

GRAVITATIONAL SEARCH ALGORITHM



GROUP MEMBERS

creative portfolio

01

6410110287 BUNYAWEE LAONGPUN

02

6410110301 PATTITHA SUKSOMBOON

03

6410110475 WANVISA CHAIUEA

CALCULATING THE MASS



```
def massCalculation(fit,PopSize,M):
    Fmax = max(fit)
    Fmin = min(fit)
    Fsum = sum(fit)
    Fmean = Fsum/len(fit)

    if Fmax == Fmin:
        M = numpy.ones(PopSize)
    else:
        best = Fmin
        worst = Fmax

        for p in range(0,PopSize):
            M[p] = (fit[p]-worst)/(best-worst)

    Msum=sum(M)
    for q in range(0,PopSize):
        M[q] = M[q]/Msum

    return M
```

fit : ค่า fitness ของแต่ละอนุภาค

PopSize : จำนวนของสมาชิก

M : ค่ามวล (mass) ของแต่ละอนุภาค

Fmax : ค่า fitness ที่มากที่สุด

Fmin : ค่า fitness ที่น้อยที่สุด

Fsum : ผลรวมของค่า fitness

Fmean : ค่าเฉลี่ยของ fitness

best : ค่า fitness ที่ดีที่สุด

worst : ค่า fitness ที่แย่ที่สุด

$M[p] = (fit[p]-worst)/(best - worst)$: สูตร normalize fitness

เพื่อคำนวณมวลของอนุภาคแต่ละตัวที่ fitness



CALCULATING THE GRAVITATIONAL CONSTANT



```
def gConstant(l,iters):
    alfa = 20
    G0 = 100
    Gimd = numpy.exp(-alfa*float(l)/iters)
    G = G0*Gimd
    return G
```

l : หมายถึงหมายเลขของการวนลูป

iters : จำนวนรอบทั้งหมดที่อัลกอริธึมจะทำงาน

alfa : ค่าคงที่ที่ใช้ในการปรับความเร็วของการลดลงของค่า G ในแต่ละรอบ

G0 : ค่าคงที่เริ่มต้นของ Gravitational Constant ซึ่งตั้งค่าเป็น 100

Gim : ค่าที่คำนวณจากพังก์ชัน exponential เพื่อใช้ในการปรับลดค่าของ Gravitational Constant

G : Gravitational Constant ในแต่ละรอบ





CALCULATING THE FORCE AND ACCELERATION



```
def gField(PopSize,dim,pos,M,l,iters,G,ElitistCheck,Rpower):
    final_per = 2
    if ElitistCheck == 1:
        kbest = final_per + (1-l/iters)*(100-final_per)
        kbest = round(PopSize*kbest/100)
    else:
        kbest = PopSize

    kbest = int(kbest)
    ds = sorted(range(len(M)), key=lambda k: M[k],reverse=True)

    Force = numpy.zeros((PopSize,dim))
    # Force = Force.astype(int)

    for r in range(0,PopSize):
        for ii in range(0,kbest):
            z = ds[ii]
            R = 0
            if z != r:
                x=pos[r,:]
                y=pos[z,:]
                esum=0
                imval = 0
                for t in range(0,dim):
                    imval = ((x[t] - y[t])** 2)
                    esum = esum + imval

                R = math.sqrt(esum)

                for k in range(0,dim):
                    randnum=random.random()
                    Force[r,k] = Force[r,k]+randnum*(M[z])*((pos[z,k]-pos[r,k])/(R**Rpower+numpy.finfo(float).eps))

    acc = numpy.zeros((PopSize,dim))
    for x in range(0,PopSize):
        for y in range (0,dim):
            acc[x,y]=Force[x,y]*G
    return acc
```

PopSize: จำนวนอนุภาคในอัลกอริธึม

dim: จำนวนมิติของพื้นที่การค้นหา

pos: ตำแหน่งปัจจุบันของอนุภาค (อาร์เรย์ 2 มิติ)

M: มวลของอนุภาค (อาร์เรย์ 1 มิติ)

l: หมายเลขของการวนลูป (iteration)

iters: จำนวนรอบทั้งหมด

G: Gravitational Constant

ElitistCheck: ตัวบ่งชี้ว่าจะแสดงผลการคัดเลือกแบบเอลิกิต (elitist selection) หรือไม่

Rpower: พลังงานที่ใช้ในการคำนวณระยะห่าง



CALCULATING THE UPDATED POSITION



```
def move(PopSize,dim,pos,vel,acc):
    for i in range(0,PopSize):
        for j in range (0,dim):
            r1=random.random()
            r2=random.random()
            vel[i,j]=r1*vel[i,j]+acc[i,j]
            pos[i,j]=pos[i,j]+vel[i,j]

    return pos, vel
```

PopSize: จำนวนอนุภาค (particles) ในการค้นหา

dim: จำนวนมิติ (dimensions) ของพื้นที่การค้นหา

pos: ตำแหน่งปัจจุบันของอนุภาคในรูปแบบของอาเรย์ที่มีขนาด (PopSize, dim)

vel: ความเร็วปัจจุบันของอนุภาคในรูปแบบของอาเรย์ที่มีขนาด (PopSize, dim)

acc: ความเร่ง (acceleration) ที่คำนวณได้จากแรงดึงดูดในอัลกอริธึม



BENCHMARK



```
def F1(x):
    """ Spere Function """
    s=numpy.sum(x**2);
    return s
```



```
def getFunctionDetails(a):
    # [name, lb, ub, dim]
    param = { 0: ["F1",-100,100,30],
              }
    return param.get(a, "nothing")
```

Function F1(x):

x : ตัวแปรอินพุต ซึ่งจะถูกใช้ในการคำนวณพื้นที่ของลูกศร Spherical

s : ตัวแปรที่เก็บผลรวมของค่าต่างๆ ใน x ยกกำลังสอง

Function getFunctionDetails(a):

param : เก็บข้อมูลของพื้นที่ต่างๆ

100(lb) : ขอบเขตล่าง (lower bound) ของค่าที่จะทดสอบ

100(ub): ขอบเขตบน (upper bound) ของค่าที่จะทดสอบ

30(dim): จำนวนมิติ (dimensionality) ของพื้นที่ คือมี 30 มิติ



GSA



```
def GSA(objf,lb,ub,dim,PopSize,iters):
    # GSA parameters
    ElitistCheck =1
    Rpower = 1

    s=solution()

    """ Initializations """
    vel=numpy.zeros((PopSize,dim))
    fit = numpy.zeros(PopSize)
    M = numpy.zeros(PopSize)
    gBest=numpy.zeros(dim)
    gBestScore=float("inf")

    pos=numpy.random.uniform(0,1,(PopSize,dim)) *(ub-lb)+lb
    convergence_curve=numpy.zeros(iters)

    print("GSA is optimizing  "+"+objf.__name__+" ")

    timerStart=time.time()
    s.startTime=time.strftime("%Y-%m-%d-%H-%M-%S")
```

pos : ตำแหน่ง

vel : ความเร็ว

M : มวล

gBestScore : คะแนนที่ดีที่สุด

objf: พังก์ชันวัตถุประสงค์ที่ใช้ประเมินความเหมาะสมของแต่ละอุปกรณ์

lb, ub: ขอบเขตล่างและบนของค่าตำแหน่ง

dim: จำนวนมิติของปัญหา (dimension)

PopSize: ขนาดของประชากร

iters: จำนวนรอบการคำนวณ (iterations)

```
for l in range(0,iters):
    for i in range(0,PopSize):
        l1 = [None] * dim
        l1=numpy.clip(pos[i,:], lb, ub)
        pos[i,:]=l1

    #Calculate objective function for each particle
    fitness=[]
    fitness=objf(l1)
    fit[i]=fitness

    if(gBestScore>fitness):
        gBestScore=fitness
        gBest=l1

    """ Calculating Mass """
    M = massCalculation.massCalculation(fit,PopSize,M)

    """ Calculating Gravitational Constant """
    G = gConstant.gConstant(l,iters)

    """ Calculating Gfield """
    acc = gField.gField(PopSize,dim,pos,M,l,iters,G,ElitistCheck,Rpower)

    """ Calculating Position """
    pos, vel = move.move(PopSize,dim,pos,vel,acc)

    convergence_curve[l]=gBestScore

    if (l%1==0):
        print(['At iteration '+ str(l+1)+ ' the best fitness is '+
str(gBestScore)]);

    timerEnd=time.time()
    s.endTime=time.strftime("%Y-%m-%d-%H-%M-%S")
    s.executionTime=timerEnd-timerStart
    s.convergence=convergence_curve
    s.Algorithm="GSA"
    s.objectivefunc=objf.__name__
    return s
```

OPTIMIZER

```
● ● ●  
  
def selector(algo,func_details,popSize,Iter):  
    function_name=func_details[0]  
    lb=func_details[1]  
    ub=func_details[2]  
    dim=func_details[3]  
  
    if(algo==0):  
        x=gsa.GSA(getattr(benchmarks, function_name),lb,ub,dim,popSize,Iter)  
    return x  
  
# Select optimizers  
GSA=True # Code by Himanshu Mittal  
  
# Select benchmark function  
F1=True  
  
Algorithm=[GSA]  
objectivefunc=[F1]  
  
# Select number of repetitions for each experiment.  
# To obtain meaningful statistical results, usually 30 independent runs  
# are executed for each algorithm.  
Runs=1  
  
# Select general parameters for all optimizers (population size, number of iterations)  
PopSize = 50  
iterations= 1000  
  
#Export results ?  
Export=True  
  
#ExportToFile="YourResultsAreHere.csv"  
#Automatically generated name by date and time  
ExportToFile="experiment"+time.strftime("%Y-%m-%d-%H-%M-%S")+".csv"  
  
# Check if it works at least once  
atLeastOneIteration=False  
  
# CSV Header for for the cinvergence  
CnvgHeader=[]  
  
for i in range (0, len(Algorithm)):  
    for j in range (0, len(objectivefunc)):  
        if((Algorithm[i]==True) and (objectivefunc[j]==True)): # start experiment if an Algorithm and an objective function  
            is selected  
            for k in range (0,Runs):  
  
                func_details=benchmarks.getFunctionDetails(j)  
                x=selector(i,func_details,PopSize,iterations)  
                if(Export==True):  
                    with open(ExportToFile, 'a') as out:  
                        writer = csv.writer(out,delimiter=',')  
                        if (atLeastOneIteration==False): # just one time to write the header of the CSV file  
                            header=  
                            numpy.concatenate([["Optimizer","objfname","startTime","EndTime","ExecutionTime"],CnvgHeader])  
                            writer.writerow(header)  
  
                            a=numpy.concatenate([[x.Algorithm,x.objectivefunc,x.startTime,x.endTime,x.executionTime],x.convergence])  
                            writer.writerow(a)  
                            out.close()  
                            atLeastOneIteration=True # at least one experiment  
  
                        if (atLeastOneIteration==False): # Faild to run at least one experiment  
                            print("No Optimizer or Cost function is selected. Check lists of available optimizers and cost functions")
```

algo: เลขที่ใช้แทนอัลกอริธึม (ในโค๊ดนี้ค่า 0 แทน GSA)

func_details: รายละเอียดฟังก์ชันวัตถุประสงค์ที่เลือก (ชื่อฟังก์ชัน,
ขอบเขตล่างและบน, จำนวนมิติ)

Iter: จำนวนรอบการคำนวณ

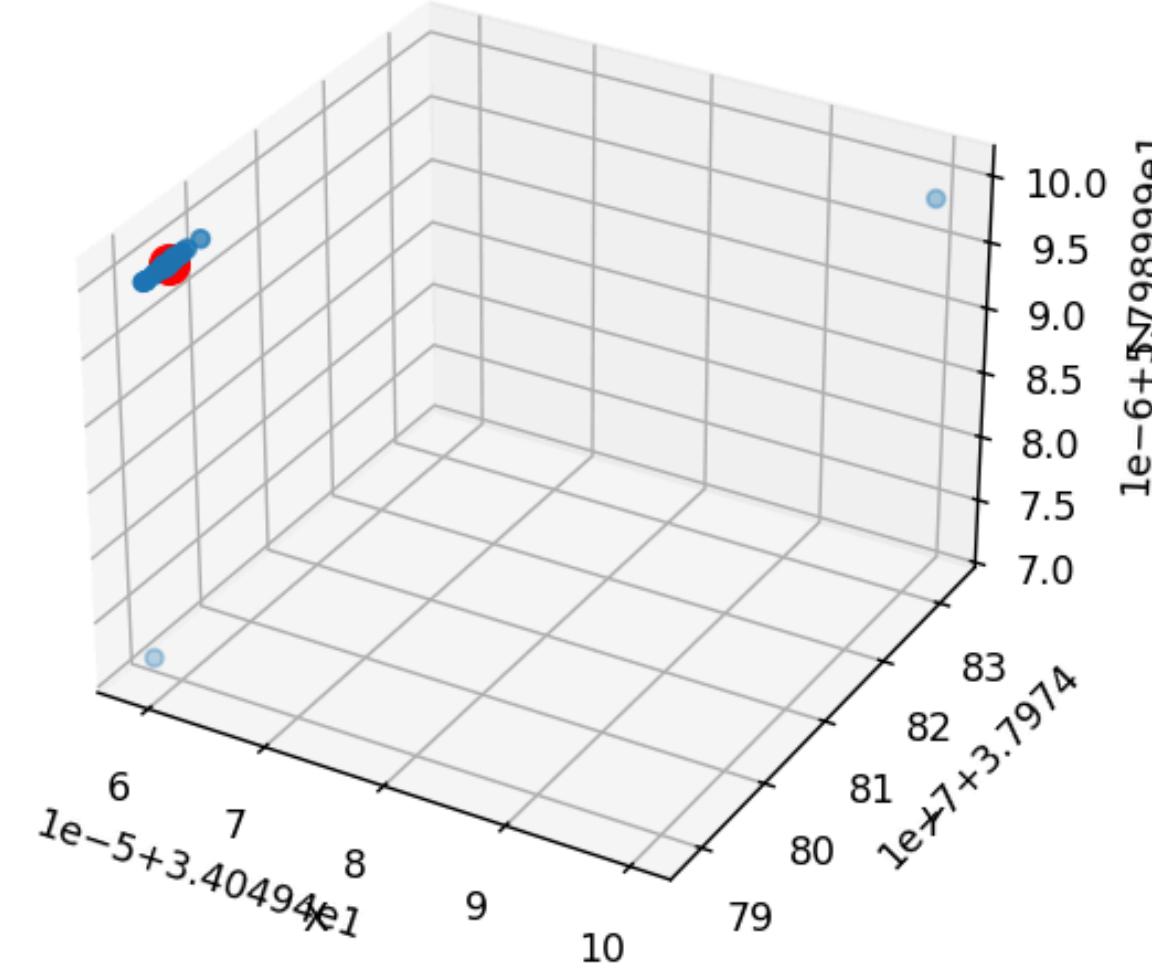
*หาก algo มีค่าเป็น 0 จะเรียกใช้ฟังก์ชัน GSA ผ่านโมดูล gsa

ព័ត៌មានខ្លួន

area	rooms	price
2104	3	399900
1600	3	329900
2400	3	369000
1416	2	232000
3000	4	539900
1985	4	299900
1534	3	314900
1427	3	198999
1380	3	212000
1494	3	242500
1940	4	239999



Iteration: 100



THANKYOU