LOVELY
PROFESSIONAL
UNIVERSITY

Intelligent Time-Table Preparation

Submitted To

Nikita Kaushik

Department of

computer Science And Engineering

Submitted By

Arun Yadav 11811135

Laksshay Bhateja 11811155

Neeraj Mishra 11810989

Content

Abstract

Timetable problem is a NP-hard problem where different constraints and various resources are applied but the resources are limited. Optimization problem is a technique which can handle different constraints. This paper focuses the Bee colony Optimization (BCO) for finding the optimal solutions of course time table.BCO is a Meta heuristic optimization scheme where NP-hard with different parameter settings are solved. There are two objectives, first objective is to provide the introduction to timetabling and second objective is the BCO and their variations with timetable design. The proposed algorithm is used to construct

the course time table and optimized that time table.

Introduction

The preparation time table manually in schools, colleges and universities is very time consuming and tedious job which requires lots of effort as we have to look after various constraints and criteria. Also proper use of resources is neither effective nor efficient by using this approach. In order to overcome all these problems and to produce a satisfactory result we propose to make an automated system which will generate timetable automatically. The system will take various inputs like number of subjects, number of teachers, subject limits of each teacher, preference value for each subject given by each teacher, etc. By taking the help of above all these inputs it will generate possible time tables making optimal use of all resources in a way that will best suit the constraints. In 1996 Wren defines the timetable as the allocation of subject teachers, subject to different constraints, with various resources to objects are being placed in space time. It also satisfies a possible set of desirable objectives, as a result, a timetable specifies at which location and what time the teacher is allocated. The timetable must satisfies a number of requirements and also

satisfy the desires of all people as possible. In a college, there are different courses are available, so there is no conflict of free timeslots available for every student within that time. Therefore teacher tries to find the timetable with the minimum conflicts [9]. An appropriate timetable is then chosen from the optimal solutions generated. Timetable is the task of creating a timetable while satisfying various constraints. Bee Colony Optimization (BCO) is very useful in designing the optimized time table where less onflict arises. This paper is organized as follows: Section-2 illustrated to literature survey and research in this specific area.Section-3 described bee colony optimization.Section-4 explained the proposed approach and working of proposed approach.Section-5 illustrated the result and discussion and Section-6 described the conclusion and future scope.

Literature review

Sophia described the timetable construction which satisfies all operational rules in an academic institution, at the same time timetable fulfills the wishes and requirements of the faculty members and the students. It is an important and difficult task for the staff those are involved. Generally, this task is left to the administrative staff to replicate the

timetables of previous years with little changes to accommodate new situations .Adriano Denise compared the PSO to Genetic Algorithm (GA) in generating lecturer timetable schedule. Based on the computational results, the amount of penalty obtained by the PSO is much smaller than the GA on 500th iteration. Fen Irene proposed University Course Timetabling Planning UCTP) through hybrid particle swarm optimization with constraint-based reasoning (PSO-CBR). This algorithm is to allocate lessons in a weekly timetable, such that all students can attend all their events (lessons) without having to attend two events at the same time (called a student clashed). According to Emilio Fortunato the objective function derivative is needed for the initial position to be set by PSO. It also sets the feasibility of the initial position of the particles. Elizabeth described such as the appearance of the new lectures and exams during the semester which more difficult to handle. So Shu-Chuan focused the discrete PSO algorithm which is used to schedule exam timetable. Some soft constraints, such as preferences have to be handled. A penalty will subtract the optimal value on every single violation of the constraints. Betar and Khader focused on the university timetabling problem through harmony search method. It generates the near optimal solution. According to Lai various artificial intelligence techniques are used for complex course time table generation. Bhaduri, proposed several studies in the field of timetabling by using operational research, artificial intelligence and computational intelligence. Paulus explained

that the code needs some mapping, from PSO to timetable and vice versa. This mapping works well. The effectiveness of the solution is relatively low since it is solved by ordinary PSO. Lastly, this paper attempts to solve many problems faced by administrative staff, such as handling preferences as it may vary in every semester.

Proposed methodology

In order to study the computational effort involved in solving the timetable generation problem through BCO, the following mathematical programming model is proposed. We define the following sets to be used in the proposed model: tno – total number of teachers available. sno – total number of subjects to be taught. pno- maximum number of subject preferences that a teacher can provide as his/her options. tlim-an array which gives information about maximum number of subjects that a teacher can teach For example, a faculty member might take more classes than a senior professor. So tlim value for faculty member might be higher than that of a senior professor. The value of tlim cannot be zero or negative. smat- a 2-D array used to store preference matrix. Row represents the total number of subjects and Column represents the total number of teachers available. The fitness function value of each solution is given by

For j=1 to sno

Fx(i)=Fx(i)+Pref(i,j)*Prob(i,j) End For

Where Fx(i) - denote fitness function value of candidate solution number 'i'

Pref(i,j) –denote preference value of teacher for that particular subject

Prob(i,j)-denote probability of selecting a particular teacher and can be calculated as

prob(i,j)=1/tlim(i,j)

For our proposed problem, we have to maximize the fitness function value to get the optimal result. Initially, a set of candidate solutions is generated which satisfies all the required constraints. Then their corresponding fitness function values are calculated and the initial best solution is memorized. In each iteration, the solutions undergoes through Employed Bee Phase and Onlooker Bee Phase. In Employed Bee Phase, any two slots of the candidate solution are chosen randomly and replaced with new random values. Then any one slot chosen randomly from the candidate solution is replaced with the value of that slot position of the current best solution. If the fitness function value of the new solution is better than the current solution then it is replaced. After Employed Bee Phase is completed, it undergoes Onlooker Bee Phase. But firstly relative fitness function value of each candidate solution is calculated. If any candidate solution is having relative fitness function value less a constant "pa" then that solution undergoes alteration by randomly

replacing a slot in the candidate solution. Usually, the value of pa lies in between 0 and 1. In this case, the value of pa is taken as 0.1. If the fitness function value of the new solution is better than the current solution then it is replaced. At the end of the iteration, the best solution is calculated and memorized. The flow chart of time table generation by using Bee Colony Optimization algorithm is depicted

∴ Algorithm :-

if (daycount > n)
end
generate()
Lb12 for day_clash_element 1 today clash element n
Retrieve Sdt from dayclash
Retrieve tdt of Sdt
rehabilitate(Sdt)
for (ts, to tsn)
if (Si exists in final_tt)
Next iteration
else
Lb13: Retrieve Si in tsi
Retrieve fi of Si
if (availability = 0 for tsi)
rehabilitate(Si)
Else
if (Ki > 0)
Set Si to tsi in final_tt
tsfi = 1
Ki - -
Else
Lb13
if (tsn has been reached)
if (clash NOT Empty)
for clash_cln to clash_cln
Retrieve each Si in clash_cln
Retrieve Si with fi
rehabilitate(Si)

Else
Daycount ++
Else
continue
generate()
for(each SubjectSi)
Place Si in sub_arr
for(each dow)
rand_seq = rand(sub_arr)
if(dow = 1)
init_tt(dow) = rand_seq
else
curr_Pos = length(rand_seq)
temp_ele = rand_seq(last)
for(each element in rand_seq)
if(curr_Pos = 1)
rand_seq(curr_POS) = temp_ele
else
randiseq(curr_POS) = rand-Seq
(curr_Pos - 1)
init_tt(dow) = rand-seq.
End generate.

Result and discussion

```
import random
from data import *
from collections import deque
import os
import sys
import logging
day_row_num = {
        'monday': 0,
        'tuesday': 1,
        'wednesday': 2,
        'thursday': 3,
        'friday': 4,
        'saturday': 5
}
def getfreeslots(tt):
        freeslots = []
        nonfinalslots = []
        for day in tt:
```

```
                for timeslot in range(1, 6+1):
                        if timeslot in tt[day]:
                                if tt[day][timeslot] == '' and tt.final[day][timeslot] != True: # time slot is free
                                        freeslots.append((day, timeslot))
                                elif tt.final[day][timeslot] != True:
                                        nonfinalslots.append((day, timeslot))
        for day in tt:
                for timeslot in range(7, 8+1):
                        if timeslot in tt[day]:
                                if tt[day][timeslot] == '' and tt.final[day][timeslot] != True: # time slot is free
                                        freeslots.append((day, timeslot))
                                elif tt.final[day][timeslot] != True:
                                        nonfinalslots.append((day, timeslot))
        return freeslots, nonfinalslots
def getnumhours(tt, subject, day):
        # return the number of hours of subject on a given day, and a list of those hours
        num = 0
        hours = []
        for timeslot in tt[day]:
                if tt[day][timeslot] != '' and subject[2] == tt[day][timeslot][2] and subject[3] == tt[day][timeslot][3]:
                        num += 1
                        hours.append(timeslot)
        return num, hours
        def is_consecutive_hour(teacher, d, h):
        previous = teacher[d][h-1] if h > 1 else None
        if (d == 'saturday' and h < 4) or (d != 'saturday' and h < 8):
                next = teacher[d][h+1]
        else:
                next = None
```

```
        if not previous and not next:
                return False
        elif previous and next:
                return True
        elif previous and h == 3:
                return False
        elif next and h == 2:
                return False
        else:
                return True
def generate(tt, subjects, faculty, location): # subjects is a list of tuples (name, hours/week, teacher, short name); faculty is a dict
        remaining_hours = [i[1] for i in subjects]
        freeslots, _ = getfreeslots(tt)
        logger = logging.getLogger('tt_algo')
        logging.basicConfig(filename = location, level = logging.DEBUG)
        logger.info('')
for _ in range(max(remaining_hours)): # repeat for as many times as the max credits
                for i in range(len(subjects)): # iterate through all subjects, allot 1 hour for each subject
                        if remaining_hours[i] > 0:
                                subject = subjects[i]
                                for day, time in freeslots:
                                        if faculty[subject[2]].final[day][time] != True and getnumhours(tt, subject, day)[0] < 1: # if teacher slot is not finalized and subject isnt already there that day
                                                tt[day][time] = subject
                                                faculty[subject[2]][day][time] = (tt.name, subject)
                                                remaining_hours[i] -= 1
                                                freeslots.remove((day, time))
                                                break
                                else:
```

```
                                        logger.info("no free slots for %s %s", tt.name, subject)
                                        dayclash.append((tt, subject))
        return tt
def print_timetable(tt, location, style = 'section', name = ''):
        logger = logging.getLogger('tt_algo')
        logging.basicConfig(filename = location, level = logging.DEBUG)
        if name == '':
                name = tt.name
        logger.info(('%-20s ' * 9) % (name, '9:00-9:55', '9:55-10:50', '11:10-12:05', '12:05-1:00', '1:00-1:55', '1:55-2:50',
'2:50-3:40', '3:40-4:30'))
        for day in tt:
                x = '%-20s' % day
                for timeslot in tt[day]:
                        if tt[day][timeslot] == '':
                                x += ' ' + '%-20s' % '-'
                        else:
                                if style == 'section':
                                        x += ' ' + '%-20s' % tt[day][timeslot][3]
                                else:
                                        section = tt[day][timeslot][0]
                                        subject = tt[day][timeslot][1][3]
                                        x += ' ' + '%-20s' % (subject + ' (' + section + ')')
                logger.info('%s', x)
        logger.info('\n')
def rehabilitate(day, section, subject, faculty):
        teacher = faculty[subject[2]]
        for timeslot in teacher[day]:
                if teacher[day][timeslot] == '' and teacher.final[day][timeslot] != True and not is_consecutive_hour(teacher,
day, timeslot) and getnumhours(section, subject, day)[0] < 2: # teacher is available
                        if section.final[day][timeslot] != True: # time slot for that section is not finalized
```

```
                                        clashing_subject = section[day][timeslot]
                                        if teacher.workload > section.final[day][timeslot]: # whatever subject has been allotted,
move it to clash
                                                if clashing_subject != '':
                                                        clash.append((section, clashing_subject))
                                                        clashing_teacher = clashing_subject[2]
                                                        faculty[clashing_teacher][day][timeslot] = ''
                                                section.final[day][timeslot] = teacher.workload # finalize the lecture by
moving the new subject into the time slot
                                                section[day][timeslot] = subject
                                                teacher[day][timeslot] = (section.name, subject)
                                                break
                else: # teacher has no free time slots where the lecture can be scheduled, on that day
                        dayclash.append((section, subject))


def utilize_free_hours(tt, faculty):
        for day in tt:
                timeslots = tt[day].keys()
                t1 = min(timeslots)
                t2 = max(timeslots)
                for t in range(t1, t2+1): # t = current timeslot that is blank
                        if tt[day][t] == '' and tt.final[day][t] == False:
                                for i in range(t2, t, -1): # i = future timeslot that has a lecture
                                        if tt[day][i] != '' and tt.final[day][i] != True:
                                                subject = tt[day][i]
                                                teacher = faculty[subject[2]]
                                                if teacher[day][t] == '' and teacher.final[day][t] == False and not
is_consecutive_hour(teacher, day, t):
                                                        # if teacher is free at timeslot t, move subject from
timeslot i to t
```

```
                                                                tt[day][t] = subject
                                                                tt[day][i] = ''
                                                                tt.final[day][t] = tt.final[day][i]
                                                                tt.final[day][i] = False
                                                                teacher[day][i] = ''
                                                                teacher[day][t] = (tt.name, subject)
                                                                break # done filling timeslot t
        pass
def adjust_clash(timetables, location, faculty):
        logger = logging.getLogger('tt_algo')
        logging.basicConfig(filename = location, level = logging.DEBUG)
        for day in ['monday', 'tuesday', 'wednesday', 'thursday', 'friday', 'saturday']:
                n_dayclash = len(dayclash)
                for i in range(n_dayclash):
                        dayclash_ele = dayclash.popleft()
                        rehabilitate(day, dayclash_ele[0], dayclash_ele[1], faculty)

                timeslots = 8 if day != 'saturday' else 4
                for timeslot in range(1, timeslots+1):
                        for sem in timetables:
                                for section in timetables[sem]:
                                        section = timetables[sem][section]
                                        if section[day][timeslot] != '' and section.final[day][timeslot] == False: # if
lecture has not been finalized
                                                subject = section[day][timeslot]
                                                teacher = faculty[subject[2]]
        try:
                                                if teacher[day][timeslot] == '' and
teacher.final[day][timeslot] != True and not is_consecutive_hour(teacher, day, timeslot):
                                                        teacher[day][timeslot] = (section.name, subject)
```

```
                                                    section.final[day][timeslot] = teacher.workload
                                                elif teacher[day][timeslot] != '' and
(teacher[day][timeslot][0] != section.name or teacher[day][timeslot][1] != subject): # teacher is not available, there is a clash
                                                    # teacher is not available
                                                    # what if teacher takes 2 subjects for same
section? 2nd condition takes care of this
                                                    section[day][timeslot] = ''
                                                    section.final[day][timeslot] = False
                                                    rehabilitate(day, section, subject, faculty)
                                                elif is_consecutive_hour(teacher, day, timeslot):
                                                    section[day][timeslot] = ''
                                                    section.final[day][timeslot] = False
                                                    teacher[day][timeslot] = ''
                                                    rehabilitate(day, section, subject, faculty)
                                                else: # if teacher is available, finalize the lecture
                                                    section.final[day][timeslot] = teacher.workload
                                            except Exception as e:
                                                logger.info('%s %s %s %s %s', section.name, subject, day,
timeslot, faculty[subject[2]][day][timeslot])
                                                print_timetable(faculty[subject[2]], location, style = 'staff')
                                                print_timetable(section, location)
                                                logger.exception(e)
                                                raise
                    while len(clash) > 0:
                        clash_ele = clash.popleft()
                        rehabilitate(day, clash_ele[0], clash_ele[1], faculty)

def finalize_lab(section, day, time, subject, hours = 3):
    global faculty
    for i in range(time, time+hours):
```

```
                section[day][i] = subject
                section.final[day][i] = True
                for teacher in subject[2]:
                        faculty[teacher][day][i] = (section.name, subject)
                        faculty[teacher].final[day][i] = True


def finalize_theory(section, day, time, subject, hours = 1):
        global faculty
        for i in range(time, time+hours):
                section[day][i] = subject
                section.final[day][i] = True
                teacher = subject[2]
                faculty[teacher][day][i] = (section.name, subject)
                faculty[teacher].final[day][i] = True


def finalize_elective(section, day, time, subjects, sub_short):
        global faculty
        section[day][time] = ('Elective', 0, 'Elective staff', sub_short) # last field is the one that matters, others are not used
        section.final[day][time] = True
        for sub in subjects:
                teacher = sub[2]
                if isinstance(teacher, list): # if elective is a lab, it may have multiple teachers
                        for t in teacher:
                                faculty[t][day][time] = (section.name, sub)
                                faculty[t].final[day][time] = True
                else:
                        faculty[teacher][day][time] = (section.name, sub)
                        faculty[teacher].final[day][time] = True
```

```
def free_faculty(teacher, time, day = 'all'):
        if day == 'all':
                for day in 'monday', 'tuesday', 'wednesday', 'thursday', 'friday', 'saturday':
                        if time == 'all':
                                for i in range(1, 8+1):
                                        if i in teacher.final[day]:
                                                teacher.final[day][i] = True
                        else:
                                teacher.final[day][time] = True
        else:
                if time == 'all':
                        for i in range(1, 8+1):
                                if i in teacher.final[day]:
                                        teacher.final[day][i] = True
                else:
                        teacher.final[day][time] = True
def print_dayclash(location):
        logger = logging.getLogger('tt_algo')
        logging.basicConfig(filename = location, level = logging.DEBUG)
        for item in dayclash:
                logger.info('%s %s', item[0].name, item[1])
        logger.info(len(dayclash))
def produce_timetable(ui, loc):
        location = loc
        logger = logging.getLogger('tt_algo')
        logging.basicConfig(filename = location, level = logging.INFO)
        global faculty
        global subjects
        global dayclash
```

```
global clash

dayclash = deque()

clash = deque()

faculty = dict()

for member in ui.faculty_list_value:

        faculty[member] = timetable(str(member))

        faculty[member].dept = ui.department

subjects = OrderedDict()

subjects_ref = dict()

timetables = OrderedDict()

for sem in ui.num_sections:

        if ui.num_sections[sem] > 0:

                subjects[sem] = OrderedDict()

                subjects_ref[sem] = dict()

                timetables[sem] = OrderedDict()

                for section in ui.sections[sem]:

                        subjects[sem][section] = []

                        subjects_ref[sem][section] = dict()

                        timetables[sem][section] = timetable(sem + ' ' + section)

                        timetables[sem][section].dept = ui.department

for sem in ui.subjects_assigned:

        for section in ui.subjects_assigned[sem]:

                for sub in ui.subjects_assigned[sem][section]:

                        sub_long, sub_short, staff = sub.split(' - ')

                        staff = staff.split(', ')

                        sub = ui.subs[sub_short]

                        if ui.subs[sub_short].lab == False:

                                staff = staff[0]

                                faculty[staff].workload += sub.credits
```

```
                                        s = [sub_long, sub.credits, staff, sub_short]
                                        subjects[sem][section].append(s)
                                        subjects_ref[sem][section][sub_short] = s
        for sem in ui.section_fixed_slots:
                for section in ui.section_fixed_slots[sem]:
                        for row in ui.section_fixed_slots[sem][section]:
                                for col in ui.section_fixed_slots[sem][section][row]:
                                        sub_short = ui.section_fixed_slots[sem][section][row][col]
                                        day = ('monday', 'tuesday', 'wednesday', 'thursday', 'friday', 'saturday')[row]
                                        hour = col+1
                                        if sub_short == '-':
                                                timetables[sem][section].final[day][hour] = True
                                        else:
                                                sub_short = sub_short.split('/')
                                                if len(sub_short) > 1: # if it's an elective, this list will have more than
1 subject
                                                        subs = []
                                                        for s in sub_short:
                                                                if s in subjects_ref[sem][section]:

        subs.append(subjects_ref[sem][section][s])
                                                        sub_short = '/'.join(sub_short)
                                                        finalize_elective(timetables[sem][section], day, hour, subs,
sub_short)
                                                else: # not elective
                                                        short = sub_short[0]
                                                        sub = subjects_ref[sem][section][short]
                                                        if ui.subs[short].lab == True:
                                                                finalize_lab(timetables[sem][section], day, hour,
sub, 1)
```

```
                                                    else:
                                                        finalize_theory(timetables[sem][section], day,
hour, sub, 1)
                        print_timetable(timetables[sem][section], location)
        for staff in ui.faculty_fixed_slots:
            for row in ui.faculty_fixed_slots[staff]:
                for column in ui.faculty_fixed_slots[staff][row]:
                    day = ('monday', 'tuesday', 'wednesday', 'thursday', 'friday', 'saturday')[row]
                    hour = column+1
                    free_faculty(faculty[staff], hour, day)
        logger.info('...generating...')
        for sem in timetables:
            for section in timetables[sem]:
                generate(timetables[sem][section], subjects[sem][section], faculty, location)
                print_timetable(timetables[sem][section], location)
        for teacher in faculty:
            faculty[teacher].calc_workload()
        logger.info('...adjusting clashes...')
        adjust_clash(timetables, location, faculty)
        adjust_clash(timetables, location, faculty)
        for sem in timetables:
            for section in timetables[sem]:
                utilize_free_hours(timetables[sem][section], faculty)
                print_timetable(timetables[sem][section], location)
        print_dayclash(location)
        return timetables, faculty, dayclash
if __name__ == '__main__':
        dayclash = deque()
        clash = deque()
```

```
        foura = timetable('4A')

        fourb = timetable('4B')

        fourc = timetable('4C')

        fourd = timetable('4D')

        sixa = timetable('6A')

        sixb = timetable('6B')

        sixc = timetable('6C')

        sixd = timetable('6D')

        eighta = timetable('8A')

        eightb = timetable('8B')

        eightc = timetable('8C')

        eightd = timetable('8D')

        logger = logging.getLogger('tt_algo')

        location = os.path.realpath(os.curdir) + '\example.log' #change this string to create log when running tt.py as a standalone script

        logging.basicConfig(filename = location, level = logging.INFO)

        logger.info('Current Directory: %s', os.path.realpath(os.curdir))

        finalize_theory(foura, 'monday', 2, subjects['4A'][0])

        finalize_theory(foura, 'tuesday', 6, subjects['4A'][0])

        finalize_theory(foura, 'thursday', 4, subjects['4A'][0])

        finalize_theory(foura, 'saturday', 3, subjects['4A'][0])

        finalize_lab(foura, 'wednesday', 2, subjects['4A'][6])

        finalize_lab(foura, 'friday', 6, subjects['4A'][6])

        finalize_theory(foura, 'monday', 3, subjects['4A'][7], hours=2)

        finalize_theory(fourb, 'tuesday', 4, subjects['4B'][0])

        finalize_theory(fourb, 'wednesday', 6, subjects['4B'][0])

        finalize_theory(fourb, 'thursday', 1, subjects['4B'][0])

        finalize_theory(fourb, 'saturday', 1, subjects['4B'][0])

        # labs - B
```

```
finalize_lab(fourb, 'monday', 6, subjects['4B'][6])

finalize_lab(fourb, 'friday', 2, subjects['4B'][6])

# ESC - B

finalize_theory(fourb, 'wednesday', 3, subjects['4B'][7], hours=2)


# maths - C

finalize_theory(fourc, 'monday', 6, subjects['4C'][0])

finalize_theory(fourc, 'tuesday', 4, subjects['4C'][0])

finalize_theory(fourc, 'wednesday', 6, subjects['4C'][0])

finalize_theory(fourc, 'thursday', 2, subjects['4C'][0])

# labs - C

finalize_lab(fourc, 'tuesday', 6, subjects['4C'][6])

finalize_lab(fourc, 'saturday', 1, subjects['4C'][6])

# ESC - C

finalize_theory(fourc, 'wednesday', 7, subjects['4C'][7], hours=2)


# maths - D

finalize_theory(fourd, 'monday', 6, subjects['4D'][0])

finalize_theory(fourd, 'tuesday', 6, subjects['4D'][0])

finalize_theory(fourd, 'thursday', 3, subjects['4D'][0])

finalize_theory(fourd, 'friday', 4, subjects['4D'][0])

# labs - D

finalize_lab(fourd, 'tuesday', 2, subjects['4D'][6])

finalize_lab(fourd, 'wednesday', 6, subjects['4D'][6])

# ESC - D

finalize_theory(fourd, 'friday', 7, subjects['4D'][7], hours=2)


# department constraints

for section in foura, fourb, fourc, fourd, sixa, sixb, sixc, sixd, eighta, eightb, eightc, eightd:
```

```
        for day in 'monday', 'tuesday', 'wednesday', 'thursday', 'friday':
                section.final[day][5] = True # lunch break
        section.final['saturday'][4] = True # saturday

        for hour in 6,7,8: # thursday afternoon
                section.final['thursday'][hour] = True

# faculty constraints
'''
free_faculty(faculty['Mr. Venugopala P S'], 1)
free_faculty(faculty['Mr. Radhakrishna Dodmane'], 8)
free_faculty(faculty['Dr. Uday Kumar Shenoy'], 1)
free_faculty(faculty['Dr. K R Uday Kumar Reddy'], time = 'all', day = 'saturday')
'''

# 6th sem lab
# CG/CN
finalize_lab(sixa, 'monday', 2, subjects['6A'][7])
finalize_lab(sixa, 'wednesday', 2, subjects['6A'][7])
finalize_lab(sixb, 'tuesday', 1, subjects['6B'][7])
finalize_lab(sixb, 'friday', 1, subjects['6B'][7])
finalize_lab(sixc, 'tuesday', 6, subjects['6C'][7])
finalize_lab(sixc, 'friday', 6, subjects['6C'][7])
finalize_lab(sixd, 'monday', 6, subjects['6D'][7])
finalize_lab(sixd, 'wednesday', 6, subjects['6D'][7])
# JIT
finalize_lab(sixa, 'tuesday', 6, subjects['6A'][8])
finalize_lab(sixa, 'friday', 6, subjects['6A'][8])
finalize_lab(sixb, 'monday', 6, subjects['6B'][8])
```

```
finalize_lab(sixb, 'wednesday', 6, subjects['6B'][8])
finalize_lab(sixc, 'monday', 1, subjects['6C'][8])
finalize_lab(sixc, 'wednesday', 1, subjects['6C'][8])
finalize_lab(sixd, 'tuesday', 1, subjects['6D'][8])
finalize_lab(sixd, 'friday', 1, subjects['6D'][8])

# 6th sem OE
# CCIM/MBD
finalize_theory(sixa, 'tuesday', 4, subjects['6A'][4])
finalize_theory(sixa, 'thursday', 2, subjects['6A'][4])
finalize_theory(sixa, 'saturday', 2, subjects['6A'][4])
finalize_theory(sixb, 'tuesday', 4, subjects['6B'][4])
finalize_theory(sixb, 'thursday', 2, subjects['6B'][4])
finalize_theory(sixb, 'saturday', 2, subjects['6B'][4])
finalize_theory(sixc, 'tuesday', 4, subjects['6C'][4])
finalize_theory(sixc, 'thursday', 2, subjects['6C'][4])
finalize_theory(sixc, 'saturday', 2, subjects['6C'][4])
finalize_theory(sixd, 'tuesday', 4, subjects['6D'][4])
finalize_theory(sixd, 'thursday', 2, subjects['6D'][4])
finalize_theory(sixd, 'saturday', 2, subjects['6D'][4])

# MCC/MCAP
finalize_theory(sixa, 'thursday', 3, subjects['6A'][5])
finalize_theory(sixa, 'friday', 4, subjects['6A'][5])
finalize_theory(sixa, 'saturday', 1, subjects['6A'][5])
finalize_theory(sixb, 'thursday', 3, subjects['6B'][5])
finalize_theory(sixb, 'friday', 4, subjects['6B'][5])
finalize_theory(sixb, 'saturday', 1, subjects['6B'][5])
finalize_theory(sixc, 'thursday', 3, subjects['6C'][5])
```

```
finalize_theory(sixc, 'friday', 4, subjects['6C'][5])
finalize_theory(sixc, 'saturday', 1, subjects['6C'][5])
finalize_theory(sixd, 'thursday', 3, subjects['6D'][5])
finalize_theory(sixd, 'friday', 4, subjects['6D'][5])
finalize_theory(sixd, 'saturday', 1, subjects['6D'][5])
'''
print_timetable(foura)
print_timetable(fourb)
print_timetable(fourc)
print_timetable(fourd)
'''
logger.info('... generating ...')
generate(sixa, subjects['6A'], faculty, location)
generate(sixb, subjects['6B'], faculty, location)
generate(sixc, subjects['6C'], faculty, location)
generate(sixd, subjects['6D'], faculty, location)

generate(foura, subjects['4A'], faculty, location)
generate(fourb, subjects['4B'], faculty, location)
generate(fourc, subjects['4C'], faculty, location)
generate(fourd, subjects['4D'], faculty, location)
        generate(eighta, subjects['8A'], faculty)
generate(eightb, subjects['8B'], faculty)
generate(eightc, subjects['8C'], faculty)
generate(eightd, subjects['8D'], faculty)
print_timetable(foura)
print_timetable(fourb)
print_timetable(fourc)
print_timetable(fourd)
```

```
print_timetable(eighta)

print_timetable(eightb)

print_timetable(eightc)

print_timetable(eightd)

print_timetable(sixa)

print_timetable(sixb)

print_timetable(sixc)

print_timetable(sixd)

'''

logger.info('... adjusting clashes ...')

timetables = OrderedDict({

        'VI': OrderedDict({

                'A': sixa,

                'B': sixb,

                'C': sixc,

                'D': sixd

        }),

        'IV': OrderedDict({

                'A': foura,

                'B': fourb,

                'C': fourc,

                'D': fourd

        })

        # }),

        # 'VIII': OrderedDict({

        #       'A': eighta,

        #       'B': eightb,

        #       'C': eightc,

        #       'D': eightd
```

```
        # })
})

for sec in subjects:
        for _, hours, name, _ in subjects[sec]:
                if hours != 0:
                        faculty[name].workload += hours
for name in faculty:
        faculty[name].calc_workload()

adjust_clash(timetables, location, faculty=faculty)
print_dayclash(location)
logger.info('... 2nd pass ...')
adjust_clash(timetables, location, faculty=faculty)
print_dayclash(location)

for sem in timetables:
        for section in timetables[sem]:
                utilize_free_hours(timetables[sem][section], faculty)



print_timetable(foura, location)
print_timetable(fourb, location)
print_timetable(fourc, location)
print_timetable(fourd, location)
print_timetable(sixa, location)
print_timetable(sixb, location)
print_timetable(sixc, location)
print_timetable(sixd, location)
```

```
print_timetable(faculty['Mr. Venugopala P S'], location, style = 'staff', name = 'Mr. Venugopal')

#print_timetable(faculty['Mr. Ramesha Shettigar'], style = 'staff', name = 'Mr. RS')

#print_timetable(faculty['Mr. Pradeep Nazareth'], style = 'staff', name = 'Mr. Pradeep')
```

Conclusion And Future Scope

Generally BCO leads the way to generate an optimal solution. The honey bee movement found the optimal solution even faster. BCO is an optimization techniqusolving complex problems like course timetable problem. This work discusses Bee Colony Optimization (BCO) to find the optimal solutions for designing the course time table. Honey bees are designed on the basis of timeslots in a course timetable which reduces the computasolution is found with the characteristics of the proposed problem and also is able to improve the satisfaction of the teachers and classes toward the schedule in time table. Any conflicts between the teachers schedule, the class schedules, or the classroom schedules are also in this work. The future scope is to optimize the course time table by using Firefly algorithm and Particle swarm optimization along with comparison.

work Distribution

Arun yadav(11811135): proposed methodology, conclusion, introduction

Laksshay Bhateja(11811155): report, abstract, reference

neeraj mishra(11810989): program , literature review ,result and discussion