1. Introduction

Garey and Johnson **[1]** deduced Graph Coloring Problem (GCP)'s NP-completeness nature, whereas Baase et al. **[2]** concluded the approximate version to be NP-hard. Presently no deterministic algorithm is configured in producing the optimal solution of GCPs in polynomial time. However, Brelaz **[3]**, and Hertz et al. **[4]** illustrated the worthiness of heuristic approaches in devising the optimal or near-optimal solutions to these problems in their works. The idea of graph coloring originated from uniquely coloring the countries of a map with minimal color indices, i.e., generalized version of face-coloring or region-coloring of a planar graph. The regions of a planar graph are considered appropriately colored if no two contiguous regions, having a common edge between them with the same color, referred to as map-coloring, which leads to the notion of edge-coloring cardinal to the concept of vertex-coloring wherein two vertices of an edge have two different colors. Formally, a proper coloring of a graph implies the coloring of adjacent vertices with distinct colors. A graph conceived by adopting a proper coloring mechanism is perceived as a properly colored graph requiring minimum colors. A graph G with k different colors for its proper coloring termed as a k-chromatic graph, and the number k is called the chromatic number of G, shown in Figure 1.

A graph in which every vertex has been assigned a color according to a proper coloring is called a properly colored graph. A graph G that requires k different colors for its proper coloring, and no less, is called a k-chromatic graph, and the number k is called the chromatic number of G. The following figures are the examples of the proper, improper, and chromatic coloring of a graph:

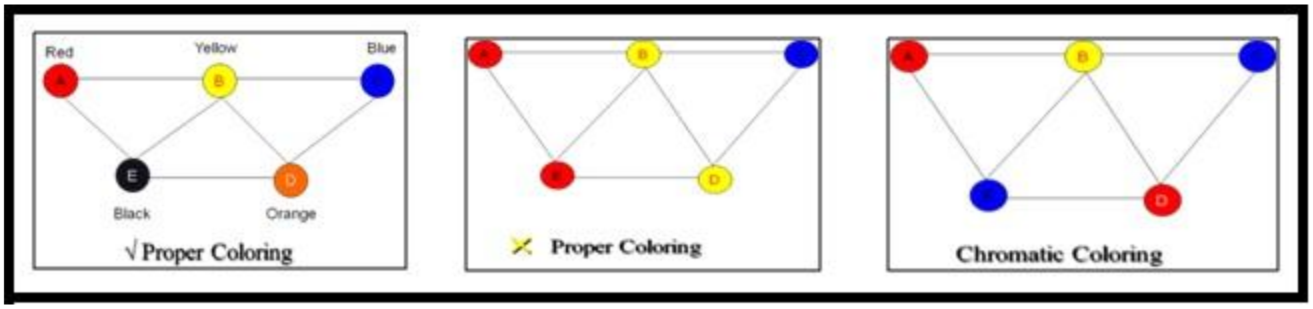

Figure 1: Proper Graph Coloring, Improper Graph coloring and Chromatic graph coloring

2. Background Study

A variety of research works in the field of computer science and mathematics have been implemented to solve the theoretical areas of graph coloring. GCP has been solved using an adjacency matrix and link list-based exact algorithm by Shukla et al. **[5]**. Some direct approaches like Branch-and-Bound **[6]**, Branch-and-Cut **[7]**, and Backtracking algorithms **[8]** have been implemented to solve GCP. Graph coloring problem has been treated as Constraint Satisfaction Problem and one possible solution has been proposed by Diaz et. al. **[9]**. With the advent of Genetic algorithms and other evolutionary approaches, the solutions of Graph Coloring Problem and its applications have been achieved in optimal or nearly sub-optimal time. A Genetic algorithm approach-based algorithm **[10]** has been proposed in to solve GCP. Dahl **[11]** applied one Artificial Neural Networks approach to solving graph coloring problem by, and more recently the algorithm has been modified by Jagota **[12]**.

3. Applications of GCP

There are some areas where efficient coloring of an undirected graph with the minimum possible number of colors directly impacts how effectively a specific target problem can be solved. Such areas include a broad spectrum of real-life issues like examination-scheduling **[13]**, register allocation **[14]**, microcode optimization **[15]**, channel routing **[16]**, printed circuit testing **[17]**, allocation of electronic bandwidth **[18]**, flexible manufacturing systems design **[19]**, and timetable scheduling **[20]**, etc.

4. Proposed Methodology

4.1. Ant Colony Optimization

Ant algorithm **[21, 22]** was first proposed as a meta-heuristic approach to solving complex optimization problems such as Travelling Salesman Problem (TSP) and Quadratic Assignment Problem (QAP). A local pheromone trail is a communication channel among the ants. Pheromone information is continuously modified during the process, which aids in searching for the optimum path by forgetting the previously found best way slowly, known as the evaporation mechanism, critical in achieving a superior solution from an inferior one through inter-communication between ants. Finally, an excellent final solution is achieved by the global cooperation of all the ants through the movement in neighboring states, based on some problem-dependent local information, pheromone trail, and ant's private information, shown in Figure 2.

```
Begin
    Initialize pheromone values
    While (not termination condition) do
        For (k= 1 to number of ants) do
            Construct a solution
            Update pheromone locally
        End for
        Update pheromone globally for best solution
    End while
End begin
```

Figure 2: General ACO Framework

4.2. Genetic Algorithm

A genetic algorithm **[23]** is a search method that is inspired by Charles Darwin's theory of natural evolution. This algorithm reflects the process of natural selection where the fittest individuals are selected for reproduction in order to produce offspring of the next generation. The process of natural selection starts with the selection of the fittest individuals from a population. They produce off spring which inherit the characteristics of the parents and will be added to the next generation. If parents have better fitness, their offspring will be better than parents and have a better chance at surviving. This process keeps on iterating and in the end, a generation with the fittest individuals will be found. The GA framework is shown in Figure 3.

```
Choose initial population
Evaluate the fitness of each individual in the population
Repeat
    Select best-ranking individuals to reproduce new generation through crossover
    and mutation (genetic operations) and give birth to offspring
    Evaluate the individual finesses of the offspring
    Replace worst ranked part of population with offspring
Until terminating condition
```

Figure 3: General GA Framework

5. Proposed Algorithms

5.1. ACO Algorithm for GCP

A sequential approach generates the initial solution in the ACO algorithm. In this approach, we have considered three sets, namely set1, set2, and set3, where all the graph vertices are placed in set3 initially. Then the first vertex is moved to set1, and all the adjacent vertices of the first vertex are moved to set2. From the remaining vertices in set3, move the first vertex again in set1, and adjacent vertices of this newly moved vertex are moved to set2 using set union operation. This process continues until set3 becomes empty. When set3 is empty, all the vertices of set1 are colored using a single color, and all the vertices in set2 are moved to set3. The process ends when all the vertices of the graph are appropriately colored. In the central execution part of the algorithm, the ant movements are based on the same approach but with some restrictions and randomness. Here, the selection of vertices to be colored is random, not sequential. Every time color assignment is made to reduce the number of colors by maintaining a list of used colors so that no conflict arises in the coloration. The coloration determines the pheromone. When all ants complete their tours, a global pheromone has been updated by taking the best coloration made by all the ants. When the execution is trapped in local minima, the solution is reconstructed by updating the global pheromone based on the worst coloration of the colony. The termination condition set for the algorithm is either maximum iteration is reached or desired chromatic number is obtained, whichever occurs earlier. The algorithm is given in Figure 4.

```
Input: A graph with n vertices.
Create an initial valid coloration by sequential approach. Set chrnum = number of distinct
colors used in the initial coloration;
Do
    Distribute m_ants to m (< n) vertices randomly;
    Construct local pheromone of m ants based on initial coloration;
    For ant = 1 to m_ants do
        ant colors all the vertices with valid coloration using the least color;
        ant updates its local pheromone table based on new coloration;
    End For
    bestcol = best coloration among the coloration made by all the ants;
    ncolor = the number of distinct colors in bestcol;
    If (ncolor < chrnum), then
        chrnum = ncolor;
        update global pheromone using bestcol;
    End if
    If chrnum not improved in successive 10 iterations, then reconstruct the solution.
While (Termination Condition not met)
Output a valid coloration.
```

Figure 4: ACO Algorithm for GCP

5.2. Genetic Algorithm for GCP

The population size is kept constant at all times and in each iteration a new chromosome is generated and added to the population of chromosomes. Here we mention new keyword WORST_DOWN_SYNDROME which means population has large number of confliction edges means it has worst fitter value. The value 50 was the least value that produced the desired results. Chromosome is form of array of size number of vertex present in a particular graph. Each Chromosome stores the color value of each vertex. The algorithm is presented in Figure 5.

```
Input: A graph with n is no of vertices
POPULATION = randomly generated chromosomes
total_colors = 1
For cycle = 1 to max_iteration or terminating condition not meet
    Parent = PARENT_SELECTION ()
    Child = CROSSOVER (Parent)
    Child = MUTATION (Child)
    Add child in place of worst fitter chromosome
Output a valid coloration

PARENTS_SELECTION ( )
Parent [1] -> Best fitter from two randomly selected chromosomes from the population;
Parent [2] -> Best fitter from two randomly selected chromosomes from the population;
Return Parent;

CROSSOVER (Parent)
Crosspoint -> random point along a chromosome from pool of chromosome;
Child -> colors up to and including cross point from Parent [1] + colors after
            cross point to the end of the chromosome from Parent [2];

MUTATION (Child)
IF WORST_DOWN_SYNDROME
    For each vertex in chromosome, do
        If vertex has the same color as an adjacent vertex, then
            vertex_color = random color from total_colors;
        Else
            For each vertex in chromosome, do
                If vertex has the same color as an adjacent vertex, then
                    adjacent_colors = all adjacent colors;
                    rest_colors = total_colors – adjacent_colors;
                    vertex_color = random color from rest_colors;
                End If
            End For
        End If
    End For
End If
```

Figure 5: Genetic Algorithm for GCP

6. Result & Discussions

In this section, we present experimental results obtained by ACO and make comparisons with GA along with various parameter settings that have been used in the algorithms. After some trial and error, we have considered number of ants (m_ants) as 10 for ACO algorithm, and the maximum iteration is set as 100000. All the algorithms have been implemented in ANSI C in Linux platform and run on a Ryzen4.2GHz machine with 16GB of RAM. All the algorithms have been tested on 56 DIMACS **[24]** benchmark graphs ten times, and the average of ten execution times is given for all the algorithms on each of the instances, respectively. Experimental result shows that out of 56 graphs tested, ACO gives optimal results for 50 graphs and GA gives optimal results for 38 graphs. Also there are 4 instances where both the algorithms fail to provide optimal results but ACO outperformed GA for such instances.

Also ACO takes less time than GA for most of the tested graph instances. Table 1 presents the experimental results.

Table1: Obtained Chromatic Number & CPU Time of ACO and GA

Sl No.	Graph instance	Vertex	Edge	Expected Result	Obtained Result (ACO)	ACO-Time (Sec)	Obtained Result (GA)	GA-Time (Sec)
1	1-Fullins_3.col	30	100	4	4	0.02	4	0.002
2	1-FullIns_4.col	93	593	5	5	0.09	5	0.0864
3	1-FullIns_5.col	282	3247	6	6	0.08	6	1.94
4	1-Insertions_4.col	67	232	4	5	24.07	5	28.066
5	1-Insertions_5.col	202	1227	6	6	0.01	6	0.056
6	1-Insertions_6.col	607	6337	7	7	0.01	7	0.0201
7	2-FullIns_3.col	52	201	5	5	0.04	5	0.073
8	2-FullIns_4.col	212	1621	6	6	0.01	6	0.001
9	2-FullIns_5.col	852	12201	7	7	43.83	10	129.04
10	2-Insertions_3.col	37	72	4	4	0.01	4	0.01
11	2-Insertions_4.col	149	541	4	5	18.11	5	23.16
12	2-Insertions_5.col	597	3936	6	6	0.01	8	164.56
13	3-FullIns_3.col	80	346	6	6	0.01	6	0.01
14	3-FullIns_4.col	405	3524	7	7	0.29	9	11.27
15	3-FullIns_5.col	2030	33751	8	9	14710.91	14	16710.98
16	3-Insertions_3.col	56	110	4	4	0.01	4	0.01
17	3-Insertions_4.col	281	1046	5	5	0.01	5	0.01
18	3-Insertions_5.col	1406	9695	6	6	0.01	9	0.08
19	4-FullIns_3.col	114	541	7	7	0.01	7	0.04
20	4-FullIns_4.col	690	6650	8	8	1.89	11	4.88
21	4-Insertions_3.col	79	156	3	4	29.30	4	47.30
22	4-Insertions_4.col	475	1795	5	5	0.01	7	3.85
23	5-FullIns_3.col	154	792	8	8	0.01	8	1.25
24	5-FullIns_4.col	1085	11395	9	9	0.68	17	2.65
25	anna.col	138	986	11	11	0.01	11	0.01
26	david.col	87	812	11	11	0.01	11	0.01
27	fpsol2.i.1.col	496	11654	65	65	0.01	80	7.04
28	fpsol2.i.2.col	451	8691	30	30	0.01	30	0.42
29	fpsol2.i.3.col	425	8688	30	30	0.01	33	0.86
30	games120.col	120	1276	9	9	0.01	9	0.01
31	homer.col	561	3258	13	13	0.01	18	46.37
32	huck.col	74	602	11	11	0.01	11	0.01
33	inithx.i.1.col	864	18707	54	54	0.12	84	7.84
34	inithx.i.2.col	645	13979	31	31	0.02	41	4.02
35	inithx.i.3.col	621	13969	31	31	0.02	35	5.16
36	jean.col	80	508	10	10	0.01	10	0.01
37	le450_15a.col	450	8168	15	20	500.56	40	5000.56
38	miles1000.col	128	6432	42	42	19.75	42	87.75

39	miles1500.col	128	10396	73	73	0.03	74	1.53
40	miles250.col	128	774	8	8	0.01	8	0.01
41	miles500.col	128	2340	20	20	0.15	20	8.15
42	miles750.col	128	1284226	31	31	16.40	34	43.40
43	mulsol.i.1.col	197	3925	49	49	0.01	49	0.02
44	mulsol.i.2.col	188	3885	31	31	0.01	31	0.08
45	mulsol.i.3.col	184	3916	31	31	0.01	31	0.01
46	mulsol.i.4.col	185	3946	31	31	0.01	31	0.01
47	mulsol.i.5.col	186	3973	31	31	0.01	31	0.04
48	myciel3.col	11	20	4	4	0.01	4	0.01
49	myciel4.col	23	71	5	5	0.01	5	0.01
50	myciel5.col	47	236	6	6	0.01	6	0.01
51	myciel6.col	95	755	7	7	0.01	7	0.01
52	myciel7.col	191	2360	8	8	0.01	8	1.01
53	queen5_5.col	25	320	5	5	0.01	5	0.01
54	queen6_6.col	36	580	7	7	0.11	7	6.11
55	queen7_7.col	49	952	7	7	0.42	7	4.42
56	queen8_8.col	64	1456	9	10	104.73	10	304.73

7. Limitations

The experimental result shows that, both ACO and GA approaches are unable to find the optimal chromatic number for some of the benchmark instances because of the complex nature of the graphs as well as because of NP-Complete nature. Also ACO algorithm has a chance to get trapped in local optima and in such cases, it is unable to produce optimal results. On the other hand, since the initial population is random in GA, the algorithm does not produce consistent time in every execution. With graphs having more nodes the execution time increases significantly.

8. Conclusion & Future Scope

The project work compares the performances of ACO – a meta-heuristic approach (MHA) and GA - one evolutionary algorithm (EA) in solving GCP which is an NP-complete problem. The experimental result shows that ACO outperforms GA for all the 56 benchmark instances but failed to generate the best-known results for the four graph instances. In the future, to overcome the limitations of the NP-complete nature of GCP, we mainly focus on designing one or more heuristic or meta-heuristic evolutionary approaches either by using some existing EAs and MHAs or through a new and robust approach to produce the optimal outturns for most of the benchmark instances in a reasonable timeframe.

References

[1] Gary, M. R., & Johnson, D. S.: “Computers and intractability: A guide to the theory of NP-Completeness”, 1979.

[2] Baase, S., & van Gelder, A.: “Computer algorithms: Introduction to design and analysis”, 2000.

[3] Brelaz, D. (1979): “New methods to color the vertices of a graph”, Communications of the ACM, 22, 251–256.

[4] Hertz, A., & de Werra, D.: “Using tabu search techniques for graph coloring”, Computing, 39, 345–351, 1987.

[5] Shukla A.N., Bharti V. and Garg M.L.: A Linked List-Based Exact Algorithm for Graph Coloring Problem, International Information and Engineering Technology Association, 33(3): pp. 189-195, 2019, DOI: 10.18280/ria.330304.

[6] Mehrotra A. and. Trick M.A.: “A column generation approach for graph coloring”, Informs Journal on Computing, 8(4): pp. 344–354, 1996, 10.1287/ijoc.8.4.344.

[7] Diaz I.M and Zabala P.: “A branch-and-cut algorithm for graph coloring”, Discrete Applied Mathematics, 154(5): pp. 826–847, 2006, DOI: 10.1016/j.dam.2005.05.022.

[8] Masson R.: “On the Analysis of Backtrack Procedures for the Coloring of Random Graphs”, Lecture Notes in Physics, 650: pp. 235–254, Springer, Berlin, Heidelberg, 2004.

[9] Diaz, Isabel Mendez, and Zabala, Paula: “A Generalization of the Graph Coloring Problem”, Departament of Computacion, University of Buenes Aires, 1999.

[10] Ray B., Pal A.J., Bhattacharyya D. and Kim T.H.: An Efficient GA with Multipoint Guided Mutation for Graph Coloring Problems, International Journal of Signal Processing, Image Processing and Pattern Recognition, 3(2): pp. 51-58, 2010.

[11] Dahl E.D.: “Neural Networks algorithms for an NP-complete problem: Map and graph coloring”, Proc. First International Conference on Neural Networks, 3: pp. 113-120, 1987.

[12] Jagota A.: “An adaptive, multiple restarts neural network algorithm for graph coloring”, European Journal of Operational Research, 93(2): pp. 257-270, 1996, DOI: 10.1016/0377-2217(96)00043-4.

[13] Leighton F.T.: “A graph coloring algorithm for large scheduling problems”, Journal of Research of the National Bureau of Standards, 84(6): pp. 489-506, 1979, DOI: 10.6028/jres.084.024.

[14] Chow F.C. and Hennessy J.L.: “Register allocation by priority-based coloring”, Proceedings of the 1984 Sigplan Symposium on Compiler construction, pp. 222-232, ACM, 1984, DOI:10.1145/502949.502896.

[15] Micheli G.D: “Synthesis and Optimization of Digital Circuits”, McGraw-Hill, 1994, DOI:10.5860/choice.32-0950.

[16] Sarma S.S., Mondal R and. Seth A: “Some sequential graph coloring algorithms for restricted channel routing”, International Journal of Electronics, 77(1): pp. 81-93, 1985.

[17] Garey M. R, Johnson D.S. and. So H.C.: “An application of graph coloring to printed circuit testing”, IEEE Transactions on circuits and systems, 23(10): pp. 591-599, 1976, DOI: 10.1109/sfcs.1975.3.

[18] Gamst A.: “Some lower bounds for class of frequency assignment problems”, IEEE Transactions on Vehicular Technology, 35(1): pp. 8-14, 1986, DOI: 10.1109/tvt.1986.24063.

[19] Stecke K.E.: “Design, planning, scheduling and control problems of flexible manufacturing”, Annals of Operations Research, 3(1): pp. 1-12, 1985.

[20] Haraty R.A, Assi M. and Halawi B.: “Genetic Algorithm Analysis using the Graph Coloring Method for Solving the University Timetable Problem”, Science Direct, Procedia Computer Science, 126: pp. 899-906, 2018, DOI: 10.1016/j.procs.2018.08.024

[21] Dorigo M., Maniezzo V. and Colorni A.: “Ant system: Optimization by a colony of operating agents”, IEEE Transactions on Systems, Man and Cybernetics-Part B (Cybernetics), 26(1): pp. 29-41, 1996, DOI: 10.1109/3477.484436.

[22] Dorigo M., Dai Caro G. and Gambardella L.M.: “Ant Algorithms for Discrete Optimization, Artificial Life”, 5(2): pp. 137-172, 1999, DOI: 10.1162/10645469956872

[23] Holland J.H.: “Adaptation in Natural and Artificial Systems”, University of Michigan Press, 1975.

[24] http://mat.gsia.cmu.edu/COLOR04/

APPENDIX:

Source Code (ACO):

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
#include<string.h>
#include<math.h>
#define size 2500
#define no_ants 10
#define max_itr 10000
int graph[size][size]={{0}};
int fset[size]={0},sset[size]={0},tset[size]={0};
int color[size]={0};
int tempcolor[size]={0};
int pher[size][size]={{0}};
int position[size];
int rand_array[size];
int antcolorcount[size];
int visited[size]={0};
int usedcolor[size]={0};
void initialcoloration(int);  //Generates Initial valid coloration of vertices
void generateantposition(int,int);  //Randomly place the ants in different nodes
int color_count(int); //Count the number colors used in coloration
int countantcolor(int,int);  //Count the number of color made by each ant
int findmin(int[],int);  //Find minimum value in a list
int findmax(int[],int);  //Find maximum value in a list
int min(int,int); //Find minimum of two numbers
void updatelocalpher(int);  //Generates coloration by each ant
void updateglobalpher(int,int);  //Updates the global coloration of the colony
int member(int[],int);
void antmove(int);
int node,edges,mincol,row,col;
int tempmin=100000,chrnum,counter=0;
int main ()
{
        int i,j,k;
        FILE *fs;
        int ch,itr=0;
        char faltu;
        system("clear");
        printf("Coloration by Ant Colony Optimization...\n");
        if((fs=fopen("queen6_6.col","r"))==NULL)
        {
                printf("\nCan't open the file.\n");
                exit(1);
        }
        while(!feof(fs))
        {
                fscanf(fs,"%d %d %d\n",&mincol,&node,&edges);
```

```
    for(j=1;j<=edges;j++)
    {
                fscanf(fs,"%c %d %d\n",&faltu,&row,&col);
            graph[row][col] = 1;
            graph[col][row] = 1;
    }
}
fclose(fs);
srand((unsigned)time(NULL));
initialcoloration(node);
printf("\nInitial Coloration:");
for(i=1;i<=node;i++)
        printf(" %d ",color[i]);
chrnum=color_count(node);
printf("\n\nInitial Chromatic Number: %d",chrnum);
getchar();
for(i=1;i<=node;i++)
{
        fset[i]=0;
        sset[i]=0;
        tset[i]=0;
}
while(chrnum>mincol)
{
        itr++;
        printf("\n\nIteration: %d",itr);
        //getchar();
        generateantposition(node,no_ants);
        /*for(i=1;i<=no_ants;i++)
                printf("\nAnt[%d]: %d",i,position[i]);
        getchar();*/
        updatelocalpher(no_ants);
        /*printf("\nLocal Pheromone:\n");
        for(i=1;i<=no_ants;i++)
        {
                for(j=1;j<=node;j++)
                {
                        printf(" %d ",pher[i][j]);
                }
                printf("\n\n");
        }
        getchar();*/
        antmove(no_ants);
        printf("\t\tChromatic Number:%d",chrnum);
        /*if(itr%10000==0)
        {
                printf("\nChromatic Number after %d Iteration: %d",itr,min(chrnum,tempmin));
                getchar();
        }*/
        if(itr==max_itr)
                break;
}
```

```
	printf("\n\nThe Final Coloration is:\n\n");
	if(tempmin>=chrnum)
	{
		for(i=1;i<=node;i++)
			printf(" %d ",color[i]);
		printf("\n\nChromatic Number: %d\n\n",color_count(node));
	}
	else
	{
		for(i=1;i<=node;i++)
			printf(" %d ",tempcolor[i]);
		printf("\n\nChromatic Number: %d\n\n",tempmin);
	}
	printf("Number of Iterations: %d\n\n",itr);
	return 0;
}
//function initialcoloration..it generates initial vertex color sequentially
void initialcoloration(int n)
{
	int i,j,k;
	int pos,ncolor=1,chkcolor=0;
	for(i=1;i<=n;i++)
		tset[i]=i;
	do
	{
		/*printf("\nfset:\t");
		for(i=1;i<=n;i++)
			printf(" %d ",fset[i]);
		getchar();
		printf("\nsset:\t");
		for(i=1;i<=n;i++)
			printf(" %d ",sset[i]);
		getchar();
		printf("\ntset:\t");
		for(i=1;i<=n;i++)
			printf(" %d ",tset[i]);
		getchar();*/
		j=1;
		pos=1;
		for(i=1;i<=n;i++)
		{
			if(tset[i]>0)
			{
				fset[pos]=tset[i];
				tset[i]=-9999;
				for(k=1;k<=n;k++)
				{
					if(tset[k]>0 && graph[fset[pos]][tset[k]]==1)
					{
						if(!member(sset,tset[k]))
						{
							sset[j++]=tset[k];
```

```
                                        tset[k]=-9999;
                                    }
                                }
                            }
                            pos++;
                        }
                    }
                    for(i=1;i<=n;i++)
                    {
                        if(fset[i]!=0)
                        {
                            color[fset[i]]=ncolor;
                            fset[i]=0;
                        }
                    }
                    for(i=1;i<=n;i++)
                    {
                        if(sset[i]!=0)
                        {
                            tset[i]=sset[i];
                            sset[i]=0;
                        }
                    }
                    ncolor++;
                    /*printf("\ncolor:\n");
                    for(i=1;i<=n;i++)
                        printf(" %d ",color[i]);
                    getchar();*/
        }while(member(color,chkcolor)==1);
}//End of initialcoloration
//function returns 1 or 0 depending on whether x belongs to set[] or not
int member(int set[],int x)
{
        int i;
   for(i=1;i<=node;i++)
        if(x==set[i])
                return 1;
   return 0;
}//End of member
//Function generateantposition..this function place ants in different vertices
void generateantposition(int n,int a)
{
        int i,rnd,j=1;
        for(i=1;i<=n;i++)
                rand_array[i]=0;
        while(j<=a)
        {
                rnd=rand()%n+1;
                if(rand_array[rnd]==0)
                {
                        position[j++]=rnd;
                        rand_array[rnd]=1;
```

```
		}
	}
}//End of generateantposition()
//Function updatelocalpher()..this function makes coloration by each ant
void updatelocalpher(int n)
{
	int i,j;
	for(i=1;i<=n;i++)
	{
		for(j=1;j<=node;j++)
		{
			pher[i][j]=color[j];
		}
	}
}//end of updatelocalpher
//Function color_count..this function counts number of colors in a coloration
int color_count(int n)
{
	int i,j;
	int temp_color[size];
	int count=0;
	for(i=1;i<=n;i++)
	{
		temp_color[i]=color[i];
	}
	for(i=1;i<n;i++)
	{
		for(j=i+1;j<=n;j++)
		{
			if(temp_color[i]==temp_color[j])
			temp_color[j]=0;
		}
	}
	for(i=1;i<=n;i++)
		if(temp_color[i]!=0)
			count++;
	return count;
}// end of color_count()
//Function antcolorcount..this function counts number of colors made by each ant locally
int countantcolor(int r,int n)
{
	int i,j;
	int temp_color[size];
	int count=0;
	for(i=1;i<=n;i++)
	{
		temp_color[i]=pher[r][i];
	}
	for(i=1;i<n;i++)
	{
		for(j=i+1;j<=n;j++)
		{
```

```
                        if(temp_color[i]==temp_color[j])
                        temp_color[j]=0;
                }
        }
        for(i=1;i<=n;i++)
                if(temp_color[i]!=0)
                        count++;
        return count;
}// end of antcolorcount()
//Function findmin().. this function returns the minimum value in a list
int findmin(int arr[],int n)
{
        int min,i,pos=1;
        min=arr[pos];
        for(i=2;i<=n;i++)
        {
                if(arr[i]<min)
                {
                        pos=i;
                        min=arr[i];
                }
        }
        return pos;
}//end of findmin
//Function findmax().. this function returns the maximum value in a list
int findmax(int arr[],int n)
{
        int max,i,pos=1;
        max=arr[pos];
        for(i=2;i<=n;i++)
        {
                if(arr[i]>max)
                {
                        pos=i;
                        max=arr[i];
                }
        }
        return pos;
}//end of findmin
//Function min().. this function returns the minimum of two numbers
int min(int x,int y)
{
        if(x<y)
                return x;
        else
                return y;
}//end of min
//Function updateglobalpher()..this function updates the global coloration of the colony
void updateglobalpher(int k,int n)
{
        int i;
        for(i=1;i<=n;i++)
```

```
                color[i]=pher[k][i];
}//end of updateglobalpher
void antmove(int x)
{
        int i,j,k,mvmnt,itrn,index;
        int pos,state,chk,complete;
        int min,max,val,clr,rnd,cl=1,choice;
        for(mvmnt=1;mvmnt<=x;mvmnt++)
        {
                //printf("\nMovement of Ant [%d]:",mvmnt);
                //printf("*************************************");
                //getchar();
                for(i=1;i<=node;i++)
                        tset[i]=i;
                j=1;
                pos=1;
                itrn=1;
                do
                {
                        if(itrn==1)
                        {
                                fset[pos]=position[mvmnt];
                                tset[position[mvmnt]]=-9999;
                        }
                        else
                        {
                                do
                                {
                                        rnd=rand()%node+1;
                                        if(tset[rnd]<=0)
                                                chk=1;
                                        else
                                                chk=0;
                                }while(chk==1);
                                fset[pos]=tset[rnd];
                                tset[rnd]=-9999;
                        }
                        do
                        {
                                for(k=1;k<=node;k++)
                                {
                                        if(tset[k]>0 && graph[fset[pos]][tset[k]]==1)
                                        {
                                                if(visited[tset[k]]==0 && !member(sset,tset[k]))
                                                {
                                                        sset[j++]=tset[k];
                                                        tset[k]=-9999;
                                                }
                                        }
                                }
                                /*printf("\nfset:");
                                for(i=1;i<=node;i++)
                                        printf(" %d ",fset[i]);
                                getchar();
                                printf("\nsset:");
                                for(i=1;i<=node;i++)
                                        printf(" %d ",sset[i]);
                                getchar();
                                printf("\ntset:");
                                for(i=1;i<=node;i++)
                                        printf(" %d ",tset[i]);
                                getchar();*/
                                for(i=1;i<=node;i++)
                                {
                                        if(tset[i]>0)
                                        {
                                                state=1;
                                                break;
                                        }
                                        else
                                                state=0;
                                }
                                if(state==1)
                                {
                                        do
                                        {
                                                index=rand()%node+1;
                                                if(tset[index]<=0)
                                                    chk=1;
                                                else
                                                    chk=0;
                                        }while(chk==1);
                                        pos++;
                                        //printf("\nindex= %d",index);
                                        //getchar();
                                        fset[pos]=tset[index];
                                        tset[index]=-9999;
                                        //printf("\nnext node= %d",fset[pos]);
                                        //getchar();
                                }
                        }while(state==1);
                        for(i=1;i<=node;i++)
                        {
                                if(fset[i]!=0 && !member(usedcolor,color[fset[i]]))
                                {
                                        clr=color[fset[i]];
                                        usedcolor[cl++]=clr;
                                        choice=0;
                                        break;
                                }
                                else
                                        choice=1;
                        }
                        if(choice==1)
```

```
{
        do
        {
                clr=rand()%node+1;
                if(member(usedcolor,clr)==1)
                        chk=1;
                else
                {
                        usedcolor[cl++]=clr;
                        choice=0;
                        chk=0;
                }
        }while(chk==1);
}
//printf("\nclr= %d",clr);
//getchar();
for(i=1;i<=node;i++)
{
        if(fset[i]!=0)
        {
                pher[mvmnt][fset[i]]=clr;
                visited[fset[i]]=1;
                fset[i]=0;
        }
}
/*printf("\nPheromone of Ant[%d]:",mvmnt);
for(i=1;i<=node;i++)
        printf(" %d ",pher[mvmnt][i]);
getchar();
printf("\nvisited list:");
for(i=1;i<=node;i++)
        printf(" %d ",visited[i]);
getchar();
printf("\nusedcolor list:");
for(i=1;i<=node;i++)
        printf(" %d ",usedcolor[i]);
getchar();*/
for(i=1;i<=node;i++)
{
        if(sset[i]!=0)
        {
                tset[i]=sset[i];
                sset[i]=0;
        }
}
/*printf("\nfset:");
for(i=1;i<=node;i++)
        printf(" %d ",fset[i]);
getchar();
printf("\nsset:");
for(i=1;i<=node;i++)
        printf(" %d ",sset[i]);
```

```
			getchar();
			printf("\ntset:");
			for(i=1;i<=node;i++)
				printf(" %d ",tset[i]);
			getchar();*/
			for(i=1;i<=node;i++)
			{
				if(visited[i]==1)
					complete=1;
				else
				{
					complete=0;
					break;
				}
			}
			itrn++;
			pos=1;
			j=1;
		}while(complete==0);
		/*printf("\nColoration of Ant[%d]:",mvmnt);
		for(i=1;i<=node;i++)
			printf(" %d ",pher[mvmnt][i]);
		getchar();*/
		antcolorcount[mvmnt]=countantcolor(mvmnt,node);
		for(i=1;i<=node;i++)
		{
			fset[i]=0;
			sset[i]=0;
			visited[i]=0;
			usedcolor[i]=0;
		}
		cl=1;
	}
	/*printf("\nAnt Coloration:\n");
	for(i=1;i<=no_ants;i++)
	{
		for(j=1;j<=node;j++)
		{
			printf(" %d ",pher[i][j]);
		}
		printf("\n");
	}*/
	//getchar();
	min=findmin(antcolorcount,no_ants);
	max=findmax(antcolorcount,no_ants);
	if(countantcolor(min,node)<chrnum)
	{
		//printf("\t\tCh. Num Changed");
		updateglobalpher(min,node);
		chrnum=countantcolor(min,node);
		counter=0;
		//getchar();
	}
	else
	{
		if(counter==10)
		{
			//printf("\t\tWorst Ch. Num");
			if(chrnum<tempmin)
			{
				tempmin=chrnum;
				//printf("\t\tempmin = %d",tempmin);
				//getchar();
				for(i=1;i<=node;i++)
					tempcolor[i]=pher[min][i];
			}
			updateglobalpher(max,node);
			chrnum=countantcolor(max,node);
			counter=0;
			//getchar();
		}
		counter++;
	}
	/*printf("\nGlobal Pheromone:");
	for(i=1;i<=node;i++)
		printf(" %d ",color[i]);
	getchar();*/
}
```

Source Code (GA):

```
# include <stdio.h>
# include <string.h>
# include <stdlib.h>
# include <time.h>
# include <math.h>
#define size 2500
int graph[size][size]={{0}};

//RANDOM POPULATION GENERATOR
void randPopulation(int pop_num, int nodes, int (*population)[nodes],int max){
    int i,j;
    for (i = 0; i < pop_num; i++)
    {
        for(j=0; j<nodes ;j++)
        {
            population[i][j] = (rand() %(max)+1);
        }
    }
}

//SELECTION WITH ELITISM
int selection_elitism(int selec_num, int pop_num, int *selected, int *fitness)
{
    int currfit=0;
    int i,j=0;
    for(i=0;i<selec_num;i++)
        selected[i] = 0;

    while(1){
        for(i=0;i<pop_num;i++){
            if(fitness[i]==currfit){
                selected[j]=i;
                j++;
                if(j==selec_num)
                    break;
            }
        }

        if(j==selec_num)
            break;
        currfit++;
    }
    return 0;
}

//SELECTION RANDOMLY
int selection_random(int selec_num, int pop_num, int *selected, int *fitness){
    int i,x,j, count=0;;
    while(count!=selec_num)
    {
        x=(rand()% pop_num );
        int f=0;
        for(int i=0;i<count+1;i++) //linear search
        {
            if(selected[i]==x)
            {
                f=1;
                break;
            }
        }
        if(f==1)
            continue;
        else
        {
            selected[count]=x;
            count++;
        }
    }
}

//PREVIOUSLY USED FITNESS FUNCTION
void fitness_func(int pop_num, int nodes, int (*population)[nodes], int(*arr)[nodes], int *fit){
    int i,j;
    for(i=0;i<pop_num;i++){
        fit[i]=0;
        for(j=0;j<nodes-1;j++){
            for(int k=j+1;k<nodes;k++){
                if(population[i][j]==population[i][k]){
                    if(arr[j][k]==1)
                        fit[i]++;
                }
            }
        }
    }
}

//CROSSOVER
void crossover(int selec_num, int pop_num, int *selected, int nodes, int (*population)[nodes], int
(*newPop)[nodes])
{
    float pc=0.7;
    int count=0;
    while(count<pop_num)
    {
        int parent1 = selected[(int)rand()%selec_num];
        int parent2 = selected[(int)rand()%selec_num];
        while(parent1==parent2)
        {
            parent2=rand()%selec_num;
        }
        float x=(float)rand()/(float)RAND_MAX;
        if(x>=pc)
        {
            int y=(rand()%(nodes-1))+1;
            for(int i=0;i<y;i++)
            {
                newPop[count][i]=population[parent1][i];
                newPop[count+1][i]=population[parent2][i];
            }
            for(int i=y;i<nodes;i++)
            {
                newPop[count][i]=population[parent2][i];
                newPop[count+1][i]=population[parent1][i];
            }
            count+=2;
        }
    }
}

//MUTATION
void mutation(int pop_num, int nodes, int (*newPop)[nodes])
{
    float pm=0.9;
    int count=0;
    while(count<pop_num)
    {
        float x=(float)rand()/(float)RAND_MAX;
        if(x>=pm)
        {
            int mutpos=rand()%nodes;
            int change = rand()%nodes+1;
            while(change==newPop[count][mutpos]){
                change=rand()%nodes+1;
            }
            newPop[count][mutpos]=change;
            count++;
        }
        else
            count++;
    }
}

//MUTATION with more probability
void mutation2(int pop_num, int nodes, int (*newPop)[nodes])
{
    float pm=0.3;
    int count=0;

    while(count<pop_num)
    {
        float x=(float)rand()/(float)RAND_MAX;
        if(x>=pm)
        {
            int mutpos=rand()%nodes;
            int change = rand()%nodes+1;
            while(change==newPop[count][mutpos]){
                change=rand()%nodes+1;
            }
            newPop[count][mutpos]=change;
            count++;
        }
        else
            count++;
    }
}

//counting distinct colors for given chromosome
int countDistinct(int a[], int n)
{
    int i, j, count = 1;
    for (i = 1; i < n; i++)
    {
        for (j = 0; j < i; j++)
        {
            if (a[i] == a[j])
            {
                break;
            }
        }
        if (i == j)
            count++;
    }
    return count;
}

//FINAL FITNESS FUNCTION which returns the minimum chromatic number as well
int fitness_chromatic(int pop_num, int optimalSol, int nodes, int (*newPop)[nodes], int(*arr)[nodes],
int *fit, int *chromatic_num, int *index){
    int i,j;
    int min_chromatic=nodes+1;
    for(i=0;i<pop_num;i++){
        fit[i]=0;
        for(j=0;j<nodes-1;j++){
            for(int k=j+1;k<nodes;k++){
                if(newPop[i][j]==newPop[i][k]){
                    if(arr[j][k]==1)
                        fit[i]++;
                }
            }
        }

        if(fit[i]==0){
            *chromatic_num = countDistinct(newPop[i],nodes);
            if(*chromatic_num<=optimalSol){
                printf("\noptimal solution reached: %d for index %d\n", *chromatic_num, i);
                for(int k=0;k<nodes;k++)
                    printf("%d ", newPop[i][k]);
                *index=i;
                printf("\n");
                if(*chromatic_num<min_chromatic){
                    min_chromatic = *chromatic_num;
                    return min_chromatic;
                }
            }
            if(*chromatic_num<min_chromatic){
                min_chromatic = *chromatic_num;
                *index=i;
            }
        }
    }
    return min_chromatic;
}

int main( )
{
    clock_t start, end;
    double cpu_time_used;

    start = clock();

    srand(time(0));

    char * S = 0;
    long length;
    int pop_num=200;
    int selec_num=120;
    char line[10];
    int row,col;
    char faltu;
    int optimalSol,nodes,edges;
    int max=0,count;
    FILE *fs;
    if((fs=fopen("games120.col","r"))==NULL)
    {
        printf("\nCan't open the file.\n");
        exit(1);
    }

    while(!feof(fs))
    {
        fscanf(fs,"%d %d %d\n",&optimalSol,&nodes,&edges);
        for(int j=1;j<=edges;j++)
        {
            fscanf(fs,"%c %d %d\n",&faltu,&row,&col);
            graph[row-1][col-1] = 1;
            graph[col-1][row-1] = 1;
        }
    }
```

```
    fclose(fs);
    //finding maxx
    for(int i=0;i<nodes;i++)
    {
        count=0;
        for(int j=0;j<nodes; j++)
        {
            if(graph[i][j]==1)
            {
                count++;
            }
        }
        if(count>max)
        {
            max=count;
        }
    }

    printf("\nOptimal solution=%d, nodes=%d, edges=%d\n", optimalSol, nodes, edges);
    getchar();

    int population[pop_num][nodes];
    randPopulation(pop_num, nodes, population,max);



    int min_chromatic=nodes+1;
    int iteration=1;
    int min_till=99;
    int flag=1 , tracker = 1;

    while(iteration<=100000)
    {
        int fitness[pop_num];
        memset(fitness,0,pop_num);
        int chromatic_num=6969, index=6969;
        int *p_chro=&chromatic_num;
        int *p_in=&index;


        min_chromatic = fitness_chromatic(pop_num, optimalSol, nodes, population, graph, fitness,
p_chro, p_in);
        if(chromatic_num<=optimalSol){
            printf("\nLOOP TERMINATED AT INTERATION %d AS THE OPTIMAL SOLUTION HAS
BEEN REACHED\n", iteration);
            printf("Minimum Chromatic number: %d at index: %d\n", min_chromatic, index);
            end = clock();
            cpu_time_used = ((double) (end - start));
            printf("\nTime in seconds: %lf\n", cpu_time_used/CLOCKS_PER_SEC);
            return 0;
        }
        if(min_chromatic<min_till){
            min_till=min_chromatic;
            tracker=0;
        }

        if(tracker==1000)
        {
            randPopulation(pop_num, nodes, population,min_till-1);
            tracker=0;
        }

        int selected[selec_num];
        selection_random(selec_num, pop_num, selected, fitness);

        int newPop[pop_num][nodes];

        memset( newPop, 0, pop_num*nodes*sizeof(int) );

        crossover(selec_num, pop_num, selected, nodes, population, newPop);

        mutation(pop_num, nodes,newPop);
        mutation2(pop_num, nodes,newPop);
        memcpy (population, newPop, pop_num*nodes*sizeof(int));

            tracker++;

        iteration++;

    }

    printf("Not found! Minimum Chromatic number: %d\n", min_till);
    end = clock();
    cpu_time_used = ((double) (end - start));
    printf("\nTime in seconds: %lf\n", cpu_time_used/CLOCKS_PER_SEC);
    return 0;
}
```