Facultad de Ingenierías
Departamento de Ingeniería en Sistemas

Estrategia 1 -Proyecto

Análisis y Diseño de Algoritmos

Alejandro González Flórez

Luis Felipe Giraldo

Jose David González Villa

Manizales, Caldas

2024

- Entrada de datos

- La TPM del sistema completo.

	primogenitalTables	
A	0,1,0,1,0,1,0,1	1,0,1,0,1,0,1,0
B	0,0,1,1,0,0,1,1	1,1,0,0,1,1,0,0
C	0,0,0,0,1,1,1,1	1,1,1,1,0,0,0,0

- El subconjunto de elementos a analizar (sistema candidato) aquí solo se requieren los elementos en t.

stateSought
100
100
100

- El subconjunto del sistema candidato a analizar (aquí deben darse los elementos tanto en t como en t+1, ya que no necesariamente se tendrán en t+1 los mismos elementos que en t)

- El estado actual de todos los elementos del sistema

$$\bullet \left(\frac{ABC^{t+1}}{ABC^{t}} \right)$$

Validación Estado Original

```
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0]
```

- Proceso:

Comenzar con un conjunto V de todos los elementos en t. Es decir, V tendrá los nodos del subsistema del conjunto candidato a analizar.

a) Inicializar W0 = ∅ y W1 = {v1}, donde v1 es un elemento arbitrario de V.

- $\left(\frac{A}{-}\right) * \left(\frac{BC}{ABC}\right)$

b) Iteración Principal: Para i = 2 hasta n (donde n es el número de nodos en V) se calcula :

- Encontrar vi ∈ V \ Wi-1 que minimiza: g(Wi-1 ∪ {vi}) - g({vi})

donde g(X) es la función EMD entre la distribución de probabilidades resultante de P(X) ⊗ P(X') y la distribución del sistema sin dividir, asi: EMD(P(X) ⊗ P(X'), P(V))

- Establecer Wi = Wi-1 ∪ {vi}

- $\left(\frac{AB}{-}\right) * \left(\frac{C}{ABC}\right)$
  - $EMD : 0.1875$
- $\left(\frac{AC}{-}\right) * \left(\frac{B}{ABC}\right)$
  - $EMD : 0.1875$
- $\left(\frac{A}{A}\right) * \left(\frac{BC}{BC}\right)$
  - $EMD : 0.0$
- $\left(\frac{A}{B}\right) * \left(\frac{BC}{AC}\right)$
  - $EMD : 0.0$

Después se Escoge el menor

Combinación Elegida

$$\bullet \left(\frac{A}{A}\right) * \left(\frac{BC}{BC}\right)$$

c) Construcción de Pares:

- El par (vn-1, vn) forma un "par candidato".

$$['AN','A','BN','CN','B','C']$$

Sistema a combinar:

$$\bullet \left(\frac{}{BC}\right) * \left(\frac{ABC}{A}\right)$$

Ya despues se considerara 0/BC que son los ultimos como un nuevo posible candidato

d) Recursión:

- Si |V| > 2, repetir el proceso con V' = V \ {vn-1, vn} ∪ {u}, donde u representa la unión de vn-1 y vn.
- Continuar hasta que |V| = 2.

$$\bullet \left(\frac{B}{BC}\right) * \left(\frac{AC}{A}\right)$$

$$'BN','B','C'], ['AN','CN','A']]$$

Constantino que los valores ahí se unieron los 2 últimos y generando la lista mencionada anteriormente

e) Evaluación Final:

- Para cada par candidato (a, b) encontrado:
  - Evaluar la división que separa {b} del resto de nodos.
- La división con el menor valor de diferencia es la solución al problema.

Al terminar se encuentra la siguiente combinación de elementos con la eficiencia de:

$$\bullet MejorCombinación : [['BN','B','C','AN',['CN','A']],[['CN','A']],0.0]$$

Entrega:

- Implementar el algoritmo descrito de manera eficiente, que:

a) Reciba la entrada de datos definida:

b) Encuentre la división del sistema siguiendo el proceso descrito.

c) Calcule el valor mínimo de diferencia asociado a la división del sistema

d) Retorne la división del sistema y el valor de diferencia respecto al sistema sin dividir.

Las pruebas se Mostraron en la parte anterior del documento

e) Halle el tiempo real empleado en su implementación.

- Consideraciones de Eficiencia:

a) Analizar la complejidad temporal y espacial del algoritmo implementado.

La complejidad temporal del algoritmo la basamos en un algoritmo de backtracking no posee mejor ni peor caso así que su eficiencia es solo una:

```
def strategy(self, ):
    for x in range(len(self.ns)):
        Todos.append(self.ns[x]+'N')

    for x in range(len(self.cs)):
        Todos.append(self.cs[x])

    while len(Todos) > 2:

        Final = self.generar_combinaciones([Todos[0]], Todos[1:], True)
        st.latex(f'{Final}')
        Arreglo = self.Cortar(Final)

        for x in Todos[3:] :
            self.min_emd = float("inf")
            Arreglo = self.generar_combinaciones(Arreglo[len(Arreglo)-1],Arreglo[:-1],True)
            Arreglo = self.Cortar(Arreglo)
            self.formatStrategie(Arreglo[0],Arreglo[1])
            st.latex(f'{Arreglo}')
            st.latex(rf"""\bullet EMD : {self.min_emd}""")
            st.latex(rf"""\bullet Mejor Combinación : {self.mejor_particion}""")

        Todos = Arreglo

    return self.mejor_particion, round(self.min_emd, 5)
```

Comenzamos con la funcion principal donde se llevara a lista de elementos y que los devolverá ordenados:

Esta tiene una eficiencia de O(N) y mas precisamente Θ (N-2)siendo N todos los nodos tanto los de futuro como presente

Y la función que genera las combinaciones

```
for i in range(len(restantes)):
    seleccionados.append(restantes[i])
    Copsel=seleccionados[:]
    Copia = restantes[:]
    Copia.remove(restantes[i])

    if not Copia:
        return seleccionados

    ns1,cs1,ns2,cs2 = self.formatStrategie(Copsel,Copia)

    arr1 = np.array(self.descomponer(ns2, cs2))
    arr2 = np.array(self.descomponer(ns1, cs1))

    partitioned_system = []

    if len(arr1) > 0:
        partitioned_system = arr1

    if len(arr2) > 0:
        partitioned_system = arr2
```

```
            # Convertir partitioned_system a array de NumPy si no lo es
            partitioned_system = np.array(partitioned_system)

            # Calcular la Distancia de Wasserstein (EMD)
            emd_distance = wasserstein_distance(self.original_system,partitioned_system)
            st.latex(rf"""\bullet EMD : {emd_distance}""")

            if (emd_distance >= 0.0) and (emd_distance < self.min_emd):
                self.min_emd = emd_distance
                self.mejor_particion = [Copsel,Copia,emd_distance]

        Opciones.append([Copsel,Copia,emd_distance])
        seleccionados.remove(restantes[i])

    st.subheader("Combinación Elegida",divider="gray")
    self.formatStrategie(self.mejor_particion[0],self.mejor_particion[1])

    seleccion = min(Opciones, key=lambda x: x[2])
    Final= self.generar_combinaciones(seleccion[0], seleccion[1], False)

    if(Combinacion[0]):
        Final=[Final[:Combinacion[1]]]+Final[Combinacion[1]:]
        #st.latex(f"{Final}")
    return Final
```

Ignorando las funciones que sólo dibujan nos genera que La eficiencia sea de O(n-1)
Lo que daría que la eficiencia sería

una sumatoria doble

la primera I = 1 hasta N-2 y dentro desde J = 2 hasta N-I

$$= \sum_{i=1}^{N-2} \sum_{j=2}^{N-i} j$$

Para separarlas como sumatoria cuadratica podríamos separarlas para que queden

$$\frac{1}{2}\sum_{k=2}^{N-1} k^2 + \frac{1}{2}\sum_{k=2}^{N-1} k$$

Ahora resolvemos cada 1

$$\sum_{k=2}^{N-1} k^2 = \frac{(N-1)N(2N-1)}{6} - 1$$

$$\sum_{k=2}^{N-1} k = \frac{(N-1)N}{2} - 1.$$

Quedando la eficiencia:

$$\frac{1}{2}\left(\frac{(N-1)N(2N-1)}{6} - 1\right) + \frac{1}{2}\left(\frac{(N-1)N}{2} - 1\right) - (N-2)$$

b) Buscar optimizaciones que puedan mejorar el rendimiento, especialmente para sistemas grandes (estructuras de datos apropiadas, repeticiones de cálculos, almacenamiento y acceso a los datos etc.).

Una forma óptima fue de hacer la función de forma backtracking para poder hacer los casos de selección por cada elemento ya seleccionado, pensamos que una forma de optimizar la forma de creación es con programación dinámica guardando dichos valores en una tabla con los casos ya calculados para poder evitar rehacer las regionalizaciones y el emd.

- Implementación y Pruebas:

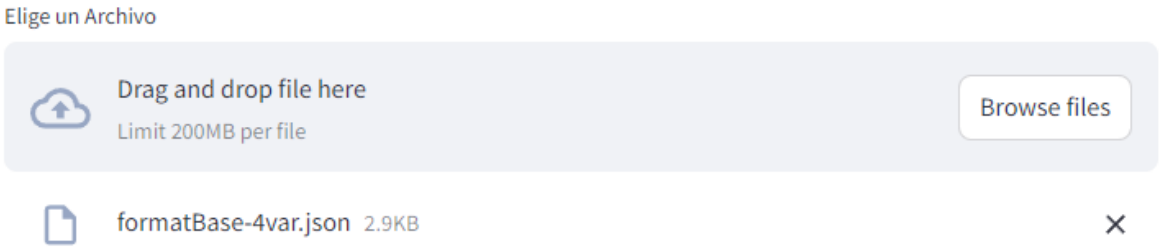
a) Implementar el algoritmo en un lenguaje de programación de su elección.

R/ Todo el código fue implementado en Python con diferentes librerías para el emd y diferentes multiplicaciones

b) Crear una interfaz para ingresar diferentes TPMs y conjuntos de datos (datos pueden ingresar por consola o leídos desde un archivo).

R/

Se hizo por medio de una carga de archivos por medio de un json



c) Desarrollar casos de prueba para verificar la correctitud del algoritmo

.

d) Comparar el rendimiento con un enfoque de fuerza bruta para sistemas pequeños.

$$[['BN','CN'],'a',['b','c','AN']]$$

$$\bullet EMD : inf$$

$$\bullet MejorCombinación : [['BN','CN','a',['b','c'],'AN'],['AN'],0.2421875]$$

Mejor Partición encontrada (('A', ''), ('BC', 'ABC'))

Mejor EMD 0.24219

En Ambos Casos se encuentra el mismo Valor

- Extensiones Opcionales:

a)Visualización de divisiones y distribuciones de probabilidad.

Sistema a combinar:

- $\left(\frac{A}{bca}\right) * \left(\overline{BC}\right)$

- $\left(\frac{A}{bc}\right) * \left(\frac{BC}{a}\right)$

$$[['b','c','AN'],['a','BN','CN']]$$

b)Optimización para sistemas separados o con estructuras particulares.
c)Análisis de cómo diferentes características del sistema afectan la función de diferencia.