# Optimization of Public Bus frequency using GA

# Author: Saurav Barua, Daffodil International University

# Problem Statement: No of stops 4 and No of routes 3

# Input data: expected wait time (fixed value), nos. of passenger waited (matrix)

# waitng time (matrix),frequency (need to optimize), proportional coefficient for waiting time (usually = 2),

# comfort index (calculated), Capacity of bus (counted), riding time (matrix), nos. of passenger riding (matrix)

# Boundary condition: Minimum frequency = 7 (to keep LOS A as per TCRP)

# and maximum frequency = 50 (maximum available buses)

# Output: frequency of buses in the three (3) routes

# Mathematical approach: Genetic Algorithm

# Coding support: Python DEAP toolbox

import random

import operator

#import matplotlib.pyplot as plt

#%matplotlib inline

from deap import tools, base, creator, algorithms

# boundary condition

MIN, MAX = 7,50

# initial values (assumed)

SOLUTION = [7, 7, 7]

VARIABLES = len(SOLUTION)

MUT\_MIN, MUT\_MAX = 1, 10

#NGEN = numbers of generation, IND\_SIZE is chromosome numbers

NGEN, IND\_SIZE, CXPB, MUTPB, TRN\_SIZE = 100, 6, 0.5, 0.5, 10

HALL\_SIZE = 10

DEFAULT\_MAIN\_ARGS = NGEN, IND\_SIZE, CXPB, MUTPB

BEST\_INSTANCE\_MSG = 'Best instance:'

NO\_SOLUTION\_MSG = 'No solution in integers. Distance is:'

def fitness(instance):

# frequency of buses in the three routes

x, y, z = instance

# fitness function

return abs(500\*x+1.46\*x\*\*-2+360\*y+4.25\*y\*\*-2+120\*z+7.5\*z\*\*-2),

def spawn\_instance():

return random.randint(MIN, MAX), random.randint(MIN, MAX)

def mutate(instance, mutpb):

if random.random() <= mutpb:

index = random.randint(0, len(instance) - 1)

instance[index] += random.randint(MUT\_MIN, MUT\_MAX)

return instance,

return instance,

def get\_best\_result(population):

if isinstance(population[0], list):

fitness\_values = list(map(fitness, population))

index = fitness\_values.index(min(fitness\_values))

return population[index]

else:

return min(population, key=operator.attrgetter('fitness'))

def terminate(population):

if fitness(get\_best\_result(population)) == (0, ):

raise StopIteration

return False

def distance\_from\_best\_result(population):

result = get\_best\_result(population)

return fitness(result)[0]

def output(best\_instance):

print(BEST\_INSTANCE\_MSG, best\_instance)

distance = fitness(best\_instance)

if distance:

print(NO\_SOLUTION\_MSG, distance)

def setup(mutpb):

creator.create("FitnessMin", base.Fitness, weights=(-1,))

creator.create("Individual", list, fitness=creator.FitnessMin)

toolbox = base.Toolbox()

toolbox.register("attribute", random.randint, MIN, MAX)

toolbox.register("individual", tools.initRepeat, creator.Individual,

toolbox.attribute, n=VARIABLES)

toolbox.register("population", tools.initRepeat, list, toolbox.individual)

toolbox.register("mate", tools.cxOnePoint)

toolbox.register("mutate", mutate, mutpb=mutpb)

toolbox.register("select", tools.selBest)

toolbox.register("evaluate", fitness)

return toolbox

# main method

def main(ngen, ind\_size, cxpb, mutpb):

toolbox = setup(ind\_size)

population = toolbox.population(n=ind\_size)

stats = tools.Statistics()

stats.register("best\_instance\_of\_population", get\_best\_result)

stats.register("distance", distance\_from\_best\_result)

stats.register("terminate", terminate)

halloffame = tools.HallOfFame(HALL\_SIZE)

#stats.register("avg", numpy.mean, axis=0)

#stats.register("std", numpy.std, axis=0)

#stats.register("min", numpy.min, axis=0)

#stats.register("max", numpy.max, axis=0)

try:

pop, logbook = algorithms.eaSimple(population, toolbox, cxpb, mutpb, ngen,

stats=stats, halloffame=halloffame)

except StopIteration:

pass

finally:

best\_instance = halloffame[0]

output(best\_instance)

return best\_instance

#constructor

if \_\_name\_\_ == '\_\_main\_\_':

main(\*DEFAULT\_MAIN\_ARGS)