Alfarama Journal of Basic & Applied Sciences
https://ajbas.journals.ekb.eg
ajbas@sci.psu.edu.eg
Faculty of Science Port Said University
http://sci.psu.edu.eg/en/
ISSN 2682-275X
January 2022, Volume 3 Issue I
DOI: 10.21608/AJBAS.2021.90703.1062
Submitted: 19-08-2021
Accepted: 07-09-2021
Pages: 138-151

Using Optimization Algorithms for Solving Shortest Path Problems

Eman Yousif [1,*], Ahmed Salama [1], M. Elsayed Wahed [2]

[1]Department of Mathematics, Faculty of Sience, Port Said University; Port Said, Egypt,

[2]Faculty Of Computers and Informatics, Suez Canal University, Ismailia ,Egypt,

*** Corresponding author**: emanyousif79@hotmail.com

ABSTRACT

In this paper, the researcher will propose two distinct genetic and ant colony techniques for addressing shortest path issues, a classic computer science problem. First, the researcher will go through the basics of the shortest path issue, genetic algorithms, and ant colony algorithms. Then, the researcher will discuss the benefits and drawbacks of employing genetic and ant colony algorithms to solve the shortest path problem, as well as my thoughts on the solutions' utility and the future of this field of computer science. In this research study, the researcher presents two possible approaches. The first technique has been explored and the performance of a proposed network in terms of packet delay, throughput, and bandwidth consumption has been given utilizing Fuzzy Optimization (FO) as a routing protocol with network coding. The second method is based on GA by using crossover, mutation and selection operator will help in determining which solutions are to be preserved and allowed to reproduce and which ones deserve to die out. Also, it will help in focusing research in promising areas of the search space.

***Keywords*:**
genetic programming, ant colony algorithms, shortest path, optimization problems.

1. INTRODUCTION

The shortest path problem (SPP) is classic in the computer science community. It has been studied by many people, but the current standard is Djikstra's shortest path algorithm, which utilizes dynamic programming to solve the problem.

Essentially what the shortest path problem deals with is if you have a graph G = (N,V); where N is a set of nodes or locations and V is a set of vertices that connect nodes in N where V is a subset of NxN. In SPP each vertex in V also has a weight associated with it and the problem that needs to be solved it how to get from any node in N to any other node in N with the lowest weight on the vertices used.

A simple example is to let N be all the airports serviced by an airline, and V is the flights for that airline. Additionally, let the weight of V be the cost of each flight. The problem to be solved in this situation would be to find the cheapest way to get from any airport serviced to any other airport.

Djikstra developed an algorithm for this that can solve the problem that runs in O(n log n) time. However, it needs to be recalculated every time there is a change. The goal is to find an algorithm that can adapt to a changing graph topology in semi-real time

2. PRE-REQUISITES

2.1 Definition

Triangular Fuzzy Number can be defined by a triplet (a, b, c) on R according to the following equation

$$\text{If } x \geq a \text{ and } x \leq b \rightarrow \mu_A = \frac{(x-a)}{(b-a)}$$

$$\text{If } x \geq b \text{ and } x \leq c \rightarrow \mu_A = \frac{(c-x)}{(c-b)}$$

$$\text{Else } \mu_A = 0$$

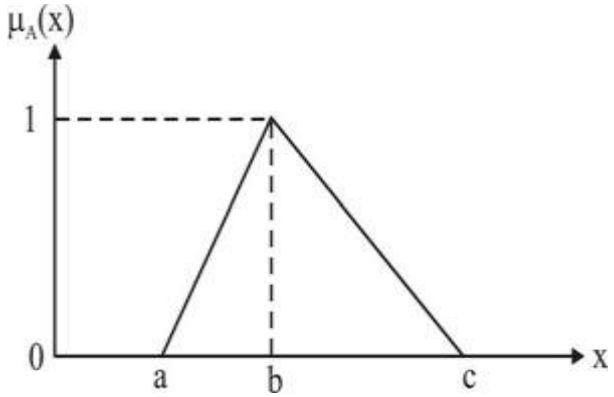


Fig. 1: Triangular fuzzy number

2.2. Definition (Trapezoidal Fuzzy Number)

A can be determined by a quadruplet (a, b, c, d) defined on R Trapezoidal Fuzzy Number with the membership function as follows:

$$If\ x \geq a\ and\ x \leq b\ \rightarrow \mu_A = \frac{x-a}{b-a}$$

$$If\ x \geq b\ and\ x \leq c\ \rightarrow \mu_A = 1$$

$$If\ x \geq \text{ʒ}\ and\ x \leq d\ \rightarrow \mu_A = \frac{d-x}{d-c}$$

$$\text{Else } \mu_A = 0$$

The fact that there are multiple points whose membership function (degree) is maximum (=1) gave rise to this shape.

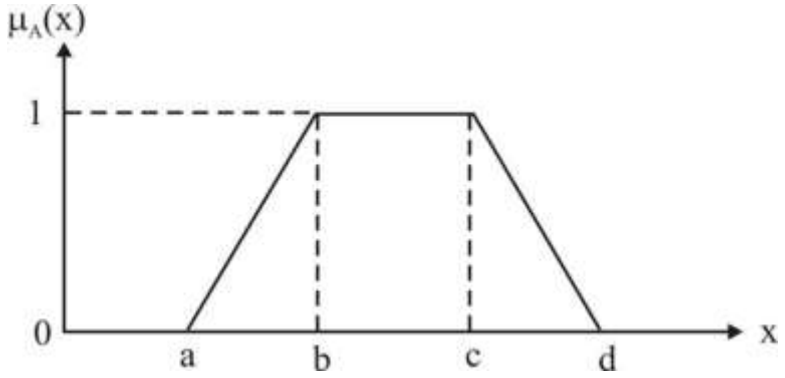


Fig.2: Trapezoidal fuzzy number

If *b* is equal to *c*, the trapezoidal fuzzy number turns into the triangular fuzzy number.

3. PROPOSED METHOD

In this part, this method is illustrated step by step as follows: the pure Type V fuzziness fuzzy graph G = (V, E), where V is the set of vertices or nodes and E is the set of edges, and G is an acyclic digraph. Let Dij be a combination of triangular and trapezoidal fuzzy numbers or simply trapezoidal fuzzy numbers for the arc length.

(Triangular Fuzzy Number). If the triangular fuzzy number *A* is defined by a triplet (*a*, *b*, *c*), the membership function will be as follows:

Input: $L_i = (a_i', b_i', c_i')$ or $L_i = (a_i', b_i', c_i', d_i')$, i = 1,2,…,n where L_i is the trapezoidal fuzzy

Output: : $D_{min} = (a, b, c)$ or $D_{min} = (a, b, c, d)$ where D_{min} the fuzzy shortest length.

Form the set S by sorting L_i in ascending orders of, b_i' and c_i'

$U = (U_1, U_2, \dots, U_n)$ where $U_i = (a_i', b_i', c_i')$, i = 1,2,…,n

If x$\geq a$ and $x \leq b \rightarrow \mu_A = \frac{(x-a)}{(b-a)}$

If x$\geq b$ and $x \leq c \rightarrow \mu_A = \frac{(c-x)}{(c-b)}$

Else $\mu_A = 0$

The greater the area of intersection between two triangles, the greater the degree of resemblance between them. Let $L_i = (a_i, b_i, c_i)$ be the length of the i[th] fuzzy path, and FSPL L_{min}= (*a*, *b*, *c*). The degree of similarity Si between L_i and L_{min} can therefore be determined as follows:

$$S_i = area(L_i \cap L_{min}) = \begin{cases} 0 & if\ L_i \cap L_{min} = \emptyset \\ \dfrac{(c-a_i)^2}{2[(c-b)+(b_i-a_i)]} & if\ L_i \cap L_{min} \neq \emptyset \end{cases}$$

It's important to remember that a, b, and c are all smaller or equal to a_i, b_i, and c_i, respectively.

A new algorithm for the fuzzy SP problem

Input: To get from one node to the next, you'll need to traverse a directed network of n nodes with fuzzy arc lengths.

Output: FSPL and SP.

Step 1: Calculate the corresponding fuzzy path lengths $L_i = (a_i, b_i, c_i)$, i = 1,2,…,m, for feasible m pathways.

Step 2: Using the FSPL heuristic technique, find the $L_{min} = (a, b, c)$

Step 3: Find the similarity degree S_i between L_{min} and L_i, i=1,2,…,m.

Step 4: Define the SP with the highest similarity degree S_i.

If $L_i = (a_i', b_i', c_i', d_i')$, i = 1,2,…,n where L_i the trapezoidal fuzzy

Step 1: Form the set *U* by sorting L_i in ascending orders of M_i' where $M_i' = \frac{(b_i'+c_i')}{2}$,

$U = (U_1, U_2, \dots, U_n)$ where $U_i = (a_i', b_i', c_i', d_i')$, i = 1,2,…,n

Step2: Set : $D_{min} = (a, b, c, d) = U_1 = (a_i', b_i', c_i', d_i')$

Step3 : Let i =2

Step 4 : Calculate (a,b,c,d)

a = min(a, a_i')

if b ≤ $a_i' \rightarrow b = b$

Else if b > $a_i' \rightarrow b = \frac{(b * b_i') - (a * a_i')}{(b + b_i') - (a + a_i')}$

And

if c ≤ $b_i' \rightarrow c = c$

Else if c > $b_i' \rightarrow c = \frac{(c * c_i') - (b * b_i')}{(c * c_i') - (b * b_i')}$

D = min (d, c_i')

Step 5: Let D_{min} = (*a*, *b*, *c*, *d*)

Step 6: And *i* = *i* + 1

Step 7: Reiterate Steps 4, 5, and 6 until *i* = *n* + 1.

Step 8: The above-mentioned procedure computes the Fuzzy Shortest Length (FSL)

Step 9: The highest resemblance Degree between FSL and L_i, for *i* = 1 to *n* using (i) Identify by SP.

4. INTRODUCTION TO GAS

Genetic algorithms are a way to apply what we know about biology and how nature solves problems to the computer science realm. Either they can be used to solve problems where the possible solutions cannot be enumerated, or it would be too costly to do so. Some examples of this would be optimal placement of multiple cameras on a map, or in this case the shortest path from point A to point B though nodes in a graph.

The problem is solved by representing each solution as a gene; each gene is made of one or more alleles; just like a DNA sequence. Two examples used in this section are optimal camera placement and a generic SPP. In the camera placement problem for example, each allele might contain a position, a cost to place there or some other information. In the SPP problem I will use as an example, each allele would be a node and there would be a cost associated with getting from the previous node in the gene to the current one; in this example the first allele in the gene would be the start, and the last allele would be the destination.

In some cases, encoding can be a big part of designing the algorithm. We will see a couple examples on how to encode an SPP later in this paper.

The basic problem is broken down into four basic steps: initialization, selection, crossover, and mutation. In some cases, a fifth step of recovery is needed to fix any genes that may be invalid or unfeasible. I will discuss all four steps next.

Initialization. The first step is to initialize the first population (generation) that you will use as a base. Most times this is done randomly or with some small guidance, but we do not want too much time to be taken for initialization. For SPP most research has shown that random initialization, without ferocious genes provide the best results.

Selection. In this part of the algorithm we will design a fitness function, this will return a numeric value for how "good" of a solution that particular gene is. This rating is used to see which of the genes of the generation will produce offspring and/or continue to the next generation.

Once you have a value for each gene, there are several ways to select parents for the next generation. Some examples of this are roulette wheel selection; where each gene is given a subsection of integers below a certain number then a random number is generated within that

range and the gene that "owns" that range continues. Another is tournament selection, where two or more genes is compared agents each other randomly and the one with the better score is used. While others exist, I have seen these ones most often.

For some algorithms, some of the top "parents" selected are also continued on to the next generation unchanged. This is done so that if the optimal solution is already found it will not be destroyed.

Crossover. In crossover, two parents are taken in and two children are returned. This is similar to the DNA crossover procedure taken from biology.

The two parent genes are examined to find a suitable point for crossover. In the example of camera placement, assuming that each allele is a camera position, all pints would be suitable. In contrast, in the shortest path, assuming each node is an allele, only pints where the same node appears in both are suitable.

Once the suitable points are found one point on each gene is selected to cross the solutions over. Below is a simple graphic of how this would work:

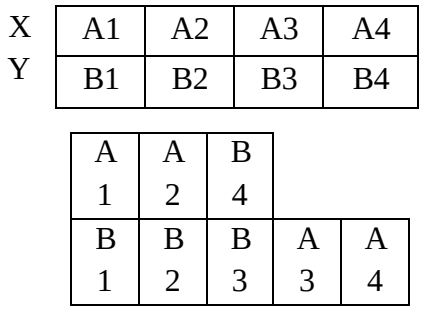


Figure 1 Crossover Example

In the figure above let X be the crossover point selected for parent A, and Y be the crossover point selected in parent B. The children produced contain some genetic information from both parents and these children will be used as part of the next generation.

In most algorithms it is okay for the genes to be of different lengths, but if that is not allowed X and Y would have to be at the same position in both genes before crossover.

Mutation. At the mutation phase, some alleles are randomly selected with a given percentage to change. Depending on the algorithm in question this may also change alleles around it. This similar to a genetic mutation in Biology where you may have a mistake in copying DNA and this results in some change in the cell produced, except here is on purpose and is enforced.

In most algorithms, the rate of mutation is very low, so this would happen depending on the size of the population, only a couple times. In the camera placement algorithm, they may just choose a new location for the camera and that would be the end of this stage. It gets more complex as the problem to be solved does the same.

Take for example the SPP algorithm explained in 2C, where each allele is a node, if you randomly change a node this may not be a path anymore, so you may want to instead choose a point on the path and randomly generate the end of the gene from that node on as in initialization. We will see in sections 3 and 4 how this problem is solved.

Recovery. This step is not always used, and in most cases is not presented in the explanation of what a GA is. Nevertheless, because so many of the SPP-GAs do have some sort of recovery function I have included it here.

What the recovery stage will do is find any malformed or incorrect genes in the population. For instance, if an SPP gene creates a loop back on itself, we would want to identify it, or if it never reaches the destination node.

Once we have identified the problem genes, we can do basically one of two things: destroy it (this includes replacement) or fix it. If we destroy the gene, again we have a couple choices. We can create a new one randomly, create a new child from the last generation, or leave the population size one smaller than the previous populations and possibly recover later.

Some "broken" genes cannot be fixed and must be removed, while others can be, for instance we could remove the loop if it exists, of complete a path that does not reach the destination. These are the decisions that would need to be made when designing the algorithm and we will see some examples of how this is handled later.

5. DESCRIPTION OF THE AHN-RAMAKRISHNA (AR) ALGORITHM

This paper was presented in 2002 and has since been used as a resource for other papers and is no longer a real candidate to replace the current mainstream SPP algorithm. With that said I still feel this algorithm provides the basis for many others and it appears to be a good venue to kick off examining how this problem would be solved in this realm.

Encoding. The encoding for this algorithm is simple, each allele is a node in the graph, so the edge used to get to any node in the gene (g) is the edge from g(i-1) to g(i); except for the first allele, because this is the source. This marked the fitness function for the gene quite simple. Because they are working with a network, the goal is to try to minimize the latency from source to destination. The fitness function is shown below:

$$f_i = \frac{1}{\sum_{j=1}^{l_i - 1} C_{g_i(j) g_i(j+1)}}$$

Figure 2 AR calculation of fittness [2]

where C_{ij} gives the latency between nodes i & j in the directed chart.

Initialization. This algorithm uses random initialization to create the first generation. Because purely random generation is not feasible for SPP the algorithm attempts to be as random as possible. They start reach gene by adding the source node. Then they randomly choose a node that has an edge from the source. Then they repeat the process from that node and so on, making sure not to add a node twice. If they get to a point where they cannot add a node without a repeat, they backtrack until there is a new node, they can add that is not one they have tired previously. They continue this until the destination is found.

Selection. Once the initial population is created, they utilize a "pairwise tournament selection without replacement" [2]. They select two genes at random, then the fittest of the two is selected as a parent. It is not put back in the pool for selection, so it cannot be a parent twice in one generation. Once the desired number of parents is selected, they go to the crossover method.

Crossover. The crossover method employed is very simple. They find all the nodes that exist in both parent genes, then they randomly pick one of these sets to be the crossover point. Then the sections are reversed, and two new children are born. Because they would only switch at a point where both genes have the same node there should still be a path from source to

destination. However, in this process they could create a loop, this will be resolved in the repair function that runs after mutation.

Mutation. Their mutation function utilizes some of the properties of the initialization function. If a gene is selected for mutation, then an allele is picked at random to be the mutation point. At this mutation point, it essentially follows the process of initialization where it randomly picks edges until it reaches the destination, but in this case, it will not regress back past the initial point of mutation.

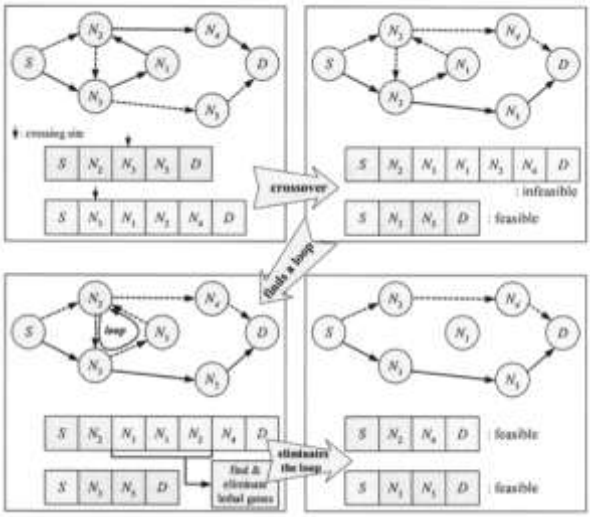


Figure 3 Diagram of AR algorithm

Repair Function. They use a very simple repair function. It traces the route that the gene encodes, if it encounters the same node twice it cuts out the path that created the loop. This can be seen in Figure 2. You can see in the third box a loop that had been created in the crossover is identified and then removed in the fourth box.

From the state at the end of Figure 2, the algorithm would use the newly created generation to produce more offspring and continue the cycle until an optimal solution is found.

6. DESCRIPTION OF THE LI-HE-GUO (LHG) ALGORITHM

This paper is presented because it uses several major differences from the AR algorithm and provides contrast even within the same field of study.

Encoding. The encoding for this algorithm is different from the AR algorithm in that it does not actually encode the path. The gene contains a waiting system for each node in the graph, thus each gene is also the same size. The genes are specifically each of size |N|. This proves to make thing mush easier in later stages of the GA.[3] The way it works is that each node has a weight relative to each other node, no two nodes should have the same weight and this is resolved in the repair function.

To get the path from this priority weight encoded gene you start from the source and then choose the node with an edge from the source that has the highest priority. Then you repeat until you reach the destination without causing a loop. If a node is encountered that has no edge from which you would not create a loop, we would create a virtual link from that node to the destination, and give it a severe penalty (high latency) so that the GA will overcome the problem and evolve past it. The whole process is described in detail in section IV of [4].

Initialization. Initialization for the given encoding is quite simple. They assign an importance value to each node in the graph, which essentially is a percentage of the edges that don't come to or from that node. The formula is given below. This value is then used along

with a random number (between 0 and 1), and a large integer constant to produce the value of that node in the current gene.

$$\forall i \in V, W(i) = \left[1 - \frac{\sum_{j} c_{i,j}}{\sum_{i}\sum_{j} c_{i,j}} \right], (i,j) \in E$$

Figure 4 LHG calculation of importance

Crossover. Simply select a number of parents from the current generation then randomly generate pairings of them for crossover. Then a random section of each parent (of identical size and position) is exchanged.

Mutation. A select number of chromosomes are selected for mutation. Once a chromosome selected, then a gene is selected given its gene weight, some probability, and a random number.

Fitness & Selection. Here you calculate the fitness of each chromosome, this is done by calculating the latency of the path, and then the fitness is the inverse of that. Select some of the chromosomes that have a better fitness compared to the others in the generation. The total number of selected chromosomes should be between .6*size & .9*size.

7. ANALYSIS OF THE ALGORITHMS

While the AR algorithm gets close to, 100% accuracy on small networks we can see as the size of the network increases their convergence percentage begins to fall. This is then counteracted with the increase of the population size. On the other hand, the LHG algorithm does not resent any results on any networks as small as those presented in the AC algorithm, they also tended to get higher convergence rates.

While the two algorithms presented here do not present results compared to each other, we can see that from the results presented they both work well, but the AC algorithm seems to do better on small networks, while the LHG algorithm does better on large networks.

8. OBSERVATIONS

In looking at many of genetic algorithms for solving the shortest path problem, I have seen that they could be a reasonable solution for use on an Ad-Hoc network, but the results are still on the same level as Djikstra. No paper I have read really presents anything that can far surpass the current standard in time complexity and results.

As an overall observation of Genetic algorithms, they are a very interesting class of solutions, but I am still not convinced that they can be used efficiently to solve any problems that are not already completed in realistic time. It seems like in most cases it makes the problem more complex than needed. I also feel that more research could be done in this area to possibly used GAs to solve some unsolved issued in computer science that people have put on the back burner while concentrating on the more "glamorous" topics. While there is some research and development being done in genetic algorithms, i think as the sciences all converge together, there is so much more I feel we can learn from their disciplines and bring to the CS community.

9. THE PROPOSED GENETIC ALGORITHM (GA)

Each candidate path in the proposed GA is represented by a binary string of length N that can be used as a chromosome. In the network topology, each chromosomal element

represents a node. As a result, each candidate solution x has N string components for a network of N nodes. At least two none zero elements must be present on each chromosome. The route in Figure [5], for example, is represented as a chromosome in Figure [6] if N = 8. The different components (operations) of the presented genetic algorithm are explained in the following subsections.

9.1. Initial Population

To be a real candidate path, the produced chromosome in the initial population must have at least two none zero elements. The steps below show how to generate the first population's pop size chromosomes:

1. Make a chromosome x at random.
2. Determine whether x is a valid candidate path, that is, if it contains at least two non-zero components.
3. To make pop size chromosomes, repeat steps (1) and (2).

9.2. The objective function

To compare the solutions and determine the best one, the cost of the candidate path is employed as an objective function. When the candidate path meets the following criteria, the cost is calculated. At least two none zero elements must be present on the chromosome. A possible path is connected to the chromosome. In other words, each node in the path is connected to at least one other.

9.3. Genetic Crossover Operation

We use a single cut point crossover in the suggested GA to generate a new offspring from two parents. If the crossover ratio (Pc=0.90) is validated, the crossover operation will be executed. The cut point is chosen at random. The crossover procedure is depicted in Figure 7.

9.4. Genetic Mutation Operation

The mutation is carried out on a bit-by-bit basis. The mutation operation will be carried out in the proposed method if the mutation ratio (Pm) is verified. The Pm in this method was chosen by trial and error to be 0.02. The point that will be modified is chosen at random. Figure 8 depicts the children produced via mutation.

Figure 8. An example of the mutation operation.

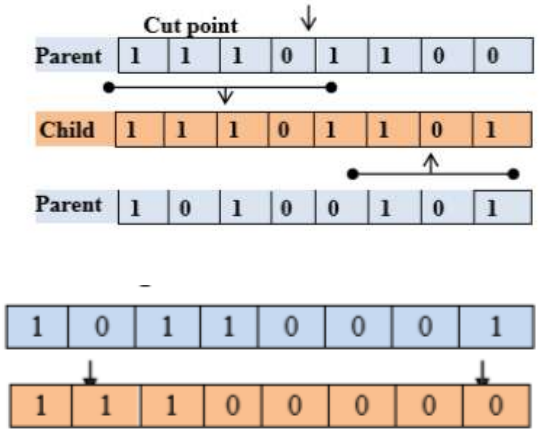


Figure 7. Example of the crossover operation.

10. THE ENTIRE ALGORITHM

The following pseudo-code demonstrates how to employ our various GA algorithm components to construct a network's minimum-cost routes tree.

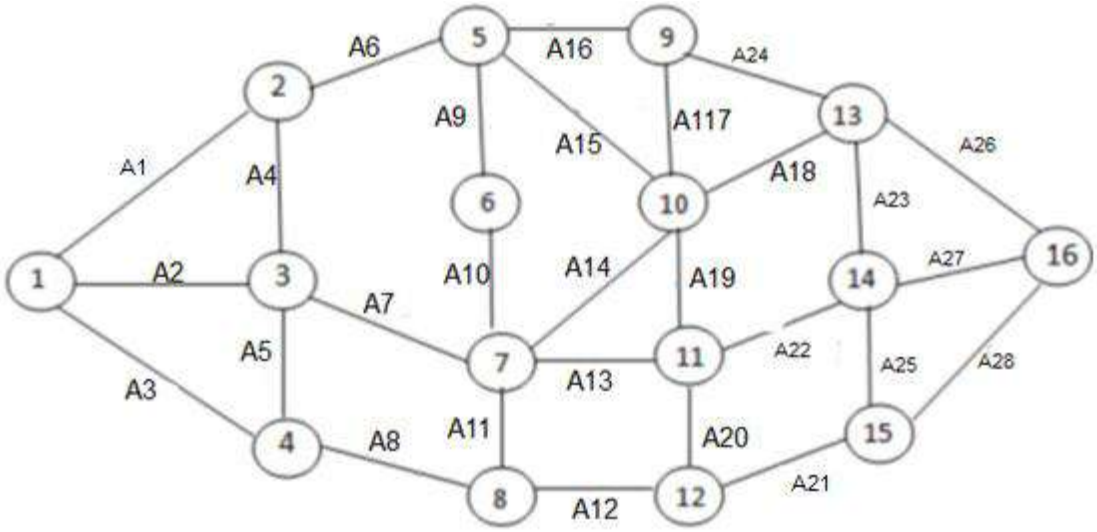
Algorithm Find minimum-cost paths tree	
Input	Set the parameters: pop_size, max_gen, Pm, Pc.
Output	Minimum-cost paths tree
1	Set j = 2, the destination node.
2	Follow the instructions in section 0 to create the starting population.
3	gen←1.
4	While (gen < = max_gen) do {
5	P ← 1
6	While (P <= pop_size) do {
7	Apply Genetic operations to obtain new population
7.1	Apply crossover according to Pc parameter (Pc >=0.90)
7.2	Apply Mutation as shown in section 4.4.
7.3	Compute the total cost of the candidate path
8	P ← P+1.
9	}
10	Set gen =gen + 1
11	if gen > max_gen then stop
12	}
13	Save the candidate path with the lowest cost for the destination j (the shortest path between the root node and the destination node j).
14	Set j = j + 1 15.
	If j <= N Goto Step 2,
	Otherwise, the process will halt and the minimum-cost routes tree will be printed.

11. NUMERICAL EXAMPLES

Solving the Shortest Path Problem By using Matlab Code for Fuzzy Optimization Algorithms

```
[n m]=size(U);
dmin=U(1,:);
for i=2:n
%-- a
a=dmin(1,1);
if U(i,1)< a
a=U(i,1);
end
%-- b
if dmin(1,2)> U(i,1)
b=((dmin(1,2)*U(i,2))-(dmin(1,1)*U(i,1)))/((dmin(1,2)+U(i,2))-
(dmin(1,1)+U(i,1)));
else
b=dmin(1,2);
end
%-- c
if dmin(1,3)> U(i,2)
c=((dmin(1,3)*U(i,3))-(dmin(1,2)*U(i,2)))/((dmin(1,3)+U(i,3))-
(dmin(1,2)+U(i,2)));
else
c=dmin(1,3);
```

```
end
%-- d
d=dmin(1,4);
if U(i,3)< d
d=U(i,3);
end
dmin=[a b c d];
end
SD=0;
for i=1:n
if U(i,1)>dmin(1,4)
SD(i)=0;
elseif U(i,2)<dmin(1,3)
SD(i)=1/2*((dmin(1,4)-U(i,1))+(dmin(1,3)-U(i,2)));
else
SD(i)=1/2*((dmin(1,4)-U(i,1))^2/((dmin(1,4)-dmin(1,3))+(U(i,2)-U(i,1))));
end
end
dmin
SD
```



The fuzzy weight of the network

Arc	Fuzzy weight	Arc	Fuzzy weight	Arc	Fuzzy weight
d12 =A1(1-2)	(10,20,30)	d59 = A16(5-9)	(6,8,10)	d1112 = A20(11-12)	(6,8,10)
d13 = A2(1-3)	(52,62,65,70)	d510 = A15(5-10)	(8,12,16)	d1114 = A22(11-14)	(6,10,12)
d14 = A3(1-4)	(30,40,50)	d78 = A11(7-8)	(2,6,8,10)	d1215 = A21(12-15)	(10,14,16,18)
d23 =A4(2-3)	(8,18,25)	d710 = A14(7-10)	(9,12,16)	d1314 = A23(13-14)	(5,6,10,12)
d34 = A5(3-4)	(5,6,7,8)	d711 = A13(7-11)	(2,8,10,14)	d1316 = A26(13-16)	(6,10,12,16)
d25 = A6(2-5)	(3,4,5)	d812 = A12(8-12)	(12,16,18,20)	d1415 = A25(14-15)	(3,6,10,12)
d37 =A7(3-7)	(10,15,20)	d910 = A17(9-10)	(6,9,12,16)	d1416 = A27(14-16)	(7,9,10,12)
d48 = A8(4-8)	(13,16,17,18)	d913 = A24(9-13)	(6,8,10,14)	d1516 = A28(15-16)	(9,10,12)
d56 = A9(5-6)	(10,14,18)	d1011 = A19(10-11)	(7,12,16,18)		
d67 = A10(6-7)	(8,10,14,18)	d1013 = A18(10,13)	(6,8,12)		

Solution:

U =

L1 = 31 40 54 75 L2 = 33 44 56 79 L3 = 37 45 62 83 L4 = 37 49 64 89 L5 = 39 49 64 87

L6 = 42 52 72 95 L7 = 43 54 72 97 L8 = 41 57 72 93 L9 = 44 56 74 99 L10 = 45 62 80 103

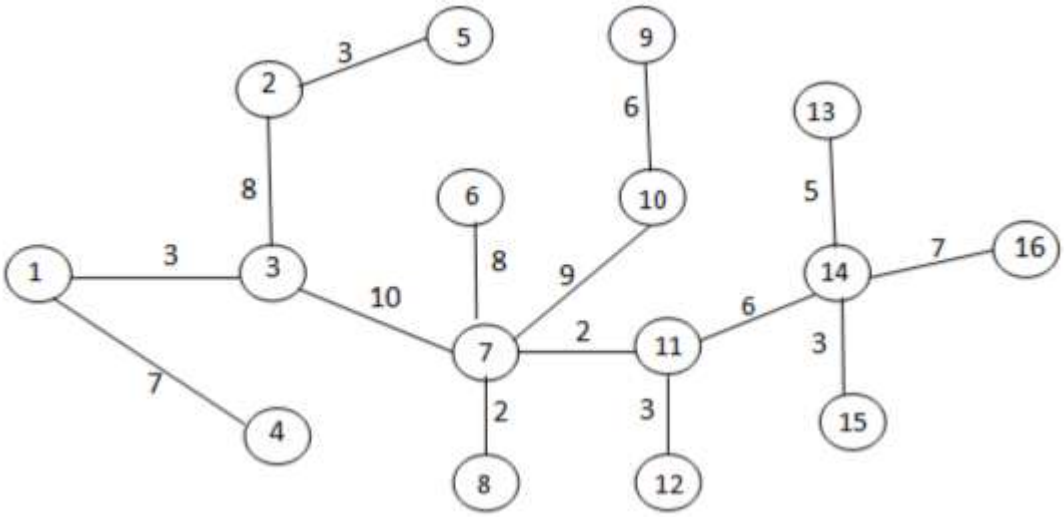
L11 = 48 61 82 109 L12 = 46 64 82 105 L13 = 46 65 82 109 L14 = 43 70 83 113 L15 = 52 68 84 115

L16 = 49 73 85 119 L17 = 53 70 86 109 L18 = 50 69 90 115 L19 = 51 72 92 121 L20 = 58 73 92 123
L21 = 48 77 93 125 L22 = 55 78 93 127 L23 = 57 75 94 119 L24 = 58 78 96 125 L25 = 55 83 97 129
L26 = 60 81 100 129 L27 = 57 86 101 133 L28 = 74 96 101 118 L29 = 63 80 102 135 L30 = 60 85 103 139
L31 = 64 84 104 131 L32= 61 89 105 135 L33 = 67 90 106 131 L34= 65 88 110 141 L35 = 77 104 110 128
L36 = 62 93 111 145 L37 = 83 107 112 134 L38 = 72 94 114 145 L39 = 69 99 115 149 L40 = 82 111 120 140
L41 = 89 112 120 142 L42 = 89 117 124 144 L43 = 91 120 128 148 L44 = 94 119 130 154 L45 = 95 123 132 150
L46 = 01 124 133 146 L47 = 96 127 138 160 L48 = 103 133 142 164
Shortest path = 1 2 5 9 13 16
dmin = 31.0000 36.8500 47.1366 56.0000
SD =
Columns 1 through 12
16.0683 13.0683 10.5683 8.6515 7.6604 5.1953 4.2541 4.5247 3.4510 2.3392 1.4636 1.8613
Columns 13 through 24
1.7945 2.3562 0.3218 0.7455 0.1740 0.6460 0.4186 0 0.8451 0.0157 0 0
Columns 25 through 36
0.0136 0 0 0 0 0 0 0 0 0 0 0
Columns 37 through 48
0 0 0 0 0 0 0 0 0 0 0 0

Solving the Shortest Path Problem By using Genetic Optimization Algorithm

Table 1: The final of the proposed algorithm (GA)

d_i	The shortest path	Ob_1 (cost)	Ob_2 (bandwidth)
2	{1,3,2}	11	12
3	{1,3}	3	12
4	{1,7,8,4}	28	10
5	{1,3,7,10,9,5}	34	10
6	{1,3,2,5,9,10,7,6}	43	10
7	{1,3,2,5,9,10,7}	35	10
8	{1,3,7,11,12,8}	30	10
9	{1,2,5,9}	21	10
10	{1,3,2,5,9,10}	26	12
11	{1,3,2,7,8,12,11}	30	10
12	{1,3,7,11,12}	18	10
13	{1,3,2,5,9,13}	28	10
14	{1,3,7,11,14}	21	12
15	{1,3,7,11,14,15}	24	12
16	{1,3,7,11,14,16}	28	12



The minimum-cost paths tree rooted at node 1

```
CM= [ 0 12  3  7  0  0  0  0  0  0  0  0  0  0  0  0
     12  0  8  0  3  0  0  0  0  0  0  0  0  0  0  0
      3  8  0  5  0  0 10  0  0  0  0  0  0  0  0  0
      7  0  5  0  0  0  0 13  0  0  0  0  0  0  0  0
      0  3  0  0  0 10  0  0  6  8  0  0  0  0  0  0
      0  0  0  0 10  0  8  0  0  0  0  0  0  0  0  0
      0  0 10  0  0  8  0  2  0  9  2  0  0  0  0  0
      0  0  0 13  0  0  2  0  0  0  0 12  0  0  0  0
      0  0  0  0  6  0  0  0  0  6  0  0  8  0  0  0
      0  0  0  0  8  0  9  0  6  0  7  0  6  0  0  0
      0  0  0  0  0  0  2  0  0  7  0  3  0  6  0  0
      0  0  0  0  0  0  0 12  0  0  3  0  0  0 10  0
      0  0  0  0  0  0  0  0  8  6  0  0  0  5  0 10
      0  0  0  0  0  0  0  0  0  0  6  0  5  0  3  7
      0  0  0  0  0  0  0  0  0  0  0 10  0  3  0  9
      0  0  0  0  0  0  0  0  0  0  0  0 10  7  9  0]
```

Cost matrix of the above network

Comparison between the previous two methods

	The used Optimization Algorithm	Objective	Purpose
First Method	Fuzzy Optimization Algorithm (FOA)	Single objective	Find the most cost-effective path between source node s and destination node d.
Second Method	Genetic Algorithm (GA)	Single objective	Find shortest paths tree (shortest path between node s and every node in network)

12.CONCLUSION

The problem of finding shortest path in network can be solved by many ways depending on our objectives. We give two different approaches to tackling this challenge in this summary. The first technique has been explored and the performance of a proposed network in terms of packet delay, throughput, and bandwidth consumption has been given utilizing Fuzzy Optimization (FO) as a routing protocol with network coding. The second method is based on GA and uses crossover and mutation operators which make it somehow more complex and doesn't always guarantee giving the right results. Also, the crossover and mutation operators are done on random basis which may lead loose the best chromosome in the new generations. There is need from adding selection operator in the algorithm. Selection operator will help in determining which solutions are to be preserved and allowed to reproduce and which ones deserve to die out. Also, it will help in focusing research in promising areas of the search space.

REFERENCES

[1] WU, A. , *Computers & Industrial Engineering* ,53(2), 277-289 , 2007 , DOI: 10.1016/j.cie.2007.06.021

[2] Ahn ,C.W, . Ramakrishna, R.S. , *IEEE Trans. on Evolutionary Computation*, 6, (6), 566-579 , 2002 , DOI: 10.1109/TEVC.2002.804323

[3] Li ,Y., He , R ., Guo, Y., *The Sixth International Symposium on Operations Research and Its Applications*. 6 (8), 380-389, 2006 .

[4] Gen , M., Cheng , R ., Wang , D., *IEEE.* , 401-406, 2007 , DOI: 10.1109/ICEC.1997.592343

[5] Gen , M., Lin , L., *GECCO'06.* , 6 , 1411-1412, 2006 , DOI: 10.1145/1143997.1144220

[6] Alshaheen , H . S. , *Journal of University of Thi-Qar* , 8 (3) , 2013 , Download from : https://www.iasj.net/iasj/download/c1836ddd9707a33c

[7] Sabri , D. , *Brunel University Research Archive(BURA)* , 2011 . Download from : https://bura.brunel.ac.uk/handle/2438/6435

[8] Dorigo , M., Maniezzo ,V. , Colorni. , A. , *IEEE Transactions on Systems, Man, and Cybernetics* - Part B, 26 , 29-41 , 1996 , DOI: 10.1109/3477.484436

[9] Dorigo , M., Bonabeau , E., Theraulaz , G., *Future Gener Comput. Syst*., 16(8) , 851-871, 2000 , DOI: 10.1016/S0167-739X(00)00042-X

[10] Hu , X. , Zhang , J . , *Mathematical Problems in Engineering*, 2013, Article ID 432686, DOI: 10.1155/2013/432686

[11] Di Caro , G. , A., Ducatelle , F. , Gambardella , L. , M. , *The Complex Coevolution of Information Technology Ecosystems* , 2008 , DOI: 10.4018/978-1-59904-627-3.ch012