International Journal of Mathematical Archive-9(5), 2018, 84-89
Available online through www.ijma.info ISSN 2229 – 5046

A STUDY ON FUZZY α -MINIMUM SPANNING TREE GRAPH WITH ALGORITHM

Dr. M. VIJAYA[1] AND B. MOHANAPRIYAA*[2]

[1,2]P.G. and Research Department of Mathematics,
Marudu Pandiyar College, Vallam, Thanjavur 613 403. India.

(Received On: 24-04-18; Revised & Accepted On: 16-05-18)

1. ABSTACT

In this paper, we make a further study of the minimum spanning tree problem with fuzzy edge weights. We propose the concept of fuzzy α-minimum spanning trees based on the credibility measure defined by Liu and Liu (2002), and then discuss this problem under different conditions. For the general case in which the edge weights are general fuzzy numbers, a hybrid intelligent algorithm is designed to solve it.

Keywords: *Minimum spanning tree, path optimality condition, fuzzy simulation, genetic algorithm.*

2. INTRODUCTION

The fuzzy α-minimum spanning tree problem is investigated on the graph with fuzzy edge weights. The notion of fuzzy α-minimum spanning tree is presented based on the credibility measure, and then the solutions of the fuzzy α-minimum spanning tree problem are discussed under different assumptions. First, we respectively, assume that all the edge weights are triangular fuzzy numbers and trapezoidal fuzzy numbers and prove that the fuzzy α-minimum spanning tree problem can be transformed to a classical problem on a crisp graph in these two cases, which can be solved by classical algorithms such as the Kruskal algorithm and the Prim algorithm in polynomial time. Subsequently, as for the case that the edge weights are general fuzzy numbers, a fuzzy simulation-based genetic algorithm using Puffer number representation is designed for solving the fuzzy α-minimum spanning tree problem. Some numerical examples are also provided for illustrating the effectiveness of the proposed solutions.

3. MINIMUM SPANNING TREE DEFINITION AND HEOREMS

3.1.1. Definition: MINIMUM MASS SPANNING TREE is a subset of the boundaries of a connected, boundaries -weighted undirected graph that connect all the vertices mutually, without any cycles and with the minimum potential total boundaries weight. That is, it is a spanning tree whose sum of boundaries weights is as small as possible. More generally, any boundaries -weighted undirected graph (not necessarily connected) has a **minimum spanning forest**, which is a union of the minimum spanning trees for its connected components.

There are quite a few use cases for minimum spanning trees. One **example** would be a telecommunications company trying to lay cable in a new neighborhood. If it is constrained to bury the cable only along certain paths (e.g. roads), then there would be a graph containing the points (e.g. houses) connected by those paths. Some of the paths might be more expensive, because they are longer, or require the cable to be buried deeper; these paths would be represented by edges with larger weights. Currency is an acceptable unit for edge weight – there is no requirement for edge lengths to obey normal rules of geometry such as the triangle inequality. A spanning tree for that graph would be a subset of those paths that has no cycles but still connects every house; there might be several spanning trees possible. A minimum spanning tree would be one with the lowest total cost, representing the least expensive path for laying the cable.

3.1. 2. Definition: A MINIMUM SPANNING TREE (MST
Given a connected graph $G = (V, E,\omega)$, a spanning tree T^0, is said to be a minimum spanning tree if

$$W(T^0, \omega) \leq W(T, \omega)$$

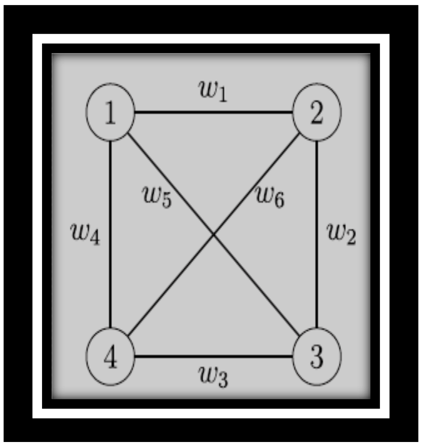
holds for any spanning tree T.

Corresponding Author: B. Mohanapriyaa*[2]
[2]P.G. and Research Department of Mathematics,
Marudu Pandiyar College, Vallam, Thanjavur 613 403. India.

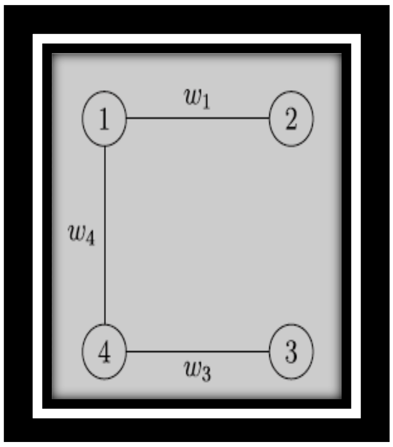
Example: A connected graph with 4 vertices and 6 edges is shown in Figure1(a), where w_i, $i = 1, 2, \dots, 6$, denote the edge weights, and a spanning tree T^0 is given in Figure 1(b). We denote the spanning tree T^0 by its edge set, i.e. $T^0 = \{e_1, e_3, e_4\}$. The set of non-tree edges is $E\backslash T^0 = \{e_2, e_5, e_6\}$. For a non-tree edge, say e_5, the tree path $P_5 = \{e_3, e_4\}$. If T^0 is a minimum spanning tree, neither w_3 nor w_4 is larger than w_5. Otherwise, we can produce a new spanning tree T' with less weight by replacing the bigger one in $\{e_3, e_4\}$ with e_5.

3.1.3. POSSIBLE MULTIPLICITY

If there are n vertices in the graph, then each spanning tree has $n - 1$ edges. This figure shows there may be more than one minimum spanning tree in a graph. In the figure, the two trees below the graph are two possibilities of minimum spanning tree of the given graph.



1. (A): Original Graph



1. (B): Spanning Tree Graph

There may be several minimum spanning trees of the same weight; in particular, if all the edge weights of a given graph are the same, then every spanning tree of that graph is minimum.

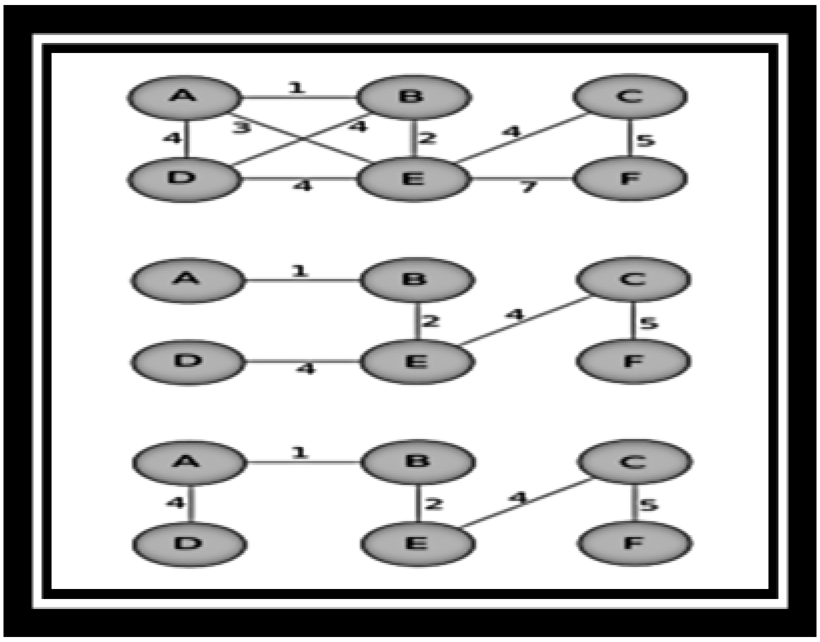


Figure-2: Minimum Spanning Tree

3.2.1. THEOREM: UNIQUENESS OF MINIMUM SPANNING TREE

If each edge has a distinct weight then there will be only one, unique minimum spanning tree. This is true in many realistic situations, such as the telecommunications company example above, where it's unlikely any two paths have exactly the same cost. This generalizes to spanning forests as well.

Proof:

1. Assume the contrary, that there are two different MSTs A and B.
2. Since A and B differ despite containing the same nodes, there is at least one edge that belongs to one but not the other. Among such edges, let e_1 be the one with least weight; this choice is unique because the edge weights are all distinct. Without loss of generality, assume e_1 is in A.

3. As *B* is a MST, $\{e_1\}$ *B* must contain a cycle *C*.
4. As a tree, *A* contains no cycles; therefore *C* must have an edge e_2 that is not in *A*.
5. Since e_1 was chosen as the unique lowest-weight edge among those belonging to exactly one of *A* and *B*, the weight of e_2 must be greater than the weight of e_1.
6. Replacing e_2 with e_1 in *B* therefore yields a spanning tree with a smaller weight.
7. This contradicts the assumption that *B* is a MST.

More generally, if the edge weights are not all distinct then only the (multi-)set of weights in minimum spanning trees is certain to be unique; it is the same for all minimum spanning trees.

3.3. Definition: MINIMUM-COST SUBGRAPH

If the weights are positive, then a minimum spanning tree is in fact a minimum-cost subgraph connecting all vertices, since subgraph containing cycles necessarily have more total weight.

3.2.2. THEOREM: CYCLE PROPERTY

For any cycle C in the graph, if the weight of an edge e of C is larger than the individual weights of all other edges of C, then this edge cannot belong to a MST.

Proof: Assume the contrary, i.e. that *e* belongs to an MST T_1. Then deleting *e* will break T_1 into two sub trees with the two ends of *e* in different sub trees. The remainder of *C* reconnects the sub trees, hence there is an edge *f* of *C* with ends in different sub trees, i.e., it reconnects the sub trees into a tree T2 with weight less than that of T_1, because the weight of *f* is less than the weight of *e*.

3.2.3. THEOREM: CUT PROPERTY

This figure shows the cut property of MSTs. T is the only MST of the given graph. If S = {A, B, D, E} thus V-S ={C, F}, then there are 3 possibilities of the edge across the cut (S, V-S), they are edges BC,EC,EF of the original graph. Then e is one of the minimum-weight –edge for the cut therefore S∪{e} is part of the MST T.

For any cut C of the graph if the weight of an edge e in the cut-set of C is strictly smaller than the weights of all other edges of the cut-set of C, then this edge belongs to all MSTs of the graph.

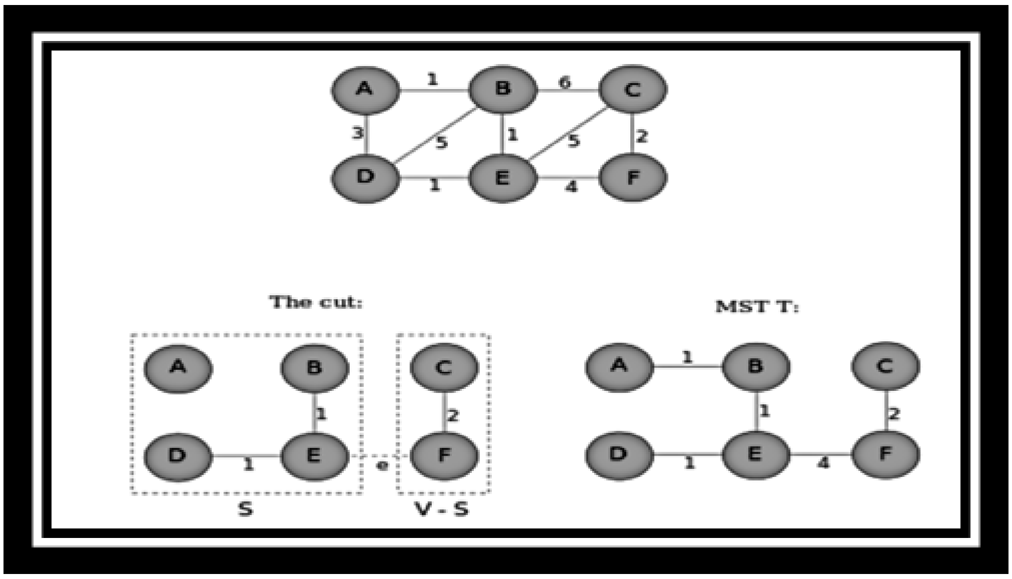


Figure-3-Minimum-Weight –Edge Tree

Proof: Assume that there is a MST *T* that does not contain *e*. Adding *e* to *T* will produce a cycle, that crosses the cut once at *e* and crosses back at another edge *e'* . Deleting *e'* we get a spanning tree *T\{e'}U{e}* of strictly smaller weight than *T*. This contradicts the assumption that *T* was a MST.

By a similar argument, if more than one edge is of minimum weight across a cut, then each such edge is contained in some minimum spanning tree.

3.2.4. THEOREM: MINIMUM-COST EDGE

If the minimum cost edge e of a graph is unique, then this edge is included in any MST.

Proof: if *e* was not included in the MST, removing any of the (larger cost) edges in the cycle formed after adding *e* to the MST would yield a spanning tree of smaller weight.

3.2.5. CONTRACTION
If T is a tree of MST edges, then we can contract T into a single vertex while maintaining the invariant that the MST of the contracted graph plus T gives the MST for the graph before contraction.

4. ALGORITHMS

In all of the algorithms below, *m* is the number of edges in the graph and *n* is the number of vertices.

4.1. CLASSIC ALGORITHMS
The first algorithm for finding a minimum spanning tree was developed by Czech scientist Otakar Borůvka's in 1926 (see Borůvka's algorithm). Its purpose was an efficient electrical coverage of Moravia. The algorithm proceeds in a sequence of stages. In each stage, called *Boruvka step*, it identifies a forest *F* consisting of the minimum-weight edge incident to each vertex in the graph *G*, and then forms the graph $G^{'}$ as the input to the next step. Here G' denotes the graph derived from *G* by contracting edges in *F* (by the Cut property, these edges belong to the MST). Each Boruvka step takes linear time. Since the number of vertices is reduced by at least half in each step, Boruvka's algorithm takes O(m log n) time.
A second algorithm is Prim's algorithm, which was invented by Jarnik in 1930 and rediscovered by Prim in 1957 and Dijkstra in 1959. Basically, it grows the MST (*T*) one edge at a time. Initially, *T* contains an arbitrary vertex. In each step, *T* is augmented with a least-weight edge (x, y) such that x is in *T* and y is not yet in *T*. By the Cut property, all edges added to *T* are in the MST. Its run-time is either O(m log n) or O($m + n$ log n), depending on the data-structures used. A third algorithm commonly in use is Kruskal's algorithm, which also takes O(m log n) time.
A fourth algorithm, not as commonly used, is the reverse-delete algorithm, which is the reverse of Kruskal's algorithm. Its runtime is O(m log n (log $n)^3$).
All these four are greedy algorithms. Since they run in polynomial time, the problem of finding such trees is in **FTP**, and related decision problems such as determining whether a particular edge is in the MST or determining if the minimum total weight exceeds a certain value are in **P**.

4.2. FASTER ALGORITHMS
Several researchers have tried to find more computationally-efficient algorithms.

In a comparison model, in which the only allowed operations on edge weights are pairwise comparisons, Karger, Klein & Tarjan (1995) found a linear time randomized algorithm based on a combination of Borůvka's algorithm and the reverse-delete algorithm.

The fastest non-randomized comparison-based algorithm with known complexity, by Bernard Chazelle, is based on the soft heap, an approximate priority queue. Its running time is $O(m\ \alpha(m,\ n))$, where α is the classical functional inverse of the Ackermann function. The function α grows extremely slowly, so that for all practical purposes it may be considered a constant no greater than 4; thus Chazelle's algorithm takes very close to linear time.

4.3. LINEAR-TIME ALGORITHMS IN SPECIAL CASE

4.3.1. DENSE GRAPHS
If the graph is dense (i.e. $m/n \geq \log n$), then a deterministic algorithm by Fredman and Tarjan finds the MST in time O(m). The algorithm executes a number of phases. Each phase executes Prim's algorithm many times, each for a limited number of steps. The run-time of each phase is O(m+n). If the number of vertices before a phase is log n, the number of vertices remaining after a phase is at most (o[m]). Hence, at most o(m) phases are needed, which gives a linear run-time for dense graphs.

There are other algorithms that work in linear time on dense graphs.

4.3.2. INTEGER WEIGHTS
If the edge weights are integers represented in binary, then deterministic algorithms are known that solve the problem in O(m + n) integer operations. Whether the problem can be solved deterministically for a general graph in linear time by a comparison-based algorithm remains an open question.

4.3.3. DECISION TREES
Given graph *G* where the nodes and edges are fixed but the weights are unknown, it is possible to construct a binary decision tree (BDT) for calculating the MST for any permutation of weights. Each internal node of the BDT contains a comparison between two edges, e.g. "Is the weight of the edge between x and y larger than the weight of the edge between w and z". The two children of the node correspond to the two possible answers "yes" or "no". In each leaf of the BDT, there is a list of edges from *G* that correspond to an MST. The runtime complexity of a BDT is the largest number of queries required to find the MST, which is just the depth of the BDT. A DT for a graph *G* is called *optimal* if it has the smallest depth of all correct DTs for *G*.

For every integer *r*, it is possible to find optimal decision trees for all graphs on *r* vertices by brute-force search. This search proceeds in two steps.

4.3.4. GENERATING ALL POTENTIAL BDTS

- There are BDT different graphs on n vertices.
- For each graph, an MST can always be found using n(n-1) comparisons, e.g. by Prim's algorithm.
- Hence, the depth of an optimal BDT is less than O(m + n).
- Hence, the number of internal nodes in an optimal BDT is less than O(m + n) ..
- Every internal node compares two edges. The number of edges is at most O(m + n). so the different number of comparisons is at most O(m + n)..
- Hence, the number of potential BDTs is less than: .

4.3.5. IDENTIFYING THE CORRECT BDTS To check if a BDT is correct, it should be checked on all possible permutations of the edge weights.

- The number of such permutations is at most O(m + n) ..
- For each permutation, solve the MST problem on the given graph using any existing algorithm, and compare the result to the answer given by the BDT.
- The running time of any MST algorithm is at most O(m + n) ., so the total time required to check all permutations is at most O(m + n) .

Hence, the total time required for finding an optimal BDT for *all* graphs with *r* vertices is: B O(m + n) .BDT, which is less than:

4.4. OPTIMAL ALGORITHM

Seth Pettie and Vijaya Ramachandran have found a provably optimal deterministic comparison-based minimum spanning tree algorithm. The following is a simplified description of the algorithm.

1. Let O(m + n) ., where n is the number of vertices. Find all optimal decision trees on r vertices. This can be done in time O(n) (see Decision trees above).
2. Partition the graph to components with at most r vertices in each component. This partition can be done in time O(m).
3. Use the optimal decision trees to find an MST for each component.
4. Contract each connected component spanned by the MSTs to a single vertex.
5. It is possible to prove that the resulting graph has at most n/r vertices. Hence, the graph is dense and we can use any algorithm which works on Dense graphs in time O(m).

The runtime of all steps in the algorithm is O(m), except for the step of using the decision trees. We don't know the runtime of this step, but we know that it is optimal - no algorithm can do better than the optimal decision tree.
Thus, this algorithm has the peculiar property that it is provably optimal although its runtime complexity is unknown.

4.5. PARALLEL AND DISTRIBUTED ALGORITHMS

Research has also considered parallel algorithms for the minimum spanning tree problem. With a linear number of processors it is possible to solve the problem in O(n) time. Bader & Cong (2006) demonstrate an algorithm that can compute MSTs 5 times faster on 8 processors than an optimized sequential algorithm.

Other specialized algorithms have been designed for computing minimum spanning trees of a graph so large that most of it must be stored on disk at all times. These external storage algorithms, for example as described in "Engineering an External Memory Minimum Spanning Tree Algorithm" by Roman, Dementiev *et al.*, can operate, by authors' claims, as little as 2 to 5 times slower than a traditional in-memory algorithm. They rely on efficient external storage sorting algorithms and on graph contraction techniques for reducing the graph's size efficiently.

The problem can also be approached in a distributed manner. If each node is considered a computer and no node knows anything except its own connected links, one can still calculate the distributed minimum spanning tree.

Vertices	Expected Size	Approximative Expected Size
2	1 / 2	0.5
3	3 / 4	0.75
4	31 / 35	0.8857143
5	893 / 924	0.9664502
6	278 / 273	1.0183151
7	30739 / 29172	1.053716
8	199462271 / 184848378	1.0790588
9	126510063932 / 115228853025	1.0979027

For uniform random weights in O(m + n)., the exact expected size of the minimum spanning tree has been computed for small complete graphs.

5. APPLICATIONS

Minimum spanning trees have direct applications in the design of networks, including computer networks, telecommunications networks, transportation networks, water supply networks, and electrical grids (which they were first invented for, as mentioned above). They are invoked as subroutines in algorithms for other problems, including the Christofides algorithm for approximating the traveling salesman problem, approximating the multi-terminal minimum cut problem (which is equivalent in the single-terminal case to the maximum flow problem), and approximating the minimum-cost weighted perfect matching. Other practical applications based on minimal spanning trees include:

- Taxonomy.
- Cluster analysis: clustering points in the plane, single-linkage clustering (a method of hierarchical clustering), graph-theoretic clustering, and clustering gene expression data.
- Constructing trees for broadcasting in computer networks. On Ethernet networks this is accomplished by means of the Spanning tree protocol.
- Image registration and segmentation– see minimum spanning tree-based segmentation.
- Curvilinear feature extraction in computer vision.
- Handwriting recognition of mathematical expressions. Circuit design: implementing efficient multiple constant multiplications, as used in finite impulse response filters.
- Regionalization of socio-geographic areas, the grouping of areas into homogeneous, contiguous regions.
- Comparing ecotoxicology data.
- Topological observability in power systems.
- Measuring homogeneity of two-dimensional materials.
- Minimax process control.
- Minimum spanning trees can also be used to describe financial markets. A correlation matrix can be created by calculating a coefficient of correlation between any two stocks. This matrix can be represented topologically as a complex network and a minimum spanning tree can be constructed to visualize relationships.

6. CONCLUSION

Since the applications of the minimum spanning tree problem encountered in practice usually involve some fuzzy issues, the edge weights cannot be explicitly determined. In this paper, we discussed the minimum spanning tree problem on a graph, whose edge weights are assumed as fuzzy variables, and proposed the notion of fuzzy α-minimum spanning trees.

7. REFERANCES

1. P. Bhattacharya, F. Suraweera, An Algorithm to compute the max–min powers and a property of fuzzy graphs, Pattern Recognition Letters 12 (1991) 413–420.
2. P. Bhattacharya, Some remarks on fuzzy graphs, Pattern Recognition Letters 6 (1987) 297–302.
3. K.R. Bhutani, On automorphisms of fuzzy graphs, Pattern Recognition Letters 9 (1989) 159–162.
4. K.R. Bhutani, A. Rosenfeld, Strong arcs in fuzzy graphs, Information Sciences 152 (2003) 319–322.
5. K.R. Bhutani, A. Rosenfeld, Fuzzy end nodes in fuzzy graphs, Information Sciences 152 (2003) 323–326.
6. K.R. Bhutani, A. Rosenfeld, Geodesics in fuzzy graphs, Electronic Notes in Discrete Mathematics 15 (2003) 51–54.
7. K.R. Bhutani, A. Battou, On M-strong fuzzy graphs, Information Sciences 1–2 (2003) 103–109.
8. M.L. McAllister, Fuzzy intersection graphs, Computational Mathematics, Applied 15 (1988) 871–886.
9. J.N. Mordeson, P.S. Nair, Fuzzy Graphs and Fuzzy Hypergraphs, Physica-Verlag, 2000.
10. J.N. Mordeson, P.S. Nair, Cycles and cocycles of fuzzy graphs, Information Sciences 90 (1996) 39–49.
11. J.N. Mordeson, Fuzzy line graphs, Pattern Recognition Letters 14 (1993) 381–384.
12. J.N. Mordeson, C.S. Peng, Operations on fuzzy graphs, Information Sciences 79 (1994) 159–170.