## csc710sbse:hw3:VivekNair:vnair2 Page 1/2 Sep 21, 14 14:14 from \_\_future\_\_ import division import sys import random import math import numpy as np from models import \* from options import \* from utilities import \* sys.dont\_write\_bytecode = True #say = Utilities().say class MaxWalkSat(): model = None minR=0 maxR=0random.seed(40) def \_\_init\_\_(self,modelName): self.model=modelName 20 def evaluate(self): model = self.model #print "Model used: %s"%model.info() minR=model.minR 25 maxR=model.maxR maxTries=int(myoptions['MaxWalkSat']['maxTries']) maxChanges=int(myoptions['MaxWalkSat']['maxChanges']) threshold=float(myoptions['MaxWalkSat']['threshold']) 30 probLocalSearch=float(myoptions['MaxWalkSat']['probLocalSearch']) bestSolution=[] 35 print "Value of p: %f"%probLocalSearch # model = Fonseca() model.baseline(minR,maxR) print model.maxVal,model.minVal 40 for i in range(0,maxTries): #Outer Loop solution=[] for x in range(0,n): solution.append(minR + random.random()\*(maxR-minR)) #print "Solution: ", 45 #print solution for j in range(0,maxChanges): #Inner Loop score = model.evaluate(solution) #print score # optional-start 50 if(score < bestScore):</pre> bestScore=score bestSolution=solution # optional-end 55 if(score < threshold):</pre> print "threshold reached|Tries: %d|Changes: %d"%(i,j) return solution, score if random.random() > probLocalSearch: 60 c = int(0 + (self.model.n-0)\*random.random()) solution[c]=model.neighbour(minR,maxR) else: tempBestScore=score tempBestSolution=solution 65 interval = (maxR-minR)/10 c = int(0 + (self.model.n-0)\*random.random()) for itr in range(0,10): solution[c] = minR + (itr\*interval)\*random.random() tempScore = model.evaluate(solution) 70 if tempBestScore > tempScore: # score is correlated to max? tempBestScore=tempScore tempBestSolution=solution

```
csc710sbse:hw3:VivekNair:vnair2
Sep 21, 14 14:14
                                                                            Page 2/2
                 solution=tempBestSolution
       return bestSolution, bestScore
   def probFunction(old,new,t):
      return math.exp(1 *(old-new)/t)
   class SA():
     model = None
     minR=0
     maxR=0
     random.seed(1)
     def __init__(self,modelName):
       self.model=modelName
     def neighbour(self, solution, minR, maxR):
       returnValue = []
       n=len(solution)
       for i in range(0,n):
          tempRand = random.random()
          if tempRand <(1/self.model.n):</pre>
           returnValue.append(minR + (maxR - minR)*random.random())
95
          else:
           returnValue.append(solution[i])
       return returnValue
     def evaluate(self):
       model=self.model
        #print "Model used: %s"%(model.info())
       minR = model.minR
       maxR = model.maxR
105
       model.baseline(minR,maxR)
       print model.maxVal, model.minVal
       s = [minR + (maxR - minR)*random.random() for z in range(0,model.n)]
       print s
       e = model.evaluate(s)
110
       emax = int(myoptions['SA']['emax'])
       sb = s
                                     #Initial Best Solution
        eh = e
                                     #Initial Best Energy
       k = 1
115
       kmax = int(myoptions['SA']['kmax'])
        while(k \le kmax \land e > emax):
          sn = self.neighbour(s,minR,maxR)
          en = model.evaluate(sn)
          if(en < eb):</pre>
           sb = sn
            eb = en
           print("!"), #we get to somewhere better globally
          tempProb = probFunction(e,en,k/kmax)
          tempRand = random.random()
125
          print " tempProb: %f tempRand: %f " %(tempProb,tempRand)
          if(en < e):
           s = sn
           e = en
            print("+"), #we get to somewhere better locally
130
          elif(tempProb ≤ tempRand):
           jump = True
            s = sn
            print("?") #we are jumping to something sub-optimal;
            count+=1
          print("."),
          k += 1
          if(k % 50 = 0):
140
            print "\n"
           # print "%f{%d}"%(sb,count),
            count=0
       return sb.eb
```