ALGORITMOS GENÉTICOS
TRABAJO PRÁCTICO 2

"Optimización"

Cutró, Gastón	Cabanellas, Ignacio	Benedetti, Juan	Fernández, Natalia	Gofiar, Cristian
44757	44987	45403	44758	44839

2020

Índice general

Capítulo 1

Introduccion

1.1. Introducción

La optimización combinatoria es una rama de la optimización en matemáticas aplicadas y en ciencias de la computación, relacionada con la investigación de operaciones, teoría de algoritmos y teoría de la complejidad computacional. También está relacionada con otros campos, como la inteligencia artificial e ingeniería de software. 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.

Los algoritmos de optimización combinatoria a menudo son implementados en lenguajes imperativos como C y C++ entre otros softwares inteligentes en lenguajes de programación lógicos tales como Prolog, o incluso en lenguajes multi-paradigma tales como Oz.

Mediante el estudio de la teoría de la complejidad computacional es posible comprender la importancia de la optimización combinatoria. Los algoritmos de optimización combinatoria se relacionan comúnmente con problemas NP-hard. Dichos problemas en general no son resueltos eficientemente, sin embargo, varias aproximaciones de la teoría de la complejidad sugieren que ciertas instancias (ej. "pequeñasïnstancias) de estos problemas pueden ser resueltas eficientemente. Dichas instancias a menudo tienen ramificaciones prácticas muy importantes.

1.1.1. Problematica

El problema preente es llamado .El problema de la mochila". Consiste en elegir dentro de un conjunto de elementos que tienen un valor y un volumen tales que puedan ser cargados en una mochila de volumen V de manera que el valor obtenido sea máximo.

Capítulo 2

Definicion del Sistema

2.1. Definición del Sistema

Desarrollamos un software con fines didácticos que encuentra la combinación óptima de objetos que cabe en la mochila consiguiendo maximizar el valor.
A saber, la cantidad de combinaciones posibles viene dada por 2^n, donde n es la cantidad de objetos totales.

Se realiza la ejecucion con tipos de busqueda predefinidos:

- Método de búsqueda exahustiva.
- Método de búsqueda con algoritmo de Greedy.

Capítulo 3

Métodos de búsqueda y resultados obtenidos

3.1. Búsqueda Exhaustiva

En informática, la búsqueda por fuerza bruta, búsqueda combinatoria, búsqueda exhaustiva o simplemente fuerza bruta, 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.

Por ejemplo, un algoritmo de fuerza bruta para encontrar el divisor de un número natural n consistiría en enumerar todos los enteros desde 1 hasta n, chequeando si cada uno de ellos divide n sin generar resto. Otro ejemplo de búsqueda por fuerza bruta, en este caso para solucionar el problema de las ocho reinas (posicionar ocho reinas en el tablero de ajedrez de forma que ninguna de ellas ataque al resto), consistiría en examinar todas las combinaciones de posición para las 8 reinas (en total 64!/56! = 178,462,987,637,760 posiciones diferentes), comprobando en cada una de ellas si las reinas se atacan mutuamente.

La búsqueda por fuerza bruta es sencilla de implementar y, siempre que exista, encuentra una solución. Sin embargo, su coste de ejecución es proporcional al número de soluciones candidatas, el cual es exponencialmente proporcional al tamaño del problema. Por el contrario, la búsqueda por fuerza bruta se usa habitualmente cuando el número de soluciones candidatas no es elevado, o bien cuando éste puede reducirse previamente usando algún otro método heurístico.

Es un método utilizado también cuando es más importante una implementación sencilla que una mayor rapidez. Este puede ser el caso en aplicaciones críticas donde cualquier error en el algoritmo puede acarrear serias consecuencias; también es útil como método "baseçuando se desea comparar el desempeño de otros algoritmos metaheurísticos. La búsqueda de fuerza bruta puede ser vista como el método metaheurístico más simple.

La búsqueda por fuerza bruta no se debe confundir con backtracking, método que descarta un gran número de conjuntos de soluciones, sin enumerar explícitamente cada una de las mismas.

3.2. Algoritmo de Greedy

En ciencias de la computación, un algoritmo voraz (también conocido como goloso, ávido, devorador o greedy) es una estrategia de búsqueda por la cual se sigue 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.

Dado un conjunto finito de entradas C, un algoritmo voraz devuelve un conjunto S (seleccionados) tal que $S \subseteq C$ y que además cumple con las restricciones del problema inicial. A cada conjunto S que satisfaga las restricciones se le suele denominar prometedor, y si este además logra que la función objetivo se minimice o maximice (según corresponda) diremos que S es una solución óptima.

Elementos de los que consta la técnica:

- El conjunto C de candidatos, entradas del problema.
- Función solución. Comprueba, en cada paso, si el subconjunto actual de candidatos elegidos forma una solución (no importa si es óptima o no lo es).
- Función de selección. Informa cuál es el elemento más prometedor para completar la solución. Éste no puede haber sido escogido con anterioridad. Cada elemento es considerado una sola vez. Luego, puede ser rechazado o aceptado y pertenecerá a $C \setminus S$.
- Función de factibilidad. Informa si a partir de un conjunto se puede llegar a una solución. Lo aplicaremos al conjunto de seleccionados unido con el elemento más prometedor.
- Función objetivo. Es aquella que queremos maximizar o minimizar, el núcleo del problema.

Funcionamiento:
El algoritmo escoge en cada paso al mejor elemento $x \in C$ posible, conocido como el elemento más prometedor. Se elimina ese elemento del conjunto de candidatos ($C \leftarrow C \setminus \{x\}$) y, acto seguido, comprueba si la inclusión de este elemento en el conjunto de elementos seleccionados ($S \cup \{x\}$) produce una solución factible.

En caso de que así sea, se incluye ese elemento en S. Si la inclusión no fuera factible, se descarta el elemento. Iteramos el bucle, comprobando si el conjunto de seleccionados es una solución y, si no es así, pasando al siguiente elemento del conjunto de candidatos.

3.3. Experimento con parámetro limitante "Volumen"

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.

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

Volumen máximo soportado por la mochila: 4200 cm^3.

3.3.1. Resultados

Método exhaustivo

Utilizamos un arbol binario para resolver el problema de forma exhaustiva. El resultado con este metodo es el siguiente:

```
[3888, 299, '1110111101']
```

La primer cifra, 3888 (cm^3) corresponde al volumen correspondiente al valor máximo alcanzado que es la segunda cifra, 299 ($). El último valor de la lista corresponde a una cadena de dígitos binarios que indican con "1" el objeto correspondiende a la posición de ese dígito. En este caso caben todos los objetos excepto el objeto 4 y el 9.

Algoritmo de Greedy

El algoritmo de Greedy realiza un análisis priorizando una característica que se le atribuye a cada objeto. Esta característica viene dada por la relación $\frac{Valor}{Volumen}$ que nosotros denominamos "Fitness" haciendo referencia al trabajo práctico anterior:

```
[[7, 0.3], [0, 0.133], [1, 0.123], [2, 0.0833], [9, 0.079],
[6, 0.070], [4, 0.058], [5, 0.053], [8, 0.049], [3, 0.044]]
```

Es una lista de listas. Las listas pequeñas están conformadas por el número de objeto como primer valor, y su segundo valor es el fitnes del mismo. Están ordenado por mayor a menor fitness.
El algoritmo nos indica que tomemos todos los objetos de mayor fitness hasta que no quepan mas en la mochila. Con estas instrucciones obtenemos:

```
[3888, 299, [7, 0, 1, 2, 9, 6, 4, 5]]
```

Notamos que son los mismos valores que el anterior método. Una coincidencia lógica ya que ambos buscan la mejor combinación.
Cabe destacar que donde antes encontrabamos el arreglo binario, por comodidad de los desarrolladores, ahora encontramos directamente la identificación de cada objeto.

3.4. Experimento con parámetro limitante “Peso”

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.

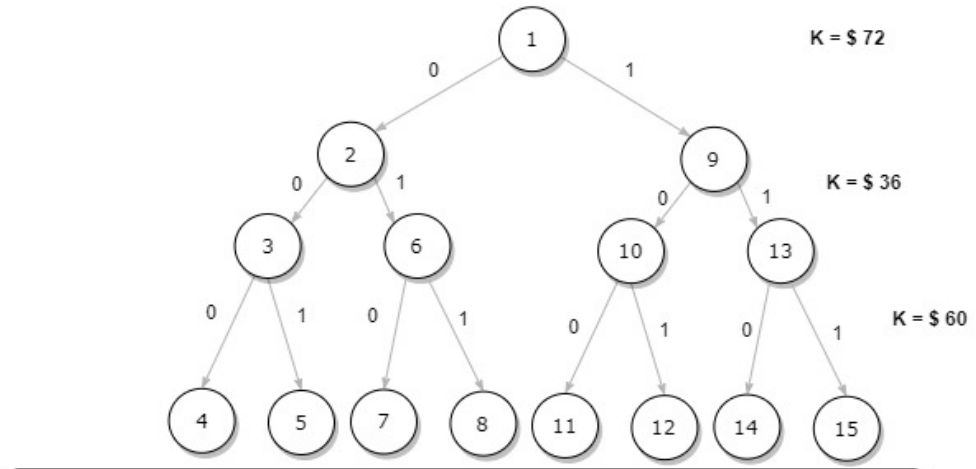
3.4.1. Resultados

Método exhaustivo

El resultado con método exhaustivo es el siguiente:

```
[3000, 132, '101']
```

Los datos corresponden a la mejor combinación posible dentro de las 8 existentes. Utilizamos un arbol binario para resolver el problema planteado. A continuación dejamos una figura para comprender de manera gráfica la solución utilizada, donde se muestran en los nodos todas las posibles soluciones y debajo sus respectivos valores en precio y volumen.



PRECIO TOTAL	0	$ 60	$ 36	$ 96	$ 72	**$ 132**	$ 108	$168
VOLUMEN TOTAL	0	1200	600	1800	1800	**3000**	2400	3600

Algoritmo de Greedy

Al igual que en el ejercicio anterior, estas listas corresponden a los fitness de cada objeto y están ordenados de mayor a menor.

```
[[1, 0.06], [2, 0.05], [0, 0.04]]
```

El resultado con el algoritmo de Greedy es el siguiente:

```
[1800, 96, [1, 2]]
```

El resultado obtenido es proveniente de tomar los objetos con mas fitness, pero en este caso la combinación de objetos es distina a la conseguida con el método exhaustivo. Por raro que parezca, esta combinación es peor que la conseguida con el otro método. Se podría decir que la mochila llena posee mas fitness, pero menos valor.

Esto sucede por que si selecciono el objeto 1 y 2, tienen mejor fitness pero falta peso para llenar la mochila al tope (por ende se desaprovecha la máxima resistencia de la mochila). Si quisiera agregar el objeto 0, no sería posible ya que la mochila se rompería. En cambio, si pongo el objeto 0 y el 2, entran justo, 3000g. Ese peso desaprovechado ahora se está usando y tiene más valor económico la suma de estos 2 objetos, pero menos fitness.

Resumiendo, los objetos 1 y 2 tienen mas fitness pero se desperdicia resistencia. Y los objetos 0 y 2 tienen menos fitness pero aprovechan la resistencia total. El objeto 0 es mas pesado y mas valioso que el 1 (en peor proporción), pero como entra justo, me conviene llevarlo ya que maximiza la ganancia.

Capítulo 4

Software Desarrollado

4.1. Código

```
import numpy as np
from operator import itemgetter

# objetos = {objeto : [volumen(0), valor(1)]}
#objetos = {0:[150, 20], 1:[325, 40], 2:[600, 50], 3:[805, 36], 4:[430, 25],\
            5:[1200, 64], 6:[770, 54], 7:[60, 18], 8:[930, 46], 9:[353, 28]}
objetos = {0:[1800, 72], 1:[600, 36], 2:[1200, 60]}

#limite = 4200       # El límite puede ser 4200cm^3 o 3000g
limite = 3000

# mochila_optima [volumen,precio, [combinacion de cosas]]
mochila_optima = [0,0,0]

def mochila_maxima(valor_tot):
    return valor_tot > mochila_optima[1]

def verificar_volumen(volumen):
    return volumen <= limite

def genera_combinaciones():
    global mochila_optima
    for i in range(2**(len(objetos))):
        volumen_tot = 0
        valor_tot = 0
        x = format(i, "0" + str(len(objetos)) + "b")
        for j in range(len(x)):
            if x[j] == "1":
                # Calculo el volumen total y el precio total
```

```
                volumen_tot +=  objetos[j][0]
                valor_tot   +=  objetos[j][1]


        if verificar_volumen(volumen_tot) and mochila_maxima(valor_tot):
            mochila_optima = [volumen_tot, valor_tot, x]

genera_combinaciones()
print(mochila_optima)


def greedy():
    fitness = []
    mochila_objetos = []
    volumen_aux = 0
    valor_aux = 0
    for i in objetos:
        fitness.append([i, objetos[i][1] / objetos[i][0]])

    fitness = sorted(fitness, key=itemgetter(1), reverse=True)
    print(fitness)

    for i in fitness:
        if (volumen_aux + objetos[i[0]][0]) <= limite:
            volumen_aux += objetos[i[0]][0]
            valor_aux   += objetos[i[0]][1]
            mochila_objetos.append(i[0])

    return [volumen_aux, valor_aux, mochila_objetos]

print(greedy())
```

Capítulo 5

Conclusión

5.1. Conclusión

Comprendimos con estas actividades que la forma de encontrar un óptimo varía dependiendo el contexto y la población que tengamos. Si la población es muy reducida y el óptimo a encontrar debe ser muy preciso, el algoritmo de Greedy no es buena opción ya que pueden surgir falencias como la analizada. Si la cantidad de resultados es inmensa, como en el problema de las 8 reinas, el método exhaustivo se vuelve completamente obsoleto ya que con el poder de cómputo actual se tardarían años en procesar varios trillones de opciones distintas.
El método exhaustivo es bueno para una reducida cantidad de resultados posibles (y si se requiere un resultado preciso, como encontrar una contraseña) y es sencillo de pensar y programar: simplemente prueba todas las combinaciones posibles.
El algoritmo de Greedy es bueno para gran cantidad de resultados posibles pero siendo conscientes del margen de error existente. Este es un poco más elaborado que el anterior a la hora de desarrollar.

Bibliografía

[1] LaTeX2 Project Team *LaTeX2ε for authors*,2019.
https://www.latex-project.org/help/documentation/usrguide.pdf

[2] Ruben Ruiz *Cómo usar Overleaf editor de LaTeXonline -Tutorial-*.
https://www.youtube.com/watch?v=Uyjo6R5z2js

[3] *Wikipedia*.
https://es.wikipedia.org