import random

from bitstring import Bits, BitStream, BitArray, ConstBitStream

from setup import readInput

from ciRegressionFun import ann\_train, ann\_enrich

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

predictDataFile = outputfile + ".predictData.csv"

enrichDataFile = outputfile + ".enrich.csv"

f1 = open(bondOrder)

line1=f1.readline()

bO1=[]

bO2=[]

while line1:

values = line1.split()

line1 = f1.readline()

bO1.append(int(values[0])-1)

bO2.append(int(values[1])-1)

orderlength = len(bO1)

# define zero and one in correct bit width

zero = BitArray(nSite)

one = zero[1 : nSite] + '0b1'

def mutation (determinantOriginal):

determinant = determinantOriginal.copy()

flag = 0

while(flag == 0):

i = random.randint(0, nSite-1)

j = random.randint(0, nSite-1)

if (determinant[i] != determinant[j]):

determinant[i] , determinant[j] = determinant[j] , determinant[i]

flag = 1

return determinant, ~determinant

def reflection (deternminantOriginal) -> int:

n = deternminantOriginal.copy()

rev = zero

for i in range(nSite):

bit = ( n >> i) & one

rev = rev | (bit << (nSite -1 -i))

return rev, ~rev

def mutationiConected (determinantOriginal):

determinant = determinantOriginal.copy()

flag = 0

while(flag == 0):

i = random.randint(0, orderlength-1)

if (determinant[bO1[i]] != determinant[bO2[i]]):

determinant[bO1[i]] , determinant[bO2[i]] = determinant[bO2[i]] , determinant[bO1[i]]

flag = 1

return determinant, ~determinant

def makeNewGeneration(subBasis):

newGen = subBasis

while (len(newGen) < int( 1.2 \* subSpace)):

#print("lenth newgen", len(newGen))

indx = random.randint(0, (len(subBasis) -1))

prob = random.random()

basisCopy = (subBasis[indx]).copy()

if (prob >= 0.5):

mutated, compliMutated = mutation(basisCopy)

if mutated not in newGen:

newGen.append(mutated)

if (Ms[0] == 0):

newGen.append(compliMutated)

if (prob < 0.5):

reflected, compliReflected = reflection( basisCopy)

if reflected not in newGen:

newGen.append(reflected)

if (Ms[0] == 0):

newGen.append(compliReflected)

return newGen, len(newGen)

def makeNewMlGeneration(subBasis, trainDataSet, newSize, allDet, allCicoef, k):

lenSub = len(subBasis)

#print(lenSub, "before newgen subBasis")

newGen = subBasis.copy()

notUpadated = 0

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

while (len(newGen) < int( 2 \* (newSize))):

indx = min([random.randint(0, lenSub), random.randint(0, lenSub),random.randint(0, lenSub), random.randint(0, lenSub)])

basisCopy = (subBasis[indx]).copy()

mutated, compliMutated = mutationiConected(basisCopy)

if mutated not in newGen :

newGen.append(mutated)

if (Ms[0] == 0):

newGen.append(compliMutated)

notUpadated = 0

for idy, elem in enumerate(mutated.bin):

if (idy < nSite-1) :

if elem == '0':

fout.write("-1,")

if elem == '1':

fout.write("1,")

if (idy == nSite-1) :

if elem == '0':

fout.write("-1\n")

if elem == '1':

fout.write("1\n")

if (Ms[0] == 0):

for idy, elem in enumerate(compliMutated.bin):

if (idy < nSite-1) :

if elem == '0':

fout.write("-1,")

if elem == '1':

fout.write("1,")

if (idy == nSite-1) :

if elem == '0':

fout.write("-1\n")

if elem == '1':

fout.write("1\n")

if mutated in newGen:

notUpadated += 1

if notUpadated == 500:

break

#print("start ANN")

#print("train set", trainDataSet)

mlPreDet = ann\_train(trainDataSet, predictDataFile)

#print(len(mlPreDet), "mlPreDet")

newGen = subBasis + mlPreDet

newGen = newGen[ : int(1.2 \* newSize) ]

#print(len(newGen), "newGen")

# for enrich the data set(use the model to updata train data set)

if (k != 0):

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

for det in allDet:

for idy, elem in enumerate(det.bin):

if (idy < nSite-1) :

if elem == '0':

fout.write("-1,")

if elem == '1':

fout.write("1,")

if (idy == nSite-1) :

if elem == '0':

fout.write("-1\n")

if elem == '1':

fout.write("1\n")

allDet, allCicoef = ann\_enrich(enrichDataFile)

return newGen, len(newGen), allDet, allCicoef