Proyecto Fin de Carrera
Ingeniería de Organización Industrial

Resolución de un Modelo Híbrido de Programación de la Producción mediante Algoritmos Heurísticos

Autor: María Caballero Saborido
Tutor: Carla Talens Fayos

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

Sevilla, 2023

Trabajo Fin de Grado
Grado en Ingeniería de Organización Industrial

Resolución de un Modelo Híbrido de Programación de la Producción mediante Algoritmos Heurísticos

Autor:
María Caballero Saborido

Tutor:
Carla Talens Fayos

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

Sevilla, 2023

Proyecto Fin de Grado: Resolución de un Modelo Híbrido de Programación de la Producción mediante Algoritmos Heurísticos

Autor: María Caballero Saborido

Tutor: Carla Talens Fayos

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2023

El Secretario del Tribunal

A mi familia

A mis maestros

A mis amigas

Agradecimientos

Me gustaría agradecer a todas las personas que me han apoyado a lo largo de esta aventura y me han servido de apoyo en los momentos duros.

A mi tutora Carla Talens Fayos, por aceptar tutorizarme este trabajo fin de grado, así como por su tiempo y dedicación durante todo el proceso.

A mis padres, por haberme ofrecido todo lo que estaba en sus manos para conseguir mi objetivo.

A mis compañeras, ya amigas Lucía y Patri, sin vosotras nada hubiera sido lo mismo.

A mis abuelos, Concha y Manuel, por ser el ejemplo en el que convertirme algún día.

María Caballero Saborido

Sevilla, 2023

Resumen

En el presente proyecto se estudia un modelo híbrido de programación de la producción, formado por dos etapas, con el objetivo de minimizar el tiempo máximo de finalización de los trabajos. Para su resolución, se plantean distintos métodos aproximados. En primer lugar, se adaptan distintas reglas de despacho de la literatura y, en segundo lugar, se proponen distintas versiones de una heurística constructiva. Para comprobar la eficiencia de los métodos propuestos, se resuelven un total de 900 instancias generadas con datos diferentes con el objetivo de analizar cómo funcionan sobre problemas de diferentes tamaños. El modelo, los métodos y la generación de instancias se programan en lenguaje *Python*.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE ILUSTRACIONES

1 INTRODUCCIÓN

Sólo el conocimiento que llega desde dentro es el verdadero conocimiento.

- Sócrates -

La Organización de la Producción consiste en la toma de decisiones que afectan al sistema productivo y a la logística de aprovisionamiento y de distribución a distintos niveles: estratégico (largo plazo), táctico (medio plazo) y operativo (corto plazo).

Las decisiones estratégicas que afectan al sistema productivo son la localización de la unidad productiva, la capacidad de fabricación, el proceso productivo y las infraestructuras de servicios, entre otros. A nivel táctico, se hacen previsiones de la demanda, se determina el nivel de stocks deseable y se planifica la producción (cuánto hay que producir y en qué momento). Por último, se encuentran las decisiones operativas. Estas decisiones son las que se toman diariamente en la empresa y la más importante es la programación de operaciones. La programación de operaciones consiste en llevar a cabo la planificación de la producción, que ha determinado cuántos productos y cuándo de manera desagregada, en las distintas operaciones de nuestro sistema productivo.

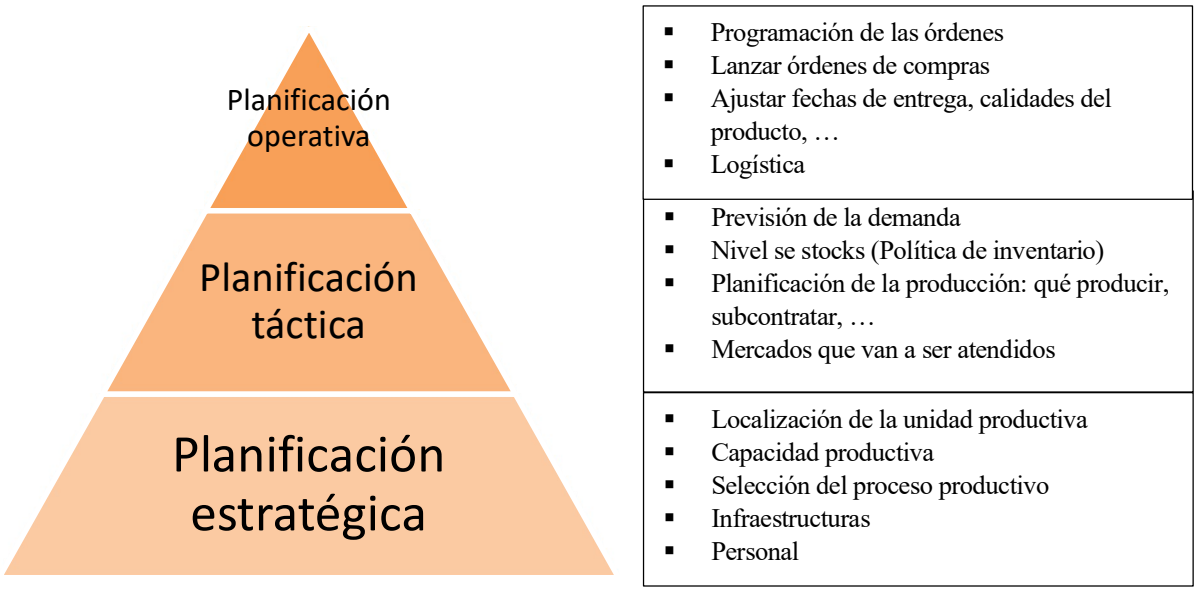

Ilustración 1. Estrategias de planificación

Es importante contextualizar las características propias de un sistema productivo, que es aquel cuyo objetivo es transformar recursos en productos aplicándoles una transformación. Los recursos que se

transforman pueden ser materias primas, con la mano de obra de los trabajadores ubicados en unas instalaciones utilizando energía y otros suministros. El producto final obtenido tras someter los recursos a una transformación (proceso productivo) debe cumplir con los requerimientos de calidad y cantidad, así como con los tiempos de entrega.

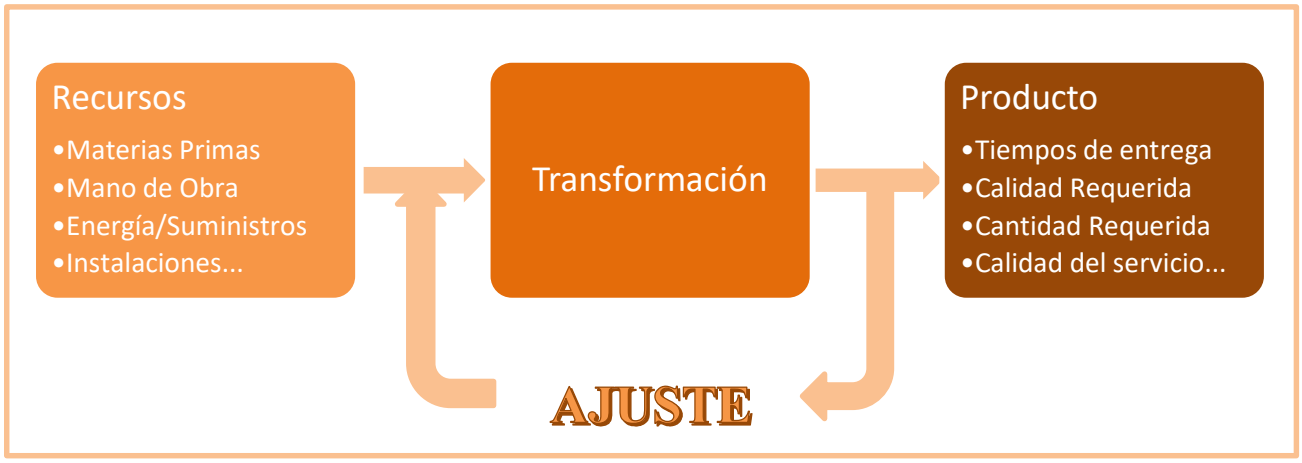

Ilustración 1. Esquema Sistema Productivo (Fuente: Elaboración propia)

El ajuste que aparece en el esquema hace referencia al conjunto de mecanismos que se llevan a cabo en la empresa con el fin de controlar que el producto cumple con las normas de calidad establecidas.

En el presente Trabajo Fin de Grado, se va a tratar un problema sobre programación de la producción basado en un entorno híbrido que consiste en un taller formado por dos etapas en el que cada uno de ellos es un entorno primario. Para este entorno, se van a considerar restricciones como tiempos de preparación de las máquinas (*setup*) entre trabajos y la compra a un proveedor externo de uno de los componentes necesarios para obtener el producto final. Se van a aplicar diferentes métodos aproximados para resolver el problema estudiado haciendo uso de un número determinado de instancias del problema. Se compararán distintas reglas de despacho y se probarán distintas versiones de una heurística adaptada con el objetivo de encontrar una buena solución para el problema.

Para ilustrar el problema de estudio, se puede tomar como ejemplo una célula de montaje de PCB (*Printed Circuit Board* o Placa de Circuito Impreso) en la que se fabrican en una primera etapa componentes que posteriormente son montados en la placa que es suministrada por un proveedor externo.

Si se relacionan las condiciones particulares del problema con los aspectos estratégicos mencionados anteriormente, podríamos decir que se trata de un proceso en línea en el que cada trabajo pasa por las dos etapas y en cuanto a la estrategia de fabricación, podría tratarse de estrategia *Make-To-Stock* que justifica que cada trabajo realice el proceso completo para almacenarse en inventario y estar disponible para cuando lo solicite el cliente.

1.1 Estructura del trabajo

Este Trabajo Fin de Grado se ha estructurado de la siguiente manera:

- Capítulo 2. Marco Teórico. Se presentarán los elementos básicos que componen un modelo de programación de la producción, así como la codificación de las soluciones. Se explicará la notación para identificar al modelo completo y la clasificación de los tipos de métodos de resolución.

- Capítulo 3. Descripción del problema. Se explicará el funcionamiento del problema a abordar, las restricciones del modelo y la función objetivo. Se presentarán unos datos de ejemplo para mostrar la configuración de los datos del problema y dos diagramas de Gantt para ilustrar el funcionamiento del modelo.

- Capítulo 4. Metodología. Se hará una breve revisión de los artículos estudiados relacionados con el problema descrito y se explicarán los métodos a aplicar.

- Capítulo 5. Análisis de resultados. Se analizarán los resultados obtenidos tras los experimentos en base a un indicador y se realizará una comparación entre métodos con la misma naturaleza.

- Capítulo 6. Conclusiones. Se reflexionará a cerca del trabajo realizado y de los resultados obtenidos.

2 UN PROBLEMA DE PROGRAMACIÓN DE LA PRODUCCIÓN

La programación de la producción no sólo tiene sentido en entornos de producción masivos o en los que el producto sea sólo tangible, lo tiene en cualquier sistema productivo en el que, como se ha explicado anteriormente, se transforman unos inputs para conseguir un producto. Este hecho, junto con la alta competitividad del mercado actual, acrecienta significativamente la importancia de esta disciplina.

Es importante saber que un problema de programación de la producción es un problema de optimización y que, por tanto, está compuesto de unas restricciones y una función objetivo que marca la dirección de búsqueda de la solución.

Para entender el modelo que se estudiará en este Trabajo Fin de Grado es preciso hacer un recorrido por la notación, las partes y la solución de un modelo de Programación de la Producción.

2.1 Elementos de un problema de programación de la producción

A continuación, se muestran los conceptos básicos que conforman un problema de programación de la producción:

- *M*, máquinas: Recurso productivo con capacidad para realizar operaciones sobre la materia prima.

En cualquier problema que se modele, las máquinas pueden representar desde máquinas reales de fabricación como pueden ser hornos, máquinas de corte, etc. a células de fabricación compuestas por operarios con herramientas o simples operarios.

Más adelante, se explicará que la disposición de las máquinas o características propias dan lugar a los distintos entornos posibles de un sistema productivo.

- *N*, trabajos: Total de productos terminados que se necesitan procesar.

Los trabajos pueden representar objetos tangibles, en el caso de industrias manufactureras, o intangibles como pedidos, referencias, gestiones, … Los trabajos son procesados en las máquinas y van transformándose a lo largo del sistema productivo.

- p_{ij}, tiempo de proceso: Es el tiempo que el trabajo *j* necesita ser procesado en la máquina *i*. Si

el tiempo de proceso no depende de la máquina se expresa como p_j

Para el análisis de un modelo concreto, los tiempos de proceso se conocerán a través de la experiencia o a través de la experimentación.

- r_j, fecha de llegada: Instante de tiempo en el que el trabajo *j* se puede comenzar a procesar.

Las fechas de llegada cobran especial importancia en problemas reales. Los modelos no son cajas negras aisladas del mercado y es usual que materias primas o componentes se suministren por proveedores externos o se subcontraten servicios o procesos, por lo que nuestro proceso estará limitado por el tiempo que tarden estos componentes o servicios en ser suministrados a nuestro sistema productivo.

- d_j, fecha de entrega: Instante de tiempo en el que el trabajo *j* debe estar completado.

Las fechas de entrega pueden indicarse por el cliente que reclame nuestro producto o directamente por la propia empresa siguiendo el Plan Maestro de Producción que impone la cantidad de producto (o servicio) que se necesita en cada horizonte temporal establecido.

- w_j, peso: Prioridad del trabajo *j* para ser procesado.

El peso puede representar distintas circunstancias del problema. Puede representar la importancia del cliente que recibe el trabajo, preferencia para finalizar un trabajo, etc. Estos pesos aparecen en el modelo generalmente en la función objetivo marcando la dirección en la que debemos conducir la búsqueda de la solución.

- R_j, ruta: Vector, para cada trabajo, que indica las máquinas en el orden por las que el trabajo *j* va siendo procesado.

2.2 Soluciones de un Modelo de Programación de la Producción

Como buena disciplina ingenieril, la programación de la producción pretende dar respuesta a un problema, que en este caso consiste básicamente en saber inicio y fin de cada operación sobre cada trabajo bajo ciertas restricciones impuestas por el mercado, la propia empresa, los recursos, … Como ante cualquier problema lo más importante es encontrar una solución y en este caso toma el nombre de "programa".

Un programa consiste en representar el tiempo que cada recurso opera sobre cada trabajo indicando principio y fin. Los programas, como cualquier otro tipo de solución, deben dar respuesta al modelo y, por tanto, cumplir con las restricciones de este. Al programa que cumple con las restricciones del

modelo se le llama "programa admisible".

Existen infinitos programas posibles (debido a que la escala temporal es infinita) por lo que para acotar el número de programas posibles se trabajará con programas semiactivos que son aquellos en los que el fin de una operación sobre un trabajo y el comienzo de la operación sobre otro se encuentran lo menos separado posible. Para poder plasmar en el programa en el orden correcto la transformación de cada trabajo es necesario seguir una "receta": la secuencia. La secuencia es una lista para cada una de las máquinas con todos los trabajos que procesa en el orden en el que lo hace.

Mientras que existen infinitos programas debido a que pueden iniciarse en cualquier instante de la escala temporal, las secuencias son finitas ya que establecen un orden entre los trabajos y, por tanto, existen tantas como combinaciones de trabajos sean posibles.

Ilustración 2. Proceso de tratamiento de un programa de Programación de la Producción
(Fuente: Elaboración propia)

En la resolución de un problema de programación de la producción entran en juego dos procesos muy importantes: la abstracción y la metodología de resolución. La primera de ellas consiste en el ejercicio de convertir las características del problema real y físico en la representación matemática cuyo procedimiento no se explicará en este trabajo, pero sí se tratarán los métodos de resolución más adelante.

2.3 Modelo $\alpha|\beta|\gamma$

Cuando se presenta un caso de estudio y una vez presentados los elementos básicos que pueden participar en el problema de programación de la producción es necesario hacer un ejercicio de abstracción para convertir el problema en el modelo correspondiente.

El modelo se construye siguiendo la notación establecida por Graham (1979). Siguiendo la notación del artículo mencionado, es posible conocer qué problema se está tratando, rellenando cada uno de los

campos del formato $\alpha|\beta|\gamma$.

¿Qué significan $\alpha, \beta\ y\ \gamma$? Las letras griegas $\alpha, \beta\ y\ \gamma$ hacen referencia al tipo de entorno, restricciones y función objetivo del modelo. A continuación, se explica detalladamente cada una de ellas.

- α, tipo de entorno**:** El tipo de entorno del modelo viene marcado por la cantidad de máquinas, la relación que guardan entre ellas y con los trabajos a procesar. En la literatura se encuentran estudios sobre entornos muy diversos, pero los más sencillos y sobre los que se construyen los más complejos son los siguientes:
    - $\alpha = 1$, una máquina: Es el caso más sencillo de todos. En este entorno sólo hay una máquina por lo que cada trabajo debe procesarse en ella.

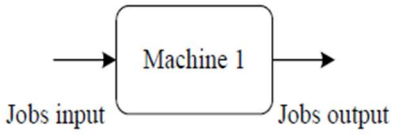

Ilustración 3. Esquema una máquina (Fuente: Libro de teoría de PCP)

Para construir el programa solución de este entorno es necesario establecer una secuencia.

En este entorno tan sencillo los trabajos pueden ordenarse de $n!$ formas distintas por tanto existen $n!$ secuencias distintas y, además en este caso, programas semiactivos.

- Máquinas paralelas: Como su nombre indica, es una disposición en la que las máquinas realizan de manera paralela alguna de las operaciones a los trabajos.

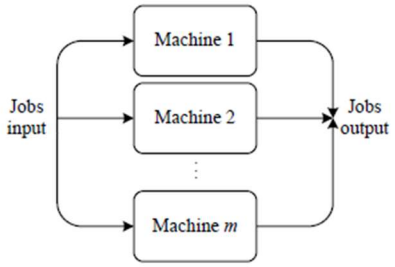

Ilustración 4. Esquema máquinas paralelas (Fuente: Libro de teoría de PCP)

Existen distintos tipos atendiendo a la relación entre los tiempos de proceso entre ellas.

- $\alpha = P_m$, Máquinas paralelas idénticas: Máquinas paralelas en las que el tiempo de proceso no depende de la máquina, es decir, no importa qué máquina

procese el trabajo ya que el tiempo de proceso será el mismo en cualquiera de ellas.

- $\alpha = Q_m$, Máquinas paralelas uniformes: Los tiempos de proceso entre estas máquinas paralelas guardan una relación de forma que el tiempo de proceso del trabajo j en la máquina i, p_{ij}, será el tiempo de proceso del trabajo p_j multiplicado por un factor que alargará el tiempo de proceso (en máquinas más deficientes) o lo acortará (en máquinas más eficientes).
- $\alpha = U_m$, Máquinas paralelas no relacionadas: Es el caso más general de todos ellos en el que los tiempos de proceso son distintos en cada máquina por lo que los datos sobre los tiempos de proceso se podrán representar en una matriz como la siguiente:

Tabla 1. Ejemplo de matriz de tiempos de proceso

$\frac{n}{m}$	1	2	3	4
1	4	5	8	3
2	5	4	7	9
3	3	6	7	6

Para identificar qué trabajos se procesan en cada máquina, sería necesario establecer una secuencia para cada máquina, pero en la práctica se simplifica mediante una secuencia general y una regla de asignación, explicadas posteriormente.

- Entornos tipo taller**:** Entorno en el que se encuentran máquinas con propósitos diferentes por lo que cada trabajo debe pasar por cada una de ellas. En este tipo de entorno participan las rutas, vector para cada trabajo que indica el orden en el que visita las máquinas.

  Se pueden encontrar los siguientes entornos tipo taller atendiendo a la configuración de las rutas:

  - $\alpha = F_m$, *Flowshop* o Taller de flujo:

    Es el caso en el que todos los trabajos tienen la misma ruta, es decir, siguen el mismo orden de máquinas que visitan. Este entorno sería el entorno adecuado

para la fabricación de productos en masa.

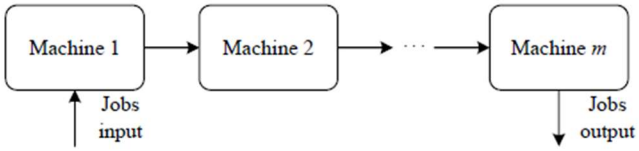

Ilustración 5. Esquema *Flowshop* (Fuente: Libro de teoría de PCP)

La s eran $(n!)^m$ programas semiactivos, sin embargo, existe un caso particular en el que el número de soluciones se simplifica: el *Flowshop* de permutación.

En el *flowshop* de permutación los trabajos comparten la misma ruta y, además, las máquinas la misma secuencia.

- $\alpha = J_m$, *Jobshop* o Taller de trabajos:

  En los talleres de trabajo cada trabajo tiene una ruta diferente como ocurriría en el caso en el que importa el orden en el que se vayan aplicando las transformaciones y puedan dar lugar a productos distintos.

  La ruta de los trabajos es un dato necesario.

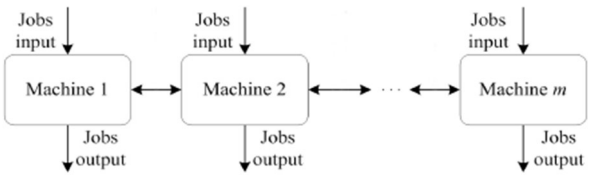

Ilustración 6. *Esquema Jobshop* (Fuente: Libro de teoría de PCP)

La solución en este caso se expresa con una secuencia extendida en la que cada trabajo aparece *m* veces en la secuencia y el proceso de asignación consiste en ir tomando cada trabajo de la secuencia extendida e ir asignando a la máquina que indica la ruta.

- $\alpha = O_m$, *Openshop* o Taller abierto:

  Es el caso más general y complejo de todos los talleres debido a que no hay una ruta predeterminada.

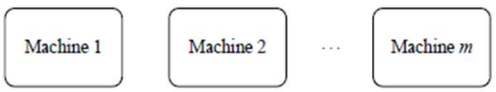

Ilustración 7. Esquema *Openshop* (Fuente: Libro de teoría de PCP)

La solución se expresa también con una secuencia extendida formada de $n \times m$ elementos, pero en este caso la ruta se construye durante el proceso de asignación.

- Entornos híbridos: Son los más comunes en la realidad y son producto de la combinación de los anteriores. Son talleres en los que cada una de las etapas (*stages*) está formada por una única máquina o por máquinas paralelas. La notación de este tipo de entorno se verá más adelante con el desarrollo del modelo a analizar.

- β, restricciones: Las restricciones son las condiciones bajo las que opera el modelo y que la solución debe cumplir.
  - $\beta = \emptyset$, suposiciones generales: Es el caso más general de todos. No hay ninguna restricción y se suponen las siguientes condiciones:
    - Los trabajos se encuentran disponibles al comienzo de la programación.
    - Los trabajos no se pueden interrumpir.
    - Las máquinas se encuentran siempre disponibles.
    - Cada máquina puede hacer un único trabajo a la vez.
    - El buffer (cola) entre máquinas se supone infinito.
    - Se desprecia el tiempo de transporte.
  - Fechas de llegada (r_j) y/o de entrega (d_j): Estos datos indican cuando el trabajo *j* está disponible para ser procesado y cuando debe entregarse respectivamente
  - Interrupción (*preemption*): Esta restricción indica que los trabajos pueden interrumpirse una vez que han empezado a procesarse. Puede ocurrir debido a la indisponibilidad de las máquinas o por el sistema.
  - Tiempos de preparación (o de *setup*): Son tiempos que necesita la máquina para prepararse antes de ejecutar el trabajo. Pueden ser dependientes de la máquina, del trabajo, de la máquina y el trabajo, de la máquina y de la secuencia… Además, estos pueden ser anticipatorios (pueden iniciarse antes del instante de disponibilidad del trabajo) o no anticipatorios en caso contrario.

- Lotes (*batch*): Las máquinas son capaces de procesar a la vez *b* trabajos. La posibilidad de agrupar los trabajos en lotes implica tomar una decisión respecto al tiempo de proceso del lote: establecerlo como la suma de los tiempos de proceso que lo componen (lotes en serie, *s-batch*) o establecerlo como el mayor tiempo de proceso (lotes en paralelo, p-*batch*). Estas decisiones dependerán, como siempre, del sistema productivo.
- Precedencia (*prec*): Según esta restricción, un trabajo *j* no puede empezar a procesarse hasta que no se hayan completado sus predecesores. Este dato puede representarse a través de un grafo de precedencia.

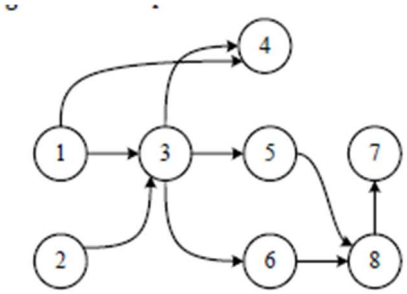

Ilustración 8. Grafo de precedencia (Fuente: Libro de teoría de PCP)

Además de las restricciones anteriores, existen otras que afectan a los entornos de tipo taller como pueden ser la restricción de permutación, no permitir tiempos ociosos en las máquinas, limitar el buffer de las máquinas, …

- γ, Función objetivo: Como todo problema de optimización se busca maximizar o minimizar una función objetivo. En programación de la producción, las funciones objetivo se clasifican según el objetivo del problema: minimizar costes, minimizar tiempos, aumentar flexibilidad, aumentar la calidad, … Algunos objetivos son:
  - L_{max}, Máxima tardanza o *Maximum Lateness*: Mide la desviación entre el tiempo de terminación del trabajo y su fecha de entrega.

    $$L_j = C_j - d_j$$

    Donde C_j es el tiempo de finalización del trabajo *j* y d_j es el tiempo de entrega del trabajo *j*
  - C_{max}, Tiempo máximo de finalización o *Makespan*: Hace referencia al tiempo requerido para procesar todos los trabajos.

    $$C_{max} = \max_{1 \le j \le n} C_j$$

Siendo C_j: Instante en el que el trabajo j termina de procesarse

- $\sum C_j$, Tiempo total de finalización: Es la suma del tiempo de terminación de todos los trabajos.
- $\sum U_j$, Número de trabajos retrasados: Total de trabajos que se han entregado tarde con respecto a sus fechas de entrega

2.4 Métodos de resolución

El método de resolución seleccionado para aplicar al problema de estudio es un punto de gran relevancia. Existen distintos métodos atendiendo a la calidad de la solución proporcionada. A continuación, se clasifican los métodos de resolución:

- Métodos exactos: Estos métodos son interesantes cuando nos encontramos ante un problema polinomial (P) y para algunos no polinomiales (NP) no demasiado grandes en cuanto a datos. Esto no significa que no puedan aplicarse a problemas *NP-Hard,* pero requieren un tiempo de cómputo que la mayoría de las veces no es realista para la situación que pretende resolverse. Dentro de este grupo encontramos:
  - Algoritmos constructivos exactos: son algoritmos que proporcionan la solución óptima de algunos problemas polinomiales reduciendo el tiempo de resolución en cuanto a lo que llevaría la comprobación de todas las combinaciones posibles. El algoritmo de *Johnson* o el algoritmo de *Lawler* son algunos ejemplos.
  - Algoritmos enumerativos: Como se presentaba anteriormente, un problema de programación de la producción se trata de un problema combinatorio por lo que la solución óptima podría obtenerse realizando todas las combinaciones posibles por lo que este tipo de algoritmo explora la totalidad de las posibilidades. Son algoritmos no polinomiales porque resuelven el problema en un número no polinomial de pasos. Para instancias grandes en el caso de problemas polinomiales y para no polinomiales el tiempo de cómputo deja de ser razonable.
    - Programación matemática: La resolución requiere expresar el modelo matemático del problema y resolverlo mediante un Solver.
    - *Branch and Bound:* Es un algoritmo diseñado para resolver problemas de optimización enteros. Consiste en resolver el problema lineal asociado y seleccionar alguna de las variables de decisión con valores fraccionarios e ir

añadiendo restricciones adicionales para obtener soluciones enteras.

- Métodos aproximados: Son aquellos que no exploran todo el campo de soluciones posibles si no que siguen una estrategia razonada teniendo en cuenta las características del problema con el objetivo de conseguir una solución factible, aunque sin garantizar la optimalidad. La eficiencia de estos métodos es comprobada experimentalmente y suele usarse para resolver sobre todo problemas *NP-hard* en tiempos pequeños.
  - Heurísticas constructivas: Son algoritmos que en un número finito de pasos construyen una solución añadiendo trabajos sin secuenciar a secuencias parciales. En este grupo también se encuentran las reglas de despacho y las reglas compuestas.
  - Metaheurísticas: Estos algoritmos parten de una solución inicial obtenida por otro método y aplican el concepto de vecindad. Las vecindades son todas las soluciones posibles que se pueden alcanzar con un movimiento local en la solución inicial. Algunas son *Simulated Annealing, Iterated Greedy, Tabu Search*, …

3 DESCRIPCIÓN DEL PROBLEMA

El problema de ensamblado planteado en este Trabajo Fin de Grado no se ha considerado hasta el momento, pero, en la literatura, se pueden encontrar otros problemas relacionados respondiendo a distintas restricciones. A continuación, se hará una breve revisión de los artículos en los que se consideran problemas relacionados y que han servido de guía para el desarrollo de este proyecto.

En Talens et al. (2020) se hace una revisión del problema general sin ninguna restricción. Se analiza el problema consistente en dos etapas, la primera procesa componentes pertenecientes a un trabajo que posteriormente se ensamblan en la segunda etapa. En este artículo, se proponen dos heurísticas constructivas para encontrar una solución que atienda a la función objetivo planteada que en su caso es el *total completion time*. De las dos, se adaptará la heurística llamada *Fast Constructive Heuristic* al problema con las restricciones que se han planteado en este proyecto.

Por otro lado, en Komaki & Kayvanfar (2015) se analiza el problema con tiempos de llegada para cada uno de los componentes en la primera etapa. En este artículo se proponen diferentes reglas de despacho y una metaheurística. De esta configuración del problema de ensamblado se ha considerado la idea para este proyecto de insertar tiempos de llegada, pero sólo para uno de los componentes de la primera etapa.

En tercer lugar, en Al-Anzi & Allahverdi (2006) se estudia el problema general sin restricciones con una sola máquina en la segunda etapa. Por último, en SupSung & Juhn (2009) se procesa un componente y otro es suministrado por un proveedor externo, que posteriormente se ensamblarán en la segunda etapa en una máquina de ensamblado.

El problema que se estudia en el presente documento se trata de un entorno híbrido compuesto por dos etapas (*stages*). La primera etapa, llamada A, estará compuesta por m_1 máquinas dispuestas paralelamente y cada una de ellas será la encargada de fabricar un componente diferente que formará parte de cada trabajo. El hecho de que cada máquina fabrique un componente distinto implica que pueden realizarse de manera simultánea componentes para el mismo trabajo en las distintas máquinas. Además, existe un último componente que es suministrado externamente por un proveedor. Tras procesarse cada uno de los componentes y recibirse el componente externo, todos ellos podrán ser ensamblados en la segunda etapa, llamada B.

En la segunda etapa se tendrán m_2 máquinas paralelas que realizan la misma actividad con los mismos tiempos de ensamblado cada una de ellas. En el ensamblado de componentes para cada trabajo, por

tanto, un trabajo visitará sólo una de las máquinas de la segunda etapa.

El objetivo establecido para la búsqueda de la solución es minimizar el *makespan* o tiempo máximo de finalización de todos los trabajos secuenciados.

El caso de estudio presentado se trata de un entorno híbrido en el que para dar una solución al problema es necesario proporcionar una secuencia y una regla de asignación que afectará a cómo se procesarán los trabajos en la segunda etapa.

Una regla de asignación es el criterio que se usa para determinar en qué máquina debe colocarse el trabajo que se esté asignando y en la realidad es un criterio que puede estar sujeto a características del sistema productivo, de la empresa u otras.

Algunas de ellas son:

- ECT, *Earliest Completion Time*: Implica asignar el trabajo a aquella máquina cuyo tiempo de terminación sea menor tras la asignación del trabajo. Es la regla que se usa por defecto.
- FAM, *First Available Machine*: Consiste en asignar el trabajo a la primera máquina disponible.
- Órdenes de etiqueta: Implica asignar el trabajo en la máquina 1, luego en la máquina 2, … hasta la *m*.

Teniendo en cuenta la función objetivo se escoge la regla *Earliest Completion Time*, es decir, se irán procesando los trabajos en la segunda etapa según estén disponibles para ser ensamblados y su tiempo de finalización sea menor.

Con la presentación del problema estudiado en este trabajo, se puede establecer la notación del problema $\alpha|\ \beta|\gamma$ establecida por Graham et al. (1979) y explicada en el apartado 2.3, teniendo en cuenta también la aportación que se hace en Framinan et al. (2019) para los modelos de *Assembly Scheduling Problems*. En este artículo se amplía la notación de Graham et al. (1979) para los casos de ensamblado, quedando con el siguiente formato:

$$\alpha_1 \rightarrow \alpha_2|\ \beta|\gamma$$

A continuación, se amplía el campo α para poder representar cada una de las etapas. Para el caso de estudio que se desarrolla en este trabajo, se tienen los siguientes campos:

- Entorno de la primera etapa: $\alpha_1 = DP_{m_1}$. Las siglas "DP" significan *dedicated parallel machines* y es la notación específica para el entorno descrito en la etapa A. El subíndice *m* hace

referencia al número de máquinas dedicadas que hay en la etapa.

- Entorno de la segunda etapa: $\alpha_2 = P_{m_2}$. Con lo descrito anteriormente se deduce que se trata de un entorno de máquinas idénticas paralelas.
- Restricciones: $\beta = r_j, s_{j,k_{ant}}$. En cuanto a las restricciones, el problema presenta tiempos de llegada para los componentes suministrados externamente y tiempos de *setup* anticipatorios dependientes de la secuencia para las máquinas de la segunda etapa.
- Función objetivo: $\gamma = C_{max}$. La función objetivo del problema consistirá en la minimización del tiempo máximo necesario para producir todos los trabajos.

$$DP_{m_1} \rightarrow P_{m_2} \left| r_j, s_{j,k_{ant}} \right| C_{max}$$

Para ilustrar la configuración de los datos del problema, se plantea un ejemplo sencillo con los siguientes datos:

N= [1, 2, 3, 4, 5] $m_1 = 2$ $m_2 = 2$ $r_j = [8, 9, 26, 16, 30]$

Los tiempos de proceso en la primera etapa, $p_{i,j}$, se presentan en la Tabla 2, y los tiempos de ensamblado son $at_j = [6\ , 7, 8\ , 5\ , 6]$.

Tabla 2. Tiempos de proceso en la primera etapa para la instancia de ejemplo

$\frac{n}{m_1}$	1	2	3	4	5
1	7	8	6	5	6
2	5	4	3	3	4

Los tiempos de preparación, $s_{j,k_{ant}}$, se presentan en la Tabla 3, en la que la primera línea de la matriz representa el tiempo de puesta en marcha necesario en cada máquina, en el caso de que el trabajo correspondiente sea el primero en cualquiera de las máquinas de la segunda etapa.

Tabla 3. Tiempos de *setup* dependientes de la secuencia para la instancia de ejemplo

$\frac{j}{k}$	1	2	3	4	5
∅	3	2	1	4	2
1	2	1	2	3	4
2	3	2	1	2	3
3	4	2	2	3	1
4	1	2	2	3	4
5	2	3	4	2	2

Con todos los datos anteriores se pueden representar algunas secuencias para mostrar su representación gráfica a través de un diagrama de Gantt.

- Primera secuencia: s_1 = [4,1,3,2,5]

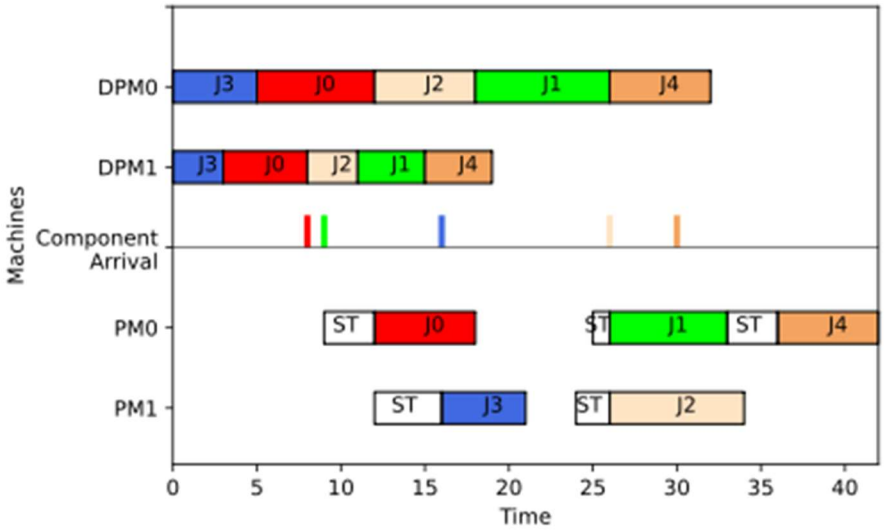

Ilustración 9. Diagrama de Gantt de la instancia de ejemplo. (Fuente: Elaboración propia)

Nota: En el diagrama de Gantt presentado en la Ilustración 10 los trabajos presentan una unidad menos que la numeración seguida en el análisis.

Como se puede apreciar en el diagrama de Gantt, en la primera etapa se representan dos máquinas paralelas en la que se procesan en cada una de ellas uno de los componentes pertenecientes a cada

trabajo. Al ser componentes diferentes implica que se puedan procesar componentes pertenecientes al mismo trabajo de manera simultánea. En la primera etapa, también se representa la llegada del componente suministrado externamente mediante un pulso para visualizar el instante en el que todos los componentes de un trabajo están listos para ser ensamblados en la segunda etapa.

A continuación, se representan las máquinas ensamblado, que como se ha indicado anteriormente son máquinas paralelas idénticas. La particularidad de la segunda etapa son los tiempos de preparación necesarios entre el ensamblado de un trabajo y otro representados por "ST".

Por último, es fácil identificar el instante de fin del último trabajo en ser ensamblado que se corresponde con el valor de la función objetivo.

El proceso para la resolución del problema se detalla a continuación:

1. Se denota *j* como el trabajo procesado en la posición *j* en la secuencia. El tiempo máximo de finalización del trabajo *j* en la primera etapa se calcula como:

$$C1_j = max_{1\leq i<m_1}\{r_j, C_{ij}\} = max_{0\leq i<m_1}\{r_j, max_{0\leq i<m_1}\{p_{ij} + C_{i,j-1}\}\} \quad (1)$$

   Donde $C_{i,0} = 0$.

2. Una vez calculados los tiempos de finalización de los trabajos en la primera etapa, se asignan los trabajos en las máquinas de ensamblado siguiendo la regla de asignación elegida, ECT, teniendo en cuenta el tiempo de finalización del trabajo pendiente de asignar en la segunda etapa.

   Se define $i^* \; como \; i^* = argmin_{i=0,\dots,m_2}\{C2_{i,j}\}$ siendo *j* el último trabajo procesado en la máquina i y *k* el trabajo pendiente de asignar:

$$C2_{i,k} = \begin{cases} \mathrm{C1_k} + at_k \;\; si \; \boldsymbol{C1_k - s_{j,k} > C2_{i^*,j}} \; si \; i = i^* \\ C2_{i^*,j} + s_{j,k} + at_k \;\; si \;\; \boldsymbol{C1_k \leq \; C2_{i^*,j}} \; si \; i = i^* \\ C_{i,j} \; en \; caso \; contrario \end{cases} \qquad \forall \; k = 1, \dots, n \qquad (2)$$

   Donde $C_{i,0}$= 0 $\forall \; i = 1 \dots m_2$

   Por último, el tiempo de finalización asociado a cada trabajo será:

$$C_k = C_{i^*,k} \; \forall \; i = 1, \dots, m_2 \; \forall \; k \qquad (3)$$

   y la función objetivo queda:

$$FO = \min\left(max_{j=1,\dots,N} \; C_j\right) \qquad (4)$$

- Si se propone una segunda secuencia: s_2= [1, 4, 2, 3, 5], el diagrama de Gantt resultante es el siguiente:

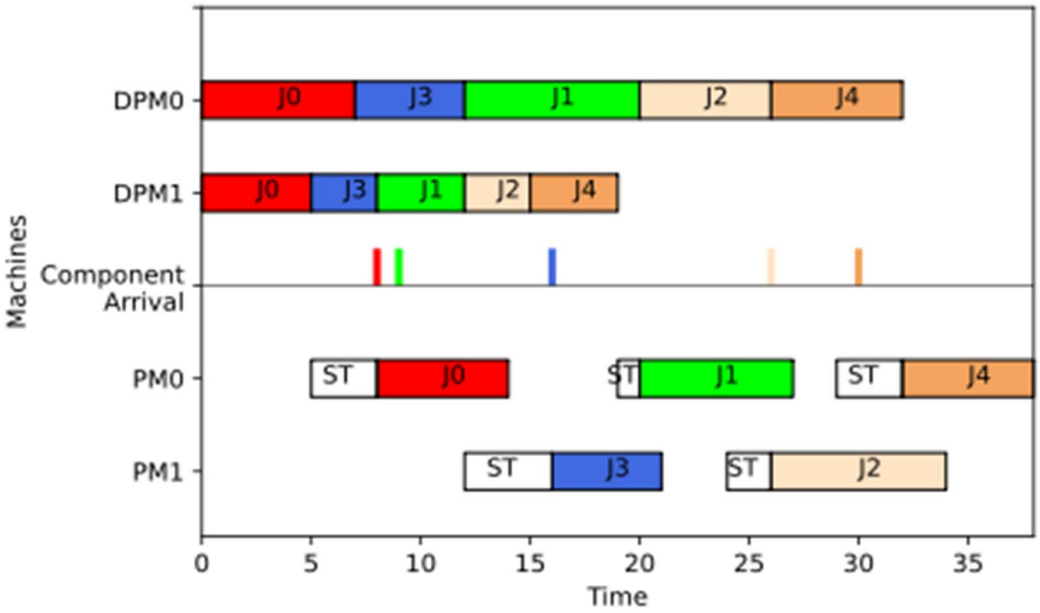

Ilustración 10. Diagrama de Gantt de la segunda secuencia de ejemplo (Fuente: Elaboración propia)

En comparación con la primera secuencia representada, se observa que los cambios más significativos se dan en la segunda etapa. En este caso, los trabajos comienzan a ensamblarse antes, lo que implica que un orden diferente en el procesamiento de trabajos da lugar a distintas configuraciones de la segunda etapa y, por tanto, distinto valor de la función objetivo. Por otro lado, también se observan intervalos mayores entre el ensamblado de un trabajo y del siguiente. Por último, se comprueba que el valor de la función objetivo en este caso es ligeramente menor que el proporcionado por la primera secuencia. Todas estas características serán de vital importancia para diseñar el método de resolución

4 METODOLOGÍA

Al evaluar una secuencia diferente en el anterior apartado, se ha obtenido un mejor resultado de la función objetivo. Un aspecto fundamental es escoger una metodología de resolución que nos proporcione secuencias con buenos resultados. El criterio para escoger aplicar una metodología de resolución u otra se basa en el tiempo disponible para obtener un resultado. Por su parte, el tiempo disponible para obtener un resultado se refleja en el tiempo computacional que necesita la metodología para proporcionar una solución que a su vez depende directamente de las características del problema.

4.1 Reglas de despacho

Las reglas de despacho que aplicaremos serán las propuestas por Komaki & Kayvanfar (2015). Siguiendo la clasificación del artículo citado, se dividen las reglas de despacho en los siguientes grupos:

- Grupo I: Reglas de despacho fundamentadas en ordenar los índices, calculados para cada trabajo, en orden creciente según los tiempos de llegada y tiempos de proceso en la primera etapa. Para el cálculo de los distintos índices se tienen en cuenta los tiempos de proceso (p_{ij}) y las fechas de llegada (r_j).

    - I_1: En el artículo anteriormente mencionado, se distingue entre dos formas de ordenar los trabajos teniendo en cuenta los tiempos de llegada en cada una de las máquinas de la primera etapa. La primera de ellas consiste en calcular el índice de cada trabajo tomando el valor máximo de tiempo de llegada de entre todos los componentes pertenecientes a un mismo trabajo y la segunda tomando el valor mínimo. En el problema abordado, no hay fechas de llegada para todos los componentes, únicamente para el componente adquirido a un proveedor externo, por lo que sólo hay asociada una fecha de llegada a cada trabajo. Por tanto, se resume el primer y segundo índice en el siguiente:

    $$I_1(j) = r_j \qquad \forall\, j \tag{5}$$

    Una vez calculados los índices para cada trabajo, se ordenan en orden creciente según el valor de su índice.

    - I_2: Este indicador consiste en tomar para cada trabajo el máximo entre el tiempo de llegada del componente suministrado externamente y el mayor tiempo de proceso de

entre los componentes que se procesan.

$$I_2(j) = \max\left\{r_j, max_{i=1,\dots,m_1}\{p_{ij}\}\right\} \quad \forall\, j \qquad (6)$$

- I_3: Usando las últimas ideas se calcula este índice como el máximo entre el tiempo de llegada del tercer componente y el tiempo de proceso medio de los componentes procesados.

$$I_3(j) = \max\left\{r_j, \sum_{i=1}^{m_1} \frac{p_{ij}}{m_1}\right\} \;\forall \quad j \qquad (7)$$

- I_4: De la misma manera que el anterior, pero tomando el mínimo de entre la fecha de llegada del componente externo y el tiempo de proceso medio.

$$I_4(j) = \min\left\{r_j, \sum_{i=1}^{m_1} \frac{p_{ij}}{m_1}\right\} \;\forall \quad j \qquad (8)$$

- I_5: El cálculo de este indicador es el antagónico al I_2, es decir, se toma el mínimo de entre las opciones.

$$I_5(j) = \min\left\{r_j, max_{i=1,\dots,m_1}\{p_{ij}\}\right\} \;\forall\; j \qquad (9)$$

- I_6: Para calcular este indicador se necesita tener en cuenta la carga de trabajo de cada una de las máquinas de la primera etapa y seleccionar, para cada trabajo, el tiempo de proceso que pertenezca a la máquina con mayor carga.

$$I_6(j) = \max\left(r_j, p_{i^* j}\right) \;\forall\; j \qquad (10)$$

Donde $T_i = \sum_{j=1}^{n} p_{ij} \;\forall\; i$ tomando como i^* aquella máquina con mayor T_i.

Una vez calculados para cada uno de los casos los índices, correspondientes a cada trabajo, se ordenan los trabajos en la secuencia en orden creciente según el valor de sus índices.

- Grupo II: Reglas de despacho fundamentadas en minimizar el tiempo ocioso (o *idle time*) de la segunda etapa, ordenando los trabajos en orden decreciente según el valor de los índices calculados. En este grupo, se tienen en cuenta los tiempos de proceso y de ensamblado.

  - I_7: Los indicadores se corresponden al tiempo de ensamblado de cada trabajo.

$$I_7(j) = at_j \;\forall\; j \qquad (11)$$

  - I_8: En este caso, se toma como valor del indicador el máximo de entre los tiempos de proceso en la primera etapa y se suma el tiempo de ensamblado.

$$I_8(j) = max_{i=1,\dots,m_1}\{p_{ij}\} + at_j \ \forall \ j \quad (12)$$

- I_9: Para calcular este índice se tiene en cuenta de nuevo el concepto de carga de trabajo en la primera etapa y se suma el tiempo de ensamblado.

$$I_9(j) = at_j + \sum_{i=1}^{m_1} \frac{p_{ij}}{m_1} \ \forall \ j \quad (13)$$

- I_{10}: El último índice de este grupo se calcula como el mayor tiempo de proceso de entre los tiempos de proceso de la primera etapa más el tiempo de ensamblado.

$$I_{10}(j) = p_{i^*j} + at_j \ \forall \ j \quad (14)$$

Donde $T_i = \sum_{j=1}^{n} p_{ij} \ \forall \ i$ tomando como i^* aquella máquina de la primera etapa mayor T_i.

De nuevo se genera la secuencia ordenando los trabajos según el valor de los índices proporcionados por cada regla, en este caso en orden decreciente.

- Grupo III: En último lugar, las reglas de despacho propuestas en este tercer grupo consideran los tiempos de proceso de la primera etapa, fechas de llegada y tiempos de ensamblado en la segunda etapa.

  - $I_{11}(j)$: Se toma el máximo de entre los tiempos de proceso en la primera etapa y el tiempo de llegada del tercer componente, y se suma el tiempo de ensamblado.

$$I_{11}(j) = \max\left\{r_j, max_{i=1,\dots,m_1}\{p_{ij}\}\right\} + at_j \quad (15)$$

  - $I_{12}(j)$: Para el cálculo de este indicador se toma el máximo de entre la carga de trabajo aportada por el trabajo en la primera etapa y el instante de llegada del componente suministrado externamente, y se suma el tiempo de ensamblado

$$I_{12}(j) = \max\left(r_j, \sum_{i=1}^{m_1} \frac{p_{ij}}{m_1}\right) + at_j \ \forall \ j \quad (16)$$

  - $I_{13}(j)$: Por último, este indicador se calcula teniendo en cuenta los instantes de llegada del componente suministrado externamente y el tiempo de proceso del componente perteneciente a la máquina de la primera etapa con mayor suma de tiempos de proceso.

$$I_{13}(j) = \max\left(r_j, p_{i^*j}\right) + at_j \quad \forall \ j \quad (17)$$

Donde $T_i = \sum_{j=1}^{n} p_{ij} \ \forall \ i$ tomando como i^* aquella máquina de la primera etapa con mayor T_i.

Por último, se vuelven a secuenciar los trabajos considerando los índices en orden

creciente.

Se puede resaltar la imposibilidad de considerar los tiempos de *setup* en las reglas de despacho adaptadas de Komaki & Kayvanfar (2015) ya que primero se calcula el indicador, luego se ordenan y, posteriormente, se determina el trabajo a asignar. Por tanto, no es posible conocer el tiempo de preparación de antemano.

4.2 Heurística Constructiva

Tras aplicar las reglas de despacho, la siguiente metodología a aplicar es un procedimiento heurístico. Se elige la heurística constructiva *CH* propuesta en Talens et al. (2020) para ser adaptada y resolver el problema estudiado en este trabajo.

La idea de la heurística *CH,* tal y como se explica en Talens et al. (2020), consiste en construir una secuencia de forma recurrente seleccionando trabajos que aún no hayan sido secuenciados e ir añadiéndolos a la secuencia parcial. Como también se indica en el artículo mencionado, la clave de estos procedimientos algorítmicos está en diseñar un indicador ψ que mida la idoneidad de un trabajo que no ha sido secuenciado para ser el próximo en ser añadido a la secuencia parcial.

El indicador ψ consistirá en una función lineal dependiente del tiempo ocioso (o *idle time*) inducido y de otro indicador CT al añadir el trabajo que se esté considerando, definido como la contribución total que supone el trabajo. Además de todo esto, se añade un coeficiente *a* que aportará mayor o menor importancia al *idle time* que suponga el trabajo. Este coeficiente se obtiene experimentalmente y en Talens et al. (2020) se probaron distintos valores de *a* para un conjunto de 10 instancias resultando generar mejores soluciones a = 5. Por tanto, las pruebas del algoritmo se harán con el coeficiente igualado a este valor. Además, para adaptar la heurística a las restricciones delproblema considerado en este trabajo, deben introducirse las fechas de llegada y los tiempos de *setup* anticipatorios dependientes de la secuencia.

En primer lugar, las fechas de llegada pasarán a formar parte del cálculo del tiempo de finalización en la primera etapa del trabajo que se esté evaluando (ω_j). Por tanto, C_1 se calcula de la siguiente manera:

$$C_1(\omega_j) \coloneqq \max\left\{\max_{1\leq i\leq m_1}\left\{C1_i + p_{i,\omega_j}\right\}, r_j\right\} \quad (18)$$

Tiene lógica tomar como instante final de proceso en la primera etapa para cada trabajo el valor que sea mayor entre el tiempo de finalización de los componentes que se procesan o de llegada del componente adquirido externamente.

En segundo lugar, los tiempos de *setup* pasarán a formar parte del indicador de *idle time* (IT) y carga de trabajo CT. En el caso del cálculo del IT, será el máximo entre el tiempo de finalización del trabajo en la primera etapa menos el menor tiempo de finalización (*completion time)* de los últimos trabajos secuenciados en la segunda etapa menos el *setup* correspondiente y cero.

$$IT_{\omega_j} = \max\left\{C_1(\omega_j) - C2_{i^*} - s_{\omega_r,\omega_j}, 0\right\} \quad (19)$$

Por último, para probar la eficacia de esta heurística constructiva, se programan cuatro versiones modificando en cada una de ellas el indicador de carga. El pseudocódigo de la heurística adaptada versión V1 queda así:

1 $Procedimiento\ Heurística\ Constructiva\ CH\_V1$
2 //Inicialmente todos los trabajos están sin secuenciar
3 $\pi := \emptyset$
4 //Los tiempos de finalización del último trabajo en las máquinas de las dos etapas:
5 $C1_i := 0 \quad i = 1, \dots, m_1$
6 $C2_i := \quad i = 1, \dots, m_2$
7 $i^* := \arg\min_{1 \le i \le m_2} C2_i$
8 Σ={1,2,...,n}
9 $\boldsymbol{for}\ j = 1\ \boldsymbol{to}\ n\ \boldsymbol{do}$
10 $\boldsymbol{for}\ each\ \omega_j\ \epsilon\ \Sigma$ **do**
11 //Calcular el tiempo de finalización en la primera etapa para el trabajo ω_j:
12 $$C_1(\omega_j) := \max\left\{\max_{1 \le i \le m_1}\left\{C1_i + p_{i,\omega_j}\right\}, r_j\right\}$$
13 //Calcular el tiempo ocioso que induciría en la segunda etapa en caso de ser asignado:
14 $$IT_{\omega_j} = \max\left\{C_1(\omega_j) - C2_{i^*} - s_{\omega_r,\omega_j}, 0\right\}$$
15 //Calcular la contribución total que supondría el trabajo ω_j con el indicador CT:
16 $$CT_{\omega_j} = \frac{at_{\omega_j}}{m_1 * m_2}$$
17 //Calcular el indicador de idoneidad para ser seleccionado como trabajo para añadir a la secuencia
18 $$\psi_{\omega_j} := \boldsymbol{a} * IT_{\omega_j} + CT_{\omega_j}$$
19 $\boldsymbol{end}$

20 $r \coloneqq \arg \min_{1 \le k \le n-j+1} \psi_k$;

21 Añadir ω_r al final de π, quedando por ejemplo $\pi \coloneqq (\pi_1, \ldots, \pi_{j-1}, \omega_r)$;

22 Extraer el trabajo ω_r del conjunto Σ;

23 //Actualizar los valores de los tiempos de finalización en las dos etapas

24 $C1_i \coloneqq C1_i + p_{i,\omega_r}$

25 // Si el tiempo de finalización del último trabajo secuenciado en la segunda etapa es menor que la diferencia entre el tiempo de finalización del trabajo en la primera etapa y *setup* correspondiente, se calcula:

26 $C2_{i^*} \coloneqq \max_{1 \le i \le m_1} \{C1_i, r_{\omega_r}\} + at_{\omega_r}$

27 // Si no, se calcula**:**

28 $C2_{i^*} \coloneqq C2_{i^*} + s_{\omega_{r-1},\omega_r} + at_{\omega_r}$

29 $i^* \coloneqq \arg \min_{1 \le i \le m_2} C2_i$

30 ***end***

Figura 1. Pseudocódigo de la heurística CH adaptada

Sobre la heurística adaptada se estudiarán las siguientes versiones que afectarán al cálculo del indicador de carga de trabajo.

CH_V2: Sólo asocia el tiempo de ensamblado a las máquinas de la segunda etapa sin tener en cuenta los tiempos de *setup.*

$$CT_{\omega_j} = \frac{at_{\omega_j}}{m_2} \quad (20)$$

CH_V3: La versión V3 de la heurística propuesta mantiene el cálculo de la carga de trabajo repercutido a todas las máquinas, pero teniendo en cuenta los tiempos de *setup.*

$$CT_{\omega_j} = \frac{at_{\omega_j} + s_{\omega_r,\omega_j}}{m_1 * m_2} \quad (21)$$

CH_V4: La última versión es análoga a la versión V3, pero asociándolo sólo a las máquinas de la segunda etapa:

$$CT_{\omega_j} = \frac{at_{\omega_j} + s_{j,k}}{m_2} \quad (22)$$

4.3 Búsqueda Local

A partir de las secuencias proporcionadas por las distintas versiones de la heurística, se analizará cómo afecta la aplicación de un método de búsqueda local. Los métodos de búsqueda local consisten en evaluar los vecinos de una secuencia dada estableciendo o no un criterio de parada. En este caso, se ha establecido el siguiente criterio de parada:

$$Número\ de\ iteraciones = \frac{n^2}{5} \qquad (23)$$

La razón para establecer criterio de parada y no evaluar todos los vecinos de una secuencia es limitar el tiempo de cómputo, ya que el objetivo de la aplicación de una búsqueda local es corroborar que la combinación de metodologías proporciona una mejora significativa de la solución.

En la búsqueda local, la generación de vecinos consiste en generar una secuencia a partir de una dada haciendo intercambio de posiciones entre pares de trabajos o insertando cada trabajo en cada una se las posiciones.

Las más conocidas son:

- *Adjacent Swap*: Se intercambian las posiciones de los trabajos contiguos. Dada una secuencia de *n* trabajos se generan *n*-1 vecinos de la secuencia primaria.

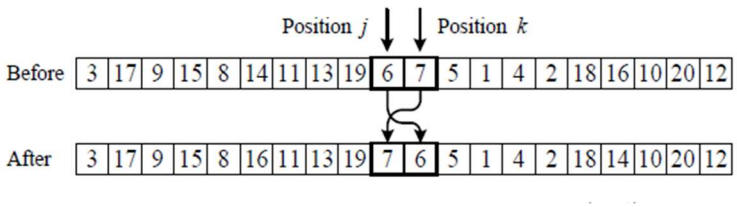

Ilustración 11. Esquema *Adjacent Swap* (Fuente: Libro de PCP)

- *General Swap*: Se toma cada trabajo y se cambia de posición con el resto de los trabajos. Se generan $\frac{n(n-1)}{2}$ vecinos

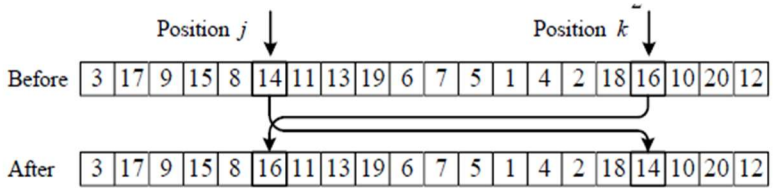

Ilustración 12. Esquema *General Swap* (Fuente:Libro de PCP)

- *Insertion:* Se toma cada trabajo y se coloca en todas las posiciones posibles. Se producen $(n-1)^2$ vecinos.

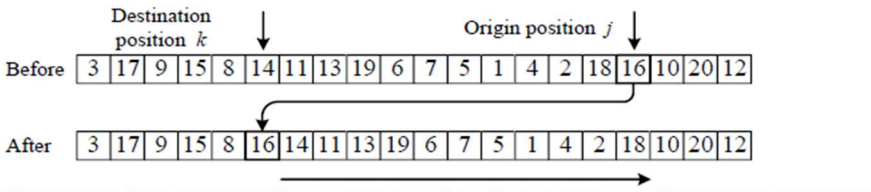

Ilustración 13. Esquema *Insertion* (Fuente:Libro de PCP)

En este caso, se realizará la búsqueda por inserción, ya que proporciona mayor número de vecinos. En cada iteración se evaluará el vecino correspondiente y, en caso de que proporcione mejor valor de la función objetivo, esta se actualizará. Por último, el método devolverá el mejor valor de la función objetivo de entre todos los vecinos considerados y, en caso de no encontrar una secuencia mejor, devolverá el de la secuencia primaria. El método descrito se conoce como *Best Improvement* y el pseudocódigo de la adaptación aplicada es:

Adaptación de Best Improvement con criterio de parada establecido

Input: datos de la instancia, π (secuencia primaria)
Output: obj_b
//Comienzo:
$obj_b := obj(\pi)$
$parada := \frac{n^2}{5}$
$iteraciones := 0$
for $j = 1$ **to** n **do:**
    for $k = 1$ **to** n **do:**
        Si $iteraciones < parada$:
            Si $j \neq k$ y $k \neq (j-1)$:
                Vecino: = π
                //elimina j de vecino
                //inserta k en el lugar de j
                $iteraciones = iteraciones + 1$
        Si $obj' < obj_b$:
            $obj_b = obj'$
Return obj_b

Ilustración 14. Pseudocódigo de la búsqueda local aplicada

5 ANÁLISIS COMPUTACIONAL

En este capítulo se explicará la generación de los problemas que se someterán a los experimentos y, posteriormente, se analizarán los resultados obtenidos. Para evaluar la calidad de cada uno de los métodos aplicados, se examinarán por separado según su naturaleza, es decir, en primer lugar, se revisarán los resultados aportados por las reglas de despacho aplicadas y se compararán entre todas ellas y, en segundo lugar, se observarán los resultados obtenidos por las heurísticas constructivas y la mejora aportada por el método de búsqueda local.

Para determinar la calidad de los métodos, se establece un indicador que será el Porcentaje de Desviación Relativa Media o *Average Relative Percentage Desviation* (ARPD):

$$ARPD_h = \frac{\sum_{\forall s} RPD_{hs}}{S}, \forall\ h = 1, \dots, H \qquad (24)$$

Donde H es el total de los métodos aplicados y S el total de instancias analizadas. Este indicador es el promedio del Porcentaje Relativo de Desviación (RPD) calculado para todos los resultados obtenidos de cada método para todas las instancias:

$$RPD_{hs} = \frac{FO_{hs} - FO^*}{FO^*} \cdot 100, \forall\ h = 1, \dots, H\ y\ s = 1, \dots, S \qquad (25)$$

Donde FO^* se corresponde con el valor mínimo de la función objetivo obtenido para cada instancia de entre todos los métodos aplicados.

Para valorar si la inversión de tiempo en encontrar una solución mejor al incorporar búsqueda local se define el indicador ACPU:

$$ACPU_h = \frac{\sum_{\forall s} T_{hs}}{S} \qquad (26)$$

Donde T_{hs} es el tiempo (en segundos) requerido por el procedimiento heurístico h para obtener una solución para la instancia s.

5.1 Generación de Instancias

Para comprobar qué métodos aplicados proporcionan mejores resultados al modelo planteado es necesario probarlos en problemas con distintos datos. Los problemas o instancias serán generadas a partir de un programa escrito en lenguaje *Python*. El número de trabajos se genera con los siguientes niveles n= {30, 40, 50, 60, 70} mientras que las máquinas de la primera etapa son $m_1 = \{2, 4, 6, 8\}$

y las de la segunda $m_2 = \{1, 2, 3\}$. Los tiempos de proceso en la primera etapa se generan según U[1,100]. Por otra parte, los tiempos de ensamblado siguen una distribución U[1,100·m_2·2] como en Talens et al.(2021). Por último, para las fechas de llegada del componente suministrado externamente se generan tres escenarios distintos como en Sung & Juhn (2009) dependiendo de un valor α. Por tanto, de una instancia con determinados valores de *n,* m_1*,* m_2*,* p_{ij}*,* at_j *y* s_{jk} tendremos tres posibles vectores de tiempos de llegada.

Así, para cada instancia, los datos se generan de la siguiente manera:

$$p_{ij} \in U[1, 100]$$

$$at_j \in U[1, 100 \cdot m_2 \cdot 2]$$

$$s_{j,k} \in U[1, 20]$$

$$r_j \in U[1, \alpha P], \text{con } P = \max\left(\sum_{i=1}^{m_1} p_{ij}\right) \text{ y } \alpha = \{0.6, 0.8, 1\}$$

La razón por la que se usan distribuciones uniformes con un ancho intervalo es porque la varianza de esta distribución es grande y si un método funciona bien con datos generados a partir de esa distribución entonces implicará que también lo hará con cualquier otra como se expresa en Al-Anzi & Allahverdi (2006).

Además, cada combinación de datos se replicará cinco veces con el fin de analizar también cómo afectan los métodos propuestos dependiendo de la combinación de datos. Con todo esto se analizarán un total de 900 instancias $(5(n)\ x\ 4(m_1)\ x\ 3(m_2)\ x\ 3(\alpha)\ x\ 5(réplicas) = 900)$.

5.2 Análisis de los resultados

5.2.1 Reglas de Despacho

Tras ejecutar las reglas de despacho propuestas sobre la batería de instancias se han obtenido los resultados presentados en la siguiente tabla con respecto al mínimo proporcionado por las reglas de despacho:

Tabla 4. ARPD por cada regla de despacho

RD	ARPD	RD	ARPD	RD	ARPD
I_1	4.603	I_6	4.603	I_{11}	2.0107
I_2	4.603	I_7	0.7529	I_{12}	2.0107
I_3	4.603	I_8	3.871	I_{13}	3.6359
I_4	3.0426	I_9	2.7072		
I_5	3.8649	I_{10}	2.9884		

Tras la presentación de estos resultados calculados sobre todas las instancias, se observa que, de forma general, las reglas de despacho pertenecientes al primer grupo (de I_1 *a* I_6) proporcionan valores de la función objetivo (*makespan)* en torno al 4% de desviación con respecto al mínimo, las del segundo grupo (I_7 *a* I_{10}) valores con el 2,60% de desviación aproximadamente y, por último, las del último grupo que proporcionan valores alrededor del 2,50% de desviación.

Analizados los valores por grupos de reglas de despacho establecidos en función de los datos considerados para formar el indicador, se observa que en general las del tercer grupo donde se consideran datos de las dos etapas proporcionan mejores resultados. Sin embargo, analizando por separado cada una de las reglas de despacho destaca significativamente la I_7 cuyo porcentaje de desviación relativa media es de 0,75% con respecto al mínimo proporcionado por las reglas de despacho. Este hecho tiene su explicación en la generación de los tiempos de ensamblado ($at_j \in U[1, 100 \cdot m_2 \cdot 2]$*)* cuyo rango de datos es mucho mayor que los tiempos de proceso de la primera etapa, por lo que construir la solución teniendo en cuenta sólo los tiempos de ensamblado justifica la mejora en los resultados.

Si se analiza la evolución del valor promedio de cada regla de despacho en función del número de trabajos se diferencian claramente dos comportamientos diferentes en la evolución del ARPD.

Mientras que en instancias de un determinado número de trabajos se produce un empeoramiento con respecto a la cantidad de trabajos anterior, en los resultados proporcionados por ciertas reglas de despacho ocurre lo contrario. Para ejemplificar esto que sucede, se observa que las reglas $I_1, I_2, I_3, I_4, I_5, I_6, I_{11}, I_{12}$ *e* I_{13} (todas pertenecientes al primer y tercer grupo) proporcionan un valor promedio peor para instancias de 40 trabajos con respecto a las de 30 mientras que el resto de reglas se comportan de manera opuesta generando mejores resultados para las instancias de 40 trabajos. Este comportamiento opuesto también se observa en el cambio de 40 a 50 trabajos, pero de manera inversa, y, en este caso, las reglas del primer y tercer grupo mejoran con respecto a resultados para instancias de 40 trabajos.

Por tanto, podemos concluir que las reglas del primer y tercer grupo generan peores resultados para instancias de 40 trabajos respecto a las de 30, mejoran para instancias de 50 respecto a las de 40 y se mantienen prácticamente constantes entre las instancias de 60 y 70 trabajos. Sin embargo, ocurre lo contrario con las reglas del segundo grupo.

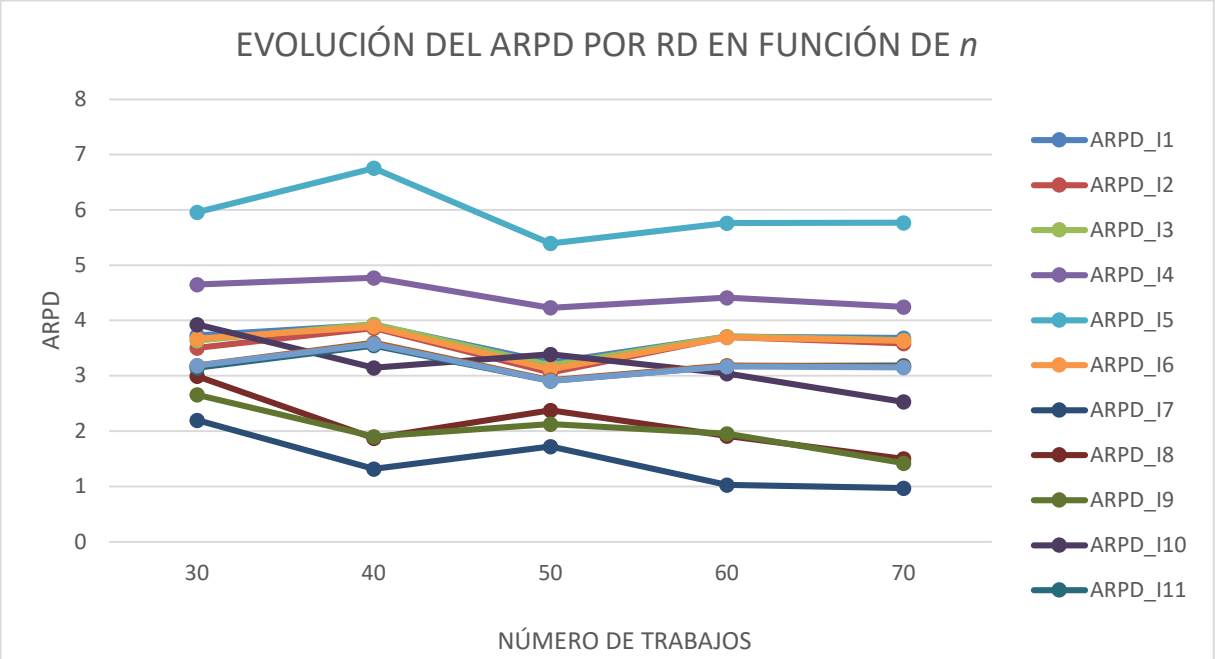

Ilustración 15. Gráfico sobre la evolución del ARPD en función del número de trabajos

Otro parámetro que puede revelar resultados interesantes es el parámetro α, con el que se han generado los valores de las fechas de llegada para el componente suministrado externamente.

En la gráfica que se muestra a continuación, se observan claramente dos tendencias: ascenso en el valor del ARPD y, por tanto, empeoramiento de las soluciones a medida que aumenta el valor de α o por el contrario descenso y por tanto mejora en los resultados según aumenta el valor de α. Por un lado, los ARPD de las reglas $I_1, I_2, I_3, I_6, I_{11}, I_{12}, I_{13}$ evolucionan de manera descendente a medida que

aumenta el valor de α, lo que indica que a medida que aumenta el valor de las fechas de llegada estas reglas generan mejores resultados. Esto se debe a que todas las reglas anteriormente mencionadas tienen en cuenta en el cálculo del indicador las fechas de llegada y que, al aumentar el valor de estas, tienen mayor peso en el modelo y, por tanto, más necesario se hace tenerlas en cuenta para construir la secuencia. Por otro lado, el ARPD de las reglas I_5, I_7, I_8, I_9 *e* I_{10} empeora a medida que aumenta el valor de α. En el caso de la I_5, el cálculo del indicador tiene más probabilidad de que se construya teniendo en cuenta el tiempo de proceso del componente, según aumenta el rango en el que se generan las fechas de llegada. El resto de reglas no tienen en cuenta las fechas de llegada, por lo que lógicamente, al aumentar el rango de valores que pueden tomar, pasan a tener mayor importancia en el modelo y, al no tenerse en cuenta, aumentan los valores de ARPD.

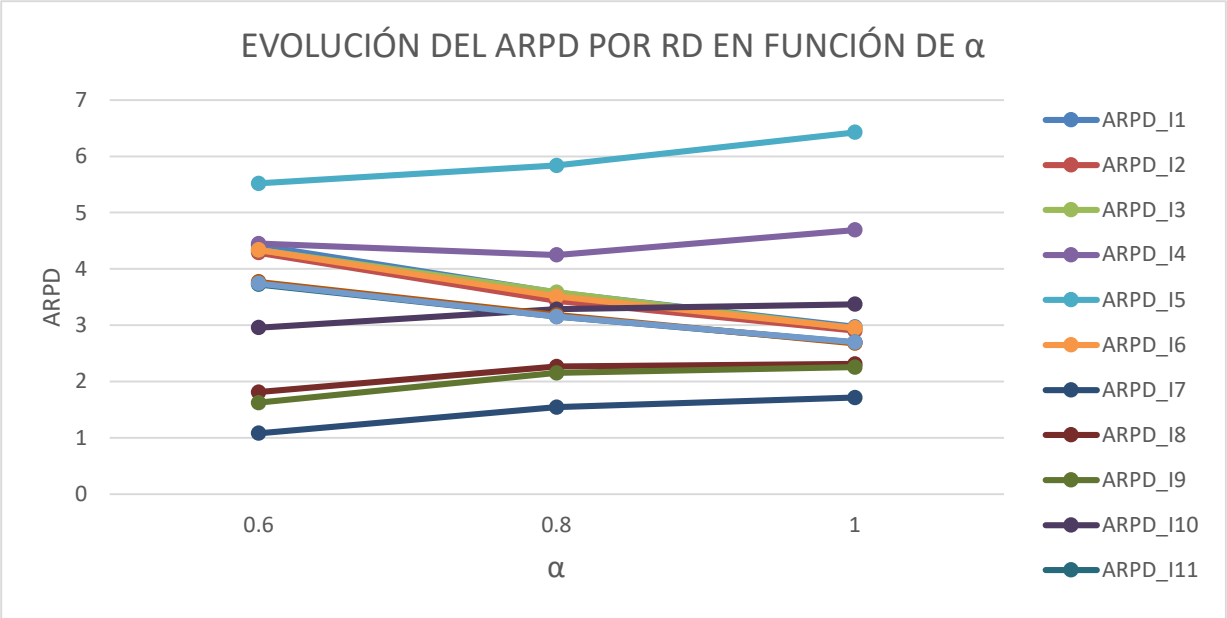

Ilustración 16. Gráfico sobre la evolución del ARPD en función del parámetro α

5.2.2 Heurísticas

Como se ha mencionado anteriormente, la complejidad y el diseño de los métodos aplicados a un problema determina la calidad de las soluciones proporcionadas. En el caso de los procedimientos heurísticos, se espera que proporcionen mejores resultados que las reglas de despacho propuestas.

Los resultados obtenidos con respecto al mínimo proporcionado por los procedimientos heurísticos y los de las búsquedas locales son:

Tabla 5. ARPD de las heurísticas constructivas y búsquedas locales

CH	ARPD	ACPU	CH+BL	ARPD	ACPU
CH_V1	0.0157	0.0261	**CH_V1_BI**	0.0041	10.8506
CH_V2	0.0155	0.0268	**CH_V2_BI**	0.0042	10.8556
CH_V3	0.01532	0.043	**CH_V3_BI**	0.0039	10.8977
CH_V4	0.01526	0.0415	**CH_B4_BI**	0.004	10.8845

Repasando las distintas versiones, en la primera de ellas, se calcula el indicador de carga teniendo en cuenta solo los tiempos de ensamblado repercutido al producto de las máquinas de la primera etapa y la segunda como en Talens et al. (2020). En la segunda versión, se modifica el denominador del indicador haciendo que los tiempos de ensamblado sólo afecten a la segunda etapa. Entre la primera y la segunda versión, se observa una ligera mejoría, al considerar únicamente las máquinas de la segunda etapa en el cálculo de CT. Esto puede indicar que, en el cálculo del indicador, la primera etapa no tiene tanta influencia como la segunda.

En la tercera y cuarta versión, se introducen los tiempos de preparación (*setups),* no tenidos en cuenta hasta ahora por ningún método, en el cálculo del indicador CT. De nuevo, en este caso es resulta más exitoso calcular el indicador de carga sólo en base a las máquinas de la segunda etapa.

En cuanto a las distintas versiones de las heurísticas, se han obtenido los resultados esperados, ya que los valores de la función objetivo han ido mejorando a medida que se han ido introduciendo las restricciones del modelo en el algoritmo, obteniendo los mejores resultados en las versiones 3 y 4.

Por otro lado, las soluciones obtenidas por cada una de las versiones de la heurística han sido alimentadas a un método de búsqueda local.

Como es de esperar, la aplicación de búsquedas locales proporciona una mejora de los resultados en todas las versiones de la heurística, aunque, como se observa en la Tabla 5, la mejor combinación es la que resulta de aplicar la búsqueda local *Best Improvement* a la solución obtenida por la versión 3.

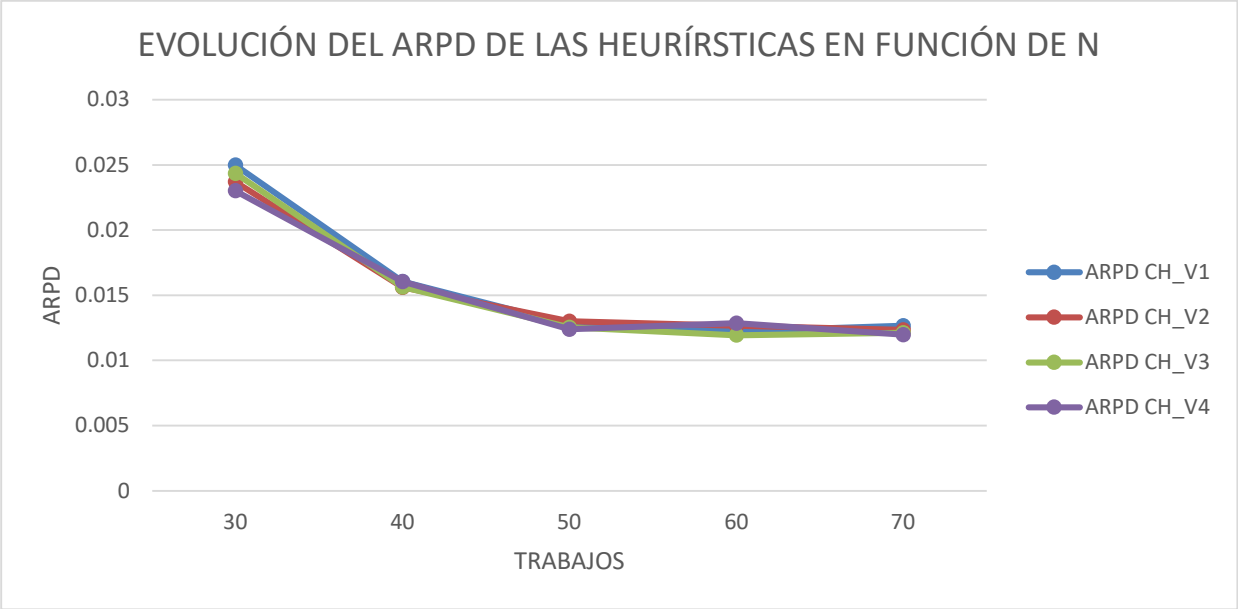

Ilustración 17. Gráfico sobre la evolución del ARPD de las heurísticas en función del número de trabajos

En el caso de los procedimientos heurísticos se observa una tendencia bastante uniforme en la evolución del ARPD de todos los procedimientos, lo que indica que el comportamiento de los algoritmos es independiente del tamaño del problema y que mejoran sus resultados a medida que aumenta el número de trabajos.

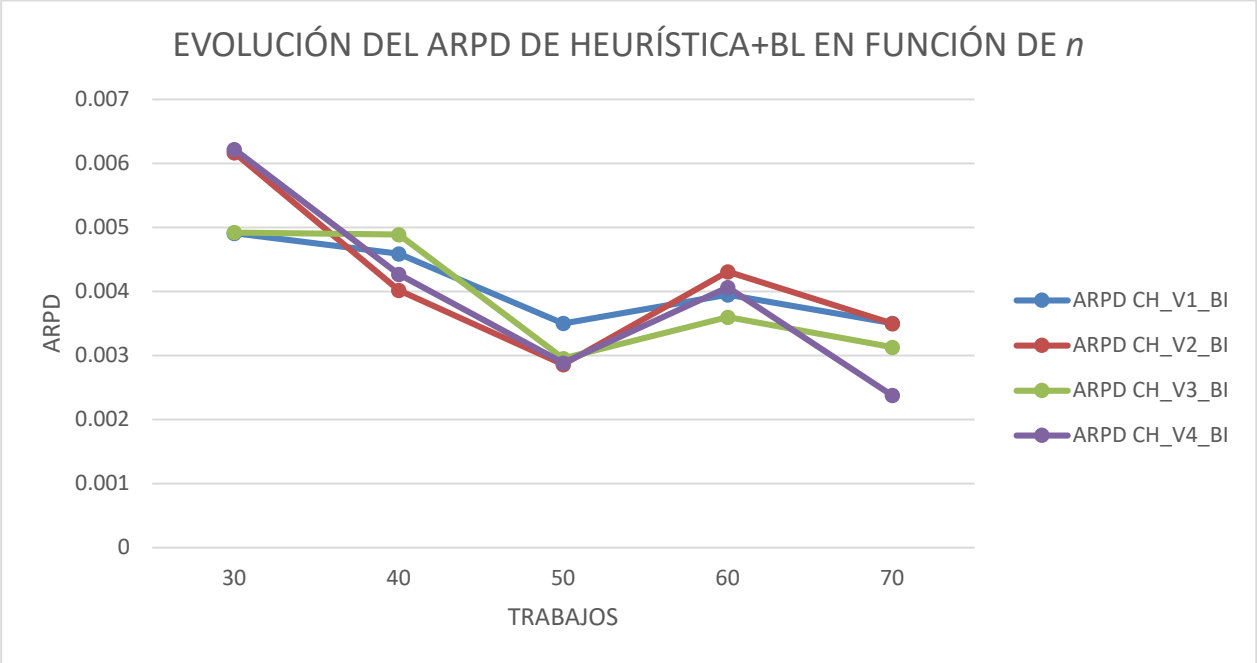

Ilustración 18. Gráfico sobre la evolución del ARPD de las heurísticas junto con búsqueda local en función del número de trabajos

En el caso de los procedimientos heurísticos junto con la búsqueda local, generan resultados más dispares entre ellos, aunque tienden a la mejora de resultados según aumenta el número de trabajos de manera general. Sin embargo, aunque el ARPD promedio de la versión 3 más la búsqueda local es mejor en general que el resto, según se muestra en la Tabla 5, para las instancias de 40 produce mejores resultados la versión 2 y para las de 70 trabajos la versión 4.

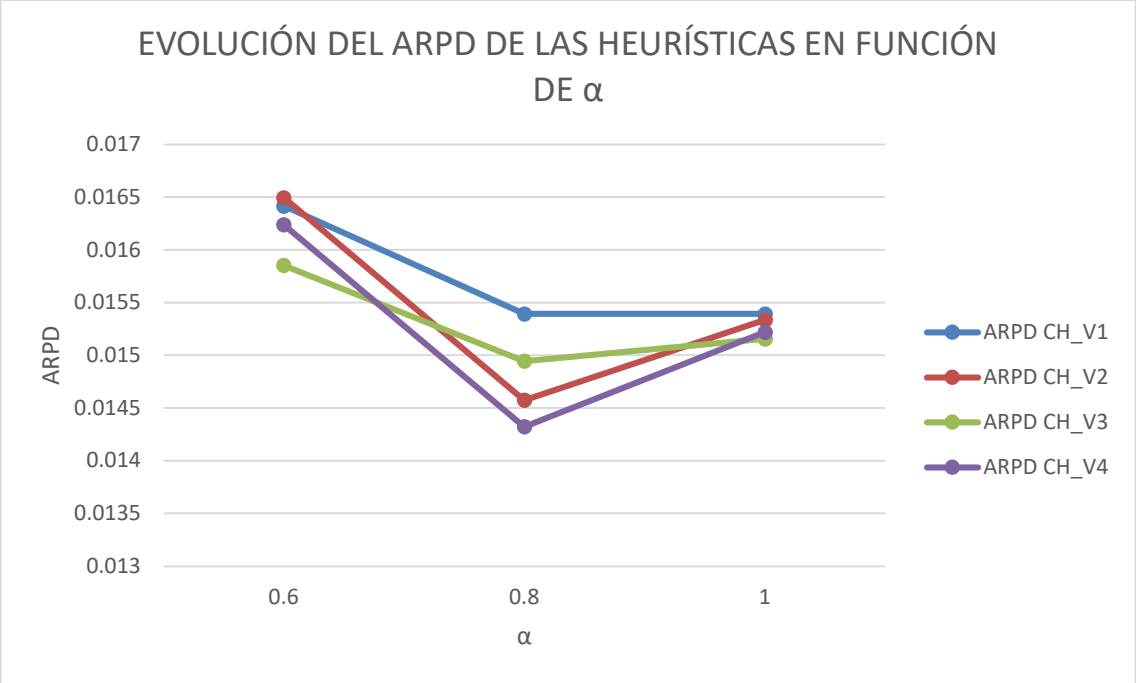

Ilustración 19. Gráfico sobre la evolución del ARPD de las heurísticas en función del parámetro α

En cuanto a los resultados segmentados en función del valor de α se obtiene que los mejores resultados obtenidos por todas las versiones de la heurística se producen para α=0,8.

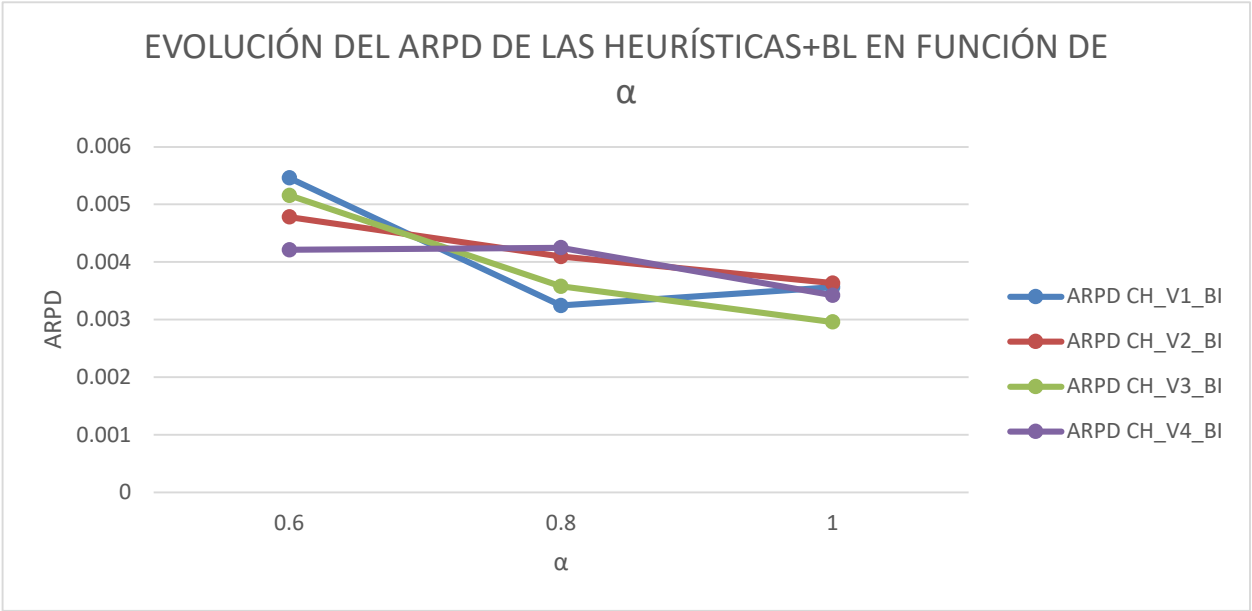

Ilustración 20. Gráfico sobre la evolución del ARPD de las heurísticas junto con búsqueda local en función del parámetro α

Es preciso comentar que mientras que las versiones 1 y 3 con búsqueda local proporcionan mejores resultados para α=0’8, en el caso de las versiones 2 y 4 se generan mejores resultados con respecto al resto de versiones sin aplicar búsqueda local.

En el caso de la evolución del ARPD de las heurísticas con búsqueda local en función del parámetro α, se observa que proporcionan de forma general mejores resultados para instancias con α=1. Esto implica que, aunque la solución de partida para la búsqueda de vecinos con α=1 es peor que para α=0’8, la solución se encuentra ubicada en una mejor zona del espacio de soluciones, por lo que la búsqueda local resulta más exitosa para α=1.

Por último, en la Tabla 5, también se presentan los resultados obtenidos del tiempo promedio de cómputo (ACPU). Como era de esperar, el tiempo de cómputo aumenta en los procedimientos que incluyen búsqueda local debido a la cantidad de inserciones realizadas y, por tanto, secuencias evaluadas. Sin embargo, es importante considerar adecuadamente el contexto en el que se encuentre el modelo para saber si es interesante invertir más tiempo en encontrar una solución mejor.

6 CONCLUSIONES

La educación es el vestido de gala para asistir a la fiesta de la vida

- Miguel Rojas Sánchez-

En este Trabajo Fin de Grado se ha tratado un modelo de programación de la producción basado en el ensamblado de componentes, siendo uno de ellos suministrado por un proveedor externo, con el objetivo de minimizar el *makespan* o tiempo necesario para completar todos los trabajos. En la literatura se han estudiado modelos similares, aunque con distintas restricciones.

Se han presentado dos metodologías para la resolución del problema propuesto. En primer lugar, se han adaptado las reglas de despacho propuestas en Komaki & Kayvanfar (2015) a las restricciones presentadas. Las reglas de despacho consisten en el cálculo de un indicador para cada trabajo y la posterior ordenación de los mismos siguiendo un criterio ascendente o descendente de secuenciación respecto al indicador. Dichas reglas de despacho se han clasificado en tres grupos diferentes en función de los datos que tuviera en cuenta del problema, resultando las del segundo grupo las que mejores resultados han aportado.

En segundo lugar, se ha adaptado la *Fast Constructive Heuristic,* propuesta en Talens et al. (2021) para el problema sin restricciones, al problema con fechas de llegada del componente suministrado externamente y con tiempos de *setup* dependientes de la secuencia para la segunda etapa, que se ha tratado a lo largo de este trabajo. Además de la versión propuesta en Talens et al. (2021), se han analizado tres versiones más afectando al cálculo del indicador de carga. Como ha podido verse en el análisis de las distintas versiones, la versión cuatro ha resultado mejor en promedio que el resto.

En tercer lugar, se han mejorado las soluciones aportadas por las distintas versiones de la heurística mediante la aplicación de un método de búsqueda local de vecinos basado en inserción. La exploración de vecinos ha consistido en la evaluación de un número concreto de secuencias vecinas calculado en base al total de trabajos de cada instancia con el objetivo de no dilatar demasiado el tiempo de cómputo. La mejor combinación de heurística con la búsqueda local ha resultado ser la tercera.

Como futuras líneas de trabajo, sería interesante adaptar otras heurísticas existentes en la literatura para problemas relacionados y llevar a cabo una evaluación computacional en la que se comparen con los métodos propuestos en este trabajo.

7 REFERENCIAS

Al-Anzi, F., & Allahverdi, A. (2006). A Hybrid Tabu Search Heuristic for the Two-Stage Assembly. *International Journal of Production Research*.

Framinan J. M., Perez-Gonzalez, P., & Fernandez-Viagas, V. ((2019)). Deterministic assembly scheduling problems: A review and classification of concurrent-type scheduling models and solution procedures. *European Journal of Operational Research*.

Graham, R., Lawler , E., Lenstra, J., & Kan , A. (1979). Optimization and approximation in deterministic sequencing and scheduling: A survey. *In Annals of Discrete Mathematics*, 287-326.

Komaki, G., & Kayvanfar, V. (2015). Grey Wolf Optimizer algorithm for the two-stage assembly flow shop scheduling problem with release time. *Journal of Computational Science*.

Sung, C., & Juhn, J. (2009). Makespan minimization for a 2-stage assembly scheduling problem subject to component available time constraint. *Int. J. Production Economics*.

Talens, C., Fernandez-Viagas, V., Perez-Gonzalez, P., & Framinan, J. (2020). New efficient constructive heuristics for the two-stage multi-machine assembly scheduling problem. *Computers & Industrial Engineering*.

Talens, C., Fernandez-Viagas, V., Perez-Gonzalez, P., & Framinan, J. (2021). New hard benchmark for the 2-stage multi-machine assembly scheduling. *Computers & Industrial Engineering*.

8 ANEXOS

Para concluir, en este último apartado se incluyen los diferentes archivos con código en Python que se han programado para para el estudio del problema de ensamblado con dos etapas para el *makespan*. Además, se incluye también el archivo para la generación de instancias que han sido resueltas y estudiadas.

8.1 Modelo

```
from scheptk.scheptk import *
from scheptk.util import *
import numpy as np
import matplotlib.pyplot as plt



#second_stage_seq returns the jobs that are processed in each assembly machine and their completion times ordered in ascending order
def second_stage_seq(self, completion_time2):

    job_times=[[]for j in range(self.m2)]
    job_order=[[] for j in range(self.m2)]
    for j in range(self.m2):
        job_times[j]=sorted_value_asc(completion_time2[j])
        job_order[j]=sorted_index_asc(completion_time2[j])
        while job_times[j][0]==0:
            job_times[j].pop(0)
            job_order[j].pop(0)

    return job_times, job_order

#this function calculate C1 which represents the time when each job can be assembly
def moment_of_availability(self,completion_time1):
    C1=[]
    aux=[]
    for i in range(self.jobs):
```

```
        for j in range(len(completion_time1)):
            aux.append(completion_time1[j][i])
        C1.append(max(aux))
        aux.clear()

    return C1

class assembly(Model):


    #construct of the class
    def __init__(self,filename):

        self.jobs=read_tag(filename,'JOBS') #total jobs

        self.m1=read_tag(filename,'DEDICATED_MACHINES') #total dedicated machines

        self.m2=read_tag(filename,'ASSEMBLY_MACHINES') #total parallel machines

        self.pt=read_tag(filename,'PROCESSING_TIMES') #component-processed processing times

        self.ass_time=read_tag(filename,'ASSEMBLY_TIMES') #time to assembly the components of a
job


        self.sjk=read_tag(filename, 'SETUP_TIMES') #parallel machines'setup times


        self.jobsname=read_tag(filename,'JOB_NAMES') #jobs name


    #This module's aim is to calcule jobs' completion times
    def ct(self,sequence,arrival_times):

        completion_time1=[[0 for j in range (self.jobs)] for i in range(self.m1)]
```

```
        # first job in all machines at first stage
        for j in range(self.m1):
            completion_time1[j][sequence[0]]=self.pt[j][sequence[0]]

        #rest of jobs at first stage
        for i in range(1,len(sequence)):
            for j in range(self.m1):
                completion_time1[j][sequence[i]]=completion_time1[j][sequence[i-
1]]+self.pt[j][sequence[i]]

        #add an extra row with release dates
        completion_time1.append([0 for j in range(len(sequence))])
        for i in range(len(sequence)):
            completion_time1[self.m1][sequence[i]]=arrival_times[sequence[i]]

        C1=moment_of_availability(self,completion_time1)

        #assign the first job at second stage
        C2=[[0 for j in range (self.jobs)] for i in range (self.m2)]
        trabajos=[j for j in range(self.jobs)]
        indice=[]

        aux=[]

        for k in range(self.m2):

            mc=np.argmin([max(C2[i]) for i in range(self.m2)])

            for j in trabajos:

                if max(C2[mc])<C1[j]-self.sjk[0][j]:
                    aux.append(C1[j]+self.ass_time[j])

                else:
```

```
            aux.append(max(C2[mc])+self.sjk[0][j]+self.ass_time[j])

    C2[mc][trabajos[np.argmin(aux)]]=min(aux)

    indice.append(trabajos[np.argmin(aux)])
    trabajos.remove(trabajos[np.argmin(aux)])

    aux=[]

#the rest of jobs at second stage
for k in range(0,self.jobs-self.m2):

    mc=np.argmin([max(C2[i]) for i in range(self.m2)])

    for j in trabajos:


        if max(C2[mc])<C1[j]-self.sjk[np.argmax(C2[mc])+1][j]:
            aux.append(C1[j]+self.ass_time[j])

        else:

            aux.append(max(C2[mc])+self.sjk[np.argmax(C2[mc])+1][j]+self.ass_time[j])


    C2[mc][trabajos[np.argmin(aux)]]=min(aux)

    indice.append(trabajos[np.argmin(aux)])

    trabajos.remove(trabajos[np.argmin(aux)])
    aux=[]


return completion_time1,C2,sequence
```

```
  def print_assembly_schedule(self,sequence,arrival_times,filename=None):

    completion_time1,completion_time2,sequence=self.ct(sequence,arrival_times)

    gantt = Schedule(self.m1,self.m2,arrival_times)

    #printing first job at first stage
    for j in range(self.m1):
      gantt.add_task(Task(sequence[0],j,0,completion_time1[j][sequence[0]]))
    #printing rest of jobs at first stage
    for i in range (1,self.jobs):
      for j in range(self.m1):
        gantt.add_task(Task(sequence[i],j,completion_time1[j][sequence[i-
1]],completion_time1[j][sequence[i]]))

    job_times=[[]for j in range(self.m2)]
    job_order=[[] for j in range(self.m2)]

    job_times, job_order=second_stage_seq(self,completion_time2)

    for j in range (self.m2):
      for i in range(len(job_order[j])):
        if i==0:
          gantt.add_NAP(NAP(self.m1+1+j,job_times[j][i]-self.ass_time[job_order[j][i]]-
self.sjk[i][job_order[j][i]],job_times[j][i]-self.ass_time[job_order[j][i]],'ST'))
        else:
          gantt.add_NAP(NAP(self.m1+1+j,job_times[j][i]-self.ass_time[job_order[j][i]]-
self.sjk[job_order[j][i-1]+1][job_order[j][i]],job_times[j][i]-self.ass_time[job_order[j][i]],'ST'))
        gantt.add_task(Task(job_order[j][i],self.m1+j+1,job_times[j][i]-
self.ass_time[job_order[j][i]],job_times[j][i]))
    gantt.print()

  def funcion_objetivo(self,completion_time2):
    fo=round((max(max(completion_time2[i]) for i in range(self.m2))),1)

    return fo
```

8.2 Archivo de Heurísticas Constructivas

```
from scheptk.scheptk import *
from scheptk.util import *
import numpy as np
import matplotlib.pyplot as plt
from untitled0 import *
import copy
import time



def CH_V1(instancia,arrival_times):
    C1_i=[0 for i in range(instancia.m1)]
    C2_i=[[0 for i in range(instancia.jobs)] for i in range(instancia.m2)]
    pi=[]
    U=[j for j in range (instancia.jobs)]
    psi_j=[]
    a=5
    indice=[]

    inicio=time.time()
    for j in range(instancia.jobs):
        for wj in (U):
            C1_wj=max(max(C1_i[i]+instancia.pt[i][wj] for i in range(instancia.m1)),arrival_times[wj])
            if len(pi)==0:
                IT_j=max(C1_wj-min([max(C2_i[i]) for i in range(instancia.m2)])-instancia.sjk[0][wj],0)
            else:
                IT_j=max(C1_wj-min([max(C2_i[i]) for i in range(instancia.m2)])-
instancia.sjk[np.argmax(C2_i[np.argmin([max(C2_i[i]) for i in range(instancia.m2)])])+1][wj],0)

            CT_j=(instancia.ass_time[wj]/instancia.m1*instancia.m2)
            psi_j.append(a*IT_j+CT_j)
            indice.append(wj)



        C1_i=[C1_i[i]+instancia.pt[i][indice[np.argmin(psi_j)]] for i in range(instancia.m1)]
```

```
        if len(pi)==0:
            b=instancia.sjk[0][indice[np.argmin(psi_j)]]
        else:
            b=instancia.sjk[np.argmax(C2_i[np.argmin([max(C2_i[i]) for i in
range(instancia.m2)])])+1][indice[np.argmin(psi_j)]]


        if min([max(C2_i[i]) for i in
range(instancia.m2)])<max(max(C1_i),arrival_times[indice[np.argmin(psi_j)]])-b:
            C2_i[np.argmin([max(C2_i[i]) for i in
range(instancia.m2)])][indice[np.argmin(psi_j)]]=max(arrival_times[indice[np.argmin(psi_j)]],max(
C1_i))+instancia.ass_time[indice[np.argmin(psi_j)]]
        else:
            C2_i[np.argmin([max(C2_i[i]) for i in
range(instancia.m2)])][indice[np.argmin(psi_j)]]=C2_i[np.argmin([max(C2_i[i]) for i in
range(instancia.m2)])][np.argmax(C2_i[np.argmin([max(C2_i[i]) for i in
range(instancia.m2)])])]+b+instancia.ass_time[indice[np.argmin(psi_j)]]
        #print(indice)
        pi.append(indice[np.argmin(psi_j)])
        #print(pi)
        U.remove(indice[np.argmin(psi_j)])
        psi_j.clear()
        indice.clear()


    t_computo=time.time()-inicio
    return pi, t_computo




def CH_V3(instancia,arrival_times):
    C1_i=[0 for i in range(instancia.m1)]
    C2_i=[[0 for i in range(instancia.jobs)] for i in range(instancia.m2)]


    pi=[]
    U=[j for j in range (instancia.jobs)]
    psi_j=[]
    a=5
```

```
indice=[]



inicio=time.time()
for j in range(instancia.jobs):
    for wj in (U):
        C1_wj=max(arrival_times[wj], max(C1_i[i]+instancia.pt[i][wj] for i in range(instancia.m1)))


        if len(pi)==0:
            IT_j=max(C1_wj-instancia.sjk[0][wj]-min([max(C2_i[i]) for i in range(instancia.m2)]),0)
            CT_j=(instancia.ass_time[wj]+instancia.sjk[0][wj])/instancia.m1*instancia.m2
        else:
            IT_j=max(C1_wj-instancia.sjk[np.argmax(C2_i[np.argmin([max(C2_i[i]) for i in
range(instancia.m2)])])+1][wj]-min([max(C2_i[i]) for i in range(instancia.m2)]),0)
            CT_j=(instancia.ass_time[wj]+instancia.sjk[np.argmax(C2_i[np.argmin([max(C2_i[i]) for i
in range(instancia.m2)])])+1][wj])/instancia.m1*instancia.m2
        psi_j.append(a*IT_j+CT_j)
        indice.append(wj)




    C1_i=[C1_i[i]+instancia.pt[i][indice[np.argmin(psi_j)]] for i in range(instancia.m1)]
    if len(pi)==0:
        b=instancia.sjk[0][indice[np.argmin(psi_j)]]
    else:
        b=instancia.sjk[np.argmax(C2_i[np.argmin([max(C2_i[i]) for i in
range(instancia.m2)])])+1][indice[np.argmin(psi_j)]]
    if min([max(C2_i[i]) for i in
range(instancia.m2)])<max(max(C1_i),arrival_times[indice[np.argmin(psi_j)]])-b:
        C2_i[np.argmin([max(C2_i[i]) for i in
range(instancia.m2)])][indice[np.argmin(psi_j)]]=max(arrival_times[indice[np.argmin(psi_j)]],max(
C1_i))+instancia.ass_time[indice[np.argmin(psi_j)]]
    else:
        C2_i[np.argmin([max(C2_i[i]) for i in
range(instancia.m2)])][indice[np.argmin(psi_j)]]=C2_i[np.argmin([max(C2_i[i]) for i in
range(instancia.m2)])][np.argmax(C2_i[np.argmin([max(C2_i[i]) for i in
range(instancia.m2)])])]+b+instancia.ass_time[indice[np.argmin(psi_j)]]
```

```
        pi.append(indice[np.argmin(psi_j)])

        U.remove(indice[np.argmin(psi_j)])
        psi_j.clear()
        indice.clear()



        psi_j.clear()
        indice.clear()
    t_computo=time.time()-inicio
    return pi, t_computo

def CH_V2(instancia,arrival_times):
    C1_i=[0 for i in range(instancia.m1)]
    C2_i=[[0 for i in range(instancia.jobs)] for i in range(instancia.m2)]

    pi=[]
    U=[j for j in range (instancia.jobs)]
    psi_j=[]
    a=5
    indice=[]
    inicio=time.time()
    for j in range(instancia.jobs):
        for wj in (U):
            C1_wj=max(arrival_times[wj], max(C1_i[i]+instancia.pt[i][wj] for i in range(instancia.m1)))

            if len(pi)==0:
                IT_j=max(C1_wj-min([max(C2_i[i]) for i in range(instancia.m2)])-instancia.sjk[0][wj],0)

            else:
                IT_j=max(C1_wj-min([max(C2_i[i]) for i in range(instancia.m2)])-
instancia.sjk[np.argmax(C2_i[np.argmin([max(C2_i[i]) for i in range(instancia.m2)])])+1][wj],0)

            CT_j=instancia.ass_time[wj]/instancia.m2
            psi_j.append(a*IT_j+CT_j)
            indice.append(wj)
```

```
        C1_i=[C1_i[i]+instancia.pt[i][indice[np.argmin(psi_j)]] for i in range(instancia.m1)]
        if len(pi)==0:
            b=instancia.sjk[0][indice[np.argmin(psi_j)]]
        else:
            b=instancia.sjk[np.argmax(C2_i[np.argmin([max(C2_i[i]) for i in
range(instancia.m2)])])+1][indice[np.argmin(psi_j)]]
        if min([max(C2_i[i]) for i in
range(instancia.m2)])<max(max(C1_i),arrival_times[indice[np.argmin(psi_j)]])-b:
            C2_i[np.argmin([max(C2_i[i]) for i in
range(instancia.m2)])][indice[np.argmin(psi_j)]]=max(arrival_times[indice[np.argmin(psi_j)]],max(
C1_i))+instancia.ass_time[indice[np.argmin(psi_j)]]
        else:
            C2_i[np.argmin([max(C2_i[i]) for i in
range(instancia.m2)])][indice[np.argmin(psi_j)]]=C2_i[np.argmin([max(C2_i[i]) for i in
range(instancia.m2)])][np.argmax(C2_i[np.argmin([max(C2_i[i]) for i in
range(instancia.m2)])])]+b+instancia.ass_time[indice[np.argmin(psi_j)]]


        pi.append(indice[np.argmin(psi_j)])


        U.remove(indice[np.argmin(psi_j)])
        psi_j.clear()
        indice.clear()



        psi_j.clear()
        indice.clear()
    t_computo=time.time()-inicio
    return pi, t_computo
def CH_V4(instancia,arrival_times):
    C1_i=[0 for i in range(instancia.m1)]
    C2_i=[[0 for i in range(instancia.jobs)] for i in range(instancia.m2)]


    pi=[]
    U=[j for j in range (instancia.jobs)]
    psi_j=[]
    a=5
    indice=[]
    inicio=time.time()
```

```
    for j in range(instancia.jobs):
        for wj in (U):
            C1_wj=max(arrival_times[wj], max(C1_i[i]+instancia.pt[i][wj] for i in range(instancia.m1)))


            if len(pi)==0:
                IT_j=max(C1_wj-min([max(C2_i[i]) for i in range(instancia.m2)])-instancia.sjk[0][wj],0)
                CT_j=(instancia.ass_time[wj]+instancia.sjk[0][wj])/instancia.m2
            else:
                IT_j=max(C1_wj-min([max(C2_i[i]) for i in range(instancia.m2)])-
instancia.sjk[np.argmax(C2_i[np.argmin([max(C2_i[i]) for i in range(instancia.m2)])])+1][wj],0)
                CT_j=(instancia.ass_time[wj]+instancia.sjk[np.argmax(C2_i[np.argmin([max(C2_i[i]) for i
in range(instancia.m2)])])+1][wj])/instancia.m2
            psi_j.append(a*IT_j+CT_j)
            indice.append(wj)
        C1_i=[C1_i[i]+instancia.pt[i][indice[np.argmin(psi_j)]] for i in range(instancia.m1)]
        if len(pi)==0:
            b=instancia.sjk[0][indice[np.argmin(psi_j)]]
        else:
            b=instancia.sjk[pi[-1]+1][indice[np.argmin(psi_j)]]
        if min([max(C2_i[i]) for i in
range(instancia.m2)])<max(max(C1_i),arrival_times[indice[np.argmin(psi_j)]])-b:
            C2_i[np.argmin([max(C2_i[i]) for i in
range(instancia.m2)])][indice[np.argmin(psi_j)]]=max(arrival_times[indice[np.argmin(psi_j)]],max(
C1_i))+instancia.ass_time[indice[np.argmin(psi_j)]]
        else:
            C2_i[np.argmin([max(C2_i[i]) for i in
range(instancia.m2)])][indice[np.argmin(psi_j)]]=C2_i[np.argmin([max(C2_i[i]) for i in
range(instancia.m2)])][np.argmax(C2_i[np.argmin([max(C2_i[i]) for i in
range(instancia.m2)])])]+b+instancia.ass_time[indice[np.argmin(psi_j)]]


        pi.append(indice[np.argmin(psi_j)])


        U.remove(indice[np.argmin(psi_j)])
        psi_j.clear()
        indice.clear()



        psi_j.clear()
```

```
        indice.clear()
    t_computo=time.time()-inicio
    return pi, t_computo
```

8.3 *Best Improvement* con criterio de parada

```
import copy
import time
from untitled0 import *

# BEST IMPROVEMENT
def BI_I(instancia, seq,fo,arrival_times):
    # best_seq=copy.deepcopy(seq)
    best_obj=copy.deepcopy(fo)
    parada=(instancia.jobs*instancia.jobs)/5
    inicio=time.time()

    iteraciones=0
    for j in range (instancia.jobs):
        for k in range(instancia.jobs):
            if iteraciones<parada:
                if k!=j and k!=(j-1):
                    vecino=copy.deepcopy(seq)
                    vecino.pop(j)
                    vecino.insert(k,seq[j])
                    c1,c2,seq=instancia.ct(vecino,arrival_times)
                    obj=instancia.funcion_objetivo(c2)
                    iteraciones=iteraciones+1

                    if obj<best_obj:
                        # best_seq=copy.deepcopy(vecino)
                        best_obj=obj
```

```
                if iteraciones>parada:
                    k=instancia.jobs
                    j=instancia.jobs

    tiempo_computo=time.time()-inicio

    return best_obj, tiempo_computo
```

8.4 Reglas de despacho

```
from scheptk.scheptk import *
from scheptk.util import *
import numpy as np
import pdb

import sys
#================================REGLAS DE
DESPACHO=========================================

#EN ORDEN CRECIENTE - reglas basadas en los arrival time y/o processing times




def I1(instancia,arrival_times):

    seq=sorted_index_asc(arrival_times)

    return seq



def I2(instancia,arrival_times): #de menor a mayor teniendo en cuenta la suma del tiempo de llegada
y la del tiempo de proceso de las DPM

    I2=[max(arrival_times[j],max(instancia.pt[i][j] for i in range(instancia.m1))) for j in
range(instancia.jobs)]
```

```
    seq=sorted_index_asc(I2)
    return seq



def I3(instancia,arrival_times):

    I3=[max(arrival_times[j],sum(instancia.pt[i][j] for i in range(instancia.m1))/instancia.m1) for j in
range(instancia.jobs)]

    seq=sorted_index_asc(I3)
    return seq

def I4(instancia,arrival_times):
    I4=[min(arrival_times[j],sum(instancia.pt[i][j] for i in range(instancia.m1))) for j in
range(instancia.jobs)]

    seq=sorted_index_asc(I4)
    return seq

def I5(instancia,arrival_times):
    I5=[min(arrival_times[j], max(instancia.pt[i][j] for i in range(instancia.m1))) for j in
range(instancia.jobs)]

    seq=sorted_index_asc(I5)
    return seq
def I6(instancia,arrival_times):
    aux=[sum(instancia.pt[i][j] for j in range(instancia.jobs)) for i in range(instancia.m1)]
    Ti=np.argmax(aux)
    I6=[max(arrival_times[j],instancia.pt[Ti][j]) for j in range(instancia.jobs)]
    seq=sorted_index_asc(I6)
    return seq
```

```
# EN ORDEN DESCENDENTE - se tienen en cuenta tiempos de proceso y/o assembly times
def I7(instancia,arrival_times):
    seq=sorted_index_desc(instancia.ass_time)


    return seq
def I8(instancia,arrival_times):
    I8=[max(instancia.pt[i][j] for i in range(instancia.m1))+instancia.ass_time[j] for j in
range(instancia.jobs)]
    seq=sorted_index_desc(I8)


    return seq
def I9(instancia,arrival_times):
    I9=[instancia.ass_time[j]+sum(instancia.pt[i][j] for i in range(instancia.m1))/(instancia.m1) for j in
range(instancia.jobs)]
    seq=sorted_index_desc(I9)


    return seq
def I10(instancia,arrival_times):
    aux=[sum(instancia.pt[i][j] for j in range(instancia.jobs)) for i in range(instancia.m1)]
    Ti=np.argmax(aux)



    I10=[instancia.pt[Ti][j]+instancia.ass_time[j] for j in range(instancia.jobs)]


    seq=sorted_index_desc(I10)
    return seq


#ORDEN CRECIENTE - se tienen en cuenta todas las propiedades: arrival times, processing times y
assembly times



def I11(instancia,arrival_times):


    I11=[max(arrival_times[k],max(instancia.pt[i][k] for i in
range(instancia.m1)))+instancia.ass_time[k] for k in range (instancia.jobs)]


    seq=sorted_index_asc(I11)
```

```
    return seq
def I12(instancia,arrival_times):
    I12=[max(arrival_times[j],sum(instancia.pt[i][j] for i in
range(instancia.m1))/(instancia.m1))+instancia.ass_time[j] for j in range(instancia.jobs)]

    seq=sorted_index_asc(I12)
    return seq



def I13(instancia,arrival_times):
    aux=[sum(instancia.pt[i][j] for j in range(instancia.jobs)) for i in range(instancia.m1)]
    Ti=np.argmax(aux)
    I13=[max(arrival_times[j],instancia.pt[Ti][j])+instancia.ass_time[j] for j in range(instancia.jobs)]
    seq=sorted_index_asc(I13)
    return seq
```

8.5 Análisis de instancias

```
from untitled0 import *
import random
import numpy as np
from dispatching_rules import *
import openpyxl
from CH_heuristic import *
from Best_Improvement import *



trabajos=[30,40,50,60,70]
DM=[2,4,6,8]
AM=[1,2,3]
replicas=5
alpha=[0.6,0.8,1]

cont2=2
```

```
seq=[]

workbook = openpyxl.load_workbook('RESULTADOS.xlsx')
sheet = workbook.get_sheet_by_name('Hoja1')

ruta_padre = os.getcwd()

for t in range(len(trabajos)):

    nombre_carpeta = str(trabajos[t])+'_trabajos'
    ruta_carpeta = os.path.join(ruta_padre, nombre_carpeta)
    os.chdir(ruta_carpeta)

    for d in range(len(DM)):
        for a in range(len(AM)):
            for k in range(replicas):
                nombre_instancia=str(trabajos[t])+'_'+str(DM[d])+'_'+str(AM[a])+' '+str(k)+' .txt'

                print("\n", nombre_instancia)


                for l in range(len(alpha)):

                    instancia=assembly(nombre_instancia)

                    arrival_times=read_tag(nombre_instancia, 'R'+str(alpha[l]))
                    sheet.cell(row=cont2, column=1, value=str(nombre_instancia))
                    sheet.cell(row=cont2,column=2,value=str(trabajos[t]))
                    sheet.cell(row=cont2,column=3,value=str(DM[d]))
                    sheet.cell(row=cont2,column=4,value=str(AM[a]))
                    sheet.cell(row=cont2, column=5, value=str(alpha[l]))
                    for i in range(1,14):

                        dispatching_rule='I'+str(i)+'(instancia,arrival_times)'
                        seq=eval(dispatching_rule)
```

```
    completion_time1,completion_time2,sequence=instancia.ct(seq, arrival_times)
    funcion_objetivo=instancia.funcion_objetivo(completion_time2)
    sheet.cell(row=cont2, column=5+i, value=funcion_objetivo)

    seq.clear()
    completion_time1.clear()
    completion_time2.clear()



for i in range(0,4):
    heuristica='CH_V'+str(i+1)+'(instancia,arrival_times)'

    C_heuristic2,T_computo=eval(heuristica)
    c1,c2,seq=instancia.ct(C_heuristic2,arrival_times)
    fo=instancia.funcion_objetivo(c2)
    sheet.cell(row=cont2, column=19+2*i, value=fo)
    sheet.cell(row=cont2,column=20+2*i,value=T_computo)


    fo_vecino,t_computo=BI_I(instancia,seq,fo,arrival_times)
    sheet.cell(row=cont2, column=27+2*i, value=fo_vecino)
    sheet.cell(row=cont2, column=28+2*i, value=t_computo)

    c1.clear()
    c2.clear()
    seq.clear()



cont2=cont2+1

arrival_times.clear()
workbook.save('RESULTADOS.xlsx')
```

```
    os.chdir(ruta_padre)

workbook.save('RESULTADOS.xlsx')
```

8.6 Generación de Instancias

```
from scheptk.util import *
import numpy as np
import random
import os


trabajos=[30,40,50,60,70]
DM=[2,4,6,8]
AM=[1,2,3]
replicas=5

ruta_padre = os.getcwd() #SE OBTIENE EL DIRECTORIO EN EL QUE SE ENCUENTRA EL ARCHIVO EN EJECUCION

for t in range(len(trabajos)):

    nombre_carpeta = str(trabajos[t])+'_trabajos'
    ruta_carpeta = os.path.join(ruta_padre, nombre_carpeta) #AL DIRECTORIO ACTUAL, SE LE AÑADE EL NOMBRE DE LA CARPETA Y TENEMOS EL DIRECTORIO DE LA CARPETA
    if not os.path.exists(ruta_carpeta):
        os.mkdir(ruta_carpeta) #SE CREA LA CARPETA

    os.chdir(ruta_carpeta) #SE CAMBIA AL DIRECTORIO DENTRO DE LA CARPETA


    PROCESSING_TIMES=[]
    SETUP_TIMES=[]
    ASSEMBLY_TIMES=[]
```

```
R=[]
JOB_NAMES=[]
MACHINE_NAMES=[]
for d in range(len(DM)):
    for a in range(len(AM)):
        for k in range(replicas):

            nombre_archivo=str(trabajos[t])+'_'+str(DM[d])+'_'+str(AM[a])+' '+str(k)+' .txt'

            write_tag('JOBS',trabajos[t],nombre_archivo)
            write_tag('DEDICATED_MACHINES',DM[d],nombre_archivo)
            write_tag('ASSEMBLY_MACHINES',AM[a],nombre_archivo)

            PROCESSING_TIMES=[[round(random.uniform(1,100),1) for j in range(trabajos[t])] for i
in range(DM[d])]
            write_tag('PROCESSING_TIMES',PROCESSING_TIMES,nombre_archivo)

            ASSEMBLY_TIMES=[round(random.uniform(1,100*AM[a]*2),1) for j in
range(trabajos[t])]
            write_tag('ASSEMBLY_TIMES',ASSEMBLY_TIMES,nombre_archivo)

            SETUP_TIMES=[[round(np.random.uniform(1,20),1) for i in range(trabajos[t])] for j in
range(trabajos[t]+1)]
            for j in range(trabajos[t]+1):
                for i in range(trabajos[t]):
                    if (j==i+1):
                        SETUP_TIMES[j][i]=0

            write_tag('SETUP_TIMES',SETUP_TIMES,nombre_archivo)
            alpha=[0.6,0.8,1]
            for l in range(len(alpha)):
                R = [round(np.random.uniform(1, alpha[l] * max([sum(PROCESSING_TIMES[i]) for i
in range(DM[d])])), 1) for j in range(trabajos[t])]
                write_tag('R'+str(alpha[l]),R,nombre_archivo)
```

```
        JOB_NAMES=[j for j in range(trabajos[t])]
        write_tag('JOB_NAMES',JOB_NAMES,nombre_archivo)
        MACHINE_NAMES=['DM'+str(i) for i in range(DM[d])]
        MACHINE_NAMES.append('AC')
        for i in range(AM[a]):
           MACHINE_NAMES.append('AM'+str(i))
        write_tag('MACHINE_NAMES',MACHINE_NAMES,nombre_archivo)

os.chdir(ruta_padre)
```