# Pseudo code

**INPUTS:**

nPeriodsPerWeek, nteachers, nRooms

nLabSubjects, labslots, nLabs, firstLabRoom, lastLabRoom

populationsize, generationlimit

tournamentsize, tempint

mutationrate

elitism, crossoversplit, labCrossverSplit

teachers, labTeachers

teacherid, labTeacherid,

MAX\_PERIODS\_PER\_WEEK

MAX\_ROOMS

MAX\_TEACHERS

POSITIVE\_INFINITY

EMPTY

initial[MAX\_ROOMS][MAX\_PERIODS\_PER\_WEEK], labInitial[MAX\_ROOMS][MAX\_PERIODS\_PER\_WEEK], availability[MAX\_TEACHERS][MAX\_PERIODS\_PER\_WEEK], periodcount[MAX\_ROOMS][MAX\_TEACHERS], labPeriodcount[MAX\_ROOMS][MAX\_TEACHERS], conflicts[MAX\_TEACHERS][MAX\_TEACHERS],

//individual class for populating

**class individual**

Start

public: table[MAX\_ROOMS][MAX\_PERIODS\_PER\_WEEK]

fitness

individual ()

fitness = 0

End individual

individual elite,

<individual> population

**Algorithm :**

//function to return teacher id

**return\_tID( labSubj)**

Let tid is the teacher id number

for m = 0 to labSubj.size

if labSubj[m] == '/'

tid = labSubj.substr(0,m)

return tid

End return\_tID

//function to return room number for lab subject

**return\_roomNo( labSubj)**

let room is the room number to be allocates

for m = 0 to labSubj.size

if labSubj[m] == 'r'

room = stringtointeger(labSubj.substr(m+1))

return room

End

//function to generate random integer

**randomint( lower, upper)**

srand(time(0)+randomoffset)

randomoffset = (randomoffset+1)%2823401239LL

if upper<lower

return lower

return rand()%(upper-lower+1)+lower

End

//function for checking constraints

**randombool(chance)**

If randomint(0,1000000) < (1000000\*chance)

return true

else

return false

//function to get minimum fitness id

**getminfitnessid()**

let minvalue = POSITIVE\_INFINITY

let minid = 0, count = 0

let kteacher, lteacher, ktid, ltid are strings

let n = 0

let kroom, lroom

let tempfitness = 0, first2Hours = 0, confAvail = 0, oneLabperday = 0

for i = 0 to population.size

tempfitness = 0

first2Hours = 0

confAvail = 0

oneLabperday = 0

for j = 0 to labslots //calculate conflicts with fixed slots in initial

for k = 0 to nLabs

if population[i].table[k][j] != EMPTY

room = return\_roomNo(labTeachers[population[i].table[k][j]])

if initial[room-1][2\*j] != EMPTY || initial[room-1][2\*j+1] !=EMPTY

// subjective to this this slot system with 6 hours a day and 2 hour labs

confAvail++ //calculate conflicts with fixed slots in initial

let count = 0

//calculate conflicts within lab classes

For j = 0 to labslots

If j%(labslots/5) == 0

count += 1

for k = 0 to nLabs

if population[i].table[k][j] == EMPTY

continue

else

kteacher = labTeachers[population[i].table[k][j]]

kroom = return\_roomNo(kteacher)

ktid = return\_tID(kteacher)

for n = j+1 to count\*(labslots/5)

for l = 0 to nLabs

if population[i].table[l][n] == EMPTY

continue

else

lteacher = labTeachers[population[i].table[l][n]]

lroom = return\_roomNo(lteacher)

lid = return\_tID(lteacher)

if kroom == lroom

//checking for one lab/day for a teacher as well as a classroom

oneLabperday += 1

If ktid.compare(ltid) == 0

oneLabperday += 1

for j = 0 to labslots

for k = 0 to nLabs

for l = k+1 to nLabs

if conflicts[population[i].table[k][j]][population[i].table[l][j]] != 0

confAvail += 1 /\* Conflict checking for teachers and corrresponding rooms called to the lab room \*/

let firstPeriod, secondPeriod

for m = 0 to nLabs

for n = 0 to 5 //5 referring to no of days in a week

firstPeriod = n\*labslots/5

secondPeriod = n\*labslots/5+1

if population[i].table[m][firstPeriod] == EMPTY

first2Hours += 1

if population[i].table[m][secondPeriod] == EMPTY

first2Hours += 1

tempfitness = 0.8\*confAvail + 0.05\*first2Hours + 0.15\*oneLabperday

population[i].fitness = tempfitness

if tempfitness < minvalue

minvalue = tempfitness

minid = i

return minid

End

**tournamentselection()**

let tournamentminfitness = POSITIVE\_INFINITY

Let tournamentwinnerid = 0

Let tempint a temporary number

For i = 0 upto tournamentsize

tempint = randomint(0,population.size()-1)

if population[tempint].fitness < tournamentminfitness

tournamentminfitness = population[tempint].fitness

tournamentwinnerid = tempint

return tournamentwinnerid

End

//Function for individual crossing over

**crossover(a, b)**

let offspring is an individual

for i = 0 upto nLabs

let weekperiod is a integer vector

for j = 0 upto labslots

if labInitial[i][j] == EMPTY

{

weekperiod.push\_back(population[b].table[i][j])

for j = 0 upto labslots

if labInitial[i][j] != EMPTY

offspring.table[i][j] = initial[i][j]

else

if j < labCrossverSplit

offspring.table[i][j] = population[a].table[i][j]

weekperiod.erase(find(weekperiod.begin(),weekperiod.end(),offspring.table[i][j]))

else

offspring.table[i][j] = weekperiod[0]

weekperiod.erase(weekperiod.begin()

return offspring

End

**Genetic\_Algorithm()**

Let elapsedgenerations = 0

Let elitismoffset = 0

if(elitism)

elitismoffset = 1 //can make elitism offset a variable

while(elapsedgenerations < generationlimit)

let vector <individual> newpopulation

//compute fitness, find minimum

Let minid = getminfitnessid()

Let minvalue = population[minid].fitness

if(elitism)

newpopulation.push\_back(population[minid])

//crossover

For i = elitismoffset upto population.size

let a = tournamentselection()

let b = tournamentselection()

let individual offspring = crossover(a,b)

newpopulation.push\_back(offspring)

//mutate

For i = elitismoffset upto population.size

for( j = 0 to nLabs )

if(randombool(mutationrate))

let a, b

do

{

a = randomint(0,labslots-1)

b = randomint(0,labslots-1)

} while((initial[j][a]!=EMPTY) || (initial[j][b]!=EMPTY))

swap(newpopulation[i].table[j][a],newpopulation[i].table[j][b])

population = newpopulation

elapsedgenerations++

minid = getminfitnessid()

End