INSTITUTO POLITÉCNICO NACIONAL

Centro de Investigación en Computación

ASIGNATURA:

Metaheurísticas

Actividad #4:

Guía Taller Laboratorio 1

PROFESORA:

Dra. Yenny Villuendas Rey

PRESENTA:

Juan René Hernández Sánchez

Adriana Montserrat García Carrillo

Centro de Investigación en Computación
Instituto Politécnico Nacional

1. Introducción

Los procedimientos metaheurísticos son una clase de métodos aproximados que están diseñados para resolver problemas complejos de optimización. Las metaheurísticas proporcionan un marco general para crear nuevos algoritmos híbridos combinando diferentes conceptos derivados de la inteligencia artificial, la evolución biológica y los mecanismos estadísticos.

Con el método de ascensión de colinas la estrategia es repetidamente expandir un nodo, inspeccionar sus sucesores recién generados, y seleccionar y expandir el mejor entre los sucesores sin mantener referencias a los padres. Cuando se llega a un nodo muerto no hay forma de hacer retroceso (salvo generar otro nodo raíz).

En el siguiente trabajo se profundizará sobre los temas de métodos heurísticos, algoritmo de ascensión de colinas, operadores, ventajas y desventajas y aplicaciones.

2. Desarrollo

Asignatura: Metaheurísticas
Actividad N_o. 4
Guía Taller N_o. 1
Título: Solución de problemas mediante Ascensión de Colinas.
Contenido:

- Métodos heurísticos de solución de problemas.
- Ascensión de Colinas.
- Ascensión de Colinas con mutación aleatoria.

Objetivo: Implementar algoritmos de Ascensión de Colinas, en lenguajes de alto nivel, para la solución de problemas de la profesión.

2.1. Enuncie las ventajas y desventajas de la Ascensión de Colinas

Ventajas:

- Útil para resolver problemas de optimización y encontrar el mejor estado.
- El algoritmo no mantiene un árbol de búsqueda, por lo que la estructura de datos del nodo sólo necesita registrar el estado y su evaluación.(Norving, 1995)

Desventajas:
Puede presentar los siguientes inconvenientes:

- *Máximos locales*: Una vez en un máximo local, el algoritmo se detendrá (aunque la solución no sea la óptima). (Norving, 1995)
- *Meseta:* una meseta es una zona del espacio de estados donde la función de evaluación es esencialmente plana. La búsqueda realizará un recorrido aleatorio.
- *Crestas:* una cresta puede tener lados muy inclinados, de modo que la búsqueda alcanza la cima de la cresta con facilidad, pero la cima puede tener una pendiente muy suave hacia un pico. A menos que haya operadores que se muevan directamente a lo largo de la cima de la cresta, la búsqueda puede oscilar de un lado a otro, avanzando poco. (Norving, 1995)

2.2. Detalle el pseudocódigo del algoritmo Steepest-ascent hill-climbing (SAHC)

Algorithm 1 Heurística NSteepest-ascent hill-climbing (SAHC)

Input: Un Estado Aleatorio del problema
Output: Un óptimo local

```
1: currentHilltop ← Random String
2: currentFitness ← ComputeFitness(currentHilltop)
3: eval ← S − 1                          ▷ Siendo S ∈ ℕ Un número determinado de Evaluaciones
4: repeat
5:     hilltopFounds ← NULL              ▷ Arreglo donde se guardan los hills
6:     fitnessFounds ← NULL              ▷ Arreglo donde se guardan los fitness
7:     for i ← 0 to currentHilltop.length do
8:         mutatedHilltop ← Mutate(i, currentHiltop)
9:         mutatedFitness ← ComputeFitness(mutatedHilltop)
10:        hilltopFounds ← mutatedHilltop
11:        fitnessFounds ← mutatedFitness
12:        eval := eval − 1
13:        if Max(fitnessFounds > currentFitness) then
14:            currentHilltop := hilltopFounds[Max(fitnessFounds).index]
15:        else
16:            currentHilltop ← Random String
17:        end if
18:    end for
19: until eval ≠ 0
20: return currentHilltop
```

(S., 1993)

2.3. Detalle el pseudocódigo del algoritmo Next-ascent hill-climbing (NAHC)

Algorithm 2 Heurística Next-Ascent Hill-climbing (NAHC)

Input: Un Estado Aleatorio del problema
Output: Un óptimo local

1: $currentHilltop \leftarrow$ random string
2: $currentFitness \leftarrow ComputeFitness(currentFitness)$
3: $eval \leftarrow S - 1$ ▷ Siendo $S \in \mathbb{N}$ Un número determinado de Evaluaciones
4: $climb \leftarrow False$
5: **repeat**
6: **for** $i \leftarrow 0$ to $currentHilltop.lenght$ **do**
7: $mutatedHilltop \leftarrow Mutate(i, currentHiltop)$
8: $mutatedFitness \leftarrow ComputeFitness(mutatedHilltop)$
9: $eval := eval - 1$
10: **if** $mutatedFitness > currentFitness$ **then**
11: $currentHilltop := mutatedHilltop$
12: $currentFitness := mutatedFitness$
13: $climb := True$
14: **end if**
15: **end for**
16: **if not** $climb$ **then**
17: $currentHilltop \leftarrow$ random string
18: $currentFitness \leftarrow ComputeFitness(currentFitness)$
19: $eval := eval - 1$
20: **end if**
21: **until** $eval \neq 0$
22: **return** $currentHilltop$

(S., 1993)

2.4. Detalle el pseudocódigo del algoritmo Random mutation hill-climbing (RMHC)

Algorithm 3 Heurística Random mutation hill-climbing (RMHC)

Input: Un Estado Aleatorio del problema
Output: Un óptimo local

```
1: iterations ← S                ▷ Siendo S ∈ ℕ Un número determinado de Evaluaciones
2: bestEvaluated ← random string
3: bestFitness ← ComputeFitness(bestEvaluated)
4: length ← bestEvaluated.lenght()
5: repeat
6:     locus ← Rand(0, length)
7:     mutatedHilltop ← Mutate(bestEvaluated, locus)
8:     mutatedFitness ← ComputeFitness(mutatedHilltop)
9:     if mutatedFitness ≥ bestFitness then
10:        bestEvaluated := mutatedHilltop
11:        bestFitness := mutatedFitness
12:    end if
13:    iterations := iterations − 1
14: until iterations ≠ 0
15: return bestEvaluated
```

(S., 1993)

3. Mencione aplicaciones de los algoritmos de Ascensión de Colinas

Estos algoritmos son utilizados en la programación "job shop" (que es un tipo de proceso de fabricación), en la programación automática, en el diseño de circuitos, en la gestión de rutas vehiculares, en la robótica para gestionar los equipos de múltiples robots, etc. (Web, 2019)

4. Realice la modelación matemática necesaria para la solución, mediante RMHC, de los problemas siguientes:

4.1. Problema de la mochila (Knapsack problem)

- Estado inicial: n productos con valor C_i y peso P_i que se deben poner en una mochila. Se tiene como límite un peso P_m a cargar por la mochila. $n \in \mathbb{N}$. Los valores $C_i, P_i \in \mathbb{R}_*^+$. El estado de la mochila se representa mediante una cadena binaria x de longitud n. El estado inicial es una cadena x con n 0's.
- Estado final: Una cadena $x\prime$ tal que $x\prime = \{x_i | Max(f(x_i), f(x_i\prime)), 0 \leq i \leq n-1\}$ y donde $f(x) = \sum_{i=1}^{n} C_i x_i$ de manera que $f(x) \leq f(x\prime)$.
- Test objetivo:
  $f(x) = \sum_{i=1}^{n} C_i x_i$ **s.a.** $\sum_{i=1}^{n} P_i x_i \leq P_m, x_i \in \{0, 1\}$
- Acciones posibles (operadores):

$f : x \mapsto x\prime$ **s.a.** $\exists x_i \neq x_i\prime, i \in \{0, \ldots, n\}$ donde $x_i = 1$ si el producto se empaqueta o está dentro de la mochila, de lo contrario $x_i = 0$ si el producto no está empaquetado o se sacó de la mochila. Resultando de ahí la condición binaria de la variable x_i.

4.2. Dado el Problema de la Mochila, proponga las estructuras de datos necesarias para su implementación.

- cadenas
- Listas
- Tuplas

4.3. Implementación del Código

4.3.1. Heurística NSteepest-ascent hill-climbing (SAHC)

```
import random


class Objetos:
    def __init__(self, valor, peso):
        self.valor = valor
        self.peso = peso


n = input('Ingresa el n mero de objetos que desea meter en la mochila: ')
n = int(n)

lista_objetos = []
for i in range(n):
    valor = input(f"Ingrese el valor del objeto {i}:  ")
    valor = int(valor)
    peso = input(f"Ingrese el peso del objeto {i}:  ")
    peso = int(peso)
    lista_objetos.append(Objetos(valor, peso))

W = input('Ingresa el peso maximo que puede soportar la mochila: ')
W = int(W)


def fitness(mochila):
    Waux = W
    valmax = 0
    for j, objeto in enumerate(mochila):
        # Evaluando los objetos que estan dentro de la mochila
        objeto_actual = lista_objetos[j]
        if objeto == '1':
            Waux = Waux - objeto_actual.peso
            valmax = valmax + objeto_actual.valor

    if Waux < 0:
        valmax = 0

    return int(valmax)

```

```
40
41 def mutar_bit(bit, i, mochila):
42     nbit = 0
43     if bit == '0':
44         nbit = 1
45     new_mochila = mochila[0:i] + str(nbit) + mochila[i+1:]
46     return new_mochila
47
48
49 def generar_random(num):
50     nmochila = ""
51     for i in range(num):
52         nmochila = nmochila + str(random.randint(0, 1))
53     return nmochila
54
55
56 N = input('Ingresa el n mero de evaluaciones: ')
57 N = int(N)
58 # La var mochila es una cadena binaria
59 mochila = generar_random(n)
60
61 fitness_actual = fitness(mochila)
62 eval = 1
63
64 print("Mochila Inicial: " + mochila +
65       " con fitness igual a: " + str(fitness_actual) + "\n")
66
67 while eval < N:
68     nuevas_fitness = []
69
70     # heapq.heapify(nuevas_fitness)
71     cadenas = []
72     # i=el indice de la iteraci n bit=el valor en el indice de la cadena
73     for i, bit in enumerate(mochila):
74         mochila = mutar_bit(bit, i, mochila)
75         cadenas.append(mochila)
76         nuevas_fitness.append(fitness(mochila))
77         eval += 1
78     maxaux = max(nuevas_fitness)
79     if maxaux > fitness_actual:
80         indice = nuevas_fitness.index(maxaux)
81         mochila = cadenas[indice]
82         fitness_actual = maxaux
83     else:
84         mochila = generar_random(n)
85         fitness_actual = fitness(mochila)
86         eval += 1
87
88 print("Mochila Final: " + mochila +
89       " con fitness igual a: " + str(fitness_actual) + "\n")
```

```
PS D:\ISC\Semestre 2022-1\01 Metaheurísticas\Tareas\Tarea Practica 02\codigos> & C:/Users/renep/AppData/Local/Programs/Python/Python310/python.exe "d:/ISC/Semestre 2022-1/01 Metaheur
isticas/Tareas/Tarea Practica 02/codigos/SAHC.py"
Ingresa el número de objetos que desea meter en la mochila: 6
Ingrese el valor del objeto 0:  10
Ingrese el peso del objeto 0:  10
Ingrese el valor del objeto 1:  10
Ingrese el peso del objeto 1:  7
Ingrese el valor del objeto 2:  10
Ingrese el peso del objeto 2:  2
Ingrese el valor del objeto 3:  10
Ingrese el peso del objeto 3:  1
Ingrese el valor del objeto 4:  10
Ingrese el peso del objeto 4:  6
Ingrese el valor del objeto 5:  10
Ingrese el peso del objeto 5:  4
Ingresa el peso maximo que puede soportar la mochila: 30
Ingresa el número de evaluaciones: 100
Mochila Inicial: 100011 con fitness igual a: 30

Mochila Final: 111111 con fitness igual a: 60
```

4.3.2. Heurística Next-Ascent Hill-climbing (NAHC)

```python
import random


class Objetos:
    def __init__(self, valor, peso):
        self.valor = valor
        self.peso = peso


n = input('Ingresa el n mero de objetos que desea meter en la mochila: ')
n = int(n)

lista_objetos = []
for i in range(n):
    valor = input(f"Ingrese el valor del objeto {i}:  ")
    valor = int(valor)
    peso = input(f"Ingrese el peso del objeto {i}:  ")
    peso = int(peso)
    lista_objetos.append(Objetos(valor, peso))

W = input('Ingresa el peso maximo que puede soportar la mochila: ')
W = int(W)


def fitness(mochila):
    Waux = W
    valmax = 0
    for j, objeto in enumerate(mochila):
        # Evaluando los objetos que estan dentro de la mochila
        objeto_actual = lista_objetos[j]
        if objeto == '1':
            Waux = Waux - objeto_actual.peso
            valmax = valmax + objeto_actual.valor

    if Waux < 0:
        valmax = 0

    return int(valmax)


def mutar_bit(bit, i, mochila):
    nbit = 0
    if bit == '0':
        nbit = 1
    new_mochila = mochila[0:i] + str(nbit) + mochila[i+1:]
```

```
    return new_mochila


def generar_random(num):
    nmochila = ""
    for i in range(num):
        nmochila = nmochila + str(random.randint(0, 1))
    return nmochila


N = input('Ingresa el n mero de evaluaciones: ')
N = int(N)
# La var mochila es una cadena binaria
mochila = generar_random(n)

fitness_actual = fitness(mochila)
eval = 1
mejora = False

print("Mochila Inicial: " + mochila +
      " con fitness igual a: " + str(fitness_actual) + "\n")

while eval < N:

    for i, bit in enumerate(mochila):
        nueva_mochila = mutar_bit(bit, i, mochila)
        nueva_fitness = fitness(nueva_mochila)
        eval += 1

        if nueva_fitness > fitness_actual:
            mochila = nueva_mochila
            fitness_actual = nueva_fitness
            mejora = True

    if not mejora:
        mochila = generar_random(n)
        fitness_actual = fitness(mochila)
        eval += 1

print("Mochila Final: " + mochila +
      " con fitness igual a: " + str(fitness_actual) + "\n")
```

```
PS D:\ISC\Semestre 2022-1\01 Metaheurísticas\Tareas\Tarea Practica 02\codigos> & C:/Users/renep/AppData/Local/Programs/Python/Python310/python.exe "d:/ISC/Semestre 2022-1/01 Metaheur
isticas/Tareas/Tarea Practica 02/codigos/NAHC.py"
Ingresa el número de objetos que desea meter en la mochila: 6
Ingrese el valor del objeto 0:  10
Ingrese el peso del objeto 0:  10
Ingrese el valor del objeto 1:  10
Ingrese el peso del objeto 1:  7
Ingrese el valor del objeto 2:  10
Ingrese el peso del objeto 2:  2
Ingrese el valor del objeto 3:  10
Ingrese el peso del objeto 3:  1
Ingrese el valor del objeto 4:  10
Ingrese el peso del objeto 4:  6
Ingrese el valor del objeto 5:  10
Ingrese el peso del objeto 5:  4
Ingresa el peso maximo que puede soportar la mochila: 30
Ingresa el número de evaluaciones: 100
Mochila Inicial: 101100 con fitness igual a: 30

Mochila Final: 111111 con fitness igual a: 60
```

4.3.3. Heurística Random mutation hill-climbing (RMHC)

```
import random


```

```
class Objetos:
    def __init__(self, valor, peso):
        self.valor = valor
        self.peso = peso


n = input('Ingresa el n mero de objetos que desea meter en la mochila: ')
n = int(n)

lista_objetos = []
for i in range(n):
    valor = input(f"Ingrese el valor del objeto {i}:  ")
    valor = int(valor)
    peso = input(f"Ingrese el peso del objeto {i}:  ")
    peso = int(peso)
    lista_objetos.append(Objetos(valor, peso))

W = input('Ingresa el peso m ximo que puede soportar la mochila: ')
W = int(W)


def fitness(mochila):
    Waux = W
    valmax = 0
    for j, objeto in enumerate(mochila):
        # Evaluando los objetos que estan dentro de la mochila
        objeto_actual = lista_objetos[j]
        if objeto == '1':
            Waux = Waux - objeto_actual.peso
            valmax = valmax + objeto_actual.valor

    if Waux < 0:
        valmax = 0

    return int(valmax)


def mutar_bit(locus, mochila):
    bit = 0
    if mochila[locus] == '0':
        bit = 1
    new_mochila = mochila[0:locus] + str(bit) + mochila[locus+1:]
    return new_mochila


def generar_random(num):
    nmochila = ""
    for i in range(num):
        nmochila = nmochila + str(random.randint(0, 1))
    return nmochila


N = input('Ingresa el n mero de evaluaciones: ')
N = int(N)
# La var mochila hace referencia a la var best evaluated
mochila = generar_random(n)

fitness_actual = fitness(mochila)
```

```
# La var n se usar  como length
eval = 1
print("Mochila Inicial: " + mochila +
      " con fitness igual a: " + str(fitness_actual) + "\n")

while eval < N:
    locus = random.randint(0, n-1)
    nueva_mochila = mutar_bit(locus, mochila)
    nueva_fitness = fitness(nueva_mochila)
    eval += 1

    if nueva_fitness >= fitness_actual:
        mochila = nueva_mochila
        fitness_actual = nueva_fitness


print("Mochila Final: " + mochila +
      " con fitness igual a: " + str(fitness_actual) + "\n")
```

```
PS D:\ISC\Semestre 2022-1\01 Metaheurísticas\Tareas\Tarea Practica 02\codigos> & C:/Users/renep/AppData/Local/Programs/Python/Python310/python.exe "d:/ISC/Semestre 2022-1/01 Metaheur
isticas/Tareas/Tarea Practica 02/codigos/RMHC.py"
Ingresa el número de objetos que desea meter en la mochila: 6
Ingrese el valor del objeto 0:  10
Ingrese el peso del objeto 0:  10
Ingrese el valor del objeto 1:  10
Ingrese el peso del objeto 1:  7
Ingrese el valor del objeto 2:  10
Ingrese el peso del objeto 2:  2
Ingrese el valor del objeto 3:  10
Ingrese el peso del objeto 3:  1
Ingrese el valor del objeto 4:  10
Ingrese el peso del objeto 4:  6
Ingrese el valor del objeto 5:  10
Ingrese el peso del objeto 5:  4
Ingresa el peso máximo que puede soportar la mochila: 30
Ingresa el número de evaluaciones: 100
Mochila Inicial: 100001 con fitness igual a: 20

Mochila Final: 111111 con fitness igual a: 60
```

5. Conclusiones

En este trabajo se pudo observar que los algoritmos metaheurísticos constituyen ideas generales que permiten un margen de maniobra muy amplio a la hora de ser aplicados. Esta gran versatilidad es la que los hace muy atractivos, ya que es posible adaptarlos a casi cualquier problema de optimización. De igual manera, se logró aprender las ventajas y desventajas de los algoritmos de ascensión de colinas. Descubriendo que son una poderosa herramienta para la resolución de problemas en diversas áreas.

Referencias

Norving, R. . (1995). *Artificial intelligence – a modern approach.* PRENTICE HALL. (Third ed.)

S., M. M. H. J. H. . F. (1993). Relative building-block fitness and the building block hypothesis. *D. Whitley, Foundations of Genetic Algorithms*, *2*(5), 109–126.

Web, P. (2019). *Inteligencia artificial avanzada - algoritmo de ascension en colina.* (https://advanceintelligence.wordpress.com/2014/11/07/algoritmo-de-ascension-en-colina/text=El20algoritmo20de20ascenso20por,y20actuar20en20ambientes20estocC3A1sticos.)