Journal of Hyperstructures 11 *(1) (2022), 84-108.*
ISSN: 2322-1666 print/2251-8436 online

TOTAL FUZZY GRAPH COLORING

SMRITI SAXENA, ANTIKA THAPAR AND RICHA BANSAL

Abstract. In this paper, a hybrid genetic algorithm (HGA) is proposed for the total fuzzy graph coloring (TFGC) problem. TFGC comprises of a graph with fuzzy vertices and edges, seeks to obtain an optimal $k-$coloring of that fuzzy graph such that the degree of the sum of incompatible vertices and edges is minimized. TFGC is also converted into an equivalent binary programming problem and solved using a CPLEX solver. The performance of both methods is examined on randomly generated fuzzy graphs and computational results are reported. An application based on TFGC is also explored and solved using both proposed methods.

Key Words: Fuzzy graph, incompatibility degree, total chromatic number, binary programming and hybrid genetic algorithm.
2010 Mathematics Subject Classification: Primary: 05C15; Secondary: 05C72, 05C85, 20D60.

1. Introduction

Fuzziness is the necessity of life as it is present almost everywhere in our life. For example, in binary logic, a fan is assigned value 1 if it is open, otherwise 0. But in the case of fuzzy logic, regulator of the fan deals with fuzziness as its high speed denotes its high membership degree and its low speed denotes its low membership degree. As fuzzy sets deals with the concept of uncertainty, ambiguity and vagueness, similarly fuzzy graph theory deals with uncertain or ambiguous practical

Received: 19 June 2021, Accepted: 15 November 2021. Communicated by Ahmad Yousefian Darani;
∗Address correspondence to Antika Thapar; E-mail: antikathapar@gmail.com

problems which can be modelled as graphs. Fuzzy graph theory has numerous applications in modern science and technology especially in the fields of information theory, neural networks, expert systems, cluster analysis, medical diagnosis, control theory, etc.

The first definition of fuzzy graph was given by Kaufmann [4] in 1973, based on Zadeh's fuzzy relations [12]. But it was Rosenfeld [9] and Yeh and Bang [11] who laid the foundations for fuzzy graph theory. Generalization of basic concepts of graph theory like paths, cycles, trees, connectedness and their properties in fuzzy graph theory has been done by Rosenfeld [9].

Coloring of the graphs is the most decent, important and recreational problem in the optimization field. Many real life problems can be solved using coloring of graphs like scheduling, telecommunications, resource allocation, bioinformatics, wiring printed circuits, management sciences, etc. Coloring problem of classical graph is concerned with minimizing the number of colors k to color a graph in such a way that no two adjacent vertices receive the same color. The minimum value of k is called chromatic number of the graph and the graph is said to be $k-$ colorable.

Many researches have been done over fuzzy graph colorings. Bershtein and Bozhenuk [2] studied the coloring of fuzzy graphs and presented definitions and properties of separation degree and fuzzy chromatic set of fuzzy graphs along with the method for finding fuzzy chromatic set. Eslahchi and Onagh [3] defined the fuzzy coloring of a fuzzy graph and generalized the concepts of vertex-strength and chromatic sum of a crisp graph to fuzzy graphs. They also proved an upper (or a lower) bound for the chromatic fuzzy sum of a fuzzy graph.

Keshavarz [5] introduced a new vertex-coloring problem of a fuzzy graph with crisp vertices and fuzzy edges. He designed a hybrid local search genetic algorithm and also formulated binary programming problem for the concerned problem. He also gave an application of his proposed approach in cell site assignment problem by modelling it as a fuzzy graph coloring problem.

Munoz et al. [7] gave an application of coloring fuzzy graphs based on the successive coloring of $\alpha-$cuts of the fuzzy graph in the traffic lights problem using the concept of incompatibility and fuzzy linguistic variables.

Behzad [1] posed independently a new concept of graph coloring known as total coloring, in 1965. This is a mixed composition of both

the graph colorings i.e. vertex coloring and edge coloring defined in such a way that no two adjacent vertices and adjacent edges receive the same color. The concept of total coloring has many applications like match scheduling, network task efficiency and the famous total coloring conjecture [10] which has been verified for some special restricted cases.

Total coloring of fuzzy graphs with fuzzy set of vertices and fuzzy set of edges can be found in Lavanya and Sattanathan [6]. Poornima and Ramaswamy [8] studied total coloring of fuzzy graph and analogous to vertex and edge coloring of a fuzzy graph, they introduced the concepts of (d, f) extended $k-$ coloring where d is the dissimilarity degree defined on a scale function f and on the color set k. They also developed an algorithm for determining (d, f) total chromatic number of a fuzzy graph.

Most of the research work on fuzzy graphs has been done on the graph having crisp vertices and fuzzy edges. But, in this paper the most general but hard to interpret case, in which both the vertices and edges are fuzzy is considered.

A total fuzzy graph coloring (TFGC) problem of a graph with fuzzy vertices and fuzzy edges is proposed in this paper. The membership degree of the vertices and edges of the fuzzy graph is considered as the incompatibility degree of respective vertices and edges. If the adjacent edges have the same color, then the common vertex between them is said to be an incompatible vertex and the edge between the two adjacent vertices having the same color is said to be an incompatible edge. The total incompatibility is defined as the sum of incompatibility degree of incompatible edges and incompatible vertices. In the present paper, for the TFGC problem, vertices and edges of a fuzzy graph are to be assigned k colors such that total incompatibility degree of the fuzzy graph is minimized. TFGC problem is solved using following two methods:
1) TFGC is converted into an equivalent binary programming problem and solved using CPLEX solver,
2) A new hybrid genetic algorithm (HGA) is proposed to deal with large size fuzzy graphs.
The performance of both methods is examined on randomly generated fuzzy graphs and computational results are reported. Furthermore, as an application of TFGC problem, a political map coloring problem is also solved using both proposed techniques.

The organization of this paper is as follows: Some basic definitions related to fuzzy graph theory are given in Section 2. In Section 3,

TFGC problem is converted to a binary programming problem to find an optimal $k-$coloring. A new HGA followed by local search heuristic is introduced in Section 4 to solve TFGC problem and parameter analysis of HGA is given in Section 5. A comparison of experimental results obtained from HGA and binary programming problem using CPLEX solver is given in Section 6. An application of fuzzy graph coloring problem in political map coloring is described and solved in Section 7 with conclusion in Section 8.

2. Preliminaries: Basic Definitions

In this section, some basic concepts of fuzzy graph theory and fuzzy graph coloring are introduced with an example.

2.1. **Fuzzy set.** A fuzzy set A over a universal set X is defined by a membership function $A : X \to [0, 1]$ which maps each $x \in X$ to a real number lying in the closed interval $[0, 1]$, where $A(x)$ denotes the membership degree of $x \in X$ in A.

2.2. **Fuzzy graph.** Let V and E denote the set of vertices and edges of a graph $G(V, E)$ respectively. A fuzzy graph is denoted as $G(A, R)$, where A is a fuzzy set on V and R is a fuzzy relation on $V \times V$ defined such that $R(u, v) \leq min(A(u), A(v)), \forall u, v \in V, u \neq v$.

2.3. **Fuzzy subgraph.** A fuzzy graph $H(\tilde{V}, B, \tilde{R})$ is called a fuzzy subgraph of $G(A, R)$ induced by $\tilde{V}$, if $\tilde{V} \subseteq V$, B is a fuzzy set on $\tilde{V}$ such that $B(u) = A(u), \forall u \in \tilde{V}$ and $\tilde{R}(u, v) = R(u, v), \forall u, v \in \tilde{V}$.

2.4. $k-$**coloring of a fuzzy graph.** A $k-$coloring of a fuzzy graph $G(A, R)$ is a coloring function $C^k : V \cup E \to \{1, 2, \ldots, k\}$ with no more than k different colors. A graph is said to be $k-$colorable if it admits a $k-$coloring. Here we use k colors for total coloring of fuzzy graphs.

2.5. **Total incompatibility degree.** Let $G(A, R)$ be a fuzzy graph with $k-$coloring $C^k : V \cup E \to \{1, 2, \ldots, k\}$. We know that if two adjacent vertices have the same color, then the edge between them is said to be an incompatible edge. Similarly, if two adjacent edges have the same color, then the vertex between them is said to be an incompatible vertex. Let $E(C^k)$ and $V(C^k)$ denotes the set of incompatible edges and set of incompatible vertices respectively. For an incompatible edge $(u, v) \in E(C^k)$, its degree of incompatibility is given by $R(u, v)$ and for

an incompatible vertex $v \in V(C^k)$, its degree of incompatibility is given as $A(v)$. The total incompatibility (TI) and degree of total incompatibility (DTI) associated with C^k is defined as follows:

$$TI(C^k) = \sum_{(u,v)\in E(C^k)} R(u,v) + \sum_{v\in V(C^k)} A(v)$$

$$DTI(C^k) = \frac{\sum_{(u,v)\in E(C^k)} R(u,v) + \sum_{v\in V(C^k)} A(v)}{\sum_{(u,v)\in E} R(u,v) + \sum_{v\in V} A(v)}$$

2.6. **Total chromatic number of a fuzzy graph.** Let C^k_{min} denotes the $k-$coloring of a fuzzy graph $G(A,R)$ with minimum DTI, where $k = 1, 2, \ldots, n, n \leq |V|$. Then the total chromatic number of G is defined as $\chi(G) = \{(k, 1 - DTI(C^k_{min}))|k = 1, 2, \ldots, n.\}$ An optimal coloring set is defined as $\tau(G) = (C^1_{min}, C^2_{min}, \ldots, C^k_{min})$. The definition can be understood through an example below.

Example 1: Consider the fuzzy graph $G(A,R)$ with vertex set $V = \{u, v, w, x, y, z\}$ and edge set $E = \{uv, uz, vw, vx, wx, xy, xz, yz\}$ with their incompatibility degrees defined as membership degrees as shown in FIGURE 1. There are several ways to color the vertices and edges of the given graph.

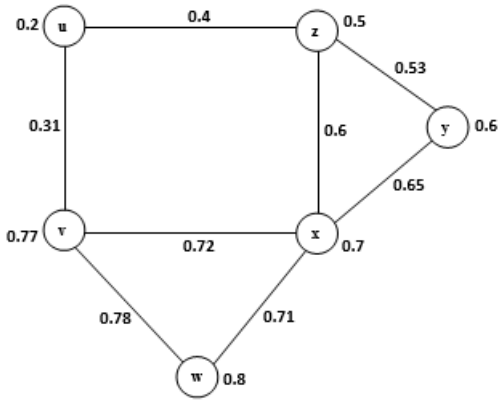

FIGURE 1. An example of a fuzzy graph

Using 1 color:

$C^1_{min} = \{(u,1),(v,1),(w,1),(x,1),(y,1),(z,1),(uv,1),(uz,1),(vw,1),$
$(vx,1),(wx,1),(xy,1),(xz,1),(yz,1)\}$
$DTI(C^1_{min}) = \frac{0.2+0.77+0.8+0.7+0.6+0.5+0.31+0.4+0.78+0.72+0.71+0.65+0.6+0.53}{0.2+0.77+0.8+0.7+0.6+0.5+0.31+0.4+0.78+0.72+0.71+0.65+0.6+0.53}$
$= \frac{8.27}{8.27} = 1$

Using 2 colors:
$C^2_{min} = \{(u,1),(v,2),(w,1),(x,1),(y,2),(z,2),(uv,1),(uz,2),(vw,1),$
$(vx,2),(wx,2),(xy,1),(xz,1),(yz,2)\}$
$DTI(C^2_{min}) = \frac{3.21}{8.27} = 0.388$

Using 3 colors:
$C^3_{min} = \{(u,1),(v,2),(w,1),(x,3),(y,1),(z,2),(uv,2),(uz,1),(vw,3),$
$(vx,1),(wx,2),(xy,1),(xz,2),(yz,3)\}$
$DTI(C^3_{min}) = \frac{0.7}{8.27} = 0.0846$

Using 4 colors:
$C^4_{min} = \{(u,1),(v,2),(w,3),(x,1),(y,3),(z,2),(uv,2),(uz,1),(vw,3),$
$(vx,1),(wx,2),(xy,3),(xz,4),(yz,2)\}$
$DTI(C^4_{min}) = \frac{0}{8.27} = 0.$

Similarly by using the 5 and 6 colors, DTI is 0. Hence, the fuzzy chromatic number and the best possible coloring set is $\chi(G) = \{(1,0),(2,$ $0.612),(3,0.9154),(4,1),(5,1),(6,1)\}$.

Total coloring of a fuzzy graph deals with assigning k colors to vertices and edges of the graph such that DTI is minimized. In the next section, a binary programming problem is formulated to solve the TFGC problem of fuzzy graph such that DTI is minimized.

3. Total fuzzy graph coloring problem as a binary programming problem

In this section, the problem of finding a $k-$coloring of $G(A,R)$ with minimum DTI is converted into a binary programming problem. For this, some binary variables w^r_i, x_{ij}, y^r_{ij} and z_{ijl} are introduced as follows:

$$w^r_i = \begin{cases} 1, & \text{if color } r \text{ is assigned to the vertex } i \\ 0, & \text{otherwise} \end{cases}$$

$$x_{ij} = \begin{cases} 1, & \text{if vertices } i \text{ and } j \text{ are assigned the same color} \\ 0, & \text{otherwise} \end{cases}$$

$$y_{ij}^r = \begin{cases} 1, & \text{if color } r \text{ is assigned to an edge } (i,j) \\ 0, & \text{otherwise} \end{cases}$$

$$z_{ijl} = \begin{cases} 1, & \text{if edges } (i,j) \text{ and } (j,l) \text{ are assigned the same color} \\ 0, & \text{otherwise} \end{cases}$$

Let $\bar{E} = \{(u,v) \in E | R(u,v) > 0\}$. Subsequently, the objective function of binary programming problem is formulated as follows:

$$min\ TI = \sum_{(i,j)\in\bar{E}} R(i,j)x_{ij} + \sum_{(i,j)\in\bar{E},(j,l)\in\bar{E}} A(j)z_{ijl} \tag{3.1}$$

$$s.t. \qquad \sum_{r=1}^{k} w_i^r = 1,\ \forall i \in V \tag{3.2}$$

$$\sum_{r=1}^{k} y_{ij}^r = 1,\ \forall (i,j) \in \bar{E} \tag{3.3}$$

$$\sum_{i\in V} w_i^r + \sum_{(i,j)\in\bar{E}} y_{ij}^r \geq 1,\ r = 1,2,\ldots,k \tag{3.4}$$

$$w_i^r + w_j^r - x_{ij} \leq 1,\ (i,j) \in \bar{E} \tag{3.5}$$

$$y_{ij}^r + y_{jl}^r - z_{ijl} \leq 1,\ (i,j),(j,l) \in \bar{E} \tag{3.6}$$

$$\begin{aligned} &x_{ij} \in \{0,1\},\ \forall (i,j) \in \bar{E} \\ &z_{ijl} \in \{0,1\},\ \forall (i,j),(j,l) \in \bar{E} \\ &w_i^r \in \{0,1\},\ \forall i \in V, \forall r = 1,2,\ldots,k \\ &y_{ij}^r \in \{0,1\},\ \forall (i,j) \in \bar{E}, \forall r = 1,2,\ldots,k. \end{aligned}$$

From equation (3.1), it is clear that the objective is to minimize the TI. Constraint (3.2) guarantees that every vertex is assigned exactly one color and similarly, constraint (3.3) guarantees that every edge is assigned exactly one color. From constraint (3.4), for each color r there exist atleast one vertex or edge assigned to it. From constraint (3.5), if

$w_i^r = w_j^r = 1$, then $x_{ij} = 1$ and if $w_i^r + w_j^r \leq 1$, then $x_{ij} = 0$ and similarly in constraint (3.6), if $y_{ij}^r = y_{jl}^r = 1$, then $z_{ijl} = 1$ and if $y_{ij}^r + y_{jl}^r \leq 1$, then $z_{ijl} = 0$. If number of colors used to color the vertices and edges of a fuzzy graph is 1, then DTI is also 1 and if number of colors are equal to the total number of vertices, then DTI is 0 as we can color vertices and edges of a fuzzy graph by colors not more than the number of vertices. Thus, we should solve the given equations (3.1) to (3.6) for the worst case of problem when number of colors lies between 2 to $n-1$ and one has to find the chromatic number of a fuzzy graph coloring problem using a sequence of optimal colorings of fuzzy graph's vertices and edges. Pseudocode for conversion of fuzzy graph coloring problem into binary programming problem is given in Algorithm 1.

Algorithm 1 TFGC problem as a binary programming problem

Given edge incompatibility degree $R(u, v)$ and vertex incompatibility degree $A(v)$ of a graph $G(A, R)$
Assign random colors from 1 to k to vertices and edges
$TI \leftarrow \sum_{(u,v)\in E} R(u, v) + \sum_{v\in V} A(v)$;
for p from 1 to population size **do**
    for i from 1 to $|V|$ **do**
        for j from 1 to $|E|$ **do**
            for r from 1 to k **do**
                if $w_i^r = 1$ or $y_{ij}^r = 1$ **then**
                $C_{min}^k(i) = r$;
                end if
            end for
        end for
    end for
    $TI_k^* \leftarrow \sum_{(u,v)\in \bar{E}} R(u, v) + \sum_{v\in V} A(v)$;
    $DTI(C_{min}^k) \longleftarrow \frac{TI_k^*}{TI}$;
end for
$\tau(G) = (C_{min}^1, C_{min}^2, \ldots, C_{min}^k)$,
$\chi(G) = \{(k, 1 - DTI(C_{min}^k)|k = 1, 2, \ldots, n)\}$.

Since classical graph coloring problem is $NP-$hard, so finding exact method to solve fuzzy graph coloring problem is also a difficult task. In this situation, heuristics with effectively less complexity are very useful

for solving the given graph coloring problem. In this paper, a new efficient meta-heuristic algorithm is designed in which genetic algorithm is combined with local search heuristic. The proposed method is described in the next section.

4. Hybrid genetic algorithm

Basic concept of simple genetic algorithm is described in this section followed by proposed HGA with local search heuristic.

4.1. **Genetic algorithm.** Genetic algorithm is used to solve optimization problems by finding the minimum or maximum value of a function. It is one of the branch of the evolutionary algorithms inspired by the biological process of reproduction, natural selection, inheritance, and crossover (also called recombination). A genetic algorithm starts with building an initial random solutions of the given problem. Then a fitness value is assigned to each solution. Now these solutions are modified by three operators namely reproduction (selection), crossover and mutation and the new set of solutions of the given problem are generated. So, again for the next generation, fitness value of each solution is evaluated and this process continues until the termination condition gets satisfied. Note that the best solution is preserved in each generation.

Reproduction operator makes only copies of good solutions but no new solution is created by it. New solutions are generated by crossover and mutation operators. In crossover operator, some part of random solution gets exchanged with some other solution and new populations are generated by taking two populations at a time. For maintaining diversity in the population, mutation operator is applied by interchanging the one element of a random solution with another one. The working of genetic algorithm is shown in Figure 2.

To improve efficiency of the genetic algorithm, local search heuristic can be embedded into it. We proposed a HGA for TFGC problem with a local search heuristic embedded in it.

4.2. **Hybrid Genetic Algorithm for TFGC with local search heuristic.** The components of proposed HGA are explained below.

Initial population: The first step of HGA is population initialization. Here the population named as *edge_vertex* is a collection of arrays of size $1 \times (|V| + |\bar{E}|)$. Each array consists of vertices from index 1 to index

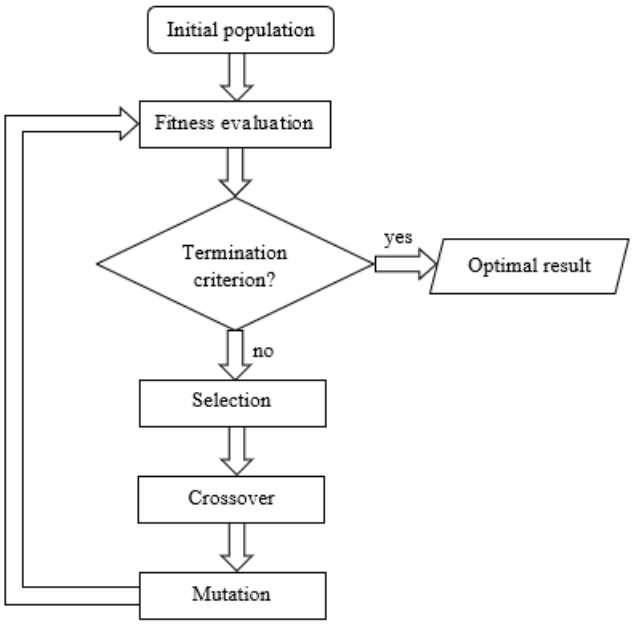

FIGURE 2. Flowchart of working of genetic algorithm

$|V|$ and edges from index $|V|+1$ to index $|V|+|\bar{E}|$ in which k colors are assigned randomly from the set $\{1, 2, \ldots, n\}$ for a given fuzzy graph. In this paper, each array is referred to as a solution. The size of the population named as *pop_size* is obtained by parameter tuning discussed in Section 5.

Fitness of a solution: Fitness of each solution is obtained by evaluating DTI as defined in Section 2.5. Between two solutions, the one having lower value of DTI is considered as a better solution as compared to the other one.

Selection: Firstly, a new population named as *selection_pop* is initialized as an empty matrix with size same as that of *edge_vertex*. Then tournament selection operator is applied on *edge_vertex* to obtain a population of good solutions. In this operator, tournaments are played between two consecutive solutions. The procedure starts by comparing the DTI of first and second solution of *edge_vertex* and the one having lower value of DTI is copied in the *selection_pop* at the first position. Again, the

next two consecutive solutions in *edge_vertex* are picked and better solution is placed at the second position of *selection_pop*. Continuing in the same way, it is to be noted that the last solution of *edge_vertex* will play tournament with solution preceding to it and with first solution of *edge_vertex* as shown in Algorithm 2. Hence, each solution in *edge_vertex* participates in tournament exactly two times and will have zero, one or two copies of it in *selection_pop*. In this way, *selection_pop* have copies of good solutions while keeping *pop_size* constant.

Algorithm 2 Tournament selection

```
1: Initialize selection_pop = edge_vertex.
2: for i from 1 to pop_size − 1 do
3:     if DTI(i) <= DTI(i + 1) then
4:         selection_pop(i) = edge_vertex(i);
5:     else
6:         selection_pop(i) = edge_vertex(i + 1);
7:     end if
8: end for
9: if DTI(1) <= DTI(pop_size) then
10:    selection_pop(pop_size) = edge_vertex(1);
11: else
12:    selection_pop(pop_size) = edge_vertex(pop_size);
13: end if
```

Crossover: After tournament selection operator, two-point crossover is applied on *selection_pop*. Firstly, *selection_pop* is copied in an empty matrix named as *crossover_pop*. Then, two random solutions are chosen from *selection_pop* (see FIGURE 3(a)). These are called parent solutions. Two different random natural numbers are generated between index 2 and index $(|V| + |\bar{E}| - 1)$. Then colors between these two indices of both parent solutions are interchanged. The new generated solutions are called *child_solutions* as shown in FIGURE 3(b). After this, fitness of both *child_solutions* is calculated and compared. If any one of them or both of *child_solutions* are found better than their respective parent solutions, then they replace their parents in *crossover_pop*. In the similar way, two parents are again selected randomly from *selection_pop* and their child replaces them in *crossover_pop*, if found better. Algorithm 3 shows the particulars of two point crossover. This process is repeated c

times; where c = crossover rate $\times$ *pop_size*. The crossover rate is fixed using parameter tuning discussed in Section 5.

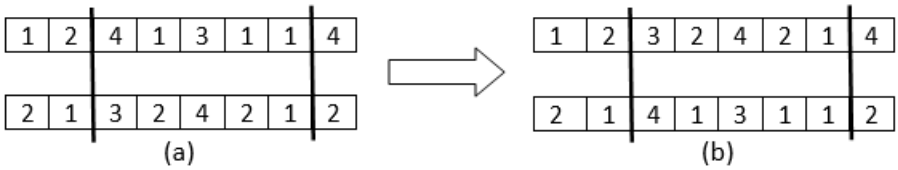

FIGURE 3. (a) Parent solutions from *selection_pop* (b) *child_solutions* after crossover

Algorithm 3 Two point crossover

1: Initialize $crossover\_pop = selection\_pop$.
2: Initialize $child\_solutions = selection\_pop$.
3: **for** i from 1 to $pop\_size \times$ crossover rate **do**
4: $child = [\,]$;
5: $p1, p2 \longleftarrow$ choose two random parent solutions from *selection_pop*
6: $n1, n2 \longleftarrow$ choose two random natural numbers between indices 2 and $(|V| + |\bar{E}| - 1)$
7: **if** $(n1 <= n2)$ **then**
8: $child(p1, n1 : n2) = crossover\_pop(p1, n1 : n2)$;
9: $crossover\_pop(p1, n1 : n2 = crossover\_pop(p2, n1 : n2)$;
10: $crossover\_pop(p2, n1 : n2) = child(p1, n1 : n2)$;
11: **end if**
12: **end for**
13: **for** $u = 1 : pop\_size$ **do**
14: **if** DTI of $crossover\_pop(u) < DTI$ of $child\_solutions(u)$ **then**
15: $child\_solutions(u) = crossover\_pop(u)$;
16: **end if**
17: **end for**

Mutation: Here, transposition mutation operator is applied on some solutions of *crossover_pop*. At first, *crossover_pop* is copied into a new empty matrix namely *mutation_pop*. Then one solution (current solution) is chosen randomly from *crossover_pop* and two integers are randomly chosen between indices 1 and $(|V|+|\bar{E}|)$ and colors at these indices

of the solution are swapped. This new solution replaces the current solution in *mutation_pop*. This process is repeated m times; where $m =$ mutation rate $\times$ *pop_size* as given in Algorithm 4. The mutation rate is fixed using parameter tuning discussed in Section 5. Mutation operator is just like a background operator in which most possible solutions can enter in the population. It is basically the reordering of the colors of solutions in the population.

Algorithm 4 Transposition mutation

1: Initialize $mutation\_pop = crossover\_pop$.
2: **for** i from 1 to $pop\_size \times$ mutation rate **do**
3: $child = [\,]$;
4: $c1 \longleftarrow$ choose a random solution from $crossover\_pop$
5: $n1, n2 \longleftarrow$ choose two random natural numbers between indices 1 and $(|V| + |\bar{E}|)$
6: $new = mutation\_pop(c1, n1)$;
7: $mutation\_pop(c1, n1) = mutation\_pop(c1, n2)$;
8: $mutation\_pop(c1, n2) = new$;
9: **end for**

Local search heuristic: The performance of genetic algorithm is enhanced when it is combined with local search heuristics for solving optimization problems. In this paper, first of all *mutation_pop* is copied into a new empty matrix named as *local_pop* and a solution is selected randomly from *mutation_pop*. Then, colors at incompatible index (i.e. incompatible vertex and incompatible edge) of this solution are replaced with different remaining colors. At each replacement, modified DTI is calculated with respect to the new color assigned at that particular index. Then, the color for which DTI is found least is assigned at that particular index. This iterative process is repeated l times, where $l =$ local search rate $\times$ *pop_size*. Here local search rate is fixed using parameter tuning as discussed in Section 5. The particulars of the presented local search operator are given in Algorithm 5.

Then *local_pop* is copied into *edge_vertex* so that *edge_vertex* is modified. This modified *edge_vertex* is then used in the next generation of genetic algorithm that undergoes for selection, crossover, mutation and local search again. Number of iterations used here are fixed using parameter tuning (see Section 5). The general structure of the presented HGA is illustrated in Algorithm 6.

Algorithm 5 Local search heuristic

```
1: Initialize local_pop = mutation_pop, DTI* = DTI (of local_pop),
   Total = 0, temp = [ ].
2: for p from 1 to pop_size × local search rate do
3:     count = 0;
4:     f1 ⟵ choose a random solution from mutation_pop
5:     while count ≤ pop_size × local search rate do
6:         Total = DTI*(f1);
7:         j ⟵ find incompatible vertices and edges of f1
8:         for r from 1 to k do                    //k = no. of colors
9:             temp(f1, j) = mutation_pop(f1, j);
10:            mutation_pop(f1, j) = r;
11:            Evaluate DTI*(f1)
12:            if DTI*(f1) < Total then
13:                Total = DTI*(f1);
14:            else
15:                mutation_pop(f1, j) = temp(f1, j);
16:                DTI*(f1) = Total;
17:            end if
18:        end for
19:        if DTI*(f1) < DTI(f1) then
20:            count = 0;
21:        else
22:            count = count + 1;
23:        end if
24:    end while
25: end for
```

5. Tuning of HGA parameters

To explore the effect of different parameters on the performance of HGA, experiments are performed to fine tune the parameters namely population size, number of iterations, crossover rate, mutation rate and local search rate. The experiments on different sized random graphs are conducted to set the parameters of HGA. The pseudocode to generate random connected graphs is given in Algorithm 7.

We generated random graphs ranging from vertex size 10 to 50 with edge density 40% of $|V|(|V|-1)/2$. For experimental purpose, five

Algorithm 6 Hybrid genetic algorithm

1: Initialize $degmin$ =0, $minDTI = inf$.
2: Generate initial population $edge\_vertex$ by assigning random colors to vertices and edges.
3: **for** *count* from 1 to w **do** // w = number of iterations
4: Evaluate DTI for $edge\_vertex$.
5: Apply tournament selection operator on $edge\_vertex$ to get $selection\_pop$.
6: Apply two-point crossover on $selection\_pop$ to obtain $crossover\_pop$.
7: Apply mutation operator on $crossover\_pop$ to obtain $mutation\_pop$.
8: Apply local search operator on $mutation\_pop$ so that $local\_pop$ is obtained.
9: Update DTI for $local\_pop$.
10: Find $degmin$. // degmin= minimum value of DTI obtained in each iteration
11: **if** $degmin < minDTI$ **then** // $minDTI$= least value of DTI
12: $minDTI = degmin$;
13: **end if**
14: $edge\_vertex = local\_pop$;
15: **end for**

instances are generated for each vertex size and HGA is performed on all 5 instances and $avgDTI$ is calculated which is the average of $minDTI$ (least value of DTI) obtained at the end for each instance. We have considered a fuzzy graph with $|V| = 10, |\bar{E}| = 18$ and $k = 4$ fixed for this entire section. The details of experiments carried out for each HGA parameter are explained below:

Algorithm 7 Pseudocode to generate random fuzzy graphs

1: $em \longleftarrow edge\_vertex$ matrix
2: **for** x from 1 to edge density d **do**
3: generate two different random integers $r1, c1$ between 1 and $|V|$
4: $em(r1, c1) \longleftarrow$ random number between 0 and 1
5: $em(c1, r1) = em(r1, c1)$;
6: **end for**

Population size: Algorithm's performance generally improves as the size of the population increases. But it affects the elapsed time of algorithm if number of iterations is constant. By fixing the parameters, mutation rate=0.05, crossover rate=0.5, local search rate= 0.4 and number of iterations=10, the effect of population size on DTI and elapsed time has been examined. HGA is executed by taking *pop_size* equal to $|V|, |V|/2, |V|/3, |V|/4$ and $|V|/5$ with different number of colors. For each fixed *pop_size*, $avgDTI$ is calculated for a particular color.

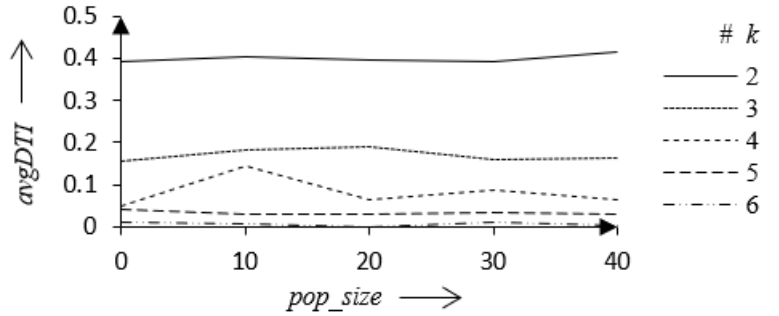

FIGURE 4. Effect of population size on $avgDTI$

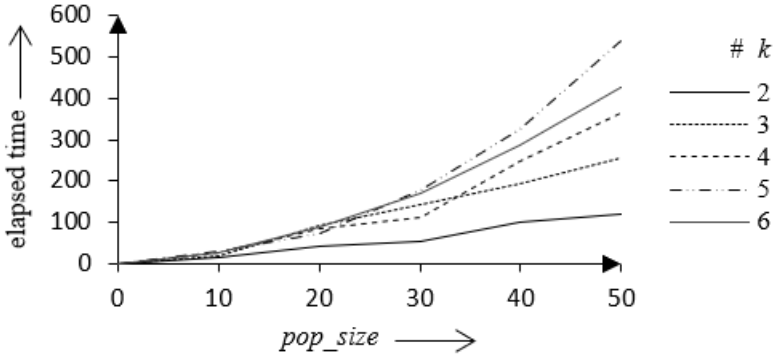

FIGURE 5. Effect of population size on elapsed time

The results shown in FIGURE 4 and FIGURE 5 signifies that large size of population have small effect on $avgDTI$ but have large impact on elapsed time. Hence rather than increasing the *pop_size*, if we increase

the number of times the algorithm run, HGA improves outcomes. So, pop_size is fixed as $|V|/2$ for further experiments.

Crossover rate: Experimental results for HGA parameter namely crossover rate are shown in TABLE 1 and TABLE 2 with different crossover rates lying between 0 and 1. For each fixed crossover rate, $avgDTI$ is calculated for a given fuzzy graph with mutation rate=0.05. TABLE 1 shows the experimental results with local search rate=0.4 and TABLE 2 shows the effect of crossover rate on $avgDTI$ and elapsed time without local search. It can be observed from both tables that HGA when executed with local search heuristic provides better value of $avgDTI$ as compared to when is executed without local search but simultaneously time behaves unfavorably. But in consideration of minimizing $avgDTI$, time does not plays any significant role. For further experiments, crossover rate is fixed as 0.5 as it gives better results in both the cases.

TABLE 1. Effect of different crossover rates on $avgDTI$ and elapsed time with local search rate

$Crossover\ Rates$	$avgDTI$	$Elapsed\ Time(sec)$
1	0.0564	4.1366
0.9	0.05054	4.3058
0.8	0.0759	4.153
0.7	0.07654	4.3348
0.6	0.7074	4.0856
0.5	0.10468	3.9698
0.4	0.22356	4.0992
0.3	0.10932	4.0728
0.2	0.1002	4.037
0.1	0.07772	4.3038
0	0.08938	3.9756

Mutation rate: TABLE 3 and TABLE 4 show the experimental results of different mutation rates lying between 0 and 1 for HGA parameter mutation rate. The experimental results with local search rate=0.4 are shown in TABLE 3 and TABLE 4 show the effect of mutation rate on $avgDTI$ without local search. For each fixed mutation rate, $avgDTI$

TABLE 2. Effect of different crossover rates on $avgDTI$ and elapsed time

$Crossover\ Rates$	$avgDTI$	$Elapsed\ Time(sec)$
1	0.2423	1.0062
0.9	0.23914	1.0262
0.8	0.21638	1.8238
0.7	0.21164	3.0764
0.6	0.21694	3.6644
0.5	0.22394	1.4532
0.4	0.2194	1.352
0.3	0.25118	1.0352
0.2	0.28364	1.0324
0.1	0.26762	1.035
0	0.24946	1.031

TABLE 3. Effect of different mutation rates on $avgDTI$ and elapsed time with local search rate

$Mutation\ Rates$	$avgDTI$	$Elapsed\ Time(sec)$
0.1	0.08368	5.792
0.09	0.06624	4.2026
0.08	0.0793	4.1448
0.07	0.07348	4.2844
0.06	0.0819	4.1064
0.05	0.06782	4.12
0.04	0.07394	4.149
0.03	0.041984	4.073
0.02	0.05156	4.2992
0.01	0.1041	4.0294
0	0.047	3.7568

is calculated for a given fuzzy graph. It can be observed from both tables that HGA when executed with local search heuristic provides better value of $avgDTI$ in comparison to without local search but simultaneously time behaves oppositely. Mutation rate is fixed as 0.05 as it gives better results in both the cases for further experiments.

TABLE 4. Effect of different mutation rates on $avgDTI$ and elapsed time

Mutation Rates	*avgDTI*	*Elapsed Time(sec)*
0.1	0.2265	2.1656
0.09	0.24082	1.0314
0.08	0.23728	1.0312
0.07	0.22782	1.022
0.06	0.20248	1.0278
0.05	0.20774	1.034
0.04	0.25286	1.022
0.03	0.26198	1.025
0.02	0.26004	1.0248
0.01	0.2013	0.8246
0	0.246	1.032

Local search rate: From previous results, the conclusion is made that the local search rate plays a vital role in finding the minimum DTI. For different values of local search rate, the FIGURE 6 shows the results of local search operator without mutation, without crossover, without crossover and mutation and with crossover and mutation.

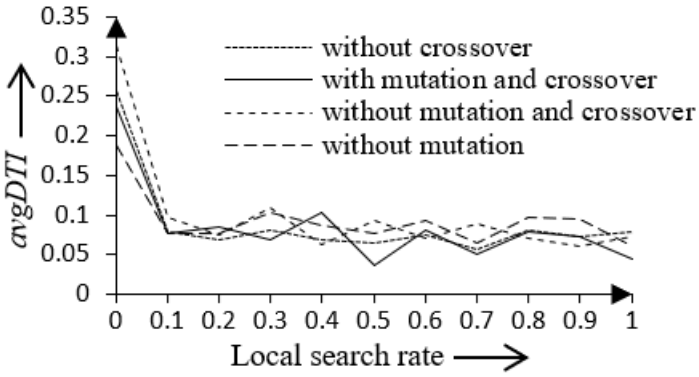

FIGURE 6. Effect of local search rate on $avgDTI$

The experimental results show that the local search operator with mutation and crossover yields good results. So, the local search rate is

fixed as 0.4 for further experiments.

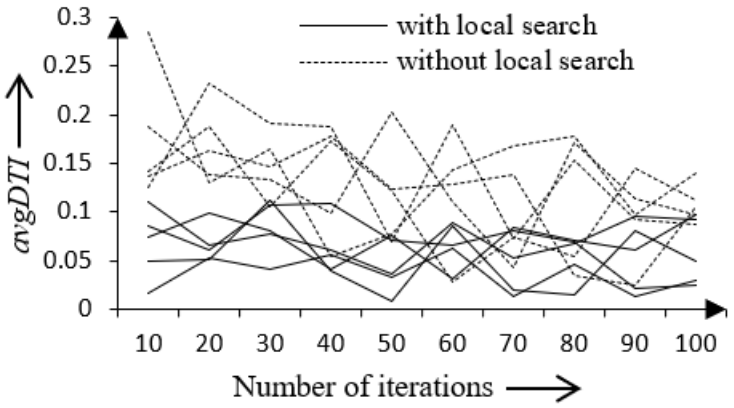

FIGURE 7. Effect of number of iterations on $avgDTI$

Number of iterations: Results are obtained by taking number of iterations from 10 to 100 while keeping all other parameters of HGA as fixed as in case of tuning of *pop_size*, crossover rate, mutation rate and local search rate. It is clear from FIGURE 7 that number of iterations with and without local search have great impact on $avgDTI$. Hence, in 10 iterations with local search heuristic, it gives better result in a reasonable time.

6. Experimental results

Finally we fixed all HGA parameters as follows: $pop\_size = |V| \setminus 2$, crossover rate = 0.504, mutation rate = 0.050, local search rate = 0.40 and number of iterations = 10. HGA is coded in MATLAB with Intel (R) Core(TM) 2 Duo CPU E6550 @2.33 GHz processor with 2 GB RAM.

TABLE 5. Fuzzy graph coloring results: 5 vertices and 4 edges

Number $of\ colors(k)$	$Optimal\ CPLEX\ value$	$Optimal\ HGA\ value$
4	0	0
3	0.00046	0.00046
2	0.044782	0.03832

TABLE 6. Fuzzy graph coloring results: 10 vertices and 18 edges

Number of colors(k)	*Optimal CPLEX value*	*Optimal HGA value*
6	0	0.0226
5	0.01715	0.02584
4	0.02987	0.04002
3	0.192352	0.18924
2	0.392675	0.37508

TABLE 7. Fuzzy graph coloring results: 15 vertices and 42 edges

Number of colors(k)	*Optimal CPLEX value*	*Optimal HGA value*
10	0	0.0275
9	0	0.0303
8	0	0.14402
7	0	0.16422
6	0.0074	0.086
5	0.02	0.1049
4	0.033	0.1588
3	0.778	0.2788
2	0.389	0.04801

TABLE 8. Fuzzy graph coloring results: 20 vertices and 76 edges

Number of colors(k)	*Optimal CPLEX value*	*Optimal HGA value*
8	0	0.0845
7	0	0.1099
6	0	0.13374
5	0.1529	0.1523
4	0.189592	0.19194
3	0.447372	0.23958
2	1.066	0.39794

To check the effectiveness of proposed HGA, the experimental results for random graphs are compared with those obtained from CPLEX solver. Performance of both CPLEX and HGA are reported in TABLE

5, TABLE 6, TABLE 7 and TABLE 8 for all random graphs. Thus, the performance of HGA is acceptable when the quantity of colors is not vast, is concluded from given 4 tables.

7. Application of fuzzy graph coloring of fuzzy graphs

Various applications of fuzzy graph coloring are presented in literature. An illustrative example of the political map coloring to confirm the applicability of this new concept of HGA is described in the following subsection.

7.1. **Political map coloring.** To show legal boundaries of continents, countries, states, regions, cities and water bodies, political maps are constructed. If particular region or area is colored on the basis of their political boundaries and political relationships, then determination of countries or particular area will be more easy. But these relationships are imprecise in real world. So, the concept of fuzzy graph coloring is very essential and can be applied here. In this problem, we have taken a country which can be separated on the basis of their political relationships or political boundaries in such a way that no two adjacent states receives the same color. Proposed algorithm gave an alternative solution and better solution for those problems where conventional deterministic methods fails to find the optimal solution.

A fuzzy graph $G(A, R)$ of given map (see FIGURE 8) is constructed in FIGURE 9, where the vertices represent states and an edge (i, j) is included in the edge set when the states shares the same boundary. Let the membership values of the vertices of the fuzzy graph are $(S1, 0.4), (S2, 0.5), (S3, 0.65), (S4, 0.8), (S5, 0.7), (S6, 0.2), (S7, 0.5)$ and $(S8, 0.2)$ and these represents the intensity of the state with respect to economy, arms, education, food capacity, technology etc. Membership values of edges of the fuzzy graph are $((S1, S2), 0.4), ((S1, S5), 0.3), ((S1, S6), 0.3), ((S1, S7), 0.9), ((S1, S8), 0.3), ((S2, S5), 1), ((S3, S4), 0.71), ((S3, S5), 0.7), ((S5, S6), 0.1), ((S5, S7), 0.2)$ and $((S7, S8), 0.6)$ denoting the strength of the political relationship between states. In this given problem, the edge (states political relationship) between two vertices (states) sharing same boundary is said to be an incompatible edge if they share the same color and if edges sharing same boundaries are colored using the same color, then common vertex between them is said to be incompatible vertex. The objective is to minimize the TI defined

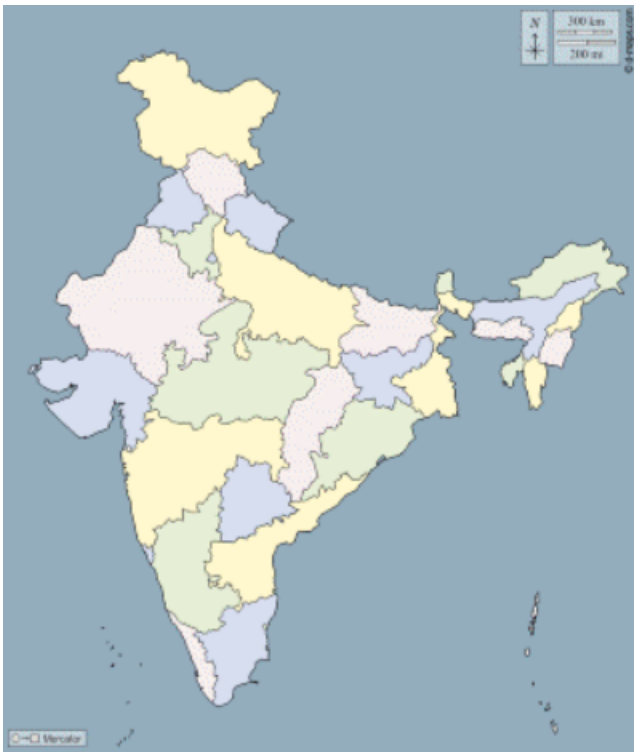

FIGURE 8. Political map

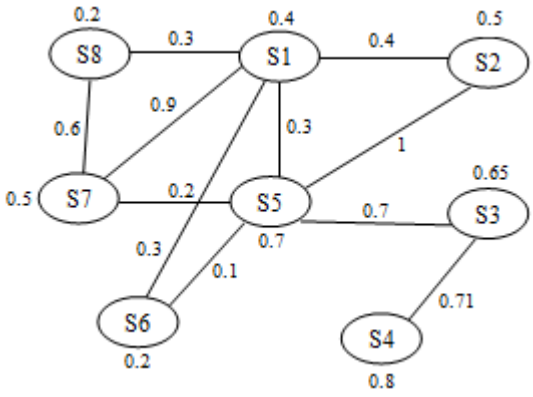

FIGURE 9. Fuzzy graph of political map

as the sum of incompatible edges and incompatible vertices of the fuzzy graph and using minimum number of colors. So, fuzzy chromatic number for political map coloring problem is to be find. Using the proposed HGA, the results for the given problem are shown in Table 9 for different number of colors. The performance of the proposed HGA is also compared with CPLEX solver.

Table 9. Political map coloring results with 8 vertices and 11 edges

$Number\ of\ colors(k)$	HGA	$CPLEX$
5	0	0
4	0.1163	0.11627
3	0.1163	0.1691
2	0.2326	0.7691
1	1	1

Thus, from Table 9 it is clear that atmost 5 colors are used for coloring of political map.

8. Conclusion

On fuzzy graphs with fuzzy vertices and fuzzy edges, the problem of TFGC is defined. Membership grades of fuzzy vertices and fuzzy edges is defined by incompatibility degrees of respective vertices and respective edges. Using this representation, DTI is defined. A binary programming problem is formulated and a HGA is proposed using the concept of minimum DTI to identify an optimal $k-$coloring.The performance of both approaches is examined on randomly generated fuzzy graphs and computational results are reported. Furthermore, as an application of this problem, a political map coloring problem is also solved using both proposed techniques.

There are some suggestions to extend or improve the proposed work.
* By constructing the new metaheuristics to solve the proposed model.
* By taking combination of crisp edges and fuzzy vertices, some enhancements may be made to the given proposed approach on fuzzy graphs.
* By calculating the total chromatic number in a different way.
* Making changes to the proposed algorithm for very large sized graphs.

References

[1] M. Behzad, *Graphs and their chromatic numbers*, Ph.D Thesis, Michigan State University, (1967).
[2] L. S. Bershtein and A. V. Bozhenuk, *Fuzzy coloring for fuzzy graphs*, The 10th IEEE International Conference on Fuzzy Systems, **3** (2001), 1101-1103.
[3] C. Eslahchi and B. N. Onagh, *Vertex-strength of fuzzy graphs*, International Journal of Mathematics and Mathematical Sciences, (2006), 43614-1.
[4] A. Kaufmann, *Introduction a la Theorie des Sous-Ensembles Flous*, Masson, Paris, (1973).
[5] E. Keshavarz, *Vertex-coloring of fuzzy graphs: A new approach*, Journal of Intelligent and Fuzzy Systems, **30** (2016), 883-893.
[6] S. Lavanya and R. Sattanathan, *Fuzzy total coloring of fuzzy graphs*, International Journal of Information Technology and Knowledge Management, **2** (2009), 37-39.
[7] Susana Munoz, M. Teresa Ortuno, Javier Ramirez and Javier Yanez, *Coloring fuzzy graphs*, Omega, **33** (2005), 211-221.
[8] B. Poornima and V. Ramaswamy, *Total coloring of a fuzzy graph*, International Journal of Computational and Applied Mathematics, **5** (2010), 11-23.
[9] A. Rosenfeld, *Fuzzy graphs*, Fuzzy sets and their applications to cognitive and decision processes, Academic press, (1975), 77-95.
[10] H. P. Yap, *Total Colorings of Graphs*, Lecture Notes in Mathematics, Springer-Verlag, **1623** (1996), Berlin.
[11] R. T. Yeh and S. Y. Bang, *Fuzzy relations, fuzzy graphs, and their applications to clustering analysis*, In Fuzzy sets and their applications to Cognitive and Decision Processes, (1975), 125-149.
[12] L. A. Zadeh, *Similarity relations and fuzzy orderings*, Information sciences, **3** (1971), 177-200.

Smriti Saxena
Department of Mathematics, Dayalbagh Educational Institute, P.O.Box 282005, Agra, INDIA
Email: smriti23saxena@gmail.com

Antika Thapar
Department of Mathematics, Dayalbagh Educational Institute, P.O.Box 282005, Agra, INDIA
Email: antikathapar@gmail.com

Richa Bansal
Department of Mathematics, Dayalbagh Educational Institute, P.O.Box 282005, Agra, INDIA
Email: richabansal2007@gmail.com