Algoritmos de optimización - Seminario

Nombre y Apellidos: **SALVADOR GIMENO DESCO**
Url: https://github.com/salvagimeno-ai/03MAIR-Algoritmos-de-optimizacion/tree/master/SEMINA
Problema:

> 3. Combinar cifras y operaciones

Descripción del problema:(copiar enunciado)

Disponemos de las 9 cifras del 1 al 9 (excluimos el cero) y de los 4 signos básicos de las operacic multiplicación(*) y división(/). Debemos combinarlos alternativamente sin repetir ninguno de ellos

(*) La respuesta es obligatoria

```
# EN PRIMER LUGAR IMPORTAMOS LAS LIBRERIAS QUE IREMOS USANDO DURANTE EL EJERCICIO:
import numpy as np
from operator import itemgetter
import matplotlib.pyplot as plt
import random
import pprint
import sys
import pandas as pd
sys.setrecursionlimit(100000)
import math
```

Pregunta 1:

> 1.1: (*)¿Cuantas posibilidades hay sin tener en cuenta las restricciones?

> 1.2: ¿Cuantas posibilidades hay teniendo en cuenta todas las restricciones.

Respuesta 1.1:

> Si no tuvieramos en cuenta la restriccion de que no se pueden repetir los numeros ni los sign serían: (9^5)*(4^4) = 15116544

```
P = (9**5)*(4**4)
P
```

```
15116544
```

Respuesta 1.2:

> Teniendo en cuenta las restricciones que no se pueden repetir ni los numeros ni las operacior $9 * 8 * 7 * 6 * 5 * 4! = 9! = 362880$ soluciones posibles.

```
P = (9*8*7*6*5)* 4*3*2*1
P
```

```
362880
```

Pregunta 2: Modelo para el espacio de soluciones

(*) ¿Cual es la estructura de datos que mejor se adapta al problema? Argumentalo.(Es posible que necesidad de cambiar, arguentalo)

Respuesta 2: Un array de todos los números posibles, ordenados de menor a mayor NUMEROS=[1 posibles operadores. OPERADORES=["+","*","-","/"]

> NUMEROS=[1,2,3,4,5,6,7,8,9]
>
> OPERADORES=["+","*","-","/"]

Pregunta 3: Según el modelo para el espacio de soluciones

3.1 (*) ¿Cual es la función objetivo?

3.2 (*)¿Es un problema de maximización o minimización?

Respuesta 3.1: Minimizar 'd' o la diferencia entre la solucion_objetivo y la solucion_propuesta.

> d = solucion_objetivo - solucion_propuesta
>
> solucion_propuesta = eval(expresion)
>
> expresion=str(n[0])+str(o[0])+str(n[1])+str(o[1])+str(n[2])+str(o[2])+str(n[3])+str(o[3]+str(n[4]))

siendo:

- n: vector de numeros: n = [4,2,6,3,1]
- o: vector de operaciones: o = ['+','-','/','*']

```
# Ejemplo: solucion_objetivo: 4 = 4+2-6/3*1
# Resolucion:
#solucion_objetivo = 4
#n = [4,2,6,3,1]    # vector de números
#o = ['+','-','/','*']     # vector de operaciones
#expresion=str(n[0])+str(o[0])+str(n[1])+str(o[1])+str(n[2])+str(o[2])+str(n[3])+str(o[3]+
#expresion   # = '4+2-6/3*1'
#solucion_propuesta = eval(expresion)
#solucion_propuesta
#d = solucion_objetivo - solucion_propuesta
#print('la diferencia entre la solucion_objetivo - solucion_propuesta es: ', d)

# ejemplo:
n = [4,2,6,3,1]    # vector de números
o = ['+','-','/','*']     # vector de operaciones
```

```
expresion=str(n[0])+str(o[0])+str(n[1])+str(o[1])+str(n[2])+str(o[2])+str(n[3])+str(o[3]+s
expresion   # = '4+2-6/3*1'

# calculamos la diferencia con el valor objetivo
def dif_objetivo(valor_objetivo, expresion):
  solucion_propuesta = eval(expresion)
  diferencia = valor_objetivo - solucion_propuesta
  return diferencia

#dif_objetivo(4, expresion)
print('la diferencia entre la solucion_objetivo - solucion_propuesta es: ', dif_objetivo(4
```

```
la diferencia entre la solucion_objetivo - solucion_propuesta es:  0.0
```

Respuesta 3.2:

Sería un problema de **minimización** de la diferencia entre la solucion_objetivo y la solucion_propu

Pregunta 4:

Diseña un algoritmo para resolver el problema por fuerza bruta

Respuesta

```
from time import time

#Función para calcular el tiempo de ejecución
def calcular_tiempo(f):
    def wrapper(*args, **kwargs):
        inicio = time()
        resultado = f(*args, **kwargs)
        tiempo = time() - inicio
        print("Tiempo de ejecución para algoritmo: "+str(tiempo))
        return resultado
    return wrapper


# Algoritmo para resolución del problema por fuerza bruta
@calcular_tiempo
def combinar_cifras_operaciones(valor):

  operadores = ['*','+','-','/']
  valores = [1,2,3,4,5,6,7,8,9]


  for d1 in range(1,10):
    for op1 in operadores:
      for d2 in valores:
        if d2 != d1:
          for op2 in operadores:
            if op1!=op2:
              for d3 in valores:
                if d3!=d1 and d3!=d2:
                  for op3 in operadores:
```

```
                    if op3!=op1 and op3!=op2:
                      for d4 in valores:
                        if d4!=d3 and d4!=d2 and d4!=d1:
                          for op4 in operadores:
                            if op4!=op3 and op4!=op2 and op4!=op1:
                              for d5 in valores:
                                if d5!=d4 and d5!=d3 and d5!=d2 and d5!=d1:
                                  formula = str(d1)+op1+str(d2)+op2+str(d3)+op3+str(d4)+op
                                  # Evaluamos la expresión
                                  if (eval(formula)==valor):
                                    print("Resultado x Fuerza Bruta:")
                                    print(formula,"=",str(valor))
                                    return


  # En caso que no hayamos encontrado ninguna solución:
  print("No hemos encontrado solución");


combinar_cifras_operaciones(57)
```

```
Resultado x Fuerza Bruta:
2+7*9-8/1 = 57
Tiempo de ejecución para algoritmo: 0.45126891136169434
```

Pregunta 5:

Calcula la complejidad del algoritmo por fuerza bruta

Respuesta:

La complejidad del algoritmo será igual a: operaciones en cada bucle * combinaciones de signos

```
Complejidad = 9 * 8 * 7 * 6 * 5 * 4 * 3 * 2 * 1
Complejidad
```

```
362880
```

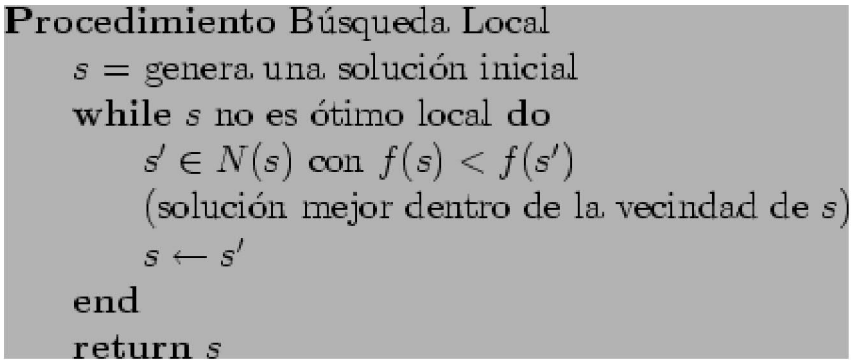
Pregunta 6:

Diseña un algoritmo que mejore la complejidad del algortimo por fuerza bruta. Argumenta porque bruta

Respuesta 6:

Aproximación Metaheuristica (Busqueda Aleatoria)

Una aproximación alternativa a la fuerza bruta sería hacer busquedas aleatorias con un límite pref final se presenta el resultado de la iteración que más se aproximó al valor objetivo.



La búsqueda aleatoria pura casi siempre converge a la solución óptima y es aplicable a casila tota debidoa los pocos supuestos necesarios para su uso.

La principal desventaja que presenta el métodoradica en que puede ser lento para encontrar unas aunqueeste problema se ve mitigado por la gran evoluciónque los sistemas cómputo tienen en la

```
import random
# Generamos una solución aleatoria
def formula_aleatoria():
    expr = e[:]
    numbers = num[:]
    solution = []
    for _ in range(len(e)):
        numChoice = random.choice(numbers)
        numbers.remove(numChoice)
        exprChoice = random.choice(expr)
        expr.remove(exprChoice)
        solution.append(numChoice)
        solution.append(exprChoice)
    solution.append(random.choice(numbers))
    #diferencia = dif_objetivo(57, str(solution))
    valor_actual = eval(''.join(solution))
    return formula, valor_actual    # solution: ['3', '/', '2', '-', '8', '+', '4', '*', '

# Calculamos la diferencia entre valor generado y valor deseado :
def diferencia(valor_deseado, valor_actual):
    return abs(valor_deseado - valor_actual)

# verificacion:
#e = ["+",'-','*','/']
#num = ["1","2","3","4","5","6","7","8","9"]
#formula, valor = formula_aleatoria()
#print(formula)
#print(valor)
#n = 57
#print(diferencia(n, formula))
#diferencia


# Generamos la formula aproximada para obtener un valor deseado:
# Parametros: n: valor deseado, it: numero de iteraciones
@calcular_tiempo
def FormulaValorDeseado(n, it):
    contador = 0
```

```
    contador = 0
    menor_diferencia = 99999999999999999
    while contador < it:
        formula, valor_actual = formula_aleatoria()
        if valor_actual == n:
            menor_diferencia = 0
            return mejor_solucion, valor_actual, menor_diferencia
        else:
          diferencia_actual = diferencia(n, valor_actual)
          if menor_diferencia > diferencia_actual:
            mejor_solucion = formula
            menor_diferencia = diferencia_actual
          contador += 1
    return mejor_solucion, valor_actual, menor_diferencia

# parametros:
e = ["+",'-','*','/']
num = ["1","2","3","4","5","6","7","8","9"]
#num = generateNumbers(9)

# EJECUTAMOS EL CALCULO:
n = 57
it = 5000
mejor_solucion, valor_actual, menor_diferencia = FormulaValorDeseado(n, it)
print(mejor_solucion)
print(valor_actual)
print(menor_diferencia)
```

```
Tiempo de ejecución para algoritmo: 0.0011212825775146484
['5', '/', '7', '*', '9', '+', '8', '-', '1']
57.0
0
```

Pregunta 7:

(*)Calcula la complejidad del algoritmo

Respuesta 7:

Podemos calcular el número de soluciones a generar para tener una probabilidad p de que la solu

$$n > \log(1-p) / \log(1- 1/m)$$

Pregunta 8:

Según el problema (y tenga sentido), diseña un juego de datos de entrada aleatorios

Respuesta 8:

```
valores_objetivo = [(random.randrange(-50,50)) for _ in range(15)]
valores_objetivo
```

```
[12, -47, -26, -29, 5, -1, 29, -2, 5, -38, -18, 19, -23, 15, 25]
```

Pregunta 9:

Aplica el algoritmo al juego de datos generado

Respuesta 9:

```
it = 5000
for i in valores_objetivo:
  n = i
  print('valor objetivo:', abs(n))
  combinar_cifras_operaciones(i)
  print('RESULTADO x BUSQUEDA ALEATORIA:')
  mejor_solucion, valor_actual, menor_diferencia = FormulaValorDeseado(n, it)
  print(mejor_solucion)
  print(valor_actual)
  print(menor_diferencia)
  print('----------------------')
```

```
-38.0
0
----------------------
valor objetivo: 18

RESULTADO FUERZA BRUTA:
1+2-7*9/3 = -18
Tiempo de ejecución para algoritmo: 0.09777569770812988
RESULTADO x BUSQUEDA ALEATORIA:
Tiempo de ejecución para algoritmo: 0.0028896331787109375
['5', '/', '7', '*', '9', '+', '8', '-', '1']
-18.0
0
----------------------
valor objetivo: 19

RESULTADO FUERZA BRUTA:
1+3*7-6/2 = 19
Tiempo de ejecución para algoritmo: 0.1099402904510498
RESULTADO x BUSQUEDA ALEATORIA:
Tiempo de ejecución para algoritmo: 0.006959199905395508
['5', '/', '7', '*', '9', '+', '8', '-', '1']
19.0
0
----------------------
valor objetivo: 23

RESULTADO FUERZA BRUTA:
1+3-6*9/2 = -23
Tiempo de ejecución para algoritmo: 0.10536503791809082
RESULTADO x BUSQUEDA ALEATORIA:
Tiempo de ejecución para algoritmo: 0.020018815994262695
['5', '/', '7', '*', '9', '+', '8', '-', '1']
-23.0
0
----------------------
valor objetivo: 15

RESULTADO FUERZA BRUTA:
1*8+9-4/2 = 15
Tiempo de ejecución para algoritmo: 0.07418513298034668
RESULTADO x BUSQUEDA ALEATORIA:
Tiempo de ejecución para algoritmo: 0.0013473033905029297
['5', '/', '7', '*', '9', '+', '8', '-', '1']
15.0
0
----------------------
valor objetivo: 25

RESULTADO FUERZA BRUTA:
1+3*9-6/2 = 25
Tiempo de ejecución para algoritmo: 0.1025705337524414
RESULTADO x BUSQUEDA ALEATORIA:
Tiempo de ejecución para algoritmo: 0.003938436508178711
['5', '/', '7', '*', '9', '+', '8', '-', '1']
25.0
0
----------------------
```