudo code for each parallel module is presented below. For each parallel module, a theoretical analysis has been performed and the speed up and total work done has been computed to determine the efficiency and work optimality of the parallel module.

```
$\boldsymbol{PAR-PATH}$
$Processor\ k\ (\boldsymbol{in\ parallel}\ for\ 1 \le k \le p)\ \boldsymbol{does}$:
{
$cp := compute\ a\ unique\ path\ connecting\ v_1\ and\ v_n$ ;
    $if(cp == VALID)\ then$
// A path is valid if it satisfies the constraints stated in the Eqs. 2.45, 2.46, 2.47, 2.49, 2.50 and 2.51.
            $D_k := cp$;
    $else$
        $DISCARD\ cp;$
}
```

The p valid paths can be computed in $O(1)$ time using p CREW PRAM processors.

Therefore,

$speed\ up = \frac{O(p)}{O(1)} = O(p)$

$total\ work\ done = O(1) * p = O(p)$

$efficiency = \frac{O(p)}{O(p)} = 1$

$Therefore, this\ parallel\ module\ is\ work - optimal.$

```
$\boldsymbol{PAR - TOT}$
$Processor\ k\ (\textbf{in parallel}\ for\ 1 \le k \le p)\ \textbf{does}:$
{
    $for(\ j = 0)\ do:$
     {
       $A_{kj} \coloneqq Calculate\ the\ TTT\ for\ each\ VALID\ path\ in\ D_k;$
       // TTT (Total Time Taken) for each VALID path is calculated using Eq. 2.17.
     }

    $for(\ j = 1) do:$

     {
       $A_{kj} \coloneqq Calculate\ the\ TS\ for\ each\ VALID\ path\ in\ D_k;$
       // TS (Total Collected Score) for each VALID path is calculated using Eq. 2.17.
     }
}
```

The values for TTT and TS for each of the paths in D_k can be obtained in $O(1)$ time using p CREW PRAM processors.

Therefore,

$$speed\ up = \frac{O(p)}{O(1)} = O(p)$$

$$total\ work\ done = O(1) * p = O(p)$$

$$efficiency = \frac{O(p)}{O(p)} = 1$$

$Therefore, this\ parallel\ module\ is\ work - optimal.$

$\boldsymbol{PAR - MEMB}$

```
$Processor\ k\ (\textbf{in parallel}\ for\ 1 \leq k \leq p)\textbf{does}$:
{
        $for(\ j = 0)\ do$:
          {
        $B_{kj} \coloneqq Calculate\ the\ MV\ for\ TTT$ in $A_{kj}\ for\ each\ VALID\ path\ in\ D_k$;
// MV (Membership Value) for TTT (Total Time Taken) is calculated using Eqs. 2.41, 2.48 and Fig. 2.22.
          }
        $for(\ j = 1)do$:
          {
          $B_{kj} \coloneqq Calculate\ the\ MV\ for\ TS$ in $A_{kj}\ for\ each\ VALID\ path\ in\ D_k$;
// MV (Membership Value) for TS (Total Collected Score) is calculated using Eqs. 2.41, 2.44 and Fig. 2.21.
          }
}
```

The membership values for TTT and TS for each of the paths in D_k can be calculated in $O(1)$ time using p CREW PRAM processors.

Therefore,

$$speed\ up = \frac{O(p)}{O(1)} = O(p)$$

$$total\ work\ done = O(1) * p = O(p)$$

$$efficiency = \frac{O(p)}{O(p)} = 1$$

$Therefore, this\ parallel\ module\ is\ work - optimal.$

```
$\boldsymbol{PAR - MIN}$
$Processor\ k\ (\boldsymbol{in\ parallel}\ for\ 1 \le k \le p)\boldsymbol{does}$:
{
    $for(\ j = 0)\ do$:
     {
       $\mathrm{L[k] := MIN(B[k][j], B[k][j+1])}$
  // Compute the minimum of the two values using definition in section 2.3.1 (b) and Eq. 2.42.
     }
}
```

The minimum of the two membership values for TTT and TS for each of the paths in D_k can be computed in $O(1)$ time using p CREW PRAM processors. Therefore,

$$speed\ up = \frac{O(p)}{O(1)} = O(p)$$

$$total\ work\ done = O(1) * p = O(p)$$

$$efficiency = \frac{O(p)}{O(p)} = 1$$

$Therefore, this\ parallel\ module\ is\ work - optimal.$

```
$\boldsymbol{PAR - MAX}$
$Processor\ k\ (\boldsymbol{in\ parallel}\ for\ 1 \le k \le p)\boldsymbol{does}$:
{
    $G[k] := GRADE\ of\ L[k]$;
// GRADE of any element L[$k$] in $L$ is equal to one plus the number of elements in $L$ smaller than L[$k$].

    PRINT the path in D with the greatest $GRADE$ in $G$.
}
```

The maximum value amongst the values present in the array L can be determined in $O(p)$ time using p CREW PRAM processors.

Therefore,

$$speed\ up = \frac{O(p^2)}{O(p)} = O(p)$$

$$total\ work\ done = O(p) * p = O(p^2)$$

$$efficiency = \frac{O(p^2)}{O(p^2)} = 1$$

$$Therefore, this\ parallel\ module\ is\ work - optimal.$$

2.4 Conclusion

In this chapter, we have considered the fuzzy version of a few graph problems and presented some solutions to those problems using fuzzy numbers and their ranking methods. We have suggested a method of ranking for the Quasi-Gaussian Fuzzy Numbers called the Link Preference Index. This was applied to the problems shortest path, minimum spanning tree and steiner tree and a solution for the fuzzy environment was generated. The method chosen for the intermediate Fuzzy Arithmetic operations leads to simpler calculations and satisfactory results. At the same time, the method can be applied in varied situations like when the value for k is small, the number of sub intervals are less but the evaluation is faster and can be implemented on a slower machine. However, for a faster machine by increasing the value of k we can get more accurate results. Also, an Extended Dijkstra's algorithm and Extended Fuzzy Prim's algorithm has been proposed. The LPI proposed in this paper is a monotonic decreasing function of $\Delta\bar{x}$. It can also take care of non symmetry in the membership function. It tends to increase if Z_{min} is long tailed towards right or Z_i is long tailed towards left (i.e., the two functions lean towards each other).

This chapter also deals with the representation of trapezoidal fuzzy number and its application to the constrained shortest path problem. We have dealt with two aspects viz. uncertainty and multiple constraints in the shortest path problem. The original PDA algorithm was suggested by Chen *et al.* (2008) and its fuzzy version for CFSPP is capable of tackling the uncertainty by representing the parameter cost as a fuzzy number and the other parameter viz. delay, is represented by the crisp value in the same manner as in (Chen, Song, & Sahni, 2008). To determine the ranking of the fuzzy numbers we have used three different techniques which include Circumcenter of Centroids, Maximizing Set / Minimizing Set and Weighting Function and by comparing different output values of cost, path discretization error and CPU execution time with respect to varying delay requirement, it has been concluded that the most efficient method of them, is the WF technique. Thus, the shortest path between a specified source and target, taking into consideration the multiple delay constraints and the uncertainty, can be most efficiently and accurately determined using WF as it is the best method in terms of execution time and cost. However, the method that gives minimum path discretization error is Circumcenter of Centroid method. Also, an application in wireless sensor networks was presented. As WSN's are battery operated, we require a path that consumes minimum energy and also satisfies an end-to-end delay constraint. As discussed in the above section, we can compute one path for every situation using CFSPP i.e., if the WSN does not have enough battery power then we compromise with the path that consumes more energy but can be obtained with lesser delay else the most desirable path that consumes minimum energy can be obtained but this increases the delay. The value of the two parameters involved, namely delay and energy cannot be predicted precisely. Therefore, here we represent them using TFN and use the COC method of ranking for determining the cheapest feasible path. Thus we show that the CFSPP algorithm performs well in transmitting the sensor outputs like temperature, pressure, gas concentration etc. to the designated destination with minimum delay while conserving the energy. Although, in this chapter we have used trapezoidal fuzzy numbers, our method can be easily extended to any other type of fuzzy numbers by choosing an appropriate ranking method. The fuzzy algorithm proposed has the same complexity as the crisp version yet the number of arithmetic operations increases. This is the price one has to pay for modelling uncertainty.

Another problem that has been dealt with was the orienteering problem and a method was suggested to solve the fuzzy orienteering problem. In the literature, the integer program formulation of OP has been stated. For these kinds of formulations, the objective function (either maximization or minimization) and the constraints are to be followed strictly in mathematical sense. However, this may not be the case in real life application where the decision maker may be willing to relax the constraints to some extent to generate a solution that can achieve a level of satisfaction which cannot be represented by the crisp formulation but the fuzzy version can take care of these requirements. In our work, the two necessary conditions of maximizing the total collected score and that too within the specified time bound is represented by two fuzzy goals and to take care of the other requirements like each vertex is visited once, no sub tours are formed etc. are represented in the form of crisp constraints. By representing the two necessary conditions in the form of fuzzy goals we provide latitude to the desired solution by relaxing the constraints. To make the proposed method applicable to large instances, we also present a work-optimal parallel formulation of FOP. When provided with several graphs as input, further speedup can be expected since the stages will form a 5 stage linear pipeline providing a speed up of 5 ideally when all the stages are busy. In this way the method suggested above provides a more practical illustration of the physical world and generates a solution that is more appropriate and realistic.