Algoritmos Genéticos

Trabajo Práctico número 2

Algoritmos Genéticos

UTN - FrRo

Mates, 8 de Agosto de 2017

**Algoritmos Genéticos**

Institución: Universidad Tecnológica Nacional, Facultad Regional Rosario.

Año: 2017.

Numero de Trabajo: 02.

Docentes:

Díaz, Daniela.

Lombardo, Víctor.

Alumnos:

Ramos, Lautaro (ramos.lauty@gmail.com).

Sgreccia, Pablo (pablosgreccia@gmail.com).

Djemdjemian, Ezequiel (ezequiel.djc@gmail.com).

**Índice:**

Enunciado - - - - - - - 3

Aclaraciones de la Resolución - - - - 4

Código

mochila.py - - - - - - 5

mochilaTres.py - - - - - - 8

Resultados - - - - - - - 10

Conclusiones - - - - - - - 11

**Enunciado del Problema:**

Consiste en elegir, de entre un conjunto de **n** elementos, (cada uno con un valor **$i**, y un volumen **Vi**), aquellos que puedan ser cargados en una mochila de volumen **V** de manera que el valor obtenido sea máximo. Resolver el siguiente problema:

¿Cuáles son los elementos de la lista siguiente que cargaremos en una mochila de 4200 cm3 de manera que su valor en $ sea máximo?

|  |  |  |
| --- | --- | --- |
| Objeto | Volumen [Cm3] | Valor [$] |
| 1 | 150 | 20 |
| 2 | 325 | 40 |
| 3 | 600 | 50 |
| 4 | 805 | 36 |
| 5 | 430 | 25 |
| 6 | 1200 | 64 |
| 7 | 770 | 54 |
| 8 | 60 | 18 |
| 9 | 930 | 46 |
| 10 | 353 | 28 |

Para su solución utilizar un procedimiento exhaustivo que consiste en evaluar para cada subconjunto de elementos el valor correspondiente y, posteriormente, clasificando los subconjuntos por su valor de mayor a menor, encontrar cual es el subconjunto solución.

**Ejercicios:**

1. Resolver el problema de la mochila utilizando una búsqueda exhaustiva.
2. Resolver el ejercicio anterior usando el algoritmo *greedy* y comentar su similitud o no con el exhaustivo
3. En algunas ocasiones planteamos el problema de la mochila teniendo en cuenta los pesos en lugar del volumen. Luego, dados 3 elementos, cuyos pesos son: **1800** grs., **600** grs., y **1200** grs. y cuyos valores son: $**72**, $**36** y $**60** respectivamente, y con una mochila que puede soportar hasta **3000** grs. se pide:
4. Hallar una solución utilizando un algoritmo *goloso* y exhaustivo
5. Analizar dicha solución respecto a su grado de optimización y elaborar las conclusiones que considere adecuadas.

**Descripciones y Aclaraciones de la resolución:**

Al igual que en el primer trabajo practico, resolvimos el problema dado utilizando Python.

Desarrollamos el trabajo en dos programas separados, para simplificar la corrección, aunque la forma de resolverla es idéntica. Uno de los programas (mochila.py) resuelve los ítems 1) y 2). Mientras que el otro (mochilaTres.py) resuelve el ítem 3 completo.

¿Cómo funciona la resolución utilizando la *búsqueda exhaustiva*?

Optamos por codificar los elementos en un array binario (solo conformado por 0s y 1s), de longitud igual a la cantidad de elementos (n).

Si en la posición i de dicho array hay un ‘0’, significa que el i-esimo elemento del array, no es candidato a estar dentro de la mochila. Por el contrario, si es un ‘1’, indica que si es candidato. De esta forma, existen 2n posibles arrays diferentes, lo que coincide con la cantidad de combinaciones de objetos candidatos a estar en la mochila.

Para llenar este array, realizamos un *for* desde 0 hasta 2n -1, rellenando el arreglo con cada valor que toma la variable del bucle, convertida a binario.

Luego recorremos las 2n posibles combinaciones, evaluando para cada una que volumen ocuparía. Si ese volumen es menor o igual al volumen de la mochila, recién ahí se calcula que valor [$] tendría dicha combinación. Si dicho valor es mayor al de las anteriores combinaciones, guardamos la posición del array en la que está dicha combinación, y su valor.

¿Cómo funciona la resolución utilizando el algoritmo *Greedy*?

Utilizamos una función fitness que para cada elemento de la mochila evalúa el “rendimiento de este”, obteniendo una relación Volumen/Valor, o Peso/Valor (en el ítem 3).

Luego de obtener dicho fitness, solo es cuestión de ordenar los elementos, basados en su fitness, e ir “metiéndolos” en la mochila mientras quepan.

**Código:**

mochila.py:

import math

objetos = [

{"volumen": 150, "valor": 20},

{"volumen": 325, "valor": 40},

{"volumen": 600, "valor": 50},

{"volumen": 805, "valor": 36},

{"volumen": 430, "valor": 25},

{"volumen": 1200, "valor": 64},

{"volumen": 770, "valor": 54},

{"volumen": 60, "valor": 18},

{"volumen": 930, "valor": 46},

{"volumen": 353, "valor": 28}

]

mochila\_volumen = 4200

num\_elementos = len(objetos)

def int2bin(int, cant\_digitos):

return ('{0:0' + str(cant\_digitos) +'b}').format(int)

def volumen\_total(combinacion):

volumenCombinacion = 0

for i in range(len(combinacion)):

if(combinacion[i] == '1'):

volumenCombinacion += int(objetos[i]["volumen"])

return volumenCombinacion

def valor\_total(combinacion):

valorCombinacion = 0

for i in range(len(combinacion)):

if (combinacion[i] == '1'):

valorCombinacion += int(objetos[i]["valor"])

return valorCombinacion

def funFitness (vol,val):

return (float(vol)/float(val))

combinaciones = []

for i in range(int(math.pow(2, num\_elementos))):

combinacion = int2bin(i, num\_elementos)

combinaciones.append(combinacion)

pos\_mayor = 0

valor\_mayor = 0

for i in range(len(combinaciones)):

if(volumen\_total(combinaciones[i]) <= mochila\_volumen):

if(valor\_total(combinaciones[i]) > valor\_mayor):

pos\_mayor = i

valor\_mayor = valor\_total(combinaciones[i])

print

print "EXHAUSTIVA: "

print

# print "Combinacion Exhaustiva: " + str (combinaciones[pos\_mayor])

print "Los elementos que entran en la mochila son: "

for i in range (len(combinaciones[pos\_mayor])):

if str(combinaciones[pos\_mayor])[i] == '1':

print " El elemento " + str(i+1) + " esta dentro."

print " Volumen:" + str(objetos[i]["volumen"]) + ". Valor: " + str(objetos[i]["valor"])

print "Volumen Exhaustivo: " + str(volumen\_total(combinaciones[pos\_mayor]))

print "Valor Exhaustivo: " + str(valor\_mayor)

print "--------------------------------"

# algoritmo greedy

# evaluo una funcion fitness para cada elemento

fitness = []

for i in range (num\_elementos):

fitness.append( {"fit":funFitness(objetos [i]["volumen"], objetos [i]["valor"]) , "pos": i} )

# Ordeno los elementos en forma decreciente

fitness.sort(reverse=True)

# Para simplificar creo un nuevo arreglo, que contenga solamente las posiciones de los mejores fitness, de mayor a menor

fitnessOnly = []

for i in range (num\_elementos):

fitnessOnly.append(fitness[i]["pos"])

combinacionesGreedy = []

for i in (fitnessOnly):

combinacionesGreedy.append(0)

volumenAcumuladoGreedy = 0

valorAcumuladoGreedy = 0

i=0

for i in fitnessOnly:

if volumenAcumuladoGreedy + objetos[i]["volumen"] <= mochila\_volumen:

volumenAcumuladoGreedy += objetos[i] ["volumen"]

combinacionesGreedy = combinacionesGreedy[:i] + [1] + combinacionesGreedy[i+1:]

valorAcumuladoGreedy += objetos[i]["valor"]

print

print "GREEDY"

print

# print "Combinacion de elementos Greedy: " + str(combinacionesGreedy)

print "Los elementos que entran en la mochila son: "

for i in range (len(combinacionesGreedy)):

if combinacionesGreedy[i] == 1:

print " El elemento " + str(i+1) + " esta dentro."

print " Volumen:" + str(objetos[i]["volumen"]) + ". Valor: " + str(objetos[i]["valor"])

print "Volumen Greedy: " + str(volumenAcumuladoGreedy)

print "Valor Greedy: " + str(valorAcumuladoGreedy)

print

mochilaTres.py:

import math

objetos = [

{"peso": 1800, "valor": 72},

{"peso": 600, "valor": 36},

{"peso": 1200, "valor": 60},

]

def int2bin(int, cant\_digitos):

return ('{0:0' + str(cant\_digitos) +'b}').format(int)

def peso\_total(combinacion):

pesoCombinacion = 0

for i in range(len(combinacion)):

if(combinacion[i] == '1'):

pesoCombinacion += int(objetos[i]["peso"])

return pesoCombinacion

def valor\_total(combinacion):

valorCombinacion = 0

for i in range(len(combinacion)):

if (combinacion[i] == '1'):

valorCombinacion += int(objetos[i]["valor"])

return valorCombinacion

def funFitness (pes,val):

return (float(pes)/float(val))

pesoMochila = 3000;

numElementos = len(objetos)

combinaciones = []

for i in range(int(math.pow(2, numElementos))):

combinaciones.append(int2bin(i,numElementos))

pos\_mayor = 0

valor\_mayor = 0

for i in range(len(combinaciones)):

if(peso\_total(combinaciones[i]) <= pesoMochila):

if(valor\_total(combinaciones[i]) > valor\_mayor):

pos\_mayor = i

valor\_mayor = valor\_total(combinaciones[i])

print

print "EXHAUSTIVA: "

print

# print "Combinacion Exhaustiva: " + str (combinaciones[pos\_mayor])

print "Los elementos que entran en la mochila son: "

for i in range (len(combinaciones[pos\_mayor])):

if str(combinaciones[pos\_mayor])[i] == '1':

print " El elemento " + str(i+1) + " esta dentro."

print " Peso:" + str(objetos[i]["peso"]) + ". Valor: " + str(objetos[i]["valor"])

print "Peso Exhaustivo: " + str(peso\_total(combinaciones[pos\_mayor]))

print "Valor Exhaustivo: " + str(valor\_mayor)

print "--------------------------------"

# GREEDY

fitness = []

for i in range (numElementos):

fitness.append( {"fit":funFitness(objetos [i]["peso"], objetos [i]["valor"]) , "pos": i} )

# Ordeno los elementos en forma decreciente

fitness.sort(reverse=True)

fitnessOnly = []

for i in range (numElementos):

fitnessOnly.append(fitness[i]["pos"])

# print fitnessOnly

combinacionesGreedy = []

# mejorar|

for i in (fitnessOnly):

combinacionesGreedy.append(0)

pesoAcumuladoGreedy = 0

valorAcumuladoGreedy = 0

i=0

for i in fitnessOnly:

if pesoAcumuladoGreedy + objetos[i]["peso"] <= pesoMochila:

pesoAcumuladoGreedy += objetos[i] ["peso"]

combinacionesGreedy = combinacionesGreedy[:i] + [1] + combinacionesGreedy[i+1:]

valorAcumuladoGreedy += objetos[i]["valor"]

print

print "GREEDY"

print

# print "Combinacion de elementos Greedy: " + str(combinacionesGreedy)

print "Los elementos que entran en la mochila son: "

for i in range (len(combinacionesGreedy)):

if combinacionesGreedy[i] == 1:

print " El elemento " + str(i+1) + " esta dentro."

print " Peso:" + str(objetos[i]["peso"]) + ". Valor: " + str(objetos[i]["valor"])

print "Peso Greedy: " + str(pesoAcumuladoGreedy)

print "Valor Greedy: " + str(valorAcumuladoGreedy)

**Resultados:**

1. Resultado utilizando exhaustiva con 10 elementos:

Elementos: 1, 2, 3, 5, 6, 7, 8 y 10 (todos, excepto 4 y 9)

Volumen: 3888 [Cm3] (de 4200 cm3 disponibles)

Valor: 299 [$]

1. Resultado utilizando Greedy con 10 elementos:

Elementos: 4, 5, 6, 7, 8, 9 y 10.

Volumen: 4195 [Cm3] (de 4200 cm3 disponibles)

Valor: 243 [$]

a. Resultado utilizando exhaustiva con 3 elementos:

Elementos: 1 y3

Peso: 3000 [grs] (de 3000 grs. disponibles. Ocupa el total de la mochila)

Valor: 132 [$]

b. Resultado utilizando exhaustiva con 3 elementos:

Elementos: 1 y3

Peso: 3000[grs] (de 3000 grs. disponibles. Ocupa el total de la mochila)

Valor: 132 [$]

**Conclusiones:**

La conclusión a la que pudimos llegar es que, al aumentar el número de objetos a ser analizados, se empiezan a notar diferencias entre los dos métodos. Cuando tenemos 3 objetos, no existe diferencia entre los métodos, pero cuando son 10, ya notamos un cambio en los resultados, además de que en algún punto se vuelve imposible realizar una búsqueda exhaustiva por el mejor resultado debido al tiempo de procesamiento que eso conllevaría.

Podríamos afirmar que el algoritmo greedy es la solución más decente en la mayoría de los casos, ya que cuando son muchos cálculos lo hace rápido con un resultado medianamente bueno, y cuando son pocos el resultado es igual (o casi) al de la búsqueda exhaustiva.