Trabajo Fin de Grado
Grado en Ingeniería de Organización Industrial

Programación de Operaciones en Talleres Abiertos. Análisis de Objetivos Relacionados con la Sostenibilidad.

Autor: Pablo Rodríguez Chazarra

Tutor: Paz Pérez González

Dpto. de Organización Industrial y Gestión de Empresas I
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2022

ORGANIZACIÓN INDUSTRIAL

Trabajo Fin de Grado
Grado en Ingeniería de Organización Industrial

Programación de Operaciones en Talleres Abiertos. Análisis de Objetivos Relacionados con la Sostenibilidad.

Autor:
Pablo Rodríguez Chazarra

Tutor:
Paz Pérez González
Profesor Titular

Dpto. de Organización Industrial y Gestión de Empresas I
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2022

Trabajo Fin de Grado: Programación de Operaciones en Talleres Abiertos. Análisis de Objetivos Relacionados con la Sostenibilidad.

Autor: Pablo Rodríguez Chazarra
Tutor: Paz Pérez González

El tribunal nombrado para juzgar el trabajo arriba indicado, compuesto por los siguientes profesores:

Presidente:

Vocal/es:

Secretario:

acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

Agradecimientos

Me gustaría comenzar agradeciendo a todos y cada uno de los profesores que me han impartido clase durante estos cuatro años por tratar de enseñarnos de la mejor forma posible, en especial, a mi tutora Paz Pérez González por todo el interés mostrado durante este tiempo y por haberme abierto las puertas a este mundo de la programación de la producción.

Por supuesto también quiero agradecer a mi familia, por el esfuerzo que han realizado para yo poder estar donde estoy y por todo el apoyo y cariño que me han hecho sentir a pesar de los 500 kilómetros de distancia que nos separan.

Por otro lado, dar las gracias a mis compañeros de clase que me han acompañado durante el camino y han hecho que esta experiencia se convierta en una etapa única en mi vida. Y, como no, a mis amigos de Orihuela, que pese a la distancia me han dado el mismo calor de siempre.

Y por último quiero agradecerte a ti, María. Por estar siempre a mi lado, apoyarme como nadie, confiar siempre en mí y no dejar que nunca me rinda.

Pablo Rodríguez Chazarra
Sevilla, 2022

Resumen

Tras revisar las contribuciones literarias existentes hasta el momento, referentes a la programación de operaciones del Open Shop, se ha observado que ninguna de ellas se estudia teniendo en cuenta objetivos de sostenibilidad. Por ello, se ha identificado la necesidad de abordar el problema de programación de operaciones del Open Shop en este Trabajo de Fin de Grado, con el objetivo de afrontar uno de los principales problemas que encontramos hoy en día como es la eficiencia energética. Fundamentalmente, se trata de encontrar la secuencia de operaciones que minimice el Makespan (que viene determinado por la fecha en la que finaliza la última operación) y el Core Idle Time (que hace referencia al tiempo que las máquinas se encuentran encendidas pero sin procesar ningún trabajo). Esto se debe a que minimizar estos objetivos contribuye a reducir el consumo energético.

Se ha estudiado el Open Shop para tres objetivos diferentes:

- Objetivo 1: Minimización del Total Core Idle Time.
- Objetivo 2: Minimización del Makespan.
- Objetivo 3: Minimización de una función multiobjetivo que es función lineal convexa combinando el Total Core Idle Time y el Makespan en función de un parámetro λ.

Por otro lado, a lo largo del documento, se van a introducir conceptos básicos de programación de la producción y se va a realizar un estudio de la literatura para, posteriormente, plantear de forma detallada los problemas a resolver mediante la programación de diferentes heurísticas. Finalmente, se analizarán los resultados conseguidos para la obtención de conclusiones.

Estos problemas se han resuelto para cuatro heurísticas que se basan en operadores de inserción y reinserción que buscan establecer una programación de las operaciones que cumpla con las restricciones del problema. Además, se comparan en términos de eficiencia y de eficacia, mediante el siguiente análisis:

- Comparación de los valores devueltos por la función objetivo a minimizar, así como los valores del resto de objetivos evaluados para la solución proporcionada.
- Comparación de los tiempos de CPU necesarios para alcanzar las soluciones.

Todas las heurísticas que se han desarrollado se han implementado mediante el lenguaje de programación Python en el entorno Anaconda/Spyder. Además, el presente documento ha sido redactado en Latex con la herramienta Overleaf.

Por último, hay que tener en cuenta que el presente Trabajo de Fin de Grado forma parte de una Beca de Iniciación a la Investigación que se está desarrollando junto con el departamento de Organización Industrial y Gestión de Empresas I (Grupo de Investigación de Organización Industrial).

Índice

Índice de Figuras

Índice de Tablas

Notación

i	Índice de las máquinas, $1 \leq i \leq m$
j	Índice de los trabajos, $1 \leq j \leq n$
$[k]$	Índice de la posición de los trabajos, $1 \leq k \leq n$
n	Número de trabajos
m	Número de máquinas
d_j	Fecha de entrega
r_j	Fecha de llegada
C_{ij}	Instante de finalización del trabajo i en la máquina j
O_{ij}	Operación del trabajo i en la máquina j
s_{ij}	Instante más temprano en el que la operación O_{ij} podría empezar
f_{ij}	Instante de tiempo en el que la operación O_{ij} acabará
atm_j	Tiempo en el que la máquina j volverá a estar disponible
at_i	Tiempo en el que el trabajo i volverá a estar disponible para la proxima operación
Pt_{ij}	Tiempo de proceso del trabajo i en la máquina j
Et_{ij}	Etiqueta correspondiente al trabajo i en la máquina j
$CIT_{i,[k]}$	Core Idle Time de la máquina i siendo procesado el trabajo de la posición k
C	Caso
H	Heurística

1 Introducción

"Una persona debe fijar sus objetivos cuanto antes y dedicar toda su energía y talento a ellos"

Walt Disney

A lo largo de este capítulo se presenta cuál es la motivación y los objetivos del proyecto, una breve descripción de cada uno de los puntos del documento y una introducción de conceptos básicos de la programación de la producción.

1.1 Motivación del Trabajo Fin de Grado

La motivación del presente Trabajo Fin de Grado proviene de la necesidad de seguir luchando contra una de las principales preocupaciones que encontramos hoy en día en nuestra sociedad: la eficiencia energética. La eficiencia energética hace referencia a la capacidad para obtener resultados óptimos en cualquier actividad empleando la menor cantidad posible de recursos energéticos para así reducir el consumo de cualquier tipo de energía y, con ello, los posibles impactos ambientales asociados a ella (Repsol, 2022).

En 2015, los Objetivos de Desarrollo Sostenible u Objetivos Globales, fueron adoptados por las Naciones Unidas como un llamamiento universal para poner fin a la pobreza, proteger el planeta y garantizar que para el 2030 todas las personas disfrutaran de paz y prosperidad. De los 17 objetivos establecidos, este Trabajo Fin de Grado se encuentra alineado principalmente con uno de ellos: Producción y Consumo Responsables (PNUD, 2022).

El continuo aumento de las emisiones de gases de efecto invernadero ha llevado a muchas empresas a investigar las actividades que tienen un mayor impacto sobre el medio ambiente. Estudios recientes estiman que alrededor del 10 % de las emisiones mundiales de CO_2 se derivan de las cadenas logísticas de suministro. La considerable cantidad de energía necesaria para la calefacción, la refrigeración y la iluminación, así como para los equipos de manipulación de materiales en los almacenes, representa alrededor del 20 % de los costes logísticos totales. La reducción del consumo energético de los almacenes supondría, por tanto, un importante beneficio desde el punto de vista medioambiental y económico (Carli et al.).

Se puede afirmar por lo tanto, que la programación de la producción es una de las tareas más importantes dentro de las empresas manufactureras, así como un problema de decisión en el que se pueden introducir variables relacionadas con los costes energéticos de forma sencilla para tener en cuenta la sostenibilidad.

Respecto al entorno, se ha decidido seleccionar el Open Shop (a pesar de ser el más complejo de los entornos) porque este trabajo forma parte de una Beca de Iniciación a la Investigación que se está desarrollando junto con el departamento de Organización Industrial y Gestión de Empresas I (Grupo de Investigación de Organización Industrial). Además, el Open Shop permite implementar soluciones derivadas de este problema en entornos más sencillos como el Job Shop o el Flow Shop.

1.2 Objetivos del Trabajo Fin de Grado

El objetivo principal de este proyecto es el análisis de un problema de programación de la producción en un entorno productivo de tipo Taller Abierto. El Open Shop es de los menos estudiados en la literatura por la complejidad que implica la codificación de las soluciones y el alto número de soluciones factibles, que hacen que el problema de optimización sea intratable (NP-hard) en instancias con tamaños pequeños, para la mayoría de los objetivos que se consideren, por ejemplo el Makespan (Ahmadian et al., 2021). Además, este entorno con objetivos de sostenibilidad aún no está siendo considerado en la literatura tal y como se puede comprobar al realizar la búsqueda en Scopus con un conjunto de palabras claves como "Open Shop", "Scheduling", "Sustainable/Sustainability/Energy Consumption" con 0 resultados relevantes en todas ellas.

En Scheduling en general, el número de publicaciones con consideraciones de sostenibilidad está aumentado de manera exponencial, para tener en cuenta en las empresas no solo la eficiencia del sistema, sino aspectos medioambientales. El estudio de un Open Shop con objetivos relacionados a la minimización de los costes energéticos o de emisiones de CO_2, es por lo tanto, el objetivo principal de este trabajo. Además, este objetivo general se puede dividir en los siguientes objetivos específicos:

- Estudio del estado del arte: Revisión bibliográfica de la codificación de las soluciones en entornos Open Shop.
- Análisis del problema: Modelado del problema y estudio de la eficiencia de las diferentes codificaciones de las soluciones de la literatura para objetivos de sostenibilidad.
- Resolución del problema mediante métodos exactos y métodos heurísticos eficientes. Este objetivo implica:
  - Programación de métodos existentes en la literatura para problemas similares.
  - Comparativa de resultados, para analizar la eficiencia y eficacia de los métodos desarrollados, y seleccionar los más apropiados al problema planteado.
- Aprender un nuevo lenguaje de programación, mediante la utilización de Python en el entorno Anaconda/Spyder, para desarrollar todas las heurísticas del documento.
- Aprender a desarrollar documentos científicos en Latex con la herramienta Overleaf.

Una vez conocidos cuales son los objetivos del proyecto, se detalla en la siguiente sección una breve descripción del documento proporcionando información acerca de lo que se puede encontrar en los diferentes capítulos.

1.3 Estructura del documento

El presente trabajo se estructura en un documento principal que consta de cinco apartados y un conjunto de anexos. En el primer apartado se introduce y caracteriza el Trabajo Fin de Grado, incluyéndose cual es la motivación y los objetivos del mismo. A continuación, se introduce la estructura documento y se explican unos conceptos básicos de programación de la producción, pero el verdadero núcleo del Trabajo Fin de Grado está constituido por los apartados 2, 3 y 4.

En el apartado 2 se encuentran los antecedentes y la descripción del problema de forma detallada. Para ello, se ha dividido en diferentes secciones tales como la revisión de la literatura, posibles aplicaciones de un Open Shop y una explicación de las hipótesis del problema y las funciones objetivos.

Por su parte, en el apartado 3, encontramos la metodología del proyecto, donde por un lado se habla de los teoremas que se han implementado para minimizar el tiempo de computo eliminando el estudio de soluciones redundantes, y, por otro lado, se realiza una breve explicación de las diferentes heurísticas.

Por último, en el apartado 4 encontramos la experimentación, el motivo por el cual se ha usado python como lenguaje de programación, la generación de las instancias del problema y, finalmente, la calibración y comparación de las diferentes heurísticas.

Además, el documento se complementa con un apartado donde se recoge la bibliografía y un anexo donde se incluye el código que se ha elaborado para el desarrollo de las heurísticas a las que se hace referencia en el documento principal. Los diferentes puntos que se encuentran en el anexo son:

- Código Heurística 1 para Makespan
- Código Heurística 2 para Makespan
- Código Heurística 3 para Makespan
- Código Heurística 4 para Makespan
- Código Función objetivo: Core Idle Time
- Código Función objetivo: Ponderación Makespan y Core Idle Time
- Código Traductor de instancias Taillard 1
- Código Traductor de instancias Taillard 2

Una vez conocida la estructura del documento, se muestra una breve descripción de ciertos conceptos que resultan de interés para la correcta compresión del mismo.

1.4 Programación de la producción

La programación de la producción se define como la asignación de recursos para la fabricación de un conjunto de productos. Además, cuando se programa la producción, se obtiene un Schedule, que es un diagrama de gant en el que se facilita información sobre cuándo debe comenzar cada máquina a procesar qué operación (Framinan et al., 2014).

1.4.1 Datos básicos del modelo

En este punto, resulta interesante incluir la descripción de unos conceptos básicos de programación de la producción (Framinan et al., 2014).

- Máquina: Cualquier recurso productivo con capacidad para realizar operaciones de transformación o transporte de material.
- Trabajo: Aquello que se debe procesar mediante operaciones en alguna de las máquinas de la fábrica.
- Operación: Actividad de cada uno de los diferentes trabajos que es realizada en las diferentes máquinas.
- Tiempo de proceso: Tiempo que tarda en realizarse una operación del trabajo j en la máquina i.
- Fecha de llegada o Release Date: Instante más temprano posible en el que se puede procesar una operación.
- Fecha de entrega o Due Date: Instante de tiempo en el que un trabajo debe estar terminado.
- Tiempo de inactividad o Idle Time: Tiempo en el que la máquina está sin trabajar.

Por otro lado, resulta importante conocer cual es la diferencia entre programa y secuencia. Se define programa o Schedule como la asignación de fechas de comienzo de cada trabajo en cada máquina, mientras que la secuencia, se refiere al orden en que cada trabajo comienza a procesarse en cada máquina. Mencionar que a lo largo del documento se utilizará de forma indistinta "Programa" o "Schedule" ya que significan lo mismo.

Entre los conceptos mencionados anteriormente se pueden establecer diferentes relaciones, pero las que serán utilizadas a lo largo del proyecto son:

- Una máquina no puede hacer todas las operaciones.
- Una máquina solo puede procesar un trabajo a la vez.

1.4.2 Clasificación de los modelos

A lo largo del documento se seguirá una clasificación basada en tres elementos (Framinan et al., 2014):

- α: Entorno del proceso.
- β: Restricciones del problema.
- γ: Función objetivo.

Entorno del proceso "α"

El entorno del proceso "α" está definido según la disposición de las máquinas y las especificaciones de las diferentes rutas. A continuación se detallan algunos de los principales entornos que podemos encontrar.

- Una máquina (Single Machine), cuyo esquema se muestra en la Figura 1.1

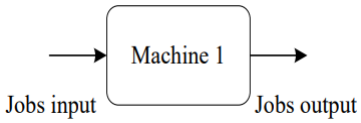

Figura 1.1 Esquema Single Machine (Framinan et al., 2014).

  - Notacion: $\alpha = 1$.
  - Cada trabajo tiene que procesarse en la máquina.
  - Cada trabajo tiene una única operación.
  - En este caso no existe ruta de los trabajos.
  - Número de soluciones (programas semiactivos): $n!$.
- Máquinas en paralelo (Parallel Machines), cuyo esquema se muestra en la Figura 1.2

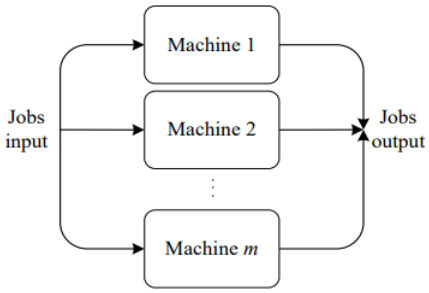

Figura 1.2 Esquema Parallel Machines (Framinan et al., 2014).

  - Notación: depende de la velocidad de las máquinas.
  - Réplica del entorno de una máquina para el incremento de la producción.
  - Cada trabajo tiene una única operación, es decir, se procesa en una de las máquinas.
  - Normalmente todas las máquinas son susceptibles de realizar el trabajo.
  - Número de soluciones (programas semiactivos): $\binom{n+m-1}{n}$.

  Podemos encontrar diferentes tipos de máquinas paralelas:
  - Idénticas $\alpha = Pm$. El tiempo de procesar cada trabajo no depende de la máquina.
  - Uniformes $\alpha = Qm$. Máquinas con diferentes velocidades.
  - No relacionadas $\alpha = Rm$. El tiempo de proceso de cada trabajo depende de la máquina a la que sea asignado.
- Taller de flujo regular (Flow Shop), cuyo esquema se muestra en la Figura 1.3

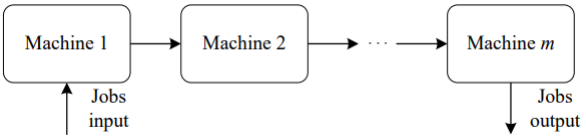

Figura 1.3 Esquema Flow Shop (Framinan et al., 2014).

  - Notación: $\alpha = Fm$.
  - Todos los trabajos tienen la misma ruta.
  - Número de soluciones (programas semiactivos): $(n!)^m$.

- Taller de trabajos (Job Shop), cuyo esquema se muestra en la Figura 1.4

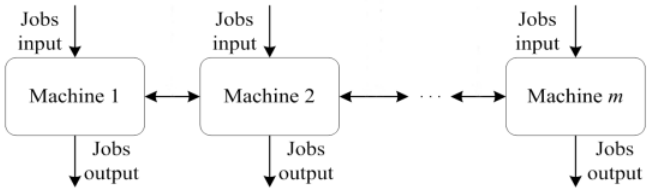

Figura 1.4 Esquema Job Shop (Framinan et al., 2014).

  - Notación: $\alpha = Jm$.
  - Cada trabajo tiene una ruta diferente.
  - Número de soluciones (programas semiactivos): $(n!)^m$.

- Taller abierto (Open Shop), cuyo esquema se muestra en la Figura 1.5

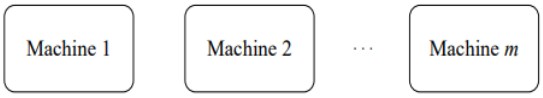

Figura 1.5 Esquema Open Shop (Framinan et al., 2014).

  - Notación: $\alpha = Om$.
  - Es el más general y complejo de los talleres.
  - No hay ruta predeterminada.
  - Número de soluciones (programas semiactivos): $m!(n!)^m$.

Restricciones del problema "β"

Existen gran cantidad de restricciones que limitan la programación de la producción. Entre ellas encontramos:

- Trabajos disponibles al principio del horizonte de programación.
- Los trabajos no se pueden interrumpir.
- Máquinas siempre disponibles.
- Cada máquina puede hacer un trabajo, y un trabajo puede ser realizado sólo en una máquina.
- El buffer o zona de almacenamiento temporal entre máquinas se supone infinito.
- El tiempo de transporte es despreciable.

Función objetivo "γ"

En la programación de la producción, las funciones objetivos se pueden clasificar en: coste, tiempo, calidad y flexibilidad. A continuación, se presenta en la Figura 1.6 un esquema con la clasificación de las funciones objetivos.

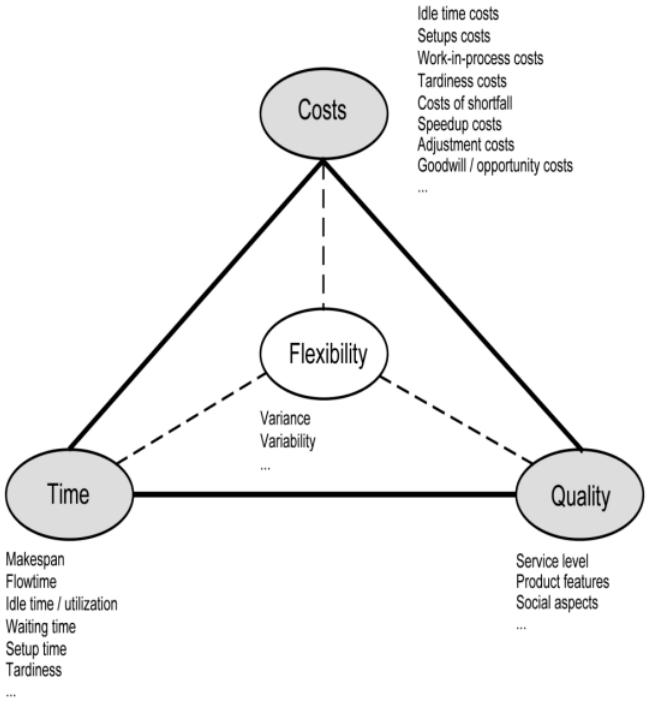

Figura 1.6 Clasificación Funciones Objetivos (Framinan et al., 2014).

Los objetivos empleados en los modelos de programación de la producción están relacionados con la optimización de algún indicador, los cuales se conocen como medidas del rendimiento. Entre ellos se pueden diferenciar (Framinan et al., 2014):

- Completion time o tiempo de finalización del trabajo j (C_j): Instante de tiempo en el que el trabajo j finaliza.
- Tardy job o trabajo tarde (U_j): Tomará el valor 1 si el trabajo finaliza más tarde y 0 en caso contrario.

$$U_j = \begin{cases} 1 \; si \; C_j \leq d_j \\ 0 \; si \; C_j > d_j \end{cases}$$

  Nota: d_j hace referencia a la fecha de entrega, es decir, al instante de tiempo en el que el trabajo j debe estar terminado.
- Tardiness o tardanza del trabajo j (T_j): Indicador que hace referencia a cuanto de tarde ha llegado el trabajo j con respecto a su fecha de entrega.

$$T_j = max\{C_j - d_j, 0\}$$

- Lateness o retraso del trabajo j (L_j): Retraso del trabajo j con respecto a su fecha de entrega.

$$L_j = C_j - d_j$$

- Earliness o adelanto del trabajo j (E_j): Se calcula como la diferencia entre la fecha de entrega y la fecha de finalización.

$$E_j = max\{d_j - C_j, 0\}$$

- Flow time o tiempo de flujo del trabajo j (F_j): Periodo de tiempo que el trabajo pasa en el sistema desde que entra hasta que sale.

$$F_j = C_j - r_j$$

  Nota: r_j hace referencia a la fecha de llegada, es decir, al instante de tiempo a partir del cual el trabajo j puede empezar a ser procesado.

2 Antecedentes y descripción del problema

"El mundo es un lugar peligroso, no a causa de los que hacen el mal sino por aquellos que no hacen nada para evitarlo"

Albert Einstein

Este proyecto se centra en el análisis de la codificación de las soluciones para los problemas de programación de la producción en entorno tipo Taller Abierto (Open Shop Scheduling). Para este tipo de problemas, la codificación de las soluciones afecta sustancialmente a la eficiencia de los métodos de resolución y, además, es altamente dependiente de los objetivos de optimización que se planteen en el problema.

2.1 Revisión de la literatura

En general, este problema ha sido ampliamente estudiado para el Makespan tal y como se puede apreciar en la literatura reciente (Ahmadian et al., 2021). En la actualidad, los objetivos considerando el punto de vista medioambiental, junto con otros objetivos de eficiencia, se están estudiando en problemas de programación de la producción multiobjetivo. Así, los artículos de programación de la producción en los que se consideran objetivos sobre sostenibilidad (Sustainable Scheduling) están aumentando en los últimos años. Estos objetivos tienen en cuenta el medio ambiente y los costes energéticos, que son aspectos fundamentales en las empresas hoy en día. Por ello, a la hora de programar de forma eficiente la producción desde el punto de vista de económico, se unen a objetivos clásicos como la minimización del Makespan (con relación directa con la eficiencia del sistema y con los costes de consumos de energía (Han et al., 2020)), nuevos objetivos como costes de consumo de electricidad (Ho et al., 2021), mano de obra (Han et al., 2019), inventario, encendido/apagado de máquinas (Shrouf et al., 2014) o tiempos ociosos de las máquinas (He et al., 2019). Así mismo, han emergido nuevos objetivos desde el punto de vista del medio ambiente, como la minimización del consumo de energía total (Zhang et al., 2019), demanda de picos de energía (Kelley et al., 2019), cantidad de deshechos como puede ser el desperdicio de un proceso de corte (Aziz et al., 2019) o el uso de agua (Liu et al., 2012).

Para profundizar en el estudio del arte, resulta de interés conocer cuáles son las posibles formas de codificar el Open Shop. Se pueden identificar tres formas diferentes de hacerlo: matriz de rango, lista de permutación y representación gráfica disyuntiva (Abreu et al., 2022) .

- Matriz de rango: Es una matriz en la que cada fila muestra el orden de las operaciones de un trabajo en diferentes máquinas y cada columna muestra el orden de los trabajos en una máquina.
- Lista de permutación: Es una lista con orden lineal de todas las operaciones.

Sin embargo, la representación gráfica disyuntiva no se estudiará debido a que exclusivamente es válida para el objetivo del Makespan y por consiguiente no es la finalidad buscada.

Por otro lado, los Talleres Abiertos presentan una amplia gama de aplicaciones en diferentes áreas, tal y como puede ser la programación del horario, la comunicación por satélite, la salud, el transporte y el turismo (Ahmadian et al., 2021). A modo de ejemplo, se detallan algunas aplicaciones ya desarrolladas de Talleres Abiertos:

- Un taller de automóviles con secciones especializadas para reparar los distintos tipos de fallos. Encontramos un conjunto de automóviles que requieren diferentes tipos de reparaciones que van desde servicios generales de mantenimiento como el cambio de aceite o la revisión de partes eléctricas a servicios especializados, como por ejemplo, el cuadro. No se pueden ejecutar dos operaciones en un automóvil simultáneamente debido a las diferentes ubicaciones de las secciones especializadas y las características de las operaciones. Por otro lado, el orden de realizar las operaciones por parte de los mecánicos es irrelevante (Ahmadian et al., 2021).
- La programación de tareas de las revisiones de los aviones. Considerando una flota de aviones, es necesario realizar un conjunto de operaciones en cada avión para prepararlo para el despegue. Cada operación debe ser realizada por un técnico (o una máquina) que solo puede realizar esa operación. Es cierto que hay operaciones que se pueden realizar simultáneamente como por ejemplo que un técnico revisa el motor mientras el otro técnico inspecciona las alas, pero hay otras operaciones que no se pueden realizar al mismo tiempo, lo que lleva, por lo tanto, al entorno clásico del Open Shop (Ahmadian et al., 2021).
- La programación de pacientes para el diagnóstico de la enfermedad coronaria. El paciente necesita someterse al diagnóstico en cualquier orden a tres etapas, análisis de sangre, cardiograma ultrasónico y computación coronaria tomografía. Cada etapa requiere múltiples instalaciones y/o personal médico para realizar el diagnóstico (Ahmadian et al., 2021).

2.2 Descripción del problema

2.2.1 Open Shop

Como se ha comentado con anterioridad, este trabajo se centra en la programación de operaciones en Talleres Abiertos (Open Shop), que son aquellos en los que se tienen diferentes máquinas que realizan distintas operaciones.

Un trabajo está formado por operaciones y, en general, deberá pasar por todas las máquinas para completar cada una de estas operaciones. Es decir, en un Open Shop se identifican m máquinas dispuestas en serie y n trabajos que tienen que visitar las m máquinas. Además, el Open Shop se diferencia de otros entornos como el Job Shop o el Flow Shop, en que la ruta que deben seguir los trabajos para ser procesados por las máquinas no está definida, es decir, forma parte de la propia decisión ya que las diferentes operaciones de un trabajo se pueden procesar con cualquier orden en las máquinas.

2.2.2 Codificación

Tal y como se ha visto en la revisión de la literatura (Abreu et al., 2022), existen diferentes codificaciones, pero este Trabajo Fin de Grado se centra en la *lista de permutación de operaciones*. Esta codificación consiste en que a cada operación, es decir, a cada trabajo en cada máquina se le asigna una etiqueta numérica que es la que se incluye en la secuencia. Esta codificación presenta numerosas ventajas como pueden ser la extraordinaria adaptabilidad a cualquier operador y la fácil implementación para el desarrollo de los algoritmos. Sin embargo, la principal desventaja que presenta dicha codificación es la redundancia del Schedule puesto que, a veces, al cambiar el orden de las operaciones, la secuencia puede proporcionar el mismo programa. Para eliminar aquellas soluciones redundantes y conseguir una ejecución más rápida, se aplicarán los diferentes teoremas que propone (Naderi et al., 2010).

Las heurísticas se codifican con una secuencia de tamaño nxm (ya que no se secuencian los trabajos sino las operaciones) a partir de la cual se obtiene un Schedule. En el apartado 2.4 se lleva a cabo un ejemplo detallado donde se explica como generar el Schedule a partir de la secuencia.

Los programas que se representan son siempre semiactivos, esto quiere decir que a la hora de la representación en el Schedule se sitúan lo más a la izquierda posible. Por otro lado, un programa non-delay tiene por definición que ninguna máquina debe estar ociosa si existe alguna operación que se pueda procesar en ella. Por consiguiente, un programa semiactivo no siempre es non-delay ya que este último proporciona mejores valores del Total Core Idle Time. Conviene resaltar que cuando a un programa se le aplican modificaciones de tal forma que se considere non-delay (el tiempo de inicio de alguna operación se ve adelantado) no se puede obtener a partir del nuevo Schedule la secuencia que representa el mismo.

En resumen, a partir de una secuencia se puede generar un Schedule, pero a partir de un Schedule modificado no se puede obtener su correspondiente secuencia. Para facilitar la interpretación se presenta la Figura 2.1 que hace referencia al Schedule obtenido a partir de la secuencia $Sec = (6,10,12,14,7,4,2,9,11,1,3,13,8,0,5)$.

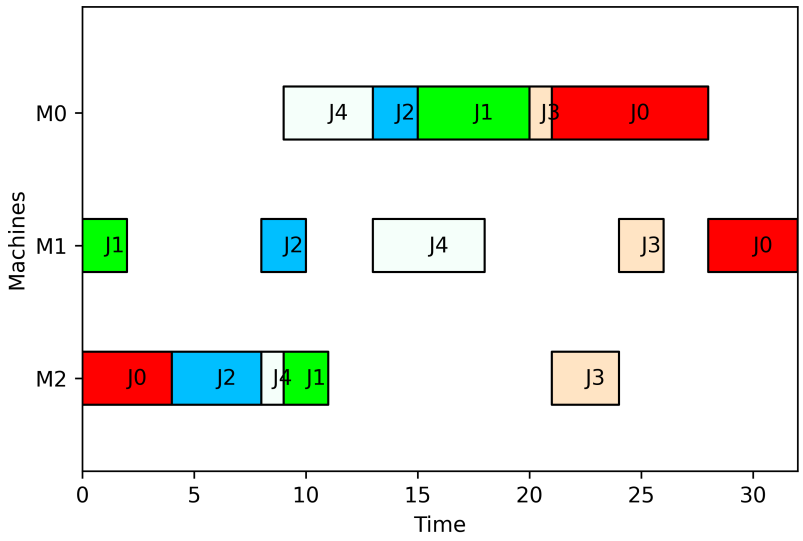

Figura 2.1 Schedule Completo (Elaboración propia).

En la figura anterior, se puede observar como el trabajo 3 de la máquina 1 tiene una duración de dos unidades de tiempo. Además, en esa misma máquina disponemos de Core Idle Time entre los trabajos 1-2 y los trabajos 2-4, siendo ambos superiores al tiempo de proceso del trabajo 3 de esa misma máquina. En esta situación, ese Trabajo 3 de la máquina 1 se podría introducir en cualquiera de esos Core Idle Time modificando por consiguiente el Schedule. En este punto ya no se podría seguir permutando puesto que se desconoce cual es la secuencia que le corresponde al nuevo Schedule.

2.2.3 Hipótesis

Finalmente, se mencionan una serie de suposiciones generales que se han establecido para la realización del documento y que, por consiguiente, son claves para la comprensión del mismo (Framinan et al., 2014).

- Todos los trabajos están disponibles al principio del horizonte de programación.
- Los trabajos no se pueden interrumpir.
- Las máquinas siempre están disponibles al principio del horizonte de programación.
- Cada máquina puede hacer un trabajo, y un trabajo puede ser realizado sólo en una máquina al mismo tiempo.
- El buffer entre máquinas se supone infinito.
- El tiempo de transporte es despreciable.

2.3 Funciones objetivos

En este apartado, se presentan dos objetivos: minimizar el Total Core Idle Time y el Makespan. Primero se estudiará el Total Core Idle Time y, para poder comprenderlo con facilidad, se muestra la Figura 2.2 en la que se puede observar un Schedule con una nomenclatura de interés.

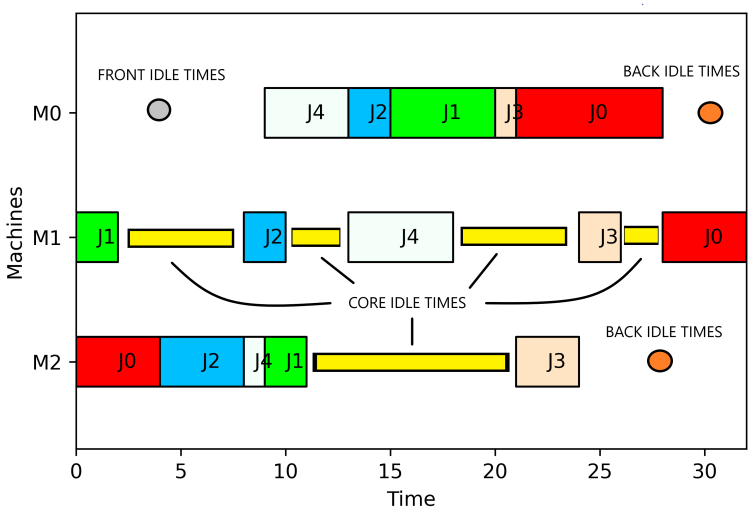

Figura 2.2 Schedule Explicación Total Core Idle Time (Elaboración propia).

El tiempo de inactividad de una máquina es aquel en el que la máquina está sin trabajar. Esto puede ser debido a varios factores:

- Tiempo en el que la máquina no tiene trabajo que realizar.
- Tiempo en el que según la planificación o programación, la máquina debe estar sin trabajar.
- Tiempo debido a una avería o mantenimiento en la máquina.

Este tiempo de inactividad o Idle Time, se puede clasificar en:

- Front Idle Time: El Idle Time que se encuentra antes de procesarse el primer trabajo, es decir tiempo en el que la máquina aun no ha procesado ningún trabajo y por consiguiente aún no se ha encendido.
- Core Idle Time: El Idle Time que se encuentra entre operaciones. Este es el Idle Time que vamos a tratar de minimizar para tratar que el tiempo que una máquina esté encendida y sin procesar ninguna operación sea el menor posible.
- Back Idle Time: El Idle Time que se sitúa después de procesarse el último trabajo, es decir, tiempo en el que la máquina ya ha terminado de procesar todos los trabajos y por consiguiente está apagada.

El objetivo final va a ser minimizar el Idle Time que se encuentra entre operaciones, es decir, no se va a tener en cuenta ni el Front Idle Times ni el Back Idle Times. Esto es debido a que el principal objetivo es tratar que el tiempo que transcurra desde que se encienda hasta que se apague la máquina sea el menor tiempo posible, reduciendo así el consumo energético. Siendo $s_{i,[k]}$ el tiempo de inicio del trabajo de la posición k en la máquina i y $C_{i,[k-1]}$ el tiempo de finalización del trabajo en la posición k-1 en la máquina i, en la Ecuación 2.1 se presenta la ecuación del Core Idle Time. Por otro lado, se considera el Total Core Idle Time como el sumario del Core Idle Time en todas las máquinas, tal como se indica en la Ecuación 2.2.

$$CIT_i = \sum_{k=2}^{n} (s_{i,[k]} - (C_{i,[k-1]}) \tag{2.1}$$

$$TCIT = \sum_{i=1}^{m} CIT_i \tag{2.2}$$

- El Core Idle Time es igual a la diferencia entre el tiempo de inicio del trabajo que se encuentra en la posición k de la máquina i menos el tiempo de finalización de la trabajo de la posición k-1 en la máquina i.
- El sumatorio de los diferentes Core Idle Time dan lugar al Total Core Idle Time.

Una vez presentado el Total Core Idle Time, se muestra el otro objetivo: el Makespan. El Makespan se define como el tiempo necesario para terminar de realizar todo el plan de producción, es decir, desde el momento en que comienza a procesarse el primer trabajo (normalmente se supone que es cero, a menos que existan tiempos de liberación u otras restricciones), hasta el momento en el que finaliza el último trabajo de la secuencia en la última máquina que visita (Framinan et al., 2014). Como se puede observar en la Figura 2.3, la primera operación se inicia en el instante 0 y la última operación finaliza en el instante 32. La expresión del Makespan se presenta en la Ecuación 2.3.

$$C_{max} = \text{máx}\left\{C_{ij}\right\} \tag{2.3}$$

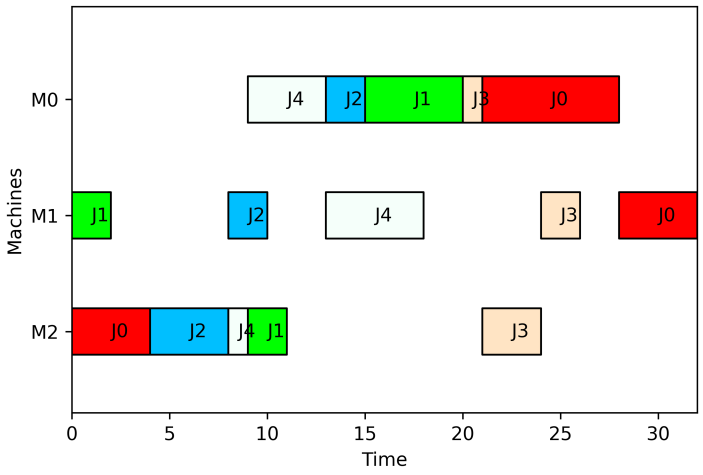

Figura 2.3 Schedule Explicación Makespan (Elaboración propia).

Finalmente, la función multiobjetivo que se utilizará en este documento es la función lineal convexa que combina los dos objetivos anteriores ponderándolos en función de una variable λ que varia entre 0 y 1.

$$mo_{\lambda} = \lambda TCIT + (1-\lambda)C_{max} \tag{2.4}$$

Como se puede deducir de la ecuación anterior, cuando $\lambda = 0$ únicamente se tiene en cuenta el Makespan y cuando $\lambda = 1$ el Total Core Idle Time.

2.4 Ejemplo detallado

A continuación se resuelve paso a paso como se representaría el Schedule para la codificación seleccionada, así como el cálculo de ambas funciones objetivos para un ejemplo propuesto. Para ello, es necesario conocer tal y como se refleja en la Tabla 2.1 los tiempos de proceso.

Tabla 2.1 Tiempos de Proceso de las Operaciones (Elaboración propia).

Pt_{ij}	0	1	2	3	4
0	7	5	2	1	4
1	4	2	2	2	5
2	4	2	4	3	1

Una vez conocida la tabla de los tiempos de proceso, se pueden establecer las etiquetas de cada una de las operaciones tal y como se muestra en la Tabla 2.2. A partir de estas etiquetas, se obtendrá la secuencia.

Tabla 2.2 Etiquetas de las Operaciones (Elaboración propia).

Et_{ij}	0	1	2	3	4
0	0	1	2	3	4
1	5	6	7	8	9
2	10	11	12	13	14

Se propone como secuencia a estudiar: $Sec = (6,10,12,14,7,4,2,9,11,1,3,13,8,0,5)$. Una vez comprendida la información anterior se puede comenzar a elaborar el Schedule. La primera operación que se debe situar es la operación 6. Tal y como muestra la Tabla 2.2 es la operación del trabajo 1 en la máquina 1. Por otro lado, la Tabla 2.1 indica la duración de esa operación, en este caso concreto, 2 unidades de tiempo. Hecho este análisis, se puede afirmar que se debe introducir la operación 6 en la máquina 1 y tendrá una duración de 2 unidades de tiempo. En la Figura 2.4 se muestra el resultado tras incluir dicha operación.

$Sec = (\mathbf{6},10,12,14,7,4,2,9,11,1,3,13,8,0,5)$

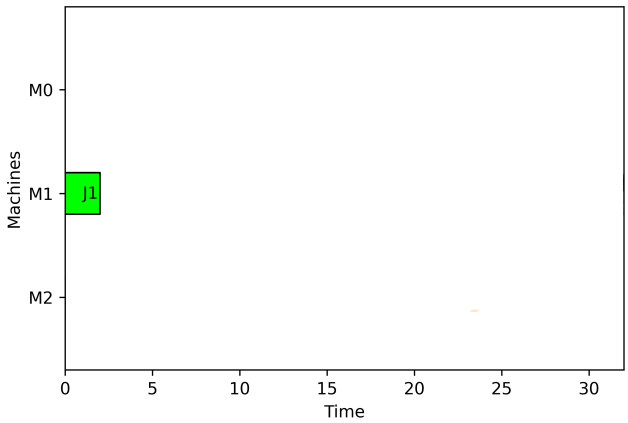

Figura 2.4 Schedule Operación 1 (Elaboración propia).

Una vez se ha colocado la operación 6, se pasa a la operación 10 siguiendo el mismo procedimiento explicado anteriormente. Tal y como muestra la Tabla 2.2 es la operación del trabajo 0 en la máquina 2. Por otro lado, la Tabla 2.1 indica la duración de esa operación, en este caso concreto, 4 unidades de tiempo. En la Figura 2.5 se muestra el resultado tras incluir dicha operación.

$Sec = (6,\mathbf{10},12,14,7,4,2,9,11,1,3,13,8,0,5)$

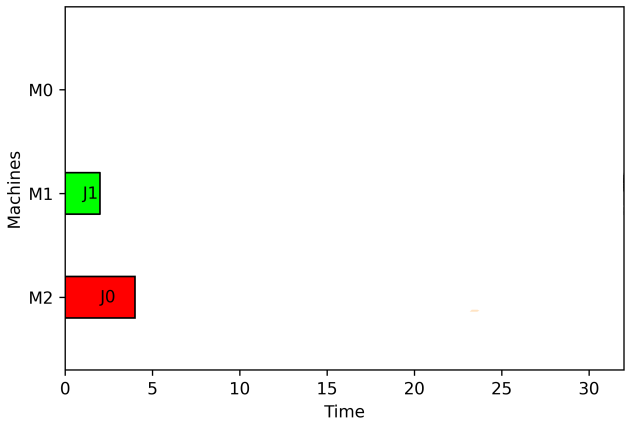

Figura 2.5 Schedule Operación 2 (Elaboración propia).

Tras la operación 10 se encuentra la operación 12. Tal y como muestra la Tabla 2.2 es la operación del trabajo 2 en la máquina 2. Por otro lado, la Tabla 2.1 indica la duración de esa operación, en este caso concreto, es 4 unidades de tiempo. En la Figura 2.6 se muestra el Schedule actualizado tras incluir dicha operación.

$Sec = (6,10,\mathbf{12},14,7,4,2,9,11,1,3,13,8,0,5)$

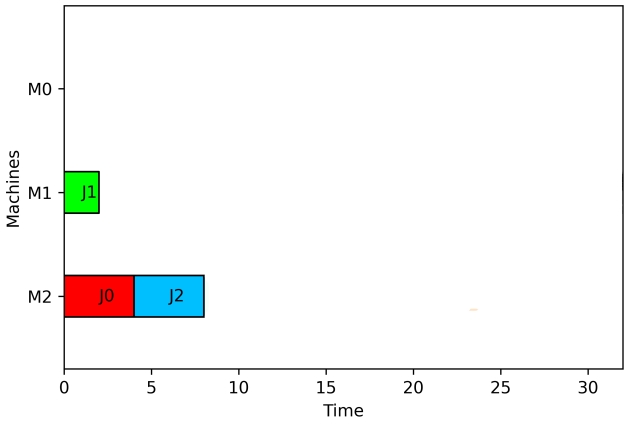

Figura 2.6 Schedule Operación 3 (Elaboración propia).

Una vez se ha colocado la operación 12, se pasa a la operación 14 siguiendo el mismo procedimiento explicado anteriormente. Tal y como muestra la Tabla 2.2 es la operación del trabajo 4 en la máquina 2. Por otro lado, la Tabla 2.1 indica la duración de esa operación, en este caso concreto, 1 unidad de tiempo. El Schedule actualizado queda detallado en la Figura 2.7.

$Sec = (6,10,12,\mathbf{14},7,4,2,9,11,1,3,13,8,0,5)$

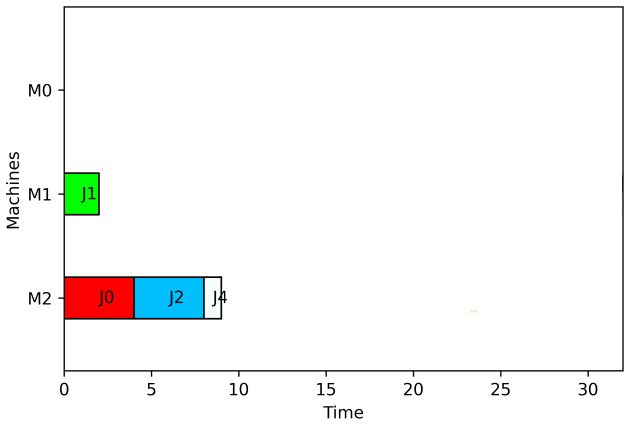

Figura 2.7 Schedule Operación 4 (Elaboración propia).

Llegados a este punto se debe colocar la operación 7, que corresponde al mismo trabajo que la operación 12 programada anteriormente pero, en vez de ser en la máquina 2, en la máquina 1. Es por ello, que se debe colocar en la máquina 1 pero no se puede iniciar dicha operación hasta que la operación 12 haya finalizado, es decir, hasta el instante 8 de tiempo. Por otro lado, tal y como se puede comprobar en la Tabla 2.1, la duración de esa operación es de 2 unidades de tiempo. El Schedule actualizado queda detallado en la Figura 2.8.

$Sec = (6,10,12,14,\mathbf{7},4,2,9,11,1,3,13,8,0,5)$

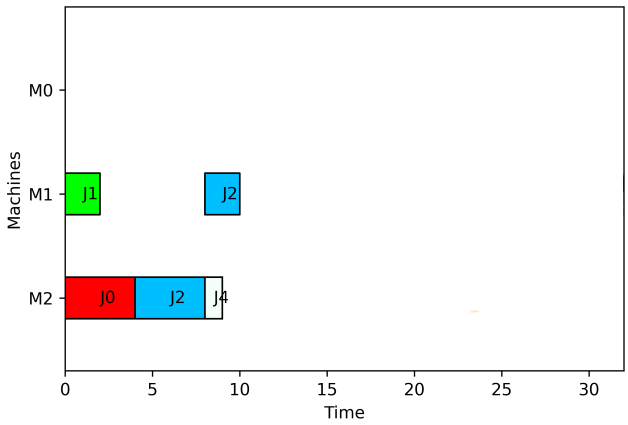

Figura 2.8 Schedule Operación 5 (Elaboración propia).

En este punto, se debe colocar la operación 4 que corresponde al mismo trabajo que la operación 14 programada anteriormente pero, en vez de ser en la máquina 2, en la máquina 0. Es por ello que se debe colocar en la máquina 0 pero no se puede iniciar dicha operación hasta que la operación 14 haya finalizado, es decir, hasta el instante 9 de tiempo. Por otro lado, tal y como se puede comprobar en la Tabla 2.1, la duración de esa operación es de 4 unidades de tiempo. El Schedule actualizado queda detallado en la Figura 2.9.

$Sec = (6,10,12,14,7,\mathbf{4},2,9,11,1,3,13,8,0,5)$

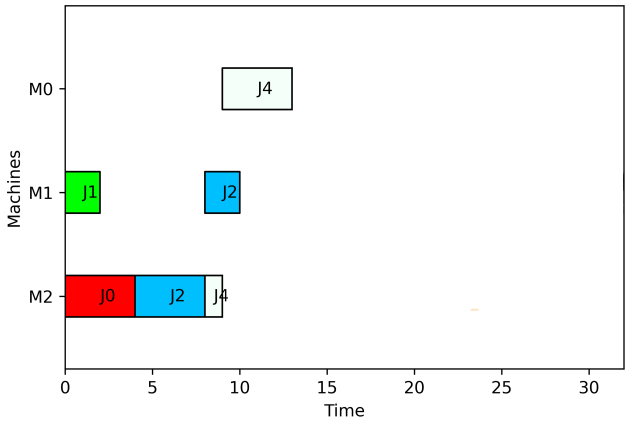

Figura 2.9 Schedule Operación 6 (Elaboración propia).

Siguiendo con el mismo procedimiento con el resto de operaciones se llegaría un Schedule como el que se muestra en la Figura 2.10.

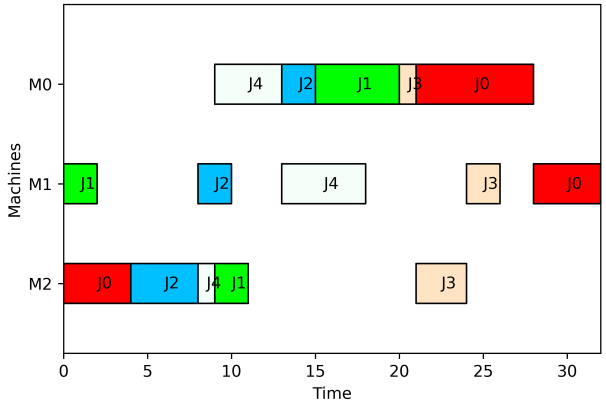

Figura 2.10 Schedule Completo (Elaboración propia).

Una vez obtenido el Schedule completo, se pueden estudiar ambos objetivos. Para el cálculo del Total Core Idle Time, resulta importante establecer claramente en que instante de tiempo empieza y finaliza cada operación. Esto se refleja en la Figura 2.11.

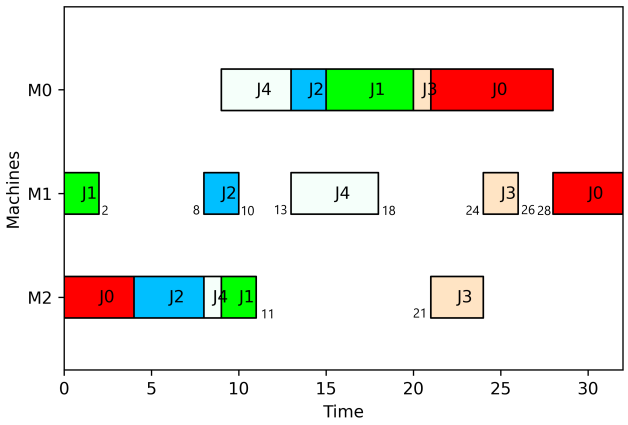

Figura 2.11 Schedule Completo con Detalle (Elaboración propia).

Tal y como se puede apreciar en la Figura 2.11 en la M0 el Core Idle Time es 0, ya que desde que se enciende la máquina hasta que se apaga se encuentra procesando operaciones. Por otro lado, en la M1 y M2, si que se encuentra parada durante algunos periodos de tiempo.

$TotalCoreIdleTime = 6(8-2)+3(13-10)+6(24-18)+2(28-26)+10(21-11) = 27$

Por otro lado, es fácil deducir que el Makespan es 32, ya que es el instante en el que finaliza la última operación.

Resulta interesante mencionar que en el proyecto, estos objetivos se han ponderado mediante λ para obtener una nueva función objetivo. Tal y como se muestra en la Tabla 2.3, esta variable toma los valores de 0, 0.25, 0.5, 0.75 y 1.

$$mo_{\lambda} = \lambda TCIT + (1-\lambda)C_{max} \tag{2.5}$$

Tabla 2.3 Ejemplo ponderación objetivos (Elaboración propia).

SECUENCIA	Core Idle Time	Makespan	$\lambda = 0$	0,25	0,5	0,75	1
[6,10,12,14,7,4,2,9,11,1,3,13,8,0,5]	26	32	32	30,5	29	27,5	26

3 Metodología

"Pregúntate si lo que estás haciendo hoy te acerca al lugar en el que quieres estar mañana"

WALT DISNEY

Una vez comprendida la naturaleza y complejidad del problema descrito con anterioridad, en este capítulo se exponen diferentes teoremas que permiten optimizar el tiempo de búsqueda, lo que reducirá el tiempo de computo. Por otro lado, también se presentan las diferentes heurísticas detalladas.

3.1 Teoremas de permutación

En este apartado se presentan los diferentes teoremas con un breve ejemplo de cada uno de ellos con el objetivo de facilitar su interpretación. Estos teoremas permiten eliminar el estudio de soluciones redundantes (Naderi et al., 2010). Siguiendo con el ejemplo detallado anteriormente, se toma como secuencia inicial: $Sec = (6,10,12,14,7,4,2,9,11,1,3,13,8,0,5)$. En la Figura 3.1 se muestra su Schedule.

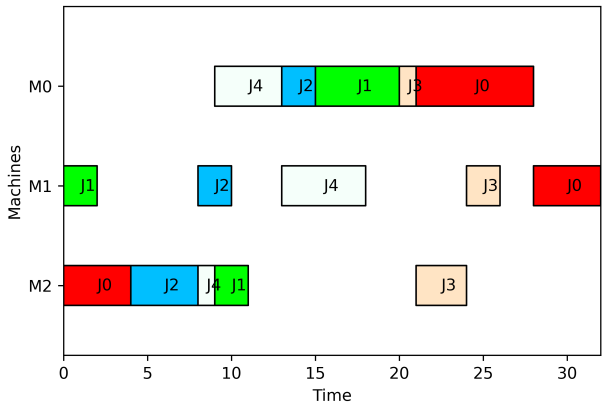

Figura 3.1 Schedule Principal (Elaboración propia).

Además, se presentan de nuevo en las Tablas 3.1 y 3.2 los tiempos de proceso y las etiquetas para comprender con mayor facilidad los ejemplos que se detallan a continuación.

Tabla 3.1 Tiempos de Proceso de las Operaciones Ejemplo Teoremas (Elaboración propia).

Pt_{ij}	0	1	2	3	4
0	7	5	2	1	4
1	4	2	2	2	5
2	4	2	4	3	1

Tabla 3.2 Etiquetas de las Operaciones Ejemplo Teoremas (Elaboración propia).

Et_{ij}	0	1	2	3	4
0	0	1	2	3	4
1	5	6	7	8	9
2	10	11	12	13	14

3.1.1 Teorema 1

Una permutación θ_2 producida por el intercambio de dos operaciones adyacentes de θ_1 representa una solución redundante (la misma solución) si y solo si las operaciones.

- No pertenecen al mismo trabajo.
- No pertenecen a la misma máquina.

Supongamos que se tiene una operación O_{ij} y otra O_{zd}. Tras el intercambio de estas operaciones, la solución será redundante en el caso en el que $i \neq z$ y $j \neq d$.

En la Figura 3.2 y 3.3 se muestran los Schedules resultantes tras aplicar este teorema al ejemplo propuesto anteriormente:

$Sec_1 = (\mathbf{6},\mathbf{10},12,14,7,4,2,9,11,1,3,13,8,0,5)$

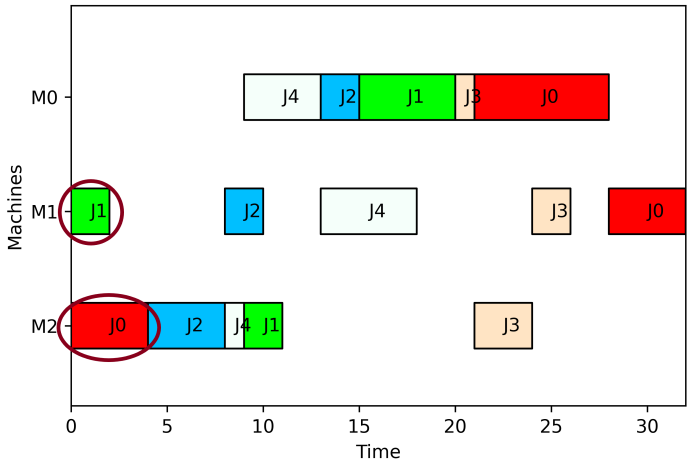

Figura 3.2 Schedule Teorema 1 - Caso 1 - Secuencia 1 (Elaboración propia).

$Sec_2 = (\mathbf{10},\mathbf{6},12,14,7,4,2,9,11,1,3,13,8,0,5)$

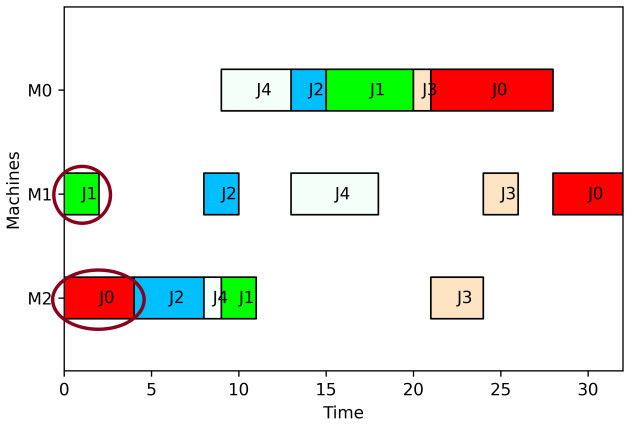

Figura 3.3 Schedule Teorema 1 - Caso 1 - Secuencia 2 (Elaboración propia).

Tal y como se puede visualizar en la Figura 3.3, puesto que la operación 6 y la 10 no pertenecen ni al mismo trabajo ni a al misma máquina, la solución de representar $Sec_1 = (\mathbf{6},\mathbf{10},12,14,7,4,2,9,11,1,3,13,8,0,5)$ y $Sec_2 = (\mathbf{10},\mathbf{6},12,14,7,4,2,9,11,1,3,13,8,0,5)$ es una solución redundante.

Por otro lado, se puede encontrar el caso en el que se tengan dos operaciones adyacentes que son del mismo trabajo pero de diferentes máquinas. En este caso tenemos O_{ij} y O_{id}. Si en primer lugar se realiza la operación O_{ij} se cumplen las siguientes condiciones:

- $atm_j = s_{ij} + Pt_{ij}$
- $at_j = s_{ij} + Pt_{ij}$

Esto quiere decir que el instante de tiempo en el cual la máquina y el trabajo volverán a estar disponible, será el instante mas temprano de inicio mas el tiempo de proceso de esa operación. Por otro lado, para representar O_{id} debemos tener en cuenta $s_{id} = max(s_{ij} + Pt_{ij}, atm_d)$ ya que el instante más temprano de inicio será el máximo entre el tiempo del proceso del trabajo i en la otra máquina mas el tiempo de proceso o la disponibilidad de la máquina en la que debemos programar el trabajo. Por último, la operación acabará $f_{id} = s_{id} + Pt_{id}$. Si por el contrario se realiza en primer lugar la operación O_{id}, el Schedule resultante será diferente. En las Figuras 3.4 y 3.5 se presentan ambos casos.

$Sec_1 = (6,10,12,14,7,4,2,9,11,1,3,13,8,\mathbf{0},\mathbf{5})$

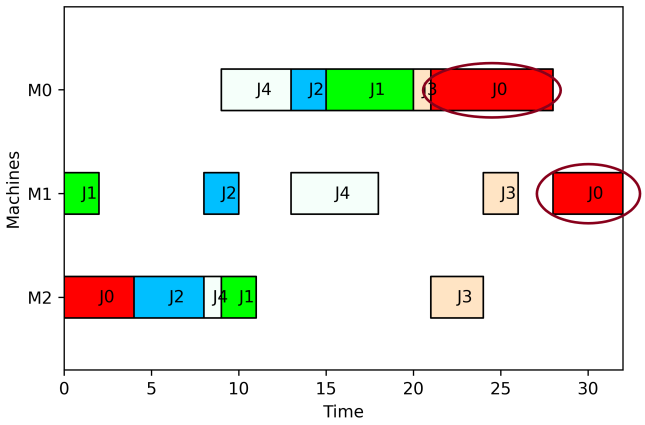

Figura 3.4 Schedule Teorema 1 - Caso 2 - Secuencia 1 (Elaboración propia).

$Sec_2 = (6,10,12,14,7,4,2,9,11,1,3,13,8,\mathbf{5},\mathbf{0})$

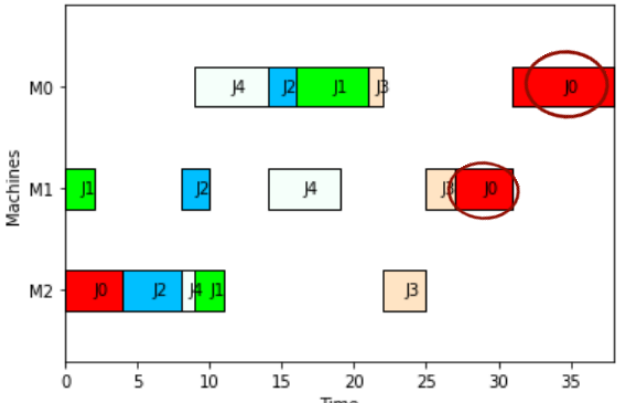

Figura 3.5 Schedule Teorema 1 - Caso 2 - Secuencia 2 (Elaboración propia).

Además, se acaba de demostrar que en este caso, (cuando se encuentran dos operaciones adyacentes del mismo trabajo en diferentes máquinas) se obtiene una nueva solución.

3.1.2 Teorema 2

Una permutación θ_2 producida por el intercambio de dos operaciones no adyacentes de θ_1 representa una solución redundante (la misma solución) si y solo si:

- Las operaciones no pertenecen al mismo trabajo.
- Las operaciones no pertenecen a la misma máquina.
- Las operaciones situadas entre estas dos operaciones no pertenecen a los trabajos i y z, ni se van a procesar en las máquinas j y d.

En las Figuras 3.6 y 3.7 se detallan los Schedules resultantes tras aplicar este teorema al ejemplo propuesto con anterioridad.

$Sec = (6,10,12,14,7,4,\mathbf{2},9,\mathbf{11},1,3,13,8,0,5)$

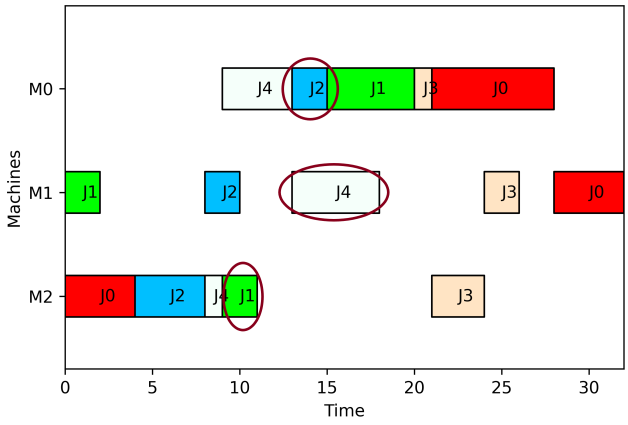

Figura 3.6 Schedule Teorema 2 - Secuencia 1 (Elaboración propia).

$Sec = (6,10,12,14,7,4,\mathbf{11},9,\mathbf{2},1,3,13,8,0,5)$

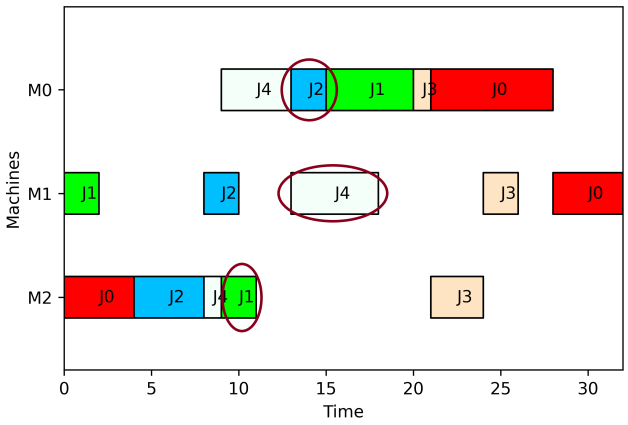

Figura 3.7 Schedule Teorema 2 - Secuencia 2 (Elaboración propia).

Tal y como se ha podido comprobar se ha obtenido una solución redundante.

3.1.3 Teorema 3

Una permutación producida por el intercambio de tres o más operaciones adyacentes representa una solución redundante (la misma solución) si y solo si las operaciones:

- No pertenecen al mismo trabajo.
- No pertenecen a la misma máquina.

3.1.4 Teorema 4

Una permutación producida por el intercambio de r operaciones no adyacentes de θ_1 representa una solución redundante (la misma solución) si y solo si:

- Las r operaciones que van a ser procesadas pertenecen a diferentes trabajos.
- Las operaciones intermedias no pertenecen a los mismos trabajos a los que pertenecen las r operaciones, ni a las mismas máquinas en las que las r operaciones se van a procesar.

  NOTA: Este teorema tiene la misma finalidad que el teorema 2 pero de forma generalizada.

3.2 Heurísticas

Una vez se conocen los diferentes teoremas que permiten eliminar el principal inconveniente de las listas de permutación que es la redundancia se puede pasar a aplicarlos en diferentes heurísticas. Para ello, se programarán cuatro heurísticas basadas principalmente en operadores de inserción y reinserción junto con los objetivos de sostenibilidad. Todas ellas son las que plantea (Naderi et al., 2010) pero teniendo en cuenta no solo el Makespan, si no también el Total Core Idle Time. Las modificaciones realizadas del algoritmo se muestran recuadradas en color rojo en las Figuras 3.8, 3.9, 3.10 y 3.11.

Las heurísticas se ejecutan para tres casos diferentes, en los que se calculan diferentes funciones objetivos para cada uno de los pasos en los que se comparan soluciones tanto parciales como completas. Estos casos son:

- Caso 1: La función objetivo es $TCIT$ (ver Ecuación 2.2).
- Caso 2: La función objetivo es C_{max} (ver Ecuación 2.3).
- Caso 3: La función objetivo es multiobjetivo, con enfoque de función lineal convexa de los dos objetivos anteriores ponderados mediante un parámetro λ (ver Ecuación 2.4).

En los tres casos, la solución proporcionada por la heurística es evaluada para el objetivo correspondiente y para los otros objetivos.

3.2.1 Heurística 1

```
Procedure IRH1
    E = Sort p_ij in non-increasing order
    W = ∅                                   % the permutation of scheduled operations
    for r = 1 to (n · m) do
        i = job the r-th operation of E belong to
        j = machine r-th operation of E is to be processed on
        for rr = 1 to r do
            redundancy = no
            if rr > 1 and job i ≠ (rr-1)-th job in W and machine j ≠ (rr-1)-th machine in W then
                redundancy = yes
            endif
            if redundancy = no then
                Test operation ij at rr-th position in W
            endif
        endfor
        Insert operation ij in W at the position resulting depending on the objective function
    endfor
```

Figura 3.8 Pseudocódigo de Heurística 1 (Adaptado de (Naderi et al., 2010)).

En esta primera heurística, cuyo pseudocódigo se halla en la Figura 3.8, se tiene un vector E ordenado en función de los tiempos de proceso de las operaciones de la instancia estudiada y un vector W que inicialmente es un conjunto vacío y será sobre el cual se vaya definiendo la secuencia. En el pseudocódigo mostrado en la figura anterior se puede observar la aplicación de lo estudiado previamente en los teoremas, es decir, que una permutación producida por el intercambio de dos operaciones no adyacentes representa una solución redundante (la misma solución) si y solo si:

- Las operaciones no pertenecen al mismo trabajo.
- Las operaciones no pertenecen a la misma máquina.
- Las operaciones situadas entre estas dos operaciones no pertenecen a los trabajos estudiados, ni se van a procesar en las máquinas de los trabajos estudiados.

Esto permite eliminar el estudio de todas aquellas soluciones cuyos valores se sabe que son redundantes y por consiguiente, reducir el tiempo de ejecución. En caso de que al introducir dicha operación que se está estudiando en el vector W, esta no sea redundante, se estudiará el valor de la función objetivo. Finalmente se introducirá en la posición que mejor valor de la función objetivo haya proporcionado.

3.2.2 Heurística 2

```
Procedure decoding scheme
    S = ∅
    U =all operations in given permutation θ
    while U ≠ ∅
        y = min { s_ij of O_ij | O_ij ∈ U}
        R = { O_ij | s_ij = y, O_ij ∈ U}  %R is a set of operations whose starting times are equal to y
        Choose O* from the set of R with the earliest relative position in permutation θ
        Extract O* from U
        Put O* into S
    endwhile
```

Figura 3.9 Pseudocódigo de Heurística 2 (Naderi et al., 2010).

La segunda heurística es una adaptación de la Heurística 1 cuyo pseudocódigo se halla en la Figura 3.9. En esta se tiene un vector S que inicialmente es un conjunto vacío y será sobre el cual se vaya definiendo la secuencia. Por otro lado se encuentra el vector U que inicialmente está formado por el vector W resultante de la primera heurística. Además, también se incluye un vector R que se actualizará en cada iteración y estará formado por todas aquellas operaciones que tienen el menor tiempo de inicio. Comprendido esto, se van a ir sacando operaciones del vector U y se van a ir introduciendo en el vector S. Esa operación que se cambia de un vector al otro debe formar parte del vector R, y en caso de que haya más de una operación en dicho vector, se cogerá aquella con la menor posición relativa. Además, al tratarse de un non-delay Schedule, ninguna máquina debe estar libre cuando alguna operación se pueda procesar. Por consiguiente, cuando se produce una mejora en el Schedule mediante la realización de modificaciones directamente en el mismo para obtener el nuevo valor de la FO, ya no se tiene ninguna secuencia del Schedule modificado puesto que, la codificación permite a partir de la secuencia obtener el Schedule, pero a partir del Schedule no podemos obtener la secuencia.

3.2.3 Heurística 3

```
Procedure IRH 3
    W = the permutation produced by IRH 2
    Make = objective function of W
    r = n·m
    improvement = yes
    while r > 0 do
        if improvement = no then
            r = r – 1
        endif
        improvement = no
        i = job at rr-th position in W
        j = machine at rr-th position in W
        for r2 = 1 to (n·m) do
            redundancy = no
            if r2 > 1 and job i ≠ (r2-1)-th job in W and machine j ≠ (r2-1)-th machine in W then
                redundancy = yes
            endif
            if redundancy = no then
                WW = produced by inserting job i at r2-th position in W
                If objective function of WW < Make then
                    Improvement = yes
                    Make = objective function of WW
                    r3 = r2
                endif
            endif
        endfor
        if improvement = yes then
            Shift the position of O_ij to r3-th position in W
            r = n·m
        endif
    endwhile
```

Figura 3.10 Pseudocódigo de Heurística 3 (Adaptado de (Naderi et al., 2010)).

La tercera heurística cuyo pseudocódigo se halla en la Figura 3.10, parte inicialmente de la secuencia resultante de la Heurística 2 y almacena en Make su valor de la función objetivo. El bucle se ejecutará r veces, siendo r el número de operaciones que contiene el vector W. En caso de que al ejecutarse una iteración, no se mejore

el valor de Make, r disminuye y cuando mejore, r vuelve a tomar el valor del número de operaciones. Durante la ejecución, al igual que se ha mencionado en la primera heurística, se tienen en cuenta los teoremas. En el desarrollo lo que hace es introducir las operaciones empezando por el final en todas las opciones posibles. Finalmente se introducirá en la posición que mejor valor de la función objetivo haya proporcionado.

3.2.4 Heurística 4

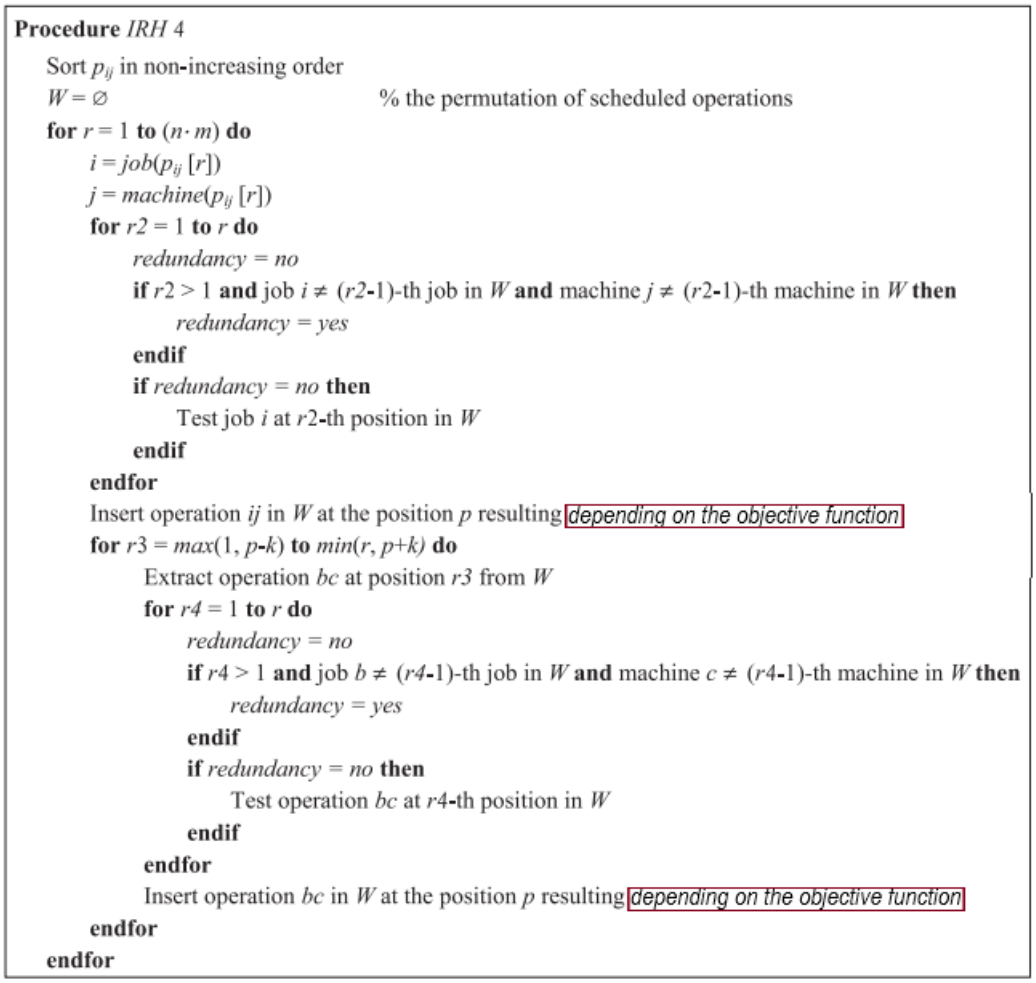

Figura 3.11 Pseudocódigo de Heurística 4 (Adaptado de (Naderi et al., 2010)).

La cuarta y última heurística se halla en la Figura 3.11 y se plantea un desarrollo muy similar al de la primera heurística, aunque esta tiene la peculiaridad de que depende de una constante K. Además, en el momento en que inserta esa operación en la mejor posición, se vuelve a estudiar llevando a cabo un proceso similar.

3.2.5 Ejemplo Schedules

En este apartado se presentan algunos ejemplos de los Schedules resultantes en función del caso. En las Figuras 3.12, 3.13 y 3.14 se presenta un ejemplo de la Heurística 3 en los Casos 1, 2 y 3 respectivamente. En las Figuras 3.15, 3.16 y 3.17 se presenta un ejemplo de la Heurística 1 en los Casos 1, 2 y 3. Se puede apreciar como las Figuras 3.12 y 3.15 correspondientes al Caso 1 que tienen como función objetivo reducir el Core Idle Time, producen como consecuencia que el valor del Makespan se dispare. Por otro lado, en las Figuras 3.13 y 3.16 correspondientes al Caso 2, que se centran en reducir el Makespan, el Core Idle Time aumenta considerablemente. Para facilitar la interpretación se muestran la Tablas 3.3 y 3.4 en la que se puede apreciar las diferentes funciones objetivos obtenidas en los Casos 1 y 2. Finalmente, las Figuras 3.14 y 3.17 correspondientes al Caso 3, representan una función objetivo ponderada y por ello se consiguen valores intermedios. Mencionar que en las Figuras se hace referencia al Caso como "C" y a la Heurística como "H".

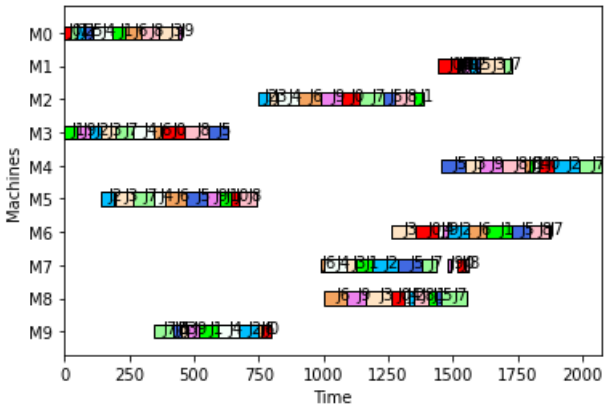

Figura 3.12 Schedule Instancia 10x10 C1H3 (Elaboración propia).

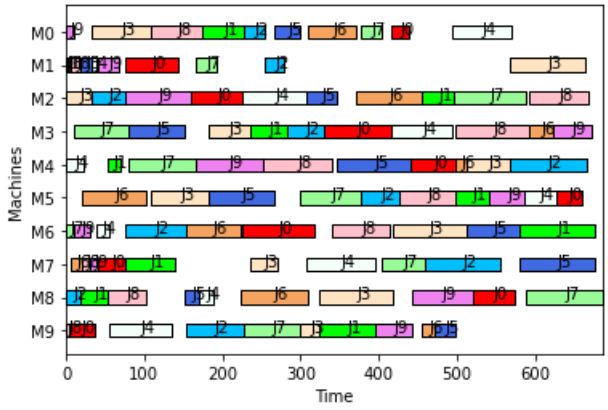

Figura 3.13 Schedule Instancia 10x10 C2H3 (Elaboración propia).

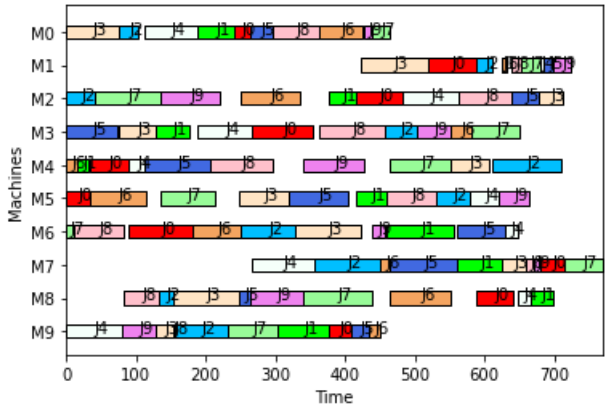

Figura 3.14 Schedule Instancia 10x10 C3H3 (Elaboración propia).

Tabla 3.3 Resumen Funciones Objetivos Schedule 10x10 C3H3.

	Total Core Idle Time	Makespan
CASO 1	71	2074
CASO 2	1064	687

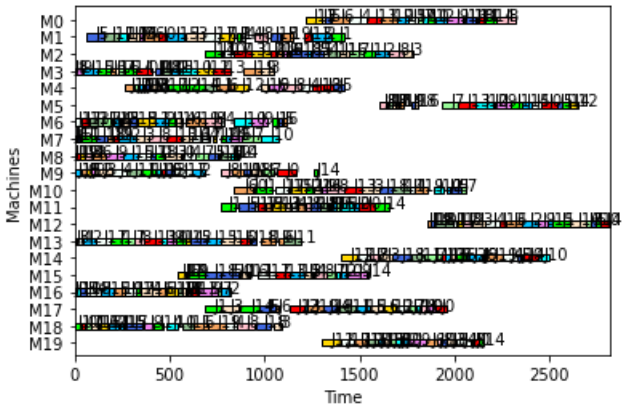

Figura 3.15 Schedule Instancia 20x20 C1H1 (Elaboración propia).

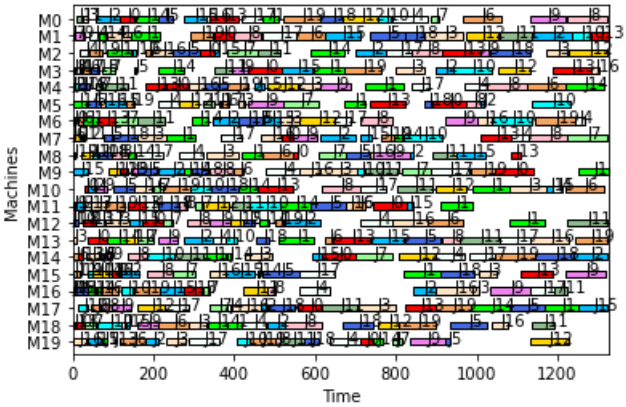

Figura 3.16 Schedule Instancia 20x20 C2H1 (Elaboración propia).

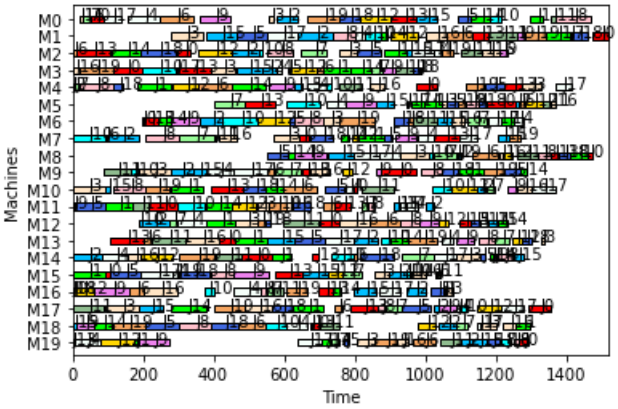

Figura 3.17 Schedule Instancia 20x20 C3H1 (Elaboración propia).

Tabla 3.4 Resumen Funciones Objetivos Schedule 20x20 C3H1.

	Total Core Idle Time	**Makespan**
CASO 1	1221	2822
CASO 2	4991	1328

4 Experimentación

En este punto se tratarán aspectos relacionados con el lenguaje que se ha decidido programar, la generación de las instancias del problema y se llevará a cabo el análisis de los resultados.

4.1 Lenguaje de programación eligido: Python

Python es un lenguaje de programación de alto nivel en el que se utiliza para desarrollar aplicaciones de todo tipo. A diferencia de otros lenguajes, se trata de un lenguaje interpretado, es decir, que no es necesario compilarlo para ejecutar las aplicaciones escritas en python, sino que se ejecutan directamente por el ordenador utilizando un programa denominado interpretador, por lo que no es necesario "traducirlo" a lenguaje máquina. Con el paso del tiempo, python ha ido ganando peso gracias a su sencillez y a sus amplias posibilidades, sobre todo en los últimos años, ya que facilita trabajar con inteligencia artificial, big data, machine learning y data science, entre muchos otros campos en auge. Además, se está probando una librería desarrollada por los profesores de la asignatura PCP para programar la producción con Python. Esta librería contiene funciones de lectura de instancias, cálculos de funciones objetivos, y representación de las soluciones gráficamente en diagramas de Gantt. Estos argumentos, junto con la motivación de adentrarme en un nuevo lenguaje de programación han sido los motivos de llevar a cabo en python dicho proyecto.

4.2 Generación de las instancias del problema

Para la experimentación, siguiendo el artículo de Naderi et al. (2010) se han utilizado 60 instancias cuadradas propuestas en (Taillard, 1993) de diferentes tamaños cuyos tiempos de proceso se encuentran generados según una distribución Uniforme (1,99). Concretamente, 10 instancias de cada una de las siguientes dimensiones: 4x4, 5x5, 7x7. 10x10, 15x15 y 20x20.

Para usar la librería Schedule para la lectura de datos, se ha realizado una función que ponga las instancias en el formato que lee la librería. El código referente a las instancias cuadradas se encuentra en el Anexo 3, mientras que el código referente a las instancias restantes se encuentra en el Anexo 4. A continuación se muestra el formato que tenían las instancias inicialmente y con el que resultan al final. Link Instancias

```
number of jobs,number of machines
          4          4
processing times :
 54 34 61 2
  9 15 89 70
 38 19 28 87
 95 34  7 29
machines :
  3  1  4  2
  4  1  2  3
  1  2  3  4
  1  3  2  4
```

Figura 4.1 Formato Inicial Instancias Cuadradas (Elaboración propia).

```
[JOBS=4]

[MACHINES=4]

[PT=34,2,54,61;
15,89,70,9;
38,19,28,87;
95,7,34,29]
```

Figura 4.2 Formato Final Instancias Cuadradas (Elaboración propia).

```
       5
       20
       54        79        16        66        58
       83         3        89        58        56
       15        11        49        31        20
       71        99        15        68        85
       77        56        89        78        53
       36        70        45        91        35
       53        99        60        13        53
       38        60        23        59        41
       27         5        57        49        69
       87        56        64        85        13
       76         3         7        85        86
       91        61         1         9        72
       14        73        63        39         8
       29        75        41        41        49
       12        47        63        56        47
       77        14        47        40        87
       32        21        26        54        58
       87        86        75        77        18
       68         5        77        51        68
       94        77        40        31        28
```

Figura 4.3 Formato Inicial Instancias mxn (Elaboración propia).

```
[JOBS=20]

[MACHINES=5]

[PT=54,83,15,71,77,36,53,38,27,87,76,91,14,29,12,77,32,87,68,94;
79,3,11,99,56,70,99,60,5,56,3,61,73,75,47,14,21,86,5,77;
16,89,49,15,89,45,60,23,57,64,7,1,63,41,63,47,26,75,77,40;
66,58,31,68,78,91,13,59,49,85,85,9,39,41,56,40,54,77,51,31;
58,56,20,85,53,35,53,41,69,13,86,72,8,49,47,87,58,18,68,28]
```

Figura 4.4 Formato Final Instancias mxn (Elaboración propia).

4.3 Medida de comparación

Tal y como se ha mencionado anteriormente, la última heurística estudiada depende de una constante K. Por ello, previo al desarrollo de los diferentes casos propuestos, conviene conocer que valor de K proporciona mejores resultados. Para ello, se ha aplicado el indicador RDI (Relative Deviation Index).

$$RDI_{inst} = \frac{TCIT_{inst} - minTCIT}{maxTCIT - minTCIT} \tag{4.1}$$

Siendo $TCIT_{inst}$ el Total Core Idle Time obtenido para la instancia inst, y $minTCIT/maxTCIT$ el mínimo/-máximo Total Core Idle Time obtenido por todos los métodos para la instancia inst.

Esta expresión muestra valores comprendidos entre [0,1]. Cuando el $CoreIdleTime$ y el $CoreIdleTime_{MAX}$ coinciden, es porque el Core Idle Time es máximo y el valor resultante es 1. Cuando el $CoreIdleTime$ y el $CoreIdleTime_{MIN}$ coinciden es porque el Core Idle Time es mínimo y el valor resultante es 0.

4.4 Calibración Heurística 4

Tal y como se ha mencionado anteriormente, la última heurística estudiada depende de una constante K. Por ello, previo al desarrollo de los diferentes casos propuestos, conviene conocer que valor de K proporciona mejores resultados. Para ello, se ha aplicado el indicador RDI (Relative Deviation Index) para los valores de K=3, 4 y 5. A continuación, se detallan los resultados de calibración de los métodos en función del tamaño de las instancias. En concreto, para el parámetro k de la cuarta heurística. A continuación, se presenta la Tabla 4.1 en la cual se realiza un análisis para los valores de K=3, K=4 y K=5.

Tabla 4.1 Resumen Valores de K (Elaboración propia).

	RDI		
Tamaño instancias	**K=3**	**K=4**	**K=5**
4x4	0,438	0,340	0,608
5x5	0,545	0,207	0,521
7x7	0,465	0,563	0,422
10x10	0,730	0,490	0,288
15x15	0,559	0,504	0,445
20x20	0,611	0,405	0,499
Resultados promedios	0,558	0,418	0,464

Tras analizar los resultados obtenidos que se presentan en la Tabla 4.1 es fácil comprobar que, a pesar de que para el caso de las instancias con tamaño 7x7, 10x10 y 15x15 es mejor el valor de K=5, en promedio, los mejores resultados que se han obtenido para todas las instancias resultan de un valor de K=4. Como consecuencia de esto, en el estudio realizado para la Heurística 4 se ha tomado como valor de la constante K=4.

4.5 Comparación Heurísticas

En este apartado se representarán tanto las tablas con los resultados del valor de la función objetivo, como los resultados obtenidos tras aplicar el indicador Relative Desviation Index (ver Ecuación 4.1). Además, se compararán las cuatro heurísticas de dos formas diferentes según muestran en las Figuras 4.5 y 4.6 para realizar un análisis más exhaustivo.

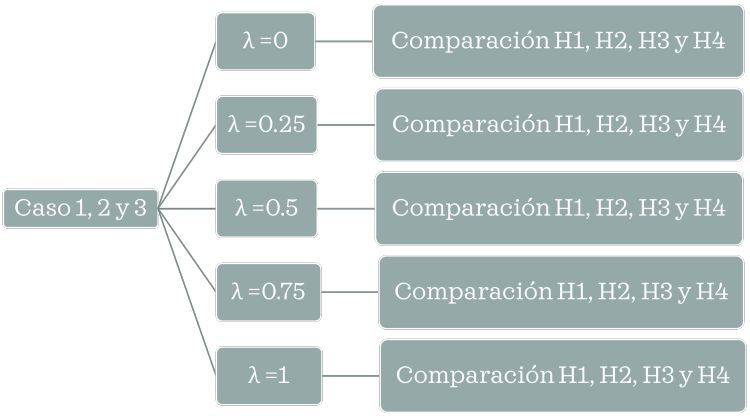

Figura 4.5 Esquema estructura experimentación (Elaboración propia).

Figura 4.6 Esquema estructura experimentación general (Elaboración propia).

Tal y como se puede apreciar en la Tablas 4.2, 4.3, 4.4, 4.5, 4.6 y 4.7, la Heurística 1 y la Heurística 2 proporcionan siempre los mismos resultados. Esto es así dado que, al tratar de introducir una operación cualquiera en un CIT, se tienen que cumplir muchas restricciones. Entre ellas, que el hueco en el cual se va a insertar sea mayor que el tiempo de proceso de la operación y que además, el hueco disponible restándole el solapamiento que se pueda producir por una operación de ese mismo trabajo en otras máquinas sea aun superior que el tiempo de proceso de la operación que se trata de insertar. Por otro lado, para que el resultado se vea modificado, puesto que se está trabajando con el Schedule, las operaciones que se inserten deben situarse al final de las máquinas.

A continuación, se recuerda que las heurísticas se ejecutan para tres casos diferentes, en los que se imponen diferentes funciones objetivos.

- Caso 1: La función objetivo es $TCIT$ (ver Ecuación 2.2).
- Caso 2: La función objetivo es C_{max} (ver Ecuación 2.3).
- Caso 3: La función objetivo es multiobjetivo, con enfoque de función lineal convexa de los dos objetivos anteriores ponderados mediante un parámetro λ (ver Ecuación 2.4).

4.5.1 Caso 1: Minimización del Total Core Idle Time

En este apartado, las heurísticas están minimizando el Total Core Idle Time pero se van a evaluar las soluciones obtenidas para todos los objetivos ajustando con el λ la ponderación tal y como se presenta en la siguiente ecuación:

$$mo_{\lambda} = \lambda TCIT + (1-\lambda)C_{max}$$

En la Tabla 4.2 se muestran los resultados correspondientes a función objetivo. Posteriormente se detalla la representación de las gráficas en las Figuras 4.7, 4.8, 4.9, 4.10 y 4.11.

Tabla 4.2 Resumen Resultados Caso 1: TCIT (Elaboración propia).

Tamaño instancias	H1	H2	H3	H4
		$\lambda = 0$		
4x4	498,30	498,30	537,00	443,80
5x5	751,80	751,80	735,70	629,70
7x7	1200,20	1200,20	1112,20	879,00
10x10	1392,50	1392,50	1409,70	1204,30
15x15	2083,80	2083,80	1953,10	1583,70
20x20	2495,70	2495,70	2475,40	2022,78*
Promedio	1403,72	1403,72	1370,52	948,10
		$\lambda = 0{,}25$		
4x4	391,78	391,78	403,63	342,05
5x5	599,30	599,30	555,38	486,10
7x7	962,48	962,48	837,28	701,18
10x10	1100,85	1100,85	1072,03	964,55
15x15	1844,20	1844,20	1619,60	1435,90
20x20	2376,63	2376,63	2340,48	2107,56
Promedio	1212,54	1212,54	1138,06	785,96
		$\lambda = 0{,}5$		
4x4	285,25	285,25	270,25	240,30
5x5	446,80	446,80	375,05	342,50
7x7	724,75	724,75	562,35	523,35
10x10	809,20	809,20	734,35	724,80
15x15	1604,60	1604,60	1286,10	1288,10
20x20	2257,55	2257,55	2205,55	2192,33*
Promedio	1021,36	1021,36	905,61	623,81
		$\lambda = 0{,}75$		
4x4	178,73	178,73	136,88	138,55
5x5	294,30	294,30	194,73	198,90
7x7	487,03	487,03	287,43	345,53
10x10	517,55	517,55	396,68	485,05
15x15	1365,00	1365,00	952,60	1140,30
20x20	2138,48	2138,48	2070,63	2277,11*
Promedio	830,18	830,18	673,15	461,67
		$\lambda = 1$		
4x4	72,20	72,20	3,50	36,80
5x5	141,80	141,80	14,40	55,30
7x7	249,30	249,30	12,50	167,70
10x10	225,90	225,90	59,00	245,30
15x15	1125,40	1125,40	619,10	992,50
20x20	2019,40	2019,40	1935,70	2361,89*
Promedio	639,00	639,00	440,70	643,25

* Para este promedio no han sido utilizadas todas las instancias por superar en alguna de ellas el tiempo límite establecido

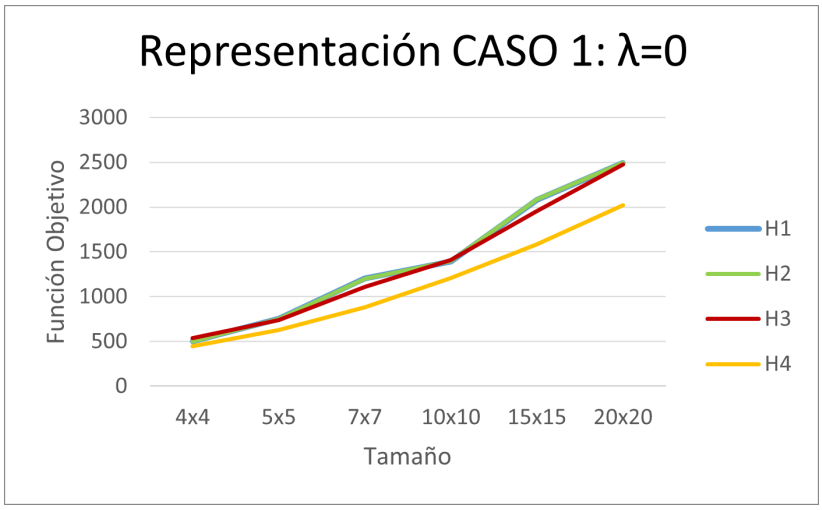

Figura 4.7 Representación Caso 1 $\lambda = 0$ (Elaboración propia).

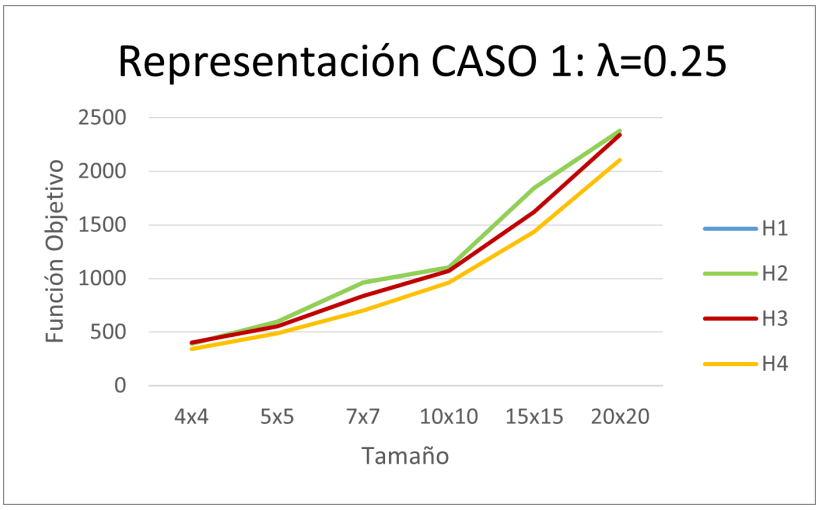

Figura 4.8 Representación Caso 1 $\lambda = 0{,}25$ (Elaboración propia).

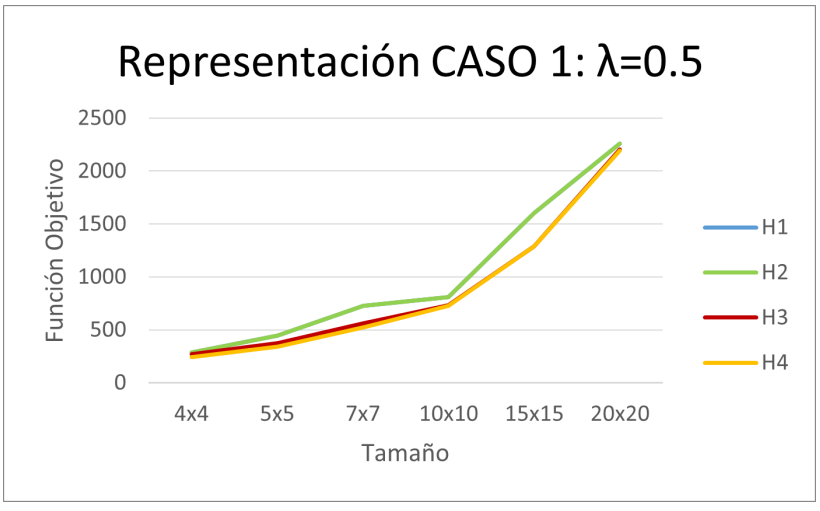

Figura 4.9 Representación Caso 1 $\lambda = 0{,}5$ (Elaboración propia).

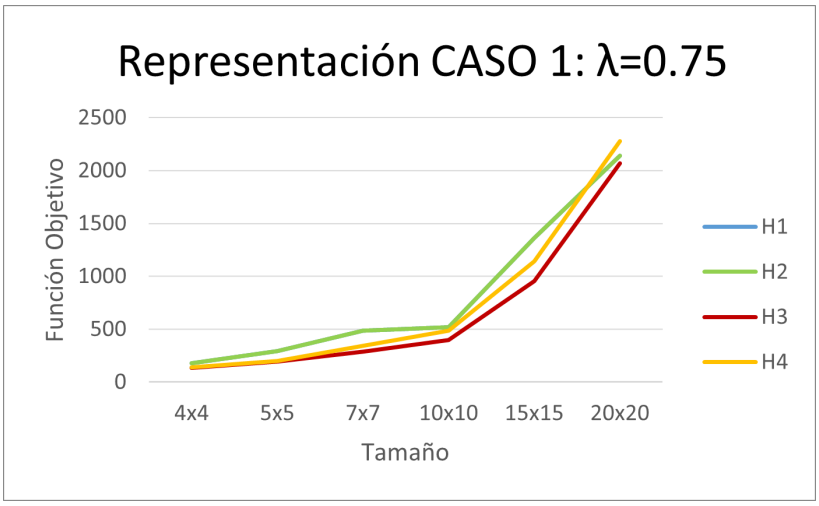

Figura 4.10 Representación Caso 1 $\lambda = 0{,}75$ (Elaboración propia).

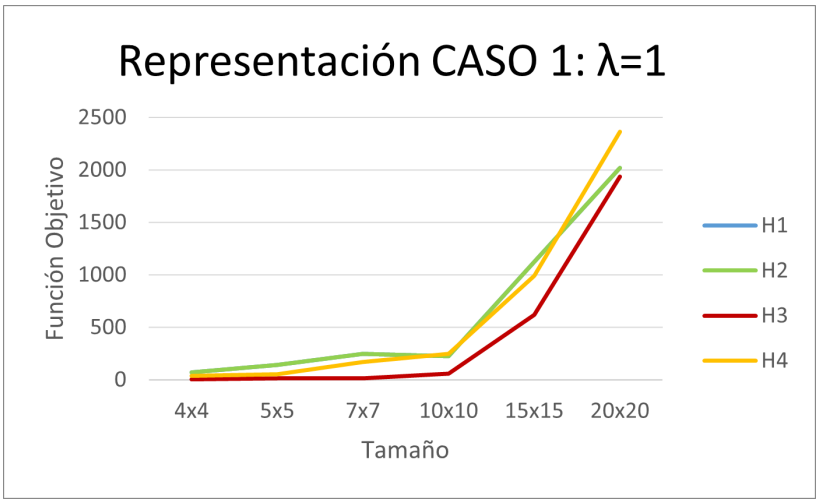

Figura 4.11 Representación Caso 1 $\lambda = 1$ (Elaboración propia).

Como conclusión es fácil afirmar que en el Caso 1, es decir, en el que durante la ejecución la operación se inserta en función del Total Core Idle Time, para valores de λ bajos ($\lambda = 0$ o $\lambda = 0{,}25$) la Heurística 4 proporciona mejores resultados que el resto. Por otro lado, conforme aumenta el valor el λ y se le va dando más peso al Core Idle Time, la Heurística 3 es la que presenta mejores valores.

4.5.2 Caso 1: Minimización del Total Core Idle Time (RDI)

En la Tabla 4.3 se muestran los resultados correspondientes a función objetivo. Posteriormente se detalla la representación de las gráficas en las Figuras 4.12, 4.13, 4.14, 4.15 y 4.16.

Tabla 4.3 Resumen Resultados RDI Caso 1: TCIT (Elaboración propia).

Tamaño instancias	H1	H2	H3	H4
	$\lambda = 0$			
4x4	0,56	0,56	0,67	0,31
5x5	0,68	0,68	0,58	0,31
7x7	0,88	0,88	0,65	0,13
10x10	0,68	0,68	0,74	0,24
15x15	0,89	0,89	0,89	0,12
20x20	0,85	0,85	0,78	0,20*
Promedio	0,58	0,76	0,72	0,22
	$\lambda = 0{,}25$			
4x4	0,66	0,66	0,61	0,27
5x5	0,73	0,73	0,55	0,31
7x7	0,89	0,89	0,58	0,14
10x10	0,76	0,76	0,58	0,31
15x15	0,90	0,90	0,37	0,18
20x20	0,74	0,74	0,61	0,30
Promedio	0,78	0,78	0,55	0,25
	$\lambda = 0{,}5$			
4x4	0,68	0,68	0,44	0,23
5x5	0,79	0,79	0,51	0,28
7x7	0,90	0,90	0,52	0,16
10x10	0,83	0,83	0,34	0,41
15x15	0,86	0,86	0,22	0,34
20x20	0,59	0,59	0,37	0,50*
Promedio	0,77	0,77	0,60	0,32
	$\lambda = 0{,}75$			
4x4	0,76	0,76	0,22	0,45
5x5	0,72	0,72	0,39	0,32
7x7	0,61	0,61	0,13	0,49
10x10	0,75	0,75	0,08	0,57
15x15	0,80	0,80	0,08	0,61
20x20	0,47	0,47	0,36	0,60*
Promedio	0,69	0,69	0,21	0,51
	$\lambda = 1$			
4x4	0,78	0,78	0,03	0,43
5x5	0,64	0,64	0,10	0,49
7x7	0,55	0,55	0,00	0,64
10x10	0,69	0,69	0,05	0,74
15x15	0,72	0,72	0,08	0,69
20x20	0,45	0,45	0,35	0,60*
Promedio	0,64	0,64	0,10	0,60

* Para este promedio no han sido utilizadas todas las instancias por superar en alguna de ellas el tiempo límite establecido

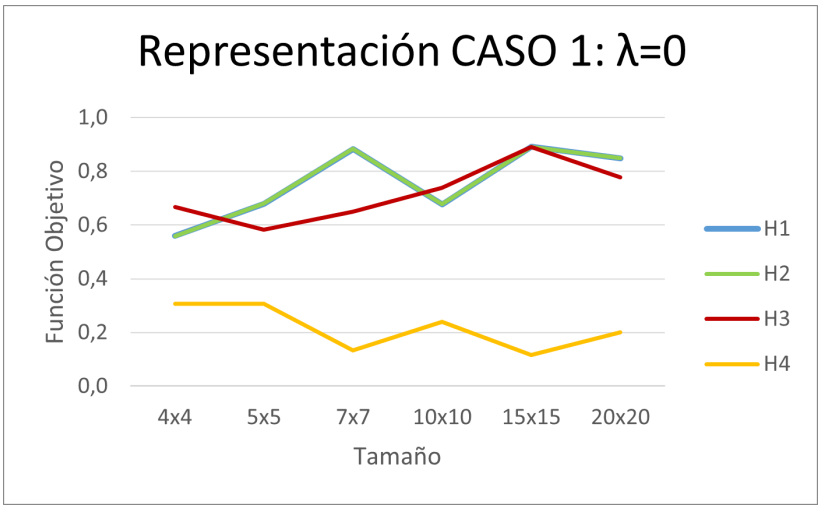

Figura 4.12 Representación RDI Caso 1 $\lambda = 0$ (Elaboración propia).

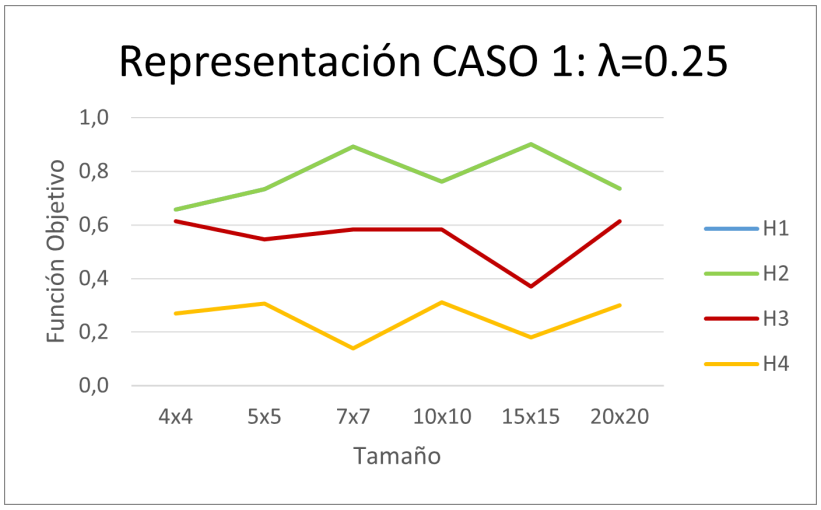

Figura 4.13 Representación RDI Caso 1 $\lambda = 0{,}25$ (Elaboración propia).

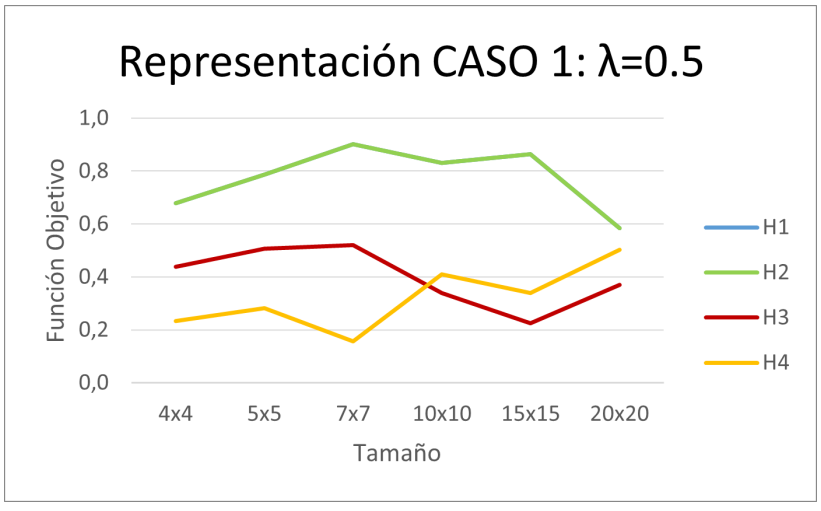

Figura 4.14 Representación RDI Caso 1 $\lambda = 0{,}5$ (Elaboración propia).

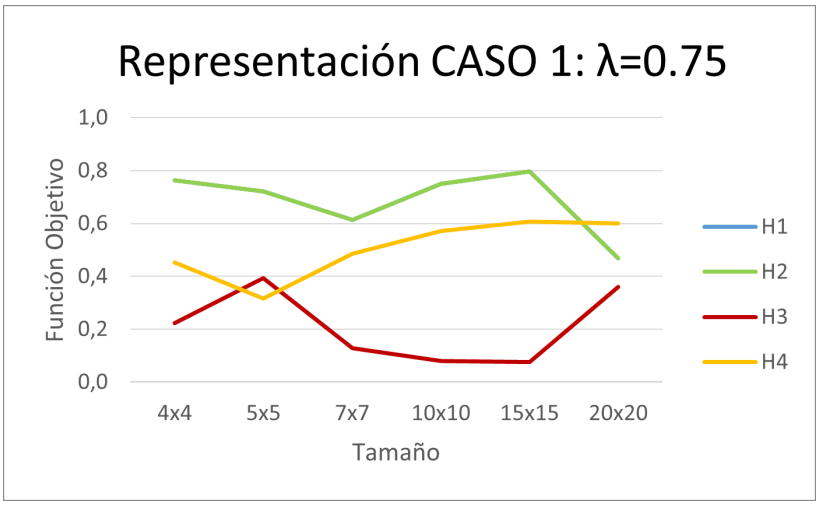

Figura 4.15 Representación RDI Caso 1 $\lambda = 0{,}75$ (Elaboración propia).

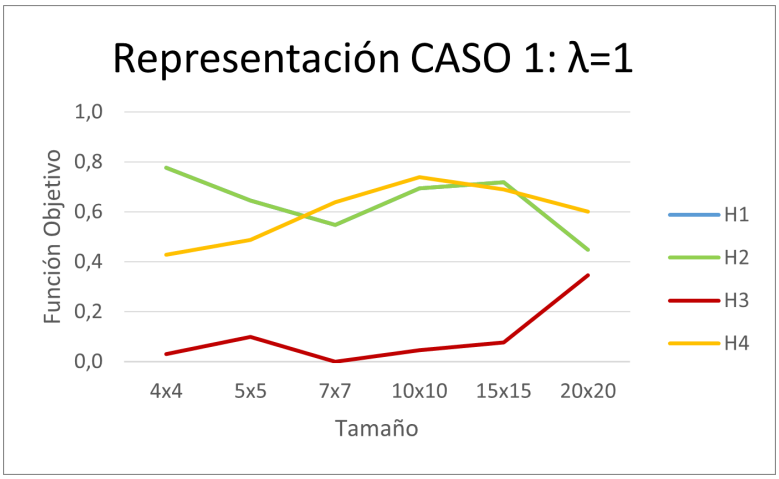

Figura 4.16 Representación RDI Caso 1 $\lambda = 1$ (Elaboración propia).

En este caso, sucede algo similar al caso estudiado anteriormente ya que para valores de λ pequeños ($\lambda = 0$ o $\lambda = 0{,}25$) la Heurística 4 proporciona mejores resultados que el resto. Por otro lado, conforme aumenta el valor el λ y se le va dando más peso al Core Idle Time, la Heurística 3 es la que presenta mejores valores.

4.5.3 Caso 2: Minimización del Makespan

En este apartado, las heurísticas están minimizando el Makespan pero se van a evaluar las soluciones obtenidas para todos los objetivos ajustando con el λ la ponderación tal y como se presenta en la siguiente ecuación:

$$mo_{\lambda} = \lambda TCIT + (1-\lambda)C_{max}$$

En la Tabla 4.4 se muestran los resultados correspondientes a función objetivo. Posteriormente se detalla la representación de las gráficas en las Figuras 4.17, 4.18, 4.19, 4.20 y 4.21.

Tabla 4.4 Resumen Resultados Caso 2: C_{max} (Elaboración propia).

Tamaño instancias	H1	H2	H3	H4
		$\lambda = 0$		
4x4	258,50	258,50	251,20	271,20
5x5	341,80	341,80	340,90	356,70
7x7	483,90	483,90	479,70	494,80
10x10	650,00	650,00	647,90	681,20
15x15	965,80	965,80	965,70	1044,20
20x20	1318,40	1318,40	1318,00	1442,50
Promedio	669,73	669,73	667,23	715,10
		$\lambda = 0{,}25$		
4x4	225,28	225,28	215,73	242,63
5x5	309,18	309,18	306,43	326,68
7x7	488,18	488,18	482,65	520,48
10x10	744,30	744,30	745,58	840,25
15x15	1369,45	1369,45	1369,95	1617,20
20x20	2248,23	2248,23	2246,58	2768,15
Promedio	897,43	897,43	894,48	1052,56
		$\lambda = 0{,}5$		
4x4	192,05	192,05	180,25	214,05
5x5	276,55	276,55	271,95	296,65
7x7	492,45	492,45	485,60	546,15
10x10	838,60	838,60	843,25	999,30
15x15	1773,10	1773,10	1774,20	2190,20
20x20	3178,05	3178,05	3175,15	4093,80
Promedio	1125,13	1125,13	1121,73	1390,03
		$\lambda = 0{,}75$		
4x4	158,83	158,83	144,78	185,48
5x5	243,93	243,93	237,48	266,63
7x7	496,73	496,73	488,55	571,83
10x10	932,90	932,90	940,93	1158,35
15x15	2176,75	2176,75	2178,45	2763,20
20x20	4107,88	4107,88	4103,73	5419,45
Promedio	1352,83	1352,83	1348,98	1727,49
		$\lambda = 1$		
4x4	125,60	125,60	109,30	156,90
5x5	211,30	211,30	203,00	236,60
7x7	501,00	501,00	491,50	597,50
10x10	1027,20	1027,20	1038,60	1317,40
15x15	2580,40	2580,40	2582,70	3336,20
20x20	5037,70	5037,70	5032,30	6745,10
Promedio	1580,53	1580,53	1576,23	2064,95

1

[1] * Para este promedio no han sido utilizadas todas las instancias por superar en alguna de ellas el tiempo límite establecido

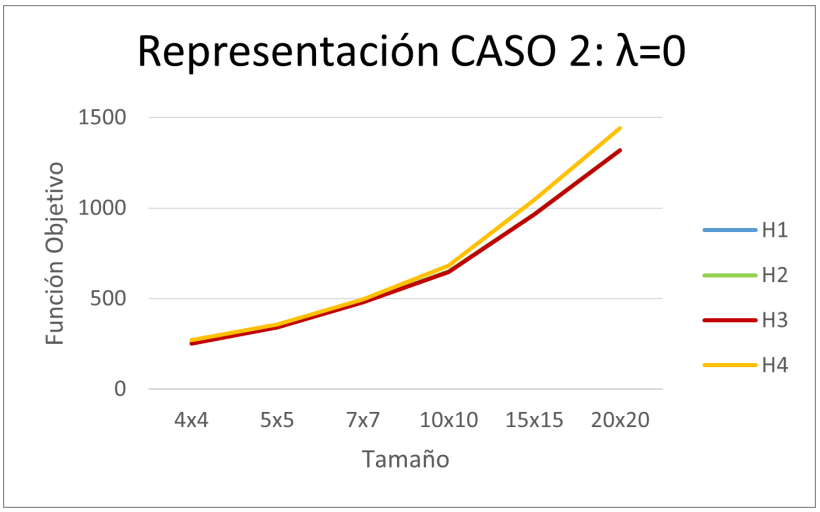

Figura 4.17 Representación Caso 2 $\lambda = 0$ (Elaboración propia).

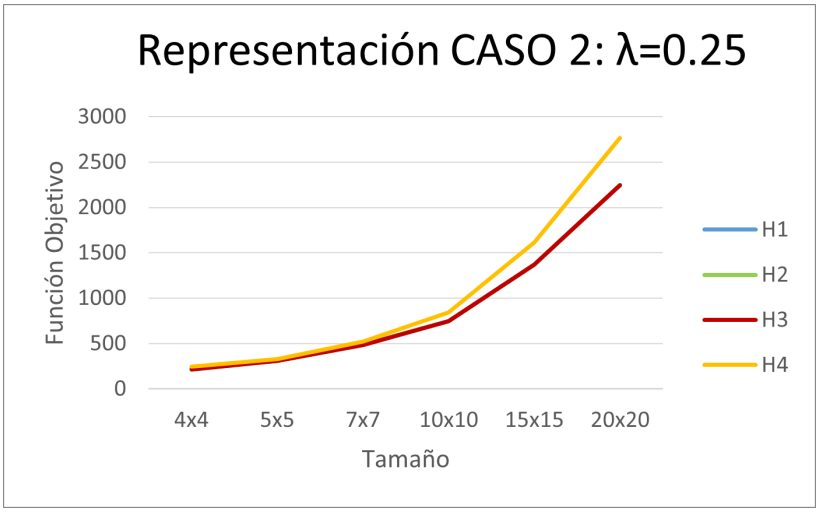

Figura 4.18 Representación Caso 2 $\lambda = 0{,}25$ (Elaboración propia).

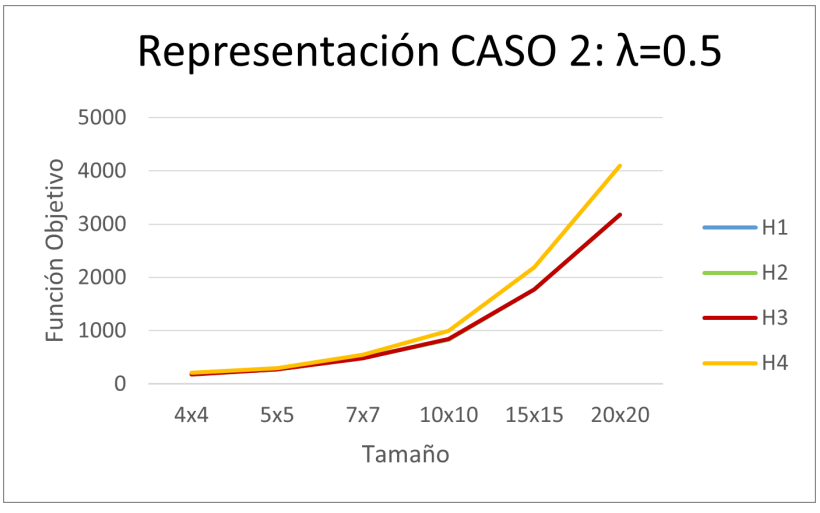

Figura 4.19 Representación Caso 2 $\lambda = 0{,}5$ (Elaboración propia).

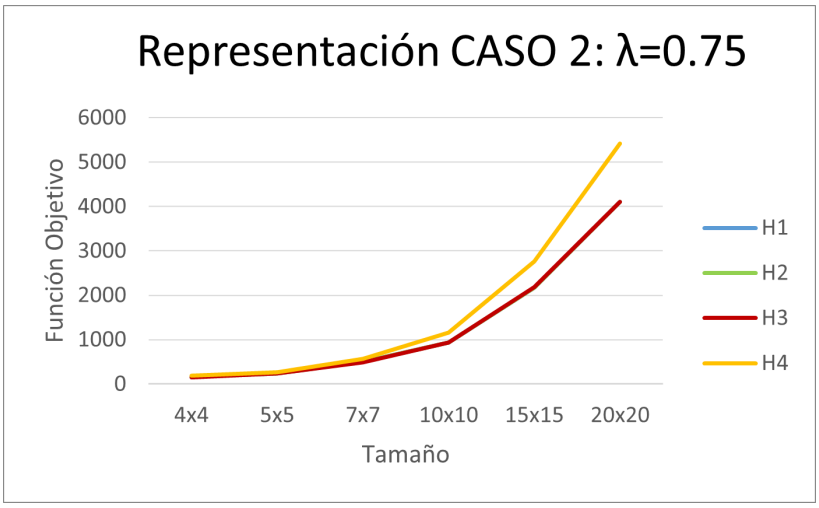

Figura 4.20 Representación Caso 2 $\lambda = 0{,}75$ (Elaboración propia).

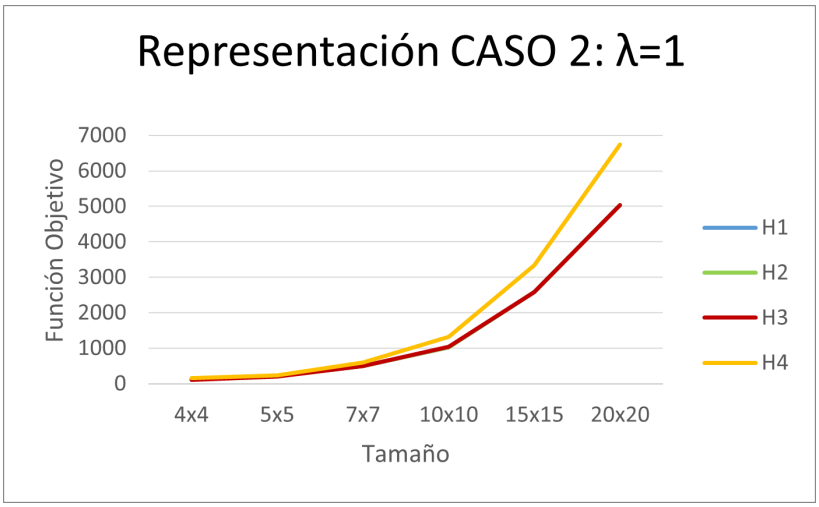

Figura 4.21 Representación Caso 2 $\lambda = 1$ (Elaboración propia).

En el Caso 2, es decir, en el que la operación se inserta en función del Makespan, es fácil concluir observando la representación de las gráficas en las Figuras 4.17, 4.18, 4.19, 4.20 y 4.21, que para cualquier valor de λ la Heurística 3 es la que proporciona mejores valores. Aunque se debe mencionar que estos resultados son considerables con respecto a la Heurística 4 pero despreciables con respecto a las Heurísticas 1 y 2.

4.5.4 Caso 2: Minimización del Makespan (RDI)

En la Tabla 4.5 se muestran los resultados correspondientes a función objetivo. Posteriormente se detalla la representación de las gráficas en las Figuras 4.22, 4.23, 4.24, 4.25 y 4.26.

Tabla 4.5 Resumen Resultados RDI Caso 2: C_{max} (Elaboración propia).

Tamaño instancias	H1	H2	H3	H4
	$\lambda = 0$			
4x4	0,58	0,58	0,27	0,62
5x5	0,31	0,31	0,26	0,70
7x7	0,53	0,53	0,32	0,56
10x10	0,18	0,18	0,10	0,90
15x15	0,00	0,00	0,00	1,00
20x20	0,00	0,00	0,00	1,00
Promedio	0,27	0,27	0,16	0,80
	$\lambda = 0{,}25$			
4x4	0,50	0,50	0,30	0,61
5x5	0,33	0,33	0,22	0,70
7x7	0,30	0,30	0,19	0,80
10x10	0,02	0,02	0,11	0,93
15x15	0,00	0,00	0,00	1,00
20x20	0,00	0,00	0,00	1,00
Promedio	0,19	0,19	0,14	0,84
	$\lambda = 0{,}5$			
4x5	0,61	0,61	0,40	0,56
5x5	0,54	0,54	0,40	0,50
7x7	0,28	0,28	0,20	0,80
10x10	0,02	0,02	0,11	0,91
15x15	0,10	0,10	0,10	0,90
20x20	0,00	0,00	0,00	1,00
Promedio	0,26	0,26	0,20	0,78
	$\lambda = 0{,}75$			
4x4	0,60	0,60	0,42	0,56
5x5	0,55	0,55	0,40	0,51
7x7	0,27	0,27	0,20	0,80
10x10	0,02	0,02	0,11	0,90
15x15	0,10	0,10	0,10	0,90
20x20	0,00	0,00	0,00	1,00
Promedio	0,26	0,26	0,21	0,78
	$\lambda = 1$			
4x4	0,59	0,59	0,44	0,56
5x5	0,55	0,55	0,40	0,52
7x7	0,26	0,26	0,20	0,80
10x10	0,01	0,01	0,11	0,90
15x15	0,10	0,10	0,10	0,90
20x20	0,00	0,00	0,00	1,00
Promedio	0,25	0,25	0,21	0,78

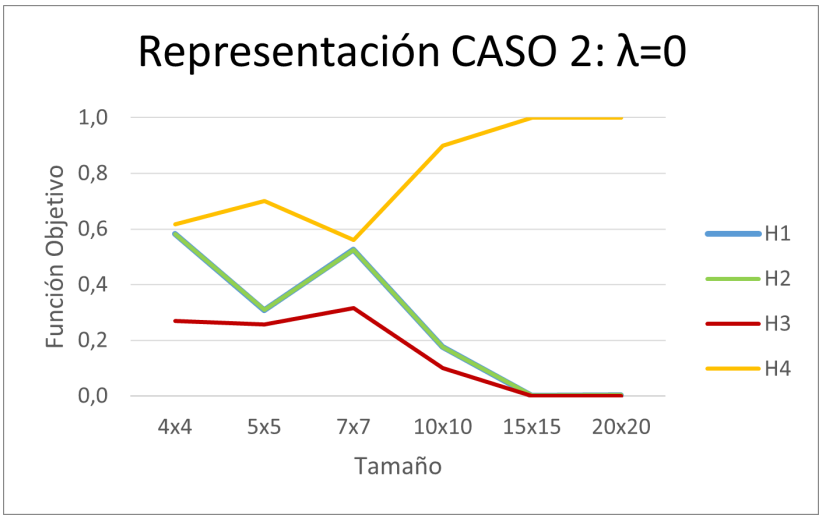

Figura 4.22 Representación RDI Caso 2 $\lambda = 0$ (Elaboración propia).

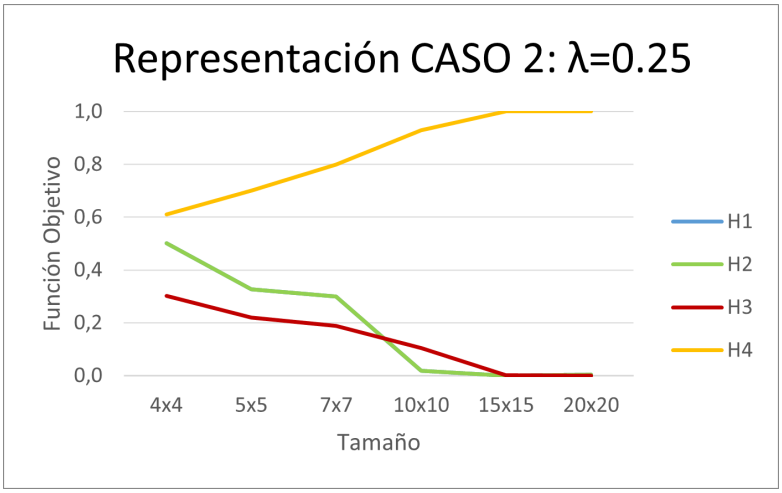

Figura 4.23 Representación RDI Caso 2 $\lambda = 0{,}25$ (Elaboración propia).

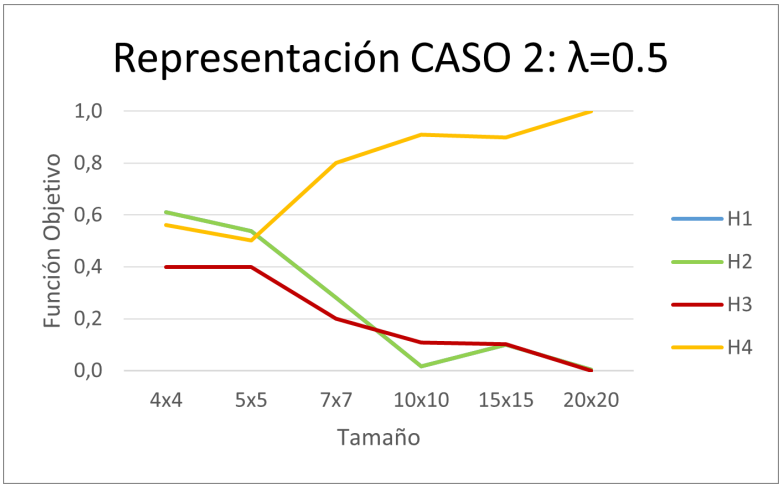

Figura 4.24 Representación RDI Caso 2 $\lambda = 0{,}5$ (Elaboración propia).

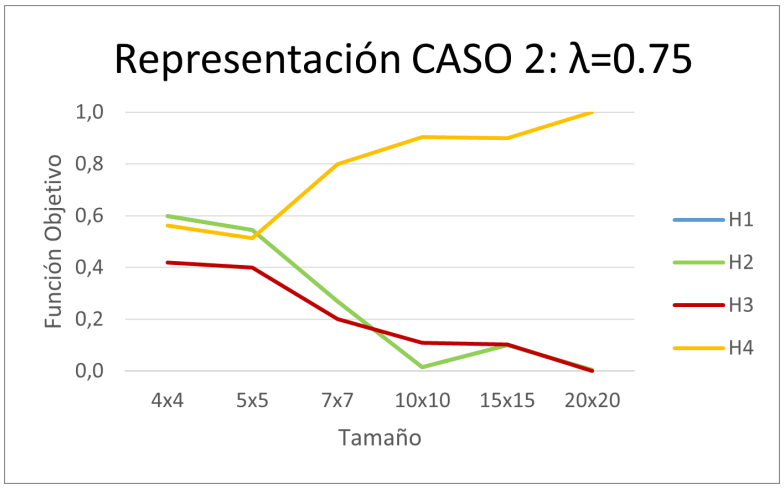

Figura 4.25 Representación RDI Caso 2 $\lambda = 0{,}75$ (Elaboración propia).

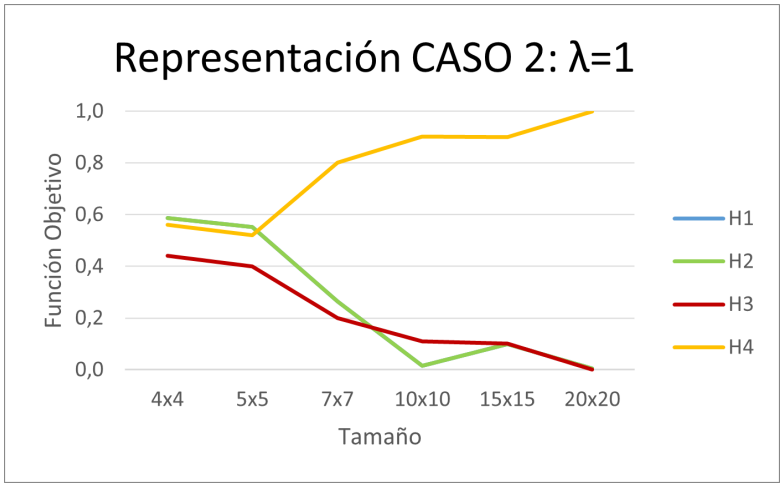

Figura 4.26 Representación RDI Caso 2 $\lambda = 1$ (Elaboración propia).

En este caso, de forma general se puede concluir que para cualquier valor de λ, la Heurística 3 es la que proporciona mejores valores. Aunque se debe mencionar que estos resultados son considerables con respecto a la Heurística 4 pero despreciables con respecto a las Heurísticas 1 y 2.

4.5.5 Caso 3: Minimización Multiobjetivo

En este apartado, las heurísticas están minimizando la función multiobjetivo. A continuación se presenta la ecuación:

$$mo_{\lambda} = \lambda TCIT + (1-\lambda)C_{max}$$

En la Tabla 4.6 se muestran los resultados correspondientes a función objetivo. Posteriormente se detalla la representación de las gráficas en las Figuras 4.27, 4.28, 4.29, 4.30 y 4.31.

Tabla 4.6 Resumen Resultados Caso 3 (Elaboración propia).

Tamaño instancias	H1	H2	H3	H4
		$\lambda = 0$		
4x4	258,50	258,50	251,20	271,20
5x5	341,80	341,80	340,90	356,70
7x7	483,90	483,90	479,70	494,80
10x10	650,00	650,00	647,90	681,20
15x15	965,80	965,80	965,70	1044,20
20x20	1318,40	1318,40	1318,10*	1442,50
Promedio	669,73	669,73	667,25	715,10
		$\lambda = 0{,}25$		
4x4	227,43	227,43	216,28	231,90
5x5	305,98	305,98	295,58	334,55
7x7	500,83	500,83	451,33	489,28
10x10	704,48	704,48	688,78	747,00
15x15	1246,15	1246,15	1246,15*	1360,88
20x20	1970,43	1970,43	1970,43	1978,98
Promedio	825,88	825,88	811,42	857,10
		$\lambda = 0{,}5$		
4x4	181,35	181,35	162,40	177,15
5x5	281,75	281,75	249,85	285,95
7x7	452,15	452,15	411,30	439,00
10x10	694,60	694,60	575,10	644,65
15x15	1330,70	1330,70	1247,50*	1323,75*
20x20	2408,35	2408,35	2407,90*	2374,42
Promedio	891,48	891,48	842,34	874,15
		$\lambda = 0{,}75$		
Tamaño instancias	**H1**	**H2**	**H3**	**H4**
4x4	136,38	136,38	94,93	121,78
5x5	217,10	217,10	156,18	192,30
7x7	361,15	361,15	278,23	361,90
10x10	607,55	607,55	426,30	616,85
15x15	1367,23	1367,23	1216,93*	1376,70
20x20	2759,10	2759,10	2732,18*	2714,40
Promedio	908,08	908,08	817,45	897,32
		$\lambda = 1$		
4x4	72,20	72,20	3,50	36,80
5x5	141,80	141,80	14,40	55,30
7x7	249,30	249,30	12,50	167,70
10x10	225,90	225,90	59,00	245,30
15x15	1125,40	1125,40	540,50*	992,50
20x20	2019,40	2019,40	1862,20*	2499,25*
Promedio	639,00	639,00	415,35	666,14

* Para este promedio no han sido utilizadas todas las instancias por superar en alguna de ellas el tiempo límite establecido

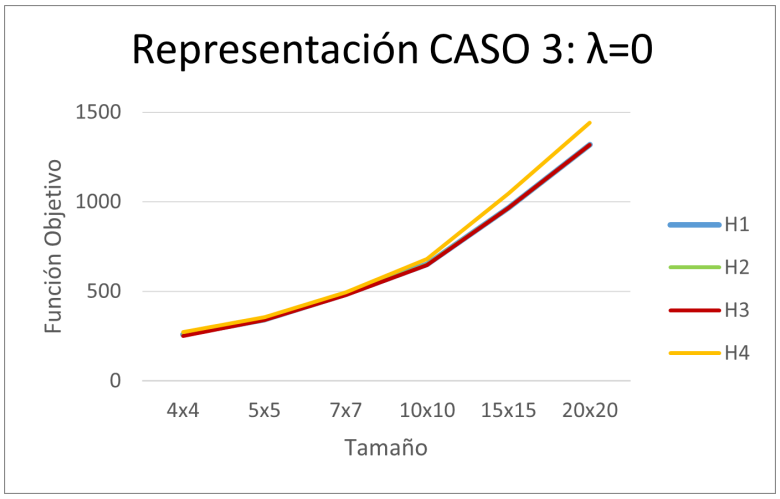

Figura 4.27 Representación Caso 3 $\lambda = 0$ (Elaboración propia).

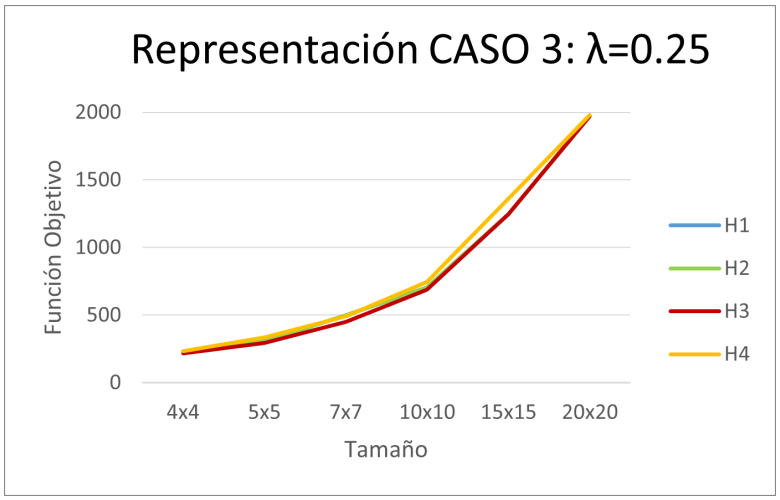

Figura 4.28 Representación Caso 3 $\lambda = 0{,}25$ (Elaboración propia).

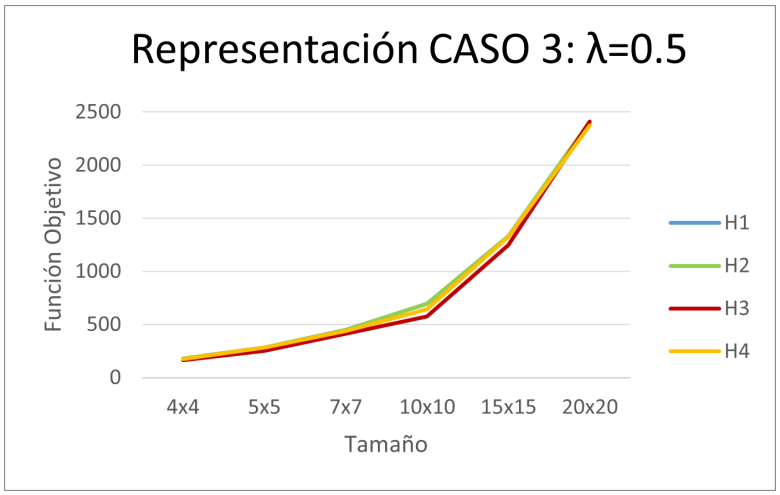

Figura 4.29 Representación Caso 3 $\lambda = 0{,}5$ (Elaboración propia).

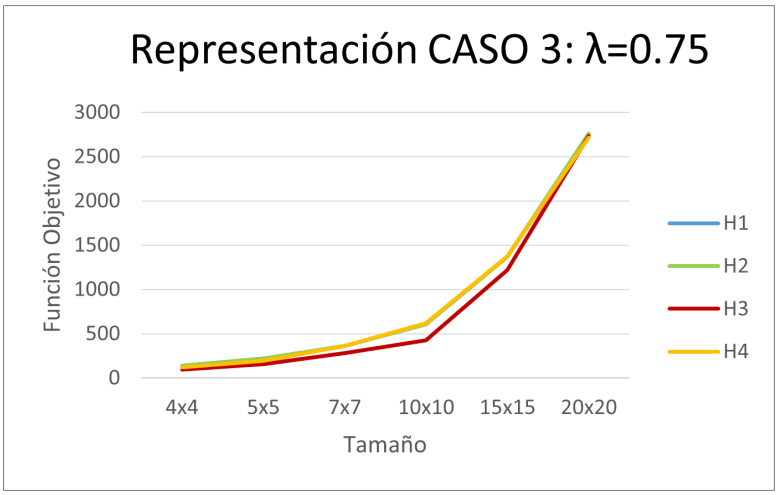

Figura 4.30 Representación Caso 3 $\lambda = 0{,}75$ (Elaboración propia).

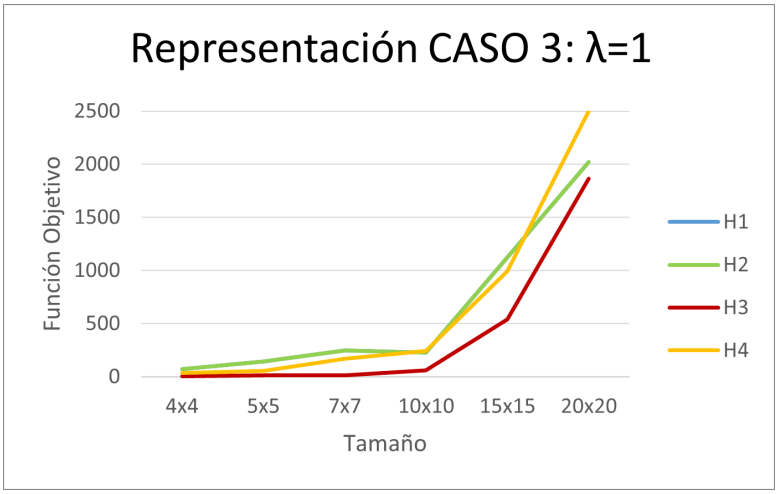

Figura 4.31 Representación Caso 3 $\lambda = 1$ (Elaboración propia).

Tal y como se puede apreciar a simple vista en las Figuras 4.27, 4.28, 4.29, 4.30 y 4.31, para valores de λ pequeños, la Heurística 1 y 2 proporcionan resultados similares a los de la Heurística 3. Si bien es cierto que, conforme aumenta el valor de λ y se le empieza a dar más peso al Core Idle Time, la Heurística 3 es la que presenta mejores valores. Es por ello, que se puede concluir afirmando que en este Caso 3, la Heurística 3 es la candidata seleccionada.

4.5.6 Caso 3: Minimización Multiobjetivo (RDI)

En la Tabla 4.7 se muestran los resultados correspondientes a función objetivo. Posteriormente se detalla la representación de las gráficas en las Figuras 4.32, 4.33, 4.34, 4.35 y 4.36.

Tabla 4.7 Resumen Resultados RDI Caso 3 (Elaboración propia).

Tamaño instancias	H1	H2	H3	H4
	$\lambda = 0$			
4x4	0,58	0,58	0,27	0,62
5x5	0,31	0,31	0,26	0,70
7x7	0,53	0,53	0,32	0,32
10x10	0,18	0,18	0,10	0,90
15x15	0,00	0,00	0,00	1,00
20x20	0,00	0,00	0,00*	1,00
Promedio	0,27	0,27	0,16	0,76
	$\lambda = 0{,}25$			
4x4	0,58	0,58	0,20	0,70
5x5	0,44	0,44	0,15	0,70
7x7	0,82	0,82	0,03	0,62
10x10	0,45	0,45	0,22	0,70
15x15	0,20	0,20	0,20*	0,80
20x20	0,40	0,40	0,40	0,60
Promedio	0,48	0,48	0,20	0,69
	$\lambda = 0{,}5$			
4x4	0,69	0,69	0,00	0,73
5x5	0,72	0,72	0,10	0,63
7x7	0,84	0,84	0,24	0,42
10x10	0,91	0,91	0,03	0,56
15x15	0,70	0,70	0,38*	0,44*
20x20	0,60	0,60	0,60*	0,40
Promedio	0,74	0,74	0,35	0,53
	$\lambda = 0{,}75$			
4x4	0,63	0,63	0,03	0,65
5x5	0,86	0,86	0,12	0,48
7x7	0,81	0,81	0,10	0,70
10x10	0,77	0,77	0,10	0,69
15x15	0,61	0,61	0,34*	0,58
20x20	0,49	0,49	0,40*	0,60
Promedio	0,69	0,69	0,18	0,62
	$\lambda = 1$			
4x4	0,78	0,78	0,03	0,43
5x5	0,64	0,64	0,10	0,49
7x7	0,55	0,55	0,00	0,64
10x10	0,69	0,69	0,05	0,74
15x15	0,74	0,74	0,01*	0,70
20x20	0,51	0,51	0,37*	0,60*
Promedio	0,65	0,65	0,09	0,60

* Para este promedio no han sido utilizadas todas las instancias por superar en alguna de ellas el tiempo límite establecido

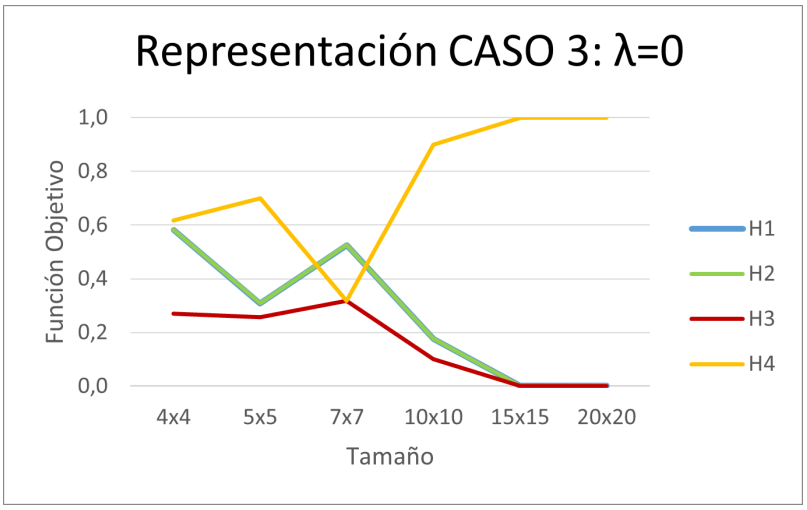

Figura 4.32 Representación RDI Caso 3 $\lambda = 0$ (Elaboración propia).

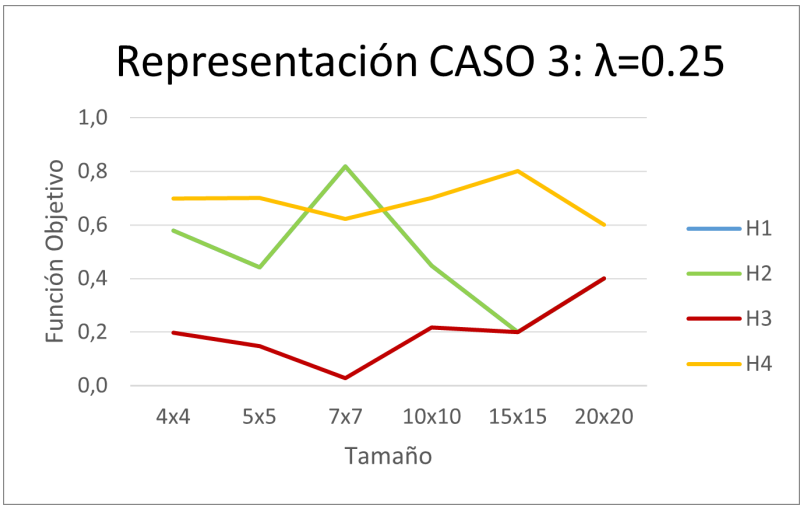

Figura 4.33 Representación RDI Caso 3 $\lambda = 0{,}25$ (Elaboración propia).

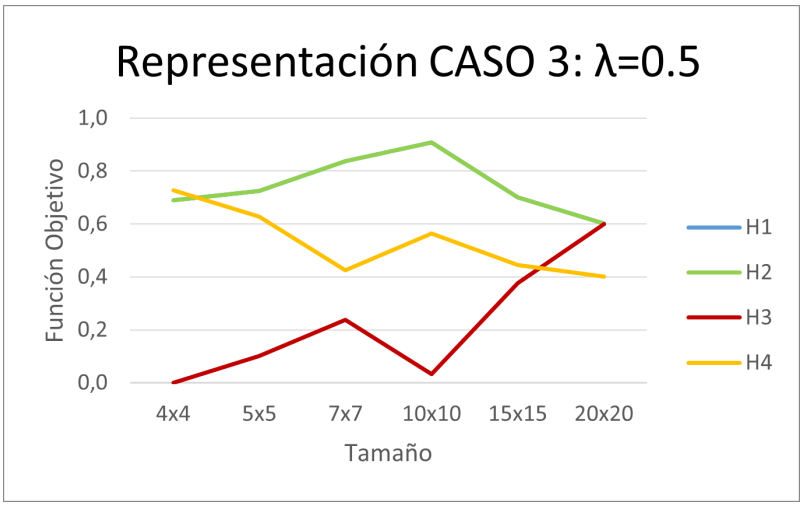

Figura 4.34 Representación RDI Caso 3 $\lambda = 0{,}5$ (Elaboración propia).

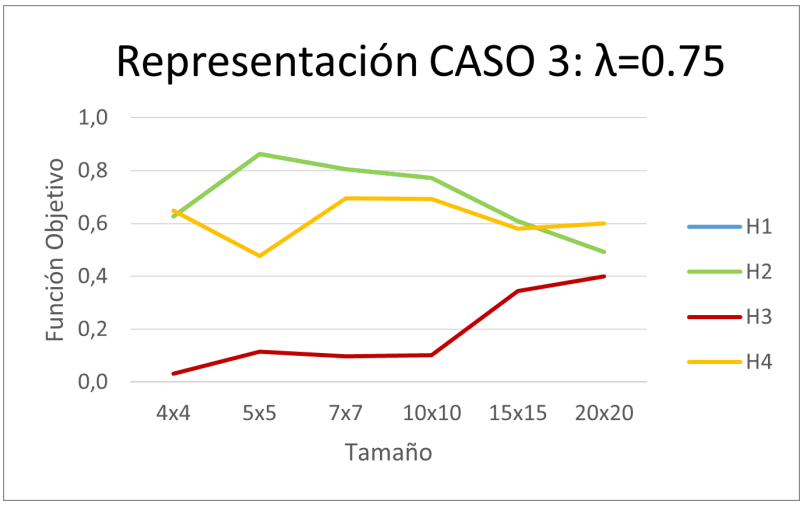

Figura 4.35 Representación RDI Caso 3 $\lambda = 0{,}75$ (Elaboración propia).

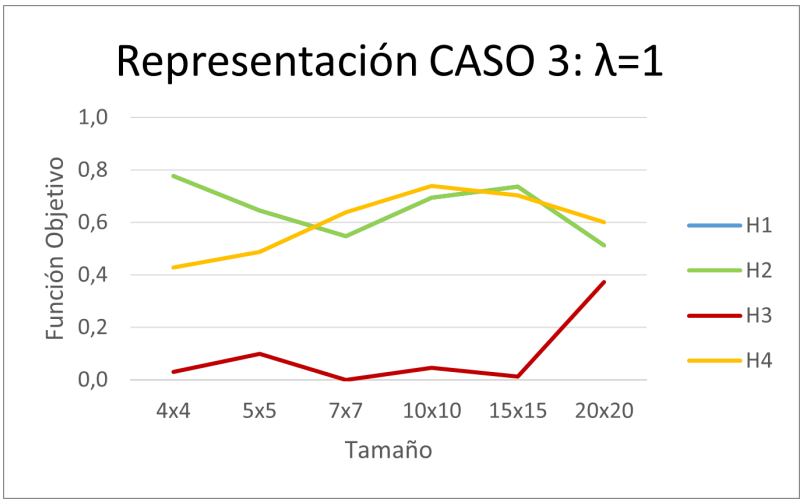

Figura 4.36 Representación RDI Caso 3 $\lambda = 1$ (Elaboración propia).

Tal y como se puede apreciar en las Figuras 4.32, 4.33, 4.34, 4.35 y 4.36, para cualquier valor de λ la Heurística 3 es la que proporciona mejores resultados.

4.5.7 Comparación general

En este apartado se presentan las Figuras 4.37, 4.38, 4.39, 4.40 y 4.41, en las que se detallan todas las heurísticas de interés en función del valor de λ. Tras esto, se realizará una explicación de las mismas. Además, para facilitar su interpretación, se han usado para las Heurísticas 1,2,3 y 4 diferentes tonalidades de azul, verde, naranja y amarillo respectivamente. Además, se han incluido en los cambios de la dimensión de las instancias diferentes geometrías en función del caso al que correspondan, para el Caso 1 se ha usado el círculo, para el Caso 2 el cuadrado y para el Caso 3 el triángulo.

Mencionar que cuando $\lambda = 0$, llama la atención que el valor de la función objetivo de las Heurísticas en el Caso 2 coinciden con las Heurísticas en el Caso 3. Esto es así dado que, cuando $\lambda = 0$, tal y como se puede apreciar en la Ecuación 2.1, únicamente se tiene en cuenta el valor del Makespan (para $\lambda = 0$ los resultados coinciden en las Tablas 4.4 y 4.6). Lo mismo ocurre cuando $\lambda = 1$, en el que las Heurísticas en el Caso 1 coinciden con las heurísticas en el Caso 3 dado que únicamente se tiene en cuenta el Core Idle Time ($\lambda = 1$ Tabla 4.2 y 4.6). Aquellos casos en lo que no sucede lo mencionado anteriormente, sino que se obtienen resultados ligeramente distintos se debe a que el resultado de alguna de las instancias no se ha tenido en cuenta en el promedio calculado dado que superaba el tiempo límite de computo establecido.

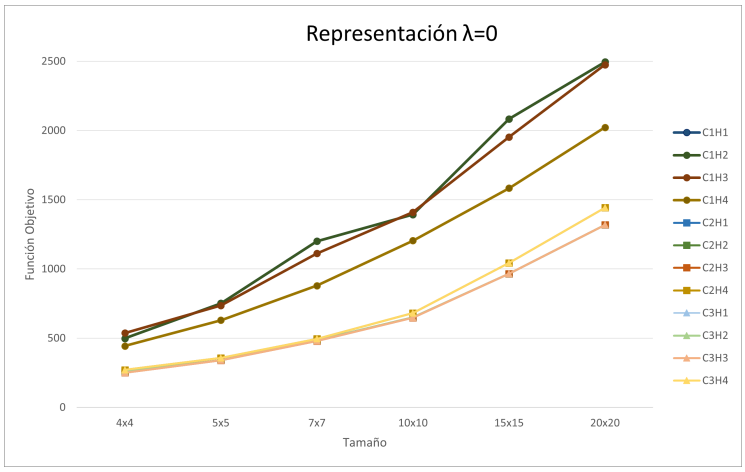

Figura 4.37 Representación General $\lambda = 0$ (Elaboración propia).

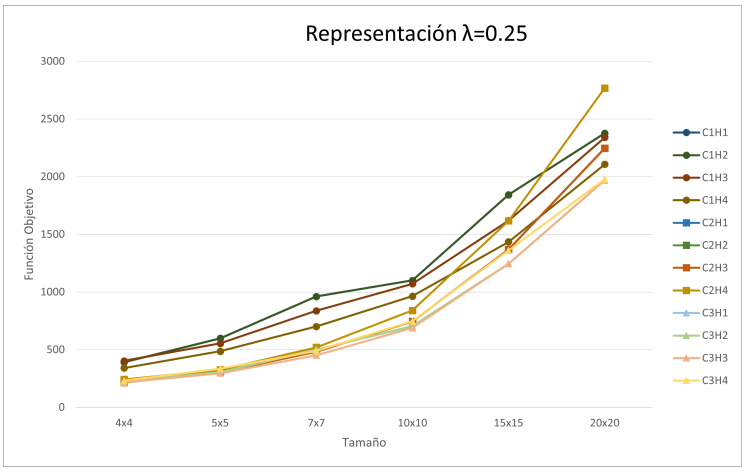

Figura 4.38 Representación General $\lambda = 0{,}25$ (Elaboración propia).

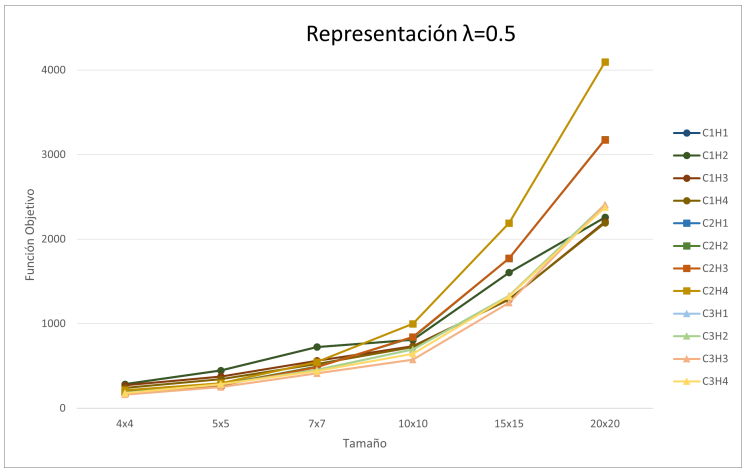

Figura 4.39 Representación General $\lambda = 0{,}5$ (Elaboración propia).

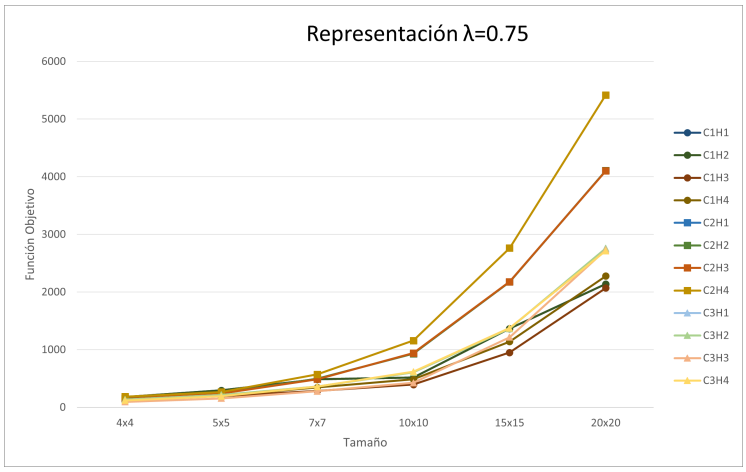

Figura 4.40 Representación General $\lambda = 0{,}75$ (Elaboración propia).

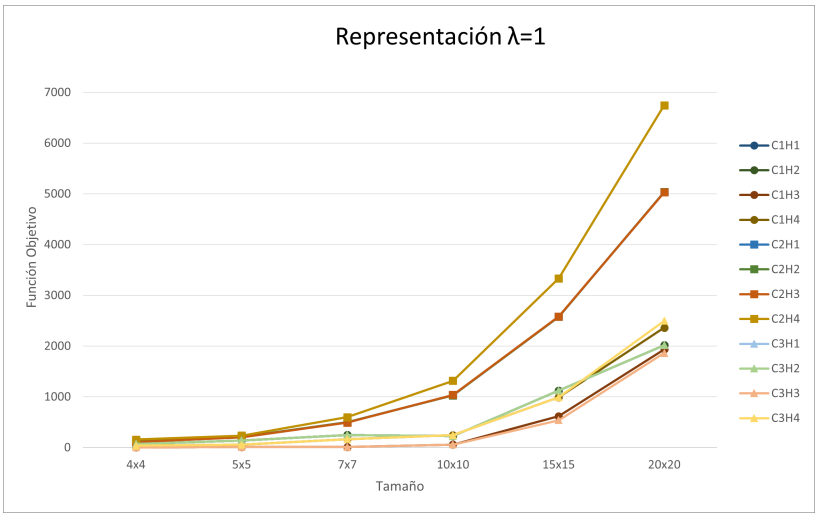

Figura 4.41 Representación General $\lambda = 1$ (Elaboración propia).

A continuación, se presenta la mejor heurística obtenida en función del valor de λ, que representa el peso que se le desea dar a cada una de las funciones objetivos respectivamente teniendo en cuenta la Ecuación 2.1.

- $\lambda = 0 \longrightarrow$ Es el caso en el que únicamente se tiene en cuenta el valor del Makespan, y tal y como se puede apreciar en la Figura 4.37, las mejores soluciones las proporcionan C2H3 o C3H3.
- $\lambda = 0{,}25 \longrightarrow$ Es el caso en el que se pondera con un 25 % el valor del Core Idle Time y con un 75 % el valor del Makespan. Tras visualizar la Figura 4.38 la Heurística que proporciona mejores resultados es C3H3.
- $\lambda = 0{,}5 \longrightarrow$ Es el caso en el que se pondera con un 50 % el valor del Core Idle Time y con un 50 % el valor del Makespan. Tras visualizar la Figura 4.39 la Heurística que proporciona en promedio mejores resultados es C3H3 aunque para instancias con tamaño superior a 15x15 resulta mejor opción C1H4.
- $\lambda = 0{,}75 \longrightarrow$ Es el caso en el que se pondera con un 75 % el valor del Core Idle Time y con un 25 % el valor del Makespan. Tras visualizar la Figura 4.40 la Heurística que proporciona en promedio mejores resultados para cualquier valor de λ es C1H3.
- $\lambda = 1 \longrightarrow$ Es el caso en el que únicamente se tiene en cuenta el valor del Core Idle Time, y tal y como se puede apreciar en la Figura 4.41, las mejores soluciones las proporcionan C1H3 o C3H3.

4.5.8 Comparación general (RDI)

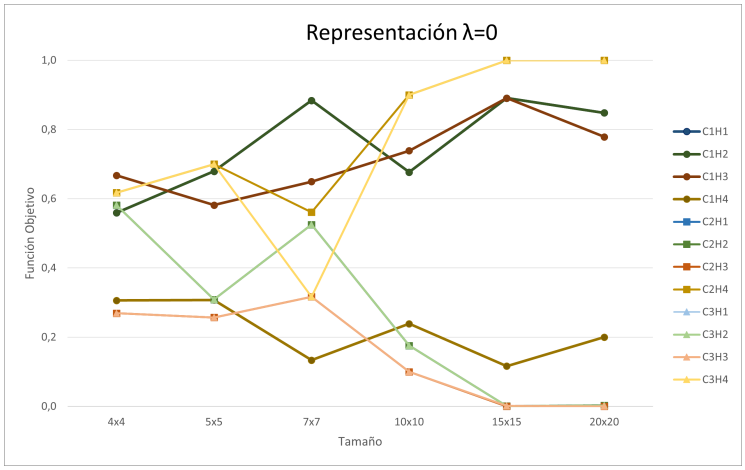

Figura 4.42 Representación RDI General $\lambda = 0$ (Elaboración propia).

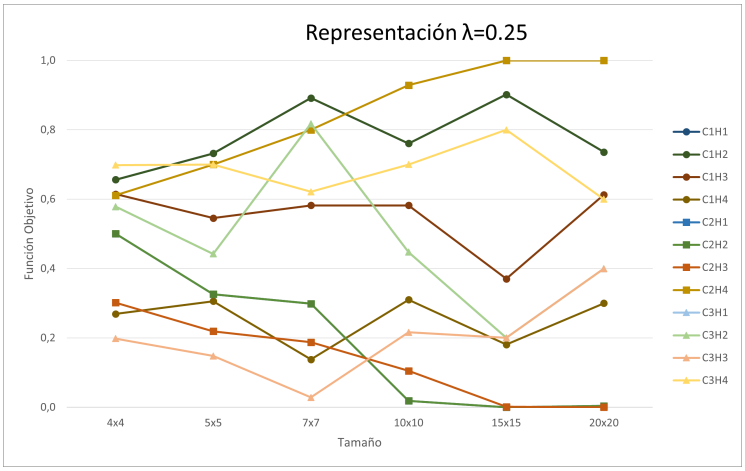

Figura 4.43 Representación RDI General $\lambda = 0{,}25$ (Elaboración propia).

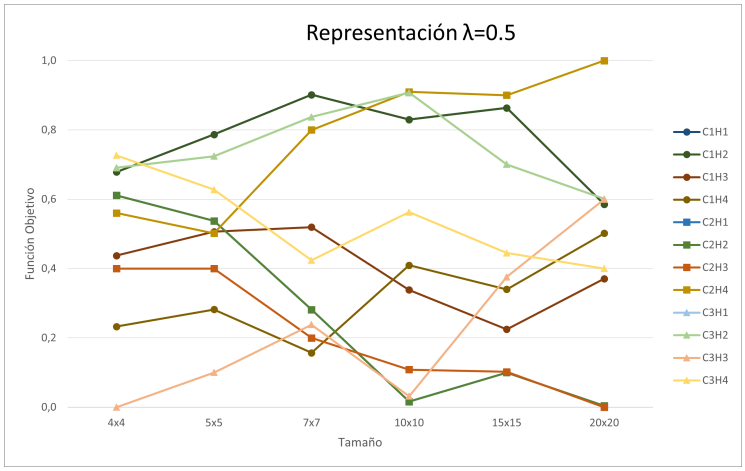

Figura 4.44 Representación RDI General $\lambda = 0{,}5$ (Elaboración propia).

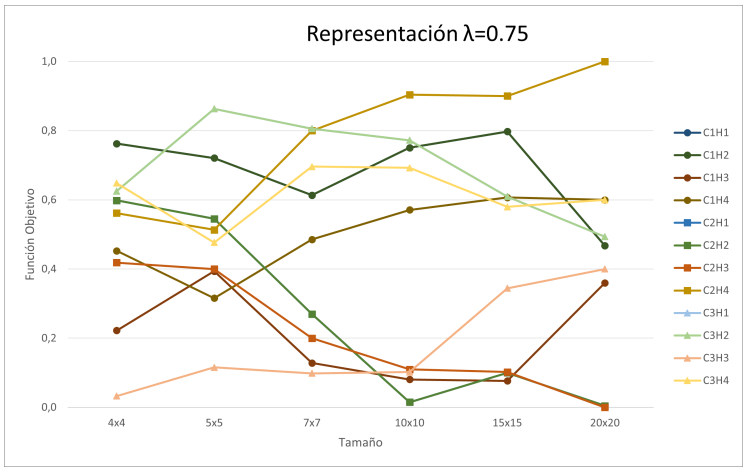

Figura 4.45 Representación RDI General $\lambda = 0{,}75$ (Elaboración propia).

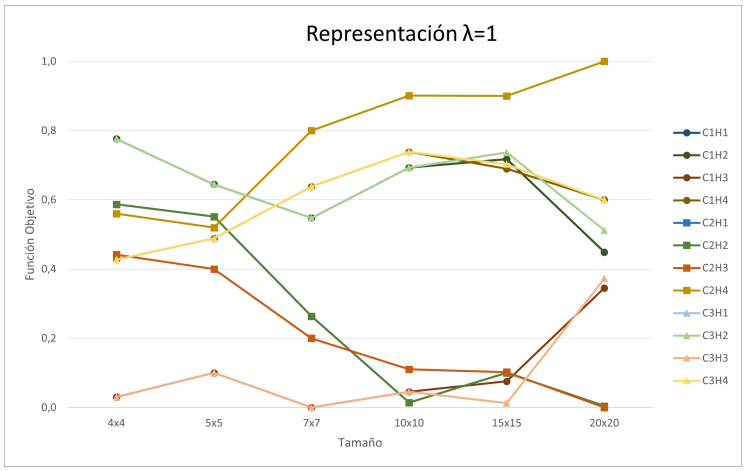

Figura 4.46 Representación RDI General $\lambda = 1$ (Elaboración propia).

En este apartado se han detallado las Figuras 4.42, 4.43, 4.44, 4.45 y 4.46. A continuación, se presenta la mejor heurística obtenida en función del valor de λ.

- $\lambda = 0 \longrightarrow$ Es el caso en el que únicamente se tiene en cuenta el valor del Makespan, y tal y como se puede apreciar en la Figura 4.42, de forma general las mejores soluciones las proporcionan C2H3 o C3H3.
- $\lambda = 0{,}25 \longrightarrow$ Es el caso en el que se pondera con un 25 % el valor del Core Idle Time y con un 75 % el valor del Makespan. Tras visualizar la Figura 4.43, para tamaños inferiores a 7x7, la Heurística que proporciona mejores resultados es C3H3. Por otro lado, para instancias superiores, el C2H2 proporciona mejores valores.
- $\lambda = 0{,}5 \longrightarrow$ Es el caso en el que se pondera con un 50 % el valor del Core Idle Time y con un 50 % el valor del Makespan. Tras visualizar la Figura 4.44, para tamaños inferiores a 7x7, la Heurística que proporciona mejores resultados es C3H3 aunque para instancias superiores, es el C2H2.
- $\lambda = 0{,}75 \longrightarrow$ Es el caso en el que se pondera con un 75 % el valor del Core Idle Time y con un 25 % el valor del Makespan. Tras visualizar la Figura 4.45, para tamaños inferiores a 7x7, la Heurística que proporciona mejores resultados es C3H3 aunque para instancias superiores, es el C2H2.
- $\lambda = 1 \longrightarrow$ Es el caso en el que únicamente se tiene en cuenta el valor del Core Idle Time, y tal y como se puede apreciar en la Figura 4.46, de forma general las mejores soluciones las proporcionan C1H3 o C3H3.

4.5.9 Comparación del tiempo de computo

A continuación, se presentan los resultados de los tiempos de computo obtenidos en función del tamaño para cada una de las heurísticas. Además, resulta de interés mencionar que se ha establecido como tiempo límite por instancia 10 minutos. Primero se muestran las Tablas 4.8, 4.9 y 4.10 correspondientes al Caso 1, 2 y 3 respectivamente. Posteriormente se detallan las gráficas con una explicación de las mismas.

Tabla 4.8 Resultados Tiempo de Computo Caso 1 (Elaboración propia).

H1	
Tamaño instancias	**Tiempo de Computo**
4x4	0,01
5x5	0,03
7x7	0,13
10x10	0,85
15x15	7,65
20x20	38,36
Resultados promedios	7,84
H2	
Tamaño instancias	**Tiempo de Computo**
4x4	0,01
5x5	0,04
7x7	0,18
10x10	1,12
15x15	9,39
20x20	75,89
Resultados promedios	14,44
H3	
Tamaño instancias	**Tiempo de Computo**
4x4	0,09
5x5	0,23
7x7	7,70
10x10	54,35
15x15	592,06
20x20	601,10
Resultados promedios	209,25
H4	
Tamaño instancias	**Tiempo de Computo**
4x4	0,05
5x5	0,17
7x7	0,99
10x10	6,71
15x15	65,47
20x20	359,87
Resultados promedios	72,21

Tabla 4.9 Resultados Tiempo de Computo Caso 2 (Elaboración propia).

H1	
Tamaño instancias	**Tiempo de Computo**
4x4	0,01
5x5	0,01
7x7	0,05
10x10	0,22
15x15	2,12
20x20	15,34
Resultados promedios	2,96
H2	
Tamaño instancias	**Tiempo de Computo**
4x4	0,01
5x5	0,02
7x7	0,08
10x10	0,36
15x15	3,84
20x20	12,77
Resultados promedios	2,84
H3	
Tamaño instancias	**Tiempo de Computo**
4x4	0,04
5x5	0,08
7x7	0,61
10x10	4,07
15x15	30,44
20x20	192,09
Resultados promedios	37,89
H4	
Tamaño instancias	**Tiempo de Computo**
4x4	0,02
5x5	0,06
7x7	0,25
10x10	1,34
15x15	10,10
20x20	46,68
Resultados promedios	9,74

Tabla 4.10 Resultados Tiempo de Computo Caso 3 (Elaboración propia).

H1	
Tamaño instancias	**Tiempo de Computo**
4x4	0,01
5x5	0,03
7x7	0,11
10x10	0,72
15x15	9,14
20x20	38,82
Resultados promedios	8,14
H2	
Tamaño instancias	**Tiempo de Computo**
4x4	0,01
5x5	0,03
7x7	0,15
10x10	1,13
15x15	11,10
20x20	49,54
Resultados promedios	10,33
H3	
Tamaño instancias	**Tiempo de Computo**
4x4	0,06
5x5	0,15
7x7	1,43
10x10	12,14
15x15	125,71
20x20	601,43
Resultados promedios	123,49
H4	
Tamaño instancias	**Tiempo de Computo**
4x4	0,04
5x5	0,12
7x7	0,78
10x10	7,27
15x15	58,87
20x20	279,21
Resultados promedios	57,71

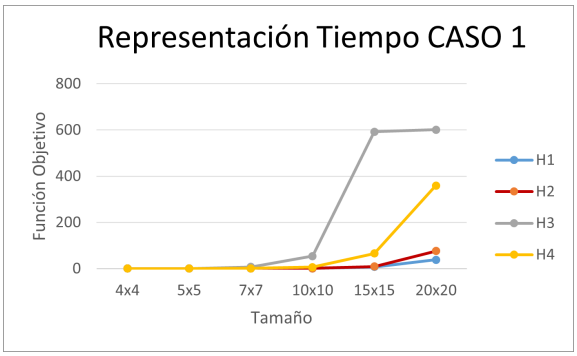

Figura 4.47 Representación Tiempo de Computo General (Elaboración propia).

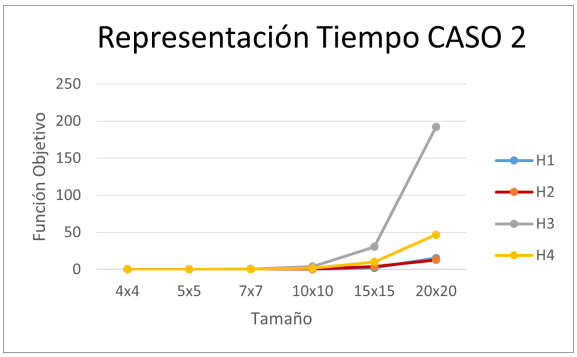

Figura 4.48 Representación Tiempo de Computo General (Elaboración propia).

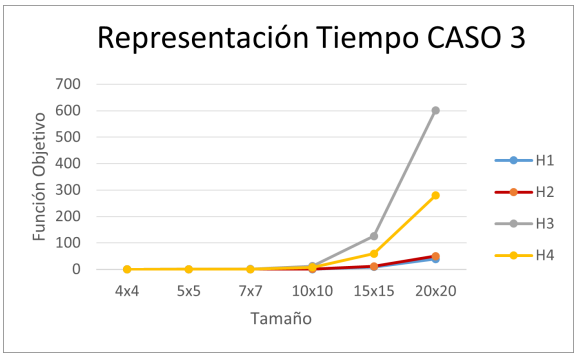

Figura 4.49 Representación Tiempo de Computo General (Elaboración propia).

Tal y como se ha podido apreciar en las Figuras 4.47, 4.48 y 4.49, al aumentar el tamaño de las instancias, también lo hace el tiempo de computo. Además, independientemente del caso, la Heurística 1 y la Heurística 2 son las que menor tiempo de computo proporcionan, mientras que por otro lado, la Heurística 3 es la que mayor tiempo refleja. Por otro lado, resulta de interés mencionar que, en general, los valores mostrados en el Caso 2 (únicamente se tiene en cuenta la FO del Makespan) son inferiores a los que se presentan en el Caso 1 y Caso 3. Además, llama la atención que la Heurística 3 en el Caso 1 mantiene constante el tiempo de computo para los tamaños 15x15 y 20x20. Esto se debe a que se ha establecido como tiempo límite de computo 600 segundos.

5 Conclusión

"Los que aseguran que es imposible no deberían interrumpir a los que estamos intentándolo"

Thomas Alva Edison

Tras revisar las contribuciones literarias existentes hasta el momento, referentes a la programación de operaciones del Open Shop, se ha observado que ninguna de ellas se aborda teniendo en cuenta objetivos de sostenibilidad. Por ello, se ha identificado la necesidad de abordar el problema de programación de operaciones del Open Shop en este Trabajo de Fin de Grado con el objetivo de afrontar uno de los principales problemas que encontramos hoy en día como es la contaminación ambiental.

Se han estudiado cuatro heurísticas mediante operadores de inserción y reinserción y, para analizar la eficiencia de estas, se han adaptando las instancias propuestas por Naderi (Naderi et al., 2010).

Tras analizar la experimentación, resultaba interesante estudiar los resultados en función del parámetro λ, es decir, en función del peso que se le quiera dar al Makespan y el Core Idle Time. Estos resultados para el caso de utilizar los valores resultantes de la función objetivo han sido:

- $\lambda = 0$: Únicamente se tiene en cuenta el valor del Makespan y las mejores soluciones las proporcionan C2H3 o C3H3.
- $\lambda = 0{,}25$: Se pondera con un 25 % el valor del Core Idle Time y con un 75 % el valor del Makespan. En este, la heurística que proporciona mejores resultados es C3H3.
- $\lambda = 0{,}5$: Se pondera con un 50 % el valor del Core Idle Time y con un 50 % el valor del Makespan y la mejor opción en general, resulta ser C3H3 aunque para instancias con tamaño superior a 15x15 resulta mejor opción C1H4.
- $\lambda = 0{,}75$: Se pondera con un 75 % el valor del Core Idle Time y con un 25 % el valor del Makespan y la heurística que proporciona mejores resultados es C1H3.
- $\lambda = 1$: Únicamente se tiene en cuenta el valor del Core Idle Time y las mejores soluciones las proporcionan C1H3 o C3H3.

Por otro lado, teniendo en cuenta los valores proporcionados por el indicador RDI se ha obtenido:

- $\lambda = 0 \longrightarrow$ Es el caso en el que únicamente se tiene en cuenta el valor del Makespan, y tal y como se puede apreciar en la Figura 4.42, para la mayoría de los tamaños las mejores soluciones las proporcionan C2H3 o C3H3.
- $\lambda = 0{,}25 \longrightarrow$ Es el caso en el que se pondera con un 25 % el valor del Core Idle Time y con un 75 % el valor del Makespan. Tras visualizar la Figura 4.43, para tamaños inferiores a 7x7, la Heurística que proporciona mejores resultados es C3H3. Mientras que, para instancias superiores, el C2H2 proporciona mejores valores.
- $\lambda = 0{,}5 \longrightarrow$ Es el caso en el que se pondera con un 50 % el valor del Core Idle Time y con un 50 % el valor del Makespan. Tras analizar la Figura 4.44, de nuevo se obtiene que para tamaños inferiores a 7x7, la Heurística que proporciona mejores resultados es C3H3 aunque para instancias superiores, es el C2H2.

- $\lambda = 0{,}75 \longrightarrow$ Es el caso en el que se pondera con un 75 % el valor del Core Idle Time y con un 25 % el valor del Makespan. Tras estudiar la Figura 4.45, otra vez se obtiene que para tamaños inferiores a 7x7, la Heurística que proporciona mejores resultados es C3H3 aunque para instancias superiores, es el C2H2.
- $\lambda = 1 \longrightarrow$ Es el caso en el que únicamente se tiene en cuenta el valor del Core Idle Time, y tal y como se puede apreciar en la Figura 4.46, de forma general las mejores soluciones las proporcionan C1H3 o C3H3.

Los resultados presentados a lo largo del documento se han obtenido a partir de instancias de diferentes tamaños obtenidas del artículo de Naderi et al. (2010). Por otro lado, para aumentar los límites del proyecto y no estudiar únicamente instancias que tengan el mismo número de máquinas que de trabajos (puesto que no es lo habitual), también se ha tratado de utilizar otras de las instancias (no cuadradas, de dimensión mxn) propuestas para el entorno del Flow Shop (Taillard, 1993). Sin embargo, el notorio aumento del tiempo de computo ha hecho imposible el estudio del mismo, ya que sobrepasaba el máximo límite de computo establecido.

El presente Trabajo de Fin de Grado forma parte de una Beca de Iniciación a la Investigación que se está desarrollando junto con el departamento de Organización Industrial y Gestión de Empresas I (Grupo de Investigación de Organización Industrial) y como futuras líneas de investigación de la beca, se proponen:

- El perfeccionamiento y explicación más exhaustiva de la Heurística 2, ya que el planteamiento de la misma en el artículo se encuentra muy escueta y la interpretación resulta muy compleja.
- Llevar a cabo un análisis similar al de este Trabajo Fin de Grado para instancias de mayor tamaño. Además, también resultaría interesante utilizar instancias que no sean cuadradas, es decir que tengan diferente número de máquinas que de trabajos, a pesar de que se necesite mayor tiempo de computo.
- Con el objetivo de obtener mejores soluciones, se propone la realización de la experimentación en un horizonte temporal mayor donde la búsqueda de la solución en cada instancia no se vea restringida por el tiempo límite de ejecución establecido.

Abreu, L., Cunha, J., Prata, B., and Framinan, J. (2020). A genetic algorithm for scheduling open shops with sequence-dependent setup times. *Computers and Operations Research*, 113.

Abreu, L. R., Prata, B. A., Framinan, J. M., and Nagano, M. S. (2022). New efficient heuristics for scheduling open shops with makespan minimization. *Computers Operations Research*, 142:105744.

Ahmadian, M., Khatami, M., Salehipour, A., and Cheng, T. (2021). Four decades of research on the open-shop scheduling problem to minimize the makespan. *European Journal of Operational Research*, 295:399–426.

Andresen, M., Bräsel, H., Mörig, M., Tusch, J., Werner, F., and Willenius, P. (2008). Simulated annealing and genetic algorithms for minimizing mean flow time in an open shop. *Mathematical and Computer Modelling*, 48:1279–1293.

Aziz, A., Razaullah, Ali, L., Naeem, K., and Shakoor, A. (2019). Simultaneous cutting of master reels and stocked rolls in solving trim loss minimization problem at paper mill. pages 147–151.

Bräsel, H., Herms, A., Mörig, M., Tautenhahn, T., Tusch, J., and Werner, F. (2008). Heuristic constructive algorithms for open shop scheduling to minimize mean flow time. *European Journal of Operational Research*, 189:856–870.

Framinan, J. M., Leisten, R., and García, R. R. (2014). Manufacturing scheduling systems: An integrated view on models, methods and tools. *Manufacturing Scheduling Systems: An Integrated View on Models, Methods and Tools*, 9781447162728:1–400.

Grinshpoun, T., Ilani, H., and Shufan, E. (1934). The representation of partially-concurrent open shop problems. *Ann Oper Res*, 252:455–469.

Han, Y., Gong, D., Jin, Y., and Pan, Q. (2019). Evolutionary multiobjective blocking lot-streaming flow shop scheduling with machine breakdowns. *IEEE Transactions on Cybernetics*, 49:184–197.

Han, Y., Li, J., Sang, H., Liu, Y., Gao, K., and Pan, Q. (2020). Discrete evolutionary multi-objective optimization for energy-efficient blocking flow shop scheduling with setup time. *Applied Soft Computing Journal*, 93.

He, F., Ma, X., and Jiang, M. (2019). Research on energy saving dispatching of forging link based on number of reheating times. volume 493.

Ho, M., Hnaien, F., and Dugardin, F. (2021). Electricity cost minimisation for optimal makespan solution in flow shop scheduling under time-of-use tariffs. *International Journal of Production Research*, 59:1041–1067.

Kelley, M., Baldick, R., and Baldea, M. (2019). Correction to: Demand response operation of electricity-intensive chemical processes for reduced greenhouse gas emissions: Application to an air separation unit (acs sustainable chemistry and engineering (2019) 7:2 (1909-1922) doi: 10.1021/acssuschemeng.8. *ACS Sustainable Chemistry and Engineering*, 7:9061.

Liu, C., Zhao, C., and Xu, Q. (2012). Integration of electroplating process design and operation for simultaneous productivity maximization, energy saving, and freshwater minimization. *Chemical Engineering Science*, 68:202–214.

Naderi, B., Ghomi, S. F., Aminnayeri, M., and Zandieh, M. (2010). A contribution and new heuristics for open shop scheduling. *Computers and Operations Research*, 37:213–221.

PNUD (2022). Programa de las naciones unidas para el desarrollo.

Repsol (2022). Eficiencia energética.

Shareh, M. B., Bargh, S. H., Hosseinabadi, A. A. R., and Slowik, A. (2021). An improved bat optimization algorithm to solve the tasks scheduling problem in open shop. *Neural Computing and Applications*, 33.

Shrouf, F., Ordieres-Meré, J., García-Sánchez, A., and Ortega-Mier, M. (2014). Optimizing the production scheduling of a single machine to minimize total energy consumption costs. *Journal of Cleaner Production*, 67:197–207.

Taillard, E. (1993). Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64:278–285.

Zhang, X., Liu, X., Tang, S., Królczyk, G., and Li, Z. (2019). Solving scheduling problem in a distributed manufacturing system using a discrete fruit fly optimization algorithm. *Energies*, 12.

Anexo

1 Primera Versión código

1.1 Código Heurística 1 para Makespan

En el siguiente código se presenta la heurística 1 para el Makespan.

```
#Cargamos la instancia
from scheptk.scheptk import OpenShop
instance=OpenShop('Instancia_3x3_desarrollo_1.txt')

#Función objetivo sostenibilidad
def FO_sostenibilidad(instance,sequence):
    #Antes de entrar a esta función los cij se ordenan de tal forma que en cada
         posición aparece el tiempo de finalización de la operación de esa
        misma posición
    #Esta primera parte de la función se encarga de poner la matriz de los
        tiempos de proceso con la misma estructura que la de los tiempos de
        finalización (ordenados de forma ascendente por filas)
    j_prima=0
    for i in range(instance.machines):
        for j in range(instance.jobs):
            j_prima=0
            for k in range(instance.jobs):
                if cij_ordenado_copia[i][j]==cij_ordenado_copia_ascendente[i][k]:

                    j_prima=k
                    processing_times_matriz[i][j_prima]=instance.pt[i][j]
    idle_time=0
    print('La secuencia evaluada es:', end=' ')
    print(sequence)
    for i in range(instance.machines):
        for j in range(instance.jobs-1):
            if(cij_ordenado_copia_ascendente[i][j]<cij_ordenado_copia_ascendente
                [i][j+1]-processing_times_matriz[i][j+1]):
                idle_time=idle_time+(cij_ordenado_copia_ascendente[i][j+1]-
                    processing_times_matriz[i][j+1])-
                    cij_ordenado_copia_ascendente[i][j]
    return idle_time

#Función que calcula la estructura general de los cij
def cij_Ordenado(cij,cj):
```

```
    #Creamos una matriz en la que vamos a almacenar los cij con la estructura
        general en la que en la posición [0][0] se encuentra el cij de la
        operación 0
    cij_ordenado_copia=[[0 for i in range(instance.jobs)] for j in range(
        instance.machines)]
    for i in range(instance.machines):
        for j in cj:
            cij_ordenado_copia[i][j]=cij[i][cj.index(j)]
    return cij_ordenado_copia

#Ordenamos los trabajos de la secuencia en orden decreciente
from scheptk.util import sorted_index_desc
#Este vector almacenará los tiempos de proceso lo que permitirá ordenar la
    secuencia
processing_times=[]
#Creamos una matriz en la que vamos a almacenar los tiempos de proceso con la
    misma estructura que la cij (ordenada de forma ascendiente por filas)
processing_times_matriz=[[0 for i in range(instance.jobs)] for j in range(
    instance.machines)]
#Establecemos M como cota superior
M=0
for i in range(instance.machines):
    for j in range(instance.jobs):
        processing_times.append(instance.pt[i][j])
        M=M+instance.pt[i][j]

#Este vector es el E de la H1
E=sorted_index_desc(processing_times)
print('Secuencia ordenada de los tiempos de proceso', end=' ')
print(E)

import numpy as np
#Creamos el vector W que inicialmente es un conjunto vacío
W=[]
#La variable r almacena el número de operaciones de la instancia
r=instance.machines*instance.jobs

for k in range(r):
    #Este vector almacena las funciones objetivos tras insertar una operación
        en distintas posiciones
    #Se inicializa a M para que, en caso de que sea redundante, no salga
        elegido
    Acum_FO=np.ones(shape=len(W)+1)*M
    Acum_FO=Acum_FO.astype(int)
    for z in range(k+1):
        redundancia=0
        #E[k]%instance.jobs indica el trabajo al que pertenece la operación E[k
            ]
        #W[z-1]%instance.jobs indica el trabajo al que pertenece la operación W
            [z-1]
        #E[k]//instance.jobs indica la máquina a la que pertenece la operación
            E[k]
        #W[z-1]//instance.jobs indica la máquina a la que pertenece la operació
            n W[z-1]
        if(z>0) and ((E[k]%instance.jobs)!=(W[z-1]%instance.jobs) and (E[k]//
            instance.jobs)!=(W[z-1]//instance.jobs)):
            redundancia=1
```

```
        if(redundancia==0):
            #Copio en el vector W_copia el vector W
            W_copia=W.copy()
            #Le inserto en la posición z la operación E[k]
            W_copia=np.insert(W_copia,z,E[k])
            W_copia=W_copia.astype(int)
            cij,cj= instance.ct(W_copia)
            #Llamamos a la función que calcula el cij con la estructura correcta
            cij_ordenado_copia=cij_Ordenado(cij,cj)
            #En el vector Acum_FO[z] voy acumulando las distintas funciones
                objetivos que voy obteniendo
            Acum_FO[z]=np.amax(cij_ordenado_copia)

    print('FO obtenidas:', end=' ')
    print(Acum_FO)
    #Establecemos el valor que se encuentra en la primera posición como mínimo
    FO_en_z=Acum_FO[0]
    posicion_menor_FO=0
    #Almacenamos la mejor función objetivo en FO_en_z y su posición
        posicion_menor_FO
    for p in range(k+1):
        if Acum_FO[p]<FO_en_z:
            posicion_menor_FO=p
            FO_en_z=Acum_FO[p]

    print('La posición en la que menor valor de la FO se ha obtenido es:', end=
        ' ')
    print(posicion_menor_FO)
    print('Insertando la operación actual en esa posición obtenemos:', end=' ')
    W=np.insert(W,posicion_menor_FO,E[k])
    print(W)
    W=W.astype(int)

print('COMPROBACIONES')
print("La secuencia final obtenida es:")
print(W)
#Muestra los cij de la librería
cij,cj=instance.ct(W)
print('Se muestran los cij y los cj')
cij=np.array(cij)
print(cij)
print(cj)

#Muestra los cij con la estructura general
cij_ordenado_copia=cij_Ordenado(cij,cj)
cij_ordenado_copia=np.array(cij_ordenado_copia)
print('Se muestran los cij y los cj ordenados')
print(cij_ordenado_copia)

#Muestra los cij ordenados de forma ascendente
cij_ordenado_copia_ascendente=cij_ordenado_copia.copy()
cij_ordenado_copia_ascendente=np.array(cij_ordenado_copia_ascendente)
cij_ordenado_copia_ascendente.sort(axis=1)
print('Se muestran los cij y los cj ordenados ascendiente')
print(cij_ordenado_copia_ascendente)

FO_final=FO_sostenibilidad(instance,W)
```

```
print('Su FO es:')
print(FO_final)

print('Finaliza en:')
print(np.amax(cij_ordenado_copia_ascendente))

gantt = instance.create_schedule(W)
gantt.print()
```

1.2 Código Heurística 2 para Makespan

En el siguiente código se presenta la heurística 2 para el Makespan.

```
#Cargamos la instancia
from scheptk.scheptk import OpenShop
instance=OpenShop('Instancia_3x3_desarrollo_1.txt')

#Función Objetivo Sostenibilidad
def FO_sostenibilidad(instance,sequence):
    #Antes de entrar a esta función los cij se ordenan de tal forma que en cada
         posición aparece el tiempo de finalización de la operación de esa
        misma posición
    #Esta primera parte de la función se encarga de poner la matriz de los
        tiempos de proceso con la misma estructura que la de los tiempos de
        finalización (ordenados de forma ascendente por filas)
    j_prima=0
    for i in range(instance.machines):
        for j in range(instance.jobs):
            j_prima=0
            for k in range(instance.jobs):
                if cij_ordenado_copia[i][j]==cij_ordenado_copia_ascendente[i][k]:

                    j_prima=k
                    processing_times_matriz[i][j_prima]=instance.pt[i][j]
    idle_time=0
    print('La secuencia evaluada es:', end=' ')
    print(sequence)
    for i in range(instance.machines):
        for j in range(instance.jobs-1):
            if(cij_ordenado_copia_ascendente[i][j]<cij_ordenado_copia_ascendente
                [i][j+1]-processing_times_matriz[i][j+1]):
                idle_time=idle_time+(cij_ordenado_copia_ascendente[i][j+1]-
                    processing_times_matriz[i][j+1])-
                    cij_ordenado_copia_ascendente[i][j]
    return idle_time

#Función que calcula la estructura general de los cij
def cij_Ordenado(cij,cj):
    #Creamos una matriz en la que vamos a almacenar los cij con la estructura
        general en la que en la posicion [0][0] se encuentra el cij de la
        operación 0
    cij_ordenado_copia=[[0 for i in range(instance.jobs)] for j in range(
        instance.machines)]
    for i in range(instance.machines):
        for j in cj:
            cij_ordenado_copia[i][j]=cij[i][cj.index(j)]
    return cij_ordenado_copia
```

```
#Ordenamos los trabajos de una secuencia en orden decreciente
from scheptk.util import sorted_index_desc
#Este vector almacenará los tiempos de proceso lo que permitirá ordenar la
    secuencia
processing_times=[]
#Creamos una matriz en la que vamos a almacenar los tiempos de proceso con la
    misma estructura que la cij ordenada de menor a mayor por filas
processing_times_matriz=[[0 for i in range(instance.jobs)] for j in range(
    instance.machines)]
#Establecemos M como cota superior
M=0
for i in range(instance.machines):
    for j in range(instance.jobs):
        processing_times.append(instance.pt[i][j])
        M=M+instance.pt[i][j]

#Este vector es el E de la H1
E=sorted_index_desc(processing_times)
print('Secuencia ordenada de los tiempos de proceso', end=' ')
print(E)

import numpy as np
#Creamos el vector W que inicialmente es un conjunto vacío
W=[]
#La variable r almacena el numero de operaciones de la instancia
r=instance.machines*instance.jobs

for k in range(r):
    #Este vector almacena las funciones objetivos tras insertar una operación
        en distintas posiciones
    #Se inicializa a M para que, en caso de que sea redundante, no salga
        elegido
    Acum_FO=np.ones(shape=len(W)+1)*M
    Acum_FO=Acum_FO.astype(int)
    for z in range(k+1):
        redundancia=0
        #E[k]%instance.jobs indica el trabajo al que pertenece la operación E[k
            ]
        #W[z-1]%instance.jobs indica el trabajo al que pertenece la operación W
            [z-1]
        #E[k]//instance.jobs indica la máquina a la que pertenece la operación
            E[k]
        #W[z-1]//instance.jobs indica la máquina a la que pertenece la operació
            n W[z-1]
        if(z>0) and ((E[k]%instance.jobs)!=(W[z-1]%instance.jobs) and (E[k]//
            instance.jobs)!=(W[z-1]//instance.jobs)):
            redundancia=1
        if(redundancia==0):
            #Copio en el vector W_copia el vector W
            W_copia=W.copy()
            #Le inserto en la posición z la operación E[k]
            W_copia=np.insert(W_copia,z,E[k])
            W_copia=W_copia.astype(int)
            cij,cj= instance.ct(W_copia)
            #Llamamos a la función que calcula el cij con la estructura correcta
            cij_ordenado_copia=cij_Ordenado(cij,cj)
```

```
            #En el vector Acum_FO[z] voy acumulando las distintas funciones
                objetivos que voy obteniendo
            Acum_FO[z]=np.amax(cij_ordenado_copia)

    print('FO obtenidas:', end=' ')
    print(Acum_FO)
    #Establecemos el valor que se encuentra en la primera posición como mínimo
    FO_en_z=Acum_FO[0]
    posicion_menor_FO=0
    #Almacenamos la mejor FO en FO_en_z y su posición posicion_menor_FO
    for p in range(k+1):
        if Acum_FO[p]<FO_en_z:
            posicion_menor_FO=p
            FO_en_z=Acum_FO[p]

    print('La posición en la que menor valor de la FO se ha obtenido es:', end=
        ' ')
    print(posicion_menor_FO)
    print('Insertando la operación actual en esa posición obtenemos:', end=' ')
    W=np.insert(W,posicion_menor_FO,E[k])
    print(W)
    W=W.astype(int)

print('COMPROBACIONES')
print("La secuencia final obtenida es:")
print(W)
#Muestra los cij de la librería
cij,cj=instance.ct(W)
print('Se muestran los cij y los cj')
cij=np.array(cij)
print(cij)
print(cj)

#Muestra los cij con la estructura general
cij_ordenado_copia=cij_Ordenado(cij,cj)
print('Se muestran los cij y los cj ordenados')
print(cij_ordenado_copia)

#Muestra los cij ordenados de forma ascendente
cij_ordenado_copia_ascendente=cij_ordenado_copia.copy()
cij_ordenado_copia_ascendente=np.array(cij_ordenado_copia_ascendente)
cij_ordenado_copia_ascendente.sort(axis=1)
print('Se muestran los cij y los cj ordenados ascendiente')
print(cij_ordenado_copia_ascendente)

FO_final=FO_sostenibilidad(instance,W)
print('Su FO es:')
print(FO_final)

#INICIO DE LA HEURÍSTICA 2
#Creamos el vector S que inicialmente es un conjunto vacío
S=[]
#El vector U esta formado por todas las operaciones
U=W.copy()
y=0

#Creamos una matriz que esta formada por las etiquetas
```

```
Etiquetas=[[0 for i in range(instance.jobs)] for j in range(instance.machines)]

a=-1
for i in range(instance.machines):
    for j in range(instance.jobs):
        Etiquetas[i][j]=a+1
        a=a+1

#La matriz Sij_inicio almacena los tiempos de inicio de las operaciones
Sij_inicio=[[0 for i in range (instance.machines)] for j in range(instance.jobs
    )]
print("Los valores de inicio son:")
Sij_inicio=np.array(Sij_inicio)
print(Sij_inicio)

W_copia=W.copy()

tiempo_minimo_inicio_filas=[[0 for i in range(instance.jobs)] for j in range(
    instance.machines)]
tiempo_minimo_inicio_columnas=[[0 for i in range(instance.jobs)] for j in range
    (instance.machines)]

while len(U)!=0:
    y=np.amin(Sij_inicio)
    print("El valor mínimo de inicio es: ", end='')
    print(y)
    #A continuación asignamos las operaciones cuyo valor sea igual a 'y' en R
    R=[]
    for i in range(instance.machines):
        for j in range(instance.jobs):
            if(Sij_inicio[i][j]==y):
                R.append(Etiquetas[i][j])

    print("El vector R es: ", end='')
    print(R)
    print("El vector U es: ", end='')
    print(U)
    print("El vector S es: ", end='')
    print(S)

    #Este bucle inserta la primera posición relativa en S y la elimina del
        vector U
    a=1
    for i in range(len(U)):
        if U[i] in R:
            print("La operación que sale del vector U y metemos en el S es:",
                end='')
            operacion_cambio=U[i]
            print(operacion_cambio)
            #Insertamos la primera posicion relativa en S
            S.append(operacion_cambio)
            #Eliminamos del vector U la primera posicion relativa que es la
                posicion i
            U=np.delete(U,i)
            a=0
        if(a==0):
            break
```

```
    cij,cj=instance.ct(S)
    print('Se muestran los cij y los cj del vector S')
    cij=np.array(cij)
    print(cij)
    print(cj)

    #Actualización de los tiempos de inicio de las operaciones
    for j in range(instance.jobs):
        tiempo_minimo_inicio_filas[operacion_cambio//instance.jobs][j]=
            cij_ordenado_copia[operacion_cambio//instance.jobs][operacion_cambio
            %instance.jobs]
    tiempo_minimo_inicio_filas=np.array(tiempo_minimo_inicio_filas)

    #Actualización de los tiempos de inicio de las operaciones
    for i in range(instance.machines):
        tiempo_minimo_inicio_columnas[i][operacion_cambio%instance.jobs]=
            cij_ordenado_copia[operacion_cambio//instance.jobs][operacion_cambio
            %instance.jobs]
    tiempo_minimo_inicio_columnas=np.array(tiempo_minimo_inicio_columnas)

    import numpy as geek
    Sij_inicio = geek.maximum(tiempo_minimo_inicio_filas,
        tiempo_minimo_inicio_columnas)
    print("Los valores de inicio Filas + Columnas son:")
    print(Sij_inicio)

    cij,cj=instance.ct(U)
    print('Se muestran los cij y los cj del vector U')
    cij=np.array(cij)
    print(cij)
    print(cj)

    #Muestra los cij con la estructura general
    cij_ordenado_copia=cij_Ordenado(cij,cj)
    cij_ordenado_copia=np.array(cij_ordenado_copia)
    print('Se muestran los cij y los cj ordenados del vector U')
    print(cij_ordenado_copia)

    for i in range(instance.machines):
        for j in range(instance.jobs):
            #Cuando saco una operacion de U, al calcular su finalización que no
                altere los mínimos
            if(cij_ordenado_copia[i][j]==0):
                Sij_inicio[i][j]=M
            if(j not in cj):
                Sij_inicio[i][j]=M

    print("Los valores de inicio actualizados son:")
    print(Sij_inicio)

print('COMPROBACIONES')
print("La secuencia final obtenida es:")
print(S)
#Muestra los cij de la librería
cij,cj=instance.ct(S)
print('Se muestran los cij y los cj')
```

```
cij=np.array(cij)
print(cij)
print(cj)

#Muestra los cij con la estructura general
cij_ordenado_copia=cij_Ordenado(cij,cj)
cij_ordenado_copia=np.array(cij_ordenado_copia)
print('Se muestran los cij y los cj ordenados')
print(cij_ordenado_copia)

#Muestra los cij ordenados de forma ascendente
cij_ordenado_copia_ascendente=cij_ordenado_copia.copy()
cij_ordenado_copia_ascendente=np.array(cij_ordenado_copia_ascendente)
cij_ordenado_copia_ascendente.sort(axis=1)
print('Se muestran los cij y los cj ordenados ascendiente')
print(cij_ordenado_copia_ascendente)

FO_final=FO_sostenibilidad(instance,S)
print('Su FO es:')
print(FO_final)

print('Finaliza en:')
print(np.amax(cij_ordenado_copia_ascendente))

gantt = instance.create_schedule(S)
gantt.print()
```

1.3 Código Heurística 3 para Makespan

En el siguiente código se presenta la heurística 3 para el Makespan.

```
#Cargamos la instancia
from scheptk.scheptk import OpenShop
instance=OpenShop('Instancia_3x3_desarrollo_1.txt')

#Función Objetivo Sostenibilidad
def FO_sostenibilidad(instance,sequence):
    #Antes de entrar a esta función los cij se ordenan de tal forma que en cada
        posición aparece el tiempo de finalización de la operación de esa
       misma posición
    #Esta primera parte de la función se encarga de poner la matriz de los
       tiempos de proceso con la misma estructura que la de los tiempos de
       finalización (ordenados de forma ascendente por filas)
    j_prima=0
    for i in range(instance.machines):
        for j in range(instance.jobs):
            j_prima=0
            for k in range(instance.jobs):
                if cij_ordenado_copia[i][j]==cij_ordenado_copia_ascendente[i][k]:

                    j_prima=k
                    processing_times_matriz[i][j_prima]=instance.pt[i][j]
    idle_time=0
    print('La secuencia evaluada es:', end=' ')
    print(sequence)
    for i in range(instance.machines):
        for j in range(instance.jobs-1):
```

```
            if(cij_ordenado_copia_ascendente[i][j]<cij_ordenado_copia_ascendente
                [i][j+1]-processing_times_matriz[i][j+1]):
                idle_time=idle_time+(cij_ordenado_copia_ascendente[i][j+1]-
                    processing_times_matriz[i][j+1])-
                    cij_ordenado_copia_ascendente[i][j]
    return idle_time

#Función que calcula la estructura general de los cij
def cij_Ordenado(cij,cj):
    #Creamos una matriz en la que vamos a almacenar los cij con la estructura
        general en la que en la posicion [0][0] se encuentra el cij de la
        operación 0
    cij_ordenado_copia=[[0 for i in range(instance.jobs)] for j in range(
        instance.machines)]
    for i in range(instance.machines):
        for j in cj:
            cij_ordenado_copia[i][j]=cij[i][cj.index(j)]
    return cij_ordenado_copia

#Ordenamos los trabajos de una secuencia en orden decreciente
from scheptk.util import sorted_index_desc
#Este vector almacenará los tiempos de proceso lo que permitirá ordenar la
    secuencia
processing_times=[]
#Creamos una matriz en la que vamos a almacenar los tiempos de proceso con la
    misma estructura que la cij ordenada de menor a mayor por filas
processing_times_matriz=[[0 for i in range(instance.jobs)] for j in range(
    instance.machines)]
#Establecemos M como cota superior
M=0
for i in range(instance.machines):
    for j in range(instance.jobs):
        processing_times.append(instance.pt[i][j])
        M=M+instance.pt[i][j]

#Este vector es el E de la H1
E=sorted_index_desc(processing_times)
print('Secuencia ordenada de los tiempos de proceso', end=' ')
print(E)

import numpy as np
#Creamos el vector W que inicialmente es un conjunto vacío
W=[]
#La variable r almacena el numero de operaciones de la instancia
r=instance.machines*instance.jobs

for k in range(r):
    #Este vector almacena las funciones objetivos tras insertar una operación
        en distintas posiciones
    #Se inicializa a M para que, en caso de que sea redundante, no salga
        elegido
    Acum_FO=np.ones(shape=len(W)+1)*M
    Acum_FO=Acum_FO.astype(int)
    for z in range(k+1):
        redundancia=0
        #E[k]%instance.jobs indica el trabajo al que pertenece la operación E[k
            ]
```

```
        #W[z-1]%instance.jobs indica el trabajo al que pertenece la operación W
            [z-1]
        #E[k]//instance.jobs indica la máquina a la que pertenece la operación
            E[k]
        #W[z-1]//instance.jobs indica la máquina a la que pertenece la operació
            n W[z-1]
        if(z>0) and ((E[k]%instance.jobs)!=(W[z-1]%instance.jobs) and (E[k]//
            instance.jobs)!=(W[z-1]//instance.jobs)):
            redundancia=1
        if(redundancia==0):
            #Copio en el vector W_copia el vector W
            W_copia=W.copy()
            #Le inserto en la posición z la operación E[k]
            W_copia=np.insert(W_copia,z,E[k])
            W_copia=W_copia.astype(int)

            cij,cj= instance.ct(W_copia)

            #Llamamos a la función que calcula el cij con la estructura correcta
            cij_ordenado_copia=cij_Ordenado(cij,cj)

            #En el vector Acum_FO[z] voy acumulando las distintas funciones
                objetivos que voy obteniendo
            Acum_FO[z]=np.amax(cij_ordenado_copia)

    print('FO obtenidas:', end=' ')
    print(Acum_FO)
    #Establecemos el valor que se encuentra en la primera posición como mínimo
    FO_en_z=Acum_FO[0]
    posicion_menor_FO=0

    #Almacenamos la mejor FO en FO_en_z y su posición posicion_menor_FO
    for p in range(k+1):
        if Acum_FO[p]<FO_en_z:
            posicion_menor_FO=p
            FO_en_z=Acum_FO[p]

    print('La posición en la que menor valor de la FO se ha obtenido es:', end=
        ' ')
    print(posicion_menor_FO)
    print('Insertando la operación actual en esa posición obtenemos:', end=' ')
    W=np.insert(W,posicion_menor_FO,E[k])
    print(W)
    W=W.astype(int)


print('COMPROBACIONES')
print("La secuencia final obtenida es:")
print(W)
#Muestra los cij de la librería
cij,cj=instance.ct(W)
print('Se muestran los cij y los cj')
cij=np.array(cij)
print(cij)
print(cj)

#Muestra los cij con la estructura general
```

```
cij_ordenado_copia=cij_Ordenado(cij,cj)
print('Se muestran los cij y los cj ORDENADOS')
print(cij_ordenado_copia)

#Muestra los cij ordenados de forma ascendente
cij_ordenado_copia_ascendente=cij_ordenado_copia.copy()
cij_ordenado_copia_ascendente=np.array(cij_ordenado_copia_ascendente)
cij_ordenado_copia_ascendente.sort(axis=1)
print('Se muestran los cij y los cj ordenados ascendiente')
print(cij_ordenado_copia_ascendente)

FO_final=FO_sostenibilidad(instance,W)
print('Su FO es:')
print(FO_final)

#INICIO DE LA HEURÍSTICA 2
#Creamos el vector S que inicialmente es un conjunto vacío
S=[]
#El vector U esta formado por todas las operaciones
U=W.copy()
y=0

#Creamos una matriz que esta formada por las etiquetas
Etiquetas=[[0 for i in range(instance.jobs)] for j in range(instance.machines)]

a=-1
for i in range(instance.machines):
    for j in range(instance.jobs):
        Etiquetas[i][j]=a+1
        a=a+1

#La matriz Sij_inicio almacena los tiempos de inicio de las operaciones
Sij_inicio=[[0 for i in range(instance.machines)] for j in range(instance.jobs)
    ]
print("Los valores de inicio son:")
Sij_inicio=np.array(Sij_inicio)
print(Sij_inicio)

W_copia=W.copy()

tiempo_minimo_inicio_filas=[[0 for i in range(instance.jobs)] for j in range(
    instance.machines)]
tiempo_minimo_inicio_columnas=[[0 for i in range(instance.jobs)] for j in range
    (instance.machines)]

while len(U)!=0:
    y=np.amin(Sij_inicio)
    print("El valor mínimo de inicio es: ", end='')
    print(y)

    #A continuación asignamos las operaciones cuyo valor sea igual a 'y' en R
    R=[]
    for i in range(instance.machines):
        for j in range(instance.jobs):
            if(Sij_inicio[i][j]==y):
                R.append(Etiquetas[i][j])
```

```
print("El vector R es: ", end='')
print(R)
print("El vector U es: ", end='')
print(U)
print("El vector S es: ", end='')
print(S)

#Este bucle inserta la primera posición relativa en S y la elimina del
    vector U
a=1
for i in range(len(U)):
    if U[i] in R:
        print("La operación que sale del vector U y metemos en el S es:",
            end='')
        operacion_cambio=U[i]
        print(operacion_cambio)
        #Insertamos la primera posición relativa en S
        S.append(operacion_cambio)
        #Eliminamos del vector U la primera posición relativa que es la
            posición i
        U=np.delete(U,i)
        a=0
    if(a==0):
        break

cij,cj=instance.ct(S)
print('Se muestran los cij y los cj del vector S')
cij=np.array(cij)
print(cij)
print(cj)

#Actualización de los tiempos de inicio de las operaciones
for j in range(instance.jobs):
    tiempo_minimo_inicio_filas[operacion_cambio//instance.jobs][j]=
        cij_ordenado_copia[operacion_cambio//instance.jobs][operacion_cambio
        %instance.jobs]
tiempo_minimo_inicio_filas=np.array(tiempo_minimo_inicio_filas)

#Actualización de los tiempos de inicio de las operaciones
for i in range(instance.machines):
    tiempo_minimo_inicio_columnas[i][operacion_cambio%instance.jobs]=
        cij_ordenado_copia[operacion_cambio//instance.jobs][operacion_cambio
        %instance.jobs]
tiempo_minimo_inicio_columnas=np.array(tiempo_minimo_inicio_columnas)

import numpy as geek
Sij_inicio = geek.maximum(tiempo_minimo_inicio_filas,
    tiempo_minimo_inicio_columnas)
print("Los valores de inicio Filas + Columnas son:")
print(Sij_inicio)

cij,cj=instance.ct(U)
print('Se muestran los cij y los cj del vector U')
cij=np.array(cij)
print(cij)
print(cj)
```

```
    #Muestra los cij con la estructura general
    cij_ordenado_copia=cij_Ordenado(cij,cj)
    cij_ordenado_copia=np.array(cij_ordenado_copia)
    print('Se muestran los cij y los cj ordenados del vector U')
    print(cij_ordenado_copia)

    for i in range(instance.machines):
        for j in range(instance.jobs):
            #Cuando saco una operación de U, al calcular su finalización que no
                altere los mínimos
            if(cij_ordenado_copia[i][j]==0):
                Sij_inicio[i][j]=M
            if(j not in cj):
                Sij_inicio[i][j]=M

    print("Los valores de inicio actualizados son:")
    print(Sij_inicio)

print('COMPROBACIONES')
print("La secuencia final obtenida es:")
print(S)
#Muestra los cij de la librería
cij,cj=instance.ct(S)
print('Se muestran los cij y los cj')
cij=np.array(cij)
print(cij)
print(cj)

#Muestra los cij con la estructura general
cij_ordenado_copia=cij_Ordenado(cij,cj)
cij_ordenado_copia=np.array(cij_ordenado_copia)
print('Se muestran los cij y los cj ordenados')
print(cij_ordenado_copia)

#Muestra los cij ordenados de forma ascendente
cij_ordenado_copia_ascendente=cij_ordenado_copia.copy()
cij_ordenado_copia_ascendente=np.array(cij_ordenado_copia_ascendente)
cij_ordenado_copia_ascendente.sort(axis=1)
print('Se muestran los cij y los cj ordenados ascendiente')
print(cij_ordenado_copia_ascendente)

FO_final=FO_sostenibilidad(instance,W)
print('Su FO es:')
print(FO_final)

#INICIO DE LA HEURÍSTICA H3
W=S.copy()
print("La secuencia obtenida resultante de la H2 es:")
print(W)
#Muestra los cij de la librería
cij,cj=instance.ct(W)
cij=np.array(cij)

#Muestra los cij con la estructura general
cij_ordenado_copia=cij_Ordenado(cij,cj)
cij_ordenado_copia=np.array(cij_ordenado_copia)
```

```
#Muestra los cij ordenados de forma ascendente
cij_ordenado_copia_ascendente=cij_ordenado_copia.copy()
cij_ordenado_copia_ascendente=np.array(cij_ordenado_copia_ascendente)
cij_ordenado_copia_ascendente.sort(axis=1)

make=np.amax(cij_ordenado_copia)
print('Su FO es:')
print(make)

improvement=1
while r>0:
    print('Comienza una nueva iteración')
    if(improvement==0):
        r=r-1

    improvement=0
    for z in range(instance.machines*instance.jobs):
        redundancia=0
        if(z>0) and ((E[k]%instance.jobs)!=(W[z-1]%instance.jobs) and (E[k]//
            instance.jobs)!=(W[z-1]//instance.jobs)):
            redundancia=1
        if(redundancia==0):
                WW_copia=W.copy()
                print("El vector copiado: ")
                print(WW_copia)
                print("El vector copiado tras eliminar el valor: ")
                WW_copia=np.delete(WW_copia, np.where(WW_copia == W[r-1]))
                print(WW_copia)
                print("El vector copiado tras insertarle el valor: ")
                WW_copia=np.insert(WW_copia,z,W[r-1])
                WW_copia = WW_copia.astype(int)
                print(WW_copia)
                cij,cj=instance.ct(WW_copia)
                cij=np.array(cij)
                #Muestra los cij con la estructura general
                cij_ordenado_copia=cij_Ordenado(cij,cj)
                cij_ordenado_copia=np.array(cij_ordenado_copia)
                makeWW=np.amax(cij_ordenado_copia)
                print("La FO es: ", end='')
                print(makeWW)
                if(makeWW<make):
                    improvement=1
                    make=makeWW
                    print("La FO ha mejorado")
                    Mejor_vector=WW_copia.copy()
    if(improvement==1):
        W=Mejor_vector.copy()
        print("El mejor vector resultante es:")
        print(W)
        r=instance.machines*instance.jobs

print('COMPROBACIONES')
print("La secuencia final obtenida es:")
print(W)
#Muestra los cij de la librería
cij,cj=instance.ct(W)
print('Se muestran los cij y los cj')
```

```
cij=np.array(cij)
print(cij)
print(cj)

#Muestra los cij con la estructura general
cij_ordenado_copia=cij_Ordenado(cij,cj)
cij_ordenado_copia=np.array(cij_ordenado_copia)
print('Se muestran los cij y los cj ordenados')
print(cij_ordenado_copia)

#Muestra los cij ordenados de forma ascendente
cij_ordenado_copia_ascendente=cij_ordenado_copia.copy()
cij_ordenado_copia_ascendente=np.array(cij_ordenado_copia_ascendente)
cij_ordenado_copia_ascendente.sort(axis=1)
print('Se muestran los cij y los cj ordenados ascendiente')
print(cij_ordenado_copia_ascendente)

FO_final=FO_sostenibilidad(instance,W)
print('Su FO sostenibilidad es:')
print(FO_final)

print('Finaliza en:')
print(np.amax(cij_ordenado_copia_ascendente))

gantt = instance.create_schedule(W)
gantt.print()
```

1.4 Código Heurística 4 para Makespan

En el siguiente código se presenta la heurística 4 para el Makespan.

```
#Cargamos la instancia
from scheptk.scheptk import OpenShop
instance=OpenShop('Instancia_3x3_desarrollo_1.txt')

#Función objetivo sostenibilidad
def FO_sostenibilidad(instance,sequence):
    #Antes de entrar a esta función los cij se ordenan de tal forma que en cada
        posición aparece el tiempo de finalización de la operación de esa
       misma posición
    #Esta primera parte de la función se encarga de poner la matriz de los
       tiempos de proceso con la misma estructura que la de los tiempos de
       finalización (ordenados de forma ascendente por filas)
    j_prima=0
    for i in range(instance.machines):
        for j in range(instance.jobs):
            j_prima=0
            for k in range(instance.jobs):
                if cij_ordenado_copia[i][j]==cij_ordenado_copia_ascendente[i][k]:

                    j_prima=k
                    processing_times_matriz[i][j_prima]=instance.pt[i][j]
    idle_time=0
    print('La secuencia evaluada es:', end=' ')
    print(sequence)
    for i in range(instance.machines):
        for j in range(instance.jobs-1):
```

```
            if(cij_ordenado_copia_ascendente[i][j]<cij_ordenado_copia_ascendente
                [i][j+1]-processing_times_matriz[i][j+1]):
                idle_time=idle_time+(cij_ordenado_copia_ascendente[i][j+1]-
                    processing_times_matriz[i][j+1])-
                    cij_ordenado_copia_ascendente[i][j]
    return idle_time

#Función que calcula la estructura general de los cij
def cij_Ordenado(cij,cj):
    #Creamos una matriz en la que vamos a almacenar los cij con la estructura
        general en la que en la posición [0][0] se encuentra el cij de la
        operacion 0
    cij_ordenado_copia=[[0 for i in range(instance.jobs)] for j in range(
        instance.machines)]
    for i in range(instance.machines):
        for j in cj:
            cij_ordenado_copia[i][j]=cij[i][cj.index(j)]
    return cij_ordenado_copia

#Ordenamos los trabajos de una secuencia en orden decreciente
from scheptk.util import sorted_index_desc
#Este vector almacenará los tiempos de proceso lo que permitirá ordenar la
    secuencia
processing_times=[]
#Creamos una matriz en la que vamos a almacenar los tiempos de proceso con la
    misma estructura que la cij ordenada de menor a mayor por filas
processing_times_matriz=[[0 for i in range(instance.jobs)] for j in range(
    instance.machines)]
#Establecemos M como cota superior
M=0
for i in range(instance.machines):
    for j in range(instance.jobs):
        processing_times.append(instance.pt[i][j])
        M=M+instance.pt[i][j]

#Este vector es el E de la H1
E=sorted_index_desc(processing_times)
print('Secuencia ordenada de los tiempos de proceso', end=' ')
print(E)

import numpy as np
#Creamos el vector W que inicialmente es un conjunto vacio
W=[]
#La variable r almacena el numero de operaciones de la instancia
r=instance.machines*instance.jobs

for k in range(r):
    #Este vector almacena las funciones objetivos tras insertar una operación
        en distintas posiciones
    #Se inicializa a M para que, en caso de que sea redundante, no salga
        elegido
    Acum_FO=np.ones(shape=len(W)+1)*M
    Acum_FO=Acum_FO.astype(int)
    for z in range(k+1):
        redundancia=0
        #E[k]%instance.jobs indica el trabajo al que pertenece la operación E[k
            ]
```

```
        #W[z-1]%instance.jobs indica el trabajo al que pertenece la operación W
            [z-1]
        #E[k]//instance.jobs indica la máquina a la que pertenece la operación
            E[k]
        #W[z-1]//instance.jobs indica la máquina a la que pertenece la operació
            n W[z-1]
        if(z>0) and ((E[k]%instance.jobs)!=(W[z-1]%instance.jobs) and (E[k]//
            instance.jobs)!=(W[z-1]//instance.jobs)):
            redundancia=1
        if(redundancia==0):
            #Copio en el vector W_copia el vector W
            W_copia=W.copy()
            #Le inserto en la posición z la operación E[k]
            W_copia=np.insert(W_copia,z,E[k])
            W_copia=W_copia.astype(int)
            cij,cj= instance.ct(W_copia)
            #Llamamos a la función que calcula el cij con la estructura correcta
            cij_ordenado_copia=cij_Ordenado(cij,cj)
            #En el vector Acum_FO[z] voy acumulando las distintas funciones
                objetivos que voy obteniendo
            Acum_FO[z]=np.amax(cij_ordenado_copia)

print('FO obtenidas:', end=' ')
print(Acum_FO)
#Establecemos el valor que se encuentra en la primera posición como mínimo
FO_en_z=Acum_FO[0]
posicion_menor_FO=0

#Almacenamos la mejor FO en FO_en_z y su posición posicion_menor_FO
for p in range(k+1):
    if Acum_FO[p]<FO_en_z:
        posicion_menor_FO=p
        FO_en_z=Acum_FO[p]

print('La posición en la que menor valor de la FO se ha obtenido es:', end=
    ' ')
print(posicion_menor_FO)
print('Insertando la operacion actual en esa posicion obtenemos:', end=' ')
W=np.insert(W,posicion_menor_FO,E[k])
print(W)
W=W.astype(int)

#Comparo los dos valores para establecer el inicio y el fin del bucle for
k_numero=5

if 1>posicion_menor_FO-k_numero:
    inicio=1
else:
    inicio=posicion_menor_FO-k_numero

if k<posicion_menor_FO+k_numero:
    fin=k
else:
    fin=posicion_menor_FO+k_numero

for r3 in range(inicio,fin):
    valor=W[r3]
```

```
        print("El vector W tras extraerle el valor es:")
        W=np.delete(W,np.where(W==W[r3]))
        print(W)
        Acum_FO_nueva=np.ones(shape=len(W)+1)*M
        Acum_FO_nueva=Acum_FO_nueva.astype(int)

        for r4 in range(k):
            redundancia=0
            if(r4>0) and ((E[r3]%instance.jobs)!=(W[r4-1]%instance.jobs) and (E[
                r3]//instance.jobs)!=(W[r4-1]//instance.jobs)):
                redundancia=1
            if(redundancia==0):
                W_copia_nueva=W.copy()
                #Le inserto en la posición r4 la operación valor
                W_copia_nueva=np.insert(W_copia_nueva,r4,valor)
                W_copia_nueva=W_copia_nueva.astype(int)

                cij,cj= instance.ct(W_copia_nueva)

                #Llamamos a la función que calcula el cij con la estructura
                    correcta
                cij_ordenado_copia=cij_Ordenado(cij,cj)

                #En el vector Acum_FO[z] voy acumulando las distintas funciones
                    objetivos que voy obteniendo
                Acum_FO_nueva[r4]=np.amax(cij_ordenado_copia)

        print('El vector con las diferentes FO obtenidas')
        print(Acum_FO_nueva)
        #Establecemos el valor que se encuentra en la primera posición como mí
            nimo
        FO_en_z_nueva=Acum_FO_nueva[0]
        posicion_menor_FO_nueva=0
        for p in range(k):
            if Acum_FO_nueva[p]<FO_en_z_nueva:
                posicion_menor_FO_nueva=p
                print(posicion_menor_FO_nueva)
                FO_en_z_nueva=Acum_FO_nueva[p]
                print(FO_en_z_nueva)

        print('La posición en la que menor valor de la FO se ha obtenido es:')
        print(posicion_menor_FO_nueva)
        print('La mejor FO que se ha obtenido es:')
        print(FO_en_z_nueva)
        print('Insertando la operación actual en esa posición obtenemos')
        W=np.insert(W,posicion_menor_FO_nueva,valor)
        print(W)
        W=W.astype(int)

print('COMPROBACIONES')
print("La secuencia final obtenida es:")
print(W)
#Muestra los cij de la librería
cij,cj=instance.ct(W)
print('Se muestran los cij y los cj')
cij=np.array(cij)
print(cij)
```

```
print(cj)

#Muestra los cij con la estructura general
cij_ordenado_copia=cij_Ordenado(cij,cj)
cij_ordenado_copia=np.array(cij_ordenado_copia)
print('Se muestran los cij y los cj ordenados')
print(cij_ordenado_copia)

#Muestra los cij ordenados de forma ascendente
cij_ordenado_copia_ascendente=cij_ordenado_copia.copy()
cij_ordenado_copia_ascendente=np.array(cij_ordenado_copia_ascendente)
cij_ordenado_copia_ascendente.sort(axis=1)
print('Se muestran los cij y los cj ordenados ascendiente')
print(cij_ordenado_copia_ascendente)

FO_final=FO_sostenibilidad(instance,W)
print('Su FO es:')
print(FO_final)

print('Finaliza en:')
print(np.amax(cij_ordenado_copia_ascendente))

gantt = instance.create_schedule(W)
gantt.print()
```

1.5 Código Función objetivo: Core Idle Time

Las función objetivo que se ha modificado en las heurísticas anteriores para el cálculo del Core Idle Time se muestra en el código que se presenta a continuación. Es la misma que se puede encontrar en la codificación expuesta anteriormente, la única diferencia es que a la hora de evaluar durante la ejecución cualquier secuencia, en vez de calcular el instante de finalización (tal y como se procede en la codificación mostrada anteriormente para calcular el Makespan) se llamaría a esta Función Objetivo de Sostenibilidad.

```
#Función objetivo sostenibilidad
def FO_sostenibilidad(instance,sequence):
    #Antes de entrar a esta función los cij se ordenan de tal forma que en cada
         posición aparece el tiempo de finalización de la operación de esa
        misma posición
    #Esta primera parte de la función se encarga de poner la matriz de los
        tiempos de proceso con la misma estructura que la de los tiempos de
        finalización (ordenados de forma ascendente por filas)
    j_prima=0
    for i in range(instance.machines):
        for j in range(instance.jobs):
            j_prima=0
            for k in range(instance.jobs):
                if cij_ordenado_copia[i][j]==cij_ordenado_copia_ascendente[i][k]:

                    j_prima=k
                    processing_times_matriz[i][j_prima]=instance.pt[i][j]

    idle_time=0
    print('La secuencia evaluada es:', end=' ')
    print(sequence)
    for i in range(instance.machines):
        for j in range(instance.jobs-1):
```

```
            if(cij_ordenado_copia_ascendente[i][j]<cij_ordenado_copia_ascendente
                [i][j+1]-processing_times_matriz[i][j+1]):
                idle_time=idle_time+(cij_ordenado_copia_ascendente[i][j+1]-
                    processing_times_matriz[i][j+1])-
                    cij_ordenado_copia_ascendente[i][j]
    return idle_time
```

1.6 Código Función objetivo: Ponderación Makespan y Core Idle Time

En el código que se muestra a continuación se presenta la función objetivo modificada para el cálculo de la ponderación de ambos objetivos.

```
#Función objetivo sostenibilidad ponderada
def FO_sostenibilidad(instance,sequence):
    #Antes de entrar a esta función los cij se ordenan de tal forma que en cada
         posición aparece el tiempo de finalización de la operación de esa
        misma posición
    #Esta primera parte de la función se encarga de poner la matriz de los
        tiempos de proceso con la misma estructura que la de los tiempos de
        finalización (ordenados de forma ascendente por filas)
    j_prima=0
    for i in range(instance.machines):
        for j in range(instance.jobs):
            j_prima=0
            for k in range(instance.jobs):
                if cij_ordenado_copia[i][j]==cij_ordenado_copia_ascendente[i][k]:

                    j_prima=k
                    processing_times_matriz[i][j_prima]=instance.pt[i][j]

    idle_time=0
    makespan=0
    print('La secuencia evaluada es:', end=' ')
    print(sequence)
    for i in range(instance.machines):
        for j in range(instance.jobs-1):
            if(cij_ordenado_copia_ascendente[i][j]<cij_ordenado_copia_ascendente
                [i][j+1]-processing_times_matriz[i][j+1]):
                idle_time=idle_time+(cij_ordenado_copia_ascendente[i][j+1]-
                    processing_times_matriz[i][j+1])-
                    cij_ordenado_copia_ascendente[i][j]
    #Una vez calculado el idle_time, procedemos a calcular el Makespan
    makespan=np.amax(cij_ordenado_copia)
    #Una vez conocemos ambas funciones objetivos, se ponderan en función del
        valor de lambda
    idle_time=idle_time*lambda+makespan*(1-lambda)
    return idle_time
```

2 Segunda Versión código

2.1 Código Heurística 1 para Makespan

En el siguiente código se presenta la heurística 1 para el Makespan.

```
#Cargamos la instancia
```

```
from scheptk.scheptk import OpenShop
#Ordenar trabajos de una secuencia en orden decreciente
from scheptk.util import sorted_index_desc

import numpy as np
import time
t_0=time.time()

instance=OpenShop('Instancia_modificada_1.txt')

#Función Objetivo Sostenibilidad
def FO_sostenibilidad(instance,sequence):
    #Esta primera parte de la función se encarga de poner la matriz de los
        tiempos de proceso con la misma estructura que la de los tiempos de
        finalización (ordenados de forma ascendente por filas)
    j_prima=0
    for i in range(instance.machines):
        for j in range(instance.jobs):
            j_prima=0
            for k in range(instance.jobs):
                if cij_ordenado_copia[i][j]==cij_ordenado_copia_ascendente[i][k]
                     and cij_ordenado_copia[i][j]>0:
                    j_prima=k
                    processing_times_matriz[i][j_prima]=instance.pt[i][j]
    idle_time=0
    for i in range(instance.machines):
        for j in range(0,instance.jobs-1):
            if(cij_ordenado_copia_ascendente[i][j]<cij_ordenado_copia_ascendente
                [i][j+1]-processing_times_matriz[i][j+1] and
                cij_ordenado_copia_ascendente[i][j]>0):
                idle_time=idle_time+(cij_ordenado_copia_ascendente[i][j+1]-
                    processing_times_matriz[i][j+1])-
                    cij_ordenado_copia_ascendente[i][j]
    return idle_time

#Función que calcula la estructura general de los cij
def cij_Ordenado(cij,cj):
    #Creamos una matriz en la que vamos a almacenar los cij con la estructura
        general en la que en la posición [0][0] se encuentra el cij de la
        operación 0
    cij_ordenado_copia=[[0 for i in range(instance.jobs)] for j in range(
        instance.machines)]
    for i in range(instance.machines):
        for j in cj:
            cij_ordenado_copia[i][j]=cij[i][cj.index(j)]
    return cij_ordenado_copia

#Este vector almacenará los tiempos de proceso lo que permitirá ordenar la
    secuencia
processing_times=[]
#Creamos una matriz en la que vamos a almacenar los tiempos de proceso con la
    misma estructura que la cij (ordenada de forma ascendiente por filas)
processing_times_matriz=[[0 for i in range(instance.jobs)] for j in range(
    instance.machines)]
#Establecemos M como cota superior
M=0
for i in range(instance.machines):
```

```
    for j in range(instance.jobs):
        processing_times.append(instance.pt[i][j])
        M=M+instance.pt[i][j]

#Este vector es el E de la H1
E=sorted_index_desc(processing_times)

#Creamos el vector W que inicialmente es un conjunto vacío
W=[]
#La variable r almacena el número de operaciones de la instancia
r=instance.machines*instance.jobs

for k in range(r):
    #Este vector almacena las funciones objetivos tras insertar una operación
        en distintas posiciones
    #Se inicializa a M para que, en caso de que sea redundante, no salga
        elegido
    Acum_FO=np.ones(shape=len(W)+1)*M
    Acum_FO=Acum_FO.astype(int)

    for z in range(k+1):
        redundancia=0
        #E[k]%instance.jobs indica el trabajo al que pertenece la operación E[k
            ]
        #W[z-1]%instance.jobs indica el trabajo al que pertenece la operación W
            [z-1]
        #E[k]//instance.jobs indica la máquina a la que pertenece la operación
            E[k]
        #W[z-1]//instance.jobs indica la máquina a la que pertenece la operació
            n W[z-1]
        if(z>1) and ((E[k]%instance.jobs)!=(W[z-1]%instance.jobs) and (E[k]//
            instance.jobs)!=(W[z-1]//instance.jobs)):
            redundancia=1

        if(redundancia==0):
            #Copio en el vector W_copia el vector W
            W_copia=W.copy()
            #Le inserto en la posición z la operación E[k]
            W_copia=np.insert(W_copia,z,E[k])
            W_copia=W_copia.astype(int)

            #Muestra los cij de la librería
            cij,cj= instance.ct(W_copia)
            #Llamamos a la función que calcula el cij con la estructura correcta
            cij_ordenado_copia=cij_Ordenado(cij,cj)
            #Ordenamos la matriz cij_ordenado_copia con la estructura ascendente
            cij_ordenado_copia=np.array(cij_ordenado_copia)
            cij_ordenado_copia_ascendente=cij_ordenado_copia.copy()
            cij_ordenado_copia_ascendente=np.array(cij_ordenado_copia_ascendente
                )
            cij_ordenado_copia_ascendente.sort(axis=1)

            #En el vector Acum_FO[z] voy acumulando las distintas funciones
                objetivos que voy obteniendo
            Acum_FO[z]=np.amax(cij_ordenado_copia_ascendente)
```

```
            #Estudio el tiempo actual y lo comparo con el inicial, si es
                superior al tiempo fijado, me salgo del bucle
            t_2=time.time()
            if(t_2-t_0>600):
                break

    print('FO obtenidas:', end=' ')
    print(Acum_FO)
    #Establecemos el valor que se encuentra en la primera posición como mínimo
    FO_en_z=Acum_FO[0]
    posicion_menor_FO=0
    #Almacenamos la mejor función objetivo en FO_en_z y su posición
        posicion_menor_FO
    for p in range(k):
        if Acum_FO[p]<FO_en_z:
            posicion_menor_FO=p
            FO_en_z=Acum_FO[p]

    print('La posición en la que menor valor de la FO se ha obtenido es:', end=
        ' ')
    print(posicion_menor_FO)
    print('Insertando la operación actual en esa posición obtenemos:', end=' ')
    W=np.insert(W,posicion_menor_FO,E[k])
    print(W)
    W=W.astype(int)

    #Estudio el tiempo actual y lo comparo con el inicial, si es superior al
        tiempo fijado, me salgo del bucle
    t_2=time.time()
    if(t_2-t_0>600):
        break

print('COMPROBACIONES')
print("La secuencia final obtenida es:")
print(W)
#Muestra los cij de la librería
cij,cj=instance.ct(W)
print('Se muestran los cij y los cj')
cij=np.array(cij)
print(cij)
print(cj)

#Muestra los cij con la estructura general
cij_ordenado_copia=cij_Ordenado(cij,cj)
cij_ordenado_copia=np.array(cij_ordenado_copia)
print('Se muestran los cij y los cj ORDENADOS')
print(cij_ordenado_copia)

#Muestra los cij ordenados de forma ascendente
cij_ordenado_copia_ascendente=cij_ordenado_copia.copy()
cij_ordenado_copia_ascendente=np.array(cij_ordenado_copia_ascendente)
cij_ordenado_copia_ascendente.sort(axis=1)
print('Se muestran los cij y los cj ORDENADOS ASCENDENTE')
print(cij_ordenado_copia_ascendente)

FO_final=FO_sostenibilidad(instance,W)
print('Su FO es:')
```

```
print(FO_final)

print('Finaliza en:')
print(np.amax(cij_ordenado_copia_ascendente))

t_1=time.time()
print("El tiempo de computo ha sido:")
print(t_1-t_0)

gantt = instance.create_schedule(W)
gantt.print()
```

2.2 Código Heurística 2 para Makespan

En el siguiente código se presenta la heurística 2 para el Makespan.

```
#Cargamos la instancia
from scheptk.scheptk import OpenShop
#Ordenamos los trabajos de una secuencia en orden decreciente
from scheptk.util import sorted_value_desc
#Ordenamos los trabajos de una secuencia en orden ascendiente
from scheptk.util import sorted_value_asc
#Ordenamos los trabajos de una secuencia en orden decreciente
from scheptk.util import sorted_index_desc

import numpy as np
import time
t_0=time.time()

instance=OpenShop('Instancia_modificada_1.txt')

#Función Objetivo Sostenibilidad
def FO_sostenibilidad(instance,sequence):
    #Esta primera parte de la función se encarga de poner la matriz de los
        tiempos de proceso con la misma estructura que la de los tiempos de
        finalización (ordenados de forma ascendente por filas)
    for i in range(instance.machines):
        for j in range(instance.jobs):
            j_prima=0
            for k in range(instance.jobs):
                if cij_ordenado_copia[i][j]==cij_ordenado_copia_ascendente[i][k]
                     and cij_ordenado_copia[i][j]>0:
                    j_prima=k
                    processing_times_matriz[i][j_prima]=instance.pt[i][j]
    idle_time=0
    for i in range(instance.machines):
        for j in range(0,instance.jobs-1):
            if(cij_ordenado_copia_ascendente[i][j]<cij_ordenado_copia_ascendente
                [i][j+1]-processing_times_matriz[i][j+1] and
                cij_ordenado_copia_ascendente[i][j]>0):
                idle_time=idle_time+(cij_ordenado_copia_ascendente[i][j+1]-
                    processing_times_matriz[i][j+1])-
                    cij_ordenado_copia_ascendente[i][j]
    return idle_time

#Función Cálculo Matriz Core Idle Time
def Calculo_matriz_CIT(instance,sequence):
```

```
    j_prima=0
    for i in range(instance.machines):
        for j in range(instance.jobs):
            j_prima=0
            for k in range(instance.jobs):
                if cij_ordenado_copia[i][j]==cij_ordenado_copia_ascendente[i][k]
                     and cij_ordenado_copia[i][j]>0:
                    j_prima=k
                    processing_times_matriz[i][j_prima]=instance.pt[i][j]
    matriz_CIT=[[0 for j in range(instance.jobs-1)] for i in range(instance.
        machines)]
    for i in range(instance.machines):
        for j in range(0,instance.jobs-1):
            if(cij_ordenado_copia_ascendente[i][j]<cij_ordenado_copia_ascendente
                [i][j+1]-processing_times_matriz[i][j+1] and
                cij_ordenado_copia_ascendente[i][j]>0):
                matriz_CIT[i][j]=cij_ordenado_copia_ascendente[i][j+1]-
                    processing_times_matriz[i][j+1]-
                    cij_ordenado_copia_ascendente[i][j]
    return matriz_CIT

#Función que calcula la estructura general de los cij
def cij_Ordenado(cij,cj):
    #Creamos una matriz en la que vamos a almacenar los cij con la estructura
        general en la que en la posición [0][0] se encuentra el cij de la
        operación 0
    cij_ordenado_copia=[[0 for j in range(instance.jobs)] for i in range(
        instance.machines)]
    for i in range(instance.machines):
        for j in cj:
            cij_ordenado_copia[i][j]=cij[i][cj.index(j)]
    return cij_ordenado_copia

#Creamos una matriz que esta formada por las etiquetas
Etiquetas=[[0 for j in range(instance.jobs)] for i in range(instance.machines)]
a=-1
for i in range(instance.machines):
    for j in range(instance.jobs):
        Etiquetas[i][j]=a+1
        a=a+1

#Este vector almacenará los tiempos de proceso lo que permitirá ordenar la
    secuencia
processing_times=[]
#Creamos una matriz en la que vamos a almacenar los tiempos de proceso con la
    misma estructura que la cij ordenada ascendiente por filas
processing_times_matriz=[[0 for i in range(instance.jobs)] for j in range(
    instance.machines)]
#Establecemos M como cota superior siendo igual a la suma de todos los tiempos
    de proceso
M=0
for i in range(instance.machines):
    for j in range(instance.jobs):
        processing_times.append(instance.pt[i][j])
        M=M+instance.pt[i][j]

#Este vector es el E de la MH1
```

```
E=sorted_index_desc(processing_times)

#Creamos el vector W que inicialmente es un conjunto vacío
W=[]
#La variable r almacena el número de operaciones de la instancia estudiada
r=instance.machines*instance.jobs

for k in range(r):
    #Este vector almacena las funciones objetivos tras insertar una operación
        en distintas posiciones
    #Se inicializa a M para que, en caso de que sea redundante, no salga
        elegido
    Acum_FO=np.ones(shape=len(W)+1)*M
    Acum_FO=Acum_FO.astype(int)

    for z in range(k+1):
        redundancia=0
        if(z>1) and ((E[k]%instance.jobs)!=(W[z-1]%instance.jobs) and (E[k]//
            instance.jobs)!=(W[z-1]//instance.jobs)):
            redundancia=1
        if(redundancia==0):
            #Copio en el vector W_copia el vector W
            W_copia=W.copy()
            #Le inserto en la posición z la operación E[k]
            W_copia=np.insert(W_copia,z,E[k])
            W_copia=W_copia.astype(int)

            #Actualizamos los tiempos de finalización
            cij,cj= instance.ct(W_copia)
            #Llamamos a la función que calcula el cij con la estructura correcta
            cij_ordenado_copia=cij_Ordenado(cij,cj)
            #Ordenamos la matriz cij_ordenado_copia con la estructura ascendente
            cij_ordenado_copia=np.array(cij_ordenado_copia)
            cij_ordenado_copia_ascendente=cij_ordenado_copia.copy()
            cij_ordenado_copia_ascendente=np.array(cij_ordenado_copia_ascendente
                )
            cij_ordenado_copia_ascendente.sort(axis=1)
            #En el vector Acum_FO[z] voy acumulando las distintas funciones
                objetivos que voy obteniendo
            Acum_FO[z]=np.amax(cij_ordenado_copia_ascendente)

            #Estudio el tiempo actual y lo comparo con el inicial, si es
                superior al tiempo fijado, me salgo del bucle
            t_2=time.time()
            if(t_2-t_0>600):
                break

    print('FO obtenidas:', end=' ')
    print(Acum_FO)
    #Establecemos el valor que se encuentra en la primera posición como mínimo
    FO_en_z=Acum_FO[0]
    posicion_menor_FO=0

    #Almacenamos la mejor FO en FO_en_z y su posición posicion_menor_FO
    for p in range(k):
        if Acum_FO[p]<FO_en_z:
            posicion_menor_FO=p
```

```
            FO_en_z=Acum_FO[p]

    print('La posición en la que menor valor de la FO se ha obtenido es:', end=
        ' ')
    print(posicion_menor_FO)
    print('Insertando la operación actual en esa posición obtenemos:', end=' ')
    W=np.insert(W,posicion_menor_FO,E[k])
    print(W)
    W=W.astype(int)

    #Estudio el tiempo actual y lo comparo con el inicial, si es superior al
        tiempo fijado, me salgo del bucle
    t_2=time.time()
    if(t_2-t_0>600):
        break

cij,cj=instance.ct(W)
cij=np.array(cij)
#Muestra los cij con la estructura general
cij_ordenado_copia=cij_Ordenado(cij,cj)
cij_ordenado_copia=np.array(cij_ordenado_copia)
#Muestra los cij ordenados de forma ascendente
cij_ordenado_copia_ascendente=cij_ordenado_copia.copy()
cij_ordenado_copia_ascendente=np.array(cij_ordenado_copia_ascendente)
cij_ordenado_copia_ascendente.sort(axis=1)

FO_final=FO_sostenibilidad(instance,W)

#Creamos el vector S que inicialmente es un conjunto vacío
S=[]
#El vector U esta formado por todas las operaciones
U=W.copy()
y=0

#La matriz Sij_inicio almacena los tiempos de inicio de las operaciones
Sij_inicio=[[0 for i in range(instance.machines)] for j in range(instance.jobs)
    ]
Sij_inicio_S=[[0 for i in range(instance.machines)] for j in range(instance.
    jobs)]
Sij_final_S=[[0 for i in range(instance.machines)] for j in range(instance.jobs
    )]

print("Los valores de inicio son:")
Sij_inicio=np.array(Sij_inicio)

#Inicialmente se calcula como la resta de cij_ordenado_copia menos los tiempos
    de proceso
for i in range(instance.machines):
    for j in range(instance.jobs):
        Sij_inicio[i][j]=cij_ordenado_copia[i][j]-instance.pt[i][j]

W_copia=W.copy()

while len(U)!=0:
    y=np.amin(Sij_inicio)
    print("El valor mínimo de inicio es: ", end='')
    print(y)
```

```
#A continuación asignamos las operaciones cuyo valor sea igual a 'y' en R
R=[]
for i in range(instance.machines):
    for j in range(instance.jobs):
        if(Sij_inicio[i][j]==y):
            R.append(Etiquetas[i][j])

print("El vector R es: ", end='')
print(R)
print("El vector U es: ", end='')
print(U)
print("El vector S es: ", end='')
print(S)

#Este bucle inserta la primera posición relativa en S y la elimina del
    vector U
a=1
operacion_cambio=0
for i in range(len(U)):
    if U[i] in R:
        print("La operación que sale del vector U y metemos en el S es:",
            end='')
        operacion_cambio=U[i]
        a=0
    if(a==0):
            break

#Muestra los cij de la librería
cij,cj=instance.ct(S)
cij=np.array(cij)
#Muestra los cij con la estructura general
cij_ordenado_copia=cij_Ordenado(cij,cj)
cij_ordenado_copia=np.array(cij_ordenado_copia)
#Muestra los cij ordenados de forma ascendente
cij_ordenado_copia_ascendente=cij_ordenado_copia.copy()
cij_ordenado_copia_ascendente=np.array(cij_ordenado_copia_ascendente)
cij_ordenado_copia_ascendente.sort(axis=1)

#Actualizamos la estructura de la matriz de los tiempos de proceso
for i in range(instance.machines):
    for j in range(instance.jobs):
        j_prima=0
        for k in range(instance.jobs):
            if cij_ordenado_copia[i][j]==cij_ordenado_copia_ascendente[i][k]
                 and cij_ordenado_copia[i][j]>0:
                j_prima=k
                processing_times_matriz[i][j_prima]=instance.pt[i][j]

matriz_CIT=Calculo_matriz_CIT(instance,S)
matriz_CIT=np.array(matriz_CIT)

#A continuación calculamos la finalización de las operaciones de ese
    trabajo en todas las máquinas
vector_finalizacion=[]
vector_inicio=[]
vector_inicio_ascendiente=[]
```

```
    print("El vector que almacena la finalización de las operaciones del
        trabajo es:")
    for i in range(instance.machines):
        vector_finalizacion.append(cij_ordenado_copia[i][operacion_cambio%
            instance.jobs])
    print(vector_finalizacion)

    #Ordenamos ese vector en orden decreciente
    vector_finalizacion_descendiente=sorted_value_desc(vector_finalizacion)
    print("El vector que almacena la finalización de las operaciones del
        trabajo ordenado descendiente es:")
    print(vector_finalizacion_descendiente)

    #Una vez tenemos el vector de finalización podemos hallar el de inicio rest
        ándole el tiempo de proceso
    for i in range(instance.machines):
        vector_inicio.append(vector_finalizacion[i]-instance.pt[i][
            operacion_cambio%instance.jobs])
    print(vector_inicio)

    #Ordenamos ese vector en orden ascendiente
    vector_inicio_ascendiente=sorted_value_asc(vector_inicio)
    print("El vector que almacena el inicio de las operaciones del trabajo
        ordenado ascendiente es:")
    print(vector_inicio_ascendiente)

    #Estudio el tiempo actual y lo comparo con el inicial, si es superior al
        tiempo fijado, me salgo del bucle
    t_2=time.time()
    if(t_2-t_0>600):
        break

    Cj_INI_CIT=0
    Cj_FIN_CIT=0
    #Comprobamos que el hueco disponible del CIT sea mayor a la duración de la
        operación que se desea insertar
    for j in range(instance.jobs-1):

        if(matriz_CIT[operacion_cambio//instance.jobs][j]>instance.pt[
            operacion_cambio//instance.jobs][operacion_cambio%instance.jobs]):
            #En este punto necesito almacenar Cj_INI_CIT y Cj_FIN_CIT
            Cj_INI_CIT=cij_ordenado_copia_ascendente[operacion_cambio//instance.
                jobs][j]
            Cj_FIN_CIT=cij_ordenado_copia_ascendente[operacion_cambio//instance.
                jobs][j+1]-processing_times_matriz[operacion_cambio//instance.
                jobs][j+1]

            #Esta variable almacena el tiempo de inicio de la operación del
                mismo trabajo que se desea insertar
            primera_operacion_superior=0
            #Esta variable almacena el tiempo de finalización de la operación
                del mismo trabajo que se desea insertar
            primera_operacion_inferior=0

            #Establezco contadores que me proporcionan el número de veces que se
                 cumple cada una de las condiciones
            contador_nulo=0
```

```
            contador_nulo_fijo=0
            contador_fijo=0
            contador_inicio_fijo=0
            contador_final_fijo=0
            contador_medio_inicio_fijo=0
            contador_medio_final_fijo=0

            for i in range(instance.machines):

                #Calculo del primer inicio de operación superior de ese trabajo
                for k in range(instance.machines):
                    if vector_inicio_ascendiente[k]>=vector_finalizacion[i]:
                        primera_operacion_superior=vector_inicio_ascendiente[k]
                    break

                #Calculo del primer fin de operación inferior de ese trabajo
                for k in range(instance.machines):
                    if vector_finalizacion_descendiente[k]>=vector_finalizacion[
                        i]:
                        primera_operacion_inferior=
                            vector_finalizacion_descendiente[k]
                    break

                #Estudio del caso en el que una operación del mismo trabajo de
                    otra máquina se introduzca parcialmente en el inicio
                if(vector_finalizacion[i]-instance.pt[i][operacion_cambio%
                    instance.jobs]<Cj_INI_CIT and vector_finalizacion[i]>
                    Cj_INI_CIT and vector_finalizacion[i]<Cj_FIN_CIT):
                    primertermino_inicio=max(Cj_INI_CIT,vector_finalizacion[i])+
                        instance.pt[operacion_cambio//instance.jobs][
                        operacion_cambio%instance.jobs]
                    segundotermino_inicio=min(Cj_FIN_CIT,
                        primera_operacion_superior)
                    if(primertermino_inicio<=segundotermino_inicio):
                        contador_inicio_fijo=1

                #Estudio del caso en el que una operación del mismo trabajo de
                    otra máquina se introduzca parcialmente en el final
                if(vector_finalizacion[i]-instance.pt[i][operacion_cambio%
                    instance.jobs]<Cj_FIN_CIT and vector_finalizacion[i]-
                    instance.pt[i][operacion_cambio%instance.jobs]>Cj_INI_CIT
                    and vector_finalizacion[i]>Cj_FIN_CIT):
                    primertermino_final=min(Cj_FIN_CIT,vector_finalizacion[i]-
                        instance.pt[i][operacion_cambio%instance.jobs])-instance.
                        pt[operacion_cambio//instance.jobs][operacion_cambio%
                        instance.jobs]
                    segundotermino_final=max(Cj_INI_CIT,
                        primera_operacion_inferior)
                    if(primertermino_final>=segundotermino_final):
                        print(primera_operacion_inferior)
                        contador_final_fijo=1

                #Estudio del caso en el que una operación del mismo trabajo de
                    otra máquina se encuentre en medio del CIT
                if(vector_finalizacion[i]-instance.pt[i][operacion_cambio%
                    instance.jobs]>Cj_INI_CIT and vector_finalizacion[i]<
                    Cj_FIN_CIT):
```

```
                primertermino_medio_final=vector_finalizacion[i]+instance.pt[
                    operacion_cambio//instance.jobs][operacion_cambio%
                    instance.jobs]
                segundotermino_medio_final=min(Cj_FIN_CIT,
                    primera_operacion_superior)
                if(primertermino_medio_final<=segundotermino_medio_final):
                    contador_medio_final_fijo=1

                primertermino_medio_inicio=vector_finalizacion[i]-instance.pt
                    [i][operacion_cambio%instance.jobs]-instance.pt[
                    operacion_cambio//instance.jobs][operacion_cambio%
                    instance.jobs]
                segundotermino_medio_inicio=max(Cj_INI_CIT,
                    primera_operacion_inferior)
                if(primertermino_medio_inicio>=segundotermino_medio_inicio):
                    contador_medio_inicio_fijo=1

            #Estudio del caso en el que no se solape con ninguna operación
            if(vector_finalizacion[i]<Cj_INI_CIT or vector_inicio[i]>
                Cj_FIN_CIT):
                contador_fijo=contador_fijo+1

            if(vector_finalizacion[i]>Cj_FIN_CIT and vector_inicio[i]<
                Cj_INI_CIT):
                contador_nulo_fijo=1

        if(contador_inicio_fijo==1 and contador_nulo_fijo==0):
            Sij_inicio_S[operacion_cambio//instance.jobs][operacion_cambio%
                instance.jobs]=max(Cj_INI_CIT,vector_finalizacion[i])
            break

        if(contador_final_fijo==1 and contador_nulo_fijo==0):
            Sij_inicio_S[operacion_cambio//instance.jobs][operacion_cambio%
                instance.jobs]=segundotermino_final
            break

        if(contador_medio_inicio_fijo==1 and contador_nulo_fijo==0):
            Sij_inicio_S[operacion_cambio//instance.jobs][operacion_cambio%
                instance.jobs]=segundotermino_medio_inicio
            break

        if(contador_medio_final_fijo==1 and contador_nulo_fijo==0):
            Sij_inicio_S[operacion_cambio//instance.jobs][operacion_cambio%
                instance.jobs]=vector_finalizacion[i]
            break

        if(contador_fijo==instance.machines):
            Sij_inicio_S[operacion_cambio//instance.jobs][operacion_cambio%
                instance.jobs]=Cj_INI_CIT
        break

a=1
operacion_cambio=0
for i in range(len(U)):
    if U[i] in R:
        print("La operación que sale del vector U y metemos en el S es:",
            end='')
```

```
            operacion_cambio=U[i]
            #Insertamos la primera posición relativa en S
            S.append(operacion_cambio)
            #Eliminamos del vector U la primera posición relativa que es la
                posición i
            U=np.delete(U,i)
            a=0
        if(a==0):
                break

    #Estudio el tiempo actual y lo comparo con el inicial, si es superior al
        tiempo fijado, me salgo del bucle
    t_2=time.time()
    if(t_2-t_0>600):
        break

    #Actualizo los tiempos de finalización del vector U para la que salgan los
        tiempos de inicio actualizados en la siguiente iteración
    cij,cj=instance.ct(U)
    cij=np.array(cij)
    #Muestra los cij con la estructura general
    cij_ordenado_copia=cij_Ordenado(cij,cj)
    cij_ordenado_copia=np.array(cij_ordenado_copia)
    #Muestra los cij ordenados de forma ascendente
    cij_ordenado_copia_ascendente=cij_ordenado_copia.copy()
    cij_ordenado_copia_ascendente=np.array(cij_ordenado_copia_ascendente)
    cij_ordenado_copia_ascendente.sort(axis=1)
    #Se calculan los tiempos de proceso con la estructura correcta

    j_prima=0
    for i in range(instance.machines):
        for j in range(instance.jobs):
            j_prima=0
            for k in range(instance.jobs):
                if cij_ordenado_copia[i][j]==cij_ordenado_copia_ascendente[i][k]
                     and cij_ordenado_copia[i][j]>0:
                    j_prima=k
                    processing_times_matriz[i][j_prima]=instance.pt[i][j]

    #Hallamos los tiempos de inicio
    for i in range(instance.machines):
        for j in range(instance.jobs):
            Sij_inicio[i][j]=cij_ordenado_copia[i][j]-instance.pt[i][j]

    for i in range(instance.machines):
        for j in range(instance.jobs):
            #Cuando saco una operación de U, al calcular su finalización para
                que no altere los mínimos se iguala a una cota superior
            if(cij_ordenado_copia[i][j]==0):
                Sij_inicio[i][j]=M
            if(j not in cj):
                Sij_inicio[i][j]=M

print('COMPROBACIONES')
print("La secuencia final obtenida es:")
print(S)
#Muestra los cij de la librería
```

```
cij,cj=instance.ct(S)
print('Se muestran los cij y los cj')
cij=np.array(cij)
print(cij)
print(cj)
#Muestra los cij con la estructura general
cij_ordenado_copia=cij_Ordenado(cij,cj)
cij_ordenado_copia=np.array(cij_ordenado_copia)
print('Se muestran los cij y los cj ORDENADOS')
print(cij_ordenado_copia)
#Muestra los cij ordenados de forma ascendente
cij_ordenado_copia_ascendente=cij_ordenado_copia.copy()
cij_ordenado_copia_ascendente=np.array(cij_ordenado_copia_ascendente)
cij_ordenado_copia_ascendente.sort(axis=1)
print('Se muestran los cij y los cj ORDENADOS ASCENDENTE')
print(cij_ordenado_copia_ascendente)

print('Se muestra el instante de inicio de aquellas operaciones que rompre la
    secuencia')
Sij_inicio_S=np.array(Sij_inicio_S)
print(Sij_inicio_S)


#A continuación creamos una matriz que almacena el instante de finalización de
    Sij_inicio_S
for i in range(instance.machines):
    for j in range(instance.jobs):
        if Sij_inicio_S[i][j]!=0:
            Sij_final_S[i][j]=Sij_inicio_S[i][j]+instance.pt[i][j]

Sij_final_S=np.array(Sij_final_S)
print('Se muestra el instante de fin de aquellas operaciones que rompre la
    secuencia')
print(Sij_final_S)

j_prima=0
for i in range(instance.machines):
    for j in range(instance.jobs):
        j_prima=0
        for k in range(instance.jobs):
            if cij_ordenado_copia[i][j]==cij_ordenado_copia_ascendente[i][k] and
                 cij_ordenado_copia[i][j]>0:
                j_prima=k
                processing_times_matriz[i][j_prima]=instance.pt[i][j]

print("Tiempos de procesos coincidente con vector ascendente")
processing_times_matriz=np.array(processing_times_matriz)
print(processing_times_matriz)

FO_final=FO_sostenibilidad(instance,S)
print('Su FO es:')
print(FO_final)

#A continuación creamos una matriz que almacena el instante de finalización
    correspondiente entre Sij_final y Cij
for i in range(instance.machines):
    for j in range(instance.jobs):
```

```
        if Sij_final_S[i][j]!=0:
            cij_ordenado_copia[i][j]=Sij_final_S[i][j]

print('Finaliza en:')
print(np.amax(cij_ordenado_copia))

t_1=time.time()
print("El tiempo de computo ha sido:")
print(t_1-t_0)

gantt=instance.create_schedule(S)
gantt.print()
```

2.3 Código Heurística 3 para Makespan

En el siguiente código se presenta la heurística 3 para el Makespan.

```
#Cargamos la instancia
from scheptk.scheptk import OpenShop
#Ordenamos los trabajos de una secuencia en orden decreciente
from scheptk.util import sorted_value_desc
#Ordenamos los trabajos de una secuencia en orden ascendiente
from scheptk.util import sorted_value_asc
#Ordenamos los trabajos de una secuencia en orden decreciente
from scheptk.util import sorted_index_desc

import numpy as np
import time
t_0=time.time()

instance=OpenShop('Instancia_modificada_1.txt')

#Función Objetivo Sostenibilidad
def FO_sostenibilidad(instance,sequence):
    #Esta primera parte de la función se encarga de poner la matriz de los
        tiempos de proceso con la misma estructura que la de los tiempos de
        finalización (ordenados de forma ascendente por filas)
    for i in range(instance.machines):
        for j in range(instance.jobs):
            j_prima=0
            for k in range(instance.jobs):
                if cij_ordenado_copia[i][j]==cij_ordenado_copia_ascendente[i][k]
                     and cij_ordenado_copia[i][j]>0:
                    j_prima=k
                    processing_times_matriz[i][j_prima]=instance.pt[i][j]
    idle_time=0
    for i in range(instance.machines):
        for j in range(0,instance.jobs-1):
            if(cij_ordenado_copia_ascendente[i][j]<cij_ordenado_copia_ascendente
                [i][j+1]-processing_times_matriz[i][j+1] and
                cij_ordenado_copia_ascendente[i][j]>0):
                idle_time=idle_time+(cij_ordenado_copia_ascendente[i][j+1]-
                    processing_times_matriz[i][j+1])-
                    cij_ordenado_copia_ascendente[i][j]
    return idle_time

#Función Cálculo Matriz Core Idle Time
```

```
def Calculo_matriz_CIT(instance,sequence):
    #Esta primera parte de la función se encarga de poner la matriz de los
        tiempos de proceso con la misma estructura que la de los tiempos de
        finalización (ordenados de forma ascendente por filas)
    j_prima=0
    for i in range(instance.machines):
        for j in range(instance.jobs):
            j_prima=0
            for k in range(instance.jobs):
                if cij_ordenado_copia[i][j]==cij_ordenado_copia_ascendente[i][k]
                     and cij_ordenado_copia[i][j]>0:
                    j_prima=k
                    processing_times_matriz[i][j_prima]=instance.pt[i][j]
    matriz_CIT=[[0 for j in range(instance.jobs-1)] for i in range(instance.
        machines)]
    for i in range(instance.machines):
        for j in range(0,instance.jobs-1):
            if(cij_ordenado_copia_ascendente[i][j]<cij_ordenado_copia_ascendente
                [i][j+1]-processing_times_matriz[i][j+1] and
                cij_ordenado_copia_ascendente[i][j]>0):
                matriz_CIT[i][j]=cij_ordenado_copia_ascendente[i][j+1]-
                    processing_times_matriz[i][j+1]-
                    cij_ordenado_copia_ascendente[i][j]
    return matriz_CIT

#Función que calcula la estructura general de los cij
def cij_Ordenado(cij,cj):
    #Creamos una matriz en la que vamos a almacenar los cij con la estructura
        general en la que en la posición [0][0] se encuentra el cij de la
        operación 0
    cij_ordenado_copia=[[0 for j in range(instance.jobs)] for i in range(
        instance.machines)]
    for i in range(instance.machines):
        for j in cj:
            cij_ordenado_copia[i][j]=cij[i][cj.index(j)]
    return cij_ordenado_copia

#Creamos una matriz que esta formada por las etiquetas
Etiquetas=[[0 for j in range(instance.jobs)] for i in range(instance.machines)]
a=-1
for i in range(instance.machines):
    for j in range(instance.jobs):
        Etiquetas[i][j]=a+1
        a=a+1

#Este vector almacenará los tiempos de proceso lo que permitirá ordenar la
    secuencia
processing_times=[]
#Creamos una matriz en la que vamos a almacenar los tiempos de proceso con la
    misma estructura que la cij ordenada ascendiente por filas
processing_times_matriz=[[0 for i in range(instance.jobs)] for j in range(
    instance.machines)]
#Establecemos M como cota superior siendo igual a la suma de todos los tiempos
    de proceso
M=0
for i in range(instance.machines):
    for j in range(instance.jobs):
```

```
        processing_times.append(instance.pt[i][j])
        M=M+instance.pt[i][j]

#Este vector es el E de la MH1
E=sorted_index_desc(processing_times)

#Creamos el vector W que inicialmente es un conjunto vacío
W=[]
#La variable r almacena el número de operaciones de la instancia estudiada
r=instance.machines*instance.jobs

for k in range(r):
    #Este vector almacena las funciones objetivos tras insertar una operación
        en distintas posiciones
    #Se inicializa a M para que, en caso de que sea redundante, no salga
        elegido
    Acum_FO=np.ones(shape=len(W)+1)*M
    Acum_FO=Acum_FO.astype(int)

    for z in range(k+1):
        redundancia=0
        if(z>1) and ((E[k]%instance.jobs)!=(W[z-1]%instance.jobs) and (E[k]//
            instance.jobs)!=(W[z-1]//instance.jobs)):
            redundancia=1
        if(redundancia==0):
            #Copio en el vector W_copia el vector W
            W_copia=W.copy()
            #Le inserto en la posición z la operación E[k]
            W_copia=np.insert(W_copia,z,E[k])
            W_copia=W_copia.astype(int)

            #Actualizamos los tiempos de finalización
            cij,cj= instance.ct(W_copia)
            #Llamamos a la función que calcula el cij con la estructura correcta
            cij_ordenado_copia=cij_Ordenado(cij,cj)
            #Ordenamos la matriz cij_ordenado_copia con la estructura ascendente
            cij_ordenado_copia=np.array(cij_ordenado_copia)
            cij_ordenado_copia_ascendente=cij_ordenado_copia.copy()
            cij_ordenado_copia_ascendente=np.array(cij_ordenado_copia_ascendente
                )
            cij_ordenado_copia_ascendente.sort(axis=1)
            #En el vector Acum_FO[z] voy acumulando las distintas funciones
                objetivos que voy obteniendo
            Acum_FO[z]=np.amax(cij_ordenado_copia_ascendente)

            #Estudio el tiempo actual y lo comparo con el inicial, si es
                superior al tiempo fijado, me salgo del bucle
            t_2=time.time()
            if(t_2-t_0>600):
                break

    print('FO obtenidas:', end=' ')
    print(Acum_FO)
    #Establecemos el valor que se encuentra en la primera posición como mínimo
    FO_en_z=Acum_FO[0]
    posicion_menor_FO=0
```

```
    #Almacenamos la mejor FO en FO_en_z y su posición posicion_menor_FO
    for p in range(k):
        if Acum_FO[p]<FO_en_z:
            posicion_menor_FO=p
            FO_en_z=Acum_FO[p]

    print('La posición en la que menor valor de la FO se ha obtenido es:', end=
        ' ')
    print(posicion_menor_FO)
    print('Insertando la operación actual en esa posición obtenemos:', end=' ')
    W=np.insert(W,posicion_menor_FO,E[k])
    print(W)
    W=W.astype(int)

    #Estudio el tiempo actual y lo comparo con el inicial, si es superior al
        tiempo fijado, me salgo del bucle
    t_2=time.time()
    if(t_2-t_0>600):
        break

#Muestra los cij de la librería
cij,cj=instance.ct(W)
cij=np.array(cij)
#Muestra los cij con la estructura general
cij_ordenado_copia=cij_Ordenado(cij,cj)
cij_ordenado_copia=np.array(cij_ordenado_copia)
#Muestra los cij ordenados de forma ascendente
cij_ordenado_copia_ascendente=cij_ordenado_copia.copy()
cij_ordenado_copia_ascendente=np.array(cij_ordenado_copia_ascendente)
cij_ordenado_copia_ascendente.sort(axis=1)

FO_final=FO_sostenibilidad(instance,W)

#Creamos el vector S que inicialmente es un conjunto vacío
S=[]
#El vector U esta formado por todas las operaciones
U=W.copy()
y=0

#La matriz Sij_inicio almacena los tiempos de inicio de las operaciones
Sij_inicio=[[0 for i in range(instance.machines)] for j in range(instance.jobs)
    ]
Sij_inicio_S=[[0 for i in range(instance.machines)] for j in range(instance.
    jobs)]
Sij_final_S=[[0 for i in range(instance.machines)] for j in range(instance.jobs
    )]

Sij_inicio=np.array(Sij_inicio)

#Inicialmente se calcula como la resta de cij_ordenado_copia menos los tiempos
    de proceso
for i in range(instance.machines):
    for j in range(instance.jobs):
        Sij_inicio[i][j]=cij_ordenado_copia[i][j]-instance.pt[i][j]

W_copia=W.copy()
```

```
while len(U)!=0:
    #Estudio el tiempo actual y lo comparo con el inicial, si es superior al
        tiempo fijado, me salgo del bucle
    t_2=time.time()
    if(t_2-t_0>600):
        break

    y=np.amin(Sij_inicio)

    #A continuación asignamos las operaciones cuyo valor sea igual a 'y' en R
    R=[]
    for i in range(instance.machines):
        for j in range(instance.jobs):
            if(Sij_inicio[i][j]==y):
                R.append(Etiquetas[i][j])

    print("El vector R es: ", end='')
    print(R)
    print("El vector U es: ", end='')
    print(U)
    print("El vector S es: ", end='')
    print(S)

    #Este bucle inserta la primera posición relativa en S y la elimina del
        vector U
    a=1
    operacion_cambio=0
    for i in range(len(U)):
        if U[i] in R:
            operacion_cambio=U[i]
            a=0
        if(a==0):
                break

    #Muestra los cij de la librería
    cij,cj=instance.ct(S)
    cij=np.array(cij)
    #Muestra los cij con la estructura general
    cij_ordenado_copia=cij_Ordenado(cij,cj)
    cij_ordenado_copia=np.array(cij_ordenado_copia)
    #Muestra los cij ordenados de forma ascendente
    cij_ordenado_copia_ascendente=cij_ordenado_copia.copy()
    cij_ordenado_copia_ascendente=np.array(cij_ordenado_copia_ascendente)
    cij_ordenado_copia_ascendente.sort(axis=1)

    #Actualizamos la estructura de la matriz de los tiempos de proceso
    for i in range(instance.machines):
        for j in range(instance.jobs):
            j_prima=0
            for k in range(instance.jobs):
                if cij_ordenado_copia[i][j]==cij_ordenado_copia_ascendente[i][k]
                     and cij_ordenado_copia[i][j]>0:
                    j_prima=k
                    processing_times_matriz[i][j_prima]=instance.pt[i][j]

    matriz_CIT=Calculo_matriz_CIT(instance,S)
    matriz_CIT=np.array(matriz_CIT)
```

```
#A continuación calculamos la finalización de las operaciones de ese
    trabajo en todas las máquinas
vector_finalizacion=[]
vector_inicio=[]
vector_inicio_ascendiente=[]
print("El vector que almacena la finalización de las operaciones del
    trabajo es:")
for i in range(instance.machines):
    vector_finalizacion.append(cij_ordenado_copia[i][operacion_cambio%
        instance.jobs])
print(vector_finalizacion)

#Ordenamos ese vector en orden decreciente
vector_finalizacion_descendiente=sorted_value_desc(vector_finalizacion)
print("El vector que almacena la finalización de las operaciones del
    trabajo ordenado descendiente es:")
print(vector_finalizacion_descendiente)

#Una vez tenemos el vector de finalización podemos hallar el de inicio rest
    ándole el tiempo de proceso
for i in range(instance.machines):
    vector_inicio.append(vector_finalizacion[i]-instance.pt[i][
        operacion_cambio%instance.jobs])
print(vector_inicio)

#Ordenamos ese vector en orden ascendiente
vector_inicio_ascendiente=sorted_value_asc(vector_inicio)
print("El vector que almacena el inicio de las operaciones del trabajo
    ordenado ascendiente es:")
print(vector_inicio_ascendiente)

Cj_INI_CIT=0
Cj_FIN_CIT=0
#Comprobamos que el hueco disponible del CIT sea mayor a la duración de la
    operación que se desea insertar
for j in range(instance.jobs-1):

    if(matriz_CIT[operacion_cambio//instance.jobs][j]>instance.pt[
        operacion_cambio//instance.jobs][operacion_cambio%instance.jobs]):
        #En este punto necesito almacenar Cj_INI_CIT y Cj_FIN_CIT
        Cj_INI_CIT=cij_ordenado_copia_ascendente[operacion_cambio//instance.
            jobs][j]
        Cj_FIN_CIT=cij_ordenado_copia_ascendente[operacion_cambio//instance.
            jobs][j+1]-processing_times_matriz[operacion_cambio//instance.
            jobs][j+1]

        #Esta variable almacena el tiempo de inicio de la operación del
            mismo trabajo que se desea insertar
        primera_operacion_superior=0
        #Esta variable almacena el tiempo de finalización de la operación
            del mismo trabajo que se desea insertar
        primera_operacion_inferior=0

        #Establezco contadores que me proporcionan el número de veces que se
             cumple cada una de las condiciones
        contador_nulo=0
```

```
            contador_nulo_fijo=0
            contador_fijo=0
            contador_inicio_fijo=0
            contador_final_fijo=0
            contador_medio_inicio_fijo=0
            contador_medio_final_fijo=0

            for i in range(instance.machines):
                #Calculo del primer inicio de operación superior de ese trabajo
                for k in range(instance.machines):
                    if vector_inicio_ascendiente[k]>=vector_finalizacion[i]:
                        primera_operacion_superior=vector_inicio_ascendiente[k]
                    break

                #Calculo del primer fin de operación inferior de ese trabajo
                for k in range(instance.machines):
                    if vector_finalizacion_descendiente[k]>=vector_finalizacion[
                        i]:
                        primera_operacion_inferior=
                            vector_finalizacion_descendiente[k]
                    break

                #Estudio del caso en el que una operación del mismo trabajo de
                    otra máquina se introduzca parcialmente en el inicio
                if(vector_finalizacion[i]-instance.pt[i][operacion_cambio%
                    instance.jobs]<Cj_INI_CIT and vector_finalizacion[i]>
                    Cj_INI_CIT and vector_finalizacion[i]<Cj_FIN_CIT):
                    primertermino_inicio=max(Cj_INI_CIT,vector_finalizacion[i])+
                        instance.pt[operacion_cambio//instance.jobs][
                        operacion_cambio%instance.jobs]
                    segundotermino_inicio=min(Cj_FIN_CIT,
                        primera_operacion_superior)
                    if(primertermino_inicio<=segundotermino_inicio):
                        contador_inicio_fijo=1

                #Estudio del caso en el que una operación del mismo trabajo de
                    otra máquina se introduzca parcialmente en el final
                if(vector_finalizacion[i]-instance.pt[i][operacion_cambio%
                    instance.jobs]<Cj_FIN_CIT and vector_finalizacion[i]-
                    instance.pt[i][operacion_cambio%instance.jobs]>Cj_INI_CIT
                    and vector_finalizacion[i]>Cj_FIN_CIT):
                    primertermino_final=min(Cj_FIN_CIT,vector_finalizacion[i]-
                        instance.pt[i][operacion_cambio%instance.jobs])-instance.
                        pt[operacion_cambio//instance.jobs][operacion_cambio%
                        instance.jobs]
                    segundotermino_final=max(Cj_INI_CIT,
                        primera_operacion_inferior)
                    if(primertermino_final>=segundotermino_final):
                        print(primera_operacion_inferior)
                        contador_final_fijo=1

                #Estudio del caso en el que una operación del mismo trabajo de
                    otra máquina se encuentre en medio del CIT
                if(vector_finalizacion[i]-instance.pt[i][operacion_cambio%
                    instance.jobs]>Cj_INI_CIT and vector_finalizacion[i]<
                    Cj_FIN_CIT):
```

```
                primertermino_medio_final=vector_finalizacion[i]+instance.pt[
                    operacion_cambio//instance.jobs][operacion_cambio%
                    instance.jobs]
                segundotermino_medio_final=min(Cj_FIN_CIT,
                    primera_operacion_superior)
                if(primertermino_medio_final<=segundotermino_medio_final):
                    contador_medio_final_fijo=1

                primertermino_medio_inicio=vector_finalizacion[i]-instance.pt
                    [i][operacion_cambio%instance.jobs]-instance.pt[
                    operacion_cambio//instance.jobs][operacion_cambio%
                    instance.jobs]
                segundotermino_medio_inicio=max(Cj_INI_CIT,
                    primera_operacion_inferior)
                if(primertermino_medio_inicio>=segundotermino_medio_inicio):
                    contador_medio_inicio_fijo=1

            #Estudio del caso en el que no se solape con ninguna operación
            if(vector_finalizacion[i]<Cj_INI_CIT or vector_inicio[i]>
                Cj_FIN_CIT):
                contador_fijo=contador_fijo+1

            if(vector_finalizacion[i]>Cj_FIN_CIT and vector_inicio[i]<
                Cj_INI_CIT):
                contador_nulo_fijo=1

        if(contador_inicio_fijo==1 and contador_nulo_fijo==0):
            Sij_inicio_S[operacion_cambio//instance.jobs][operacion_cambio%
                instance.jobs]=max(Cj_INI_CIT,vector_finalizacion[i])
            break

        if(contador_final_fijo==1 and contador_nulo_fijo==0):
            Sij_inicio_S[operacion_cambio//instance.jobs][operacion_cambio%
                instance.jobs]=segundotermino_final
            break

        if(contador_medio_inicio_fijo==1 and contador_nulo_fijo==0):
            Sij_inicio_S[operacion_cambio//instance.jobs][operacion_cambio%
                instance.jobs]=segundotermino_medio_inicio
            break

        if(contador_medio_final_fijo==1 and contador_nulo_fijo==0):
            Sij_inicio_S[operacion_cambio//instance.jobs][operacion_cambio%
                instance.jobs]=vector_finalizacion[i]
            break

        if(contador_fijo==instance.machines):
            Sij_inicio_S[operacion_cambio//instance.jobs][operacion_cambio%
                instance.jobs]=Cj_INI_CIT
        break

a=1
operacion_cambio=0
for i in range(len(U)):
    if U[i] in R:
        print("La operación que sale del vector U y metemos en el S es:",
            end='')
```

```
            operacion_cambio=U[i]
            #Insertamos la primera posición relativa en S
            S.append(operacion_cambio)
            #Eliminamos del vector U la primera posición relativa que es la
                posición i
            U=np.delete(U,i)
            a=0
        if(a==0):
                break

    #Estudio el tiempo actual y lo comparo con el inicial, si es superior al
        tiempo fijado, me salgo del bucle
    t_2=time.time()
    if(t_2-t_0>600):
        break

    #Actualizo los tiempos de finalización del vector U para la que salgan los
        tiempos de inicio actualizados en la siguiente iteración
    cij,cj=instance.ct(U)
    cij=np.array(cij)
    #Muestra los cij con la estructura general
    cij_ordenado_copia=cij_Ordenado(cij,cj)
    cij_ordenado_copia=np.array(cij_ordenado_copia)
    #Muestra los cij ordenados de forma ascendente
    cij_ordenado_copia_ascendente=cij_ordenado_copia.copy()
    cij_ordenado_copia_ascendente=np.array(cij_ordenado_copia_ascendente)
    cij_ordenado_copia_ascendente.sort(axis=1)
    #Se calculan los tiempos de proceso con la estructura correcta

    j_prima=0
    for i in range(instance.machines):
        for j in range(instance.jobs):
            j_prima=0
            for k in range(instance.jobs):
                if cij_ordenado_copia[i][j]==cij_ordenado_copia_ascendente[i][k]
                     and cij_ordenado_copia[i][j]>0:
                    j_prima=k
                    processing_times_matriz[i][j_prima]=instance.pt[i][j]

    #Hallamos los tiempos de inicio
    for i in range(instance.machines):
        for j in range(instance.jobs):
            Sij_inicio[i][j]=cij_ordenado_copia[i][j]-instance.pt[i][j]

    for i in range(instance.machines):
        for j in range(instance.jobs):
            #Cuando saco una operación de U, al calcular su finalización para
                que no altere los mínimos se iguala a una cota superior
            if(cij_ordenado_copia[i][j]==0):
                Sij_inicio[i][j]=M
            if(j not in cj):
                Sij_inicio[i][j]=M

#Muestra los cij de la librería
cij,cj=instance.ct(S)
cij=np.array(cij)
#Muestra los cij con la estructura general
```

```
cij_ordenado_copia=cij_Ordenado(cij,cj)
cij_ordenado_copia=np.array(cij_ordenado_copia)
#Muestra los cij ordenados de forma ascendente
cij_ordenado_copia_ascendente=cij_ordenado_copia.copy()
cij_ordenado_copia_ascendente=np.array(cij_ordenado_copia_ascendente)
cij_ordenado_copia_ascendente.sort(axis=1)

j_prima=0
for i in range(instance.machines):
    for j in range(instance.jobs):
        j_prima=0
        for k in range(instance.jobs):
            if cij_ordenado_copia[i][j]==cij_ordenado_copia_ascendente[i][k] and
                 cij_ordenado_copia[i][j]>0:
                j_prima=k
                processing_times_matriz[i][j_prima]=instance.pt[i][j]

processing_times_matriz=np.array(processing_times_matriz)
FO_final=FO_sostenibilidad(instance,S)
W=S.copy()

#Muestra los cij de la librería
cij,cj=instance.ct(W)
cij=np.array(cij)
#Muestra los cij con la estructura general
cij_ordenado_copia=cij_Ordenado(cij,cj)
cij_ordenado_copia=np.array(cij_ordenado_copia)
#Muestra los cij ordenados de forma ascendente
cij_ordenado_copia_ascendente=cij_ordenado_copia.copy()
cij_ordenado_copia_ascendente=np.array(cij_ordenado_copia_ascendente)
cij_ordenado_copia_ascendente.sort(axis=1)

make=np.amax(cij_ordenado_copia_ascendente)
WW_copia=[]
WW_copia_inicial=[]
improvement=1
while r>0:

    #Estudio el tiempo actual y lo comparo con el inicial, si es superior al
        tiempo fijado, me salgo del bucle
    t_2=time.time()
    if(t_2-t_0>600):
        break

    if(improvement==0):
        r=r-1

    improvement=0
    for z in range(instance.machines*instance.jobs):
        redundancia=0

        if(z>1) and ((W[r-1]%instance.jobs)!=(WW_copia[z-1]%instance.jobs) and
            (W[r-1]//instance.jobs)!=(WW_copia[z-1]//instance.jobs)):
            redundancia=1
        if(redundancia==0):
                WW_copia_inicial=W.copy()
                WW_copia=W.copy()
```

```
                print("El vector copiado: ")
                print(WW_copia)
                print("El vector copiado tras eliminar el valor: ")
                WW_copia_inicial=np.delete(WW_copia_inicial, np.where(WW_copia
                    == W[r-1]))
                print(WW_copia_inicial)
                print("El vector copiado tras insertarle el valor: ")
                WW_copia=np.insert(WW_copia_inicial,z,W[r-1])
                WW_copia = WW_copia.astype(int)
                print(WW_copia)

                #Muestra los cij de la librería
                cij,cj=instance.ct(WW_copia)
                cij=np.array(cij)
                #Muestra los cij con la estructura general
                cij_ordenado_copia=cij_Ordenado(cij,cj)
                cij_ordenado_copia=np.array(cij_ordenado_copia)
                #Muestra los cij ordenados de forma ascendente
                cij_ordenado_copia_ascendente=cij_ordenado_copia.copy()
                cij_ordenado_copia_ascendente=np.array(
                    cij_ordenado_copia_ascendente)
                cij_ordenado_copia_ascendente.sort(axis=1)

                makeWW=np.amax(cij_ordenado_copia_ascendente)
                if(makeWW<make):
                    improvement=1
                    make=makeWW
                    print("La FO ha mejorado")
                    Mejor_vector=WW_copia.copy()
    if(improvement==1):
        W=Mejor_vector.copy()
        print("El mejor vector resultante es:")
        print(W)
        r=instance.machines*instance.jobs

print('COMPROBACIONES')
print("La secuencia final obtenida es:")
print(W)

#Muestra los cij de la librería
cij,cj=instance.ct(W)
print('Se muestran los cij y los cj')
cij=np.array(cij)
print(cij)
print(cj)
#Muestra los cij con la estructura general
cij_ordenado_copia=cij_Ordenado(cij,cj)
cij_ordenado_copia=np.array(cij_ordenado_copia)
print('Se muestran los cij y los cj ORDENADOS')
print(cij_ordenado_copia)
#Muestra los cij ordenados de forma ascendente
cij_ordenado_copia_ascendente=cij_ordenado_copia.copy()
cij_ordenado_copia_ascendente=np.array(cij_ordenado_copia_ascendente)
cij_ordenado_copia_ascendente.sort(axis=1)
print('Se muestran los cij y los cj ORDENADOS ASCENDENTE')
print(cij_ordenado_copia_ascendente)
```

```
FO_final=FO_sostenibilidad(instance,W)
print('Su FO es:')
print(FO_final)

print('Finaliza en:')
print(np.amax(cij_ordenado_copia_ascendente))

t_1=time.time()
print("El tiempo de computo ha sido:")
print(t_1-t_0)

gantt=instance.create_schedule(W)
gantt.print()
```

2.4 Código Heurística 4 para Makespan

En el siguiente código se presenta la heurística 4 para el Makespan.

```
#Cargamos la instancia
from scheptk.scheptk import OpenShop
#Ordenamos los trabajos de una secuencia en orden decreciente
from scheptk.util import sorted_value_desc
#Ordenamos los trabajos de una secuencia en orden ascendiente
from scheptk.util import sorted_value_asc
#Ordenamos los trabajos de una secuencia en orden decreciente
from scheptk.util import sorted_index_desc

import numpy as np
import time
t_0=time.time()

instance=OpenShop('Instancia_modificada_1.txt')

#Función Objetivo Sostenibilidad
def FO_sostenibilidad(instance,sequence):
    #Esta primera parte de la función se encarga de poner la matriz de los
        tiempos de proceso con la misma estructura que la de los tiempos de
        finalización (ordenados de forma ascendente por filas)
    for i in range(instance.machines):
        for j in range(instance.jobs):
            j_prima=0
            for k in range(instance.jobs):
                if cij_ordenado_copia[i][j]==cij_ordenado_copia_ascendente[i][k]
                     and cij_ordenado_copia[i][j]>0:
                    j_prima=k
                    processing_times_matriz[i][j_prima]=instance.pt[i][j]
    idle_time=0
    for i in range(instance.machines):
        for j in range(0,instance.jobs-1):
            if(cij_ordenado_copia_ascendente[i][j]<cij_ordenado_copia_ascendente
                [i][j+1]-processing_times_matriz[i][j+1] and
                cij_ordenado_copia_ascendente[i][j]>0):
                idle_time=idle_time+(cij_ordenado_copia_ascendente[i][j+1]-
                    processing_times_matriz[i][j+1])-
                    cij_ordenado_copia_ascendente[i][j]
    return idle_time
```

```
#Función Cálculo Matriz Core Idle Time
def Calculo_matriz_CIT(instance,sequence):
    #Esta primera parte de la función se encarga de poner la matriz de los
        tiempos de proceso con la misma estructura que la de los tiempos de
        finalización (ordenados de forma ascendente por filas)
    j_prima=0
    for i in range(instance.machines):
        for j in range(instance.jobs):
            j_prima=0
            for k in range(instance.jobs):
                if cij_ordenado_copia[i][j]==cij_ordenado_copia_ascendente[i][k]
                     and cij_ordenado_copia[i][j]>0:
                    j_prima=k
                    processing_times_matriz[i][j_prima]=instance.pt[i][j]
    matriz_CIT=[[0 for j in range(instance.jobs-1)] for i in range(instance.
        machines)]
    for i in range(instance.machines):
        for j in range(0,instance.jobs-1):
            if(cij_ordenado_copia_ascendente[i][j]<cij_ordenado_copia_ascendente
                [i][j+1]-processing_times_matriz[i][j+1] and
                cij_ordenado_copia_ascendente[i][j]>0):
                matriz_CIT[i][j]=cij_ordenado_copia_ascendente[i][j+1]-
                    processing_times_matriz[i][j+1]-
                    cij_ordenado_copia_ascendente[i][j]
    return matriz_CIT

#Función que calcula la estructura general de los cij
def cij_Ordenado(cij,cj):
    #Creamos una matriz en la que vamos a almacenar los cij con la estructura
        general en la que en la posición [0][0] se encuentra el cij de la
        operación 0
    cij_ordenado_copia=[[0 for j in range(instance.jobs)] for i in range(
        instance.machines)]
    for i in range(instance.machines):
        for j in cj:
            cij_ordenado_copia[i][j]=cij[i][cj.index(j)]
    return cij_ordenado_copia

#Creamos una matriz que esta formada por las etiquetas
Etiquetas=[[0 for j in range(instance.jobs)] for i in range(instance.machines)]

a=-1
for i in range(instance.machines):
    for j in range(instance.jobs):
        Etiquetas[i][j]=a+1
        a=a+1


Sij_inicio_S=[[0 for i in range(instance.machines)] for j in range(instance.
    jobs)]
Sij_final_S=[[0 for i in range(instance.machines)] for j in range(instance.jobs
    )]
#Este vector almacenará los tiempos de proceso lo que permitirá ordenar la
    secuencia
processing_times=[]
#Creamos una matriz en la que vamos a almacenar los tiempos de proceso con la
    misma estructura que la cij ordenada ascendiente por filas
```

```
processing_times_matriz=[[0 for i in range(instance.jobs)] for j in range(
    instance.machines)]
#Establecemos M como cota superior siendo igual a la suma de todos los tiempos
    de proceso
M=0
for i in range(instance.machines):
    for j in range(instance.jobs):
        processing_times.append(instance.pt[i][j])
        M=M+instance.pt[i][j]

#Este vector es el E de la MH1
E=sorted_index_desc(processing_times)
#Creamos el vector W que inicialmente es un conjunto vacio
W=[]
#La variable r almacena el número de operaciones de la instancia estudiada
r=instance.machines*instance.jobs

for k in range(r):
    #Este vector almacena las funciones objetivos tras insertar una operación
        en distintas posiciones
    #Se inicializa a M para que, en caso de que sea redundante, no salga
        elegido
    Acum_FO=np.ones(shape=len(W)+1)*M
    Acum_FO=Acum_FO.astype(int)

    for z in range(k):
        redundancia=0
        if(z>1) and ((E[k]%instance.jobs)!=(W[z-1]%instance.jobs) and (E[k]//
            instance.jobs)!=(W[z-1]//instance.jobs)):
            redundancia=1
        if(redundancia==0):
            #Copio en el vector W_copia el vector W
            W_copia=W.copy()
            #Le inserto en la posición z la operación E[k]
            W_copia=np.insert(W_copia,z,E[k])
            W_copia=W_copia.astype(int)

            cij,cj= instance.ct(W_copia)
            #Llamamos a la función que calcula el cij con la estructura correcta
            cij_ordenado_copia=cij_Ordenado(cij,cj)
            #Ordenamos la matriz cij_ordenado_copia con la estructura ascendente
            cij_ordenado_copia=np.array(cij_ordenado_copia)
            cij_ordenado_copia_ascendente=cij_ordenado_copia.copy()
            cij_ordenado_copia_ascendente=np.array(cij_ordenado_copia_ascendente
                )
            cij_ordenado_copia_ascendente.sort(axis=1)

            #En el vector Acum_FO[z] voy acumulando las distintas funciones
                objetivos que voy obteniendo
            Acum_FO[z]=np.amax(cij_ordenado_copia_ascendente)

            #Estudio el tiempo actual y lo comparo con el inicial, si es
                superior al tiempo fijado, me salgo del bucle
            t_2=time.time()
            if(t_2-t_0>600):
                break
```

```
print('FO obtenidas en primera iteración:', end=' ')
print(Acum_FO)
#Establecemos el valor que se encuentra en la primera posición como mínimo
FO_en_z=Acum_FO[0]
posicion_menor_FO=0

#Almacenamos la mejor FO en FO_en_z y su posición posicion_menor_FO
for p in range(k):
    if Acum_FO[p]<FO_en_z:
        posicion_menor_FO=p
        FO_en_z=Acum_FO[p]

#Estudio el tiempo actual y lo comparo con el inicial, si es superior al
    tiempo fijado, me salgo del bucle
t_2=time.time()
if(t_2-t_0>600):
    break

print('La posición en la que menor valor de la FO se ha obtenido 1 iteració
    n es:', end=' ')
print(posicion_menor_FO)
print('El valor de la FO que se ha obtenido es:')
print(FO_en_z)
print('Insertando la operación actual en esa posición obtenemos:', end=' ')
W=np.insert(W,posicion_menor_FO,E[k])
print(W)
W=W.astype(int)

#Comparo los dos valores para establecer el inicio y el fin del bucle for
k_numero=9

if 1>posicion_menor_FO-k_numero:
    inicio=1
else:
    inicio=posicion_menor_FO-k_numero

if k<posicion_menor_FO+k_numero:
    fin=k
else:
    fin=posicion_menor_FO+k_numero

for r3 in range(inicio,fin):

    #Estudio el tiempo actual y lo comparo con el inicial, si es superior
        al tiempo fijado, me salgo del bucle
    t_2=time.time()
    if(t_2-t_0>600):
        break

    valor=W[r3]
    W=np.delete(W,np.where(W==valor))
    print(W)
    Acum_FO_nueva=np.ones(shape=len(W)+1)*M
    Acum_FO_nueva=Acum_FO_nueva.astype(int)

    for r4 in range(k):
        redundancia=0
```

```
            if(r4>1) and ((W[r3]%instance.jobs)!=(W[r4-1]%instance.jobs) and (W[
                r3]//instance.jobs)!=(W[r4-1]//instance.jobs)):
                redundancia=1
            if(redundancia==0):
                W_copia_nueva=W.copy()

                #Le inserto en la posición r4 la operación valor
                W_copia_nueva=np.insert(W_copia_nueva,r4,valor)
                W_copia_nueva=W_copia_nueva.astype(int)

                cij,cj= instance.ct(W_copia_nueva)

                #Llamamos a la función que calcula el cij con la estructura
                    correcta
                cij_ordenado_copia=cij_Ordenado(cij,cj)

                #Ordenamos la matriz cij_ordenado_copia con la estructura
                    ascendente
                cij_ordenado_copia=np.array(cij_ordenado_copia)
                cij_ordenado_copia_ascendente=cij_ordenado_copia.copy()
                cij_ordenado_copia_ascendente=np.array(
                    cij_ordenado_copia_ascendente)
                cij_ordenado_copia_ascendente.sort(axis=1)

                #En el vector Acum_FO[z] voy acumulando las distintas funciones
                    objetivos que voy obteniendo
                Acum_FO_nueva[r4]=np.amax(cij_ordenado_copia_ascendente)

                #Estudio el tiempo actual y lo comparo con el inicial, si es
                    superior al tiempo fijado, me salgo del bucle
                t_2=time.time()
                if(t_2-t_0>600):
                    break

        print('El vector con las diferentes FO obtenidas')
        print(Acum_FO_nueva)
        #Establecemos el valor que se encuentra en la primera posición como mí
            nimo
        FO_en_z_nueva=Acum_FO_nueva[0]
        posicion_menor_FO_nueva=0
        for p in range(k):
            if Acum_FO_nueva[p]<FO_en_z_nueva:
                posicion_menor_FO_nueva=p
                FO_en_z_nueva=Acum_FO_nueva[p]

        W=np.insert(W,posicion_menor_FO_nueva,valor)
        print(W)
        W=W.astype(int)


#Muestra los cij de la librería
cij,cj=instance.ct(W)
cij=np.array(cij)

#Muestra los cij con la estructura general
cij_ordenado_copia=cij_Ordenado(cij,cj)
cij_ordenado_copia=np.array(cij_ordenado_copia)
```

```
#Muestra los cij ordenados de forma ascendente
cij_ordenado_copia_ascendente=cij_ordenado_copia.copy()
cij_ordenado_copia_ascendente=np.array(cij_ordenado_copia_ascendente)
cij_ordenado_copia_ascendente.sort(axis=1)

FO_final=FO_sostenibilidad(instance,W)

#Creamos el vector S que inicialmente es un conjunto vacío
S=[]
#El vector U esta formado por todas las operaciones
U=W.copy()
y=0

#La matriz Sij_inicio almacena los tiempos de inicio de las operaciones
Sij_inicio=[[0 for i in range(instance.machines)] for j in range(instance.jobs)
    ]

#Inicialmente se calcula como la resta de cij_ordenado_copia menos los tiempos
    de proceso
for i in range(instance.machines):
    for j in range(instance.jobs):
        Sij_inicio[i][j]=cij_ordenado_copia[i][j]-instance.pt[i][j]

W_copia=W.copy()

while len(U)!=0:
    #Estudio el tiempo actual y lo comparo con el inicial, si es superior al
        tiempo fijado, me salgo del bucle
    t_2=time.time()
    if(t_2-t_0>600):
        break

    y=np.amin(Sij_inicio)

    #A continuación asignamos las operaciones cuyo valor sea igual a 'y' en R
    R=[]
    for i in range(instance.machines):
        for j in range(instance.jobs):
            if(Sij_inicio[i][j]==y):
                R.append(Etiquetas[i][j])

    print("El vector R es: ", end='')
    print(R)
    print("El vector U es: ", end='')
    print(U)
    print("El vector S es: ", end='')
    print(S)
    #Este bucle inserta la primera posición relativa en S y la elimina del
        vector U
    a=1
    operacion_cambio=0
    for i in range(len(U)):
        if U[i] in R:
            print("La operación que sale del vector U y metemos en el S es:",
                end='')
            operacion_cambio=U[i]
```

```
            #Insertamos la primera posición relativa en S
            S_copia=S.copy()
            S.append(operacion_cambio)
            #Eliminamos del vector U la primera posición relativa que es la
                posición i
            U=np.delete(U,i)
            a=0
        if(a==0):
                break

    #Muestra los cij de la librería
    cij,cj=instance.ct(S)
    cij=np.array(cij)
    #Muestra los cij con la estructura general
    cij_ordenado_copia=cij_Ordenado(cij,cj)
    cij_ordenado_copia=np.array(cij_ordenado_copia)
    #Muestra los cij ordenados de forma ascendente
    cij_ordenado_copia_ascendente=cij_ordenado_copia.copy()
    cij_ordenado_copia_ascendente=np.array(cij_ordenado_copia_ascendente)
    cij_ordenado_copia_ascendente.sort(axis=1)

    #Actualizamos la estructura de la matriz de los tiempos de proceso
    for i in range(instance.machines):
        for j in range(instance.jobs):
            j_prima=0
            for k in range(instance.jobs):
                if cij_ordenado_copia[i][j]==cij_ordenado_copia_ascendente[i][k]
                     and cij_ordenado_copia[i][j]>0:
                    j_prima=k
                    processing_times_matriz[i][j_prima]=instance.pt[i][j]

    matriz_CIT=Calculo_matriz_CIT(instance,S)
    matriz_CIT=np.array(matriz_CIT)

    #A continuación calculamos la finalización de las operaciones de ese
        trabajo en todas las maquinas
    vector_finalizacion=[]
    vector_inicio=[]
    vector_inicio_ascendiente=[]
    print("El vector que almacena la finalización de las operaciones del
        trabajo es:")
    for i in range(instance.machines):
        vector_finalizacion.append(cij_ordenado_copia[i][operacion_cambio%
            instance.jobs])
    print(vector_finalizacion)

    #Ordenamos ese vector en orden decreciente
    vector_finalizacion_descendiente=sorted_value_desc(vector_finalizacion)
    print("El vector que almacena la finalización de las operaciones del
        trabajo ordenado descendiente es:")
    print(vector_finalizacion_descendiente)

    #Una vez tenemos el vector de finalización podemos hallar el de inicio
        restandole el tiempo de proceso
    for i in range(instance.machines):
        vector_inicio.append(vector_finalizacion[i]-instance.pt[i][
            operacion_cambio%instance.jobs])
```

```
print(vector_inicio)

#Ordenamos ese vector en orden ascendiente
vector_inicio_ascendiente=sorted_value_asc(vector_inicio)
print("El vector que almacena el inicio de las operaciones del trabajo
    ordenado ascendiente es:")
print(vector_inicio_ascendiente)

Cj_INI_CIT=0
Cj_FIN_CIT=0
#Comprobamos que el hueco disponible del CIT sea mayor a la duracion de la
    operación que se desea insertar
for j in range(instance.jobs-1):
    if(matriz_CIT[operacion_cambio//instance.jobs][j]>instance.pt[
        operacion_cambio//instance.jobs][operacion_cambio%instance.jobs]):
        #En este punto necesito almacenar Cj_INI_CIT y Cj_FIN_CIT
        Cj_INI_CIT=cij_ordenado_copia_ascendente[operacion_cambio//instance.
            jobs][j]
        Cj_FIN_CIT=cij_ordenado_copia_ascendente[operacion_cambio//instance.
            jobs][j+1]-processing_times_matriz[operacion_cambio//instance.
            jobs][j+1]

        #Esta variable almacena el tiempo de inicio de la operación del
            mismo trabajo que se desea insertar
        primera_operacion_superior=0
        #Esta variable almacena el tiempo de finalización de la operación
            del mismo trabajo que se desea insertar
        primera_operacion_inferior=0

        #Establezco contadores que me proporcionan el numero de veces que se
             cumple cada una de las condiciones
        contador_nulo=0
        contador_nulo_fijo=0
        contador_fijo=0
        contador_inicio_fijo=0
        contador_final_fijo=0
        contador_medio_inicio_fijo=0
        contador_medio_final_fijo=0

        for i in range(instance.machines):

            #Calculo del primer inicio de operación superior de ese trabajo
            for k in range(instance.machines):
                if vector_inicio_ascendiente[k]>=vector_finalizacion[i]:
                    primera_operacion_superior=vector_inicio_ascendiente[k]
                break

            #Calculo del primer fin de operación inferior de ese trabajo
            for k in range(instance.machines):
                if vector_finalizacion_descendiente[k]>=vector_finalizacion[
                    i]:
                    primera_operacion_inferior=
                        vector_finalizacion_descendiente[k]
                break

            #Estudio del caso en el que una operación del mismo trabajo de
                otra máquina se introduzca parcialmente en el inicio
```

```
                if(vector_finalizacion[i]-instance.pt[i][operacion_cambio%
                    instance.jobs]<Cj_INI_CIT and vector_finalizacion[i]>
                    Cj_INI_CIT and vector_finalizacion[i]<Cj_FIN_CIT):
                    primertermino_inicio=max(Cj_INI_CIT,vector_finalizacion[i])+
                        instance.pt[operacion_cambio//instance.jobs][
                        operacion_cambio%instance.jobs]
                    segundotermino_inicio=min(Cj_FIN_CIT,
                        primera_operacion_superior)
                    if(primertermino_inicio<=segundotermino_inicio):
                        contador_inicio_fijo=1

                #Estudio del caso en el que una operación del mismo trabajo de
                    otra máquina se introduzca parcialmente en el final
                if(vector_finalizacion[i]-instance.pt[i][operacion_cambio%
                    instance.jobs]<Cj_FIN_CIT and vector_finalizacion[i]-
                    instance.pt[i][operacion_cambio%instance.jobs]>Cj_INI_CIT
                    and vector_finalizacion[i]>Cj_FIN_CIT):
                    primertermino_final=min(Cj_FIN_CIT,vector_finalizacion[i]-
                        instance.pt[i][operacion_cambio%instance.jobs])-instance.
                        pt[operacion_cambio//instance.jobs][operacion_cambio%
                        instance.jobs]
                    segundotermino_final=max(Cj_INI_CIT,
                        primera_operacion_inferior)
                    if(primertermino_final>=segundotermino_final):
                        print(primera_operacion_inferior)
                        contador_final_fijo=1

                #Estudio del caso en el que una operación del mismo trabajo de
                    otra máquina se encuentre en medio del CIT
                if(vector_finalizacion[i]-instance.pt[i][operacion_cambio%
                    instance.jobs]>Cj_INI_CIT and vector_finalizacion[i]<
                    Cj_FIN_CIT):
                    primertermino_medio_final=vector_finalizacion[i]+instance.pt[
                        operacion_cambio//instance.jobs][operacion_cambio%
                        instance.jobs]
                    segundotermino_medio_final=min(Cj_FIN_CIT,
                        primera_operacion_superior)
                    if(primertermino_medio_final<=segundotermino_medio_final):
                        contador_medio_final_fijo=1

                    primertermino_medio_inicio=vector_finalizacion[i]-instance.pt
                        [i][operacion_cambio%instance.jobs]-instance.pt[
                        operacion_cambio//instance.jobs][operacion_cambio%
                        instance.jobs]
                    segundotermino_medio_inicio=max(Cj_INI_CIT,
                        primera_operacion_inferior)
                    if(primertermino_medio_inicio>=segundotermino_medio_inicio):
                        contador_medio_inicio_fijo=1

                #Estudio del caso en el que no se solape con ninguna operación
                if(vector_finalizacion[i]<Cj_INI_CIT or vector_inicio[i]>
                    Cj_FIN_CIT):
                    contador_fijo=contador_fijo+1

                if(vector_finalizacion[i]>Cj_FIN_CIT and vector_inicio[i]<
                    Cj_INI_CIT):
                    contador_nulo_fijo=1
```

```
        if(contador_inicio_fijo==1 and contador_nulo_fijo==0):
            Sij_inicio_S[operacion_cambio//instance.jobs][operacion_cambio%
                instance.jobs]=max(Cj_INI_CIT,vector_finalizacion[i])
            break

        if(contador_final_fijo==1 and contador_nulo_fijo==0):
            Sij_inicio_S[operacion_cambio//instance.jobs][operacion_cambio%
                instance.jobs]=segundotermino_final
            break

        if(contador_medio_inicio_fijo==1 and contador_nulo_fijo==0):
            Sij_inicio_S[operacion_cambio//instance.jobs][operacion_cambio%
                instance.jobs]=segundotermino_medio_inicio
            break

        if(contador_medio_final_fijo==1 and contador_nulo_fijo==0):
            Sij_inicio_S[operacion_cambio//instance.jobs][operacion_cambio%
                instance.jobs]=vector_finalizacion[i]
            break

        if(contador_fijo==instance.machines):
            Sij_inicio_S[operacion_cambio//instance.jobs][operacion_cambio%
                instance.jobs]=Cj_INI_CIT
        break


a=1
operacion_cambio=0
for i in range(len(U)):
    if U[i] in R:
        print("La operación que sale del vector U y metemos en el S es:",
            end='')
        operacion_cambio=U[i]
        #Insertamos la primera posición relativa en S
        S.append(operacion_cambio)
        #Eliminamos del vector U la primera posición relativa que es la
            posición i
        U=np.delete(U,i)
        a=0
    if(a==0):
            break

#Estudio el tiempo actual y lo comparo con el inicial, si es superior al
    tiempo fijado, me salgo del bucle
t_2=time.time()
if(t_2-t_0>600):
    break

#Actualizo los tiempos de finalización del vector U para la que salgan los
    tiempos de inicio actualizados en la siguiente iteración
cij,cj=instance.ct(U)
cij=np.array(cij)
#Muestra los cij con la estructura general
cij_ordenado_copia=cij_Ordenado(cij,cj)
cij_ordenado_copia=np.array(cij_ordenado_copia)
#Muestra los cij ordenados de forma ascendente
```

```
    cij_ordenado_copia_ascendente=cij_ordenado_copia.copy()
    cij_ordenado_copia_ascendente=np.array(cij_ordenado_copia_ascendente)
    cij_ordenado_copia_ascendente.sort(axis=1)
    #Se calculan los tiempos de proceso con la estructura correcta

    j_prima=0
    for i in range(instance.machines):
        for j in range(instance.jobs):
            j_prima=0
            for k in range(instance.jobs):
                if cij_ordenado_copia[i][j]==cij_ordenado_copia_ascendente[i][k]
                     and cij_ordenado_copia[i][j]>0:
                    j_prima=k
                    processing_times_matriz[i][j_prima]=instance.pt[i][j]

    #Hallamos los tiempos de inicio
    for i in range(instance.machines):
        for j in range(instance.jobs):
            Sij_inicio[i][j]=cij_ordenado_copia[i][j]-instance.pt[i][j]

    for i in range(instance.machines):
        for j in range(instance.jobs):
            #Cuando saco una operacion de U, al calcular su finalización para
                que no altere los mínimos se iguala a una cota superior
            if(cij_ordenado_copia[i][j]==0):
                Sij_inicio[i][j]=M
            if(j not in cj):
                Sij_inicio[i][j]=M


print('COMPROBACIONES')
print("La secuencia final obtenida es:")
print(S)

#Muestra los cij de la librería
cij,cj=instance.ct(S)
print('Se muestran los cij y los cj')
cij=np.array(cij)
print(cij)
print(cj)
#Muestra los cij con la estructura general
cij_ordenado_copia=cij_Ordenado(cij,cj)
cij_ordenado_copia=np.array(cij_ordenado_copia)
print('Se muestran los cij y los cj ORDENADOS')
print(cij_ordenado_copia)
#Muestra los cij ordenados de forma ascendente
cij_ordenado_copia_ascendente=cij_ordenado_copia.copy()
cij_ordenado_copia_ascendente=np.array(cij_ordenado_copia_ascendente)
cij_ordenado_copia_ascendente.sort(axis=1)
print('Se muestran los cij y los cj ORDENADOS ASCENDENTE')
print(cij_ordenado_copia_ascendente)

FO_final=FO_sostenibilidad(instance,S)
print('Su FO es:')
print(FO_final)

print('Finaliza en:')
```

```
print(np.amax(cij_ordenado_copia_ascendente))

t_1=time.time()
print("El tiempo de computo ha sido:")
print(t_1-t_0)

gantt=instance.create_schedule(S)
gantt.print()
```

2.5 Código Función objetivo: Core Idle Time

Las función objetivo que se ha modificado en las heurísticas anteriores para el cálculo del Core Idle Time se muestra en el código que se presenta a continuación. Es la misma que se puede encontrar en la codificación expuesta anteriormente, la única diferencia es que a la hora de evaluar durante la ejecución cualquier secuencia, en vez de calcular el instante de finalización (tal y como se procede en la codificación mostrada anteriormente para calcular el Makespan) se llamaría a esta Función Objetivo de Sostenibilidad.

```
#Función Objetivo Sostenibilidad
def FO_sostenibilidad(instance,sequence):
    #Esta primera parte de la función se encarga de poner la matriz de los
        tiempos de proceso con la misma estructura que la de los tiempos de
        finalización (ordenados de forma ascendente por filas)
    for i in range(instance.machines):
        for j in range(instance.jobs):
            j_prima=0
            for k in range(instance.jobs):
                if cij_ordenado_copia[i][j]==cij_ordenado_copia_ascendente[i][k]
                     and cij_ordenado_copia[i][j]>0:
                    j_prima=k
                    processing_times_matriz[i][j_prima]=instance.pt[i][j]
    idle_time=0
    for i in range(instance.machines):
        for j in range(0,instance.jobs-1):
            if(cij_ordenado_copia_ascendente[i][j]<cij_ordenado_copia_ascendente
                [i][j+1]-processing_times_matriz[i][j+1] and
                cij_ordenado_copia_ascendente[i][j]>0):
                idle_time=idle_time+(cij_ordenado_copia_ascendente[i][j+1]-
                    processing_times_matriz[i][j+1])-
                    cij_ordenado_copia_ascendente[i][j]
    return idle_time
```

2.6 Código Función objetivo: Ponderación Makespan y Core Idle Time

En el código que se muestra a continuación se presenta la función objetivo modificada para el cálculo de la ponderación de ambos objetivos.

```
    #Función objetivo ponderada
    def FO_sostenibilidad(instance,sequence):
        #Esta primera parte de la función se encarga de poner la matriz de los
            tiempos de proceso con la misma estructura que la de los tiempos de
            finalización (ordenados de forma ascendente por filas)
        j_prima=0
        for i in range(instance.machines):
            for j in range(instance.jobs):
                j_prima=0
                for k in range(instance.jobs):
```

```
                if cij_ordenado_copia[i][j]==cij_ordenado_copia_ascendente[i
                    ][k] and cij_ordenado_copia[i][j]>0:
                    j_prima=k
                    processing_times_matriz[i][j_prima]=instance.pt[i][j]
        idle_time=0
        makespan=0
        print('La secuencia evaluada es:', end=' ')
        print(sequence)
        for i in range(instance.machines):
            for j in range(0,instance.jobs-1):
                if(cij_ordenado_copia_ascendente[i][j]<
                    cij_ordenado_copia_ascendente[i][j+1]-
                    processing_times_matriz[i][j+1] and
                    cij_ordenado_copia_ascendente[i][j]>0):
                    idle_time=idle_time+(cij_ordenado_copia_ascendente[i][j+1]-
                        processing_times_matriz[i][j+1])-
                        cij_ordenado_copia_ascendente[i][j]

        makespan=np.amax(cij_ordenado_copia)
        idle_time=idle_time*lambda+makespan*(1-lambda)
        return idle_time
```

3 Código Traductor de instancias cuadradas Taillard

En el siguiente apartado se presenta el código que se ha utilizado para traducir el formato inicial de las instancias al necesario para ejecutar dichas instancias en phyton.

```
Entrada = open('Instancia_4x4_1.txt', 'r')
#Me salto la línea de number of jobs,number of machines
Entrada.readline()

numero_trab_maq = [int(x) for x in next(Entrada).split()]

Jobs = numero_trab_maq[0]
Machines = numero_trab_maq[1]

#Me salto la línea de processing times
Entrada.readline()

#A continuación almaceno en tiempos_proceso los tiempos, primero del primer
    trabajo, luego del segundo...
tiempos_proceso = []
for i in range(Jobs):
    tiempos_proceso.append([int(x) for x in next(Entrada).split()])

print('Los tiempos de proceso almacenados son:')
print(tiempos_proceso)
#Me salto la línea de machines
Entrada.readline()

maquinas = []
for i in range(Jobs):
    maquinas.append([(int(x) - 1) for x in next(Entrada).split()])

print('Las máquinas almacenadas son:')
```

```
print(maquinas)

Instancia_4x4_1_modificada = '[JOBS=' + str(Jobs) + ']\n'
Instancia_4x4_1_modificada += '[MACHINES=' + str(Machines) + ']\n'
Instancia_4x4_1_modificada += '[PT='
for i in range(Jobs):
    for j in range(Machines):
        j_prima = 0
        for k, trabajo in enumerate(maquinas[i]):
            if j == trabajo:
                j_prima = k
        Instancia_4x4_1_modificada += str(tiempos_proceso[i][j_prima]) + ','
    Instancia_4x4_1_modificada = Instancia_4x4_1_modificada[:-1] + ';'
Instancia_4x4_1_modificada = Instancia_4x4_1_modificada[:-1] + ']\n'

print(Instancia_4x4_1_modificada)

Entrada.close()

Salida = open('Instancia_4x4_1_modificada.txt', 'w')
Salida.write(Instancia_4x4_1_modificada)
Salida.close()
```

4 Código Traductor de instancias mxn Taillard

En el siguiente apartado se presenta el código que se ha utilizado para traducir el formato inicial de las instancias al necesario para ejecutar dichas instancias en phyton.

```
import numpy as np


Entrada = open('TA051', 'r')

numero_maq = [int(x) for x in next(Entrada).split()]
Machines = numero_maq[0]

numero_trab = [int(x) for x in next(Entrada).split()]
Jobs = numero_trab[0]

#A continuación almaceno en tiempos_proceso los tiempos, primero del primer
    trabajo, luego del segundo...
tiempos_proceso = []
for i in range(Jobs):
    tiempos_proceso.append([int(x) for x in next(Entrada).split()])

print('Los tiempos de proceso almacenados son:')
print(tiempos_proceso)

#Aquí hallamos la transpuesta de la matriz
np.array(tiempos_proceso)
transpuesta=np.transpose(tiempos_proceso)
print(transpuesta)


TA051_modificada = '[JOBS=' + str(Jobs) + ']\n'
```

```
TA051_modificada += '[MACHINES=' + str(Machines) + ']\n'
TA051_modificada += '[PT='
for j in range(Machines):
    for i in range(Jobs):
        TA051_modificada += str(tiempos_proceso[i][j]) + ','
    TA051_modificada = TA051_modificada[:-1] + ';'
TA051_modificada = TA051_modificada[:-1] + ']\n'

print(TA051_modificada)

Entrada.close()

Salida = open('TA051_modificada.txt', 'w')
Salida.write(TA051_modificada)
Salida.close()
```

5 Resultados por Instancias en cada Caso

Link Excel Resultados Completos