**CSE422\_Lab\_Assignment02**

# -\*- coding: utf-8 -\*-

"""

@author: Simon Biswas

"""

import random

def gen\_queen(population, fitness):

mutation\_probability = 0.03

new\_population = []

probabilities = [probability(n, fitness) for n in population]

for i in range(len(population)):

x = random\_pick(population, probabilities)

y = random\_pick(population, probabilities)

child = crossover(x, y)

if random.random() < mutation\_probability:

child = mutation(child)

print\_chr\_fit(child)

new\_population.append(child)

if fitness(child) == maxFit: break

return new\_population

def crossover(x, y):

n = len(x)

c = random.randint(0, n - 1)

return x[0:c] + y[c:n]

def mutation(x):

n = len(x)

c = random.randint(0, n - 1)

m = random.randint(1, n)

x[c] = m

return x

def print\_chr\_fit(chrom):

print("Chromosome = {}, Fitness = {}"

.format(str(chrom), fit(chrom)))

def random\_chromosome(size):

return [ random.randint(1, nq) for \_ in range(nq) ]

def fit(chromosome):

horizontal\_collisions = sum([chromosome.count(queen)-1 for queen in chromosome])/2

diagonal\_collisions = 0

n = len(chromosome)

left\_diagonal = [0] \* 2\*n

right\_diagonal = [0] \* 2\*n

for i in range(n):

left\_diagonal[i + chromosome[i] - 1] += 1

right\_diagonal[len(chromosome) - i + chromosome[i] - 2] += 1

diagonal\_collisions = 0

for x in range(2\*n-1):

counter = 0

if left\_diagonal[x] > 1:

counter += left\_diagonal[i]-1

if right\_diagonal[x] > 1:

counter += right\_diagonal[i]-1

diagonal\_collisions += counter / (n-abs(i-n+1))

res = int(maxFit - (horizontal\_collisions + diagonal\_collisions))

return res

def probability(chromosome, fitness):

p = fitness(chromosome) / maxFit

return p

def random\_pick(population, probabilities):

populationWithProbabilty = zip(population, probabilities)

total = sum(w for c, w in populationWithProbabilty)

r = random.uniform(0, total)

upto = 0

for c, w in zip(population, probabilities):

if upto + w >= r:

return c

upto += w

assert False, "Exception occured"

if \_\_name\_\_ == "\_\_main\_\_":

nq = int(input("Number of Queens?"))

maxFit = (nq\*(nq-1))/2

pop = [random\_chromosome(nq) for \_ in range(4)]

generation = 1

while not maxFit in [fit(chrom) for chrom in pop]:

print("Generation {}".format(generation))

pop = gen\_queen(pop, fit)

print("")

print("Maximum Fit = {}".format(max([fit(n) for n in pop])))

generation += 1

chrom\_out = []

print("Solved in Gen {}!".format(generation-1))

for chrom in pop:

if fit(chrom) == maxFit:

print("");

print("A solution: ")

chrom\_out = chrom

print\_chr\_fit(chrom)

mat = []

for i in range(nq):

mat.append(["i"] \* nq)

for j in range(nq):

mat[nq-chrom\_out[j]][j]="Q"

def print\_board(board):

for row in board:

print (" ".join(row))

print\_board(mat)