25/06/2019

Trabajo Práctico 2: Búsquedas Exhaustivas y Heurística

ALGORITMOS GENÉTICOS 2019

ANTONELLI (44852) – RECALDE (44704) – ROHN (41355)

Índice

Introducción

El Problema de la Mochila

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.

Ejercicio 1

Resolver el Problema de la Mochila utilizando una Búsqueda Exhaustiva.

Ejercicio 2

Resolver el ejercicio anterior usando el algoritmo Greedy (Heurística) y comentar su similitud o no con el Exhaustivo.

Ejercicio 3

Plantear el Problema de la Mochila teniendo en cuenta los pesos en lugar del volumen, y luego:

A)- Resolverlo con Exhaustivo.

B)- Resolverlo con Greedy (Heurística).

Para lograr la implementación del algoritmo genético se tuvieron en cuenta los siguientes datos:

- Peso Máximo de la Mochila
- Cantidad N de elementos
- Tiempo de arranque y finalización del algoritmo
- Posibilidad de Generación de cada elemento manualmente, o de forma aleatoria

Código

Ejercicio 1

- __init__

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

"""
ENUNCIADO DEL TRABAJO PRÁCTICO N° 2

El Problema de la Mochila
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.

Ejercicio 1
Resolver el Problema de la Mochila utilizando una Búsqueda Exhaustiva

Ejercicio 2
Resolver el ejercicio anterior usando el algoritmo Greedy (Heurística) y comentar
su similitud o no con el Exhaustivo.

Ejercicio 3
Plantear el Problema de la Mochila teniendo en cuenta los pesos en lugar del
volumen, y luego:
    A)- Resolverlo con Exhaustivo
    B)- Resolverlo con Greedy (Heurística)


FECHA DE ENTREGA DEL TRABAJO PRÁCTICO: 30 de Abril de 2019

--> Genetic-Algorithm TP2 --- V1.0 ---  Created on 20 jun. 2019

            Antonelli, Nicolás - Recalde, Alejando - Rohn, Alex
"""

import random
from time import time
from itertools import combinations
# Main Function is located at the end


# Elements may be created randomly or with inputs
def generateElements(N, isRandom, maxVol, maxPrice):
    elements = [[0] * N for _ in range(3)]  # Every Element has 3 values: Position
(0), Price Value (0) and Volume (1)
    if isRandom:
        for i in range(N):
            num = i+1
            p = round(random.uniform(0, maxPrice), 3)
            v = round(random.uniform(0, maxVol), 3)
            elements[0][i] = num
            elements[1][i] = p
            elements[2][i] = v
```

```
    else:
        for i in range(N):
            num = i+1
            print("Ingrese Precio de elemento", i+1, ":", end="")
            p = round(float(input()), 3)
            print("Ingrese Volumen de elemento", i+1, ":", end="")
            v = round(float(input()), 3)
            print()
            elements[0][i] = num
            elements[1][i] = p
            elements[2][i] = v
    print("Elementos:")
    for j in range(N):
        print("Num "+str(elements[0][j])+": $"+str(elements[1][j])+",
"+str(elements[2][j])+" gramos")
    print()
    return elements


# Calculate all the Possible Solutions sub-sets
def calculatePossibleSolutions(elements):
    N = len(elements)
    print("Índice de Combinaciones:")
    combinationsIndex = sum([list(map(list, combinations(elements[0], i))) for i in
range(len(elements[0]) + 1)], [])
    print(combinationsIndex)
    solutions = []  # Array of 2^N sub-sets, one for every possible solution
    for x in range(len(combinationsIndex)):
        solutions.append([])
        partialP = 0
        partialV = 0
        for y in range(len(combinationsIndex[x])):
            z = combinationsIndex[x][y]
            partialP += elements[1][z-1]
            partialV += elements[2][z-1]
        solutions[x].append(round(partialP, 3))
        solutions[x].append(round(partialV, 3))
        solutions[x].append(combinationsIndex[x])
    print()
    print("Conjunto de Posibles Soluciones:")
    print("Solución --- Precio --- Volumen --- Elementos")
    for j in range(len(solutions)):
        print("S"+str(j)+": $"+str(solutions[j][0])+", "+str(solutions[j][1])+"
gramos, elementos:", solutions[j][2])
    print()
    return solutions


# Exhaustive Search Algorithm and Exhibition of the Best Solution
def findBestSolution(solutions, maxVol, initialTime):
    best = 0
    bestValue = 0
    for i in range(len(solutions)):
        if solutions[i][1] <= maxVol and solutions[i][0] > bestValue:
            bestValue = solutions[i][0]
            best = i
    print("Total de Elementos: ", len(elements[0]))
    print("Peso Máximo de la Mochila: ", maxVol, "gramos")
    print()
    print("La Solución Óptima es: S" + str(best))
```

```
    if best != 0:
        print("Valor Acumulado de Precios: $" + str(solutions[best][0]))
        print("Peso Total: " + str(solutions[best][1]) + " gramos")
        print("Elementos:", solutions[best][2])
        for j in range(len(solutions[best][2])):
            k = solutions[best][2][j]-1
            print("    Num "+str(elements[0][k])+": $"+str(elements[1][k])+",
"+str(elements[2][k])+" gramos")
    else:
        print("Ningún elemento entra en la mochila")
    finalTime = time()
    return finalTime-initialTime


# Main Function
if __name__ == '__main__':
    # Important Values
    totalElements = 10  # Amount of Elements for the Problem
    randomCreation = True  # False: You must specify all elements value - True:
They will create randomly
    bagMaxVolume = 4200  # Maximum Bag Capacity (in grams)
    maxPriceValue = 6000  # Maximum Money Value for an Element (only in Random
Creation)
    initialTime = time()  # Initial Time Mark

    # Generate all the N Elements
    elements = generateElements(totalElements, randomCreation, bagMaxVolume,
maxPriceValue)

    # Calculate all the Possible Solutions
    solutions = calculatePossibleSolutions(elements)

    # Exhaustive Search on the Solutions Set
    totalTime = findBestSolution(solutions, bagMaxVolume, initialTime)

    print()
    print("Solución obtenida correctamente en", totalTime, "segundos")
```

Ejercicio 2…

Ejercicio 3-A…

Ejercicio 3-B…

Lenguaje Utilizado: PYTHON

Repositorio GitHub:
https://github.com/utn-frro-geneticos19-g16/Genetic-Algorithms-TP2

Nuestra Organización en GitHub con nuestros TPs:
https://github.com/utn-frro-geneticos19-g16/

Explicación de Funcionamiento

Ejercicio 1

Función generateElements:

- Su objetivo es crear cada uno de los N Elementos.
- Cada Elemento tiene un “Número de Elemento”, un “Volumen en Gramos”, y un “Valor de Precio”.
- El parámetro “isRandom” pasado como True, hace que los valores de Volumen y precio de cada elemento sean generados aleatoriamente, mientras que pasado como False te deja agregar cada uno manualmente.
- Genera una Lista Bidimensional de tamaño 3xN, donde almacenamos las 3 propiedades antes mencionadas de cada uno de los N Elementos.

Función calculatePossibleSolutions

- Su objetivo es definir el conjunto de posibles soluciones. Este tendrá 2^N subconjuntos, donde cada uno es una posible solución.
- Primero se toman los Números de Elementos, y se hace una Lista con todas las combinaciones posibles entre Elementos (Lista de Índices de Combinaciones).
- Luego, generamos una Lista vacía donde se calcularán e irán agregando los valores de cada posible solución. Para esto se recorre cada combinación, y por cada Número de Elemento que aparece, se suman sus volúmenes y valores de precio (volumen y precio parcial) para cada subconjunto.
- Finalmente se completa la anterior Lista, ahora de tamaño 2^N, donde en cada posición encontramos una posible solución ya calculada con las siguientes 3 características: “Suma parcial de los volúmenes”, “Suma parcial de los precios”, “Lista de los Números de Elementos que la componen”.

Función findBestSolution

- Su objetivo es hacer la búsqueda exhaustiva en el conjunto de posibles soluciones, para encontrar la mejor solución. Es decir, la que mayor valor de precio acumulado (Suma Parcial de los Precios), y que la Suma Parcial de los volúmenes no supere el máximo de la Mochila, por su puesto.
- Se toma como default la solución S0 (conjunto vacío con precio y volumen 0), y luego se recorren todas las posibles soluciones y una a una se consulta si es una mejor solución que la última guardada como la mejor, hasta llegar al final del conjunto.
- Finalmente, se obtiene el número y características (valores) de la mejor solución, y son exhibidos en pantalla junto a una lista de las características de los elementos que componen esta solución.

Función __init__

- Función principal (o Main).
- Acá se especifican los parámetros principales de nuestro algoritmo (valor de N, tamaño de la mochila, etc).
- Se llama a las funciones antes mencionadas.
- Como adicional, en el final se exhibe cuanto tiempo tardó el algoritmo en ejecutarse completamente (desde el principio donde se generan los elementos, hasta el final donde se encuentra la mejor solución).

Ejercicio 2,3-A,3-B…

Salida en Pantalla

Ejercicio 1

Elementos:

Num 1: $2740.561, 3121.362 gramos
Num 2: $2792.671, 446.226 gramos
Num 3: $1188.645, 525.662 gramos
Num 4: $2152.667, 2122.888 gramos
Num 5: $1522.986, 3674.656 gramos
Num 6: $2338.963, 1444.389 gramos
Num 7: $1946.642, 3591.7 gramos
Num 8: $1734.195, 2228.656 gramos
Num 9: $4890.869, 3016.242 gramos
Num 10: $1298.265, 2376.221 gramos

Índice de Combinaciones:

[[], [1], [2], [3], [4], [5], [6], [7], [8], [9], [10], [1, 2], [1, 3], [1, 4], [1, 5], [1, 6], [1, 7], [1, 8], [1, 9], [1, 10], [2, 3], [2, 4], [2, 5], [2, 6], [2, 7], [2, 8], [2, 9], [2, 10], [3, 4], [3, 5], [3, 6], [3, 7], [3, 8], [3, 9], [3, 10], [4, 5], [4, 6], [4, 7], [4, 8], [4, 9], [4, 10], [5, 6], [5, 7], [5, 8], [5, 9], [5, 10], [6, 7], [6, 8], [6, 9], [6, 10], [7, 8], [7, 9], [7, 10], [8, 9], [8, 10], [9, 10], [1, 2, 3], [1, 2, 4], [1, 2, 5], [1, 2, 6], [1, 2, 7], [1, 2, 8], [1, 2, 9], [1, 2, 10], [1, 3, 4], …, [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]]

Conjunto de Posibles Soluciones:

Solución --- Precio --- Volumen --- Elementos
S0: $0, 0 gramos, elementos: []
S1: $2740.561, 3121.362 gramos, elementos: [1]
S2: $2792.671, 446.226 gramos, elementos: [2]
S3: $1188.645, 525.662 gramos, elementos: [3]
S4: $2152.667, 2122.888 gramos, elementos: [4]
S5: $1522.986, 3674.656 gramos, elementos: [5]
S6: $2338.963, 1444.389 gramos, elementos: [6]
S7: $1946.642, 3591.7 gramos, elementos: [7]
S8: $1734.195, 2228.656 gramos, elementos: [8]
S9: $4890.869, 3016.242 gramos, elementos: [9]
S10: $1298.265, 2376.221 gramos, elementos: [10]
S11: $5533.232, 3567.588 gramos, elementos: [1, 2]
S12: $3929.206, 3647.024 gramos, elementos: [1, 3]
S13: $4893.228, 5244.25 gramos, elementos: [1, 4]
S14: $4263.547, 6796.018 gramos, elementos: [1, 5]
S15: $5079.524, 4565.751 gramos, elementos: [1, 6]
S16: $4687.203, 6713.062 gramos, elementos: [1, 7]
S17: $4474.756, 5350.018 gramos, elementos: [1, 8]
S18: $7631.43, 6137.604 gramos, elementos: [1, 9]

S19: $4038.826, 5497.583 gramos, elementos: [1, 10]
S20: $3981.316, 971.888 gramos, elementos: [2, 3]
S21: $4945.338, 2569.114 gramos, elementos: [2, 4]
S22: $4315.657, 4120.882 gramos, elementos: [2, 5]
S23: $5131.634, 1890.615 gramos, elementos: [2, 6]
S24: $4739.313, 4037.926 gramos, elementos: [2, 7]
S25: $4526.866, 2674.882 gramos, elementos: [2, 8]
S26: $7683.54, 3462.468 gramos, elementos: [2, 9]
S27: $4090.936, 2822.447 gramos, elementos: [2, 10]
S28: $3341.312, 2648.55 gramos, elementos: [3, 4]
S29: $2711.631, 4200.318 gramos, elementos: [3, 5]
S30: $3527.608, 1970.051 gramos, elementos: [3, 6]
…

S1023: $22606.464, 22548.002 gramos, elementos: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

Total de Elementos: 10

Peso Máximo de la Mochila: 4200 gramos

La Solución Óptima es: S97
Valor Acumulado de Precios: $8872.185
Peso Total: 3988.13 gramos
Elementos: [2, 3, 9]
  Num 2: $2792.671, 446.226 gramos
  Num 3: $1188.645, 525.662 gramos
  Num 9: $4890.869, 3016.242 gramos

Solución obtenida correctamente en 0.03812718391418457 segundos

Process finished with exit code 0

Ejercicio 2...

Ejercicio 3-A...

Ejercicio 3-B...

Conclusión

Utilizando el método de búsqueda exhaustivo, se obtiene la solución óptima sin lugar a dudas. Pero tiene un gran coste de tiempo de procesamiento si es que el valor de N es muy grande. Esto se debe a que el conjunto de posibles soluciones es de 2^N, y por cada elemento a agregar se está duplicando la cantidad de subconjuntos a calcular sus valores. Si N tiene un valor por ejemplo de 10, el resultado se obtiene en 0.38 segundos; pero si N es de 20, ya tarda 23 segundos, N de 25 ya tarda 15 minutos, y así con un N muy grande puede tardar horas, días… y mucho más.

Si el N es mucho más grande que lo expresado hasta acá, es muy conveniente reducir el espacio de búsqueda. Para esto, el método heurístico constructivo “Greedy” nos fue de muchísima utilidad, ya que nos redujo el conjunto de 2^N, a tan solo N subconjuntos. La desventaja que nos ofrece, es que la solución que selecciona no es siempre la óptima. Sin embargo, es de esperar que nos de una buena solución (de las mejores posibles).

El coste de eficiencia (encontrar una buena solución en vez de la solución óptima), es perfectamente coherente y necesaria con un N mayor a 27. Sin embargo con un N que va hasta 27, la búsqueda exhaustiva se lleva nuestra elección ya que hasta 60 minutos aprox. de procesamiento (pensando en un solo equipo de un solo procesador) nos parece un tiempo coherente para este tipo de problemática.