人工智慧
用基因演算法解決背包問題

讓背包中的物品價值最高
—第五組

參數設定

1.初始染色體每個物品有30%的機率被拿取

2.適應函數:背包內總價值，若超重則歸零

3.終止條件:迭代50000次

4.採單點交配、突變率為10%，一個世代有20條染色體

※版本二每世代前六好染色體會直接放到下一代

程式碼部分

染色體物件與功能

```python
class Gene:
    def __init__(self):
        self.take = []  # 是否裝該編號的物品
        self.gene_weight = 0  # 背包內總重量
        self.gene_price = 0  # 背包內總價值
        for i in range(0, 30):
            if (
                random.randint(0, 9) <= 6
            ):
                self.take.append(0)  # 不拿該物品
            else:
                self.take.append(1)  # 拿該物品
        self.calculate()

    # 突變
    def mutation(self):
        location = random.randint(0, 29)  # 突變位置
        self.take[location] = (self.take[location] + 1) % 2  # 改變拿或不拿

    # 計算總重量與總價值
    def calculate(self):
        self.gene_price = 0
        self.gene_weight = 0  # 歸零
        for i in range(0, 30):
            if self.take[i] == 1:  # 判斷是否拿取該物品
                self.gene_weight += all_items[i].weight  # 總重量加上物品重量
                self.gene_price += all_items[i].price  # 總價值加上物品價值
```

程式碼部分

紀錄資料中物品

```
class Item:
    def __init__(self, index):
        self.name = data.iat[index, 1]  # 讀入物品名稱
        self.weight = data.iat[index, 2]  # 讀入物品重量
        self.price = data.iat[index, 3]  # 讀入物品價值
```

```
for i in range(0, 30):
    all_items.append(Item(i))    #建立物品數值陣列
```

程式碼部分

選擇要交配的染色體

```
#每個世代交配10次
for i in range(0, 10):
    #根據染色體價值比例抽取兩個相異的染色體
    random_num=random.randint(1, total_price)   #產生一個介於1到總價值的隨機數
    for j in gene_pool:
        random_num-=j.gene_price   #將隨機數逐一減去總價值(分母) 直到<=0
        if random_num<=0:
            gene_a=j    #選出該染色體a
            break
    gene_b=gene_a   #先把染色體b設成跟染色體a同一條
    #確保兩條染色體相異
    while(gene_b==gene_a):
        random_num=random.randint(1, total_price)
        for j in gene_pool:
            random_num-=j.gene_price
            if random_num<=0:
                gene_b=j    #選出該染色體b
                break
    gene_a=copy.deepcopy(gene_a)    #複製一個一樣的染色體
    gene_b=copy.deepcopy(gene_b)    #複製
```

程式碼部分

進行交配與突變，並更新下一代的資料庫

```
#染色體交配
swap_point=random.randint(0, 29)    #隨機選出兩染色體單點交配的起始點
gene_a.take[swap_point:30], gene_b.take[swap_point:30]=gene_b.take[swap_point:30], gene_a.take[swap_point:30]    #單點交配
#突變
if random.randint(0, 9)==0:    #10%機率a發生突變
    gene_a.mutation()   #呼叫突變函式
if random.randint(0, 9)==0:    #10%機率b發生突變
    gene_b.mutation()   #呼叫突變函式
#放進下一代基因庫
gene_a.calculate()  #計算重量跟價值
if gene_a.gene_weight>max_weight:
    gene_a.gene_price=0 #如果超重就把價值歸零
next_pool.append(gene_a)    #把新的染色體放進下一代的基因庫
gene_b.calculate()  #計算重量跟價值
if gene_b.gene_weight>max_weight:
    gene_b.gene_price=0 #如果超重就把價值歸零
next_pool.append(gene_b)    #把新的染色體放進下一代的基因庫
```

程式碼部分

世代交替

```
#世代交替
gene_pool=next_pool;     #把下一代的基因庫丟回來 原本的染色體們就用不到了
gene_pool.sort(key=lambda x: x.gene_price, reverse=True)     #排序(方便找出最大值)
next_pool=[]     #清空下一代基因庫
total_price=0    #歸零總價值重新計算
for i in gene_pool:
    total_price+=i.gene_price    #計算基因池內總價值
```

額外討論

隨機生成染色體

初始拿取機率若直接採用randint(0, 1)來決定染色體該位置是否拿取，會讓拿取機率變成50%，容易把過多東西都拿進背包，這樣背包內的物品重量就容易過高，使得需要重複生成染色體
效能不佳！

若機率過低會增加後續迭代次數
效能不佳!

經過考慮與測試我們決定以30%作為拿取的機率

```
for i in range(0, 30):
    if(random.randint(0,9)<=6):
        #隨機產生初始染色體是否拿取某物品(拿取機率30%)
        self.take.append(0) #不拿該物品
    else:
        self.take.append(1) #拿該物品
self.calculate()
```

兩種方法的比較
方法一:正規作法(每次迭代不複製前幾好)

由圖可知，在足夠的迭代次數便高機率可以達到全域最優解

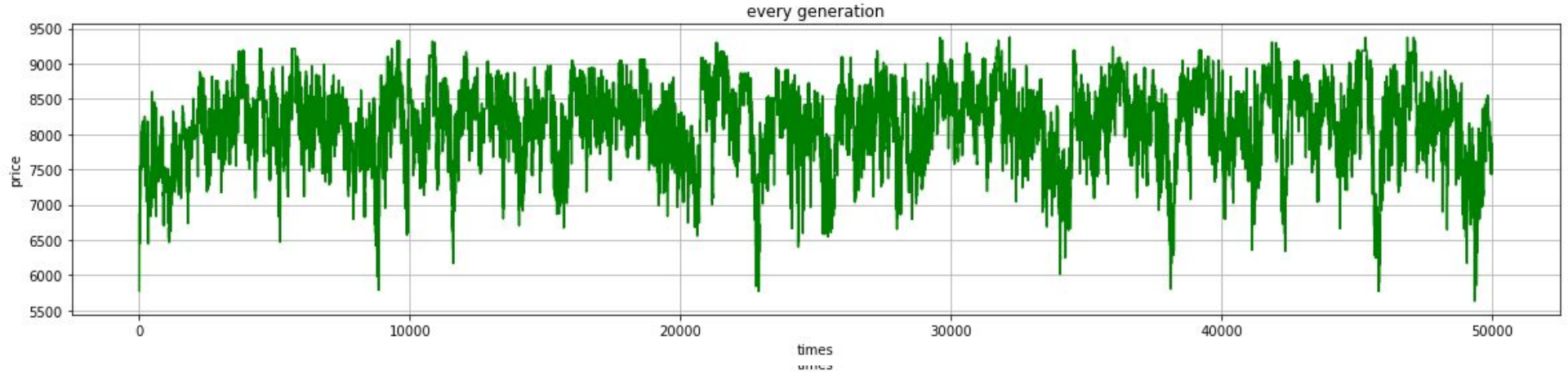
(gen為第一次達到該price的世代)

```
price: 9374 gen: 7458
generation: 10000
generation: 20000
generation: 30000
generation: 40000
generation: 50000
```

```
price: 9144 gen: 1369
price: 9334 gen: 5560
generation: 10000
price: 9374 gen: 17853
generation: 20000
generation: 30000
generation: 40000
generation: 50000
```

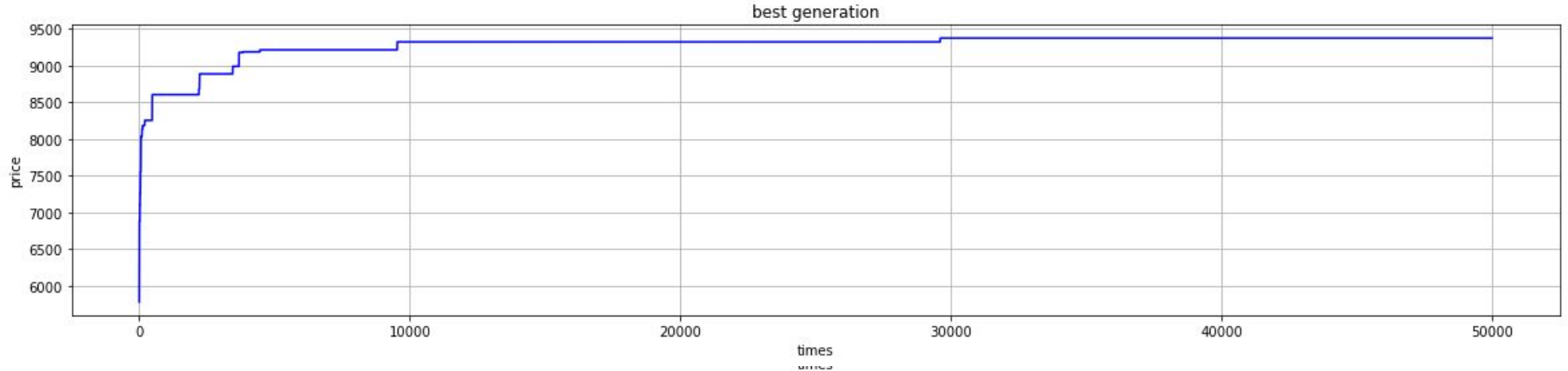
```
price: 9228 gen: 5113
price: 9298 gen: 8486
generation: 10000
price: 9334 gen: 17710
generation: 20000
price: 9374 gen: 29797
generation: 30000
generation: 40000
generation: 50000
```

每個generation都取一次值



因為每一個世代都是隨機的，所以price有上有下

當下最好的generation取一次值



price值遞增，次數夠多便可以達到全域最佳解

方法二: 每次迭代保留前面一代前幾優秀的基因

由圖可知，這種做法較不穩定，有時很快達到最佳解，有時卻達到局部最優解便很難繼續往上

```
price: 9298 gen: 2218
generation: 10000
generation: 20000
generation: 30000
generation: 40000
generation: 50000
```

```
price: 9298 gen: 2605
generation: 10000
generation: 20000
generation: 30000
generation: 40000
generation: 50000
```

```
price: 9204 gen: 5926
generation: 10000
generation: 20000
generation: 30000
generation: 40000
generation: 50000
```

```
generation: 30000
price: 9374 gen: 30105
generation: 40000
generation: 50000
```

```
price: 9021 gen: 72
price: 9238 gen: 76
price: 9334 gen: 117
price: 9374 gen: 194
```

```
price: 9374 gen: 1331
generation: 10000
generation: 20000
generation: 30000
generation: 40000
generation: 50000
```

只能達到局部最優解的原因:
迭代到後面都會是一模一樣的優秀染色體
兩條一樣的交配後高機率產生更多一樣的染色體

```
for i in range(0, 6):
    next_pool.append(copy.deepcopy(gene_pool[i]))    #將這一代前幾個優秀染色體直接複製到下一代
#每個世代交配7次 產生14個染色體
for i in range(0, 7):
    #根據染色體價值比例抽取兩個相異的染色體
    random_num=random.randint(1, total_price)    #產生一個介於1到總價值的隨機數
    for j in gene_pool:
        random_num-=j.gene_price    #將隨機數逐一減去總價值(分母) 直到<=0
        if random_num<=0:
            gene_a=j     #選出該染色體a
            break
```

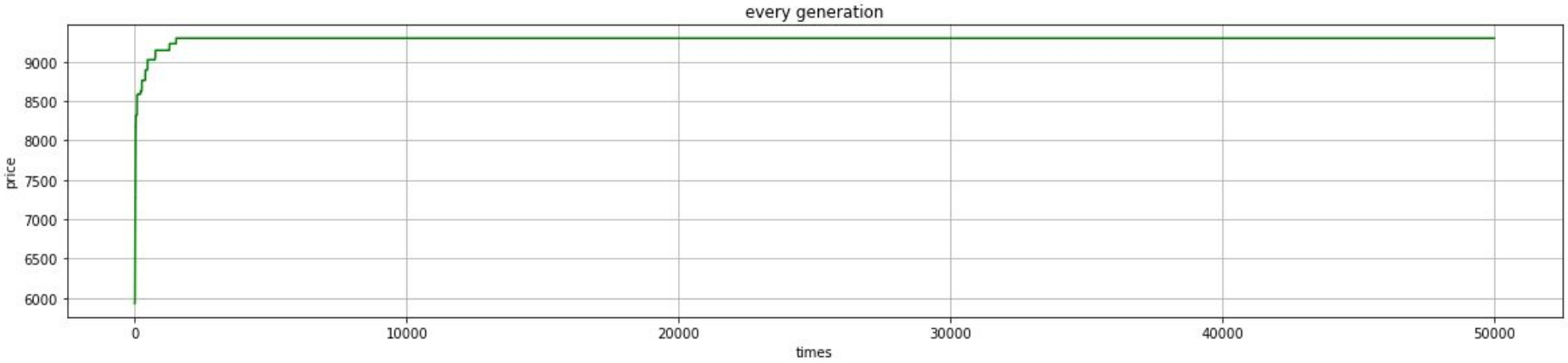
而突變只能一次突變一個的情況下
容易讓局部最優解的price值下降
導致無法被複製到下一個世代
形成由局部最優解構成的世代迴圈

```
#突變
if random.randint(0, 9)==0:     #10%機率a發生突變
    gene_a.mutation()    #呼叫突變函式
if random.randint(0, 9)==0:     #10%機率b發生突變
    gene_b.mutation()    #呼叫突變函式
```

可能的解決方法:判斷不要讓基因池出現過多一樣的染色體

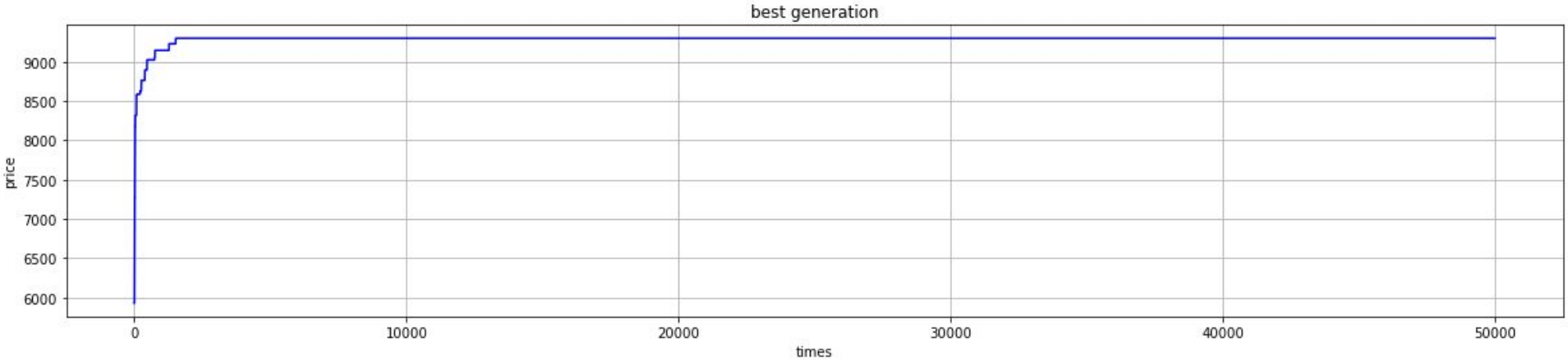
```
list     30 [1, 1, 0, 1, 1, 0, 1, 1, 1, 0, ...] False 5.take
```

每個generation都取一次值



因為會複製前面一代的優秀染色體, 因此每次迭代的最高price值會遞增

當下最好的generation取一次值



剛開始price值漲幅很大，很快就能達到局部最優解，但有可能達不到全域最優解

兩種方法的比較

方法一:中規中矩的做法，迭代次數足夠時能得到最佳解，但超重則歸零的做法可能有極低機率剛好全部都歸零，導致出錯，可以改變超重的處理方式例如重新產生新的一條染色體來解決。

方法二:由於每代的最優解會保存下來，以至於迭代速度較快，且不會產生都歸零的問題，順利的話可快速得出解，但有機會陷入局部最佳解，反而較難達到全域最佳