```
from __future__ import division
import sys, random, math
from models import *
from sk import *
from base import *
import numpy as np
from xtile import *
from ga import *
sys.dont_write_bytecode = True
@printlook
def sa(model):
  def P(old, new, t):
    prob = math.e**((old − new)/t)
    return prob
  min_energy, max_energy = model.baseline()
  s = model.generate_x()
  e = model.norm(model.getDepen(s))
  sb = s
  eb = e
  k = 1
  icontrol = Control(model)
  while k < Settings.sa.kmax:
    stopsign = icontrol.next(k) #true −−−stop
    if stopsign:
      break
    sn = model.sa_neighbor(s)
    en = model.norm(model.getDepen(sn))
    icontrol.logxy(sn)
    temp = (k/Settings.sa.kmax)*Settings.sa.cooling
    if en < eb:
          sb = sn
          eb = en
          say('!')
    if en < e:
      s = sn
      e = en
      say('+')
    elif P(e, en, temp) < random.random():
      s = sn
      e = en
      say('?')
    say('.')
    k = k + 1
    if k % 30 ≡ 0:
      print "\n"
      say(str(round(eb,3)))
  print "\n"
  # printReport(model)
  print "\n−−−−−−\n:Normalized Sum of Objectives : ",str(round(eb,3)),"\n:Solution",sn
  lohi=printRange(model)
  return eb,lohi
#
@printlook
def mws(model):

  min_energy, max_energy = model.baseline()
  total_changes = 0
  total_tries = 0
  norm_energy = 0
  eraScore = []
  control = Control(model)
  optimalsign = False
  solution = model.generate_x()
  norm_energy = model.norm(model.getDepen(solution))
  for k in range(Settings.mws.max_tries):
    total_tries += 1
    for _ in range(Settings.mws.max_changes):
      stopsign = control.next(total_changes) #true −−−stop
      if stopsign:
        break
      if norm_energy ≤ Settings.mws.threshold:
        optimalsign = True
        break
```

```
      if  random.random()≤Settings.mws.prob:
        solution[random.randint(0,model.n−1)] = model.generate_x()[random.randin
t(0,model.n−1)]
        control.logxy(solution)
        say("+")
      else:
        solution = model.mws_neighbor(solution)
        control.logxy(solution)
        say("!")
      say(".")
      if total_changes % 30 ≡ 0:
        print "\n"
        say(str(round(model.norm(model.getDepen(solution)), 3)))
      total_changes +=1
    if optimalsign ∨ k ≡ Settings.mws.max_tries−1:
      say("\n")
      say(str(round(model.norm(model.getDepen(solution)), 3)))
      print "\n"
      print "total tries: %s" % total_tries
      print "total changes: %s" % total_changes
      print "min_energy:{0}, max_energy:{1}".format(min_energy, max_energy)
      print "min_energy_obtained: %s" % model.getDepen(solution)
      # printReport(model)
      lohi =printRange(model)
      print "\n−−−−−−\n:Normalized Sum of Objectives: ",str(round(norm_energy,3)),"\n:Solution
",solution, "\n"
      return norm_energy, lohi


def printReport(m):
  for i, f in enumerate(m.log.y):
    print "\n <f%s" %i
    for era in sorted(m.history.keys()):
      # pdb.set_trace()
      log = m.history[era].log.y[i]
      print str(era).rjust(7), xtile(log._cache, width = 33, show = "%5.2f", lo
= 0, hi = 1)

def printRange(m):
  lo = []
  lohi = []
  # print sorted(m.history.keys())
  for i, f in enumerate(m.log.y):
    tlo=10**5
    thi=−10**5
    for era in sorted(m.history.keys()):
      # pdb.set_trace()
      if m.history[era].log.y[i].lo < tlo:
        tlo= m.history[era].log.y[i].lo
      if m.history[era].log.y[i].hi > tlo:
        thi= m.history[era].log.y[i].hi
    lohi.append(tlo)
    lohi.append(thi)
  return  lohi
    # print "\n the range of f%s is %s to %s " % (i, str(tlo), str(thi))

@demo
def start(): #part 5 with part 3 and part4
  r = Settings.other.repeats
  rlohi=[] # stupid codes here, to be fixed
  f1lo = []
  f1hi = []
  f0lo = []
  f0hi =[]
  f2lo =[]
  f2hi =[]
  for klass in [Schaffer, Fonseca, Kursawe, ZDT1, ZDT3, Viennet3]:
    print "\n !!!!", klass.__name__
    for searcher in [ga, sa, mws]:
      name = klass.__name__
      n = 0.0
      reseed()
      # scorelist = []
      for _ in range(r):
        x, lohi=searcher(klass()) # lohi is a list containing [lo,hi] paris of f1&f2
       #========part 5==========
        rlohi.append(lohi)
      for i in range(0, r):
        f0lo.append(rlohi[i][0])
        f0hi.append(rlohi[i][1])
        f1lo.append(rlohi[i][2])
        f1hi.append(rlohi[i][3])
        if name ≡ "Viennet3": # f1, f2, f3
          f2lo.append(rlohi[i][4])
          f2hi.append(rlohi[i][5])
      print "\n # The range of f0 during %s repeats is from %s to %s " \
             % (r, str(round(sorted(f0lo)[0], 3)), str( round(sorted(f0hi)[-1])))
      print "\n # The range of f1 during %s repeats is from %s to %s " \
             % (r, str(round(sorted(f1lo)[0],3)), str(round(sorted(f1hi)[-1])))
      if name ≡"Viennet3":
        print "\n # The range of f1 during %s repeats is from %s to %s "\
             % (r, str(round(sorted(f2lo)[0],3)), str(round(sorted(f2hi)[-1])))
      rlohi = []
      #=====part 5 ends===========

      #the following codes for hw3
      # n += float(x)
      # scorelist +=[float(x)]
      # print xtile(scorelist,lo=0, hi=1.0,width = 25)
      # print "# {0}:{1}".format(name, n/r)
@demo
def part6():
  r = 5
  lastera = []
  searchcount = 0
  for klass in [ZDT1]:
    print "\n !!!!", klass.__name__
    for searcher in [sa, mws]:
      reseed()
      for k in range(r):
        Settings.sa.cooling = rand() # get variants of sa, mws
        Settings.mws.prob = rand()
        Settings.mws.max_changes = int(1000*rand())
        model = klass()
        x, lohi = searcher(model)
        for i, f in enumerate(model.log.y):
          temp = []
          searchername = "mws" if searchcount else "sa"
          label = searchername + str(k) +"f%s" %i
          temp = (model.history[sorted(model.history.keys())[-1]].log.y[i]._cache)
          temp = [ float(i) for i in temp]
          temp.insert(0,str(label))
          lastera.append(temp)
      rdivDemo(lastera)
      searchcount +=1
      lastera = []
@demo
def HW5():
  for klass in [ Schaffer, Fonseca, Kursawe, ZDT1, ZDT3, Viennet3, DTLZ7]:
    print "\n !!!!", klass.__name__
    allEB  = []
    for searcher in [ga, sa, mws]:
      repeats = 5
      eb = 5*[0]
      name = klass.__name__
      reseed()
      for r in range(repeats):
        results=searcher(klass()) # lohi is a list containing [lo,hi] paris of f1&f2
        eb[r] = results[0][0]
      eb.insert(0, results[1])
      allEB.append(eb)
      rdivDemo(allEB)
```

```


@demo
def testmodel():
  # model = ZDT3()
  #model = Schaffer()
  model = DTLZ7()
  depen = model.getDepen(model.generate_x())
  print depen

if __name__ ≡ "__main__": eval(cmd())
```

```
from __future__ import division
from log import *
from models import *
from xtile import *
from optimizer import *
from base import *
import sys, random, math, datetime, time,re, pdb, operator
sys.dont_write_bytecode = True

@printlook
def ga(model):
  mutationRate = 1/model.n
  population = []
  solution =[]
  children = []
  fitness = {}
  mateNum = 20
  def selection(sortedFitness):
    return [population[sortedFitness[0][0]], population[sortedFitness[1][0]]] # sroted[0] and [1] are the smallest two we preferred
  def crossover(selected):
    '''crossover will do this way: offsprint1 = p* parent 1+ (1−p)* parent2 for numbers between two points '''
    def what(lst):
      return lst[0] if isinstance(lst, list) else lst
    children1 = []
    if rand()> Settings.ga.crossRate:
      return selected[0]
    else:
      if model.n ≡1:
        children1 = [(what(selected[0]) + what(selected[1]))*0.5]
      else:
        index = sorted([random.randint(0, model.n − 1) for _ in xrange(Settings.ga.crossPoints)])
        parent1 = selected[0]
        parent2 = selected[1]
        children1 = parent1[:]
        children1[index[0]:index[1]] = parent2[index[0]:index[1]]
      return children1
  def mutate(children, selected):
    # print children
    for k, n in enumerate(children):
      if rand()< mutationRate:
        children[k]= selected[random.randint(0,1)][random.randint(0, model.n−1)] # pick value from mom or dad
    # print children
    return children
  def tournament(sortedFitness, m=10): # do tornament selection, select the best daddy or mom in  m = 10 candidates
    index = []
    for _ in range(m):
      index.append(random.randint(0, Settings.ga.pop−1))
    betterIndex = list(set(sorted(index)))
    parentlst = [population[sortedFitness[betterIndex[0]][0]], population[sortedFitness[betterIndex[1]][0]]]
    return parentlst
  def fit(fitness):
    sortedFitness = sorted(fitness.items(), key = lambda x:x[1]) # a sorted list
    return sortedFitness[:Settings.ga.pop] # just return the top 50 candidates as new populatioin
  def produce(selected):
        children = crossover(selected)
        children = mutate(children, selected)
        return children

  min_energy, max_energy = model.baseline()
  eb= 0
  solution = []
  control = Control(model)
  for _ in xrange(Settings.ga.pop):
    temp = model.generate_x()
    population.append(temp)
  # for num in Settings.ga.genNum:
```

```
  t = 0
  while(t < Settings.ga.genNum): # figure stop out
    stopsign = control.next(t) #true ---stop
    if stopsign:
      break
    for (k, xlst) in enumerate(population):
      fitness[k] = model.getDepen(xlst)
    newpopfitness = fit(fitness)
    for n, k in newpopfitness:
      population[n] = population[newpopfitness[0][0]] # new generation
      control.logxy(population[n])
    # for n, k in population:
    #   control.logxy(k) # log new generation
    eb = model.norm(newpopfitness[0][1])
    solution = population[newpopfitness[0][0]]
    for _ in range(mateNum):
      selected = tournament(newpopfitness)
      children.append(produce(selected))
    population.extend(children)
    t +=1
  print "best solution : %s" % str(solution)
  print "best normalized results: %s" % str(eb)
  print "−"*20
  # printReport(model)
  lohi=printRange(model)
  return eb,lohi

def startga():
  for klass in [Schaffer, Fonseca, Kursawe, ZDT1, ZDT3, Viennet3]:
  # for klass in [DTLZ7]:
    print "="*50
    print "!!!!", klass.__name__,
    print "\nSearcher: GA"
    reseed()
    ga(klass())


# if __name__ == "__main__":startga()
     # print sortedFitness
```

```
from __future__ import division
from log import *
import sys, random, math, datetime, time,re, pdb
sys.dont_write_bytecode = True


exp = math.e
sqrt = math.sqrt
sin = math.sin
pi = math.pi

class Model:

  def name(i):
    return i.__class__.__name__
  def setup(i):
    i.xy = Options(x = [i.generate_x()], y = [i.f1, i.f2])
    i.log = Options(x = [ Num() for _ in range(i.n)], y = [ Num() for _ in range(i.fn)]) # hardcode 2
    i.history = {} # hold all logs for eras
  def generate_x(i):
    x= [i.lo + (i.hi−i.lo)*random.random() for _ in range(i.n)]
    return x
  def getDepen(i, xlst):
    # y = [i.f1, i.f2]
    return sum([f(xlst) for f in i.xy.y])
  def getDepenlst(i, xlst):
    return [f(xlst) for f in i.xy.y]
  def cloneModel(i): # from Dr.Menzies’
    return i.__class__()
  def logxy(i, x):
    for val, log in zip(x, i.log.x): log += val
    y = i.getDepenlst(x)
    for val, log in zip(y, i.log.y): log += val
  def better(news,olds): # from Dr.Menzies’
    def worsed():
      return  ((same ∧ ¬ betterIqr) ∨
               (¬ same ∧ ¬ betterMed))
    def bettered():
      return  ¬ same ∧ betterMed
    out = False
    for new,old in zip(news.log.y, olds.log.y):
      betterMed, same, betterIqr = new.better(old)
      # print betterMed, same, betterIqr
      # pdb.set_trace()
      if worsed()  : return False # never any worsed
      if bettered(): out= out ∨ True # at least one bettered
    return out
  def sa_neighbor(i, old):
    p = 1/i.n
    new = old
    for j in range(len(old)):
      if random.random() < p:
        new_gen = i.generate_x()
        old[j] = new_gen[random.randint(0, i.n−1)]
    return old
  def mws_neighbor(i,solution):
    optimized_index = random.randint(0, len(solution)−1)
    increment = (i.hi − i.lo)/10
    temp_min = 10*(5)
#   print "old solution : %s" % solution
    for _ in range(10):
      solution[optimized_index] = i.lo + increment
      temp = i.norm(i.getDepen(solution))
      if temp < temp_min:
        temp_min = temp
#   print "new solution : %s" % solution
    return solution
  def baseline(i):
  # model = eval(model+"()")
    i.min = 10**(5)
    i.max = −10**(5)
    for _ in xrange(10000):
```

```
      temp = i.getDepen(i.generate_x())
      if temp > i.max:
        i.max = temp
      if temp < i.min:
            i.min = temp
    return i.min, i.max
  def norm(i, x):
        e = (x − i.min)/(i.max − i.min)
        return e #avoid values <0 or >1

class Control(object): # based on Dr.Menzies’ codes
  def __init__(i, model):
    i.kmax = Settings.sa.kmax
    i.era = Settings.other.era
    i.lives = Settings.other.lives
    i.logAll = {}
    i.model = model
  def __call__(i, k):
    i.next(k)
  def logxy(i, results):
    both = [i.model.history, i.logAll]
    for log in both:
      if ¬ i.era in i.logAll:
        log[i.era] = i.model.cloneModel()
    for log in both:
      log[i.era].logxy(results)
  def checkimprove(i):
      if len(i.logAll) ≥ 100:
        current = i.era
        before = i.era − Settings.other.era
        currentLog = i.logAll[current]
        beforeLog = i.logAll[before]
        # pdb.set_trace()
        if ¬ currentLog.better(beforeLog):
          pass
        else:
          i.lives += 1
  def next(i, k):
    if k ≥ i.era:
      i.checkimprove()
      i.era +=Settings.other.era
      if i.lives ≡ 0:
        return True
      else:
        i.lives −=1
        return False



’’’Schaffer’’’
class Schaffer(Model):
  def __init__(i):
    i.lo = −2
    i.hi = 2
    i.n = 1
    i.fn = 2
    i.setup()
  def f1(i, x):
    return x[0] * x[0]
  def f2(i, x):
    return (x[0]−2) ** 2

’’’Fonseca’’’
class Fonseca(Model):
  def __init__(i):
    i.lo = −4
    i.hi = 4
    i.n = 3
    i.fn = 2
    i.setup()
  def f1(i, xlst):
    return (1 − exp**(−1 * sum([(xlst[k] − 1/sqrt(i.n))**2 for k in xrange(i.n)])))
  def f2(i, xlst):
    return (1 − exp**(−1 * sum([(xlst[k] + 1/sqrt(i.n))**2 for k in xrange(i.n)]
)))

'''Kusarvs'''
class Kursawe(Model):
  def __init__(i):
    i.lo = −5
    i.hi = 5
    i.n = 3
    i.fn = 2
    i.setup()
  def f1(i, xlst):
    return sum([−10*exp**(−0.2 * sqrt(xlst[k]**2 + xlst[k+1]**2)) for k in xrang
e(i.n −1)])
  def f2(i, xlst):
    a = 0.8
    b = 3
    return sum([abs(x)**a + 5*sin(x)**b for x in xlst])

'''ZDT1'''
class ZDT1(Model):
  def __init__(i):
    i.lo = 0
    i.hi = 1
    i.n = 30
    i.fn = 2
    i.setup()
  def f1(i, xlst):
    return xlst[0]
  def g(i, xlst):
    return (1 + 9 * (sum(xlst[1:]))/(i.n−1))
  def f2(i,xlst):
    g1 = i.g(xlst)
    return g1*(1−sqrt(xlst[0]/g1))

'''ZDT3'''
class ZDT3(Model):
  def __init__(i):
    i.lo = 0
    i.hi = 1
    i.n = 30
    i.fn = 2
    i.setup()
  def f1(i, xlst):
    return xlst[0]
  def g(i, xlst):
    return (1 +  (9/(i.n−1)) * sum(xlst[1:]))
  def h(i,f1,g):
    return (1 − sqrt(f1/g) − f1/g) * sin(10 * pi * f1)
  def f2(i, xlst):
    return i.g(xlst) * i.h(i.f1(xlst),i.g(xlst))

'''Viennet3'''
class Viennet3(Model):
  def __init__(i):
    i.lo = −3
    i.hi = 3
    i.n = 2
    i.fn = 3
    i.setup1()
  def setup1(i):
    i.xy = Options(x = [i.generate_x()], y = [i.f1, i.f2, i.f3])
    i.log = Options(x = [ Num() for _ in range(i.n)], y = [ Num() for _ in range
(i.fn)]) # hardcode 2
    i.history = {} # hold all logs for eras
  def f1(i, xlst):
    xy2 = xlst[0]**2 + xlst[1]**2
    return 0.5* (xy2) + sin(xy2)
  def f2(i, xlst):
    x = xlst[0]
    y = xlst[1]
    return ((3*x −2*y +4)**2/8 + (x−y+1)**2/27 + 15)
```

```
  def f3(i, xlst):
    xy2 = xlst[0]**2 + xlst[1]**2
    return (1/(xy2+1) − 1.1* exp**(−xy2))

'''DTLZ7'''
class DTLZ7(Model):
  def __init__(i):
    i.M = 20
    i.K = 20
    i.lo = 0
    i.hi = 1
    i.n = i.M + i.K −1
    i.fn = i.M
    i.setup()
  def fi(i, x): # the frist one is x[0]
    return x
  def fm(i, xh=0):
    return (1 + i.g())*i.h()
  def g(i):
    return 1 + (9/i.K) * sum(i.xy.x[:i.M−1])
  def h(i):
    sumtemp = 0
    for n,x in enumerate(i.xy.x):
      if n ≡i.M−2:
        break
      sumtemp +=(i.xy.y[n](x)/(1.0+i.g()))*(1+sin(3.0*pi*i.xy.y[n](x)))
    return (i.M − sumtemp)# k = 0,...., M−2
  def setup(i):
    tempx = i.generate_x()
    tempy = [i.fi for k in tempx[:−1]]
    tempy.append(i.fm)
    i.xy = Options(x = tempx, y = tempy)
    i.log = Options(x = [ Num() for _ in range(i.n)], y = [ Num() for _ in range
(i.fn)])
    i.history = {} # hold all logs for eras
  def getDepen(i, xlst):
    temp = i.fm()
    return sum(xlst[:i.M])+temp
```

```
from __future__ import division
import sys, random, math
from base import *
from a12 import *
sys.dont_write_bytecode = True


’’’All these are based on Dr.Menzies’ tricks ∧ sample codes’’’

class Log():
  def __init__(i, tolog = []):
    i._cache, i.n, i._report = [], 0, None
    i.setup()
    map(i.__iadd__, tolog)
  def __iadd__(i, tolog):
    if tolog == None: return tolog
    i.n += 1
    updated = False
    if len(i._cache) < Settings.other.keep:
      i._cache +=[tolog]
      updated = True
    else:
      if rand() <= Settings.other.keep/i.n:
         i._cache[int(rand()*Settings.other.keep)] = tolog
         updated = True
    if updated:
      i._report = None
      i.updateLoHi(tolog)
    return i
  def has(i):
    if i._report == None:
      i._report = i.report()
    return i._report

class Num(Log):
  def setup(i):
    i.lo = 10**5
    i.hi = −10**5
  def updateLoHi(i,x):
    i.lo = min (i.lo, x)
    i.hi = max(i.hi, x)
  def median(i):
    n = len(i._cache)
    p = n//2
    if (n % 2) : return i._cache[p]
    q = p +1
    q = max(0, min(q,n))
    return (i._cache[p] + i._cache[q])/2
  def better(new,old):
    "better if (1)less median or (2)same and less iqr"
    t = Settings.other.a12
    betterIqr = new.has().iqr < old.has().iqr
    new.lessp = False
    if new.lessp:
      betterMed = new.has().median >= old.has().median
      same      = a12(old._cache, new._cache)  <= t
    else:
      betterMed = new.has().median <= old.has().median
      same      = a12(new._cache, old._cache) <= t
    return betterMed, same, betterIqr
  def report(i):
    sortedCache = sorted(i._cache)
    n = len (sortedCache)
    return Options(
           median = i.median(),
           iqr = sortedCache[int(n*0.75)− int(n*0.5)],
           lo = i.lo,
           hi = i.hi)

@demo
def demoNum():
  for size in [16,32, 64,128, 256]:
    Settings.other.keep = size
```

```
    log = Num()
    for x in xrange(100000): log +=x
    print size, ":", log.has().median



if __name__ == "__main__": eval(cmd())
```

```
from __future__ import division
import sys, random, math, datetime, time,re, pdb
sys.dont_write_bytecode = True

rand= random.random


class Options: #"Thanks for Peter Norvig’s trick"
  def __init__(i, **d): i.__dict__.update(d)

Settings = Options(sa = Options(kmax = 1000,
                                      baseline = 1000,
                                score = {},
                                cooling = 0.5),
                   mws = Options(threshold = 0.0001,
                                max_tries = 20,
                                max_changes = 1000,
                                prob = 0.25,
                                score = {}
                                ),
                   ga = Options( pop = 50,
                                 crossRate = 0.6,
                                 crossPoints = 2,
                                 genNum = [100, 200, 400, 800]
                                ),
                   other = Options(keep = 64 ,
                                   era = 50,
                                   lives = 1,
                                   baseline = 1000,
                                   a12 = [0.56, 0.64, 0.71][0],
                                   repeats = 30))
def atom(x):
  try : return int(x)
  except ValueError:
    try : return float(x)
    except ValueError : return x

def cmd(com="demo(’−h’)"):
  "Convert command line to a function call."
  if len(sys.argv) < 2: return com
  def strp(x): return isinstance(x,basestring)
  def wrap(x): return "’%s’"%x if strp(x) else str(x)
  words = map(wrap,map(atom,sys.argv[2:]))
  return sys.argv[1] + ’(’ + ’,’.join(words) + ’)’

def demo(f=None,cache=[]):
  def doc(d):
    return ’# ’+d.__doc__ if d.__doc__ else ""
  if f ≡ ’−h’:
    print ’# sample demos’
    for n,d in enumerate(cache):
      print ’%3s) ’ %(n+1),d.func_name,doc(d)
  elif f:
    cache.append(f);
  else:
    s=’|’+’=’*40 +’\n’
    for d in cache:
      print ’\n==|’,d.func_name,s,doc(d),d()
  return f

def reseed():
        seed = 1
        return random.seed(seed)

def say(mark):
  sys.stdout.write(mark)
  sys.stdout.flush()

def printlook(f):
  def wrapper(*lst): #tricks from Dr.Menzies
    ShowDate = datetime.datetime.now().strftime
    print "\n###", f.__name__, "#" * 50
    print "#", ShowDate("%Y−%m−%d %H:%M:%S")
```

```
    beginTime = time.time()
    x = f(*lst)
    endTime = time.time()
    print "\n" +("−"*60)
    # dump(Settings, f.__name__)
    print "\n# Runtime: %.3f secs" % (endTime−beginTime)
    x = [x, f.__name__]
    return x # return the searcher name and the results
  return wrapper

def dump(d, searchname, lvl = 0): # tricks from Dr. Menzies
  d = d if isinstance(d, dict) else d.__dict__
  callableKey, line , gap = [], "", " "*lvl
  for k in sorted(d.keys()):
    val= d[k]
    if isinstance(val, (dict, Options)):
      callableKey += [k]
    else:
      #if callable(val):
      # val = val.__name__
      line +=(" {0} :{1}".format(k, val))
  print gap + line
  for k in callableKey:
    if k ≡ searchname ∨ k ≡ "other":
      print gap + (" :{0} {1}".format(k, "options"))
      dump(d[k], lvl+1)
```