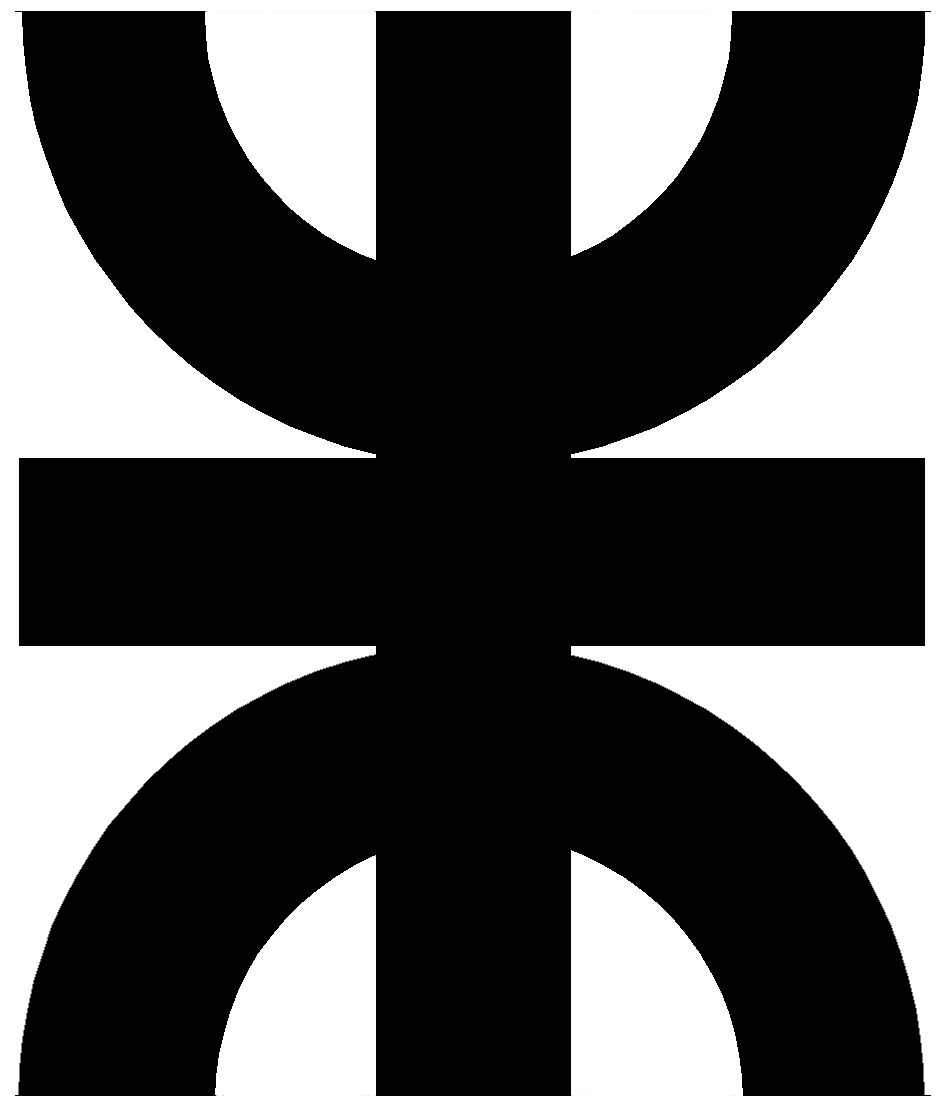
TRABAJO PRÁCTICO Nº 2

Cátedra: Algoritmos Genéticos



**Integrantes Legajo**

Albizuri, Gastón 40412

Belletti, Kristal 40568

Giordano, Nicolás 40467

**UNIVERSIDAD TECNOLÓGICA NACIONAL**

**FACULTAD REGIONAL ROSARIO**

**INGENIERÍA EN SISTEMAS DE INFORMACIÓN**

Índice

**Pág.**

**Introducción** **1**

**Problemática** **2**

**Resolución** **3**

**Código Fuente** **5**

**Resultados**……………………………………………………………………………………………………**11**

**Conclusiones** …**12**

Introducción

*Los* ***algoritmos de optimización combinatoria*** *resuelven instancias de problemas que se creen ser difíciles en general, explorando el espacio de soluciones (usualmente grande) para estas instancias. Los algoritmos de optimización combinatoria logran esto reduciendo el tamaño efectivo del espacio, y explorando el espacio de búsqueda eficientemente.*

*El* ***Problema de la Mochila****, comúnmente abreviado por* ***KP*** *(del inglés Knapsack problem) es un problema de* ***optimización combinatoria****: buscar la mejor solución entre un conjunto de posibles soluciones a un problema. Modela una situación análoga al llenar una mochila, incapaz de soportar más de un peso determinado, con todo o parte de un conjunto de objetos, cada uno con un peso y valor específicos. Los objetos colocados en la mochila deben maximizar el valor total sin exceder el peso máximo.*

*Sin embargo, también existen los* ***algoritmos voraces*** *(también conocido como ávidos, devoradores o golosos) que son aquellos, en los que para resolver un determinado problema, siguen una* ***heurística*** *consistente en* ***elegir la opción óptima en cada paso local******con la esperanza de llegar a una solución general óptima****. Este esquema algorítmico es el que* ***menos dificultades*** *plantea a la hora de diseñar y comprobar su funcionamiento. Normalmente se aplica a los problemas de optimización.*

Problemática

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.

Utilizando una computadora, 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.***

|  |  |  |
| --- | --- | --- |
| **Volumen máximo soportado por la mochila: 4200 cm3** | | |
| **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** |

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, 600 y 1200 grs., y cuyos valores son $72, $36 y $60 respectivamente, y con una mochila que puede soportar hasta 3000 grs. Se pide:
   1. Hallar una solución utilizando un algoritmo goloso y uno exhaustivo

Analizar dicha solución respecto a su grao de optimización y elaborar las conclusiones que considere adecuadas.

Resolución

Puntos a tener en cuenta para resolver el problema:

* Se tienen **n objetos** (en una primera instancia 10 objetos, y luego, sólo 3) y **una mochila**.
* El **objeto i** tiene un **valor monetario (precio) pi** y **un volumen** **vi** (en el siguiente problema, un valor monetario y un peso xi).
* El objetivo en definitiva es **llenar la mochila, de capacidad C** (en el primer problema la mochila tiene una capacidad de 4200 cm3, luego de 3000 grs.)**, de manera que se maximice el beneficio.**

**Maximizar**

**sujeto a**

**con**

**Búsqueda exhaustiva**

La búsqueda exhaustiva es una técnica trivial pero a menudo usada, que consiste en enumerar sistemáticamente todos los posibles candidatos para la solución de un problema, con el fin de chequear si dicho candidato satisface la solución al mismo.

**PASOS:**

1. **Codificación binaria:** La codificación utilizada para cada posibilidad (de longitud N, donde cada posición representa un objeto de mis N objetos disponibles) de mi universo de posibilidades sigue el formato de una cadena binaria (genes 0 y 1): donde el “0” (cero) significa que ese objeto NO se guarda en la mochila, mientras que el “1” (uno), SI se guarda el objeto.

El número de elementos N está asociado a la cantidad de elementos que posee el espacio solución. Si el conjunto es de N elementos, el conjunto solución tendrá la cantidad de elementos del conjunto de Partes, es decir, 2N.

Por lo tanto, si tengo 10 elementos en el primer problema, mi universo va a constar de 210 posibilidades (1024 posibilidades: desde “0000000000” –no llevo ningún objeto hasta “1111111111” –llevo todos los objetos-). Mientras que en el segundo problema, solo tengo 3 elementos, por lo que tengo 8 posibilidades (mi universo de posibilidades tiene un tamaño de 23).

1. **Decodificación y Fitness de cada individuo:** La decodificación de binario a decimal, sirve para obtener por cada posibilidad (cada cromosoma) la ganancia total (sumatoria de valores monetarios) y el volumen/peso total de la posibilidad (sumatoria de los volúmenes/pesos).

Para ello, hemos creados dos (tres) funciones recorren el conjunto de posibilidades, y para cada una, suman los valores monetarios y volúmenes o pesos a partir de la recorrida de los genes que la componen (desde la ubicación 1 a la 10): si encuentra un “0” en una ubicación i-ésima significa que el objeto ubicado en dicha posición de la lista original (tabla dada del problema) no se roba; en cambio, si aparece un “1” en una ubicación i-ésima, ese objeto se roba, por ende se le pide a dicho objeto: su valor monetario y su peso o su volumen, y se suma a un contador de valor (valor total), peso (peso total) y/o volumen (volumen total) de cada posibilidad.

1. Al realizar una búsqueda exhaustiva, debemos recorrer todo el conjunto de posibilidades (2N, donde N representa los objetos disponibles). Para ello lo primero que debemos hacer es **clasificar los subconjuntos por su valor total de mayor a menor.**
2. **Recorrer todas las posibilidades** de mayor a menor para encontrar cuál es el subconjunto solución, siguiendo **la condición de que el volumen total (o peso total) no supere la capacidad máxima de la mochila**.

Código fuente

**Método Exhaustivo**

#bloque def constantes

combin = pow(2,10)

vol = 4200

#bloque funciones

#programa principal

elementos = [] #arreglo elementos

elementos.append([150,20])

elementos.append([325,40])

elementos.append([600,50])

elementos.append([805,36])

elementos.append([430,25])

elementos.append([1200,64])

elementos.append([770,54])

elementos.append([60,18])

elementos.append([930,46])

elementos.append([353,28])

maxi = 0

for i in xrange(combin):

nroBin = bin(i)[2:].zfill(10)

valorMochila = 0

pesoMochila = 0

for j in xrange(10):

valorMochila = valorMochila + elementos[j][1]\*int(nroBin[j], base=2)

pesoMochila = pesoMochila + elementos[j][0]\*int(nroBin[j], base=2)

if(pesoMochila <= vol):

if(valorMochila > maxi):

maxi = valorMochila

maxPeso = pesoMochila

nroMax = nroBin

print('Elementos de la mochila')

for i in xrange(10):

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

print(i+1,elementos[i])

print('Valor maximo mochila: ',maxi)

print('Peso mochila: ',maxPeso)

**Método Greedy**

#bloque def constantes

combin = pow(2,10)

vol = 4200

#bloque funciones

def proporcion(val,peso):

return val/float(peso)

#programa principal

elementos = [] #arreglo elementos

propor = []

orden = []

elementos.append([150,20])

elementos.append([325,40])

elementos.append([600,50])

elementos.append([805,36])

elementos.append([430,25])

elementos.append([1200,64])

elementos.append([770,54])

elementos.append([60,18])

elementos.append([930,46])

elementos.append([353,28])

for i in xrange(10):

p = proporcion(elementos[i][1],elementos[i][0])

propor.append([p,i])

propor.sort(reverse=True)

acum = 0

for i in xrange(10):

acum = acum + elementos[propor[i][1]][0] #peso del objeto i

if(acum <= vol):

orden.append(propor[i][1])

else:

acum = acum - elementos[propor[i][1]][0]

sumValor = 0

sumPeso = 0

print('Elementos de la mochila')

for i in xrange(len(orden)):

print(orden[i]+1,elementos[orden[i]])

sumValor = sumValor+ elementos[orden[i]][1]

sumPeso = sumPeso + elementos[orden[i]][0]

print('Valor de la mochila: ',sumValor)

print('Peso de la mochila: ',sumPeso)

**Método Exhaustivo ( 3 elementos)**

#bloque def constantes

combin = pow(2,3)

vol = 3000

#bloque funciones

#programa principal

elementos = [] #arreglo elementos

elementos.append([1800,72])

elementos.append([600,36])

elementos.append([1200,60])

maxi = 0

for i in xrange(combin):

nroBin = bin(i)[2:].zfill(3)

valorMochila = 0

pesoMochila = 0

for j in xrange(3):

valorMochila = valorMochila + elementos[j][1]\*int(nroBin[j], base=2)

pesoMochila = pesoMochila + elementos[j][0]\*int(nroBin[j], base=2)

if(pesoMochila <= vol):

if(valorMochila > maxi):

maxi = valorMochila

maxPeso = pesoMochila

nroMax = nroBin

print('Elementos de la mochila')

for i in xrange(3):

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

print(i+1,elementos[i])

print('Valor maximo mochila: ',maxi)

print('Peso mochila: ',maxPeso)

**Método Greedy ( 3 elementos)**

#bloque def constantes

combin = pow(2,10)

vol = 3000

#bloque funciones

def proporcion(val,peso):

return val/float(peso)

#programa principal

elementos = [] #arreglo elementos

propor = []

orden = []

elementos.append([1800,72])

elementos.append([600,36])

elementos.append([1200,60])

for i in xrange(3):

p = proporcion(elementos[i][1],elementos[i][0])

propor.append([p,i])

propor.sort(reverse=True)

acum = 0

for i in xrange(3):

acum = acum + elementos[propor[i][1]][0] #peso del objeto i

if(acum <= vol):

orden.append(propor[i][1])

else:

acum = acum - elementos[propor[i][1]][0]

sumValor = 0

sumPeso = 0

print('Elementos de la mochila')

for i in xrange(len(orden)):

print(orden[i]+1,elementos[orden[i]])

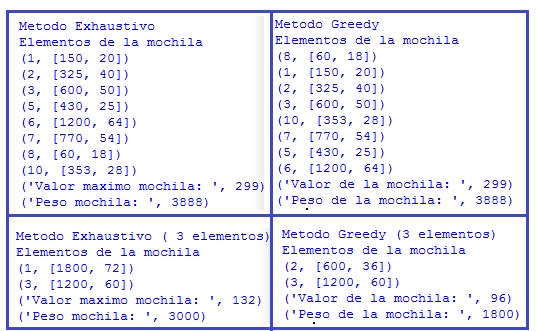
sumValor = sumValor+ elementos[orden[i]][1]

sumPeso = sumPeso + elementos[orden[i]][0]

print('Valor de la mochila: ',sumValor)

print('Peso de la mochila: ',sumPeso)

Resultados



Conclusiones

El propósito de este Trabajo Práctico es aplicar dos formas de resolución llegar a una solución óptimo o lo más cercano a esta (una solución factible).

Por un lado, la solución típica: la **búsqueda exhaustiva aplicando optimización combinatoria**: calcular todas las posibles combinaciones de objetos, con el valor total de la solución, y quedarnos con el mejor. Esta es la **solución correcta**, y **garantiza que el valor máximo ha sido encontrado**.

Por el otro lado, la **búsqueda utilizando métodos heurísticos**: una técnica que es utilizada cuando es necesario **resolver un problema inmediatamente**, es una manera de resolución bastante buena que **no nos da** **(en la mayoría de los casos)** **la solución óptima sino una buena solución** y que está basada en el sentido común. Es decir que proporcionan **soluciones factibles** (que satisfacen las restricciones del problema), que, aunque no optimicen la función objetivo, **se acercan al valor óptimo** en un tiempo de cálculo razonable.

Esto lo podemos verificar en el resultado del **segundo problema** (tenemos 3 elementos):

* Por medio de la **búsqueda exhaustiva** *encontramos que la mejor solución (la óptima) es robar los objetos 1 y 3, generando una ganancia de $132 y completando en la totalidad la capacidad de la mochila.*
* Sin embargo, por otro lado, utilizando el **algoritmo greedy (goloso)**, la *solución recomendada es llevarse los objetos 2 y 3, generando una ganancia de $96 y ocupando solo 1800grs de la capacidad de la mochila.*

Podemos observar que la solución a partir de métodos heurísticos se acercan a la solución óptima que hemos encontramos resolviendo con el método exhaustivo. Es decir, **obtuvimos una solución factible pero no la óptima**.

Sin embargo también podemos verificar que **el algoritmo greedy, también puede darnos la solución óptima** (la correcta): en el **problema 1** donde tenemos 10 elementos:

* A través de la **búsqueda exhaustiva**, encontramos que *la mejor solución es la de llevar 8 elementos, donde obtenemos una ganancia de $299 y un volumen total en la mochila de 3888 cm3.*
* Consecuentemente, por la **búsqueda heurística**, también *llegamos a esa solución: 8 elementos, una ganancia de $299 y un volumen total de 3888 cm3.*

Con la **búsqueda exhaustiva** encontramos siempre la solución óptima, la mejor solución, la mejor posibilidad, sin embargo el **problema** aparece cuando **tenemos una gran lista de objetos**. Calcular todas las posibilidades es demasiado costoso en tiempo y recursos. Existen problemas en los que es imposible obtener una solución con este tipo de cálculo extensivo en un tiempo lógico. Es por ello, que esta técnica se usa habitualmente cuando *el número de soluciones candidatas no es elevado (conjunto de soluciones)*.

Es por ello que aparecen los métodos heurísticos, como el algoritmo devorador (greedy): que también se utilizan para resolver problemas de optimización (obtener el máximo o el mínimo), suelen ser rápidos y fáciles de implementar, pero no siempre garantizan alcanzar la solución óptima: *proporcionan una solución aceptable a un problema mediante métodos que carecen de justificación formal*.