COMPARATIVE ANALYSIS OF PERFORMANCES OF GENETIC ALGORITHM & ANT COLONY OPTIMIZATION ALGORITHM FOR SOLVING GRAPH COLORING PROBLEM

(A MAJOR PROJECT REPORT)

Submitted By

Bina Rani Saha, Univ. Roll No.-21301217094, Univ. Reg. No.- 172131010027 of 2017-18

Odyssey Roy, Univ. Roll No.-21301217063, Univ. Reg. No.- 172131010058 of 2017-18

Divankit Sha, Univ. Roll No.-21301217082, Univ. Reg. No.- 172131010039 of 2017-18

Soumyajit Saha, Univ. Roll No.-21301217024, Univ. Reg. No.- 172131010097 of 2017-18

Under the Supervision of

Prof. Arnab Kole

Assistant Professor

In partial fulfillment for the award of the degree of

BACHELOR OF COMPUTER APPLICATION

THE HERITAGE ACADEMY
MAULANA ABUL KALAM AZAD UNIVERSITY OF TECHNOLOGY

2020

ACKNOWLEDGEMENT

We would take the opportunity to thank ***Prof. (Dr.) Gour Banerjee, Principal, The Heritage Academy*** for allowing us to form a group of 4 people and for supporting us with the necessary facilities to make our project worth.

We are thankful to ***Prof. Arnab Kole (Assistant Professor, BCA)*** our ***Project Guide*** who constantly supported us, and ***Prof. Avik Mitra (Assistant Professor, BCA), the Project Coordinator*** for providing and clarifying the administrative formalities related to project proceedings. Their words and instructions have assisted us to excel in our project.

We thank all our other faculty members and technical assistants at The Heritage Academy for paying a significant role during the development of the project. Last but not the least, we thank all our friends for their cooperation and encouragement.

Signature: _____

(Bina Rani Saha)

Signature: _____

(Odyssey Roy)

Signature: _____

(Divankit Sha)

Signature: _____

(Soumyajit Saha)

The Heritage Academy

Kolkata

PROJECT CERTIFICATE

This is to certify that the following students:

Name of the students	Roll No.	Registration No.
1. BINA RANI SAHA	21301217094	172131010027
2. ODYSSEY ROY	21301217063	172131010058
3. DIVANKIT SHA	21301217082	172131010039
4. SOUMYAJIT SAHA	21301217024	172131010097

of 3rd Year 2nd Semester in **BCA(H)** have successfully completed their Major Project Work on

COMPARATIVE ANALYSIS OF PERFORMANCES OF GENETIC ALGORITHM & ANT COLONY OPTIMIZATION ALGORITHM FOR SOLVING GRAPH COLORING PROBLEM

towards *partial fulfillment* of Bachelor of Computer Applications from Maulana Abul Kalam Azad University of Technology, West Bengal in the year **2020.**

Prof.(Dr.)Gour Banerjee **Principal** **The Heritage Academy**	**Prof. Arnab Kole** **Project Guide** **Assistant Professor** **Department of BCA** **The Heritage Academy**	**Prof. Avik Mitra** **Project Coordinator** **Assistant Professor** **Department of BCA** **The Heritage Academy**

ABSTRACT

Graph coloring plays an important role in the field of graph theory. Though various advancements have been made in the arena of modern computer algorithms, still graph coloring algorithms are well known research topic in Mathematics and Computer Science around the world. Meta-heuristic Search Algorithms are very efficient optimization techniques when it comes to problems that require large solution space. Since the working principle of evolutionary algorithms is based on the use of different heuristics in probabilistic manner, it can produce optimal solution for graph coloring. In this project work, we have tried to solve Graph Coloring Problem using two metaheuristic techniques – Genetic Algorithm and Ant Colony Optimization Algorithm and compare the performances of the algorithms. The genetic algorithm described here utilizes more than one parent selection and mutation methods depending on the state of fitness of its best solution. This results in shifting the solution to the global optimum more quickly than using a single parent selection or mutation method. An ACO algorithm is presented for the graph coloring problem. This ACO algorithm confirms to Max-Min Ant System structure and exploits a local search heuristic to improve its performance. Instead of giving exact solution, it tries to produce an approximate one that takes less time.

TABLE OF CONTENTS

List of Figures:

List of Tables:

1. INTRODUCTION

It has already been shown that the decision version of graph coloring problem is NP complete **[1]** and the approximate version is NP hard **[2]**. Hence till date no deterministic algorithm has been proposed which can produce the optimal solution of this problem in polynomial time. It has also been shown over the years that heuristic approaches **[3]** are worthy of use to produce optimal or near optimal solution for this type of problems. Graph coloring problem is meant to be assigning colors to the certain elements of a graph subject to certain constraints. Graph coloring is a special case of graph labeling. The most common graph coloring problem is Vertex coloring. The problem is, m colors are given, and we have to find a way of coloring the vertices of a graph in such a way that no two adjacent vertices are colored using the same color. Similarly, an edge coloring assigns a color to each edge so that no two adjacent edges are of the same color, and a face coloring of a planar graph assigns a color to each face or region so that no two faces that share a boundary have the same color. 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 proper, improper and chromatic coloring of a graph:

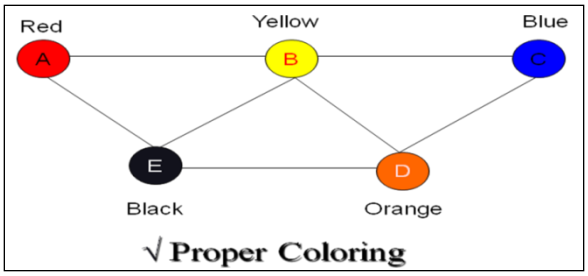

Figure 1: Proper Graph Coloring

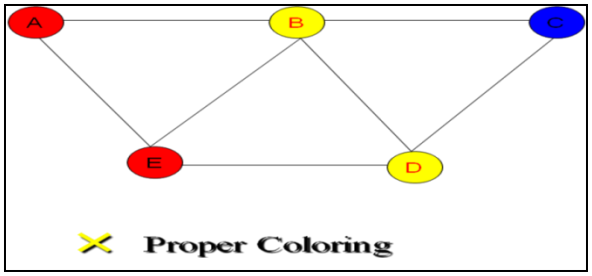

Figure 2: Improper Graph Coloring

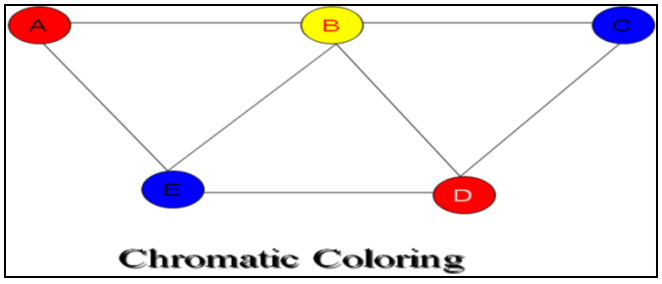

Figure 3: Chromatic Graph Coloring

2. EARLIER WORK

A variety of research works in the field of computer science and mathematics have been implemented to solve the theoretical areas of graph coloring. Hertz et. al. **[4]** developed one Tabu Search method to solve graph coloring problem in which a solution can be obtained based on the movement to a neighbor solution. Another evolutionary method has been proposed by Fleurent et. al. **[5]** for graph coloring that uses entropy measurement. Deep Learning **[6]** method has also been used to solve GCP. Filho et. al. **[7]** developed a Constructive Genetic Algorithm (CGA) with column generation. Sethumadhavan et. al. **[8]** proposed a genetic algorithm for graph with conflict gene removal procedures. Ant Colony Optimization proposed by Bui et. al. **[9]** for graph coloring improved the performance of Jagota for some benchmark graphs. Han et. al. **[10]** designed a bi-objective genetic algorithm to solve GCPGCP has been solved using adjacency matrix and link list based exact algorithm by Shukla et. al. **[11, 12]**. Some direct approaches like Branch-and-Bound, Branch-and Cut and Backtracking algorithms **[13, 14, 15]** 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. **[16]**. 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 **[17, 18]**. An Ant Colony approach based algorithm has been proposed in **[19, 20]** to solve GCP. Dahl **[21]** applied one Artificial Neural Networks approach to solve graph coloring problem by, and more recently the algorithm has been modified by Jagota **[22]**.

3. APPLICATIONS

There are many areas where graph coloring has been used to solve practical problems like timetable scheduling of schools and colleges **[23]**, examination scheduling of colleges and universities **[24]**, register allocation **[25]**, microcode optimization **[26]**, channel routing **[27],** printed circuit testing **[28],** allocation of electronic bandwidth **[29]**, the design and operation of flexible manufacturing systems **[30],** timetable scheduling **[31]** etc where the evolutionary algorithms that use heuristic approaches have been proposed to produce optimal or suboptimal solution in a satisfactory amount of time.

4. ANT COLONY OPTIMIZATION

Ant colony optimization (ACO) is a population-based metaheuristic that can be used to find approximate solutions to difficult optimization problems. Ant algorithms were first proposed by Dorigo et. al. **[32]** as a multi-agent approach to solve difficult combinatorial optimization problems such as Travelling Salesman Problem TSP **[33]** and Quadratic Assignment Problem (QAP). ACO algorithm has also been used to solve Constraint Satisfaction Problem **[34].** ACO algorithm is an iterative optimization technique. In each iteration, a number of ants (a set of software simulated agents) fabricate feasible solutions using heuristic information and the exploit the solution information of preceding batch of ants. These exploitations are individualized by a trail of digital pheromones, which is embedded on the individual segments of a solution. Small amounts are deposited during initialization while larger quantities are embedded, at the conclusion of each generation, in proportion to solution grade. Pheromone can be embedded on the constituents and/or the association used in a solution according to the problem. At each instance of newly generated solutions, each ant hand-picks probabilistically from the set of applicable constituents based on an amalgamation of the quantity of pheromone on that constituent and a heuristic value of that constituents' utility. The general ACO framework is:

```
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 4: General ACO Framework

5. PROPOSED ACO ALGORITHM FOR GCP

ALGORITHM:

INPUT: Adjacency Matrix (n x n) where N is the number of vertices

1. Create one coloration using a single color (1) to all the vertices
2. For cycle = 1 to max_iteration or terminating condition not meet
3. Randomly distribute population of ants to vertices
4. For cycle = 1 to no_of_ants
5. If xth ant discover confliction on yth vertices with maximum degree
6. q = color of yth vertex
7. While confliction not removed && color of yth vertex not equal to its adjacent vertices
8. q = q+1
9. color of yth vertices = q
10. Update local pheromone
11. At certain interval of iteration update global pheromone
12. Output a Best coloration

In the graph coloring problem the pheromone concept of the real world situation of ant colony has been implemented by the confliction of colors in different vertices. When a particular ant is visiting a vertex it will check for a conflict i.e. adjacent vertices have same colors, it's trying to remove the conflict by choosing another color from the set of available colors (max_degree[graph] + 1) if no conflict is found the color set doesn't change. From the result of those the local pheromone table is updated. For all the ants when they should updated their local pheromone table; after coming outside the loop, the global pheromone table is then updated with the result of local pheromones.

6. GENETIC ALGORITHM

Genetic Algorithms (GAs) **[35]** is one search strategies that are inspired by natural evolutionary processes and have the capacity to overcome the problem posed by the existence of local optima in large search spaces. The basic idea behind GA is same as the natural evolution process through natural selection based on the theory of survival of the fittest invented by Darwin. The working principle of GA starts with a set of (suboptimal) solutions and the process evolves through a number of iterations until some predefined termination condition is satisfied. At each intermediate stage, the old solution set is replaced by a newly generated solution set, known as new population where the individuals of a population are processed with the help of some commonly used GA operators in such a way that the quality of the new generation, in general, is improved in comparison with old generation. In this way a better solution is expected to be obtained and the process continues until the end of the search with the expectation that the best, or a near-best solution will be returned by the GA process. There are certain features associated with a GA. These are: (i) Chromosomes, (ii) Procedures to encode a solution as a chromosome, and procedure to decode a chromosome to the corresponding solution, (iii) Fitness function to evaluate each solution, i.e., each chromosome, (iv) Population size, (v) Initial population, (vi) The mating pool, i.e., the set of chromosomes selected from current population who will generate the new population/generation, (vii) GA operators, e.g., selection, crossover, and mutation, (viii) Various GA parameters, e.g., crossover probability (pc), mutation probability ($p\mu$), population size etc and (ix) Termination condition. The basic GA procedure is:

Step 1.	Initialize the population. Call this the current population.
Step 2.	Repeat Step 3 through Step 5 till termination condition is satisfied.
Step 3.	Apply selection operation on the current population to obtain the mating pool.
Step 4.	Apply crossover and mutation operators on the mating pool to generate the new population.
Step 5.	Replace the current population by the new population.
Step 6.	Return the best solution of the current population.

Figure 5: Basic GA Procedure

7. PROPOSED GENETIC ALGORITHM FOR GCP

ALGORITHM:

INPUT: Adjacency Matrix (n x n) where N is the number of vertices

Step 1) Create random initial pool of chromosomes (population)
Step 2) Sort the chromosomes of the pool in ascending order of their fitness value
Step 3) Apply the rep_ op to the invalid chromosomes
Step 4) Calculate the fitness of the pool
Step 5) Pbest = best individual
Step 6) For generation = 1 to max_iteration
    Step 6.1) Perform crossover between any two random pair of chromosomes with probability pc
    Step 6.2) Apply the rep_ op to the invalid offspring
    Step 6.3) Select the best individuals from the newly generated offspring
    Step 6.4) Apply mp_sp_mutation to the pool of chromosome for certain no_of_iteration
    Step 6.5) Calculate the fitness of the pool
    Step 6.6) Take improved chromosomes for next generation
Step 7) Output the best coloration

Function: mp_sp_mutation:

Step 1) Choose a particular chromosome
Step 2) Reduce the multiple number of colors in that chromosome (e.g. by replacing two of the used colors by other used colors at two places)
Step 3) If the coloration is invalid then apply the rep_ op (here it'll try to repair the invalid coloration to valid one using the reduced number of colors available)
Step 4) If it succeeds or the coloration is valid then place this chromosome in the pool
    Else goto step 1

8. RESULT AND DISCUSSION

The algorithms have been tested on some of the DIMACS **[36]** benchmarks graphs. The algorithms have been run in Dev C++ compiler with Operating System: Windows 10 Home 64 bit, Processor: Intel Core i3-5005U CPU @ 2.10GHz x 4 and RAM: 2 GB. GA has been run with crossover probability 0.1 and population size 10. Max_Iteration has been set 10 for the graphs having less than 100 nodes and 20-100 for the others. The following table represents the detailed description of the result found by the algorithms.

Table 1: Obtained Chromatic Number & ACPU Time

Graph Instance	Vertex	Edge	Expected Chromatic Number	Genetic Algorithm Approach		Ant Colony Optimization Approach	
				Chromatic Number Obtained	ACPU Time (Sec)	Chromatic Number Obtained	ACPU Time (Sec)
myciel3.col	11	20	4	4	1.250	4	0.922
myciel4.col	23	71	5	5	1.687	5	0.172
myciel5.col	47	236	6	6	1.797	6	0.281
myciel6.col	95	755	7	7	7.547	7	0.344
myciel7.col	191	2360	8	8	8.423	8	1.031
mug88_1.col	88	146	4	4	16.933	4	0.508
miles250.col	128	387	8	8	8.657	9	0.863
3-Insertions_3.col	56	110	4	4	2.390	4	0.032
huck.col	74	602	11	11	2.604	11	0.165
games120.col	120	638	9	9	4.358	9	1.876
jean.col	80	254	10	10	3.504	10	0.353
anna.col	138	493	11	11	8.654	12	1.021
1-FullIns_4.col	93	593	?	11	1.046	11	0.645
1-FullIns_3.col	30	100	?	9	0.047	8	0.17
2-FullIns_3.col	52	201	?	10	1.567	10	0.15

9. LIMITATION AND FUTURE SCOPE

9.1 LIMITATION

The GAGCP produces optimal coloring in minimum possible time for almost all of the above benchmark graphs. ACOGCP produces optimal coloring except 2 graphs. However, time of execution increases slightly in case dense graphs in GA approach.

9.2 FUTURE SCOPE

To get further better results, we need to make modifications to the proposed algorithm. We might also use different metaheuristic approaches like ANN or Quantum Annealing or different algorithms. We might also build a hybrid algorithm using GA and ACO or GA & Simulated Annealing. Other meta-heuristic algorithms like Tabu Search and Swarm Optimization techniques can also be tried.

10. CONCLUSION

The above table shows that ACO approach is giving more promising result than GA approach in terms of ACPU time for all the above mentioned graphs. But GA approach gives better result than ACO approach in terms of optimal chromatic number for all of the above benchmarks. For some of the graphs where expected chromatic number has not been given in the benchmark sets, both the algorithms find the chromatic numbers and in those cases performance of ACO Algorithm found more suitable than Genetic Algorithm.

11. REFERENCES

[1] Garey M.R. and Johnson D.S. Computers Intractability: A Guide to the Theory of NP-Cpmpleteness. W.H. Freedman and Co., 1979.

[2] Baase, S. and Gelder A.V., Computer Algorithms: Introduction To Design and Analysis. Addison-Wesley, 1999.

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

[4] Hertz A and Werra D,: "Using tabu search techniques for graph coloring", Computing, vol. 39, no. 4, pp. 345-351, 1988.

[5] Fleurent C and Ferland J.A.: "Genetic and hybrid algorithms for graph coloring", Annals of Operations Research, vol. 63, pp. 437-464, 1996.

[6] Henrique Lemos, Marcelo Prates, Pedro Avelar, Luis Lamb, Graph Colouring Meets Deep Learning: Effective Graph Neural Network Models for Combinatorial Problems, 31st International Conference on Tools with Artificial Intelligence (ICTAI), IEEE, 2019

[7] Filho Geraldo Ribeiro and Lorena Luiz Antonio Nogueira: "Constructive Genetic Algorithm and Column Generation: an Application to Graph Coloring",

[8] Sethumadhavan Gopalakrishnan , Marappan Raja : A Genetic Algorithm for Graph Coloring using Single Parent Conflict Gene Crossover and Mutation with Conflict Gene Removal Procedure, 978-1-4799-1597-2/13/$31.00 ©2013 IEEE.

[9] T. N. Bui and C. Patel, "An Ant system Algorithm for Coloring Graphs", Computational Symposium on Graph Coloring and its Generalizations, COLOR02, Cornell University, September 2002.

[10] Lixia Han and Zhanli Han, A Novel Bi-objective Genetic Algorithm for the Graph Coloring Problem, 2010 Second International Conference on Computer Modeling and Simulation.

[11] Ajay Narayan Shukla, M.L.Garg, An approach to solve graph coloring problem using adjacency matrix, Bioscience Biotechnology Research Communications, 12(2):472-477, June, 2019

[12] Ajay Narayan Shukla, Vishal Bharti, Madan L.Garg, A Linked List-Based Exact Algorithm for Graph Coloring Problem, International Information and Engineering Technology Association, Vol. 33, No. 3, pp. 189-195, June, 2019

[13] Anuj Mehrotra and Michael A. Trick. A column generation approach for graph coloring. INFORMS Journal on Computing, 8(4):344–354, 1996.

[14] Isabel Méndez-Diaz and Paula Zabala. A branch-and-cut algorithm for graph coloring. Discrete Mathematics, 154(5):826–847, 2006.

[15] R. Monasson. On the analysis of backtrack procedures for the coloring of random graphs. In E. Ben-Naim, H. Frauenfelder, and Z. Toroczkai, Complex Networks, pages 235–254. Springer, 2004.

[16] Díaz, Isabel Méndez, and Zabala, Paula. 1999, A Generalization of the Graph Coloring Problem, Departamento de Computacion, Universidad de Buenes Aires.

[17] Ali, F. F., Nakao, Z., Tan, R. B., and Yen-Wei, Chen. 1999. An evolutionary approach for graph coloring. In Proceedings of The International Conference on Systems, Man, and Cybernetics, 527- 532. IEEE.

[18] Tagawa, K., Kanesige, K., Inoue, K., and Haneda, H. 1999. Distance based hybrid genetic algorithm: an application for the graph coloring problem. In Proceedings of Congress on Evolutionary Computation, 2332. Washington DC.

[19] Costa D., Hertz A. and Dubuis O.: "Ants Can Color Graphs", Journal of the Operational Research Society, 48(1997), 295-305

[20] Anindya J. Pal, Biman Ray, Nordin Zakaria, Ken Naono & Samar Sarma, "Generating Chromatic Number of a Graph using Ant Colony Optimization and Comparing the Performance with Simulated Annealing", Vol.50, pp 629-639, Procedia Engineering, 2012.

[21] E. D. Dahl, "Neural Networks algorithms for an NP complete problem: Map and graph coloring", IEEE International Conference on Neural Networks, vol. 3, pp. 113-120.

[22] A. Jagota, "An adaptive, multiple restarts neural network algorithm for graph coloring", European Journal of Operational Research, vol. 93, pp. 257-270, 1996.

[23] Wood D.C.: "A technique for coloring a graph applicable to large scale time-tabling problems", Computer Journal, vol. 12, pp. 317-319, 1969.

[24] Leighton F.T., "A graph coloring algorithm for large scheduling problems", Journal of Research of the National Bureau of Standards, vol. 84, no. 6, pp. 489-505, 1979.

[25] Chow F.C. and Hennessy J.L., "Register allocation by priority based coloring", Proceedings of the ACM Sigplan 84 symposium on compiler construction New York, pp. 222-232, 1984.

[26] Micheli G. D.: Synthesis and Optimization of Digital Circuits. McGraw-Hill, 1994.

[27] Sarma S.S, Mondal R. and A. Seth: “Some sequential graph coloring algorithms for restricted channel routing”, INT. J. Electronics, vol. 77, no. 1, pp. 81-93, 1985.

[28] A. Gamst, Some lower bounds for class of frequency assignment problems, IEEE Transactions on Vehicular Technology, vol. 35, no. 1, pp. 8-14, 1986.

[29] Micheli G.D., Synthesis And Optimization Of Digital Circuits. McGraw-Hill, 1994.

[30] S.S. Sarma, R. Mondal and A. Seth, “Some sequential graph coloring algorithms for restricted channel routing”, INT. J.Electronics, vol. 77, no. 1, pp. 81-93, 1985.

[31] Ramzi A. Haraty, Maram Assi, Bahia Halawi, “Genetic Algorithm Analysis using the Graph Coloring Method for Solving the University Timetable Problem”, Science Direct, Procedia Computer Science, Volume 126, Pages 899-906, 2018

[32] M. Dorigo, V. Maniezzo, A. Colorni, Ant system: Optimization by a colony of operating agents, IEEE Transactions on Systems, Man and Cybernatics-Part B, vol. 26, pp. 29-41, 1996

[33] M. Dorigo, L. M. Gambardella, Ant colony systems: A cooperative learning approach to the travelling salesman problem, IEEE Transactions on Evolutionary Computation, vol. 1, pp. 53-66, 1997

[34] Takuya Masukane, Kazunori Mizuno, Hiroto Shinohara, Ant Colony Optimization with Negative Feedback for Solving Constraint Satisfaction Problems, Conference on Technologies and Applications of Artificial Intelligence (TAAI), DOI**:** 10.1109 / TAAI.2018.00041, IEEE, December, 2018

[35] Introduction to Soft Computing-Neuro-Fuzzy and Genetic Algoritms, Samir Roy & Udit Chakraborty, Pearson

[36] https://mat.tepper.cmu.edu/COLOR/instances.html.

APPENDIX

ACO APPROACH FOR GCP:

//Program for solving problems of Graph Coloring using Ant Colony Optimizaton(ACO)//

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
#include<string.h>
#define size 2500
int graph[size][size]={{0}};
int color[size];
int max,opt=1;
void c_graph(int);  //Check for complete graph
int max_deg(int); //To find maximum degree of the graph
void compute(int,int); //Main computation
int color_count(int); //Count the number colors used in coloration
int check(int n,int c[],int a[][size]); //Check for the validity of a coloration
void writetofile(int coloration[],char file_out[],double tm); //Writes output coloration to file
int new_color;
int result=0;
int node,edges,mincol,row,col;
char filename[100],ipfile[100],ipfl[100];
char file_out [100] = "ACO_";
char opfile[100]="RESULT.txt";
double tm;
int main ()
{
        int i,j;
        int maxdeg;
```

```
FILE *fs,*fs1;
int ch;
int value;
char c, faltu;
int nclr;
clock_t first,second;
system("clear");
srand((unsigned)time(NULL));
printf("\nColoration by Ant Colony Optimization...\n");
printf("\nFilename\tVertex\tEdge\tExpected Chromatic Number\tAchieved Chromatic
Number\tCPU Time(SEC)");
if((fs=fopen("miles250.col.txt","r"))==NULL)
{
        printf("\nCan't open the file.\n");
        exit(1);
}
while(!feof(fs))
{
        fscanf(fs,"%s\n",ipfile);
        strcpy(ipfl,ipfile);
        if((fs1=fopen(strcat(ipfile,"miles250.col.txt"),"r"))==NULL)
        {
        printf("\nCan't open the file.\n");
                exit(1);
         }
        strcpy(filename,file_out);
        strcat(filename,ipfile);
        fscanf(fs1,"%d %d %d\n ",&mincol,&node,&edges);
        first=clock(); /* Gets system time */
        for(j=1;j<=edges;j++)
        {
```

```
        fscanf(fs1,"%c %d %d\n",&faltu,&row,&col);
        graph [row][col] = 1;
        graph[col][row] = 1;
}
fclose(fs1);
maxdeg=max_deg(node);
nclr=maxdeg;
c_graph(node);
for(i=1;i<=node;i++)
{
        color[i]=1;
}
do
{
        compute(node,nclr);
        result=check(node,color,graph);
         new_color=color_count(node);
}while(result!= 1);
second=clock(); /*Grts system time again*/
new_color=color_count(node);
tm=(double)(second-first)/CLOCKS_PER_SEC;
        printf("\n\n%s\t%d\t%d\t%d\t%d\t%lf",ipfl,node,edges,mincol,new_color,
tm);
writetofile(color,filename,tm);
maxdeg=0;
nclr=0;
result=0;
for(i=1;i<=size;i++)
        for(j=1;j<=size;j++)
        graph[i][j]=0;
for(i=1;i<=size;i++)
```

```
                color[size]=0;
    }
fclose(fs);
printf("\n\nComprehensive Output stored in the output file: %s",opfile);
printf("\n\nPress Enter to End the Program....");
        getch();
        return 0;
}//End of main()
//Function compute().This function assigns colors to nodes
void compute(int n,int d) //d=maxdeg
{
        int value,j,k;
        for(j=2;j<=n;j++)
        {
                new_color=color_count(node);
                result=check(node,color,graph);
                if(result==1)
                        return;
                for(k=1;k<=n;k++)
                {
                        if(j!=k )
                        {
                                value=graph[j][k]*color[k];
                                if( value==color[j])
                                {
                                        if(color[k]<=d+1)
                                                color[k]=color[k]+1;
                                        else
                                        color[k]=color[k]%(d+1)+1;
                                        j=2;
                                        k=1;
```

```
                                }
                        }
                }
        }
}//End of compute()
//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 max_deg().this function calculates the maximum degree of the graph
```

```
int max_deg(int node)
{
	int v[size];
	int i,j,sum;
	int max; // maximum degree of the graph
	char c;
	for(i=1;i<=node;i++)
	{
		sum=0;
		for(j=1;j<=node;j++)
		{
			sum=sum+graph[i][j];
		}
		v[i]=sum;
	}
	max=0;
	for(i=1;i<=node;i++)
	{
		if(max<v[i])
		max=v[i];
	}
	scanf("%c",&c);
	return max;
}//end of max_deg()
//Function c_graph.this function checks whether the graph is complete
void c_graph(int n)
{
	int i,j,count=0;
	for(i=1;i<=n;i++)
	{
		for(j=i;j<=n;j++)
```

```
                {
                        if(graph[i][j])
                                count++;
                }
        }
        if(count==((n*n)-n))
        {
                printf("\n\n This is a complete graph!!!");
                printf(" so number of colors would require for this graph: %d",n);
                exit(1);
        }
}// end of c_graph()
//Function check..this function checks the validity of a coloration
int check(int n,int c[],int a[][size])
{
        int i,j;
        char ch;
        for(i=1;i<n;i++)
                for(j=i+1;j<=n;j++)
                {
                        if(c[i]==0||c[j]==0)
                        {
                                printf("\n\t\t\tCaution:: value 0 entered into the color!!!\n");
                                scanf("%c",&ch);
                        }
                        if(c[i]==c[j])
                                if(a[i][j]==1)
                                        return 0;
                }
        return 1;
}//End of check()
```

```
//Function writetofile..this function write valid coloration with elapsed time to output file
void writetofile(int coloration[],char file_out[],double tm)
{
	FILE *fout;
	int j, no_clr;
	no_clr= color_count(node);
	if(no_clr<=mincol+5)
	{
		if((fout=fopen(file_out,"w"))== NULL)
		{
			printf("\n Can't open the file.\n");
			exit(1);
		 }
		fprintf(fout,"Coloration by Ant Colony Optimization...");
		for(j=1;j<=node;j++)
		{
			fprintf(fout,"\n\nnode[%d]-> %d",j,coloration[j]);
		}
		fprintf(fout,"\n\nNumber of colors used: %d",new_color);
		fprintf(fout,"\n\nTime taken: %lf seconds",tm);
		fclose(fout);
		if((fout=fopen(opfile,"a"))== NULL)
		{
			printf("\n Can't open the file.\n"); exit(1);
		}
		fprintf(fout,"%s\t%d\t\%d\t%d\t%d\t%lf\n\n",ipfl,node,edges,mincol,new_color,t
		m);
		fclose(fout);
	 }
}//End of writetofile()
```

GA APPROACH FOR GCP:

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
#include<string.h>
#include<ctype.h>
#include<math.h>
#define size 200
void graphcolor(void);
int check(int,int[],int[][size]);//validity of coloration
int fitness(int,int[]);//counts the no of colors used in coloration
void selection(int popu[size][size],int fit[size],int vertex);//sorts the population on fitness
int max_deg(int node);//calculation of max degeree
void write_to_file();//output file
void writetm_to_file(double);//output time to file
void satoda(int[][size],int[],int,int);//copies single dimension to double dimension
void datosa(int[],int[][size],int,int); //copies row from a 2d array to 1d array
void satosa(int[],int[],int); // copy array to another array
void datoda(int[][size],int[][size],int,int,int); //copy row from a 2d array to anothe 2d array
void cross_modi_mut(); //working with crossover and modified mutation operation
void modi_mut(int k,int fault[],int nupopu[][size]); //modified mutation
void norml_mut(int i,int j); //working for single point mutation
void special_mut(int i,int j); //working for multi point mutation
char c;
int fault[size]={0},gcount = 1,maxdgree= 0;
int ranval = 0,v,fitest = 1000, fitcnt=0, ft=1000, popusize,nclr, whilecntr=1;
int i,j,k=0,l,m,x,z=1,clor,vertx,edges=0,row,col,mincol,no_clr;
int ran,ran1,ran2,ran3,f1,max=0;
int adjm[size][size]={{0}};
int popu[size][size]={{0}},nupopu[size][size]={{0}};
int supply[size][size]={{0}};
```

```
int color[size]={0}, mat[size][size]={{0}},ya[size]={0},y=0;
int fit[size]={0},fit1[size]={0},splyfit[size]={0},result,nuclor;
int temp[size]={0},temp1[size]={0};
int status[size]={0};
int crosd1[size]={0},crosd2[size]={0};
int array[size][size]={{0}} ;
char faltu;
double tm;
FILE * fout;
char file_out [ 30 ] = "GA_";
int last_color;
int main()
{
	FILE *fptr;
	int size1,choice=0,ch=0;
	int flag=1,i=1,j=1,n=1,v=0,e=0;
	char filename [20];
	clock_t first,second;
	strcat(file_out,"anna.col.txt");
	system("clear");
	//clrscr();
	srand((unsigned)time(NULL));
	if((fptr=fopen("anna.col.txt","r"))==NULL)
	{
		printf("\nCan't open the file\n");
		exit(1);
	}
	first=clock();
	fscanf(fptr, "%d %d %d\n",&mincol, &vertx, &edges);
	for(j=1;j<=edges;j++)
	{
```

```
            fscanf(fptr, "%c %d %d\n",&faltu, &row, &col);
            adjm[row][col] = 1;  //creation of adjacency matrix
            adjm[col][row] = 1;
        }
        fclose(fptr);
        maxdgree=max_deg(vertx);
        printf("\nExpected Chromatic Number:%d\n",mincol);
        system("pause");
        //getch();
        last_color=maxdgree + 1;
        graphcolor();
        second=clock();
        tm=(double)(second-first)/CLOCKS_PER_SEC;
        printf("\nChromatic number Obtained Using GA: %d\n",no_clr);
        printf("\nTime Taken: %lf seconds\n",tm);
        printf("\nDetailed Output Stored in the File: %s\n",file_out);
        writetm_to_file(tm);
        system("pause");
        //getch();
        return 0;
} //End of main
/*Function Graph Color.This function colors a graph using mutation and crossover operators*/
void graphcolor(void)
{
        int flag=0,newflag=0,tempfit,temp1fit,dcount=0;
        int tempp[size],ii,jj;
        nclr=maxdgree;
        popusize=10;
        j=1;
        do
        {
```

```
        for(i=1; i<=vertx; i++)
        {
                color[i]=rand()%nclr+1;  //Random coloration generation
        }
        satoda(popu,color,j,vertx); //Initial population creation
        fit[j] = fitness(vertx,color);
        j++;
}while(j<=popusize);
cross_modi_mut(); //Calling cross_modi_mut function
for(i=1, j=1; i<=popusize, j<=popusize; i++, j++)
{
        datoda(popu,nupopu,i,j,vertx);
        fit[i]=fit1[j];
        datoda(supply,nupopu,i,j,vertx);
        splyfit[i]=fit1[j];
}
while(1)
{
        selection(popu,fit,popusize);
        datosa(temp,popu,1,vertx);
        if(check(vertx,temp,adjm))
        {
                write_to_file();
                //printf("\n1st Fitness: %d\n",fitness(vertx,temp));
                //system("pause");
                if(fitness(vertx,temp) <= mincol)
                break;
        }
        ranval = rand () % 100 + 1;
        if( ranval <= 40 )
        {
```

```
        //crossover
        cross_modi_mut();
        selection(nupopu,fit1,f1);
        for(i=(popusize/2)+1 , j=1; i<=popusize; i++ , j++)
        {
                datoda(popu,nupopu,i,j,vertx);
                fit[i]= fit1[j];
        }
        selection(popu,fit,popusize);
        datosa(temp,popu,1,vertx);
        ft= fitness(vertx,temp);
        if(check(vertx,temp,adjm))
        {
                write_to_file();
                //printf("\n2nd Fitness: %d\n",fitness(vertx,temp));
                //system("pause");
                if(fitness(vertx,temp) <= mincol )
                        break;
        }
}//end of if (ranval <= 40)
else
{
        //Mutation
        ii=rand() % popusize + 1;
        jj = rand() % vertx + 1;
        norml_mut( ii,jj );
        selection(popu,fit,f1);
        datosa(temp,popu,1,vertx);
        tempfit = fitness(vertx , temp);
        flag = 1;
        newflag=0;
```

```
for(ii=1 ; ii<= f1; ii++)
{
        jj = rand () % vertx + 1;
        special_mut(ii,jj); //Calling special function
        datosa(temp,popu,1,vertx); //Obtain best coloration so far
        tempfit= fitness( vertx,temp);
        if(fitness(vertx,temp)<=mincol && check(vertx,temp,adjm))
                break;
        special_mut(ii,jj); //Calling special function
        datosa(temp1,popu,ii,vertx);
        temp1fit= fitness( vertx, temp1);
        if( temp1fit <= tempfit )
        {
                if( temp1fit == tempfit )
                {
                        if( flag <= 2 )
                        {
                                ii = 1;
                                flag++;
                                continue;
                        }// End of  ( flag <= 2 )
                        else
                        {
                                if( ii == f1 )
                                {
                                        if(newflag == 0 &&  z % 2 == 1)
                                        {
                                                newflag = 1;
                                                for( i=2, j=1; i<=f1; i=i+2,
                                                j++ )
                                                {
```

```
                                                    datoda(popu,supply,i,j,vertex);
                                                        fit[i]=splyfit[j];
                                                    }
                                                    cross_modi_mut();

                for(i=(popusize/2)+1,j=1;i<=f1;i=i++,j++ )
                {
                        datoda(popu,nupopu,i,j,vertex);
                        fit[i]=fit1[j];
                }
        } // Enf  of if(newflag == 0 &&  z % 2 == 1)
        selection(popu,fit,f1);
        break;
}//Enf of if( ii == f1 )
                else
                {
                        special_mut(ii,jj); //Calling special mutation
                        continue;
                }//End of else of if(ii == f1)
        }//End of else of if(flaf <= 2)
}//Enf of if( temp1fit == tempfit )
else
{
        if( temp1fit < tempfit )
        {
                ii=1;
                flag=0;
                tempfit=temp1fit;
                continue;
        }
```

```
}//End of else of if if( temp1fit == tempfit )
}//End of else of if if( temp1fit <= tempfit )
}//End of for(ii=1 ; ii <= f1; ii++)
selection(popu,fit,f1);
datosa(temp,popu,1,vertx);
ft = fitness(vertx,temp);
fitest=ft;
if(check(vertx,temp,adjm))
{
        write_to_file();
        //printf("\n3rd Fitness: %d\n",fitness(vertx,temp));
        //system("pause");
        if( fitness(vertx,temp) <= mincol )
                break;
  }
}//End of else of if of(ranval<=40)
z++;
 //if(fitness(vertx,temp)<=mincol && check(vertx,temp,adjm))
//break;
}//End of whie(1)
datosa(temp,popu,1,vertx);
} //End of function
//Function cross_modi_mut.This function performs the crossover between two chromosomes
//and tries to make newly generated chromosomes valid coloration using modified mutation
void cross_modi_mut()
{
        result=0;
        k=0;
        f1=1;
        for(j=1; j<= popusize; j++)
        {
```

```
        status [ j ] = 0;
        fault [ j ] = 0;
}
for (v= 1; v <= ( popusize - 1 ); v = v + 2 )
{
        ran1 = rand ( ) % vertx + 1;
        //printf("\nran1= %d",ran1);
        ran2 = rand ( ) % vertx + 1;
        //printf("\nran2= %d",ran2);
        //getch();
        datosa(crosd1,popu,v,vertx);
        datosa(crosd2,popu,v+1,vertx);
        if ( ran1 < ran2 )
        {
                i= ran2;
                ran2= ran1;
                ran1= i;
        }
        for ( i = 1,j = vertx - ran2 + 1; i <= ran2; i++, j++ )
        {
                ran1 = crosd1 [ i ];
                crosd1 [ i ] = crosd2 [ j ];
                crosd2 [ j ] = ran1;
        }
        result= check ( vertx, crosd1, adjm );
        satoda ( nupopu, crosd1, f1, vertx  );
        if(result == 1)
        status [ f1 ] = result;
        else
                fault[++k]=f1;
        fit1 [ f1 ] = fitness ( vertx, crosd1 );
```

```
                result = check ( vertx, crosd2,adjm );
                satoda ( nupopu, crosd2, ++f1, vertx );
                if ( result == 1 )
                        status [ f1 ] = result;
                else
                        fault [ ++ k ] = f1;
                fit1 [ f1 ] = fitness ( vertx, crosd2 );
                f1++;
        }//end of loop
        f1--;
        modi_mut( k, fault, nupopu );
        selection( nupopu, fit1, f1 );
}//End of cross_modi_mut
//Function Modified Mutation.this function tries to make an invalid colaration to valid one using modified mutation
void modi_mut(int k,int fault[],int nupopu[][size])
{
        int counter=0;
        if(k>0)
        {
                for(i=1;i<=k;i++)
                {
                        m=fault[i];
                        result=0;
                        max=1;
                        datosa(temp,nupopu,m,vertx);
                        //starting of mutation using modified method


                        counter=0;
                        do
```

```
{
        for(j=1;j<vertx;j++) // matrix of confliction assg
        {
                for(l=j+1;l<vertx;l++)
                        if(temp[j]==temp[l])
                                if(adjm[j][l]==1)
                                {
                                        mat[j][l]=1;
                                        mat[l][j]=1;
                                        y++;
                                }
                ya[j]=y;
                if(ya[max]<y)
                        max=j;
                y=0;
        }
        do
        {
                nuclor=(rand()%last_color)+1;
        }while(temp[max]==nuclor);
        temp[max]=nuclor;
        result=check(vertx,temp,adjm);
        if(result==1)
        {
                mat[max][l]=0;
                mat[l][max]=0;
                ya[max]=0;
        }
        counter++;
        if(counter==vertx*4) //for fourceful exit from do while
        {
```

```
                                        break;
                                }
                        }while(result!=1); //until one valid coloration obtained
                        //Inserting the valid color into population
                        if(result==1)
                        {
                                satoda(nupopu,temp,m,vertx);
                                fit1[m]= fitness(vertx,temp);
                                status[m]=1;
                        }
                }
        }// end of if
        selection(nupopu,fit1,f1);
}// end of modi_mut
//Normal mutation
//this function performs single point simple mutation operation on chromosome
void norml_mut(int i, int j)
{
        int counter=0;
        result=0;
        datosa(temp,popu,i,vertx);
        do
        {
                nuclor=(rand()%last_color)+1;
        }while(temp[j]==nuclor);
        temp[j]=nuclor;
        result=check(vertx,temp,adjm);
        if(result!=1)
        {
                do
                {
```

```
        for (j = 1; j < vertx; j++)
        {
                for (l = j+1; l <= vertx; l++)
                        if (temp[j]==temp[l])
                                if (adjm[j][l]==1)
                                {
                                        mat[j][l]=1;
                                        mat[l][j]=1;
                                        y++;
                                }
                ya[j]=y;
                if(ya[max]<y)
                        max=j;
                y=0;
        }
        do
        {
                nuclor=(rand()%last_color)+1;
        }while(temp[max]==nuclor);
        temp[max]=nuclor;
        result=check(vertx,temp,adjm);
        if(result==1)
        {
                mat[max][l]=0;
                mat[l][max]=0;
                ya[max]=0;
        }
        counter++;
        if(counter>10)
                break;
}while (result!=1);
```

```
        }
        if(check(vertx,temp,adjm))
        {
                write_to_file();
                //printf("\n4th Fitness: %d\n",fitness(vertx,temp));
                //system("pause");
        }
        else
        {
                if(check(vertx,temp,adjm))
                {
                        satoda(popu,temp,popusize,vertx);
                        fit[popusize]=fitness(vertx,temp);
                }
        }
        selection(popu,fit,popusize);
} //End of normal mutation
//Special Mutation
//this function performs multi-point quided mutation on a chromosome
void special_mut(int ii, int jj)
{
        struct array
        {
                int clor; // array[1][]
                int nclor; //array[2][]
                int posclor[size/4];
        }tmpclr[size];
        int k,kk,i,j,t,flag=0,cnt=0,small=1000,smalst=1000;
        int smallc,smalstc,pos,tmpfit;
        int isit2, avoidclr, avoidclr1, avoidclr2, avoid_i;
        pos =ii;
```

```
result=0;
gcount=0;
whilecntr=1;
do
{
        for(k=1;k<=vertx;k++)
        {
                tmpclr[k].clor=0;
                tmpclr[k].nclor=0;
                for (j=1; j<=size/4;j++)
                        tmpclr[k].posclor[j]=0;
        }
        cnt=1;
        datosa(temp,popu,pos,vertx);
        tmpfit= fitness(vertx,temp);
        tmpclr[1].clor = temp[1];
        for(k=1;k<=vertx;k++)
        {
                flag=1;
                for(t=1;t<=vertx;t++)
                {
                        if(tmpclr[t].clor == temp[k])
                        {
                                tmpclr[t].nclor++;
                                tmpclr[t].posclor[tmpclr[t].nclor]=k;
                                flag=0;
                                break;
                        }
                        if (tmpclr[t].clor==0)
                                break;
                }
```

```
                if(flag==1 && tmpclr[t].clor==0)
                {
                        tmpclr[t].clor = temp[k];
                        tmpclr[t].nclor++;
                        tmpclr[t].posclor[tmpclr[t].nclor] =k;
                }
        }
        for (t=1;t<=vertx;t++)
        {
                if(tmpclr[t].clor==0)
                {
                        t--;
                        break;
                }
        }

//////////////////Decreasing two colors at once//////////////////////////
                //Decreasing first time
        do
        {
                i=(rand()%t)+1;
                k=(rand()%t)+1;
        }while(k==i);
        avoid_i=i;
        for(j=1;j<=tmpclr[k].nclor;j++)
                temp[tmpclr[k].posclor[j]] = tmpclr[i].clor;
                avoidclr1=k;    //end of decreasing no.of color
                //Decreasing no.of color_SECOND TIME
        do
        {
                i=( rand()% t )+ 1;
```

```
                k=( rand()% t )+ 1;
        }while (k==i || i== avoid_i ||k == avoidclr1);
        for (j=1; j<=tmpclr[k].nclor;j++)
                temp[tmpclr[k].posclor[j]] = tmpclr[i].clor;
        avoidclr2=k;    // end of decreasing no.of color
////////////////////end of decreasing two colors at once//////////////////////
        isit2=1;
        while (isit2<=2)
        {
                result= check(vertx,temp,adjm);
                if (result!=1)
                {
                        do
                        {
                                for(j=1;j< vertx;j++) //Matrix of confliction assignment
                                {
                                        for (l=j+1; l<=vertx;l++)
                                                if(temp[j]==temp[l])
                                                        if(adjm[j][l]==1) //checking present
                                                        {
                                                                mat[j][l]=1;
                                                                mat[l][j]=1;
                                                                y++;
                                                        }
                                        ya[j]=y;
                                        if(ya[max]<y)
                                                max=j;
                                        y=0;
                                }
                                tmpfit = fitness(vertx,temp);
                                do
```

```
                {
                        kk= (rand()%t)+1;
                        if(kk!= avoidclr1 && kk!= avoidclr2)
                                nuclor= tmpclr[kk].clor;
                }while(temp[max]== nuclor);
                temp[max]=nuclor;
                result = check(vertx,temp,adjm);
                if(result==1)
                {
                        mat[max][l]=0;
                        mat[l][max]=0;
                        ya[max]=0;
                        write_to_file();
                        //printf("\n5th Fitness: %d\n",fitness(vertx,temp));
                        //system("pause");
                }
                if(cnt> vertx*2)
                {
                        break;
                }
                cnt++;
        }while(result!=1);
}// end of if (result!=1)
if(result==1)
        break; //brek from while(isit2<=2)
datosa(temp,popu,pos,vertx);
tmpfit = fitness(vertx,temp);
for(j=1;j<=tmpclr[i].nclor;j++)
        temp[tmpclr[i].posclor[j]]= tmpclr[k].clor;
```

```
                avoidclr=1;
                isit2++;
                cnt=1;
            }// end of while(isit2<=2)
            if (result==1)
            {
                if(fitness(vertx,temp)<=mincol)
                {
                    satoda(popu,temp,1,vertx);
                    fit[1]= fitness(vertx,temp);
                    break;
                }
                else
                {
                    satoda(popu,temp,popusize,vertx);
                    fit[popusize]= fitness(vertx,temp);
                    selection(popu,fit,popusize);
                }
            }//end of if(result==1)
            whilecntr++;
        }while(whilecntr<t); //end of outer while
} // end of special mutation
//Function max_deg()
//this function calculates the maximum degree of the graph
int max_deg(int node)
{
        int v[size];
        int i,j,sum;
        int max; //to store maximim degree of the graph
        for(i=1; i<=node;i++)
        {
```

```
                sum=0;
                for (j=1;j<=node;j++)
                {
                        sum = sum + adjm[i][j];
                }
                v[i]=sum;
                adjm[i][node+1]= sum ;
        }
        max=0;
        for (i=1;i<=node;i++)
        {
                if(max< v[i])
                        max= v[i];
        }
        //printf("\n Maximum degree is %d",max);
        //printf("\n Maximum color requirement: %d", max +1);
        //scanf("%c", &c);
        return (max);
} // end of max_deg
//function check
//this function checks the validity of a coloration
int check(int n, int c[], int a[][size])
{
        int i,j;
        for(i=1; i<n; i++)
                for(j=i+1; j<=n; j++)
                {
                        if(c[i]==0 || c[j]==0)
                        {
                                printf("\n\t\t\tCaution:: value 0 entered into chromosme\n");
                                scanf("%c",c);
```

```
                        }
                        if(c[i]==c[j])
                                if(a[i][j]==1)
                                        return 0;
                }
        return 1;
}// end of function check()
//Function Fitness
//This function counts the no.of colors in a coloration
int fitness( int n, int d[])
{
        int i,j ,xy[size];
        int f=0;
        for(i=1; i<=n; i++)
                xy[i]=d[i];
        for(i=1; i<n; i++)
                for(j=i+1; j<=n; j++)
                        if(xy[i] == xy[j])
                                xy[j]=0;
        for(i=1; i<=n; i++)
                if(xy[i]!=0)
                        f++;
        return (f);
}// end of function fitness
//Selection
void selection(int npopu[size][size],int nfit[size],int nvertx)
{
        int i,j,temp,stemp[size];
        for(i=1;i<nvertx;i++)
                for(j=i+1;j<=nvertx;j++)
                        if(nfit[i]>nfit[j])
```

```
                        {
                                temp=nfit[i]; nfit[i]=nfit[j]; nfit[j]=temp;
                                datosa(stemp,npopu,i,vertx);  datoda(npopu,npopu,i,j,vertx);
                                satoda(npopu,stemp,j,vertx);
                        }
}//end of function
void write_to_file()
{
        int i,j;
        no_clr=fitness(vertx,temp);
        if(no_clr<=mincol+5)
        {
                if(last_color>no_clr)
                {
                        last_color=fitness(vertx,temp);
                        if((fout=fopen(file_out,"w"))==NULL)
                        {
                                printf("\nCant open the file!!");
                                exit(1);
                        }
                        for(j=1;j<=vertx;j++)
                        {
                                fprintf(fout,"node[%d]-> %d\n",j,temp[j]);
                        }
                        fprintf(fout,"\n\nColoration by Genetic Algorithm");
                        fprintf(fout,"\n\nIN GA NO OF COLORS USED: %d\n",no_clr);
                        fclose(fout);
                }
        }
} //End of function
void writetm_to_file(double tm)
```

```
{
        if((fout=fopen(file_out,"a"))==NULL)
        {
                printf("\nCant open the file!!"); exit(1);
        }
        fprintf(fout,"\nTIME TAKEN: %lf seconds\n",tm);  fclose(fout);
} //End of function
void datosa(int d[],int s[][size],int r,int c)
{
        int i;
        for(i=1;i<=c;i++)
                d[i]=s[r][i];
}
void satoda(int d[][size],int s[],int r,int c)
{
        int i;
        for(i=1;i<=c;i++)
                d[r][i]=s[i];
}
void satosa(int d[],int s[],int c)
{
        int i;
        for(i=1;i<=c;i++)
                d[i]=s[i];
}
void datoda(int d[][size],int s[][size],int dr,int sr,int vertx)
{
        int i;
        for(i=1;i<=vertx;i++)
                d[dr][i]=s[sr][i];
}
```