import sys

import random

import statistics as stat

import math

from numba.typed import List

from bitstring import Bits, BitStream, BitArray, ConstBitStream

from setup import readInput

import os

model, nSite, subSpace, nStates, s2Target, maxItr, startSpinTargetItr, energyTola, spinTola, beta, jVal, det, Ms, posibleDet, bondOrder, outputfile, restart, saveBasis = readInput()

def updateDeterminatList(allDet, allCi, newGen, ci, dataFile, step ):

for idx, elem in enumerate(newGen):

if elem in allDet:

allDet.pop(idx)

allCi.pop(idx)

allDet.append(elem)

allCi.append(ci[idx])

if (step == 0):

#randomly shuffle the list in same order

temp = list(zip(allDet, allCi))

if (step == 1):

temp = list(zip(newGen, ci)) # Train data Set only consist with current iteration data

random.shuffle(temp)

res1, res2 = zip(\*temp)

# res1 and res2 come out as tuples, and so must be converted to lists.

detShuffle, ciShuffle = list(res1), list(res2)

with open(dataFile,"w") as fout:

for idx, elem in enumerate(detShuffle):

for sp in (elem.bin):

if sp == "0":

newline = ("%s,")%("-1")

fout.write(newline)

else:

newline = ("%s,")%(sp)

fout.write(newline)

newline = ("%f\n")%(abs(math.log10(abs(ciShuffle[idx]) + 1e-16 )))

fout.write(newline)

return allDet, allCi

def makeFitGeneration(basis, ci, newSize):

ciOrdered = sorted(abs(ci ), reverse =True)

ciOrdered = ciOrdered[: newSize]

fitness=[]

ciFit = []

#for elem in ciOrdered:

# index\_pos\_list = [ i for i in range(len(ci)) if abs(ci[i]) == elem ]

#for ix in index\_pos\_list:

for x in ciOrdered:

ix = list(abs(ci)).index(x)

if basis[ix] not in fitness:

fitness.append( basis[ix])

ciFit.append(ci[ix])

if (Ms[0] == 0):

fitness.append( ~basis[ix])

ixx = basis.index(~basis[ix])

ciFit.append(ci[ixx])

#print("fit basis length", len(fitness))

return fitness, ciFit

def convInitializer():

# store the n'th state( target state number ) for i'th and i+1'th steps, initialized with garbage value

targetState = [100, 101]

#store the diff of s^2 value of n'th state for i'th and i+1'th steps, initialized with garbage value

s2ValDiff = [0.0, 0.0] # 0'th should be higher than 1'th

# to check convergence, store change in energy for i'th to (i-4)'th steps

energyChange = [1.0, 1.0, 1.0, 1.0, 1.0]

spinChange = [10.0, 10.0, 10.0, 10.0, 10.0]

s2ValList = List()

[s2ValList.append(0.0) for x in range(nStates)]

return targetState, s2ValList, s2ValDiff, energyChange, spinChange

def update( energy, ciCoef, basis, lenSB ):

energySave = energy

ciSave = ciCoef

basisSave = basis

return energySave, ciSave, basisSave

def checkConvergence( eMin, eNew, ciMin, ciNew, s2Min, s2New, targetState, newGen, s2ValDiff, itr, newSize):

Eith = eMin

fitGen = []

if (s2ValDiff[1] - s2ValDiff[0] <= spinTola) :

if ((eNew <= eMin) or (random.random() < math.exp( -( beta \* (eNew - eMin) ) ))):

eMin = eNew

s2Min = s2New

fitGen, ciMin= makeFitGeneration(newGen, ciNew, newSize)

s2ValDiff[0] = s2ValDiff[1]

targetState[0] = targetState[1]

energyUpdate = True

else:

fitGen = newGen[: newSize]

energyUpdate = False

else:

fitGen = newGen[: newSize]

energyUpdate = False

newline = ("ite->\t%d ; spece->\t%d ; Energy->\t%f ; State->\t%d ; s^2 Expe Val->\t%2.4f ;\n")%((itr +1), len(newGen), round( eMin, 6), targetState[0] + 1, round(s2Min,4))

with open(outputfile, "a") as fout:

fout.write(newline)

#print ("ite->\t",(itr + 1),"; Energy->\t",round( eMin, 6),"; State->\t", targetState[0] + 1, "; s^2 Expe Val->\t", round(s2Min,4),";")

return fitGen[: int(0.8 \* newSize)], eMin, ciMin, s2ValDiff, s2Min, energyUpdate

def checkFinalConv(energyChange, spinChange, eOld, eNew, spinChangeIth, convReach):

energyChange = energyChange[1 :]+ [abs(eOld - eNew)]

spinChange = spinChange[1 :]+ [spinChangeIth]

if (( stat.mean(spinChange ) < spinTola) or ( stat.mean( energyChange ) < energyTola)):

convReach = True

return energyChange, spinChange, convReach