

# UNIVERSIDAD DE CALDAS

## ANÁLISIS Y DISEÑO DE ALGORITMOS

### PARTE II

---

## PROYECTO SEGUNDA ENTREGA

---

*Autor:*

Lucas Bohórquez Naranjo  
Marco Fernando Contreras Rojas

*Código de estudiante:*

1701716343  
1701711791

**Facultad de Ingenierías.**

12 de diciembre de 2021

# Índice

<b>1. Introducción</b>	<b>3</b>
<b>2. Introducción a la libería</b>	<b>3</b>
<b>3. Descripción del Problema</b>	<b>4</b>
3.1. Puntos a tratar . . . . .	6
<b>4. Propuesta de mejoramiento</b>	<b>7</b>
4.1. Funcionamiento Método all_partitions() . . . . .	9
4.1.1. Algoritmo . . . . .	9
4.2. Estrategia . . . . .	10
4.3. Aplicación . . . . .	11
4.4. Análisis de complejidad . . . . .	11
4.5. Explicación de la solución de mejoramiento . . . . .	11
4.5.1. Algoritmo . . . . .	12
4.5.2. Resultados . . . . .	13

## 1. Introducción

La teoría de la información integrada proporciona un marco matemático para caracterizar completamente la estructura causa-efecto de un sistema físico. Aquí, nos PyPhi, un paquete de software de Python que implementa este marco para el análisis causal y despliega la estructura completa de causa-efecto de los sistemas dinámicos discretos de elementos binarios. El software permite a los usuarios estudiar fácilmente estas estructuras, sirve como una implementación de referencia actualizada de los formalismos de la teoría de la información integrada y se ha aplicado en la investigación sobre complejidad, emergencia y ciertas cuestiones biológicas. Primero proporcionamos una visión general del algoritmo principal y demostramos la funcionalidad de PyPhi en el curso del análisis de un sistema de ejemplo, y luego describimos los detalles del diseño e implementación del algoritmo. PyPhi se puede instalar con el administrador de paquetes de Python a través del comando 'pip install pyphi' en sistemas Linux y macOS equipados con Python 3.4 o superior. PyPhi es de código abierto y está licenciado bajo la GPLv3.

Para el presente proyecto se planteó por parte de la docente, la necesidad de realizar una optimización del código existente, toda vez que al analizar los algoritmos que procesan las entradas de datos se logró establecer que son computacionalmente altamente costosos; motivo por el cual se busca disminuir los tiempos de procesamiento y reducir la complejidad de los cálculos realizados por la librería dentro de los diferentes sistemas operativos y arquitecturas.

## 2. Introducción a la librería

En este punto procederemos a dar una breve introducción a la librería PyPhi; extracto del paper publicado por los desarrolladores de la librería.

La teoría de la información integrada (IIT sus siglas en inglés) se ha propuesto como una teoría de la conciencia. La hipótesis central es que un sistema físico tiene que cumplir cinco requisitos ('postulados') para ser un sustrato físico de la experiencia subjetiva: (1) existencia intrínseca (el sistema debe ser capaz de hacer una diferencia para sí mismo); (2) composición (debe estar compuesta de partes que tengan poder causal dentro del todo); (3) información (su poder causal debe ser específico); (4) integración (su poder causal no debe ser reducible al de sus partes); y (5) exclusión (debe ser máximamente irreducible).

A partir de estos postulados, el IIT desarrolla un marco matemático para evaluar la estructura causa-efecto (CES) de un sistema físico que es aplicable a sistemas dinámicos discretos. Este marco ha demostrado ser útil no solo para el estudio de la conciencia, sino que también se ha aplicado en la investigación sobre la complejidad, la emergencia, y ciertas cuestiones biológicas.

La medida principal del poder causa-efecto, la información integrada (denotada  $\phi$ ), cuantifica cuán irreducible es el CES de un sistema a los de sus partes.  $\phi$  también sirve como una medida general de complejidad que capta hasta qué punto un sistema está integrado y diferenciado (informativo).

Aquí describimos PyPhi, un paquete de software Python que implementa el marco de IIT para el análisis causal y despliega el CES completo de sistemas dinámicos Markovianos discretos de elementos binarios. El software permite a los usuarios estudiar fácilmente estos CES y sirve como una implementación de referencia actualizada de los formalismos de IIT.

### 3. Descripción del Problema

Debido a que este toolbox calcula una medida de la información, para ello se desarrollan ciertas especificaciones que requiere muchísimas operaciones. Como parte de los objetivos del curso está el comprender el comportamiento de los algoritmos y aplicar las técnicas más adecuadas con el fin de lograr mejores rendimientos abordaremos el problema que se presenta a continuación. En esta parte final del Proyecto-Taller nos centraremos en una de las subrutinas cuya función de eficiencia es de orden exponencial. Específicamente se revisará la subrutina que busca la MIP, la cual debe hacer todas las particiones posibles con el fin de encontrar aquella que produce la menor pérdida de información. En términos de operaciones gráficas se prueba un par mecanismo-purview particionándolo en partes y cortando las conexiones entre ellos, después de cortar esas conexiones se obtiene un repertorio particionado.

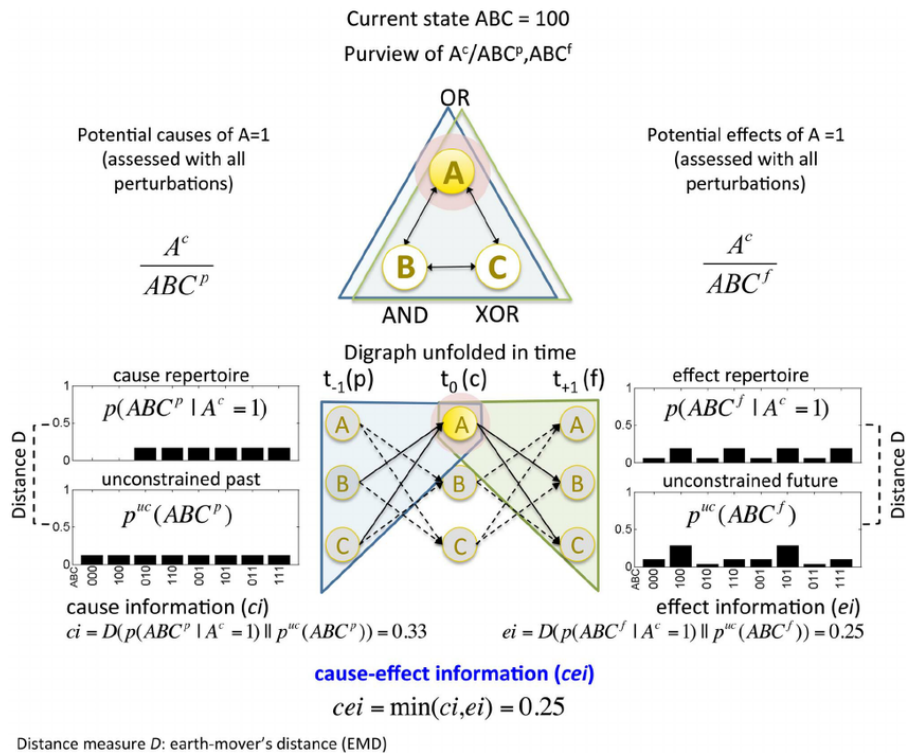


Figura 1

En este ejemplo solo mirando el repertorio efecto tendríamos que después de hacer todas las

particiones posibles para el mecanismo- purview que estamos revisando, la partición que aparece a continuación es la que genera la menor pérdida de información (ojo que es solo mirando el repertorio efecto).

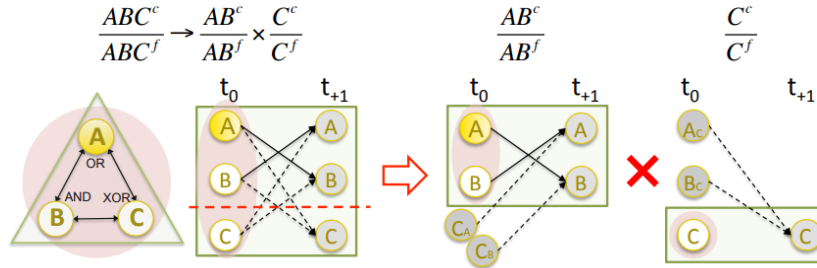


Figura 2

Como se aprecia este es un problema bastante costoso porque se deben generar todas las particiones posibles (en este ejemplo será para el efecto o futuro, pero lo mismo se debe hacer para la causa o pasado) y cada partición se debe medir (aquí es donde se usan las medidas de distancia que ustedes conocen) y la que produzca la mínima pérdida se llama MIP.

El procedimiento de búsqueda de la MIP se implementa mediante los métodos `Subsystem.cause_mip()` y `Subsystem.effect_mip()`. Cada uno devuelve un objeto `RepertoireIrreducibilityAnalysis` que contiene la MIP, así como el valor phi, mecanismo, purview, dirección temporal (causa o efecto), repertorio no particionado y repertorio particionado. Para el ejemplo que les puse en este documento calculamos la MIP del efecto del mecanismo ABC sobre el purview ABC así:

```

>>> mechanism = (A, B, C)
>>> purview = (A, B, C)
>>> mip = subsystem.effect_mip(mechanism, purview)
>>> print(mip)
Repertoire irreducibility analysis
 $\phi = 1/4$ 
Mechanism: [A, B, C]
Purview = [A, B, C]
Direction: EFFECT
Partition:
   $\emptyset$     A,B,C
  --- x ---
    B    A,C
Repertoire:

```

S	Pr(S)
000	0
100	0
010	0
110	0
001	1
101	0
011	0
111	0

Partitioned repertoire:

S	Pr(S)
000	0
100	0
010	0
110	0
001	3/4
101	0
011	1/4
111	0

Figura 3

### 3.1. Puntos a tratar

1. La idea es habilitar esas opciones (que más influyen para hallar MIP) y trabajar con las diferentes redes usadas en las pruebas y mirar cómo se comportan los tiempos. En el documento se indica cuales son las subrutinas a habilitar. (Tabla de pruebas)  
Por ejemplo, si desean probar la aproximación cut-one, deben revisar como se comporta de acuerdo con la arquitectura de la maquina (procesadores, núcleos) aquí tendrían que entrar a revisar un poco aspectos de hardware, para entender que pasa y como se usarían los procesadores y/o los núcleos (si dispusiéramos de varios).
2. Será muy interesante ver el comportamiento con diferentes ejemplos de topología de redes, como los grafos que ustedes han utilizado etc, porque luego se podrá tratar de concluir

sobre el tipo de red que podría generar mayor información integrada, como es la eficiencia respecto al tiempo empleado etc.

3. Con todas estas pruebas se va conociendo un poco más lo que pasa, para luego hacer propuestas de mejoramiento sobre la opción que ustedes determinen. Aquí pueden resultar muy importante las técnicas como divide y vencerás, programación dinámica, backtracking etc solo por citar algunas.

Tabla de pruebas


Entrada de datos	Clase de mejora	Descripción estrategia PyPhi	Resultado Original	Resultado con MEJORA	Observaciones
Grafo:  Candidate system ABC TPM: [ [0, 0, 0], [0, 0, 1], [1, 0, 1], [1, 0, 0], [1, 1, 0], [1, 1, 1], [1, 1, 1], [1, 1, 0] ] CM: [ [0, 0, 1], [1, 0, 1], [1, 1, 0] ]	A	Cut one approximation	MIP: _____ Phi: _____ Tiempo: _____ Función de eficiencia: _____ Ver anexo _____	MIP: _____ Phi: _____ Tiempo: _____ Función de eficiencia: _____ Ver anexo _____	
	A	"no new concepts"	MIP: _____ Phi: _____ Tiempo: _____	MIP: _____ Phi: _____ Tiempo: _____	
	O	Memoization and caching	MIP: _____ Phi: _____ Tiempo: _____ Función de eficiencia: _____ Ver anexo _____	MIP: _____ Phi: _____ Tiempo: _____ Función de eficiencia: _____ Ver anexo _____	

Figura 4

## 4. Propuesta de mejoramiento

A continuación haremos una descripción de los elementos considerados en el presente proyecto, para realizar la optimización de la librería. Para ello consideramos pertinente analizar las diferentes clases con que cuenta el toolbox, y en particular las clases que se utilizan para realizar las particiones:

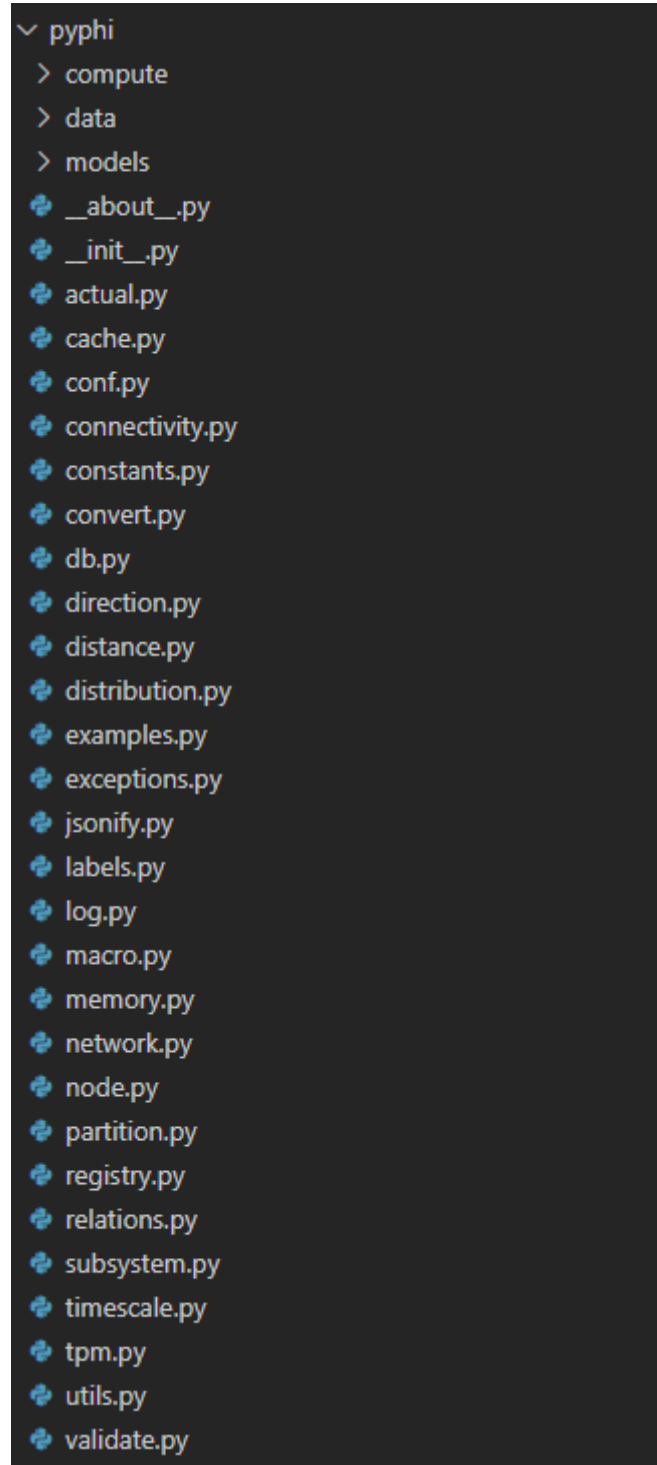


Figura 5: Estructura Librería

Después de un análisis de las diferentes clases, consideramos que la optimización debería ser realizada en la clase que realiza las particiones, es decir, `-partition.py`-, ya que cuenta con el siguiente método:



```

@partition_types.register("ALL")
def all_partitions(mechanism, purview, node_labels=None):
    """Return all possible partitions of a mechanism and purview.

    Partitions can consist of any number of parts.

    Args:
        mechanism (tuple[int]): A mechanism.
        purview (tuple[int]): A purview.

    Yields:
        KPartition: A partition of this mechanism and purview into ``k`` parts.
    """
    for mechanism_partition in partitions(mechanism):
        mechanism_partition.append([])
        n_mechanism_parts = len(mechanism_partition)
        max_purview_partition = min(len(purview), n_mechanism_parts)
        for n_purview_parts in range(1, max_purview_partition + 1):
            n_empty = n_mechanism_parts - n_purview_parts
            for purview_partition in k_partitions(purview, n_purview_parts):
                purview_partition = [tuple(_list) for _list in purview_partition]
                # Extend with empty tuples so purview partition has same size
                # as mechanism purview
                purview_partition.extend([()] * n_empty)

                # Unique permutations to avoid duplicates empties
                for purview_permutation in set(permutations(purview_partition)):
                    parts = [
                        Part(tuple(m), tuple(p))
                        for m, p in zip(mechanism_partition, purview_permutation)
                    ]

                    # Must partition the mechanism, unless the purview is fully
                    # cut away from the mechanism.
                    if parts[0].mechanism == mechanism and parts[0].purview:
                        continue

                    yield KPartition(*parts, node_labels=node_labels)

```

Figura 6: Clase partition.py - Método all\_partitions()

## 4.1. Funcionamiento Método all\_partitions()

El método precedente recibe como parámetros un mecanismo y un purview, así como las etiquetas de los nodo, que pueden ser opcionales, y retorna todas las posibles particiones del mecanismo y del purview.

### 4.1.1. Algoritmo

El algoritmo consiste en:

1. Se recibe una tupla de enteros correspondiente a el mecanismo.
2. Se recibe una tupla de enteros correspondiente al purview.
3. Se realiza un llamado a la función partitions con el mecanismo recibido; método que retorna una colección de particiones, cabe aclarar que nos encontramos en un ciclo (primer ciclo).

4. Por cada colección de particiones se agrega una lista vacía.
5. Se establece una variable llamada `-n_mechanism_parts-` con el valor de la longitud de la colección anterior.
6. Se establece una variable denominada `-max_purview_partition-` con el valor menor comparado entre la longitud la tupla `purview` y el valor de la variable llamada `-n_mechanism_parts-`
7. En este punto pasamos al siguiente ciclo (anidado), en el cual se itera desde la posición número uno hasta el valor de la variable `-n_mechanism_parts-` más uno.
8. Por cada iteración se instancia la variable `n_empty` con el valor resultante de restar `-n_mechanism_parts-` con `n_purview_parts`.
9. Se procede a realizar por cada iteración un ciclo (nuevamente anidado) en el cual se hace un llamado a la función `k_partitions` con la tupla `purview` y el valor de la variable `n_purview_parts`.
10. Acto seguido se instancia una variable con el resultado (tupla) generada dentro de un nuevo ciclo con la variable `purview_partition`.
11. Se hace un llamado al método `extend` del lenguaje python, el cual agrega varios elementos al final de la lista con el mismo tamaño de la mecanismo `purview`.
12. Nuevamente se realiza otro ciclo (también anidado) en el cual se realiza una permutación de la partición del `purview`.
13. Se genera un variable con la clase `-Part-`; haciendose un llamado al método `zip`; el cual mapea los mismos índices en más de un iterable.
14. Se realizan entonces las particiones sobre el mecanismo hasta que el `purview` se corte del mecanismo.
15. Finalmente se hace un llamado a la función de python `-Yield-` la cuál retorna una lista en memoria, sin salir del ciclo en el cual se encuentra, con las partes realizadas en el paso anterior.

## 4.2. Estrategia

La idea de diseño del método *divide y vencerás* es: Dividir un gran problema que sea difícil de resolver directamente en algunos de menor escala, los mismos problemas, de modo que puedan romperse, dividirse y superarse. Para un problema de escala  $n$ , si el problema se puede resolver fácilmente, de lo contrario se descompone en  $k$  subproblemas de menor escala; que son independientes entre sí y del problema original. La forma es la misma, estos subproblemas que serán resueltos de forma recursiva, y luego se combinan las soluciones de los subproblemas para obtener la solución del problema original en este caso, la aplicación repetida de métodos de *divide y vencerás* puede hacer que el subproblema sea coherente con el tipo de problema original, pero su escala continúa reduciéndose, reduciendo finalmente el subproblema a una solución fácil.

### 4.3. Aplicación

1. El problema se puede resolver fácilmente reduciendo la escala.
2. El problema se puede descomponer en varios problemas idénticos de menor escala, es decir, el problema tiene la naturaleza más subestructurada.
3. Las soluciones de los subproblemas descompuestos por el problema se pueden combinar en la solución del problema.

La primera característica es que la mayoría de los problemas pueden resolverse, porque la complejidad computacional del problema generalmente aumenta con el aumento de la escala del problema.

La segunda característica es el requisito previo para la aplicación de divide y vencerás, que también puede satisfacerse con la mayoría de los problemas. Esta característica refleja la aplicación del pensamiento recursivo.

La tercera característica es la clave. La posibilidad de utilizar el método de dividir y conquistar depende totalmente de si el problema tiene la tercera característica. Si la primera y la segunda característica están disponibles, pero la tercera característica no está disponible, entonces se puede considerar el método voraz o método de programación dinámica.

### 4.4. Análisis de complejidad

Un método de divide y vencerás divide el problema de escala  $n$  en  $k$  subproblemas de escala  $n$  sobre  $m$  para resolver. Suponga que el umbral de descomposición es  $n_0 = 1$ , y el problema con una escala de solución ad hoc de 1 consume 1 unidad de tiempo. Suponga que se necesitan  $f(n)$  unidades de tiempo para descomponer el problema original en  $k$  subproblemas y fusionar las soluciones de los  $k$  subproblemas en el problema original.

Use  $T(n)$  para denotar el tiempo de cálculo requerido por el método de dividir y conquistar para resolver un problema con una escala de  $IPI = n$ , luego:

$T(n) = k T(n \text{ sobre } m) + f(n)$ . Obtenga la solución de la ecuación por método iterativo:

La ecuación recursiva y su solución solo dan el valor de  $T(n)$  cuando  $n$  es igual a la potencia de  $m$ , pero si  $T(n)$  se considera lo suficientemente suave, entonces  $n$  es igual a la potencia de  $m$ . El valor de  $T(n)$  puede estimar la tasa de crecimiento de  $T(n)$ . Generalmente se asume que  $T(n)$  aumenta monótonamente, de modo que cuando  $m_i$  menor o igual que  $n$  menor que  $m_i + 1$ ,  $T(m_i)$  menor o igual que  $T(n)$  menor que  $T(m_i + 1)$ .

### 4.5. Explicación de la solución de mejoramiento

La propuesta de mejoramiento la trabajamos en la estrategia `Parallel_Cut_Evaluation`. Para solucionar este problema observado se implementó el método de divide and conquer, tal y como se describió anteriormente; trabajando sobre el archivo `partitions.py`, esto con el objetivo de dividir todos los procesos para así poder retornar todas las posibles particiones del `mechanism` and `purview`, usando 3 métodos adicionales dividiendo también la entrada de datos y haciendo un llamado para cada parte de la entrada y así disminuir el trabajo del método principal y hacer más eficiente su funcionamiento haciendo que corriera paralelamente. Una vez terminados todos los procesos retornamos una tupla de dos listas implementado un merge de ambos.

### 4.5.1. Algoritmo

```

@partition_types.register("ALL")
def all_partitions(mechanism, purview, node_labels=None):

    for mechanism_partition in partitions(mechanism):
        mechanism_partition.append([])
        n_mechanism_parts = len(mechanism_partition)
        max_purview_partition = min((len(purview)/2), n_mechanism_parts)
        check, check2 = checkEmptyValuesPurview(max_purview_partition)
        if (check == 0 and check2 == 0):
            break

def checkEmptyValuesPurview(max_purview_partition, n_mechanism_parts, purview, mechanism_partition, mechanism, node_labels):

    for n_purview_parts in range(1, max_purview_partition + 1):
        n_empty = n_mechanism_parts - n_purview_parts
        if (n_empty == 0):
            return 0
        else:
            for purview_partition in k_partitions(purview, n_purview_parts):
                purview_partition = [tuple(_list) for _list in purview_partition]

                purview_partition.extend([()] * n_empty)

                return setPermutations(purview_partition, mechanism_partition, mechanism, node_labels)

def setPermutations(purview_partition, mechanism_partition, mechanism, node_labels):
    checkList1 = []
    checkList2 = []
    for purview_permutation in set(permutations(purview_partition)):

        parts = [
            Part(tuple(m), tuple(p))
            for m, p in zip(mechanism_partition, purview_permutation)
        ]

        if parts[0].mechanism == mechanism and parts[0].purview:
            continue

        yieldKpartitions(parts, node_labels)
        checkList1.append(parts[0].mechanism)
        checkList2.append(parts[0].purview)
    return checkList1, checkList2

def yieldKpartitions(parts, node_labels):
    yield KPartition(*parts, node_labels=node_labels)

```

Figura 7: Nuestra propuesta de mejoramiento

4.5.2. Resultados

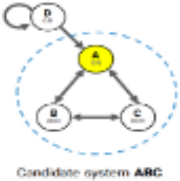
- GRAFO 1					
Entrada de datos	Clase de mejora	Descripción estrategia PyPhi	Resultado Original	Resultado con MEJORA	Observaciones
<div><div>Grafo:</div><div></div><div>Candidate system <b>ABC</b></div></div>	A	Parallel_Cut_Evaluation	<div>MIP: Cut [A, B, C] → / → [D]</div> <div>Phi: <math>\Phi = 0</math></div> <div>Tiempo: 15.907953 s</div> <div>Función de eficiencia:2n</div>	<div>MIP: Cut [A, B, C] → / → [D]</div> <div>Phi: <math>\Phi = 0</math></div> <div>Tiempo: 7.516224 s</div> <div>Función de eficiencia:2<sup>n</sup></div>	

Figura 8: Grafo 1

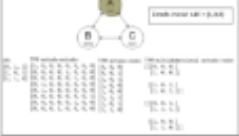
- GRAFO 2					
Entrada de datos	Clase de mejora	Descripción estrategia PyPhi	Resultado Original	Resultado con MEJORA	Observaciones
<div><div>Grafo:</div><div></div><div>Candidate system <b>ABC</b></div></div>	A	Parallel_Cut_Evaluation	<div>MIP: Cut [A, B] → / → [C]</div> <div>Phi: <math>\Phi = 1.916665</math></div> <div>Tiempo: 0.612021 s</div> <div>Función de eficiencia:2n</div>	<div>MIP: Cut [A, B] → / → [C]</div> <div>Phi: <math>\Phi = 1.916665</math></div> <div>Tiempo: 1.066230 s</div> <div>Función de eficiencia:2<sup>n</sup></div>	

Figura 9: Grafo 2

- GRAFO 3


Entrada de datos	Clase de mejora	Descripción estrategia PyPhi	Resultado Original	Resultado con MEJORA	Observaciones
<div><div>Grafo:</div></div>	A	Parallel_Cut_Evaluation	MIP: Cut [A, B] $\frac{\text{---}}{\text{---}} /$ $\frac{\text{---}}{\text{---}} \rightarrow$ [C] Phi: $\Phi = 0.520834$  Tiempo: 1.102348 s  Función de eficiencia: $2^n$	MIP: Cut [A, B] $\frac{\text{---}}{\text{---}} /$ $\frac{\text{---}}{\text{---}} \rightarrow$ [C] Phi: $\Phi = 0.520834$  Tiempo: 0.432971 s  Función de eficiencia: $2^n$	

Figura 10: Grafo 3

- GRAFO 4

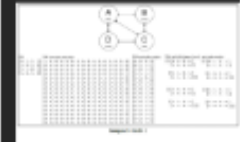
Entrada de datos	Clase de mejora	Descripción estrategia PyPhi	Resultado Original	Resultado con MEJORA	Observaciones
<div><div>Grafo:</div></div>	A	Parallel_Cut_Evaluation	MIP: Cut [A, B, C] $\frac{\text{---}}{\text{---}} / \frac{\text{---}}{\text{---}} \rightarrow$ [D] Phi: $\Phi = 0.308035$  Tiempo: 8.165864 s  Función de eficiencia: $2^n$	MIP: Cut [A, B, C] $\frac{\text{---}}{\text{---}} / \frac{\text{---}}{\text{---}} \rightarrow$ [D] Phi: $\Phi = 0.308035$  Tiempo: 6.315788 s  Función de eficiencia: $2^n$	

Figura 11: Grafo 4

- GRAFO 5


Entrada de datos	Clase de mejora	Descripción estrategia PyPhi	Resultado Original	Resultado con MEJORA	Observaciones
<div><div>Grafo:</div></div>	A	Parallel_Cut_Evaluation	<div>MIP: Cut [A, B, C, D] <math>\xrightarrow{\quad}</math> / <math>\xrightarrow{\quad}</math> [E] Phi: <math>\Phi =</math> 0.048788  Tiempo: 61.499611 s  Función de eficiencia:2n</div>	<div>MIP: Cut [A, B, C, D] <math>\xrightarrow{\quad}</math> / <math>\xrightarrow{\quad}</math> [E] Phi: <math>\Phi =</math> 0.048788  Tiempo: 73.865130 s  Función de eficiencia:2<sup>n</sup></div>	

Figura 12: Grafo 5

- GRAFO 6

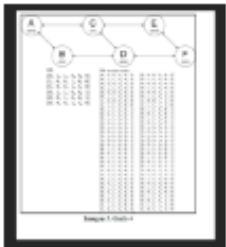
Entrada de datos	Clase de mejora	Descripción estrategia PyPhi	Resultado Original	Resultado con MEJORA	Observaciones
<div><div>Grafo:</div></div>	A	Parallel_Cut_Evaluation	<div>MIP: Cut [A, B, C, E, F] <math>\xrightarrow{\quad}</math> / <math>\xrightarrow{\quad}</math> [D] Phi: <math>\Phi = 0.250981</math>  Tiempo: 1104.147865 s  Función de eficiencia:2n</div>	<div>MIP: Cut [A, B, C, E, F] <math>\xrightarrow{\quad}</math> / <math>\xrightarrow{\quad}</math> [D] Phi: <math>\Phi = 0.250981</math>  Tiempo: 1032.875204s  Función de eficiencia:2<sup>n</sup></div>	

Figura 13: Grafo 6

## - GRAFO 7

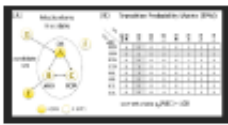
Entrada de datos	Clase de mejora	Descripción estrategia PyPhi	Resultado Original	Resultado con MEJORA	Observaciones
<b>Grafo:</b> 	A	Parallel_Cut_Evaluation	MIP: Cut [D, F] $\xrightarrow{\quad}$ / $\xrightarrow{\quad}$ [A, B, C, E] Phi: $\Phi = 4.805374$  Tiempo: 4181.075092s  Función de eficiencia: $2n$	MIP: Cut [D, F] $\xrightarrow{\quad}$ / $\xrightarrow{\quad}$ [A, B, C, E] Phi: $\Phi = 4.805374$  Tiempo: 2375.698527s  Función de eficiencia: $2^n$	

Figura 14: Grafo 7

## - GRAFO 9


Entrada de datos	Clase de mejora	Descripción estrategia PyPhi	Resultado Original	Resultado con MEJORA	Observaciones
<b>Grafo:</b> 	A	Parallel_Cut_Evaluation	MIP: Cut [A, B] $\xrightarrow{\quad}$ / $\xrightarrow{\quad}$ [C, D, E, F] Phi: $\Phi = 0.531249$  Tiempo: 480.404822s  Función de eficiencia: $2n$	MIP: Cut [A, B] $\xrightarrow{\quad}$ / $\xrightarrow{\quad}$ [C, D, E, F] Phi: $\Phi = 0.531249$  Tiempo: 480.547253s  Función de eficiencia: $2^n$	

Figura 15: Grafo 9



- GRAFO 10


Entrada de datos	Clase de mejora	Descripción estrategia PyPhi	Resultado Original	Resultado con MEJORA	Observaciones
<div><div>Grafo:</div></div>	A	Parallel_Cut_Evaluation	<div>MIP: Cut [A] ——/ /————&gt; [B, C, D, E] Phi: Φ = 0.003057  Tiempo: 444.673053 s  Función de eficiencia:2n</div>	<div>MIP: Cut [A] ——/ /————&gt; [B, C, D, E] Phi: Φ = 0.003057  Tiempo: 366.435743 s</div>	

Figura 16: Grafo 10


- GRAFO 11					
Entrada de datos	Clase de mejora	Descripción estrategia PyPhi	Resultado Original	Resultado con MEJORA	Observaciones
Grafo: 	A	Parallel_Cut_Evaluation	MIP: Cut [A] ————/ ————/ /————> [B, C, D, E] Phi: $\Phi =$ 0.009583  Tiempo: 486.660469 s  Función de eficiencia:2n	MIP: Cut [A] ———/ /————> [B, C, D, E] Phi: $\Phi = 0.009583$  Tiempo: 398.418324s  Función de eficiencia:2 <sup>n</sup>	

Figura 17: Grafo 11