Algoritmos Genéticos

UNIVERSIDAD TECNOLÓGICA NACIONAL

FACULTAD REGIONAL ROSARIO

Ornella Corsetti: orcorsetti@gmail.com

Santiago Ciriaci: santiagociriaci@gmail.com

Andrés Botello: andresjbotello@gmail.com

GRUPO 2 Ciclo Lectivo 2018

TRABAJO PRACTICO 2 – PROBLEMA DE LA MOCHILA.

INDICE

ENUNCIADO

Cuales son los elementos de la lista siguiente que cargaremos en una mochila de 4200 cm^3 de manera que su valor en $ sea máximo.

Objeto	Volumen (cm^3)	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

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, $60 respectivamente, y con una mochila que puede soportar hasta 3000 grs. Se pide:
    A. Hallar una solución utilizando un algoritmo goloso y exhaustivo.
    B. Analizar dicha solución respecto a su grado de optimización y elaborar las conclusiones que considere adecuadas.

PUNTO 1.

CODIGO.

```
lista=[[150],[325],[600],[805],[430],[1200],[770],[60],[930],[353]]
listavol=[150,325,600,805,430,1200,770,60,930,353]
listapre=[[150,20],[325,40],[600,50],[805,36],[430,25],[1200,64],[770,54],[60,
18],[930,46],[353,28]]
combinaciones=[]

for i in range(len(lista)): #Crea Lista de combinaciones.
    combinaciones.append([])

for a in range(len(lista)): #Crea todas las combinaciones posibles.
    combinaciones[0].append(lista[a])
    for b in range(a+1,len(lista)):
        combinaciones[1].append(lista[a] + lista[b])
        for c in range(b+1,len(lista)):
            combinaciones[2].append(lista[a] + lista[b] + lista[c])
            for d in range(c+1,len(lista)):
                combinaciones[3].append(lista[a] + lista[b]+ lista[c]+
lista[d])
                for e in range(d+1,len(lista)):
                    combinaciones[4].append(lista[a] + lista[b]+ lista[c]+
lista[d]+ lista[e])
                    for f in range (e+1,len(lista)):
                        combinaciones[5].append(lista[a] + lista[b]+ lista[c]+
lista[d]+ lista[e]+lista[f])
                        for g in range (f+1,len(lista)):
                            combinaciones[6].append(lista[a] + lista[b]+
lista[c]+ lista[d]+ lista[e]+lista[f]+lista[g])
                            for h in range (g+1,len(lista)):
                                combinaciones[7].append(lista[a] + lista[b]+
lista[c]+ lista[d]+ lista[e]+lista[f]+lista[g]+lista[h])
                                for i in range (h+1,len(lista)):
                                    combinaciones[8].append(lista[a] +
lista[b]+ lista[c]+ lista[d]+ lista[e]+lista[f]+lista[g]+lista[h]+lista[i])
                                    for j in range (i+1,len(lista)):
                                        combinaciones[9].append(lista[a] +
lista[b]+ lista[c]+ lista[d]+
lista[e]+lista[f]+lista[g]+lista[h]+lista[i]+lista[j])

combinacionesvol=[]
for i in range (len(combinaciones)): #Filtra volúmenes menores o iguales a
4200.
    for j in range (len(combinaciones[i])):
        sum=0
        for k in range (len(combinaciones[i][j])):
```

```
            sum= sum + combinaciones[i][j][k]
        if (sum<=4200):
            combinacionesvol.append(combinaciones[i][j])
mx=0
sumVol=0
listaindice=[]
for i in range (len(combinacionesvol)): #Busca y suma el beneficio de cada combinación, y guarda la de mayor beneficio.
    sum=0
    sumV=0
    for j in range (len(combinacionesvol[i])):
        ind=listavol.index(combinacionesvol[i][j])
        sum= sum+ listapre[ind][1]
        sumV= sumV+ listapre[ind][0]
    if (sum>mx):
        indmax=i
        mx=sum
        sumVol= sumV
for i in range (len(combinacionesvol[indmax])):
    ind=listavol.index(combinacionesvol[indmax][i])
    listaindice.append(ind+1)

print(combinacionesvol[indmax]) #Combinación de mayor beneficio.
print(listaindice) #Índices de los elementos.
print(mx) #Valor del beneficio.
print(sumVol) #Volúmen total.
```

RESULTADOS

```
[150, 325, 600, 430, 1200, 770, 60, 353]
[1, 2, 3, 5, 6, 7, 8, 10]
299
3888
```

PUNTO 2.

CODIGO.

```
listapre=[[150,20],[325,40],[600,50],[805,36],[430,25],[1200,64],[770,54],[60,18],[930,46],[353,28]]
listaGreedy=[]
listaOrdenada=[] #Crea listas.

for i in range(10): #Calcula y carga lista greedy.
   listaGreedy.append(float(listapre[i][1]/listapre[i][0]))
listaOrdenada=sorted(listaGreedy,reverse=True) #Ordena y guarda lista greedy.

mochila=[] #Crea mochila.
sumVol=0
sumPrecio=0
for i in range(10): #Busca objeto que coincida con el orden de la lista.
   for j in range(10):
      if (listaOrdenada[i]==float(listapre[j][1]/listapre[j][0])):
         ind=j
   sumVol= sumVol + listapre[ind][0] #Suma y calcula que el volumen sea válido.
   if sumVol<=4200:
      mochila.append(listapre[ind]) #De ser válido suma el precio.
      sumPrecio= sumPrecio + listapre[ind][1]
   else:
      sumVol=sumVol - listapre[ind][0] #De lo contrario descarta el volumen sumado.

volumenes=[]
greedy=[]
indices=[]
for i in range(len(mochila)):
   volumenes.append(mochila[i][0])#Carga volúmenes de cada objeto.
for i in range(len(mochila)):
   greedy.append(float(mochila[i][1]/mochila[i][0]))#Carga cociente Greedy de cada objeto.
```

```
for i in range(len(mochila)):
    for j in range(10): #Carga Índices de cada objeto.
        if (mochila[i]==listapre[j]):
            indices.append(j+1)
print(volumenes) #Combinación de mayor beneficio.
print(greedy)  #Cociente Greedy de cada uno.
print(indices)  #Índicies de los elementos.
print(sumPrecio) #Beneficio total.
print(sumVol) #Volumen total.
```

RESULTADOS

[60, 150, 325, 600, 353, 770, 430, 1200]

[0.3, 0.13333333333333333, 0.12307692307692308, 0.08333333333333333, 0.07932011331444759, 0.07012987012987013, 0.05813953488372093, 0.05333333333333334]

[8, 1, 2, 3, 10, 7, 5, 6]

299

3888

COMPARACION GREEDY/EXHAUSTIVO.

Una similitud que notamos en ambos métodos para este problema en específco es que ambos llegan al resultado más óptimo posible, solo que podemos notar una gran diferencia en la forma de implementación de uno y de otro.

En el caso del método de búsqueda exhaustivo, éste debe primero calcular todas las posibles combinaciones, luego filtrar sólo las que son válidas y así hacer un análisis de cuál es la más óptima con respecto al beneficio que otorga.

Mientras tanto, en el método de búsqueda greedy se puede notar una gran diferencia en cuanto a la eficiencia del algoritmo. Este método consiste en calcular la razón Beneficio/Volumen de cada objeto ($\frac{Precio\ (\$)}{Volumen\ (\mathrm{cm3})}$), para posteriormente ordenarlos de mayor a menor e ir incluyéndolos en dicho orden en la mochila, siempre y cuando no exceda el volumen permitido por la misma. En el caso que un objeto exceda dicho volumen, pero uno posterior sí pueda ser ingresado, se lo ingresa y se continúa el análisis desde este mismo.

Ya que ambos llegan al mismo resultado, es conveniente utilizar el método greedy debido a que este tiene un menor tiempo de procesamiento.

PUNTO 3-A.

CODIGO BUSQUEDA EXHAUSTIVA.

```
lista=[[1800],[600],[1200]]
listapeso=[1800,600,1200]
listapre=[[1800,72],[600,36],[1200,60]]
combinaciones=[]
for i in range(len(lista)): #Crea Lista de combinaciones
    combinaciones.append([])

for a in range(len(lista)): #Crea todas las combinaciones posibles.
    combinaciones[0].append(lista[a])
    for b in range(a+1,len(lista)):
        combinaciones[1].append(lista[a] + lista[b])
        for c in range(b+1,len(lista)):
            combinaciones[2].append(lista[a] + lista[b] + lista[c])

combinacionesvol=[]
for i in range (len(combinaciones)): #Filtra pesos menores o iguales a 3000
    for j in range (len(combinaciones[i])):
        sum=0
        for k in range (len(combinaciones[i][j])):
            sum= sum + combinaciones[i][j][k]
        if (sum<=3000):
            combinacionesvol.append(combinaciones[i][j])

mx=0
sumPeso=0
listaindice=[]
for i in range (len(combinacionesvol)): #Busca y suma el beneficio de cada combinación, y
guarda la de mayor beneficio
    sum=0
    sumP=0
    for j in range (len(combinacionesvol[i])):
        ind=listapeso.index(combinacionesvol[i][j])
        sum= sum+ listapre[ind][1]
        sumP= sumP + listapre[ind][0]
```

```
    if (sum>mx):
        indmax=i
        mx=sum
        sumPeso= sumP
for i in range (len(combinacionesvol[indmax])):
    ind=listapeso.index(combinacionesvol[indmax][i])
    listaindice.append(ind+1)

print(combinacionesvol[indmax]) #Combinación de mayor beneficio
print(listaindice) #Índices de los objetos
print(mx) #Valor del beneficio
print(sumPeso) #Peso total de la mochila
```

RESULTADOS

```
[1800, 1200]
[1, 3]
132
3000
```

CODIGO BUSQUEDA GREEDY

```
listapre=[[1800,72],[600,36],[1200,60]]
listaGreedy=[]
listaOrdenada=[] #Crea listas.

for i in range(3): #Calcula lista greedy
    listaGreedy.append(float(listapre[i][1]/listapre[i][0]))
listaOrdenada=sorted(listaGreedy,reverse=True) #Ordena lista de mayor a menor

mochila=[] #Crea mochila
sumPeso=0
sumPrecio=0
for i in range(3): #Evalúa los objetos a completar en la mochila.
    for j in range(3): #Busca el índice del objeto que sigue en la lista.
        if (listaOrdenada[i]==float(listapre[j][1]/listapre[j][0])):
            ind=j
```

```
    sumPeso= sumPeso + listapre[ind][0]
    if sumPeso<=3000: #Analiza que no supere el peso y agrega el precio
        mochila.append(listapre[ind])
        sumPrecio= sumPrecio + listapre[ind][1]
    else: #Si supera el peso descarta el objeto
        sumPeso=sumPeso - listapre[ind][0]


pesos=[]
greedy=[]
indices=[]
for i in range(len(mochila)):
    pesos.append(mochila[i][0])
for i in range(len(mochila)):
    greedy.append(float(mochila[i][1]/mochila[i][0])) #Muestra cociente Greedy
for i in range(len(mochila)): #Muestra número de objeto
    for j in range(3):
        if (mochila[i]==listapre[j]):
            indices.append(j+1)



print(pesos) #Combinacion de mayor beneficio.
print(greedy) #Cociente greedy de cada uno.
print(indices) #Índice de cada objeto.
print(sumPrecio) #Valor del beneficio.
print(sumPeso) #Peso total de la mochila
```

RESULTADOS

```
[600, 1200]
[0.06, 0.05]
[2, 3]
96
1800
```

PUNTO 3-B

Para este ejercicio notamos que la búsqueda exhaustiva resulta ser la más óptima, ya que, esta devuelve el resultado que mas aprovecha la capacidad de la mochila y al ser solo 3 objetos, no lleva mucho tiempo en procesar todas las combinaciones.

En cambio, en el método greedy podemos notar que éste al priorizar el objeto con mayor razón (Beneficio/Peso) no aprovecha la totalidad de la mochila, por lo tanto, no es conveniente utilizarlo.

CONCLUSIÓN GENERAL

Podemos notar que por ambos métodos de búsqueda, el método greedy es más rápido que el exhaustivo, por lo cual se vuelve ventajoso usarlo cuando la cantidad de elementos es muy grande.

Debido a que éste método no tiene en cuenta todas las posibilidades, sino que da prioridad en la mochila a aquellos objetos que poseen la mayor razón beneficio/volumen, si alguna combinación que no posea, por ejemplo, al objeto de mayor razón es la más óptima, éste no será capaz de encontrarla.

Por lo tanto, en algunas ocasiones, nos arriesgamos a que no se encuentre el mejor de los resultados.

Por otra parte, el método exhaustivo considera la mejor de todas las opciones posibles por lo tanto el resultado es siempre el mas óptimo. Sin embargo, es eficiente solo ante una cantidad pequeña de objetos, ya que de lo contrario toma demasiado tiempo su procesamiento.

Una vez realizadas éstas conclusiones, definimos que, dependiendo de la cantidad de objetos a procesar daremos prioridad a uno u otro método para definir el resultado.