```
#From Class Discussion 8/26/2014
from __future__ import division
import sys,re,random,math
import numpy as np
sys.dont_write_bytecode = True

kmax = 5000
cooling = .6

#Structure from SA Lecture
def say(x):
  sys.stdout.write(str(x)); sys.stdout.flush()

rand = random.random

class ZDT1:
  smin = 0
  smax = 1
  n = 30
  XVar = [random.uniform(smin, smax) for i in range (0, n)]
  XVarMax = XVar
  eMax = 0
  eMin = 0
  slices = 10

  def Energy(self):
    X = self.XVar
    f1 = X[0]
    g = 1+9*(np.sum([X[i] for i in range (1, self.n)])/(self.n-1))
    f2 = g*(1-np.sqrt(X[0]/g))
    return (math.fabs(f1-f2) - self.eMin) / (self.eMax - self.eMin)

  def RawEnergy(self):
    X = self.XVar
    f1 = X[0]
    g = 1+9*(np.sum([X[i] for i in range (1, self.n)])/(self.n-1))
    f2 = g*(1-np.sqrt(X[0]/g))
    return math.fabs(f1-f2)

  def Neighbor(self):
    self.XVar[random.randint(0, self.n-1)] = random.uniform(self.smin, self.smax)

  def BestNeighbor(self):
    toChange = random.randint(0, self.n-1)
    toIncrement = (self.smax - self.smin) / self.slices
    curMax = 1
    maxVal = self.XVar[toChange]
    for i in xrange(self.slices):
      self.XVar[toChange] = self.smin + toIncrement
      x = self.Energy()
      if x < curMax:
        curMax = x
        maxVal = self.XVar[toChange]

  def Chaos(self):
    for vars in self.XVar:
      vars = random.uniform(self.smin, self.smax)

  def Baseline(self, numRuns):
    self.Chaos()
    self.eMax = self.eMin = self.RawEnergy()
    runs = 1
    while runs < numRuns:
      self.Neighbor()
      eNew = self.RawEnergy()
      if eNew > self.eMax: #find largest difference
        self.eMax = eNew
        self.XVarMax = self.XVar
        #print self.XVarMax, eNew
      if eNew < self.eMin: #find smallest difference
        self.eMin = eNew
        #print 'Min: ', self.XVar, eNew
      runs += 1
    #print 'Baseline: ', self.eMin, ', ', self.eMax

  def __init__(self):
    self.Baseline(10000)
```



```
    self.XVar = self.XVarMax
```

```
#From Class Discussion 8/26/2014
from __future__ import division
import sys,re,random,math
import numpy as np
sys.dont_write_bytecode = True

rand = random.random

class Fonseca:
  smin = -4
  smax = 4
  XVar = [random.uniform(smin, smax) for i in range (0, 3)]
  XVarMax = XVar
  eMax = 0
  eMin = 0
  slices = 10

  def Energy(self):
    f1 = (1-math.e**(-np.sum([self.XVar[i]-(1/np.sqrt(i+1))**2 for i in range (0, 3)])))
    f2 = (1-math.e**(-np.sum([self.XVar[i]+(1/np.sqrt(i+1))**2 for i in range (0, 3)])))
    return (math.fabs(f1+f2) - self.eMin) / (self.eMax - self.eMin)

  def RawEnergy(self):
    f1 = (1-math.exp(-np.sum([self.XVar[i]-(1/np.sqrt(i+1))**2 for i in range (0, 3)])))
    f2 = (1-math.exp(-np.sum([self.XVar[i]+(1/np.sqrt(i+1))**2 for i in range (0, 3)])))
    return math.fabs(f1+f2)

  def Neighbor(self):
    r = random.randint(0, 2)
    self.XVar[r] = self.smin + (self.smax - self.smin) * random.uniform(0,1)

  def BestNeighbor(self):
    toChange = random.randint(0,2)
    toIncrement = (self.smax - self.smin) / self.slices
    curMax = 1
    maxVal = self.XVar[toChange]
    for i in xrange(self.slices):
      self.XVar[toChange] = self.smin + toIncrement
      x = self.Energy()
      if x < curMax:
        curMax = x
        maxVal = self.XVar[toChange]

  def Chaos(self):
    self.XVar[0] = random.uniform(self.smin, self.smax)
    self.XVar[1] = random.uniform(self.smin, self.smax)
    self.XVar[2] = random.uniform(self.smin, self.smax)

  def Baseline(self, numRuns):
    self.Chaos()
    self.eMax = self.eMin = self.RawEnergy()
    runs = 1
    while runs < numRuns:
      self.Neighbor()
      eNew = self.RawEnergy()
      if eNew > self.eMax: #find largest difference
        self.eMax = eNew
        self.XVarMax = self.XVar
        #print self.XVarMax, eNew
      if eNew < self.eMin: #find smallest difference
        self.eMin = eNew
        #print 'Min: ', self.XVar, eNew
      runs += 1
    #print 'Baseline: ', self.eMin, ', ', self.eMax

  def EnergyChecker(self, x, y, z):
    self.XVar[0] = x
    self.XVar[1] = y
    self.XVar[2] = z
    print 'Energy @ ', x, ' ', y, ' ', z, ':', self.Energy()

  def __init__(self):
    self.Baseline(10000)
    self.XVar = self.XVarMax
```

```
#From Class Discussion 8/26/2014
from __future__ import division
import sys,re,random,math
import numpy as np
sys.dont_write_bytecode = True

rand = random.random

class Kursawe:
  smin = -5
  smax = 5
  XVar = [random.uniform(smin, smax) for i in range (0, 3)]
  XVarMax = XVar
  eMax = 0
  eMin = 0
  a = 0.8
  b = 3
  slices = 10

  def Energy(self):
    X = self.XVar
    f1 = np.sum([-10*math.exp(-0.2*(np.sqrt(X[i]**2+X[i]**2))) for i in range (0, 3-1)])
    f2 = np.sum([math.fabs(X[i])**self.a + 5*np.sin(X[i])**self.b for i in range (0, 3)])
    return (math.fabs(f1-f2) - self.eMin) / (self.eMax - self.eMin)

  def RawEnergy(self):
    X = self.XVar
    f1 = np.sum([-10*math.exp(-0.2*(np.sqrt(X[i]**2+X[i]**2))) for i in range (0, 3-1)])
    f2 = np.sum([math.fabs(X[i])**self.a + 5*np.sin(X[i])**self.b for i in range (0, 3)])
    return math.fabs(f1-f2)

  def Neighbor(self):
    self.XVar[random.randint(0, 2)] = random.uniform(self.smin, self.smax)

  def BestNeighbor(self):
    toChange = random.randint(0,2)
    toIncrement = (self.smax - self.smin) / self.slices
    curMax = 1
    maxVal = self.XVar[toChange]
    for i in xrange(self.slices):
      self.XVar[toChange] = self.smin + toIncrement
      x = self.Energy()
      if x < curMax:
        curMax = x
        maxVal = self.XVar[toChange]

  def Chaos(self):
    self.XVar[0] = random.uniform(self.smin, self.smax)
    self.XVar[1] = random.uniform(self.smin, self.smax)
    self.XVar[2] = random.uniform(self.smin, self.smax)

  def Baseline(self, numRuns):
    self.Chaos()
    self.eMax = self.eMin = self.RawEnergy()
    runs = 1
    while runs < numRuns:
      self.Neighbor()
      eNew = self.RawEnergy()
      if eNew > self.eMax: #find largest difference
        self.eMax = eNew
        self.XVarMax = self.XVar
        #print self.XVarMax, eNew
      if eNew < self.eMin: #find smallest difference
        self.eMin = eNew
        #print 'Min: ', self.XVar, eNew
      runs += 1
    #print 'Baseline: ', self.eMin, ', ', self.eMax

  def __init__(self):
    self.Baseline(10000)
    self.XVar = self.XVarMax
```

```
#Structure from SA Lecture
import sys,re,random,math
sys.dont_write_bytecode = True

class MWS:
  maxTries = 500
  maxChanges = 500
  cooling = .6
  threshold = .000001
  debug = False

  def say(self, x):
    if self.debug:
      sys.stdout.write(str(x)); sys.stdout.flush()

  def specificRun(self, probability, klass):
    fon = klass()
    XVarBest = fon.XVar
    eBest = e = 1
    k = 1
    self.say(int(math.fabs(eBest-1)*100))
    self.say(' ')
    for i in xrange(self.maxTries):
      fon.Chaos()
      for j in xrange(self.maxChanges):
        eNew = fon.Energy()
        if(eNew < self.threshold):
          #% means found a solution and quit
          self.say('%')
          eBest = eNew
          XVarBest = list(fon.XVar)
          #print '\nQuitting...'
          return eBest, XVarBest
        else:
          #modify random part of solution
          if probability > random.uniform(0,1):
            fon.Neighbor()
            self.say('+')
          #maximize for some random
          else:
            fon.BestNeighbor()
          self.say('.')
          if (i+1)*(j+1) % 40 == 0:
            #print ''
            self.say(int(math.fabs(eNew-1)*100))
            self.say(' ')
      #print ''
      return -1, XVarBest

  def run(self, klass):
    theBest = 0.0
    totalFound = 0
    for i in [0.25, 0.5, 0.75]:
      eBest, XVarBest = self.specificRun(i, klass)
      if eBest == -1:
        #print 'No Best Found for prob = ', i
        self.say('')
      else:
        #print 'Found best - e: ', eBest, ' for prob = ', i
        #print 'Variables: '
        for vars in XVarBest:
          self.say(vars)
          self.say(",")
        #print "\n"
        theBest += eBest
        totalFound += 1
    if totalFound != 0:
      return theBest/totalFound
    else:
      return 2

  def __init__(self, newMaxTries, newCooling, newMaxChanges, newT, newDebug):
    self.maxTries = newMaxTries
    self.cooling = newCooling
    self.maxChanges = newMaxChanges
    self.threshold = newT
    self.debug = newDebug
```

```
#From Class Discussion 8/26/2014
from __future__ import division
import sys,re,random,math
import numpy as np
sys.dont_write_bytecode = True

rand = random.random

class Schaffer:
  smin = -10
  smax = 10
  XVar = [random.uniform(smin, smax) for i in range (0, 1)]
  XVarMax = XVar
  eMax = 0
  eMin = 0
  slices = 10

  def Energy(self):
    f1 = self.XVar[0]*self.XVar[0]
    f2 = (self.XVar[0]-2)*(self.XVar[0]-2)
    return (math.fabs(f1+f2) - self.eMin) / (self.eMax - self.eMin)

  def RawEnergy(self):
    f1 = self.XVar[0]*self.XVar[0]
    f2 = (self.XVar[0]-2)*(self.XVar[0]-2)
    return math.fabs(f1+f2)

  def Neighbor(self):
    self.XVar[0] = self.smin + (self.smax - self.smin) * random.uniform(0,1)

  def BestNeighbor(self):
    toIncrement = (self.smax - self.smin) / self.slices
    curMax = 1
    maxVal = self.XVar[0]
    for i in xrange(self.slices):
      self.XVar[0] = self.smin + toIncrement
      x = self.Energy()
      if x < curMax:
        curMax = x
        maxVal = self.XVar[0]

  def Chaos(self):
    self.XVar[0] = random.uniform(self.smin, self.smax)

  def Baseline(self, numRuns):
    self.Chaos()
    self.eMax = self.eMin = self.RawEnergy()
    runs = 1
    while runs < numRuns:
      self.Neighbor()
      eNew = self.RawEnergy()
      if eNew > self.eMax: #find largest difference
        self.eMax = eNew
        self.XVarMax = self.XVar
        #print self.XVarMax, eNew
      if eNew < self.eMin: #find smallest difference
        self.eMin = eNew
        #print 'Min: ', self.XVar, eNew
      runs += 1
    #print 'Baseline: ', self.eMin, ', ', self.eMax

  def EnergyChecker(self, x):
    self.XVar[0] = x
    print 'Energy @ ', x, ': ', self.Energy()

  def __init__(self):
    self.Baseline(10000)
    self.XVar = self.XVarMax
```

```
import sys

sys.dont_write_bytecode = True

from fonseca_model import *
from schaffer_model import *
from kursawe_model import *
from ZDT1_model import *

from sim_anneal import *
from max_walk_sat import *

r = 1
for klass in [Fonseca, Schaffer, Kursawe, ZDT1]:
  print klass.__name__ + ":"
  for searcher in [SA, MWS]:
    #searcher(500, .6, 500, .00001).run(klass)
    n = 0.0
    for _ in range(r):
      n += searcher(500, .6, 500, .00001, False).run(klass)
      print searcher.__name__ + ':',  n/r


#TODOs (roughly prioritized)

#Implement models folder and searchers folder to import

# Actually implement the numerical analysis functions in searchers
#     -this includes saving of the seed. THIS IS REALLY IMPORTANT DO NOT FORGET

#Further abstraction of models; lots of repeated code

# Break out prints of the algorithm into a debug setting (can turn off/on as needed)
#     -this kind of works now, but I need to handle prints as well without ugly
#      if statements. It's bad and I should feel bad. Will be important for
#      checking if new models work!
```

```
#Structure from SA Lecture
import sys,re,random,math
sys.dont_write_bytecode = True

class SA:
  kmax = 5000
  cooling = .6
  debug = False

  def say(self, x):
    if self.debug:
      sys.stdout.write(str(x)); sys.stdout.flush()

  def run(self, klass):
    sa = klass()
    XVarBest = sa.XVar
    eBest = e = 1
    #print 'start energy: ', eBest
    k = 1
    self.say(int(math.fabs(eBest-1)*100))
    self.say(' ')
    while k < self.kmax:
      sa.Neighbor()
      eNew = sa.Energy()
      if eNew < eBest:
        eBest = eNew
        XVarBest = list(sa.XVar)
        self.say('!')

      if eNew < e:
        e = eNew
        self.say('+')
      #Probability Check from SA Lecture
      elif math.exp(-1*(eNew-e)/(k/self.kmax**self.cooling)) < random.uniform(0,1):
      #P function should be between 0 and 1
      #more random hops early, then decreasing as time goes on
        sa.Chaos()
        self.say('?')
      self.say('.')
      k = k + 1
      if k % 50 ≡ 0 ∧ k ≠ self.kmax:
        #print ''
        self.say(int(math.fabs(eBest-1)*100))
        self.say(' ')

    if self.debug:
      #print '\nFound best - e: ', eBest
      #print 'Variables: '
      for vars in XVarBest:
        self.say(vars)
        self.say(",")
      #print "\n"
    return eBest

  def __init__(self, newKmax, newCooling, nan1, nan2, newDebug):
    self.kmax = newKmax
    self.cooling = newCooling
    self.debug = newDebug
```