Particle Swarm Optimization

粒子群最佳化

組員:羅紹華、陳建宏、范姜永岩

何謂粒子群最佳化?

粒子群演算法, 也稱粒子群優化演算法或鳥群覓食演算法(Particle Swarm Optimization), 縮寫為 PSO, 是近年來由J. Kennedy和R. C. Eberhart等開發的一種新的進化演算法(Evolutionary Algorithm - EA)。是**人類觀察鳥類覓食行為**所發展出來的演算法, 在「粒子群演算法」中一個粒子代表鳥群中的一隻鳥, 各個粒子具有「記憶性」並會參考其它粒子的「訊息」來決定移動的方向。

粒子群最佳化

PSO 初始化為一群隨機粒子(隨機解)。然後**通過迭代找到最佳解**。在每一次迭代中，粒子通過跟蹤兩個"極值"來更新自己。

第一個就是粒子本身所找到的最優解，這個解叫做**個體極值pBest**。

另一個極值是整個種群目前找到的最優解，這個極值是**全域性極值gBest**。

「粒子群演算法」能做什麼?

PSO 演算法屬於進化演算法的一種，它是通過適應度來評價解的品質，但它比基因演算法規則更為簡單，它沒有基因演算法的“交叉”(Crossover) 和“突變”(Mutation) 操作，它通過追隨當前搜尋到的最佳值來尋找最佳解。這種演算法以其實現容易、精度高、收斂快等優點引起了學術界的重視，並且在解決實際問題中展示了其優越性。粒子群演算法是一種用來解決**最佳化問題**的工具。

何謂最佳化問題?

Travelling Salesman Problem (TSP):

從「旅行推銷員問題」來看, 我們可以隨便決定一條路線 (城市的先後順序), 而這條路線會對應到一段距離 (歷經全部城市所需要的距離), 目標就是找到一條路線, 並且這條路線對應到的距離是最短的。

Scheduling problem:

「工作排程問題」也是差不多的, 我們有n個工作及m台機器, 每個工作在每台機器上的執行時間可能有所不同, 因此我們需要搜尋一個工作分配方式, 所需要花費的時間最短。

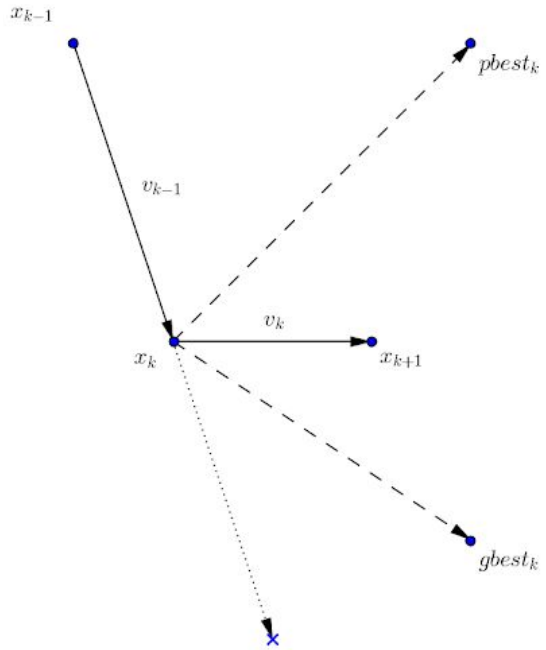
最佳化問題

對這類問題通常對會有一個最大的問題, 就是**要將全部可行的解 (路徑or工作分配方式) 在有限時間內計算完畢是不可能的**, 而此類問題大部份也都可能是個NP問題。

「粒子群演算法」就是為了解決這種問題的工具, 但做為折衷方案「粒子群演算法」**找到的解不保證是最佳解** (可能只是次佳解)。

演算法

目的:找出最佳解x , y 使目標函式值最小



k: iteration

x: position

pbest: personal best solution

gbest: global best solution

v: velocity

$$x_{k_{+1}} = x_k + v_k$$

$$v_{k+1} = w \cdot v_k + c_1 \cdot r_1 \cdot (pbest - x_k) + c_2 \cdot r_2 \cdot (gbest - x_k)$$

{W:慣性(記憶)權重　　c1,c2:加速權重　　r1,r2: 隨機值:[0,1]}

圖片來源:http://chenyuren.blogspot.com/2012/06/particle-swarm-optimization-in-r.html

參數:

目標函式: $f(x,y) = (x-20)^2 + (y-20)^2 + 1$

粒子個數:100個

迭代次數:100次

粒子:

```
class Particle():
    def __init__(self):
        self.position = np.array([(-1) ** (bool(random.getrandbits(1))) * \
                        random.random()*50, (-1)**(bool(random.getrandbits(1))) *\
                        random.random()*50])
        self.pbest_position = self.position
        self.pbest_value = float('inf')
        self.velocity = np.array([0,0])

    def move(self):
        self.position = self.position +self.velocity
```

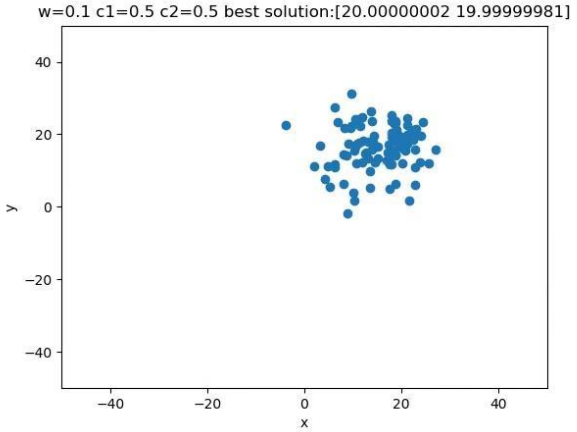
搜尋範圍:

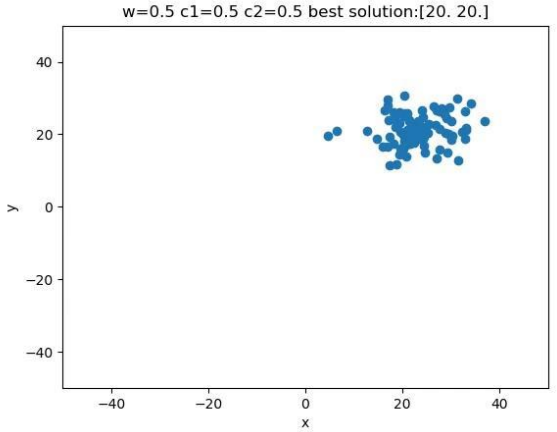
```
class Space():
    def __init__(self, n_particles):
        self.n_particles = n_particles
        self.particles = []
        self.gbest_position = np.array([random.random()*50, random.random()*50])
        self.gbest_value = float('inf')

    def fitness(self, particle):
        return (particle.position[0]-20)**2+(particle.position[1]-20)**2+1

    def update_pbest(self):
        for particle in self.particles:
            fitness = self.fitness(particle)
            if(fitness < particle.pbest_value):
                particle.pbest_value = fitness
                particle.pbest_position = particle.position

    def update_gbest(self):
        for particle in self.particles:
            fitness = self.fitness(particle)
            if(fitness < self.gbest_value):
                self.gbest_value = fitness
                self.gbest_position = particle.position
                print(self.gbest_value)

    def move_particles(self):
        for particle in self.particles:
            global W
            new_velocity = W*particle.velocity + \
                           c1*random.random()*(particle.pbest_position - particle.position) + \
                           c2*random.random()*(self.gbest_position - particle.position)
            particle.velocity = new_velocity
            particle.move()
```

開始搜尋:

```
iteration = 0
while(iteration < n_iterations):
    space.update_pbest()
    space.update_gbest()
    particles_frame.append([])
    for i in range(space.n_particles):
                particles_frame[iteration].append(list(space.particles[i].position))
    space.move_particles()
    iteration += 1
```

参数比較:


w=0.1 c1=0.5 c2=0.5 best solution:[20.00000002 19.99999981]
40
20
0
−20
−40
y
−40
−20
0
20
40
x


w=0.5 c1=0.5 c2=0.5 best solution:[20. 20.]
40
20
0
−20
−40
y
−40
−20
0
20
40
x

粒子群演算法(PSO) VS 基因演算法(GA)

共同特徵

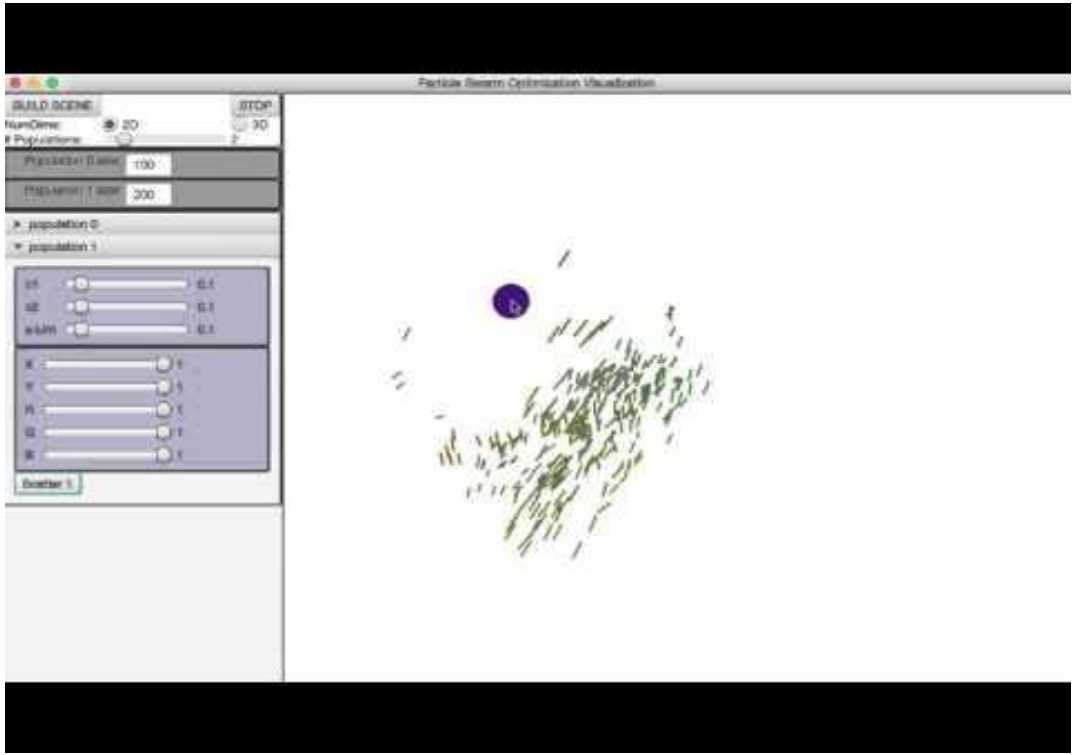
1. 都屬於仿生演算法
2. 都是解決最佳化問題的工具
3. 都使用適應度函數來評價系統

粒子群演算法(PSO) V.S. 基因演算法(GA)

差異

粒子群演算法(PSO)	基因演算法(GA)
比較簡單、快速	比較複雜(交配、突變)
有記憶體	沒有記憶體
單向的信息流動	染色體之間相互共享資訊
不具有淘汰機制	具有淘汰機制

粒子群演算法(PSO)視覺化



THANKS