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

class Options:
  #Globals
  debug = False
  seed = 1

  #MaxWalkSat options
  mws_prob = 0.75
  mws_maxTries = 500
  mws_maxChanges = 500
  mws_threshold = .000001
  mws_slices = 10

  #Simulated Annealing options
  sa_kmax = 5000
  sa_cooling = .6

  def printGlobals(self):
    print "Seed:", self.seed
```

```
============================================================
Model Name:  Fonseca
Searcher Name:  SA
Seed: 1
SA Options:
KMAX: 5000 Cooling: 0.6
Time to run (s):  3.777827
Runs:  10
Average per run (s):  0.3777827
*           |            , −0.00000,  0.00000,  0.00000,  0.00000,  0.00000
============================================================
============================================================
Model Name:  Fonseca
Searcher Name:  MaxWalkSat
Seed: 1
MaxWalkSat Options:
Prob: 0.75
MaxTries: 500 MaxChanges 500
Threshold: 1e−06 Slices: 10
Time to run (s):  0.570184
Runs:  10
Average per run (s):  0.0570184
*           |            , 0.00000,  0.00000,  0.00000,  0.00000,  0.00000
============================================================
============================================================
Model Name:  Schaffer
Searcher Name:  SA
Seed: 1
SA Options:
KMAX: 5000 Cooling: 0.6
Time to run (s):  0.227492
Runs:  10
Average per run (s):  0.0227492
*           |            , −0.00000,  −0.00000,  0.00000,  0.00000,  0.00000
============================================================
============================================================
Model Name:  Schaffer
Searcher Name:  MaxWalkSat
Seed: 1
MaxWalkSat Options:
Prob: 0.75
MaxTries: 500 MaxChanges 500
Threshold: 1e−06 Slices: 10
Time to run (s):  0.030728
Runs:  9
Average per run (s):  0.00341422222222
*           |            , 0.00000,  0.00000,  0.00000,  0.00000,  0.00000
============================================================
============================================================
Model Name:  Kursawe
Searcher Name:  SA
Seed: 1
SA Options:
KMAX: 5000 Cooling: 0.6
Time to run (s):  3.083479
Runs:  10
Average per run (s):  0.3083479
------   *  |    --------, −0.00547,  0.00199,  0.00597,  0.01490,  0.02413
============================================================
============================================================
Model Name:  Kursawe
Searcher Name:  MaxWalkSat
Seed: 1
MaxWalkSat Options:
Prob: 0.75
MaxTries: 500 MaxChanges 500
Threshold: 1e−06 Slices: 10
Time to run (s):  0.860436
Runs:  6
Average per run (s):  0.143406
*           |            , −0.00092,  −0.00092,  −0.00092,  −0.00092,  −0.00092
============================================================
============================================================
Model Name:  ZDT1
Searcher Name:  SA
Seed: 1
SA Options:
```



```
KMAX: 5000 Cooling: 0.6
Time to run (s):  1.824436
Runs:  10
Average per run (s):  0.1824436
           |      *    −, 0.06181,  0.06355,  0.07678,  0.10068,  0.10510
============================================================
============================================================
Model Name:  ZDT1
Searcher Name:  MaxWalkSat
Seed: 1
MaxWalkSat Options:
Prob: 0.75
MaxTries: 500 MaxChanges 500
Threshold: 1e−06 Slices: 10
Time to run (s):  0.301238
Runs:  10
Average per run (s):  0.0301238
*           |            , −0.02102,  −0.02102,  −0.02102,  −0.02102,  −0.02102
============================================================
```

```
import sys
from datetime import datetime
import random

sys.dont_write_bytecode = True

from models import *
from searchers import *
from utils import *
from options import *

myOpt = Options()

#Inspired by vivekaxl's display function
def display(model, searcher, startTime, scores, r):
  print "============================================================"
  print "Model Name: ", model.__name__
  print "Searcher Name: ", searcher.name
  diff = (datetime.now() - startTime).total_seconds()
  myOpt.printGlobals()
  searcher.printOptions()
  print "Time to run (s): ", diff
  if r == 0:
    print "No valid runs!"
  else:
    print "Runs: ", r
    print "Average per run (s): ", diff/r
    print xtile(scores,width=25,show=" %1.5f")
  print "============================================================"

def main(modelList, searcherList):
  r = 10
  for klass in modelList:
    for searcher in searcherList:
      startTime = datetime.now()
      scores = []
      myKlass = klass()
      mySearcher = searcher()
      random.seed(myOpt.seed)
      for _ in range(r):
        result, valid = mySearcher.run(myKlass)
        if valid == True:
          scores.append(result)
      display(klass, mySearcher, startTime, scores, len(scores))

modelList = [Fonseca, Schaffer, Kursawe, ZDT1]
searcherList = [SA, MWS]

main(modelList, searcherList)
```

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

from options import *

#Taken verbatim from the class website.

def pairs(lst):
  last=lst[0]
  for i in lst[1:]:
    yield last,i
    last = i

def xtile(lst,lo=0,hi=0.001,width=50,
             chops=[0.1 ,0.3,0.5,0.7,0.9],
             marks=["-" ," "," "," ","-"," "],
             bar="|",star="*",show=" %3.0f"):
  """The function _xtile_ takes a list of (possibly)
 unsorted numbers and presents them as a horizontal
 xtile chart (in ascii format). The default is a
 contracted _quintile_ that shows the
 10,30,50,70,90 breaks in the data (but this can be
 changed- see the optional flags of the function).
 """
  def pos(p): return ordered[int(len(lst)*p)]
  def place(x):
    return int(width*float((x - lo))/(hi - lo))
  def pretty(lst):
    return ','.join([show % x for x in lst])
  ordered = sorted(lst)
  lo      = min(lo,ordered[0])
  hi      = max(hi,ordered[-1])
  what    = [pos(p)   for p in chops]
  where   = [place(n) for n in  what]
  out     = [" "] * width
  for one,two in pairs(where):
    for i in range(one,two):
      out[i] = marks[0]
    marks = marks[1:]
  out[int(width/2)]    = bar
  out[place(pos(0.5))] = star
  return ''.join(out) +  "," +  pretty(what)
```

```
from sim_anneal import *
from max_walk_sat import *
```

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

from options import *

myOpt = Options()

class MWS:
  debug = False
  name = "MaxWalkSat"

  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(myOpt.mws_maxTries):
      fon.Chaos()
      for j in xrange(myOpt.mws_maxChanges):
        eNew = fon.Energy()
        if(eNew < myOpt.mws_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 = -1
    valid = False
    eBest, XVarBest = self.specificRun(myOpt.mws_prob, klass)
    if eBest == -1:
      #print 'No Best Found for prob = ', i
      self.say('')
    else:
      theBest = eBest
      valid = True
    return theBest, valid

  def printOptions(self):
    print "MaxWalkSat Options:"
    print "Prob:", myOpt.mws_prob
    print "MaxTries:", myOpt.mws_maxTries, "MaxChanges", myOpt.mws_maxChanges
    print "Threshold:", myOpt.mws_threshold, "Slices:", myOpt.mws_slices
```

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

from options import *
myOpt = Options()

class SA:
  name = "SA"

  def say(self, x):
    if myOpt.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 < myOpt.sa_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/myOpt.sa_kmax**myOpt.sa_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 ≠ myOpt.sa_kmax:
        #print ''
        self.say(int(math.fabs(eBest-1)*100))
        self.say(' ')

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

  def printOptions(self):
    print "SA Options:"
    print "KMAX:", myOpt.sa_kmax, "Cooling:", myOpt.sa_cooling
```

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

from model_base import *
from options import *

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

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

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

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

from model_base import *
from options import *

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

  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 __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

from model_base import *
from options import *

myOpt = Options()

class Kursawe(Model):
  n = 3
  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

  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 __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

from options import *
myOpt = Options()
rand = random.random

class Model:
  #Default Values overwritten by subclass; should have better defaults, but...
  n = 1
  smin = 1
  smax = 1
  XVar = [random.uniform(smin, smax) for i in range (0, n)]
  XVarMax = XVar
  eMax = 0
  eMin = 0

  def Energy(self):
    print "Energy Class Undefined!"

  def RawEnergy(self):
    print "RawEnergy Class Undefined!"

  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) / myOpt.mws_slices
    curMax = 1
    maxVal = self.XVar[toChange]
    for i in xrange(myOpt.mws_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):
    print "Default init Shouldn't be used!"
```

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

from model_base import *
from options import *

sys.dont_write_bytecode = True

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

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