遺傳演算法求解背包問題

原創lilong117194 發佈於2017-12-26 21:03:28 閱讀數 7126 收藏

其實遺傳演算法是一種處理問題的思想，因為遺傳演算法整個體系都是在說對於一種問題的處理思路和原則，而不是一個具體的代碼編寫過程。

1. 演算法過程

關鍵步驟如下：

（1）基因編碼：在這個過程中，嘗試對一些個體的基因做一個描述，構造這些基因的結構，有點像確定函數引數的過程。

（2）設計初始群體：在這裡需要造一個種群出來，這些種群有很多生物個體但基因不同。

（3）適應度計算（剪枝）：這裡對那些不符合要求的後代進行剔除，不讓他們產生後代。否則他們產生的後代只會讓計算量更大而對逼近目標沒有增益。

（4）產生下一代：有3種方法，即：直接選擇，基因重組，基因突變

而後回到步驟（3）進行迴圈，適應度計算，產生下一代，這樣一代一代找下去，直到找到最優解為止。

遺傳演算法的應用領域：如TSP問題（旅行商問題），九宮問題，生產調度問題，背包問題等。

2. 背包問題

這裡用一個具體的示例來說明遺傳演算法的應用。

背包問題是一種組合優化的NP（多項式複雜程度的非確定性問題）完全問題，這類問題的特點很明顯，即“生成問題的一個解通常比驗證一個給定的解需要花更多的時間”。

背包問題的大意是，**有N件物品和一個容量為V的背包，第i件物品的重量是w[i]w[i]，價值是v[i]v[i]，求解將哪些物品裝入背包可使這些物品的重量總和不超過背包容量，且價值的總和最大。**

## 一. 0-1背包問題

給了一些物品的重量和價值，背包只能裝下一定重量的物品，求背包能裝物品的最大價值（**0-1背包問題約定每個物品最多只能用一次**）。

**二、完全背包問題**

**有N種物品和一個容量為V的背包，每種物品都有無限件可用。第i種物品的費用是c[i]，價值是w[i]。求解將哪些物品裝入背包可使這些物品的費用總和不超過背包容量，且價值總和最大。可以看出，兩種問題的不同是完全背包問題的物品可以無限用。**

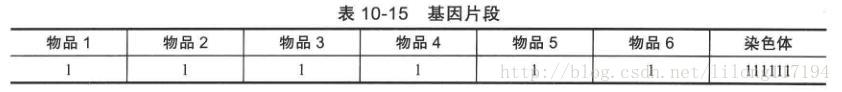
這種問題就是典型的NP問題，驗證一個猜想的解比算出一組解要快的多。

例：假設有一個背包，可以放置80公斤的物品，此外還有表：



這時如果用普通的演算法就有一種是窮舉法，這裡還可以實施，但如果有128種物品的話，就不能再用窮舉法了，這時遺傳演算法就派上用場了。

（1）基因編碼：一共有6種物品，每種物品的有無可以作為一個獨立的基因片段，如：



假如只有物品2，3，6，這時的染色體是：011001

（2）設計初始群體：

為了計算方便設置初始群體為4個初始生物個體，隨機產生。注意：這裡的初始個數的選擇要視具體情況而定，如果初始數量太少可能會導致在向量空間中覆蓋面積過小而導致收斂到了非最優解就終止了演算法。

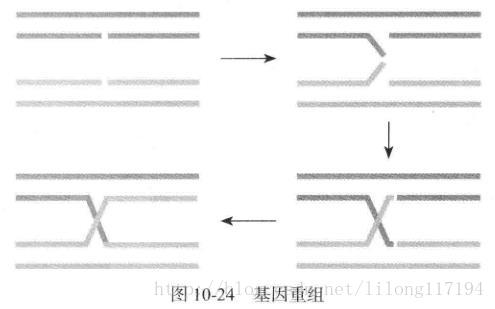
（3）適應度計算：首先適應度計算要用一個適應函數來做尺規，設計適應度的函數為物品的總價值。這裡同時還要計算物品的重量，超過80公斤的直接淘汰。然後對剩下的染色體進行一個用類似輪盤賭來進行遴選的過程，每次輪盤轉動中獎的基因組就允許繁殖一次，如果一次都沒中的基因組將無法得到延續。而遴選的原則是從生物多樣化中進行挑選，淘汰比較弱的是可以的，但不建議淘汰的比例太大。

（4）生產下一代

被輪盤賭選中的基因需要進行基因重組產生下一代，計算過程如下：



其實就是把基因片段從中間的某個地方斷開，然後交叉進行組合來形成新的基因，如：



這裡基因重組的斷開的位置是可以隨意進行的，同時注意不要一個基因自身和自身去做重組，沒有意義，因為怎樣重組還是自己。

這裡先暫不考慮基因突變的情況

（5）反覆運算計算

這裡一代一代用這種規則做下去，直接求重量和價值。這裡可以發現一個現象，就是適應度函數的值比上一代的適應度更好了。在反覆運算的過程中如果發現連續幾代的適應度函數值基本不增加或者甚至減少的情況，那說明函數已經收斂了。其實收斂的速度會受到很多因素而變化，如基因的長度，基因重組時的方案，基因變異的程度，每一代產生個體的數量等。

3. 代碼實現

# coding=utf-8

import random

#背包問題

# 物品 重量 價格

X = {1: [10, 15],

2: [15, 25],

3: [20, 35],

4: [25, 45],

5: [30, 55],

6: [35, 70]}

#終止界限

FINISHED\_LIMIT = 5

#重量界限

WEIGHT\_LIMIT = 80

#染色體長度

CHROMOSOME\_SIZE = 6

#遴選次數

SELECT\_NUMBER = 4

max\_last = 0

diff\_last = 10000

#判斷退出

def is\_finished(fitnesses):

global max\_last

global diff\_last

max\_current = 0

for v in fitnesses:

if v[1] > max\_current:

max\_current = v[1]

print 'max\_current:',max\_current # 得到當前最大的價值

diff = max\_current - max\_last # 價值差，也就是適應度的改變的大小

# 這裡判斷連續兩代的改變量如果都小於5，則停止反覆運算

if diff < FINISHED\_LIMIT and diff\_last < FINISHED\_LIMIT:

return True

else:

diff\_last = diff

max\_last = max\_current

return False

#初始染色體樣態

def init():

chromosome\_state1 = '100100'

chromosome\_state2 = '101010'

chromosome\_state3 = '010101'

chromosome\_state4 = '101011'

chromosome\_states = [chromosome\_state1,

chromosome\_state2,

chromosome\_state3,

chromosome\_state4]

return chromosome\_states

#計算適應度

def fitness(chromosome\_states):

fitnesses = []

for chromosome\_state in chromosome\_states: # 遍歷所有的染色體

value\_sum = 0 # 物品重量

weight\_sum = 0 # 物品價值

# 將一個可遍歷的資料物件組合為一個索引序列，同時列出資料和資料下標

for i, v in enumerate(chromosome\_state):

# 對染色體中的1，即存在的物品體重和價格求和

if int(v) == 1:

weight\_sum += X[i + 1][0]

value\_sum += X[i + 1][1]

fitnesses.append([value\_sum, weight\_sum])

return fitnesses

#篩選

def filter(chromosome\_states, fitnesses):

#重量大於80的被淘汰

index = len(fitnesses) - 1

while index >= 0:

index -= 1

if fitnesses[index][1] > WEIGHT\_LIMIT:

chromosome\_states.pop(index) # 彈出不符合條件的染色體

fitnesses.pop(index) # 彈出不符合條件的適應度

#print chromosome\_states,'\n',fitnesses

#遴選

selected\_index = [0] \* len(chromosome\_states) # 如果[0]\*3得到的是[0,0,0]

for i in range(SELECT\_NUMBER):

# 隨機選擇染色體，然後得到相應的索引

j = chromosome\_states.index(random.choice(chromosome\_states))

selected\_index[j] += 1

return selected\_index

# 交叉產生下一代

def crossover(chromosome\_states, selected\_index):

chromosome\_states\_new = []

index = len(chromosome\_states) - 1

#print 'index:',index

while index >= 0: # 遍歷完所有的染色體組的染色體（其中下標-1代表最後一個染色體的索引）

print 'index:',index

index -= 1

chromosome\_state = chromosome\_states.pop(index)

print 'chromosome\_states\_3:',chromosome\_states # 彈出後的染色體組

print 'chromosome\_state:',chromosome\_state # 彈出的染色體

for i in range(selected\_index[index]):

chromosome\_state\_x = random.choice(chromosome\_states) # 隨機選擇一個染色體

print 'chromosome\_state\_x:',chromosome\_state\_x

pos = random.choice(range(1, CHROMOSOME\_SIZE - 1)) # 隨機[1, 2, 3, 4]其中的一個數

print 'pos:',pos

chromosome\_states\_new.append(chromosome\_state[:pos] + chromosome\_state\_x[pos:])

print 'chromosome\_states\_new:',chromosome\_states\_new

chromosome\_states.insert(index, chromosome\_state) # 恢復原染色體組

print 'chromosome\_states\_4:', chromosome\_states

return chromosome\_states\_new # 返回得到的新的染色體組

if \_\_name\_\_ == '\_\_main\_\_':

# 初始群體

chromosome\_states = init() # 是全域的

print 'chromosome\_states:',chromosome\_states

n = 100 # 反覆運算次數

while n > 0:

n -= 1

#適應度計算

fitnesses = fitness(chromosome\_states)

#print 'fitnesses:',fitnesses

if is\_finished(fitnesses):

break # 如果符合條件，立刻停止迴圈

print '1:', fitnesses

#遴選

selected\_index = filter(chromosome\_states, fitnesses)

print '2:', selected\_index

print 'chromosome\_states\_2:',chromosome\_states

#產生下一代

chromosome\_states = crossover(chromosome\_states, selected\_index)

print '3:', chromosome\_states

print str(n)+'..................................' # 反覆運算次數

fitnesses = fitness(chromosome\_states)

print 'fitnesses:',fitnesses

print chromosome\_states

運行結果：

chromosome\_states: ['100100', '101010', '010101', '101011']

max\_current: 95

1: [[60, 35], [105, 60], [140, 75], [175, 95]]

2: [1, 2, 1]

chromosome\_states\_2: ['100100', '101010', '010101']

index: 2

chromosome\_states\_3: ['100100', '010101']

chromosome\_state: 101010

chromosome\_state\_x: 010101

pos: 4

chromosome\_states\_new: ['101001']

chromosome\_state\_x: 100100

pos: 1

chromosome\_states\_new: ['101001', '100100']

chromosome\_states\_4: ['100100', '101010', '010101']

index: 1

chromosome\_states\_3: ['101010', '010101']

chromosome\_state: 100100

chromosome\_state\_x: 010101

pos: 4

chromosome\_states\_new: ['101001', '100100', '100101']

chromosome\_states\_4: ['100100', '101010', '010101']

index: 0

chromosome\_states\_3: ['100100', '101010']

chromosome\_state: 010101

chromosome\_state\_x: 101010

pos: 2

chromosome\_states\_new: ['101001', '100100', '100101', '011010']

chromosome\_states\_4: ['100100', '010101', '101010']

3: ['101001', '100100', '100101', '011010']

99..................................

max\_current: 70

1: [[120, 65], [60, 35], [130, 70], [115, 65]]

2: [1, 2, 0, 1]

chromosome\_states\_2: ['101001', '100100', '100101', '011010']

index: 3

chromosome\_states\_3: ['101001', '100100', '011010']

chromosome\_state: 100101

chromosome\_states\_4: ['101001', '100100', '100101', '011010']

index: 2

chromosome\_states\_3: ['101001', '100101', '011010']

chromosome\_state: 100100

chromosome\_state\_x: 011010

pos: 2

chromosome\_states\_new: ['101010']

chromosome\_state\_x: 011010

pos: 2

chromosome\_states\_new: ['101010', '101010']

chromosome\_states\_4: ['101001', '100100', '100101', '011010']

index: 1

chromosome\_states\_3: ['100100', '100101', '011010']

chromosome\_state: 101001

chromosome\_state\_x: 100100

pos: 2

chromosome\_states\_new: ['101010', '101010', '100100']

chromosome\_states\_4: ['101001', '100100', '100101', '011010']

index: 0

chromosome\_states\_3: ['101001', '100100', '100101']

chromosome\_state: 011010

chromosome\_state\_x: 100101

pos: 4

chromosome\_states\_new: ['101010', '101010', '100100', '011001']

chromosome\_states\_4: ['101001', '100100', '011010', '100101']

3: ['101010', '101010', '100100', '011001']

98..................................

max\_current: 70

fitnesses: [[105, 60], [105, 60], [60, 35], [130, 70]]

['101010', '101010', '100100', '011001']

從運行結果可以看出詳細的運行過程，但每次的運行結果不同，如果基於代碼中的各種條件設置，最終的效果應該是最大的適應度（總價值）為130，即物品2，3，6的組合。

這裡用的收斂條件是連續兩代的適應函數最大值不再增加，出現這種情況則判斷為收斂。

4. 筆記

1. Python enumerate() 函數：

enumerate() 函數用於將一個可遍歷的資料物件(如清單、元組或字串)組合為一個索引序列，同時列出資料和資料下標，一般用在 for 迴圈當中。

enumerate(sequence, [start=0])：

sequence – 一個序列、反覆運算器或其他支援反覆運算物件。

start – 下標起始位置。

返回 enumerate(枚舉) 對象

示例：

>>> seasons = ['Spring', 'Summer', 'Fall', 'Winter']

>>> tt=enumerate(seasons)

<enumerate object at 0x00000000063B1630>

>>> tt

<enumerate object at 0x00000000063B1678>

>>> list(tt)

[(0, 'Spring'), (1, 'Summer'), (2, 'Fall'), (3, 'Winter')]

>>> list(enumerate(seasons, start=1)) # 小標從 1 開始

[(1, 'Spring'), (2, 'Summer'), (3, 'Fall'), (4, 'Winter')]

用在for迴圈中：

普通的for迴圈

>>> i = 0

>>> seq = ['one', 'two', 'three']

>>> for element in seq:

... print i, seq[i]

... i +=1

...

0 one

1 two

2 three

>>>

for 迴圈使用 enumerate

>>>seq = ['one', 'two', 'three']

>>> for i, element in enumerate(seq):

... print i, seq[i]

...

0 one

1 two

2 three

>>>

參考：http://www.runoob.com/python/python-func-enumerate.html

2.

chromosome\_states.index(random.choice(chromosome\_states))使用：

>>> chromosome\_states=['100100', '101010', '010101']

>>> import random

>>>chromosome\_states.index(random.choice(chromosome\_states))

1

>>> chromosome\_states.index(random.choice(chromosome\_states))

0

>>> random.choice(chromosome\_states)

'101010'

>>> chromosome\_states.index(random.choice(chromosome\_states))

2

>>> chromosome\_states.index(random.choice(chromosome\_states))

0

>>> chromosome\_states.index(random.choice(chromosome\_states))

0

>>>

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

參考：《白話大數據與機器學習》

————————————————

版權聲明：本文為CSDN博主「lilong117194」的原創文章，遵循 CC 4.0 BY-SA 版權協議，轉載請附上原文出處連結及本聲明。

原文連結：https://blog.csdn.net/lilong117194/article/details/78906488