

Informe Proyecto Final

Análisis y Diseño de Algoritmos

Trabajo realizado por:

JUAN FELIPE DUQUE PUERTA

Docente:

luzenith_g@ucaldas.edu.co

Ingeniería de Sistemas y Computación

Universidad de Caldas

Manizales - Caldas

Febrero del 2023

Despliegue y Documentación del Usuario:

Guía de Instalación:

- Requisitos Previos:

1. Lenguaje de programación Python3

- Pasos de Instalación:

1. Clonar el Repositorio:

```
> git clone https://github.com/pipeduque/descomposition-cut-algorithms.git
```

2. Instalar Dependencias:

```
> pip install numpy
```

```
> pip install networkx
```

```
> pip install scipy
```

```
> pip install matplotlib
```

4. Ajustar las siguientes constantes en main.py según las pruebas a realizar

```
# Lista de estados del sistema
states = [
    [0, 0, 0],
    [1, 0, 0],
    [0, 1, 0],
    [1, 1, 0],
    [0, 0, 1],
    [1, 0, 1],
    [0, 1, 1],
    [1, 1, 1],
]

# Lista de probabilidades del sistema
probabilities = [
    [1, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 1, 0, 0, 0],
```

```
[0, 0, 0, 0, 0, 1, 0, 0],  
[0, 1, 0, 0, 0, 0, 0, 0],  
[0, 1, 0, 0, 0, 0, 0, 0],  
[0, 0, 0, 0, 0, 0, 0, 1],  
[0, 0, 0, 0, 0, 1, 0, 0],  
[0, 0, 0, 1, 0, 0, 0, 0],  
]  
  
# Estado futuro del sistema  
ns = "AC"  
  
# Estado actual del sistema  
cs = "ABC"  
  
# Valor del estado actual del sistema  
cs_value = [1, 0, 0]
```

5. Iniciar el Servidor de descomposición y cortes

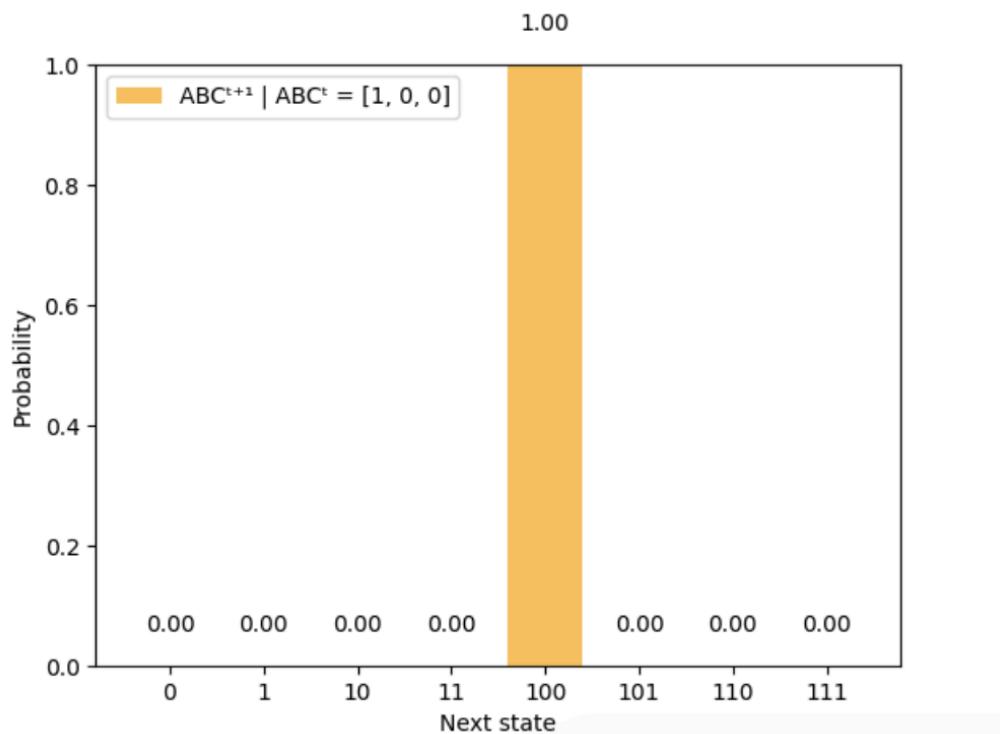
```
> cd src  
> main.py
```

Manual de Usuario:

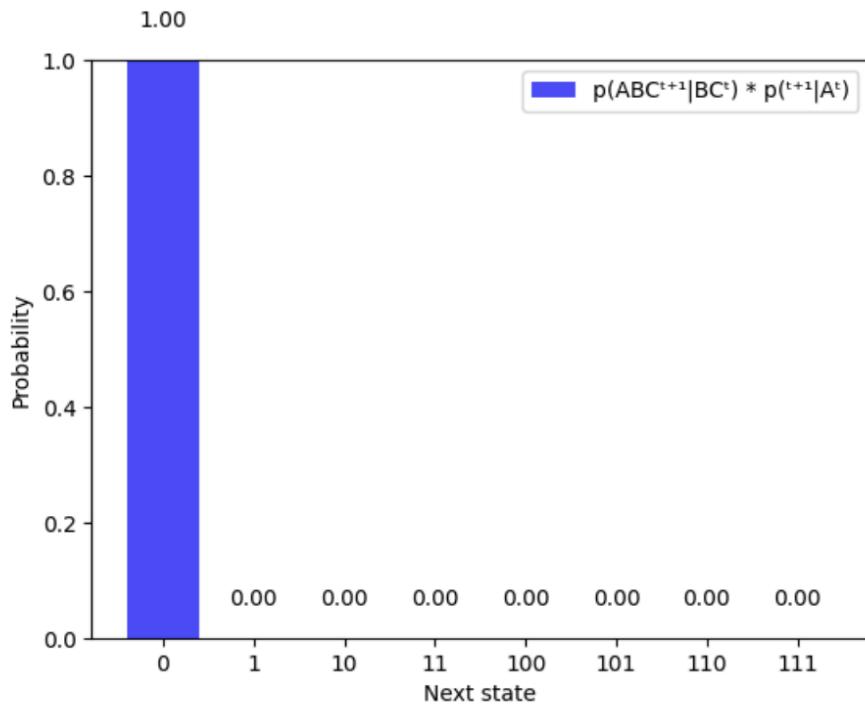
Después de correr el programa tendremos los siguientes resultados en consola

Para Decomposition:

```
Probabilities
[1, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 1, 0, 0, 0]
[0, 0, 0, 0, 0, 1, 0, 0]
[0, 1, 0, 0, 0, 0, 0, 0]
[0, 1, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 1]
[0, 0, 0, 0, 0, 1, 0, 0]
[0, 0, 0, 1, 0, 0, 0, 0]
*****
Sistema original
ABCt+1 | ABCt
```

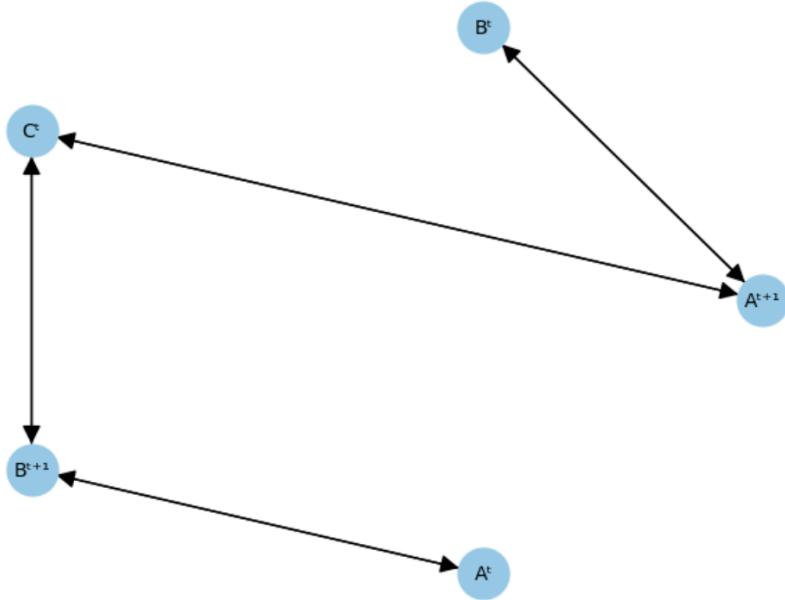


 $(ABC \mid BC) * (\mid A)$
Earth Mover's Distance: 0.0
Partición sin perdida

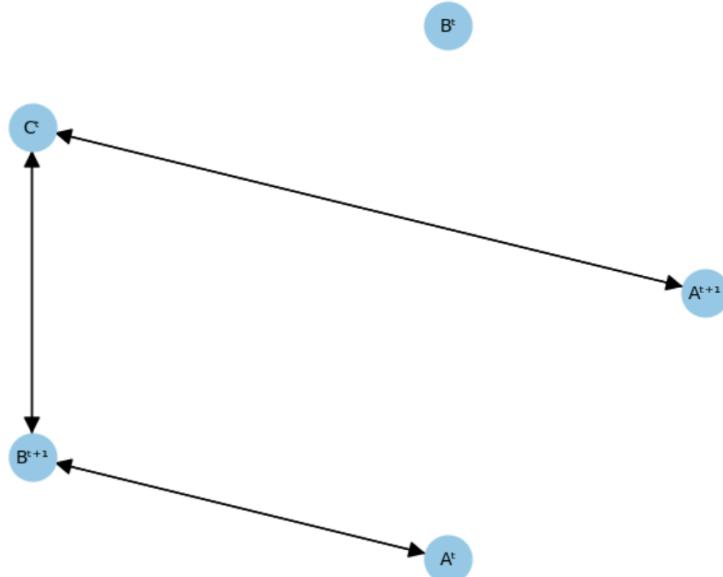


Para Cut:

Grafo principal

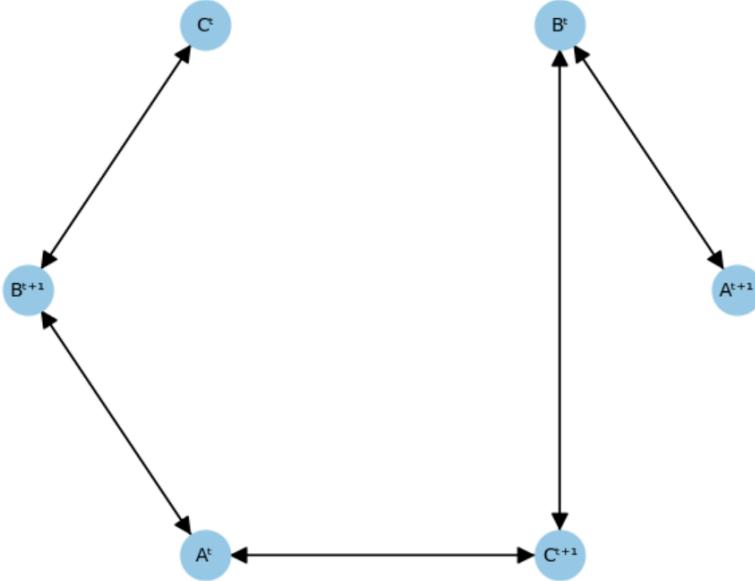


Variable actual A^{t+1}
cortando B^t de A^{t+1}
partition: $(\emptyset^{t+1} | B^{t+1})$ y $(AB^{t+1} | AC^t)$



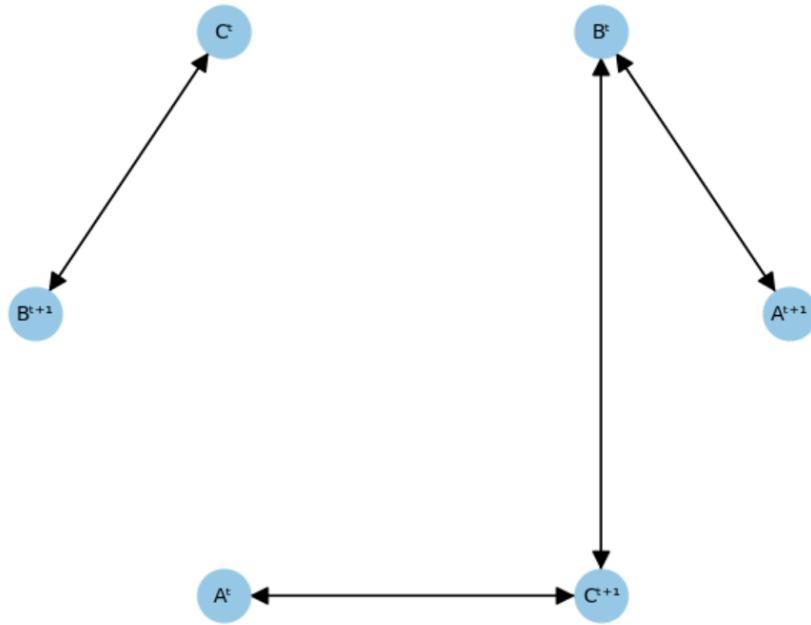
Earth Mover's Distance: 0.515625
Bipartición generada
minima partición actualizada
bipartición restaurada con costo: 0.515625
----- ***** -----

Variable actual A^{t+1}
cortando C^t de A^{t+1}
partition: $(\emptyset^{t+1} | C^{t+1})$ y $(ABC^{t+1} | AB^t)$



Earth Mover's Distance: 0.0
No bipartición generada
conexión elimina sin perdida de información

Variable actual B^{t+1}
cortando A^t de B^{t+1}
partition: $(\emptyset^{t+1} | A^{t+1})$ y $(ABC^{t+1} | BC^t)$



Earth Mover's Distance: 0.0
Bipartición generada
minima partición alcanzada

Perspectivas Futuras:

- Se planea proporcionar una interfaz front-end para la entrada de datos y la presentación de resultados

Descomposición (Programación dinámica)

Diseño del Algoritmo:

Descripción General del Algoritmo:

La estrategia del algoritmo se basa en utilizar programación dinámica para calcular la distribución de probabilidades del sistema original y las distribuciones resultantes de cada posible descomposición en dos subsistemas. La idea principal es encontrar la descomposición que implique menor pérdida de información al dividir el sistema en dos partes sin el uso de fuerza bruta durante el cálculo de cada solución.

1. Cálculo de Distribución Original:

- Utilizar la matriz de probabilidades para calcular la distribución de probabilidades del sistema en el estado inicial.
- Aplicar marginalización en el futuro y en el tiempo actual para obtener el cálculo de la Distribución Original.

2. Descomposición del Sistema:

- Dividir el sistema original en soluciones óptimas, trabajando con una parte a la vez.
- Calcular las distribuciones de probabilidades para cada parte utilizando programación dinámica:

Ejemplo:

$$\underbrace{ABC^{t+1} \mid AC^t = 10}_{\text{Distribución original}}$$

La Distribución original dada, puede ser distribuida en las siguientes particiones:

34. $\frac{A}{\emptyset} \text{ y } \frac{BC}{AC}$ 13 ⊗ 16	32. $\frac{B}{\emptyset} \text{ y } \frac{AC}{AC}$ 14 ⊗ 17	33. $\frac{AB}{\emptyset} \text{ y } \frac{C}{AC}$ 18 ⊗ 3	
$\frac{AB}{C} \text{ y } \frac{C}{A}$ 19 ⊗ 7	35. $\frac{AC}{\emptyset} \text{ y } \frac{B}{AC}$ 20 ⊗ 2	36. $\frac{BC}{\emptyset} \text{ y } \frac{A}{AC}$ 21 ⊗ 1	37. $\frac{ABC}{\emptyset} \text{ y } \frac{\emptyset}{AC}$ 22 ⊗ 4
$\frac{\emptyset}{A} \text{ y } \frac{ABC}{C}$ 8 ⊗ 23	39. $\frac{A}{A} \text{ y } \frac{BC}{C}$ 5 ⊗ 24	40. $\frac{B}{A} \text{ y } \frac{AC}{C}$ 6 ⊗ 25	41. $\frac{AB}{A} \text{ y } \frac{C}{C}$ 26 ⊗ 11
$\frac{AB}{AC} \text{ y } \frac{C}{\emptyset}$ 27 ⊗ 15	43. $\frac{AC}{A} \text{ y } \frac{B}{C}$ 28 ⊗ 10	44. $\frac{BC}{A} \text{ y } \frac{A}{C}$ 29 ⊗ 9	45. $\frac{AC}{A} \text{ y } \frac{\emptyset}{C}$ 30 ⊗ 12

Sí empezamos con la partición

$$\frac{A}{\emptyset} \text{ y } \frac{BC}{AC}$$

Revisamos la primera parte ($A^{t+1} | \emptyset^t$)

- La cuál no se puede descomponer más y no ha sido calculada anteriormente (no está en memoria), debemos calcular esta probabilidad y guardar el resultado en memoria.

Luego Revisamos la segunda parte ($BC^{t+1} | AC^t$), para este, nos damos cuenta que se puede descomponer de 4 maneras:

- $(B^{t+1} | A^t)$
- $(C^{t+1} | A^t)$
- $(B^{t+1} | C^t)$
- $(C^{t+1} | C^t)$

como ninguna de estas ha sido calculada anteriormente procedemos a calcular todas las probabilidades y guardar el resultado de cada una en memoria

Continuamos con la siguiente partición

$$\frac{B}{\emptyset} \text{ y } \frac{AC}{AC}$$

Empezamos con la primera parte ($B^{t+1} | \emptyset$), la cual no se puede descomponer más y no ha sido calculada anteriormente (no está en memoria), debemos calcular esta probabilidad y guardar el resultado en memoria.

Luego continuamos con la segunda parte ($AC^{t+1} | AC^t$), para este, nos damos cuenta que se puede descomponer de 4 maneras:

- $(A^{t+1} | A^t)$
- $(C^{t+1} | A^t)$
- $(A^{t+1} | C^t)$
- $(C^{t+1} | C^t)$

Para este punto en memoria ya se encuentra calculado dos de ellas:

- $(C^{t+1} | A^t)$
- $(C^{t+1} | C^t)$

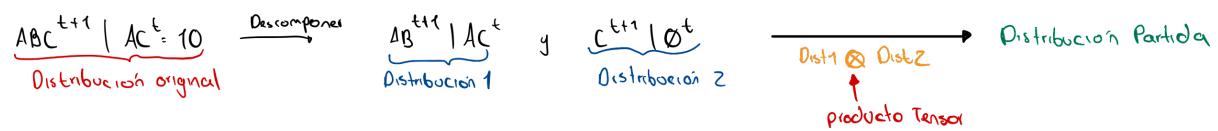
Entonces podemos ahorrarnos este proceso y solo calcular las dos faltantes y guardar el valor en memoria. Continuamos con este proceso y gracias a memoization se habrán guardado en memoria las siguientes soluciones óptimas:

$1 \ P(A^{t+1} AC^t)$	$5 \ P(A^{t+1} A)$	$9 \ P(A^{t+1} C)$	$13 \ P(A^{t+1} \emptyset)$
$2 \ P(B^{t+1} AC^t)$	$6 \ P(\emptyset^{t+1} A)$	$10 \ P(\emptyset^{t+1} C)$	$14 \ P(\emptyset^{t+1} \emptyset)$
$3 \ P(C^{t+1} AC^t)$	$7 \ P(C^{t+1} A)$	$11 \ P(C^{t+1} C)$	$15 \ P(C^{t+1} \emptyset)$
$4 \ P(\emptyset^{t+1} AC^t)$	$8 \ P(\emptyset^{t+1} A)$	$12 \ P(\emptyset^{t+1} C)$	

De las cuales podemos calcular mediante Producto Tensor la solución a particiones con menor grado de descomposición, este proceso se explica en el siguiente punto.

3. Producto Tensor y Distribución Partida:

- Aplicar el producto tensor a las distribuciones resultantes de las dos partes.
- Obtener la Distribución Partida, que representa la distribución del sistema descompuesto.



4. Comparación y Cálculo de Información Ganada:

- Comparar la Distribución Original con la Distribución Partida utilizando la Earth Mover's Distance.
- Calcular la Información Ganada como la diferencia entre estas distribuciones.

5. Búsqueda de la Descomposición Óptima:

- Iterar sobre todas las posibles formas de descomponer el sistema en dos subsistemas hasta encontrar una descomposición que no implique pérdida.
- En cada iteración, calcular la pérdida de información con respecto al Sistema Original y registrar la descomposición que minimiza esta pérdida de información.

Justificación de la Estrategia:

La elección de la programación dinámica se basa en la eficiencia de los subproblemas solucionados y la posibilidad de reutilizar resultados previos, reduciendo así la redundancia en los cálculos.

Cortes (Enfoque voraz)

Diseño del Algoritmo:

Descripción General del Algoritmo:

El algoritmo propuesto sigue una estrategia de eliminación de conexiones de manera iterativa para verificar si la eliminación de cada conexión afecta la información del sistema. Aquí se presenta una descripción detallada del proceso:

Inicialización:

- Representar el sistema como un grafo dirigido de nodos interconectados, donde cada nodo representa un canal o elemento, y las aristas representan las conexiones causales.
- Inicializar la matriz de probabilidades del sistema.

Iteración de Eliminación:

- Seleccionar una conexión a eliminar.
- Modificar la matriz de probabilidades del nodo destino de la conexión, marginalizado sobre los estados del nodo fuente.
- Verificar si la eliminación de la conexión genera una bi-partición en el sistema.

Verificación de bi-partición:

Sí genera k-partición (el grafo está dividido en 2 o más conjuntos):

- Recuperar la conexión si implica pérdida y actualizar la mínima partición si esta es la menor encontrada.
- Si el corte no implica pérdida termina el proceso ya que esta es la mínima partición encontrada.

No genera k-partición:

- Recupera la conexión si implica pérdida
- No recupera la conexión si no implica pérdida y continua con el siguiente corte

Repetición del Proceso:

- Si no se ha generado una partición sin pérdida , repetir el proceso seleccionando la siguiente conexión a eliminar
- Repetir hasta que se logre una partición o no haya más conexiones para eliminar.

Justificación de la Estrategia:

Ventajas:

- La estrategia es sencilla y fácil de entender.
- Se basa en la eliminación de conexiones en manera ordenada hasta encontrar una partición sin pérdida, permitiendo ahorrar recursos al no hacerse una exploración exhaustiva del impacto de cada conexión en la información del sistema.

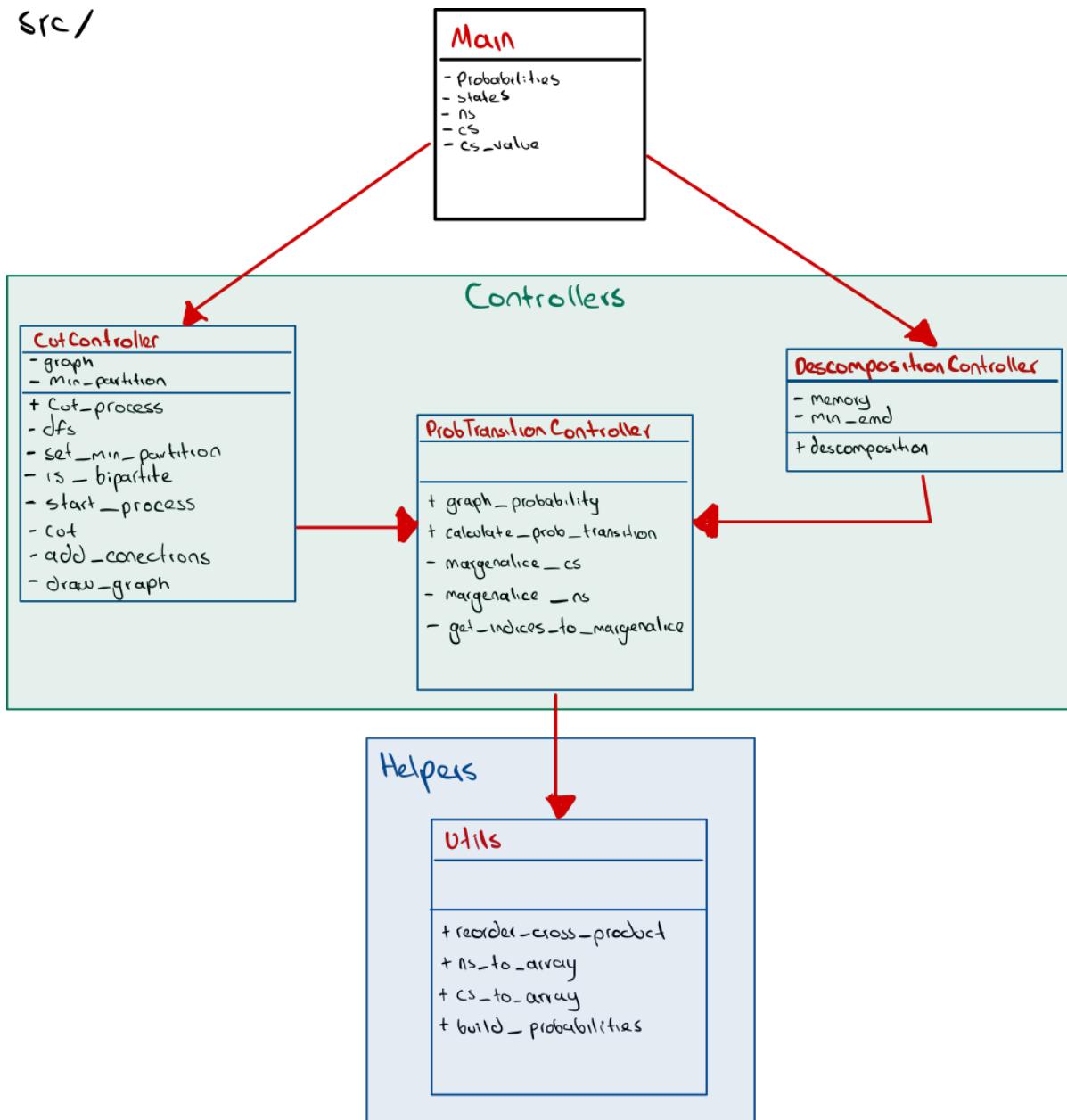
Desventajas:

- La estrategia puede volverse costosa en términos de tiempo para sistemas grandes debido a su enfoque iterativo.
- No garantiza encontrar la solución óptima global, ya que depende del orden en que se eliminan las conexiones.

En resumen, la estrategia voraz fue elegida por su simplicidad y capacidad para explorar diferentes configuraciones de conexiones, aunque podría mejorarse en eficiencia mediante técnicas más avanzadas en la selección de conexiones.

Diagramas y Pseudocódigo:

src /



Main			
Atributo	Descripción	Tipo	Ejemplo
probabilites	probabilidades del sistema para cada estado-nodo	lista de listas	<pre>probabilities = [[1, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 1, 0, 0, 0], [0, 0, 0, 0, 0, 1, 0, 0], [0, 1, 0, 0, 0, 0, 0, 0], [0, 1, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 1], [0, 0, 0, 0, 0, 1, 0, 0], [0, 0, 0, 1, 0, 0, 0, 0],]</pre>
states	estados presentes en el sistema	lista de listas	<pre>states = [[0, 0, 0], [1, 0, 0], [0, 1, 0], [1, 1, 0], [0, 0, 1], [1, 0, 1], [0, 1, 1], [1, 1, 1],]</pre>
ns	valor para estado futuro	string	<pre>ns = "AC" You, 3 hours ago</pre>
cs	valor para estado actual	string	<pre>cs = "ABC"</pre>

DescompositionController			
Atributo	Descripción	Tipo	Ejemplo
memory	estructura de datos clave-valor para guardar por cada clave la partición descompuesta	diccionario	<pre>{ 'BC': {'A': array([1., 0.])}, 'AC': {'A': array([0.5, 0.5])}, 'AB': {'A': array([0.5, 0.5])}, 'C': {'A': array([0.5, 0.5])},</pre>

	y como valor el cálculo de la probabilidad para esta, sirve para ahorrar cálculos		'A': {'A': array([0.25, 0.75])}, ''': {'A': array([0.25, 0.75])}, 'B': {'A': array([0.5, 0.5])}, 'ABC': {'A': array([1., 0.])} }
min_emd	valor del mínimo emd distance encontrado para las descomposiciones	double	Earth Mover's Distance: 0.375

Funciones:

- **start_process(ns, cs, cs_value, probabilities, states):**

Encarga de iniciar la memoria y el proceso de descomposición, armando las posibles particiones y llamando para cada una de ellas la función de programación dinámica “decomposition”

- **decomposition(ns, cs, memory, states):**

Función recursiva que calcula las distribuciones de probabilidades para cada partición utilizando programación dinámica:

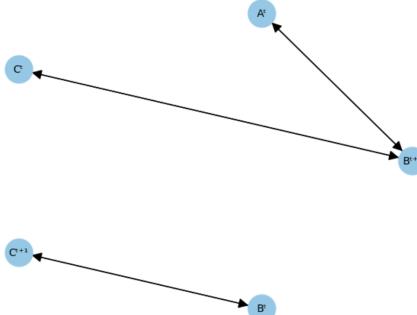
- Al principio de la función recursiva se busca en la memoria si ya fue calculada previamente la distribución de probabilidad de la descomposición en curso, así se logra ahorrar el cálculo de varias descomposiciones
- Si no se encuentra en la memoria, se intenta descomponer más si aún es posible
- Si ya no se puede descomponer más, se procede a calcular la distribución de probabilidad para la descomposición actual y se guarda en memoria la descomposición junto a su resultado

descomposition(ns, cs, memory)

if $\text{Memory.get}(ns) \neq \text{null}$ and $\text{Memory.get}(ns).\text{get}(cs) \neq \text{null}$
 ↑ $\text{Memory.get}(ns).\text{get}(cs)$

if $\text{len}(ns) = 1$
 Value \leftarrow calculate transition
 ↑ Value

Value $\leftarrow \emptyset$
for $i = 0$ to $\text{len}(ns)$
 Value $\leftarrow \text{tensor}(\text{Value}, \text{descomposition}(ns[i], cs, memory))$
endfor
memory.set(ns).set(cs).set(Value)
↑ Value

CutController			
Atributo	Descripción	Tipo	Ejemplo
graph	grafo de la librería nx para guardar los nodos y conexiones	nx.DiGraph()	 <pre> graph TD A[A^t] --> B[B^{t+1}] A[A^t] --> C[C] C[C] --> Ct[C^{t+1}] B[B^{t+1}] --> Ct[C^{t+1}] </pre>
min_bipartition	<p>estructura de datos con la información de la mínima bipartición encontrada de las iteraciones realizadas</p> <ul style="list-style-type: none"> partitioned_system: distribución de probabilidad del sistema con mínima bipartición. partition: bipartición mínima, edge_to_remove_1: arista a remover edge_to_remove_2: arista a remover emd: emd distance de la mínima bipartición encontrada 	diccionario	<pre> { 'partitioned_system': array([3.0 625, 0., 0., 0.]), 'partition': '(ø^{t+1} B^t) y (AB^{t+1} AC^t)', 'edge_to_remove_1': 'B^t', 'edge_to_remove_2': 'A^{t+1}', 'emd': 0.515625 } </pre>

Funciones:

- **dfs(G, node, visited, end_node):**

Realiza recorrido por profundidad de un grafo en busca de un nodo final desde un nodo dado, en caso de encontrarlo termina el recorrido

- **is_bipartite(G, start_node, end_node):**

Determina si un grafo está bi-particionado o no, inicializando la lista de nodos visitados y llamando la función de recorrido por profundidad con un nodo inicial y un nodo final

- **set_min_partition (min_partition, partition, partitioned_system,**

emd_distance, edge_to_remove_1, edge_to_remove_2):

actualiza los datos de la mínima bi partición encontrada

- **add_connections(G, ns, cs):**

crea nodos y conexiones al grafo según el estado futuro y estado actual dados

- **draw_graph(G):**

función encargada de graficar el grafo de la librería nx con la librería matplotlib.pyplot

- **cut_process(G, ns, cs, cs_value, min_partition, probabilities, states):**

Encarga de iniciar la memoria y el proceso de cortado, recorriendo las conexiones existentes en el grafo y generando las particiones después de cada cortado y según la Distancia de Wasserstein (EMD) de cada corte:

Sí genera bipartición:

- Recupera la conexión si implica pérdida y actualiza la mínima partición si esta es la menor encontrada
- Si el corte no implica pérdida termina el proceso ya que esta es la mínima partición encontrada

No genera bipartición:

- Recupera la conexión si implica pérdida
 - No recupera la conexión si no implica pérdida y continua con el siguiente corte
-
- **start_process(ns, cs, cs_value, probabilities, states):**
Encargada de:
 - iniciar el grafo
 - llamar a la función de creación de nodos y conexiones
 - verificar si el grafo es bipartido desde el inicio para no continuar con el proceso
 - llamar al proceso de cortado
 - imprimir resultados

ProbabilityTransitionController

Funciones:

- **graphProbability(array, color, label)**
Función encargada de graficar la distribución de probabilidad con la librería matplotlib.pyplot
- **getIndicesToMarginalize(states, state)**
Función encargada de obtener los índices que deben ser marginalizados según el estado actual dado y el total de estados, también retorna el índice donde se encuentra el estado actual

```

getIndicesToMarginalice
( statuses [statuses][elements] , state [elements])
    indices = []
    availableIndices ← []
    for i ← 0 to elements:
        if state[i] ≠ null
            availableIndices.push(i)
            csValue = str(state[i]) + csValue
    endfor
    for i ← ∅ to statuses:
        key = ''
        for j ← 0 to len(availableIndices):
            key ← key + string(statuses[i][availableIndices[j]])
        endfor
        indices[key] = indices.get(key) + int(csValue, 2)
    endfor
    ↑ indices , int(csValue, 2)

```

- **marginaliceNextState(nsIndices, probabilities)**

Marginaliza las columnas de la matriz de probabilidad según los índices del estado futuro dados a marginalizar, sumando los valores de las columnas marginalizadas

```

marginaliceNextstate (nsIndices { }, probabilities[statuses][statuses])
nSTransitionTable ← [statuses] [len(nsIndices)]
currentColumn = 0
for Indices in nsIndices
    for i ← ∅ to statuses
        probability ← 0
        for j ← 0 to len(Indices):
            probability += probabilities[i][Indices[j]]
        endfor
        nSTransitionTable[i][currentColumn] = probability
    endfor
    currentColumn += 1
endfor
↑ nSTransitionTable

```

- **marginaliceCurrentState(csIndices, nsTransitionTable)**

Marginaliza las filas de la matriz resultante de haber marginalizado el estado futuro,

según los índices del estado presente dados a marginalizar, sumando los valores de las columnas marginalizadas y dividiéndolos por la cantidad de filas marginalizadas

```

MarginaliceCurrentState (csIndices [], nsTransitionTable [states][nselements]
    csTransitionTable ← [len(csIndices)][nselements]
    CurrentRow = 0

    for indices in csIndices
        for i ← 0 to nselements
            probability ← 0
            for j ← 0 to len(indices)
                probability += nsTransitionTable [indices[j]][i]
            endfor
            csTransitionTable [CurrentRow][i] = probability / len(indices)
        endfor
        CurrentRow ++
    endfor

    ↑ csTransitionTable

```

- **probabilityTransitionTable(currentState, nextState, probabilities, states)**

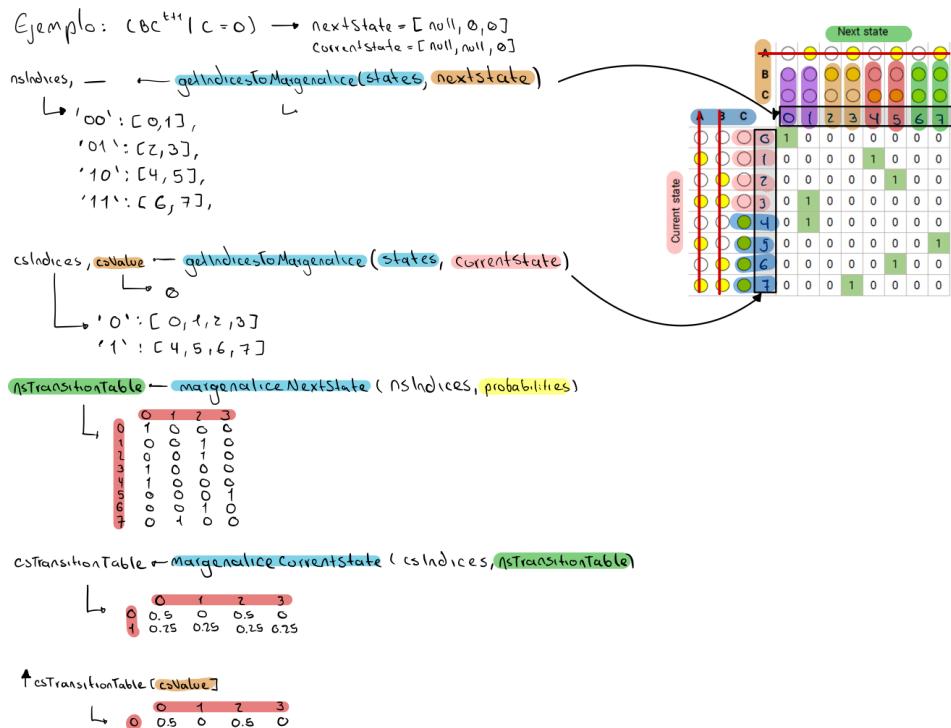
Obtiene el resultado del proceso de cálculo de probabilidad de transición dado un estado actual, un estado futuro, los estados del sistema y la matriz de probabilidades

```

probabilityTransitionTable (probabilities [states][nstates], states [states][elements], currentState [elements], nextState [elements])
    nsIndices, _ ← getIndicesToMarginalice(states, nextState)
    csIndices, csValue ← getIndicesToMarginalice (states, currentState)
    nsTransitionTable ← marginaliceNextState (nsIndices, probabilities)
    csTransitionTable ← marginaliceCurrentState (csIndices, nsTransitionTable)
    ↑ csTransitionTable [csValue]

```

Ejemplo:



Helpers

Funciones:

- **reorder_cross_product(cross_product)**

reorganiza el arreglo resultante del producto tensor de dos particiones para seguir la convención establecida

- **ns_to_array(letras)**

convierte las letras dadas para el estado futuro en un arreglo

- **cs_to_array(cs, cs_value)**

convierte las letras dadas para el estado actual en un arreglo que corresponda a los valores dados para dichas letras

- **build_probabilities(probabilities, len_cs):**

Extiende la matriz de probabilidad para que cumpla con la convención estado-estado

Manejo de Casos Especiales y Límites:

El algoritmo maneja casos especiales al explorar todas las posibles descomposiciones, asegurando que no se pasen por alto combinaciones únicas.

Optimizaciones Implementadas:

Uso de memoization para almacenar resultados de subproblemas ya resueltos y evitar recálculos innecesarios durante la programación dinámica.

Comparaciones con Otros Enfoques:

Se descartaron enfoques de fuerza bruta debido a su ineficiencia para explorar todas las combinaciones posibles, especialmente en sistemas grandes.

Manejo de Recursos:

- El algoritmo utiliza memoization para gestionar eficientemente los recursos de memoria.
- Se implementan técnicas para minimizar el uso de recursos, especialmente durante la búsqueda de la descomposición óptima.

Escalabilidad:

El diseño algorítmico permite que el sistema sea escalable, aunque la complejidad exponencial en la búsqueda de descomposiciones puede ser un factor limitante.

Documento de Diseño del Software (DDS):

Arquitectura del Sistema:

La arquitectura del sistema sigue el modelo MVC (Modelo-Vista-Controlador) para garantizar la separación de preocupaciones y facilitar la escalabilidad y mantenimiento del sistema.

Capa de Presentación (Front-End):

La capa de presentación está implementada utilizando las siguientes librerías de python:

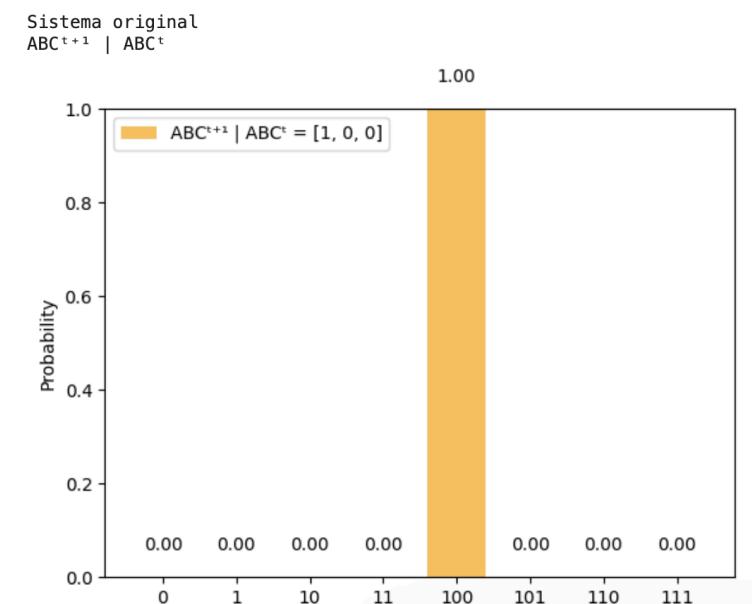
- Matplotlib.pyplot:

- Módulo de la biblioteca Matplotlib que proporciona una interfaz para crear gráficos estáticos, visualizaciones y trazados en 2D. Matplotlib es una biblioteca de visualización ampliamente utilizada en la comunidad de Python para crear gráficos de alta calidad.

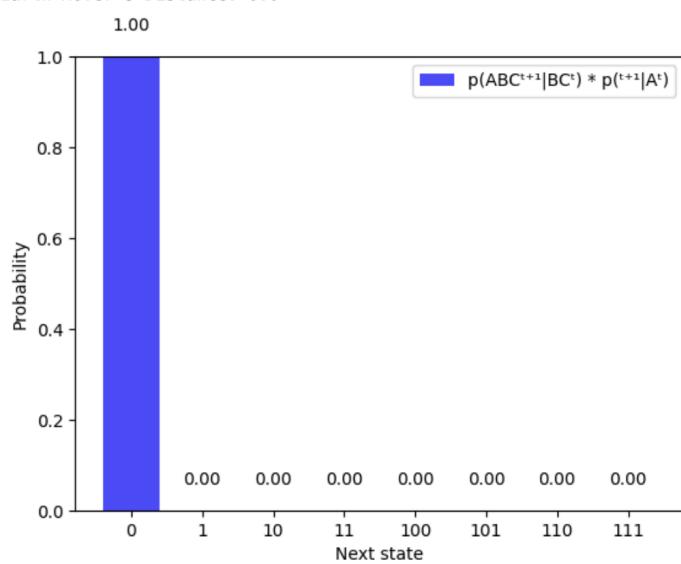
- NetworkX:

- Biblioteca de Python utilizada para la creación, manipulación y estudio de la estructura, dinámica y funciones de redes complejas. Ofrece herramientas para trabajar con grafos y realizar análisis de redes.

- Vista principal del gráfico de probabilidad del Sistema Original y la Distribución partida:

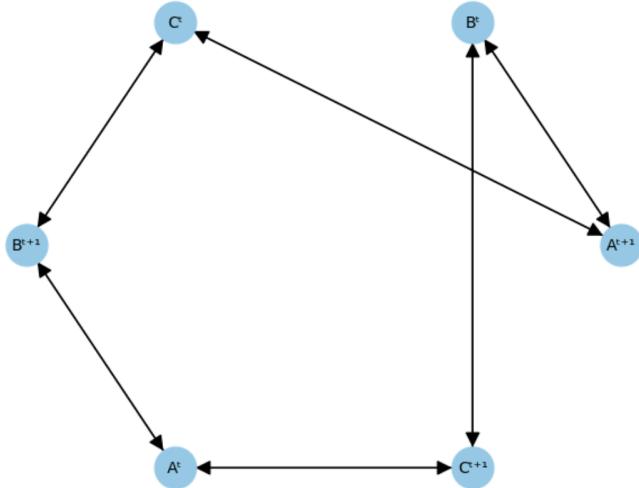


 $(ABC \mid BC) * (\mid A)$
 Earth Mover's Distance: 0.0

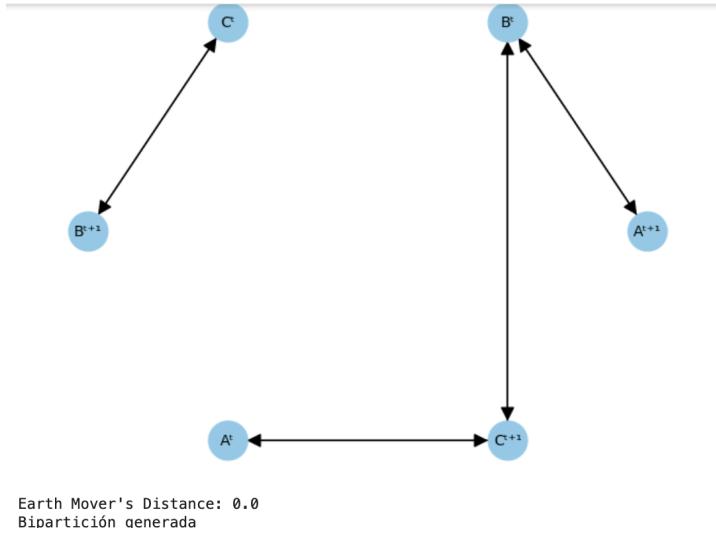


- Vista principal del grafo principal de la partición inicial vs la partición con cortes resultante

Grafo principal



Grafo final



Capa de Lógica de Negocio (Back-End):

El back-end se encarga de la lógica de negocio, la gestión de datos y la comunicación con la memoria. Se implementa utilizando python.

Diseño de la Base de Datos:

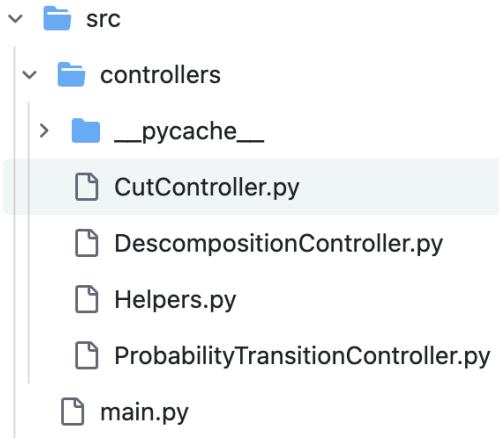
- Tabla de Probabilidades: Almacena la matriz de probabilidades del sistema.
- Registro de Descomposiciones (memoria): Guarda información sobre las descomposiciones realizadas, incluyendo la distribución original, distribuciones de partes y la información ganada.

Implementación:

Estructura del Código:

Back-End:

La estructura del código del back-end sigue una organización modular basada en el modelo MVC. Se utilizan buenas prácticas de codificación.



Tecnologías Utilizadas:

- Lenguaje: Python

- Librerías:

Matplotlib (`import matplotlib.pyplot as plt`):

- Matplotlib es una biblioteca de visualización en Python que se utiliza para crear gráficos estáticos, gráficos interactivos y visualizaciones en 2D y 3D.

NumPy (`import numpy as np`):

- NumPy es una biblioteca fundamental para la computación numérica en Python. Proporciona soporte para arreglos multidimensionales, funciones matemáticas de alto rendimiento y operaciones de álgebra lineal.

NetworkX (`import networkx as nx`):

- NetworkX es una biblioteca para el análisis de redes y grafos en Python. Proporciona estructuras de datos y algoritmos para la creación, manipulación y estudio de la estructura, dinámica y funciones de redes complejas.

Scipy (`from scipy.stats import wasserstein_distance`):

- Propósito: SciPy es una biblioteca para matemáticas, ciencia e ingeniería que se basa en NumPy. Proporciona módulos adicionales

para optimización, álgebra lineal, estadísticas, procesamiento de señales y más.

- El módulo `scipy.stats` incluye funciones estadísticas, y `wasserstein_distance` en particular se utiliza para calcular la distancia de Wasserstein, que mide la diferencia entre dos distribuciones de probabilidad.

Manejo de Errores y Excepciones:

- Se implementa un manejo adecuado de errores y excepciones en todas las capas del sistema.

Problemas y Soluciones:

- Problema: Rendimiento en la búsqueda exhaustiva de descomposiciones.
- Solución: Se implementó memoization para almacenar resultados intermedios y evitar recálculos.

Perspectivas Futuras:

- Optimización del Algoritmo: Explorar técnicas de optimización con concurrencia para mejorar la eficiencia en la búsqueda de descomposiciones.
- Interfaz Gráfica Mejorada: Mejorar la interfaz de usuario para proporcionar una experiencia más intuitiva.

Evaluación de la Eficiencia del Diseño Algorítmico:

- Posibles Mejoras:
- Explorar algoritmos más eficientes para la comparación de distribuciones.
- Evaluar el rendimiento en conjuntos de datos más grandes y considerar técnicas de paralelización.

- Identificación de Áreas de Mejora:
- Refinar la estrategia de programación dinámica para optimizar el cálculo de distribuciones.

- Considerar la implementación de heurísticas para acelerar la búsqueda de descomposiciones.

Pruebas y Validación del Algoritmo:

Se realizaron pruebas específicas para validar la eficiencia del algoritmo, utilizando conjuntos de datos pequeños y grandes.

Pruebas para Decomposition:

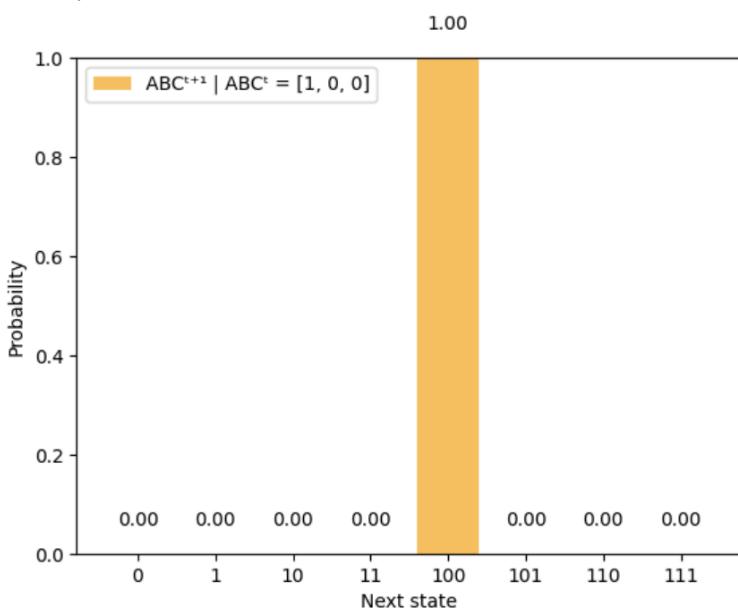
```
states = [
    [0, 0, 0],
    [1, 0, 0],
    [0, 1, 0],
    [1, 1, 0],
    [0, 0, 1],
    [1, 0, 1],
    [0, 1, 1],
    [1, 1, 1],
]

probabilities = [
    [1, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 1, 0, 0, 0],
    [0, 0, 0, 0, 0, 1, 0, 0],
    [0, 1, 0, 0, 0, 0, 0, 0],
    [0, 1, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 1],
    [0, 0, 0, 0, 0, 1, 0, 0],
    [0, 0, 0, 1, 0, 0, 0, 0],
]

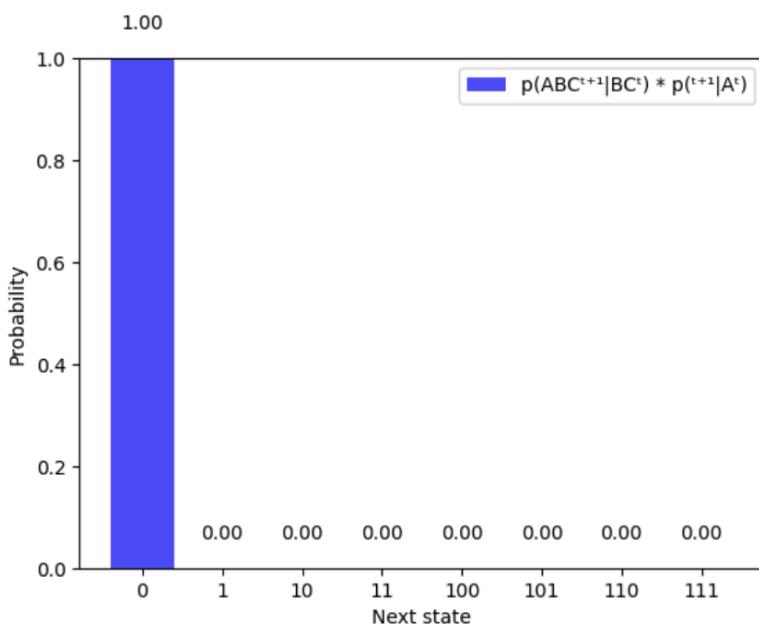
#Ejemplo 1 proyecto
ns = "ABC"
cs = "ABC"
cs_value = [1, 0, 0]

decomposition(ns, cs, cs_value)
```

Sistema original
 $ABC^{t+1} \mid ABC^t$

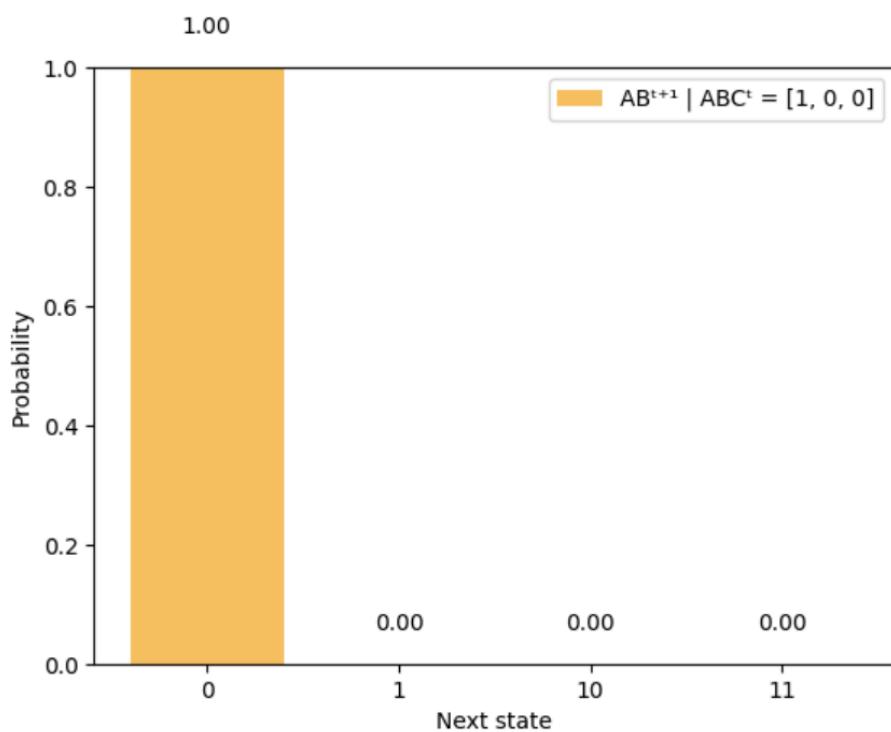


 $(ABC \mid BC) * (\mid A)$
Earth Mover's Distance: 0.0
Partición sin perdida



Sistema original

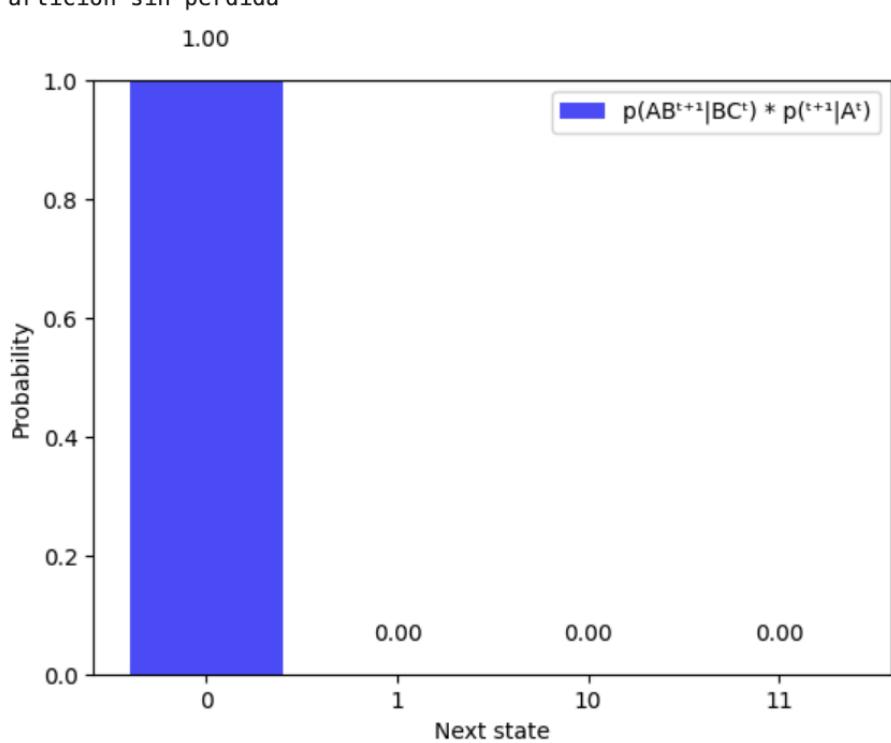
$AB^{t+1} \mid ABC^t$



(AB | BC) * (| A)

Earth Mover's Distance: 0.0

Partición sin perdida



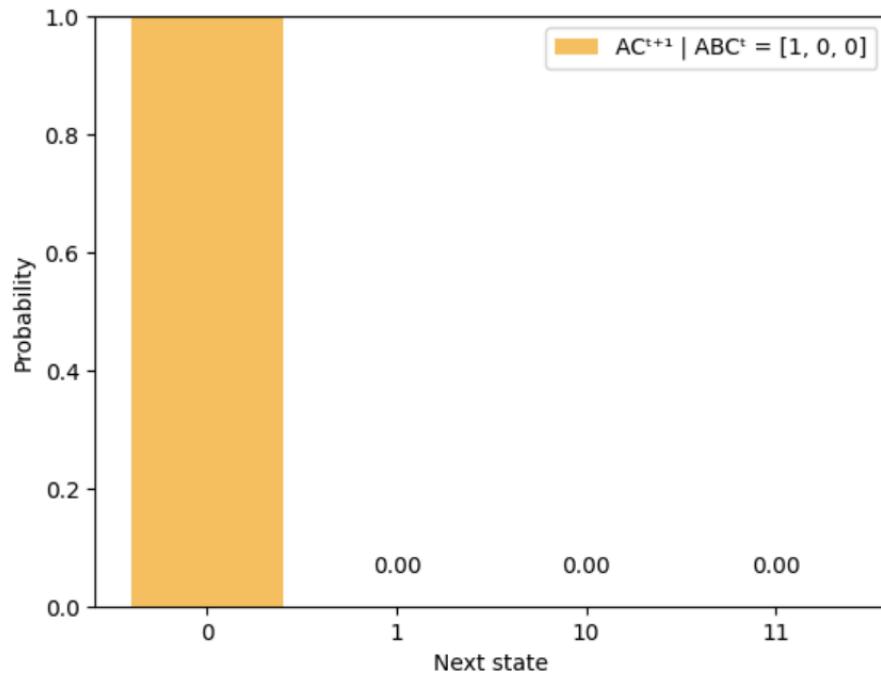
```
#Ejemplo 3 proyecto
ns = "AC"
cs = "ABC"
cs_value = [1, 0, 0]

decomposition(ns, cs, cs_value)
```

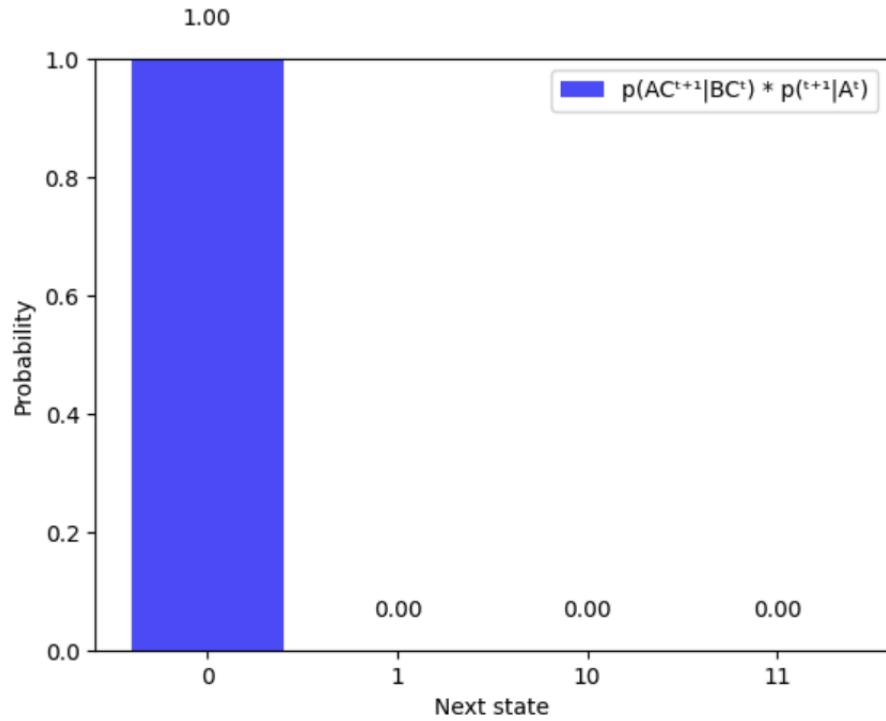
Sistema original

$AC^{t+1} \mid ABC^t$

1.00

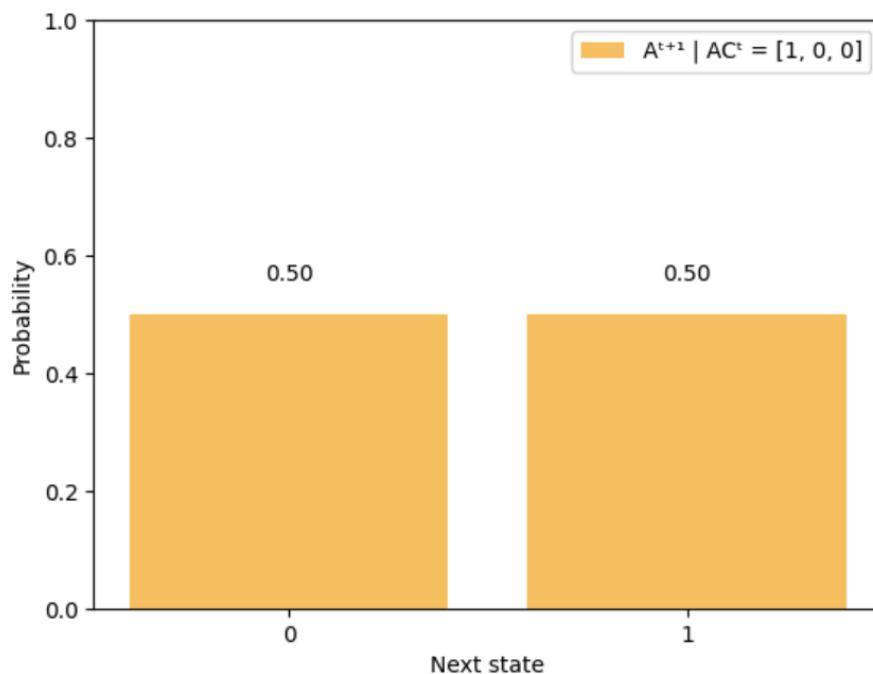


(AC | BC) * (| A)
Earth Mover's Distance: 0.0
Partición sin perdida

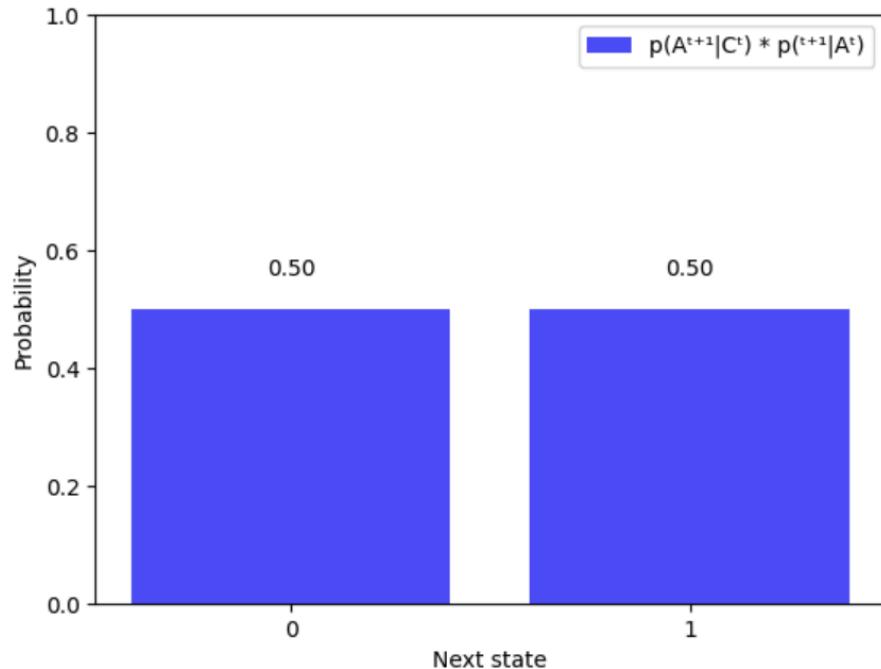


```
▶ #Ejemplo 4 proyecto  
ns = "A"  
cs = "AC"  
cs_value = [1, 0, 0]  
  
decomposition(ns, cs, cs_value)
```

```
*****  
Sistema original  
 $A^{t+1} \mid AC^t$ 
```



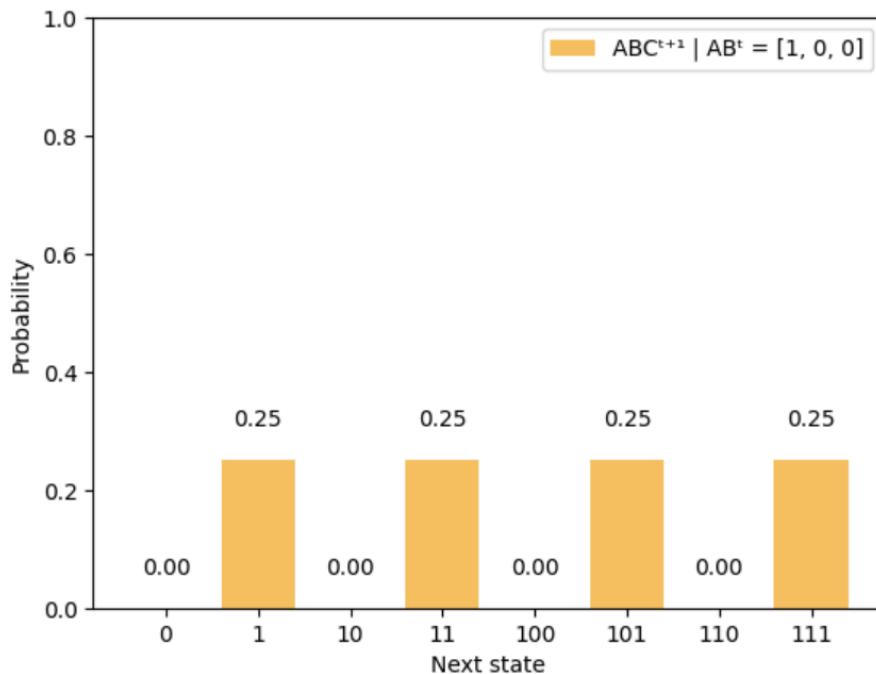
 $(A \mid C) * (\mid A)$
Earth Mover's Distance: 0.0
Partición sin perdida



```
#Ejemplo 5 proyecto
ns = "ABC"
cs = "AB"
cs_value = [1, 0, 0]

decomposition(ns, cs, cs_value)
```

Sistema original
 $ABC^{t+1} \mid AB^t$



```
*****
(ABC | B) * ( | A)
Earth Mover's Distance: 0.125
(ABC | A) * ( | B)
Earth Mover's Distance: 0.08203125
(ABC | ) * ( | AB)
Earth Mover's Distance: 0.08203125
(BC | AB) * (A | )
Earth Mover's Distance: 0.0625
(BC | B) * (A | A)
Earth Mover's Distance: 0.0625
(BC | A) * (A | B)
Earth Mover's Distance: 0.078125
(BC | ) * (A | AB)
Earth Mover's Distance: 0.078125
(AC | AB) * (B | )
Earth Mover's Distance: 0.0625
(AC | B) * (B | A)
Earth Mover's Distance: 0.0625
(AC | A) * (B | B)
Earth Mover's Distance: 0.078125
(AC | ) * (B | AB)
Earth Mover's Distance: 0.078125
(AB | AB) * (C | )
Earth Mover's Distance: 0.0625
(AB | B) * (C | A)
Earth Mover's Distance: 0.0625
(AB | A) * (C | B)
Earth Mover's Distance: 0.078125
(AB | ) * (C | AB)
Earth Mover's Distance: 0.078125
```

No se encontró partición con coste 0, se listan el valor de pérdida para cada partición posible

Pruebas para Cut:

```
states = [
    [0, 0, 0, 0, 0, 0],
    [1, 0, 0, 0, 0, 0],
    [0, 1, 0, 0, 0, 0],
    [1, 1, 0, 0, 0, 0],
    [0, 0, 1, 0, 0, 0],
    [1, 0, 1, 0, 0, 0],
    [0, 1, 1, 0, 0, 0],
    [1, 1, 1, 0, 0, 0],
    [0, 0, 0, 1, 0, 0],
    [1, 0, 0, 1, 0, 0],
    [0, 1, 0, 1, 0, 0],
    [1, 1, 0, 1, 0, 0],
    [0, 0, 1, 1, 0, 0],
    [1, 0, 1, 1, 0, 0],
    [0, 1, 1, 1, 0, 0],
    [1, 1, 1, 1, 0, 0],
```

```
[0, 0, 0, 0, 1, 0],  
[1, 0, 0, 0, 1, 0],  
[0, 1, 0, 0, 1, 0],  
[1, 1, 0, 0, 1, 0],  
[0, 0, 1, 0, 1, 0],  
[1, 0, 1, 0, 1, 0],  
[0, 1, 0, 1, 1, 0],  
[1, 1, 0, 1, 1, 0],  
[0, 0, 1, 1, 1, 0],  
[1, 0, 0, 1, 1, 0],  
[0, 1, 0, 1, 1, 0],  
[1, 1, 0, 1, 1, 0],  
[0, 0, 1, 1, 1, 0],  
[1, 0, 1, 1, 1, 0],  
[0, 1, 1, 1, 1, 0],  
[1, 1, 1, 1, 1, 0],  
[0, 0, 0, 0, 0, 1],  
[1, 0, 0, 0, 0, 1],  
[0, 1, 0, 0, 0, 1],  
[1, 1, 0, 0, 0, 1],  
[0, 0, 1, 0, 0, 1],  
[1, 1, 0, 0, 0, 1],  
[0, 1, 1, 0, 0, 1],  
[1, 1, 1, 0, 0, 1],  
[0, 0, 0, 1, 0, 1],  
[1, 0, 0, 1, 0, 1],  
[0, 1, 0, 1, 0, 1],  
[1, 1, 0, 1, 0, 1],  
[0, 0, 1, 1, 0, 1],  
[1, 0, 1, 1, 0, 1],  
[0, 1, 0, 0, 1, 1],  
[1, 1, 0, 0, 1, 1],  
[0, 0, 1, 0, 1, 1],  
[1, 0, 1, 0, 1, 1],  
[0, 1, 1, 0, 1, 1],  
[1, 1, 1, 0, 1, 1],  
[0, 0, 0, 1, 1, 1],  
[1, 0, 0, 1, 1, 1],  
[0, 1, 0, 1, 1, 1],  
[1, 1, 0, 1, 1, 1]
```

```
[1, 1, 0, 1, 1, 1],  
[0, 0, 1, 1, 1, 1],  
[1, 0, 1, 1, 1, 1],  
[0, 1, 1, 1, 1, 1],  
[1, 1, 1, 1, 1, 1],  
]  
]
```

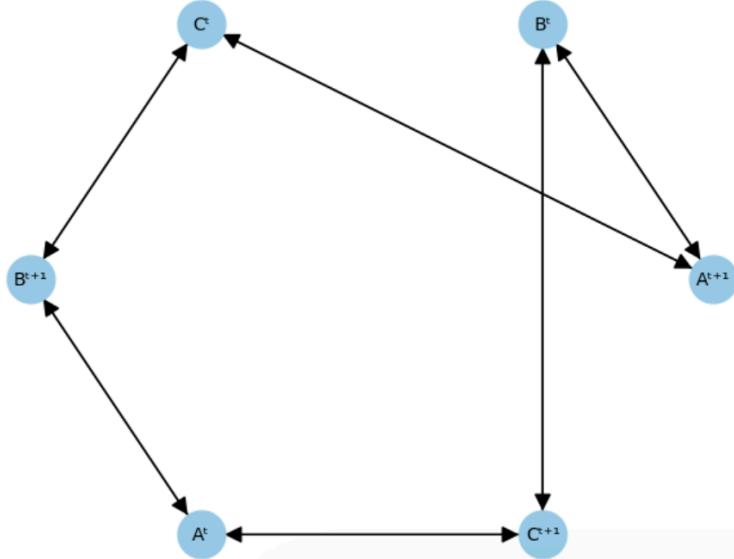
```
probabilities = [
    [0, 0, 0, 0, 0, 0, 0],
    [0, 0, 1, 0, 0, 0, 0],
    [1, 0, 1, 0, 1, 0, 0],
    [1, 0, 0, 0, 1, 0, 0],
    [1, 0, 0, 0, 0, 0, 0],
    [1, 1, 1, 0, 0, 0, 0],
    [1, 0, 1, 0, 1, 0, 0],
    [1, 1, 0, 0, 1, 0, 0],
    [1, 0, 0, 0, 0, 0, 0],
    [1, 0, 1, 0, 0, 0, 0],
    [1, 0, 1, 0, 1, 0, 0],
    [1, 0, 0, 0, 1, 0, 0],
    [1, 0, 0, 0, 0, 0, 0],
    [1, 1, 1, 0, 0, 0, 0],
    [1, 0, 1, 0, 1, 0, 0],
    [1, 1, 0, 0, 1, 0, 0],
    [1, 0, 0, 0, 0, 0, 0],
    [1, 0, 0, 0, 0, 0, 0],
    [1, 0, 1, 0, 0, 0, 0],
    [1, 0, 1, 0, 1, 0, 0],
    [1, 0, 0, 0, 1, 0, 0],
    [1, 0, 1, 0, 0, 1, 0],
    [1, 0, 0, 0, 0, 1, 0],
    [1, 0, 0, 0, 0, 0, 1],
    [0, 0, 1, 0, 0, 0, 0],
    [1, 0, 1, 0, 1, 0, 0],
    [1, 0, 0, 0, 0, 1, 0],
    [1, 0, 0, 0, 0, 0, 1],
    [1, 1, 1, 0, 0, 0, 0],
    [1, 0, 1, 0, 0, 1, 0],
    [1, 1, 0, 0, 0, 1, 0],
    [1, 0, 0, 0, 0, 0, 1],
    [1, 0, 1, 0, 0, 0, 1],
    [1, 0, 1, 0, 1, 0, 0],
    [1, 0, 0, 0, 0, 0, 1],
    [1, 0, 0, 0, 0, 0, 0],
    [1, 0, 0, 0, 0, 1, 0],
    [1, 0, 0, 0, 0, 0, 0],
    [1, 1, 1, 0, 0, 0, 0],
    [1, 0, 1, 0, 0, 1, 0],
    [1, 1, 0, 0, 0, 1, 0],
    [1, 0, 0, 0, 0, 0, 1],
    [1, 0, 1, 0, 0, 0, 0],
    [1, 0, 1, 0, 1, 0, 0],
    [1, 0, 0, 0, 0, 1, 0],
    [1, 0, 0, 0, 0, 0, 1],
    [1, 1, 1, 0, 0, 0, 0],
    [1, 1, 0, 0, 0, 1, 0],
    [0, 0, 0, 0, 0, 0, 0],
    [0, 0, 1, 0, 0, 0, 0]]
```

```
[1, 0, 1, 0, 1, 0],  
[1, 0, 0, 0, 1, 0],  
[1, 0, 0, 0, 0, 0],  
[1, 1, 1, 0, 0, 0],  
[1, 0, 1, 0, 1, 0],  
[1, 1, 0, 0, 1, 0],  
[1, 0, 0, 0, 0, 0],  
[1, 0, 1, 0, 0, 0],  
[1, 0, 1, 0, 1, 0],  
[1, 0, 0, 0, 1, 0],  
[1, 0, 0, 0, 0, 0],  
[1, 1, 1, 0, 0, 0],  
[1, 0, 1, 0, 1, 0],  
[1, 1, 0, 0, 1, 0],  
[0, 0, 0, 0, 0, 0],  
[0, 0, 1, 0, 0, 0],  
[1, 0, 1, 0, 1, 0],  
[1, 0, 0, 0, 1, 0],  
[1, 0, 0, 0, 0, 0],  
[1, 1, 1, 0, 0, 0],  
[1, 0, 1, 0, 1, 0],  
[1, 1, 0, 0, 1, 0],  
[1, 0, 0, 0, 0, 0],  
[1, 0, 1, 0, 0, 0],  
[1, 0, 0, 0, 1, 0],  
[1, 1, 1, 0, 0, 0],  
[1, 0, 1, 0, 1, 0],  
[1, 1, 0, 0, 1, 0],  
cs_value = [1, 0, 0, 0, 1, 0]
```

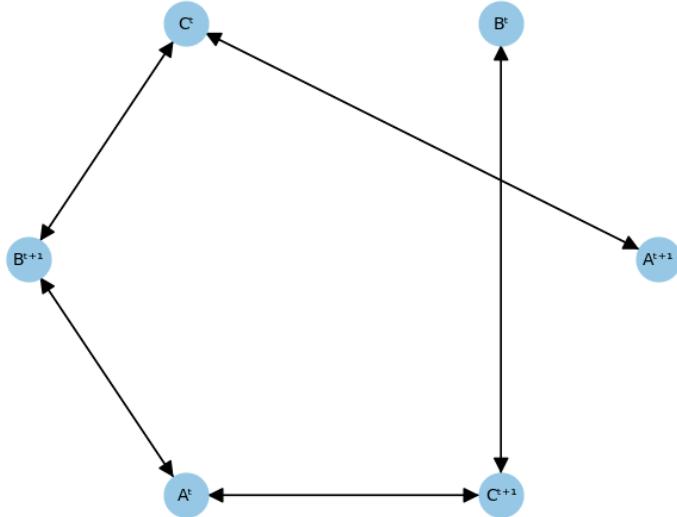
```
[13] #A
ns = "ABC"
cs = "ABC"

cut_process(ns, cs, cs_value, probabilities, states)
```

Grafo principal

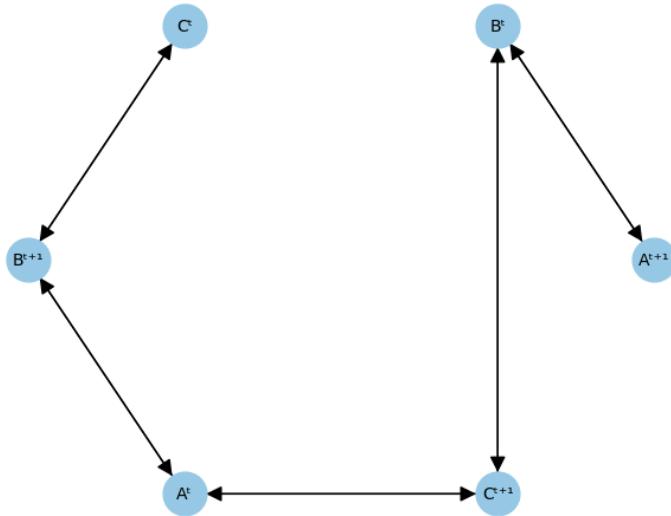


Variable actual A^{t+1}
cortando B^t de A^{t+1}
partition: $(\emptyset^{t+1} | B^{t+1})$ y $(ABC^{t+1} | AC^t)$



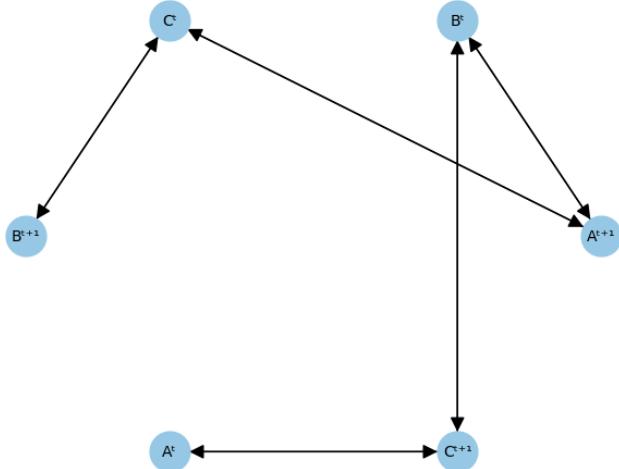
Earth Mover's Distance: 0.21875
No bipartición generada
conexión restaurada con costo: 0.21875

Variable actual A^{t+1}
cortando C^t de A^{t+1}
partition: $(\emptyset^{t+1} | C^{tt})$ y $(ABC^{t+1} | AB^t)$



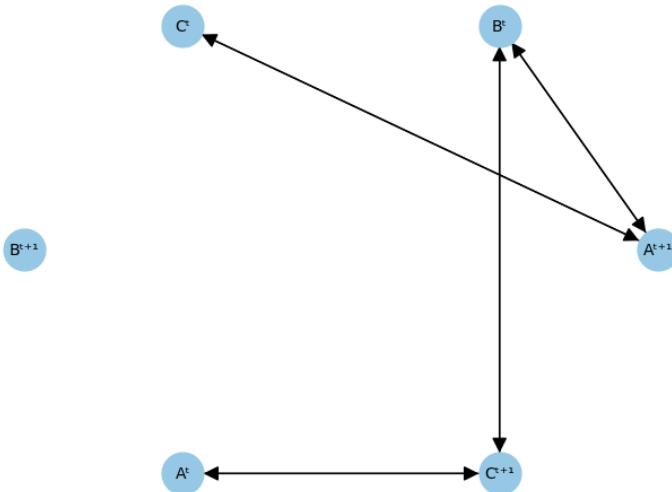
Earth Mover's Distance: 0.21875
No bipartición generada
conexión restaurada con costo: 0.21875

Variable actual B^{t+1}
cortando A^t de B^{t+1}
partition: $(\emptyset^{t+1} | A^{tt})$ y $(ABC^{t+1} | BC^t)$



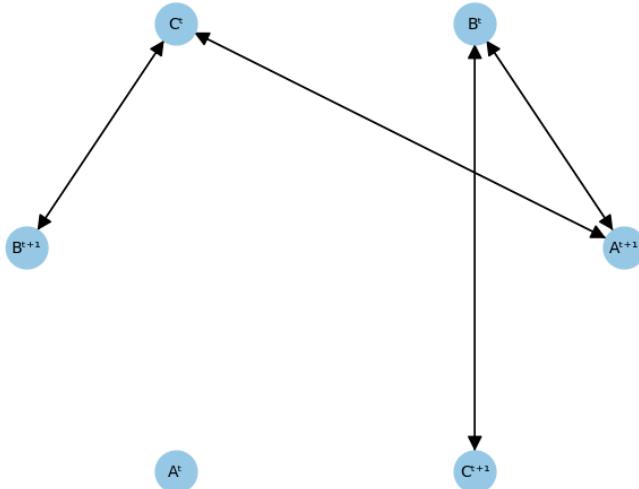
Earth Mover's Distance: 0.0
No bipartición generada
conexión elimina sin perdida de información

 Variable actual B^{t+1}
 cortando C^t de B^{t+1}
 partition: $(\emptyset^{t+1} | C^{t+1})$ y $(ABC^{t+1} | AB^t)$

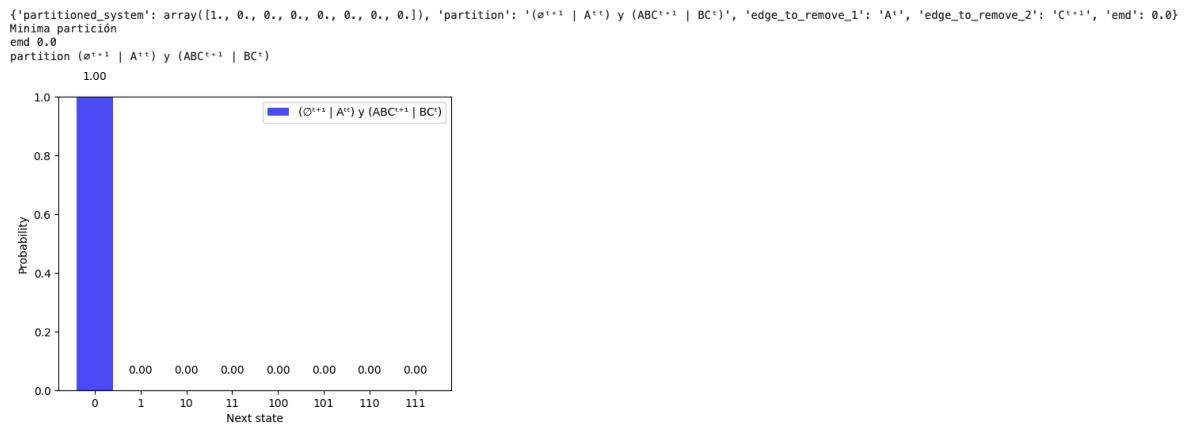


Earth Mover's Distance: 0.21875
 Bipartición generada
 minima partición actualizada
 bipartición restaurada con costo: 0.21875

 Variable actual C^{t+1}
 cortando A^t de C^{t+1}
 partition: $(\emptyset^{t+1} | A^{t+1})$ y $(ABC^{t+1} | BC^t)$



Earth Mover's Distance: 0.0
 Bipartición generada
 minima partición alcanzada



```

❶ #B
ns = "B"
cs = "A"

cut_process(ns, cs, cs_value, probabilities, states)

☒ Grafo principal

```

The code defines a variable `ns` as "B" and `cs` as "A". It then calls the `cut_process` function with parameters `ns`, `cs`, `cs_value`, `probabilities`, and `states`. Below the code, there is a section titled "Grafo principal" which contains a diagram of a directed graph.

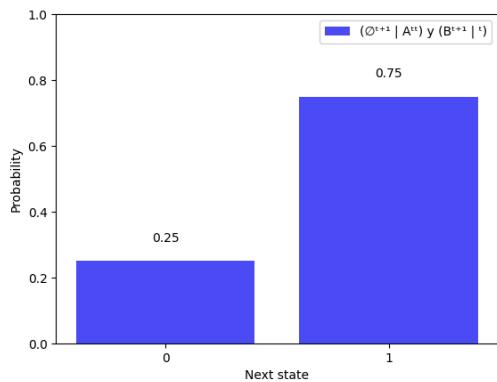


Variable actual B^{t+1}
cortando A^t de B^{t+1}
partition: $(\emptyset^{t+1} | A^{tt}) y (B^{t+1} | t)$



Earth Mover's Distance: 0.0
Bipartición generada
minima partición alcanzada

```
{
'partitioned_system': array([0.25, 0.75]), 'partition': '(o^{t+1} | A^t) y (B^{t+1} | ^t)', 'edge_to_remove_1': 'A^t', 'edge_to_remove_2': 'B^{t+1}', 'emd': 0.0}
Minima partición
end 0.0
partition (o^{t+1} | A^t) y (B^{t+1} | ^t)
```



```
#C
ns = "A"
cs = "B"

cut_process(ns, cs, cs_value, probabilities, states)
```

Grafo principal



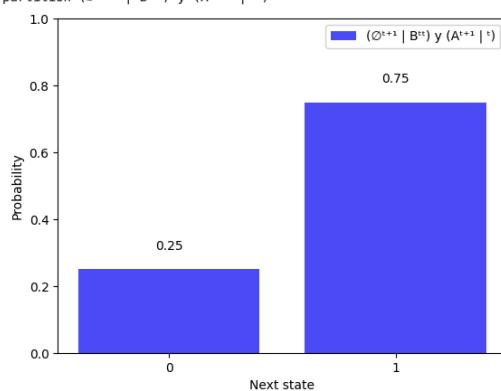
Variable actual A^{t+1}
cortando B^t de A^{t+1}
partition: $(\emptyset^{t+1} | B^{tt})$ y $(A^{t+1} | t)$

A^{t+1}

B^t

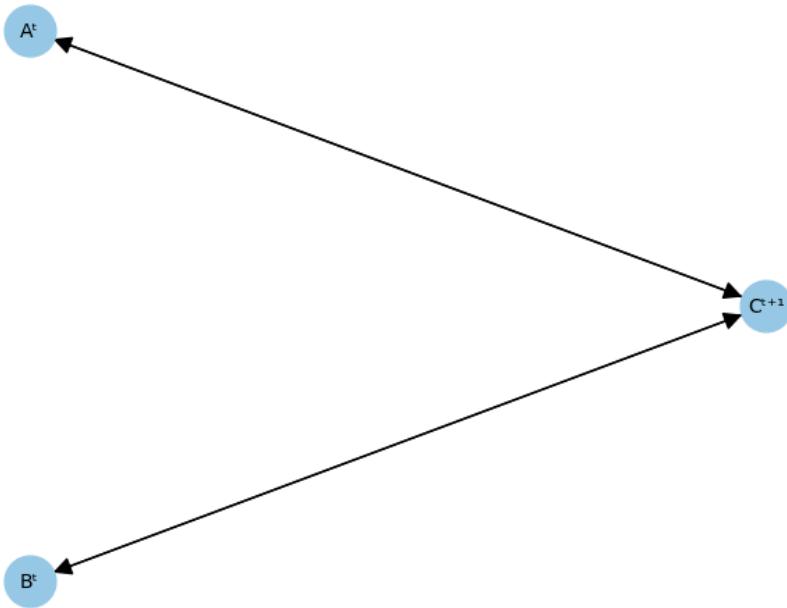
Earth Mover's Distance: 0.25
Bipartición generada
minima partición actualizada
bipartición restaurada con costo: 0.25

{'partitioned_system': array([0.25, 0.75]), 'partition': '(\emptyset^{t+1} | B^{tt}) y (A^{t+1} | t)', 'edge_to_remove_1': 'B^t', 'edge_to_remove_2': 'A^{t+1}', 'emd': 0.25}
Minima partición
emd 0.25
partition $(\emptyset^{t+1} | B^{tt})$ y $(A^{t+1} | t)$



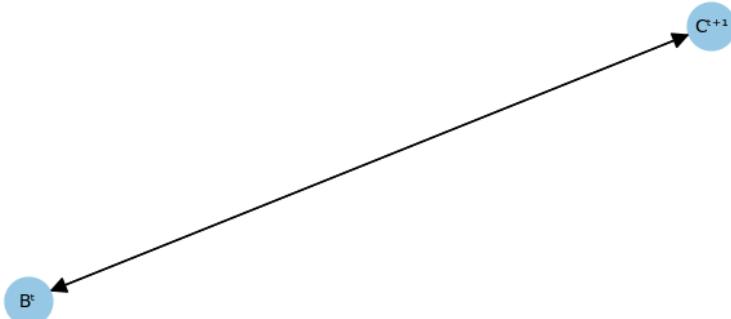
```
#D  
ns = "C"  
cs = "ABC"  
  
cut_process(ns, cs, cs_value, probabilities, states)
```

Grafo principal



Variable actual C^{t+1}
cortando A^t de C^{t+1}
partition: $(\emptyset^{t+1} \mid A^{t+1})$ y $(C^{t+1} \mid BC^t)$

A^t

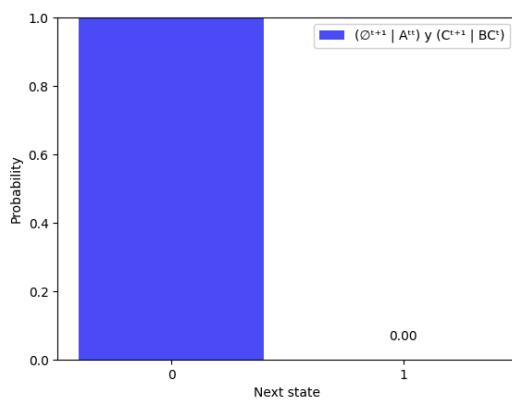


Earth Mover's Distance: 0.0
Bipartición generada
minima partición alcanzada

```

{'partitioned_system': array([1., 0.]), 'partition': '(ø^{t+1} | A^t) y (C^{t+1} | BC^t)', 'edge_to_remove_1': 'A^t', 'edge_to_remove_2': 'C^{t+1}', 'emd': 0.0}
Minima partición
end 0.0
partition (ø^{t+1} | A^t) y (C^{t+1} | BC^t)

```



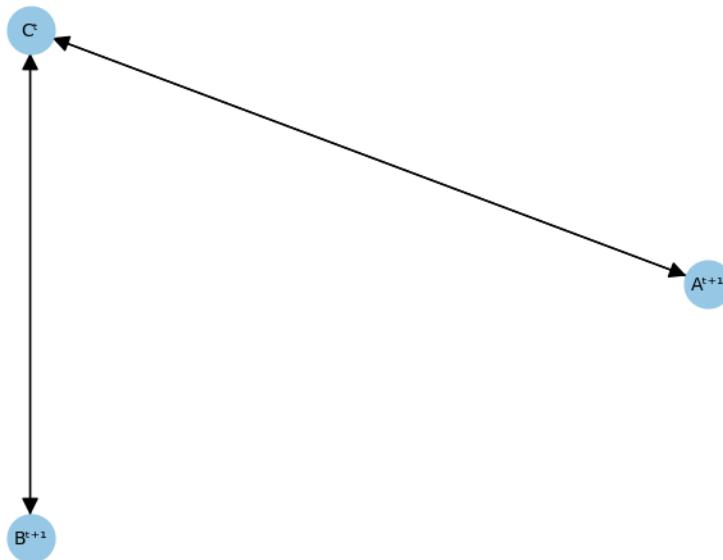
```

▶ #E
ns = "AB"
cs = "C"

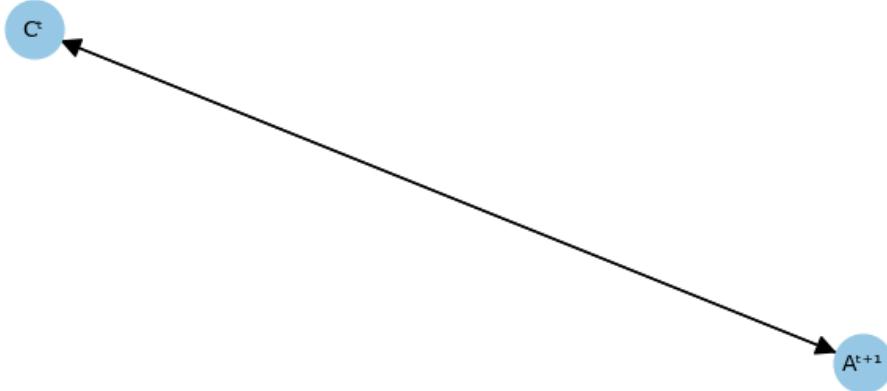
cut_process(ns, cs, cs_value, probabilities, states)

```

☒ Grafo principal

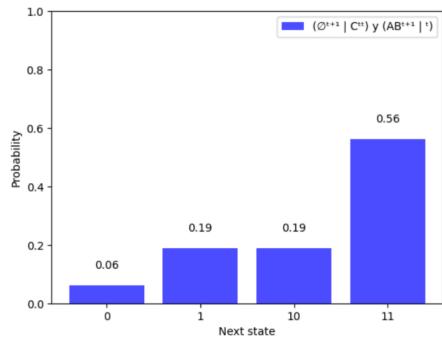


Grafo final



B^{t+1}

```
{
'partitioned_system': array([0.0625, 0.1875, 0.1875, 0.5625]), 'partition': '(ø^{t+1} | C^{t+1}) y (AB^{t+1} | 1)', 'edge_to_remove_1': 'C^{t+1}', 'edge_to_remove_2': 'B^{t+1}', 'emd': 0.15625
Minima partición
emd 0.15625
partition (ø^{t+1} | C^{t+1}) y (AB^{t+1} | 1)
}
```



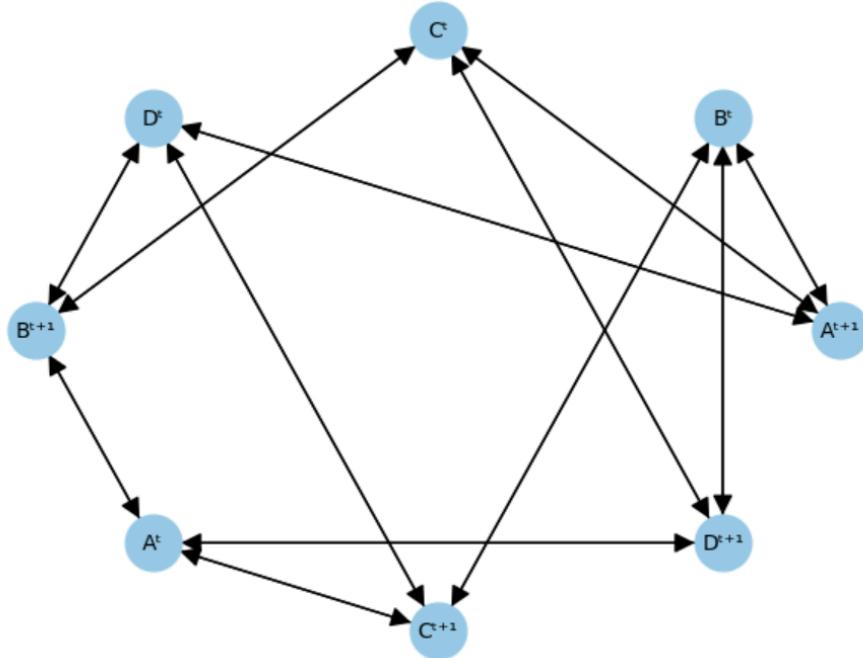
```

#F
ns = "ABCD"
cs = "ABCD"

cut_process(ns, cs, cs_value, probabilities, states)

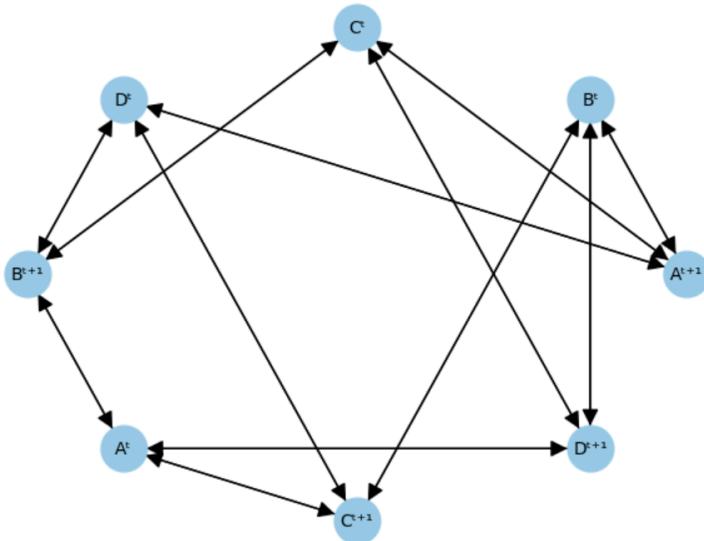
```

Grafo principal



----- ***** -----

Grafo final



```
{'partitioned_system': [], 'partition': '', 'edge_to_remove_1': '', 'edge_to_remove_2': '', 'emd': 0}
```

No se lograron cortes

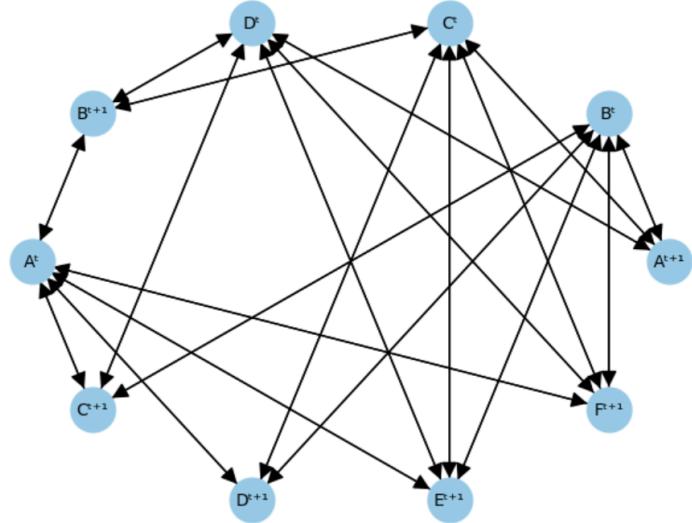
```

#G
ns = "ABCDEF"
cs = "ABCD"

cut_process(ns, cs, cs_value, probabilities, states)

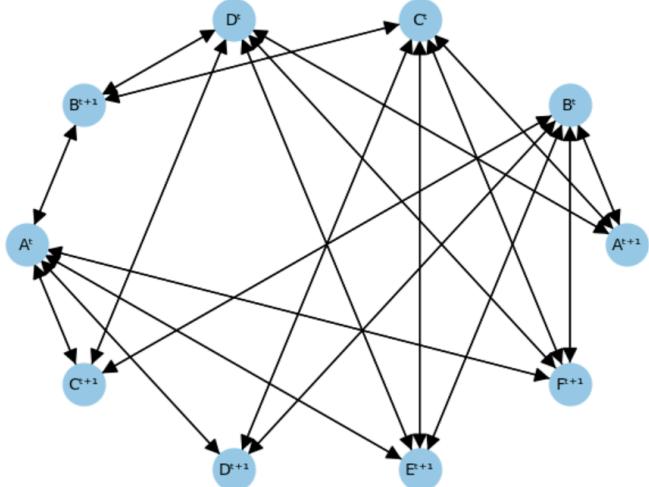
Grafo principal

```



```

*****
Grafo final
*****
```



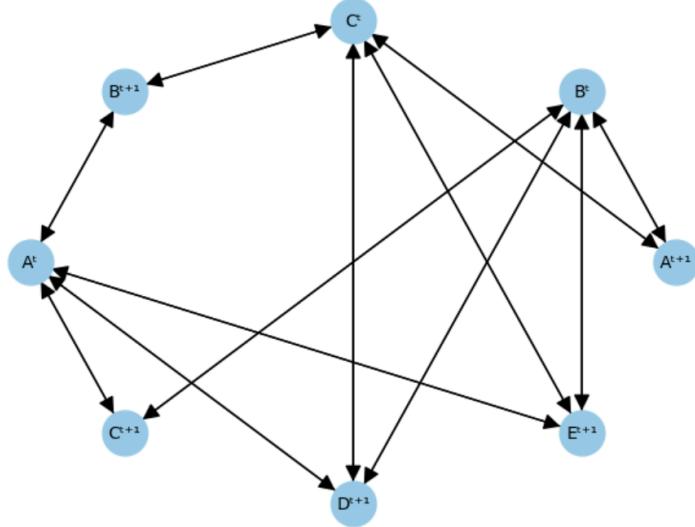
```
{'partitioned_system': [], 'partition': '', 'edge_to_remove_1': '', 'edge_to_remove_2': '', 'emd': 0}
```

No se lograron cortes

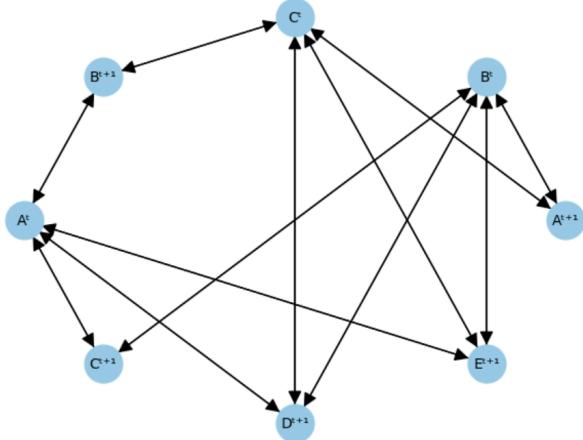
```
#H
ns = "ABCDE"
cs = "ABC"

cut_process(ns, cs, cs_value, probabilities, states)
```

Grafo principal



Grafo final

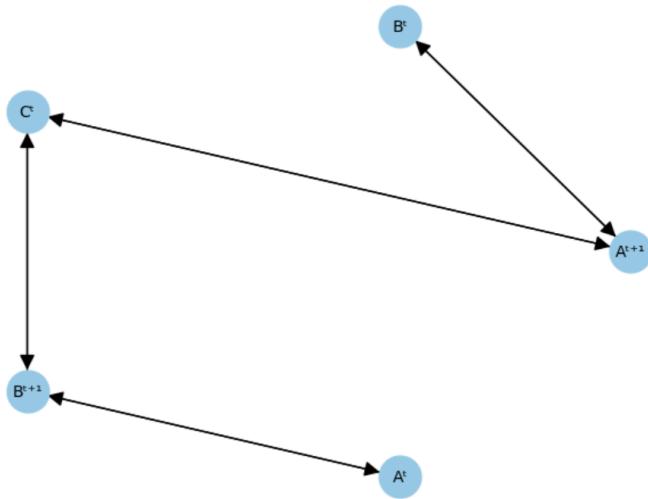


```
{'partitioned_system': [], 'partition': '', 'edge_to_remove_1': '', 'edge_to_remove_2': '', 'emd': 0}
```

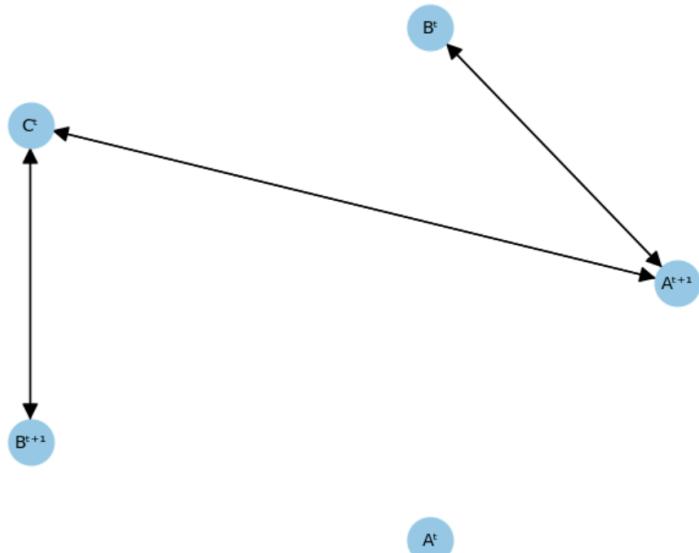
No se lograron cortes

```
#I  
ns = "AB"  
cs = "ABC"  
  
cut_process(ns, cs, cs_value, probabilities, states)
```

Grafo principal



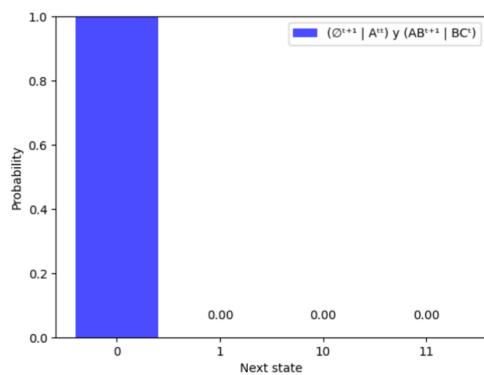
Grafo final



```

{'partitioned_system': array([1., 0., 0., 0.]), 'partition': '(\emptyset^{t+1} | A^t) y (AB^{t+1} | BC^t)', 'edge_to_remove_1': 'A^t', 'edge_to_remove_2': 'B^{t+1}', 'emd': 0.3125}
Minima partición
emd 0.3125
partition (\emptyset^{t+1} | A^t) y (AB^{t+1} | BC^t)
1.00

```



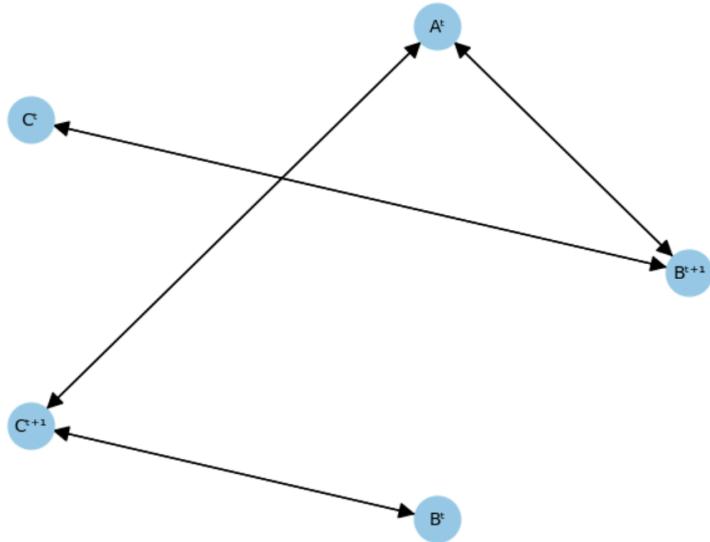
```

#J
ns = "BC"
cs = "ABC"

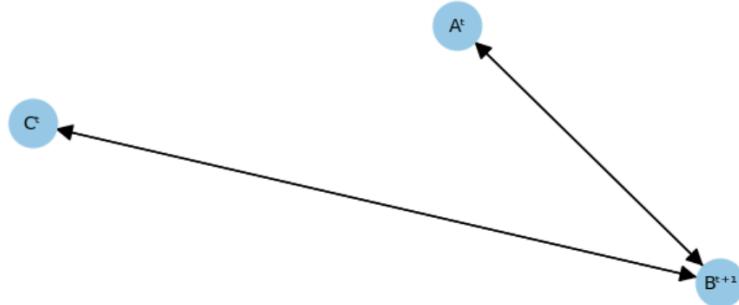
cut_process(ns, cs, cs_value, probabilities, states)

```

Grafo principal



Grafo final



```
{'partitioned_system': array([1., 0., 0., 0.]), 'partition': '(\emptyset^{t+1} | A^t) y (BC^{t+1} | BC^t)', 'edge_to_remove_1': 'A^t', 'edge_to_remove_2': 'C^{t+1}', 'emd': 0.3125}
Minima particion
end 0.3125
```

```
partition (\emptyset^{t+1} | A^t) y (BC^{t+1} | BC^t)
```

```
1.00
```

