CS258

Design & Analysis of Algorithm

COURSE TIMETABLING

Analysis and implementation

PROJECT REPORT

Under the Guidance of

Dr. Kapil Ahuja

Department of Computer Science and Engineering

Indian Institute of Technology Indore

Spring 2019

NEERAJ VERMA
170005020

DHRUV SINGHAL
170001022

TABLE OF CONTENTS

INTRODUCTION

The aim of this project is to construct an efficient and optimal algorithm for Course Timetabling. Timetable construction problems are interesting objects to study because neither modelling nor solving them is straightforward. It is difficult to make a clear-cut distinction between acceptable and not acceptable timetables. Because of the large diversity in acceptance criteria, realistic timetable construction problems are multidimensional. Each dimension may introduce its own characteristic aspects that add to the complexity of the problem.

Brute Force approach for construction of the timetables might end up taking days or months to produce results. This project deals with highly efficient algorithms to get the desired results. We get to learn how a totally different problem can be converted to solve Course Timetabling. In the, end results are compared for the different algorithms that are included for the project to get an idea for the choice of solution to implement for a particular type of problem.

ALGORITHMS

1. **Genetic Algorithms:** These are adaptive heuristic search algorithm based on the evolutionary ideas of natural selection and genetics. As such they represent an intelligent exploitation of a random search used to solve optimization problems. Although randomised, GAs are by no means random, instead they exploit historical information to direct the search into the region of better performance within the search space. The basic techniques of the GAs are designed to simulate processes in natural systems necessary for evolution, especially those follow the principles first laid down by Charles Darwin of "survival of the fittest.". Competition among individuals for scanty resources results in the fittest individuals dominating over the weaker ones.

2. **Ant Colony Optimisation:** This is an algorithm for finding optimal paths that is based on the behaviour of ants searching for food. At first, the ants wander randomly. When an ant finds a source of food, it walks back to the colony leaving "markers" (pheromones) that show the path has food. When other ants come across the markers, they are likely to follow the path with a certain probability. If they do, they then populate the path with their own markers as they bring the food back. As more ants find the path, it gets stronger until there are a couple streams of ants traveling to various food sources near the colony.

THE PROBLEM

- We have to design a Timetable for a University or School.
- The timetable will have the following structure:

	PERIOD_1	PERIOD_2	. . . .	PERIOD_M
DAY_1	A	B		D
DAY_2	B	C		D
:				
DAY_N	A	A		E

- There will be Professors assigned to the Subjects. There may be one to man relationship from professors to subjects.
- Timetable will have following constraints: -
    - i Each subject will have maximum periods assigned in a week.
    - ii Each professor will have maximum periods assigned in a week.
    - iii The algorithm must try not to put two consecutive periods of a subject.
    - iv The algorithm must try not to put two consecutive periods for a professor.
- If all the constraints are not able to be satisfied then the algorithm must provide the most optimal solution.

IMPLEMENTATION

1. GENETIC ALGORITHM:
   a. Goal is to create a string of n*m characters.
   b. Each character denotes a single subject.
   c. This string must contain all the possible characters(subjects) with most of the constraints satisfied.
   d. Start with a pool of 100 random strings of the size equal to solution string.
   e. Calculate the fitness of each string.
   f. Sort according to the fitness value.
   g. Now these strings undergo mating (recombination+mutation) to produce new strings.
   h. Strings chosen for mating are based on their fitness level.
   i. So, the new strings generated will be more suited for the solution.
   j. Replace the initial population with the new strings.
   k. Repeat from ‘e’ until optimal solution is reached.

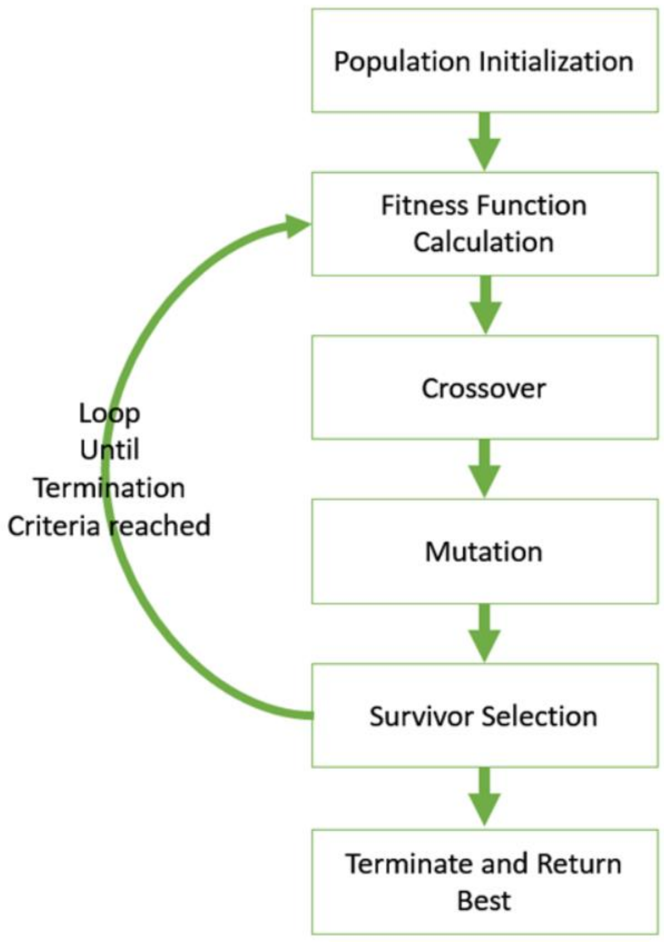
2. **ANT COLONY OPTIMISATION:**

a. We will convert the ‘Travelling Salesman Problem’ to Course Timetabling.
b. The most basic constraint i.e. number of periods assigned to each subject in a week will be solved in the first step.
c. We make n*m nodes which denote the cities.
d. Each node will denote a subject.
e. Multiple nodes can also denote a single subject.
f. Distance between these nodes will be decided as per the constraints.
g. We begin by initializing 5 ants.
h. They start moving in the random path from source to destination.
i. Their movement from node A to node B states that after the period of subject A next period will be of subject B.
j. While moving on the random paths, each ant will secrete some pheromone.
k. This pheromone is sensed by other ants over the time.
l. Smaller the path means higher the pheromone level means more ants on this path means most optimal solution.

BASIC STRUCTURE

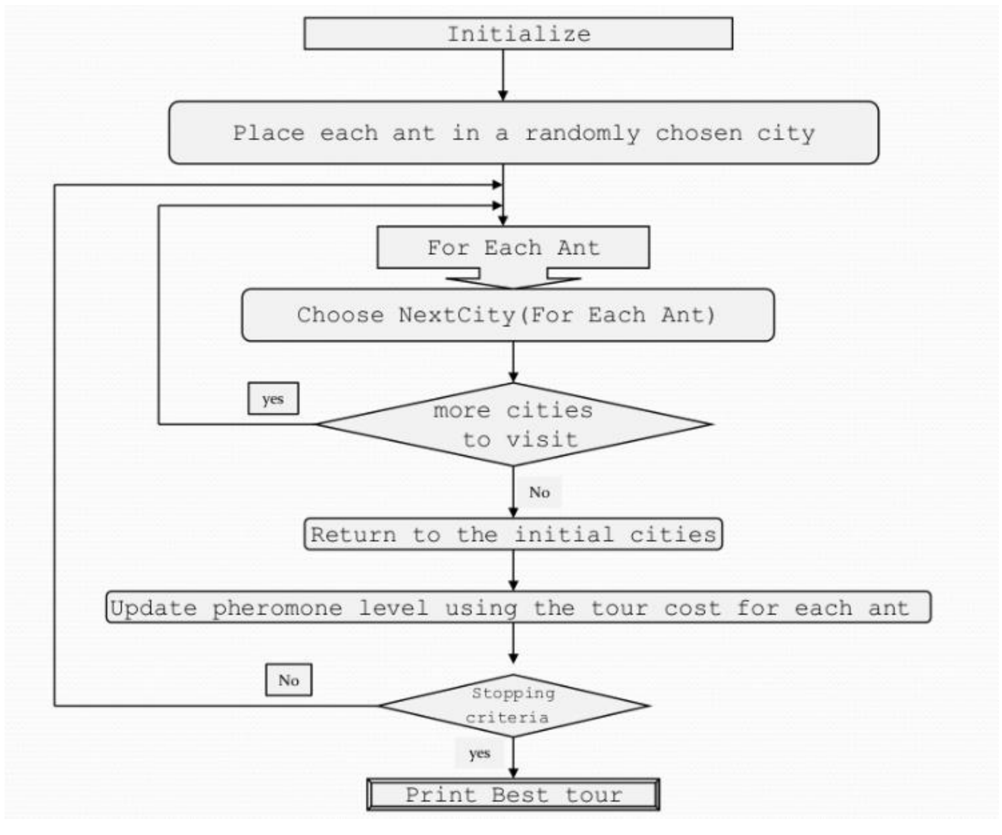
1. GENETIC ALGORITHM

    a. Start with an initial population (random/heuristic).
    b. Select parents for mating.
    c. Apply crossover and mutation on the parents to generate new off-springs.
    d. These off-springs replace the existing individuals in the population.


Population Initialization
Fitness Function Calculation
Crossover
Mutation
Survivor Selection
Terminate and Return Best
Loop Until Termination Criteria reached

2. ANT COLONY OPTIMISATION

a. Initialize the ants.
b. Allow them to move randomly from a source while leaving pheromones.
c. Virtual trail accumulated on the path segments.
d. Path selected at random based on the amount of “trail” present.
e. Ant reaches next node.
f. Continues until reaches starting node.
g. Finished tour is a solution.


Initialize
Place each ant in a randomly chosen city
For Each Ant
Choose NextCity(For Each Ant)
yes
more cities to visit
No
Return to the initial cities
Update pheromone level using the tour cost for each ant
No
Stopping criteria
yes
Print Best tour

DESIGN

1. GENETIC ALGORITHM

Generalised Pseudo Code:

```
GA()
   initialize population
   find fitness of population

   while (termination criteria is reached)
do
      parent selection
      crossover with probability pc
      mutation with probability pm
      decode and fitness calculation
      survivor selection
      find best
   return best
```

C++ Code :

```
#include<bits/stdc++.h>

using namespace std;

#define POPULATION_SIZE 100
#define SUBJECTS_SIZE 6
#define PROFESSORS_SIZE 3
#define PERIODS 6
#define DAYS 5

const string SUBJECTS="ABCDEF";
const string PROFESSORS="xyz";
const string MAP="yxxzyx";

const int SUB[SUBJECTS_SIZE]={7,3,6,4,5,5};
const int PROF[PROFESSORS_SIZE]={13,12,5};


int random_num(int start, int end) {
    int range=end-start+1;
    return start+(rand()%range);
}

char mutatedGene() {
    int len=SUBJECTS_SIZE;
    int r=random_num(0,len-1);

    return SUBJECTS[r];
}

string createGnome() {
    int len=PERIODS*DAYS;

    string gnome="";

    for(int i=0;i<len;i++) {
        gnome+=mutatedGene();
    }

    return gnome;
}

class Individual {
    public:

    string chromosome;
    int fitness;

    Individual(string chromo) {

        chromosome=chromo;
        fitness=calcFitness();
    }
```

```
Individual mate(Individual parent2) {
    string childChromosome;

    int len=PERIODS*DAYS;

    childChromosome="";

    for(int i=0;i<len;i++) {
        float p=random_num(0,100)/100;

        if(p<0.45)
            childChromosome+=chromosome[i];
        else if(p<0.90)
            childChromosome+=parent2.chromosome[i];
        else
            childChromosome+=mutatedGene();

    }

    return Individual(childChromosome);
}

int conflictSubHours() {
    int len=PERIODS*DAYS;

    int count=0,conflicts=0;

    for(int i=0;i<SUBJECTS_SIZE;i++) {
        count=0;

        char sub=SUBJECTS[i];

        for(int j=0;j<len;j++) {
            if(chromosome[j]==sub)
                count++;
        }

        conflicts+=abs(count-SUB[i]);
    }

    return conflicts;
}

int conflictSubCont() {
    int len=PERIODS*DAYS;

    int conflicts=0;

    for(int i=0;i<len-1;i++) {
        if(chromosome[i]==chromosome[i+1])
            conflicts++;
    }

    return conflicts;
}

int conflictProfHours() {
    int len=PERIODS*DAYS;
```

```
        int conflicts=0;

        for(int i=0;i<PROFESSORS_SIZE;i++) {
            int count=0;
            char prof=PROFESSORS[i];

            for(int j=0;j<len;j++) {
                if(prof==MAP[chromosome[j]-'A'])
                    count++;
            }

            conflicts+=abs(count-PROF[i]);
        }

        return conflicts;
    }

    int calcFitness() {

        int fitness=0;

        fitness+=conflictSubHours();
        fitness+=conflictSubCont();
        fitness+=conflictProfHours();

        return fitness;
    }
};

void display(vector<Individual> population) {
    cout<<"display";

    int len=PERIODS*DAYS;

    for(int i=0;i<len;i++) {
        if(i%PERIODS==0)
            cout<<"\n";

        cout<<"\t"<<population[0].chromosome[i];
    }

    cout<<"\n\n\tfitness                                :
"<<population[0].fitness<<endl;
}

bool  operator<(const  Individual  &ind1,  const  Individual
&ind2) {
    return ind1.fitness < ind2.fitness;
}

int main() {
    srand((unsigned)(time(0)));

    vector<Individual> population;
    bool found=false;

    for(int i=0;i<POPULATION_SIZE;i++) {
            string gnome=createGnome();
            population.push_back(Individual(gnome));
        }

        int generation=0,count=0;

        while(!found) {
            sort(population.begin(),population.end());

            if(population[0].fitness<=0 || count==500) {
                found=true;
                break;
            }

            vector <Individual>newGeneration;

            int s=(10*POPULATION_SIZE)/100;

            for(int i=0;i<s;i++) {
                newGeneration.push_back(population[i]);
            }

            s=(90*POPULATION_SIZE)/100;

            for(int i=0;i<s;i++) {
                int r=random_num(0,50);
                Individual parent1=population[r];
                r=random_num(0,50);
                Individual parent2=population[r];

                Individual offspring=parent1.mate(parent2);

                newGeneration.push_back(offspring);
            }

            sort(newGeneration.begin(), newGeneration.end());


    if(newGeneration[0].fitness==population[0].fitness)
                count++;
            else
                count=0;


            population=newGeneration;

            // display(population);
            generation++;
        }

        display(population);
    }
```

2. ANT COLONY OPTIMISATION

Generalised Pseudo Code:

```
Begin
   Initialize
   while stopping criterion not satisfied do
      position each ant at a node
      repeat
         for each ant do
            choose next node by applying the transition rules
            apply step by step pheromone update
         end for
      until every ant has built a solution
      update best solution
   end while
End
```

C++ Code :

```
#include <iostream>
#include <cstdio>
#include <cstdlib>
#include <algorithm>
#include <cfloat>
#include <cmath>
#include <fstream>


#define numCities 30
#define numAnts 5
#define maxTime 1000

//int numCities;
//int numAnts;

using namespace std;

const string SUBJECTS="ABCDEF";
const int SUB[]={7,3,6,4,5,5};
const string PROFESSORS="xyz";
const int PROF[]={14,12,4};
const string MAP="xxyzyx";

int alpha = 3;                   // influence of pheromone in
direction
int beta = 2;                    // influence of adjacent node
distance
double rho = 0.01;               // pheromone decrease factor
double Q = 2.0;                  // pheromone increase factor

int assign(int i, int j, int temp[]) {
     int d=0;

     int k,l;

     for(k=0;k<6;k++)
          if(i<temp[k])
               break;

     for(l=0;l<6;l++)
          if(j<temp[l])
               break;

     if(l==k)
          d=100;
     else
          d=1;

     if(MAP[l]==MAP[k])
          d+=50;
     else
          d+=1;
```

```
    return d;
}

void Graph(int dist[numCities][numCities])
{
    int temp[6];
    temp[0]=SUB[0];

    for(int i=1;i<6;i++) {
        temp[i]=temp[i-1]+SUB[i];
    }

    for(int i=0;i<numCities;i++) {
        for(int j=i;j<numCities;j++) {
            if(j==i)
                dist[i][j]=0;
            else {
                int d=assign(i,j,temp);

                dist[i][j]=d;
                dist[j][i]=d;
            }

        }
    }

}

void printants(int ants[][numCities])
{
    for (int i = 0; i < numAnts; i++) {
        for (int j = 0; j < numCities; j++) {
            cout << ants[i][j] << " ";
        }
        cout << endl;
    }
    cout << endl;
}


void printdist(int dist[][numCities])
{
    for (int i = 0; i < numCities; i++) {
        for (int j = 0; j < numCities; j++) {
            cout << dist[i][j] << " ";
        }
        cout << endl;
    }
    cout << endl;
}

void printpheromones(double pheromones[][numCities])
{
    for (int i = 0; i < numCities; i++) {
        for (int j = 0; j < numCities; j++) {
            cout << pheromones[i][j] << " ";
        }
        cout << endl;
```

```
     }
     cout << endl;
}

void InitPheromones(double pheromones[][numCities])
{
     for (int i = 0; i < numCities; i++) {
          for (int j = 0; j < numCities; j++) {
               pheromones[i][j] = 0.01;
          }
     }
}

int IndexOfTarget(int target, int trail[])
{
     int i;
     for (i = 0; i < numCities; i++) {
          if (trail[i] == target) return i;
     }
     cout << "Target not Found in IndexOfTarget:    " << i <<
endl;
}

void InitAnts(int ants[numAnts][numCities])
{
     for (int i = 0; i < numAnts; i++) {
          int start = rand() % numCities;
          int trail[numCities];
          for (int j = 0; j < numCities; j++) {
               trail[j] = j;
          }
          random_shuffle(trail, trail + numCities);
          int index = IndexOfTarget(start, trail);
          swap(trail[0], trail[index]);
          for (int j = 0; j < numCities; j++) {
               ants[i][j] = trail[j];
          }
     }
}

double Length(int x[], int dist[][numCities])
{
     double ans = 0.0;
     for (int i = 0; i < numCities - 1; i++) {
          ans += dist[x[i]][x[i + 1]];
     }
     return ans;
}

void ShowLength(int ants[][numCities], int dist[][numCities])
{
     for (int i = 0; i < numAnts; i++) {
          cout << i << ": [ ";
          for (int j = 0; j < 4; j++) {
               cout << ants[i][j] << " ";
          }
          cout << " . . . ";
          for (int j = numCities - 4; j < numCities; j++) {
                cout << ants[i][j] << " ";
            }
            cout << "] len = ";
            double len = Length(ants[i], dist);
            cout << len << endl;
        }
}

void BestTrail(int ants[][numCities], int dist[][numCities],
int bestTrail[])
{
        double bestlength = Length(ants[0], dist);
        int index = 0;
        for (int i = 1; i < numAnts; i++) {
            double len = Length(ants[i], dist);
            if (len < bestlength) {
                bestlength = len;
                index = i;
            }
        }

        for (int i = 0; i < numCities; i++) {
            bestTrail[i] = ants[index][i];
        }
}

void Probability(int k, int cityX, bool visited[], double
pheromones[][numCities], int dist[][numCities], double probs[])
{
        double tau[numCities];
        double sum = 0.0;
        for (int i = 0; i < numCities; i++) {
            if (i == cityX) {
                tau[i] = 0.0;
            } else if (visited[i] == true) {
                tau[i] = 0.0;
            } else {
                tau[i] = (pow(pheromones[cityX][i], alpha*1.0))
* (pow((1.0 / dist[cityX][i]*1.0), beta*1.0));
                if (tau[i] < 0.0001) {
                    tau[i] = 0.0001;
                } else if (tau[i] > DBL_MAX / numCities * 100) {
                    tau[i] = DBL_MAX / numCities * 100;
                }
            }
            sum += tau[i];
        }
        for (int i = 0; i < numCities; i++) {
            probs[i] = tau[i] / sum;
        }
}

int NextCity(int k, int cityX, bool visited[], double
pheromones[][numCities], int dist[][numCities])
{
        double probs[numCities];
        Probability(k, cityX, visited, pheromones, dist, probs);
        double cum[numCities + 1];
```

```
     cum[0] = 0.0;
     for (int i = 0; i < numCities; i++) {
          cum[i + 1] = cum[i] + probs[i];
     }
     double p = (double)rand() / (double)RAND_MAX;
     for (int i = 0; i < numCities; i++) {
          if (p >= cum[i] && p < cum[i + 1]) {
               return i;
          }
     }
     cout << "Failure\n";
}


void     BuildTrail(int     k,     int     start,     double
pheromones[][numCities], int dist[][numCities], int newTrail[])
{
     int trail[numCities];
     bool visited[numCities];
     for (int i = 0; i < numCities; i++) {
          visited[i] = false;
     }
     trail[0] = start;
     visited[start] = true;
     for (int i = 0; i < numCities - 1; i++) {
          int cityX = trail[i];
          int  next  =  NextCity(k,  cityX,  visited,  pheromones,
dist);
          trail[i + 1] = next;
          visited[next] = true;
     }
     for (int i = 0; i < numCities; i++) {
          newTrail[i] = trail[i];
     }
}

void      UpdateAnts(int      ants[][numCities],      double
pheromones[][numCities], int dist[][numCities])
{
     for (int i = 0; i < numAnts; i++) {
          int start = rand() % numCities;
          int newTrail[numCities];
          BuildTrail(i, start, pheromones, dist, newTrail);
          for (int j = 0; j < numCities; j++) {
               ants[i][j] = newTrail[j];
          }
     }
}

bool EdgePresentinTrail(int cityX, int cityY, int trail[])
{
     int lastIndex = numCities - 1;
     int index = IndexOfTarget(cityX, trail);
     if (index == 0 && trail[1] == cityY) return true;
        else if (index == 0 && trail[lastIndex] == cityY) return
true;
        else if (index == 0) return false;
        else if (index == lastIndex && trail[lastIndex - 1] ==
cityY) return true;
        else if (index == lastIndex && trail[0] == cityY) return
true;
        else if (index == lastIndex) return false;
        else if (trail[index - 1] == cityY) return true;
        else if (trail[index + 1] == cityY) return true;
        else return false;
}

void    UpdatePheromones(int    ants[][numCities],    double
pheromones[][numCities], int dist[][numCities])
{
     for (int i = 0; i < numCities; i++) {
          for (int j = i + 1; j < numCities; j++) {
               for (int k = 0; k < numAnts; k++) {
                    double length = Length(ants[k], dist);
                    double   decrease   =   (1.0   -   rho)   *
pheromones[i][j];
                    double increase = 0.0;
                    if  (EdgePresentinTrail(i,  j,  ants[k])  ==
true) {
                         increase = Q / length;
                    }
                    pheromones[i][j] = increase + decrease;
                    if (pheromones[i][j] < 0.0001) {
                         pheromones[i][j] = 0.0001;
                    } else if (pheromones[i][j] > 100000.0) {
                         pheromones[i][j] = 100000.0;
                    }
                    pheromones[j][i] = pheromones[i][j];
               }
          }
     }
}

void Display(int besttrail[])
{
     for (int i = 0; i < numCities; i++) {
          cout << besttrail[i] << " ";
          if (i > 0 && i % 20 == 0) {
               cout << endl;
          }
     }
     cout << endl;

     int temp[6];
     temp[0]=SUB[0];

     for(int i=1;i<6;i++) {
          temp[i]=temp[i-1]+SUB[i];

     }

     for(int i=0;i<numCities;i++) {
          int j;
          for(j=0;j<6;j++) {
               if(besttrail[i]<temp[j])
```

```
                    break;
          }

          if(i%6==0)
               cout<<"\n";

          cout<<"\t"<<SUBJECTS[j];
     }
}

int main()
{
     cout << "\nBegin Ant Colony Demo:\n\n";
     cout << "Number of cities in the problem: " << numCities
<< endl;
     cout << "Number of ants: " << numAnts << endl;
     cout << "Maximum Time: " << maxTime << endl;
     cout << "Alpha (pheromone influence): " << alpha << endl;
     cout << "Beta (local node influence): " << beta << endl;
     cout << "Rho (pheromone evaporation coefficient): " << rho
<< endl;
     cout << "Q (pheromone deposit factor) " << Q << endl;
     cout << "\nCreating Dummy Undirected Graph with Random edge
lengths:\n";

     int dist[numCities][numCities];
     Graph(dist);                                              //
Making Graph
     ofstream f;
     f.open("Output.txt");
     if (f.good()) {
          cout << "hello";
     }
     for (int i = 0; i < numCities; i++) {
          for (int j = 0; j < numCities; j++) {
               f << dist[i][j];
               f << " ";
          }
          f << endl;
     }

     cout << "\nInitialising ants to random trails:\n";     //
Initialising trails to ants
     int ants[numAnts][numCities];
     InitAnts(ants);
     ShowLength(ants, dist);

     int bestTrail[numCities];
     BestTrail(ants, dist, bestTrail);

     double bestlength = Length(bestTrail, dist);
     bestlength  +=  dist[bestTrail[0]][bestTrail[numCities  -
1]];

     cout << "\nBest Initial Trail Length: " << bestlength <<
endl << endl;

     cout << "\nInitialising Pheromones on trails: \n";
     double                  pheromones[numCities][numCities];
//Initialising Pheromones
     InitPheromones(pheromones);
     int time = 0;
     while (time < maxTime) {
          UpdateAnts(ants, pheromones, dist);
          UpdatePheromones(ants, pheromones, dist);
          int currBestTrail[numCities];
          BestTrail(ants, dist, currBestTrail);
          double currbestlength = Length(currBestTrail, dist);
          currbestlength                                      +=
dist[currBestTrail[0]][currBestTrail[numCities - 1]];

          if (currbestlength < bestlength) {
               bestlength = currbestlength;
               for (int i = 0; i < numCities; i++) {
                    bestTrail[i] = currBestTrail[i];
               }
               cout << "New Best Trail of: " << bestlength << "
found at time : " << time << endl;
          }
          ++time;
     }

     cout << "\nTime Complete:\n";
     cout << "\nBest Trail Found:\n";

     Display(bestTrail);

     cout << "\nLength of Best trail found: " << bestlength <<
endl;

     return 0;
}
```

ANALYSIS

1. GENETIC ALGORITHM

Following methods are used in the implementation:

i. mutatedGene()
ii. createGnome()
iii. mate()
iv. conflictSubHours()
v. conflictProfHours()
vi. conflictSubCount()
vii. calcFitness()
viii. main()

Variables:

- N : Subject size
- P : Professor size
- p : Period size
- d : Day size
- X : Stopping criteria (if unchanged after X iterations)
- F: Maximum fitness value
- Z : Population size

Complexity:
```
main()
     initialization  -> N*p*d*(S+P)
     while    -> runs at most FX times
    sort()   -> Z.log(Z)
    for each Individual   -> Z
       mate()     -> p*d
    end for
    offspring  ->p*d*(S+P)
     end while
end main
```

O(N*p*d*(S+P)+FX(Z.log(Z) +Z*p*d+p*d*(S+P)))

Z is mostly constant, hence, complexity depends upon fitness function.

Complexity of fitness function -> O(p*d*(S+P))

2. ANT COLONY OPTIMISATION

Following methods are used in the implementation:

i. graph()
ii. initPheromones()
iii. indexofTarget()
iv. initAnts()
v. length()
vi. bestTrail()
vii. probability()
viii. buildTrail()
ix. updatePheromones()

Variables:

- N : Number of cities
- M : number of ants
- T : maxtime

Complexity:

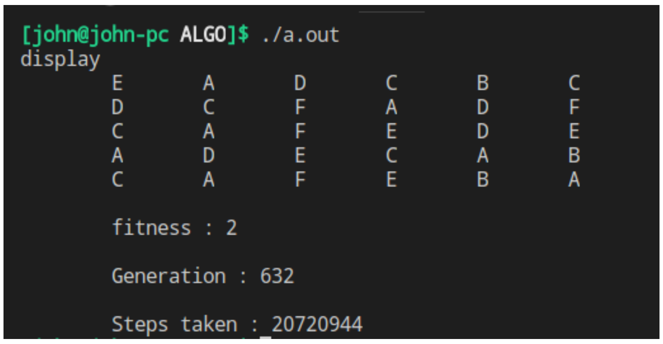
```
main()
     graph()   -> $N^2$
     while   -> at most T
    updateAnts()    ->  M*N
    updatePheromones() -> $M*N^2$
    bestTrail()    -> M+N
end main
```

$O(N^2 + T(M*N+M* N^2 + (M+N))$

Therefore, **overall complexity is $O(T*M* N^2)$.**

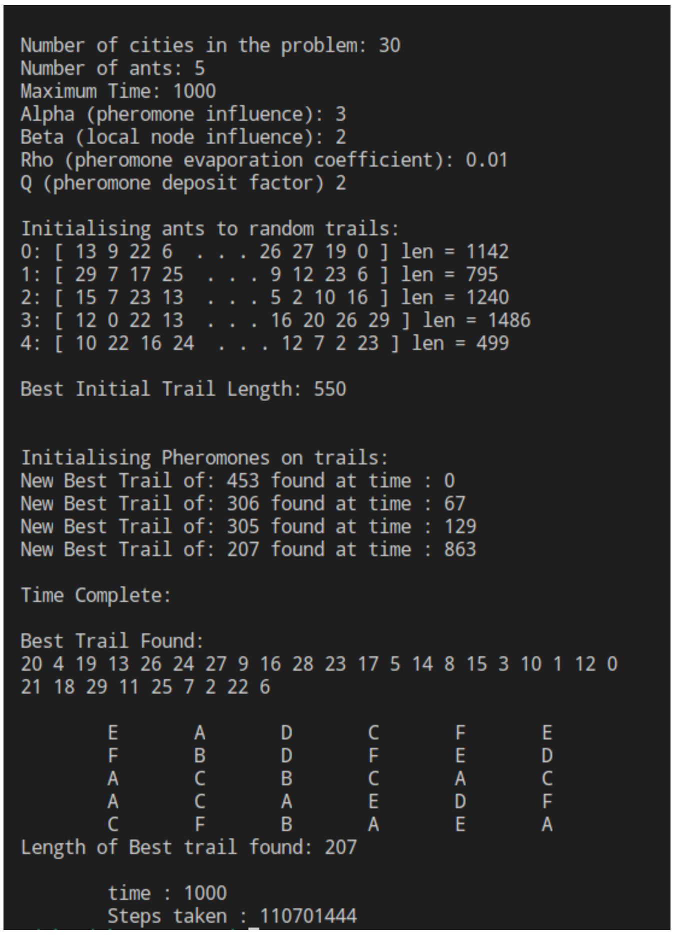
RESULT

Comparison for same set of input in Genetic Algorithm & Ant Colony Optimisation:



```
[john@john-pc ALGO]$ ./a.out
display
        E       A       D       C       B       C
        D       C       F       A       D       F
        C       A       F       E       D       E
        A       D       E       C       A       B
        C       A       F       E       B       A

        fitness : 2

        Generation : 632

        Steps taken : 20720944
```

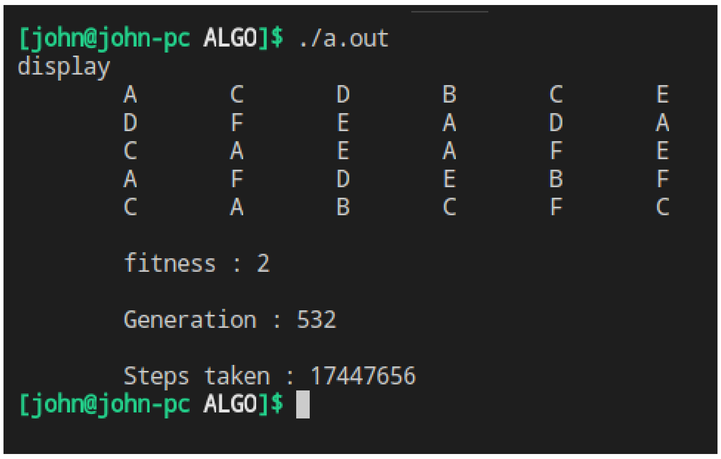
Genetic Algorithm : 2,07,20,944 steps



```
Number of cities in the problem: 30
Number of ants: 5
Maximum Time: 1000
Alpha (pheromone influence): 3
Beta (local node influence): 2
Rho (pheromone evaporation coefficient): 0.01
Q (pheromone deposit factor) 2

Initialising ants to random trails:
0: [ 13 9 22 6  . . . 26 27 19 0 ] len = 1142
1: [ 29 7 17 25  . . . 9 12 23 6 ] len = 795
2: [ 15 7 23 13  . . . 5 2 10 16 ] len = 1240
3: [ 12 0 22 13  . . . 16 20 26 29 ] len = 1486
4: [ 10 22 16 24  . . . 12 7 2 23 ] len = 499

Best Initial Trail Length: 550


Initialising Pheromones on trails:
New Best Trail of: 453 found at time : 0
New Best Trail of: 306 found at time : 67
New Best Trail of: 305 found at time : 129
New Best Trail of: 207 found at time : 863

Time Complete:

Best Trail Found:
20 4 19 13 26 24 27 9 16 28 23 17 5 14 8 15 3 10 1 12 0
21 18 29 11 25 7 2 22 6

        E       A       D       C       F       E
        F       B       D       F       E       D
        A       C       B       C       A       C
        A       C       A       E       D       F
        C       F       B       A       E       A
Length of Best trail found: 207

        time : 1000
        Steps taken : 110701444
```

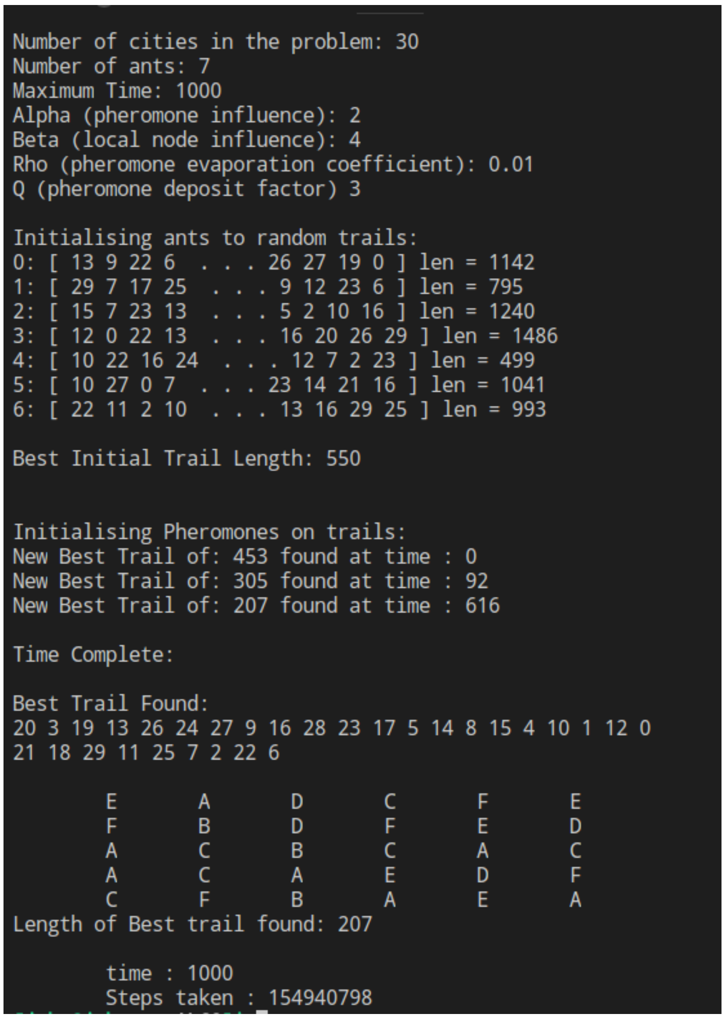
Ant Colony Optimisation :

11,07,01,444 steps



```
[john@john-pc ALGO]$ ./a.out
display
         A        C        D        B        C        E
         D        F        E        A        D        A
         C        A        E        A        F        E
         A        F        D        E        B        F
         C        A        B        C        F        C

         fitness : 2

         Generation : 532

         Steps taken : 17447656
[john@john-pc ALGO]$
```

Genetic Algorithm : 1,74,47,656 steps



```
Number of cities in the problem: 30
Number of ants: 7
Maximum Time: 1000
Alpha (pheromone influence): 2
Beta (local node influence): 4
Rho (pheromone evaporation coefficient): 0.01
Q (pheromone deposit factor) 3

Initialising ants to random trails:
0: [ 13 9 22 6  . . . 26 27 19 0 ] len = 1142
1: [ 29 7 17 25  . . . 9 12 23 6 ] len = 795
2: [ 15 7 23 13  . . . 5 2 10 16 ] len = 1240
3: [ 12 0 22 13  . . . 16 20 26 29 ] len = 1486
4: [ 10 22 16 24  . . . 12 7 2 23 ] len = 499
5: [ 10 27 0 7  . . . 23 14 21 16 ] len = 1041
6: [ 22 11 2 10  . . . 13 16 29 25 ] len = 993

Best Initial Trail Length: 550


Initialising Pheromones on trails:
New Best Trail of: 453 found at time : 0
New Best Trail of: 305 found at time : 92
New Best Trail of: 207 found at time : 616

Time Complete:

Best Trail Found:
20 3 19 13 26 24 27 9 16 28 23 17 5 14 8 15 4 10 1 12 0
21 18 29 11 25 7 2 22 6

        E       A       D       C       F       E
        F       B       D       F       E       D
        A       C       B       C       A       C
        A       C       A       E       D       F
        C       F       B       A       E       A
Length of Best trail found: 207

        time : 1000
        Steps taken : 154940798
```

Ant Colony Optimisation :

15,49,40,798 steps

CONCLUSION & FUTURE WORK

According to the above results, with the given constraints, Genetic algorithms prove to better than Ant Colony Optimisation.

We can the algorithms against more strict constraints like Number of rooms, Addition of semesters, etc.

Some other algorithms like Bee Colony Algorithms have also been used to design Course timetable.

REFERENCES

https://www.tutorialspoint.com/genetic_algorithms/genetic_algorithms_introduction.htm

https://www.geeksforgeeks.org/genetic-algorithms/

https://ieeexplore.ieee.org/document/4129846

http://www.scholarpedia.org/article/Ant_colony_optimization