Proyecto Fin de Carrera
Grado de Ingeniería en Tecnologías Industriales

Comparación de reglas de despacho para el problema de ensamblado en dos etapas con varios objetivos

Autor: María Garrido Lupiáñez
Tutor: Carla Talens Fayos

Departamento de Organización Industrial y Gestión de Empresa I
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2022

Proyecto Fin de Carrera
Grado de Ingeniería en Tecnologías Industriales

Comparación de reglas de despacho para el problema de ensamblado en dos etapas con varios objetivos

Autor:
María Garrido Lupiáñez

Tutor:
Carla Talens Fayos

Departamento de Organización Industrial y Gestión de Empresa I
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2022

Proyecto Fin de Grado: Comparación de reglas de despacho para el problema de ensamblado en dos etapas con varios objetivos

Autor: María Garrido Lupiáñez

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, 2022

El Secretario del Tribunal

A mi familia y amigos

A mis profesores

Agradecimientos

Agradecer a mis padres y a toda mi familia, por su apoyo incondicional fuera y dentro del ámbito estudiantil, sin ellos nada habría sido posible. Por compartir cada momento bueno y malo junto a mí. A mi hermano, por ser pilar fundamental de mi vida.

A mis compañeros de la Escuela, a los de Industriales y a los que no, porque sin el apoyo grupal, este largo camino no hubiera terminado. Gracias a ellos he aprendido el sentido del compañerismo y por encima de todo, el de la amistad. En especial a Manuel, por ser otro hermano para mí, cuidarme y apoyarme siempre.

A la Escuela, a mis profesores y a mi tutora por ayudarme a cumplir uno de mis sueños, y no dejarme desistir en ningún momento.

María Garrido Lupiáñez

Sevilla, 2022

Resumen

El objetivo de toda empresa es intentar optimizar su producción todo lo posible para así obtener los mejores resultados. Es aquí donde entra en juego la programación de operaciones. El presente trabajo consiste en resolver un problema de ensamblado dividido en dos etapas.

La primera etapa procesa las componentes de cada trabajo en un entorno de máquinas paralelas y la segunda etapa las une en máquinas paralelas no relacionadas. Para obtener la mejor de las secuencias se aplicarán diferentes reglas de despacho para dos objetivos no ponderados, *makespan* (C_{max}) y *total completion time* ($\sum C_j$).

El programa se ejecutará para diferentes instancias con datos generados de manera aleatoria. La desviación de los datos obtenidos va a ser cuantificada con el indicador ARPD el cual permitirá comparar entre sí las diferentes reglas de despacho. De esta manera se podrá determina cual es la que funciona mejor según el objetivo correspondiente.

Abstract

The objective of every company is to optimise the production as much as possible to obtain better results. In this point we will analyse the programming of operations. This project consists of solving an assembly problem divided into two stages.

The first stage processes the components of each job in a parallel machine environment. The second stage joins them together on unrelated parallel machines. In order to obtain the best of the sequences, we will implement different dispatching rules will for two unweighted objectives, makespan (Cmax) and total completion time ($\sum Cj$).

The program will be run for different instances with randomly generated data. The deviation of the obtained data will be quantified with the ARPD indicator, which will allow comparing the different dispatch rules with each other. By calculating this indicator, we will obtain the best dispatching rule for each objective.

Índice

Índice de Figuras

Índice de Gráficas

Índice de Tablas

Notación

α	Características de la máquina
β	Restricción
γ	Objetivo
P_i	*i* máquina idénticas
Q_i	*i* máquina relacionadas
R_i	*i* máquina no relacionadas
F_m	Taller de flujo con *m* etapas
J_m	Entorno taller con *m* etapas
O_m	Taller abierto con *m* etapas
Pmtn	Permutación
Prec	Precedencia
s_{ij}	*Set-up* trabajo *j* en máquina *i*
s_{ijk}	*Set-up* trabajo *j* en máquina *i* con secuencia *k*
Prmu	Permutación
j	Trabajo
i	Máquina
p_{ji}	Tiempo trabajo *j* máquina producción *i*
at_{ji}	Tiempo trabajo *j* máquina de ensamblaje *i*
C_{ij}	Tiempo finalización trabajo *j* en la maquina *i*
F_j	Tiempo de flujo trabajo *j*
L_j	Tardío trabajo *j*
T_j	Retraso trabajo *j*
E_j	Adelanto trabajo *j*
U_j	Nº trabajos atrasados o adelantados
w_j	Peso trabajo *j*
FAM	Primera máquina disponible
ECT	Menor tiempo de finalización
FCFS	Primer trabajo terminado
SPT	Creciente
LPT	Decreciente
EDD	Orden fechas de entrega más tempranas
WSPT	Orden según tiempo de proceso y peso de trabajos
HILO	Intercalación trabajos bajos y altos valores
$\sum C_j$	Suma tiempos de finalización trabajos *j*

C_{max}	Tiempo de finalización máximo
r_j	Fecha de llegada del trabajo *j* a la fábrica para procesarse
d_j	Fecha de entrega del trabajo *j*
ARPD	Average Relative Percentage Deviation

1 INTRODUCCIÓN

"En algún lugar, algo increíble está esperando ser conocido".

- Carl Sagan -

Este trabajo se basa en la programación de operaciones, siendo ésta una técnica para la organización lógica de tareas en un proyecto de fabricación, ofreciendo un programa de producción ordenado y con las tareas ubicadas en el tiempo para ser realizadas.

Este proceso productivo consiste en trasformar en salida ciertas entradas (recursos) ofrecidas al programa, siendo estas salidas el producto final a ofrecer al público. Todo ello bajo unos requerimientos predefinidos que acoten el problema a las necesidades específicas para las que se programen.

Una correcta planificación de la producción puede ofrecer a nuestro proyecto múltiples ventajas que lo hagan diferenciarse del resto y obtener una óptima programación. Por ejemplo, realizar pedidos de materiales y programación de los transportes de distribución de manera óptima o controlar una posible desviación o retraso en los pedidos. En definitiva, soluciones que llevarán a nuestra empresa a reducir costes y aumentar sus beneficios.

El uso de la tecnología y los avances en la programación y automatización de procesos en fábricas permite gestionar todo tipo de procesamientos dentro de la fabricación. Estas mejoras van a ser influyentes en aspectos fundamentales como la previsión de recursos, la gestión de las operaciones a realizar en la propia producción o el posterior análisis de los procesos para futuras mejoras e implantaciones.

Estos avances tecnológicos se ven directamente reflejados en parámetros como los tiempos de producción, mantenimiento de las máquinas, trazabilidad entre las diferentes etapas de la producción, previsión de la demanda, minimización de los retrasos en las órdenes de entrega.

La programación óptima es compleja debido a la tipología de problemas a tratar, es por ello por lo que en principio el objetivo fundamental será encontrar un programa que sea admisible para nuestro problema, y posteriormente se tratará de mejorar la solución hasta obtener la óptima o la más cercana a ella en caso de que sea imposible obtenerla en un tiempo de ejecución dentro de valores con lo que poder trabajar.

El programa final debe contar con unas horas de comienzo y fin para cada máquina y trabajo a procesar en función a una secuencia que ordene los trabajos según criterios definidos (será el objetivo de este proyecto fin de grado).

1.1 Objetivo del proyecto

La programación de operaciones se realiza con el fin de optimizar algún objetivo concreto. A continuación, se detalla el problema de estudio de este proyecto, así como los objetivos que persigue y las características generales que lo definen.

Se trata de un problema de producción de ensamblaje con múltiples máquinas no relacionadas en dos etapas diferentes. En la primera etapa hay varias máquinas paralelas dedicadas, en las que se

producen los componentes del producto final, fabricándose cada uno de ellos en una máquina diferente. Los tiempos de proceso de cada trabajo en cada una de las máquinas se denotan en el resto del proyecto como p_{ij}, siendo diferentes para cada una de ellas.

La siguiente etapa es la de ensamblado donde todos los trabajos que han sido procesados en la etapa anterior se unen. Es por eso por lo que en este momento cada trabajo solo debe acudir a una de las máquinas. Al ser máquinas no relacionadas, cada una posee un tiempo diferente de producción. Esos tiempos son definidos con la notación at_{ij}. Siendo en todo momento i la máquina y j el trabajo a realizar.

En esta última etapa existe un conflicto de asignación ya que los tiempos son diferentes, es por eso por lo que hay que determinar un criterio. En el caso de este proyecto se tienen en cuenta dos de ellos. En primer lugar, el criterio FAM (primera máquina libre) y posteriormente ECT (máquina con menor tiempo de finalización para dicho trabajo).

Además, a todo programa hay que indicarle una secuencia de trabajos sea o no la que proporcione la mejor solución ya que en ocasiones es imposible encontrarla inicialmente. Para ello existen multitud de métodos en la teoría. Este proyecto se centrará en la prueba de varias reglas de despacho con diferentes combinaciones y métodos de ordenación. Finalmente, se escogerá la mejor solución de las probadas ya que las reglas de despacho no proporcionan una solución óptima al tratarse de un método aproximado.

Una regla de despacho es un método de resolución aproximado, que devuelve una secuencia de trabajos ordenándolos en función de un indicador, que puede tener en cuenta diversas características del problema. Estos indicadores en nuestro caso serán combinaciones matemáticas obtenidas de operar con los tiempos de proceso y las máquinas de cada etapa para cada trabajo. Posteriormente, se le aplicará unos criterios de ordenación definidos en la teoría y en ocasiones se recurrirá al algoritmo de Johnson. Finalmente se obtendrá la mejor solución para cada una de las ordenaciones establecidas, en función de los datos de las instancias cargadas.

El desarrollo de la programación para la ejecución de estas reglas de despacho se realizará en lenguaje Python, mediante el intérprete gratuito *Spyder*. Se ejecutarán trescientas sesenta instancias con diferentes tiempos de proceso, número de máquinas en cada etapa y trabajos. Todos los datos serán volcados a una hoja de cálculo en la que se analizarán los resultados obtenidos y se determinará que regla de despacho funciona mejor para cada uno de los objetivos definidos.

1.2 Estructura del documento

A continuación, se detallan los capítulos de los que cuenta este trabajo, así como una breve explicación de lo que engloba cada uno de ellos.

1) Introducción: Breve contextualización en la programación de las operaciones de manera general. A continuación, una breve descripción del problema que se va a acometer en el presente trabajo, así como una mención a la estructura del documento para su mejor asimilación.
2) Marco teórico: Desarrollo de la teoría principal aplicada al proyecto. Definición de las características principales a la hora de definir un problema en un entorno de programación de operaciones. Los temas principales desarrollados serán: los entornos de trabajo, las restricciones posibles de aplicar a un programa y los principales objetivos de la teoría.
3) Descripción del problema: Se describe de forma detallada el problema, así como las funciones objetivo a analizar y el procedimiento. Además, se realiza una especificación de todas las restricciones y suposiciones para tener en cuenta para la programación de este.

4) Metodología: Explicación detallada del método para la creación de reglas de despacho y posterior obtención de los valores para los objetivos explicados con anterioridad. Además, se adjuntan funciones del código de Python relevantes en la estructura de la ejecución del programa.
5) Evaluación computacional: Obtención de resultados al ejecutar la totalidad de las instancias cargadas con el programa. Tablas comparativas de resultados mediante el indicador APRD (Average Relative Percentage Deviation). Ofrece información sobre la desviación a la que se encuentra nuestra función objetivo, en nuestro caso, del mínimo obtenido en cada instancia.
6) Conclusiones: Conclusión sobre la mejor solución obtenida, así como el objetivo para el que mejor funciona el método. Además, conclusiones extraídas sobre la automatización y mejora del código de resolución en Python.

2 MARCO TEÓRICO

"Cada día sabemos más y entendemos menos".

- Albert Einstein -

El objetivo de este proyecto es la programación de trabajos para varios objetivos según unas secuencias de trabajos determinadas. A continuación, se expone la teoría empleada a la hora de la realización del proyecto. Inicialmente, se muestra como es la notación de estos tipos de problemas y posteriormente se desarrollan todas las posibilidades de estos.

Típicamente la forma en la que se denotan este tipo de problemas es mediante tres condiciones, a saber, características de las máquinas (α), restricciones del problema (β) y objetivos perseguidos por el problema (γ).

*Notación de Graham et al. (1979): "*α / β / γ".

2.1 Entornos (α)

Se clasifican según el tipo de entorno de trabajo. Se pueden combinar entre ellas siempre que sean compatibles y se puede realizar una programación.

2.1.1 *Single machine* (α = 1)

No hay una ruta de trabajo porque todos los trabajos deben procesarse en esa máquina. El trabajo posee una única operación para completarse y finalizarse.

2.1.2 *Parallel machines* (α = Pm / Qm / Rm)

Pueden existir tantas máquinas como se requiera, pero se encuentran en paralelo. Depende de la velocidad de producción de la máquina. Por ejemplo, es posible que sea una máquina con las mismas características que otra presente en taller, pero al ser un modelo nuevo la velocidad de producción sea menor.

Todas las máquinas suelen ser apropiadas para realizar el trabajo y los tiempos de proceso pueden depender de la máquina paralela en la que se esté produciendo.

Se detallan las variantes de máquinas paralelas principales:

- Idénticas ($\alpha = P_m$): Como son todas las máquinas exactamente iguales, tan sólo están condicionadas por un tiempo de proceso, por trabajo, en cualquier máquina ya que son coincidentes.
- No relacionadas ($\alpha = R_m$): Son máquinas diferentes, pero se encuentran en paralelo. Todas pueden realizar todos los trabajos, pero en cada una de ellas el tiempo será independiente de la otra. Por este motivo para asignar un trabajo a alguna de ellas se necesita definir una regla de

asignación concreta, que se desarrollará más adelante.

- Uniformes ($\alpha = Q_m$): Se tratan de máquinas paralelas que están relacionadas ya que realizan la misma función, pero a diferentes velocidades. Es por ello, por lo que hay que otorgarle, como dato al problema la velocidad en la que la máquina trabaja.

2.1.3 Entornos de trabajo tipo taller

Se trata, en este caso, de máquinas en serie, de manera que los trabajos pasan por todas las máquinas que se definan. El orden y las características de cada entorno de trabajo hacen que se pueda distinguir en función de las rutas de trabajo empleada en cada caso:

- *Flowshop* ($\alpha = F_m$): Misma ruta predeterminada. Se precisa por tanto de una matriz en la que se definan los tiempos de proceso en cada máquina y en todas ellas se debe cumplir la secuencia inicial proporcionada en el caso de que no se permita permutación. Los trabajos pasan por todas las máquinas sin una ruta definida.
- *Jobshop* ($\alpha = J_m$): Hay diferentes rutas de trabajo, en función de la máquina en la que me encuentre. Se ofrece una ruta determinada ya que el trabajo debe pasar por cada máquina en un orden determinado, por ello las secuencias en cada máquina siempre van a ser diferentes. En la práctica hay que codificar una ruta para cada máquina de manera que se eviten ciclos entre máquinas y que un trabajo se procese en dos a la vez.
- *Openshop* ($\alpha = O_m$): No hay nada definido, es el más general y por tanto el más complejo de programar. No hay rutas definidas, se da una secuencia para cada máquina y de ese modo se genera la especie de ruta a seguir. Es modificable continuamente en función de las secuencias.

2.1.4 Entorno híbrido

Se trata de un taller en lo que se conjugan todos los entornos vistos anteriormente, con relación entre ellos. A cada entorno se le llama etapa, de modo que el trabajo para completarse pasa por cada una de ellas en el orden establecido. Cada etapa estará separada por "/" en la notación, lo que definirá cada problema en concreto. Al ser combinables la tipología de definiciones de problemas es muy elevada.

2.2 Restricciones (β)

Son las condiciones que afectan al problema, y pueden estar referidas al entorno, trabajo, característica de la máquina o cualquier elemento que interfiera en el proceso. De esta manera limitan las soluciones y pueden llegar a hacer de un problema complejo algo trivial o viceversa.

2.2.1 Relación con interrupciones

Cuando el trabajo indicado comienza a producirse, existe la opción de interrumpir la operación. Puede ser por motivos técnicos o de prioridad a otro trabajo. En ocasiones es imposible ya que los tiempos de reinicio de la máquina pueden ser elevados o el trabajo en cuestión es imposible que se divida su producción.

Los posibles tipos de interrupción a tratar son las siguientes:

- No recuperable (β = *Pmtn-non-resumable*): El trabajo que se encuentra en proceso se pierde, cuando se reinicia el sistema comienza uno nuevo. Es útil para trabajos que no suponga valor perder una unidad.
- Semi-recuperable (β = *Pmtn-semi-resumable*): Se recupera parte del trabajo cuando se reinicien las operaciones.

- Recuperable (β = *Pmtn-resumable*): La interrupción no genera ninguna pérdida ya que cuando se vuelva a reiniciar continuará por el mismo lugar donde se quedó.

2.2.2 Relacionadas con fechas de entrega

- Llegada (β = r_j): Se tratan de las fechas en las cuales los trabajos están disponibles para comenzar a procesarse.
- Entrega (β = d_j): Sólo serán restricciones cuando se trate de uno de los siguientes casos porque normalmente será a modo informativo, sirve para contabilizar los retrasos o la prioridad en la entrega a clientes.
  - Fechas fijas (β = $\overline{d}_j$): El trabajo no puede superar esa fecha de entrega. Si la supera no se puede realizar. Es muy restrictiva, el objetivo T_j deja de tener sentido.
  - Fechas comunes (β = $d_j = d$): Las fechas de entrega de todos los trabajos es la misma. Si se sobrepasa un trabajo dicha fecha, ya todos van a ir retrasados, aunque se modifique el orden de las operaciones.

2.2.3 Precedencia (β = prec)

Un trabajo no puede procesarse hasta que lo hagan sus predecesores. El orden puede ofrecerse de cualquier modo, por ejemplo:

- Normalmente se ofrece un árbol de precedencia que se trata de un grafo cuyos nodos son los propios trabajos.

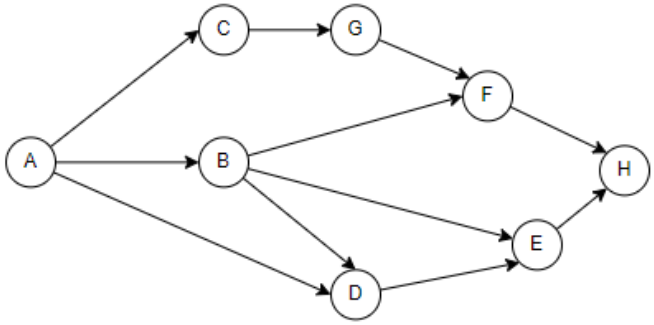


Figura 2-1: Ejemplo árbol de precedencia de trabajos. *[Fuente: Saúl Zalimben, 2022]*

- También puede expresarse en forma de matriz de precedencia:

sucesoras

antecesoras

	A	B	C	D	E	F	G
A			X	X			
B				X			
C					X		
D						X	
E							X
F							X
G							

Figura 2-2: Ejemplo matriz de precedencia de trabajos *[Fuente: María Teresa Arevalo, 2013]*

- Otra opción es la tabla de precedencia de cada trabajo:

Actividad	Predecesor(es)
A	None
B	A
C	B
D	A
E	D
F	C, D
G	F, E

Figura 2-3: Ejemplo tabla de precedencia de trabajos *[Fuente: Ignacio Manzanera]*

2.2.4 Tiempos de *set-up*

Se trata del tiempo de cambio, preparación, de la maquinaria para que se comience a procesar un trabajo nuevo. Puede ser intrínseco de la máquina, depender del trabajo procesado anteriormente, del trabajo en sí o de todo en conjunto.

- Dependiente del trabajo ($\beta = S_j$): Es independiente de la máquina dónde se procese el trabajo. Cualquiera de ellas precisa de ese *set-up* ya que está relacionado con el trabajo en cuestión.
- Dependiente del trabajo y de la máquina ($\beta = S_{ij}$): En este caso no sólo depende del trabajo que se trate sino también de la máquina donde se procese.
- Dependiente trabajo, máquina y secuencia ($\beta = S_{ijk}$): Ahora depende de todo lo anterior, trabajo y máquina, pero también de la secuencia de procesamiento de trabajo, es decir, del que lo precede. Puede ser por ejemplo que al tratarse de ciertos trabajos haya que modificar utillaje o realizar labores de limpieza en ciertas máquinas.

Además, dichos tiempos de set-up se dividen en anticipatorio o no en función de cuando se realiza:

- Anticipatorios: El *set-up* puede adelantarse a que el trabajo llegue a esa máquina de manera que cuando acabe en la anterior comience directamente a procesarse en la siguiente.
- No anticipatorio: Es imposible que se realice el *set-up* si el trabajo no ha llegado a dicha máquina.

2.2.5 Lotes (*batch*)

Sucede cuando las máquinas pueden procesar a la vez un lote de trabajos concreto. De esta manera los tiempos de proceso ahora pasan a ser los de los lotes de las máquinas y no lo de los trabajos en sí.

- En paralelo (β = P-*batch*): El tiempo de proceso de los lotes pasa a ser el mayor de los trabajos de dicho lote ya que hasta que no se procese ese último no se completa el bloque y no se pasa de máquina.
- En serie (β = S-*batch*): Los trabajos se realizan de manera sucesiva dentro del bloque por tanto el tiempo de proceso será la suma de todos ellos. Normalmente provoca que se ahorre tiempos de

set-up.

2.2.6 Entorno taller

Se tratan de restricciones que condicionan directamente aspectos relacionados con el taller y la organización de este.

- Permutación regular (β = Prmu): Los trabajos tienen una permutación regular, es decir, en todas las máquinas del *flowshop* se sigue la misma secuencia de trabajos.
- No tiempos ociosos (β = *No-idle*): No se permiten los tiempos ociosos (sin trabajos) en las máquinas. Una vez empiece el programa no puede detenerse la producción de la máquina.
- No espera (β = *No-wait*): Los trabajos una vez iniciados en una máquina no pueden esperar para pasar a la siguiente, así hasta que se finalice el proceso completo. De este modo no se puede iniciar un trabajo, hasta que no se asegure que el resto de las máquinas van a permanecer libres a continuación de este.
- Almacén de la máquina (β = *Buffer*): Espacio dónde esperan los trabajos a ser procesados. Dependen de la máquina en cuestión. Para los casos teóricos suele suponerse infinito, ya que no se tiene en cuenta el tiempo de trasladar el trabajo del lugar donde este almacenado en la fábrica a la máquina en cuestión.

2.3 Objetivos (γ)

Una programación se realiza para cumplir un objetivo marcado por la empresa, en función de las necesidades de ésta. Como norma general, se entiende que el objetivo común será el de reducir tiempos de producción, ya que hará que se cumplan los relacionados con fechas de entrega, satisfacción con cliente o costes de producción. Aun así, existen ejemplos de objetivos en los que lo interesante no es minimizar. A continuación, se detallan los objetivos más típicos en la programación de operaciones siendo j un trabajo determinado de la secuencia a planificar.

2.3.1 Tiempo de finalización ($\gamma = C_j$)

Como objetivo general se cuantifica el tiempo de finalización (cuando recorre todas las máquinas o etapas correspondientes). Aun así, para cálculos intermedios es necesaria o útil la obtención de los tiempos de finalización en máquinas o etapas intermedias.

2.3.2 Tiempos de flujo ($\gamma = F_j$)

Se trata del tiempo que el trabajo se encuentra en el entorno de trabajo, es decir, en producción. La definición matemática es la siguiente, siento rj el instante en el que trabajo puede comenzar y C_j el tiempo de finalización de este en dicha maquina o etapa.

$$F_j = C_j - r_j \qquad [2.1]$$

2.3.3 Retraso ($\gamma = L_j$)

Puede ser mayor o menor que cero ya que relaciona directamente las fechas de entrega con los tiempos de finalización de cada trabajo. Si es positivo quiere decir que el trabajo se ha realizado a tiempo, sin embargo, cuando es menor que cero el trabajo va retrasado.

$$L_j = C_j - d_j \qquad [2.2]$$

2.3.4 Tardanza ($\gamma = T_j$)

Se establece un máximo entre 0 y el tiempo de retraso ya que se contabilizan sólo los tiempos de retraso de los trabajos atrasados, si se entregan antes de la fecha de entrega no contabilizan porque no pasa nada.

$$T_j = max\{0, L_j = C_j - d_j\} \qquad [2.3]$$

2.3.5 Adelanto ($\gamma = E_j$)

Ahora el máximo es entre 0 y el menos tiempo de retraso ya que se contabilizan sólo los trabajos no atrasados, si se entregan después de la fecha de entrega no contabilizan porque no pasa nada. La idea es que estos trabajos no se pueden entregar antes de la fecha acordada, por ejemplo, si se trata de productos perecederos.

$$E_j = max\{0, -L_j = d_j - C_j\} \qquad [2.4]$$

2.3.6 Número de trabajos tardíos ($\gamma = U_j$)

Este objetivo contabiliza el número de trabajos que acaban tarde sin tener en cuenta cuando tiempo se ha retrasado cada uno. Se aprovecha la fórmula de las tardanzas, T_j para formular dicho objetivo como:

$$U_j = \begin{cases} 1 \; T_j > 0 \\ 0 \; cc \end{cases} \qquad [2.5]$$

2.3.7 Objetivos ponderados

Los objetivos pueden tener un peso o importancia superior en el caso de ser un trabajo u otro. Para ello se definen unos w_j (pesos) a cada trabajo. Se puede emplear en todos los objetivos anteriormente mencionados.

Todos los objetivos son cuantificados en función del sumatorio de finalización (o suma de cualquier objetivo) de cada trabajo, ofrece el total de la producción, o bien, el máximo de cualquier objetivo que acumula y sólo cuantifica el término final del problema.

En la descripción del problema es necesaria una secuencia que ordene la manera en la que se procesan los trabajos. Esta obtención de la secuencia suele ser unos de los objetivos a obtener cuando se acomete un programa de operaciones.

Para conseguir la secuenciación óptima existen métodos para problemas concretos, así como algoritmos que ofrecen soluciones admisibles que mejoran nuestro problema. En función de la polinomialidad del mismo estas soluciones pueden ser exactas o no.

En nuestro caso el problema contará con el análisis de diferentes secuenciaciones de trabajos aplicando reglas de despacho diferentes para dos objetivos no ponderados, *makespan* (C_{max}) *y total completion time* $\sum C_j$.

3 DESCRIPCIÓN DEL PROBLEMA

"No entiendes realmente algo a menos que seas capaz de explicárselo a tu abuela".

- Albert Einstein -

En este apartado se detalla, en primer lugar, el problema objeto de estudio de este trabajo, programación de un proceso de ensamble de componentes en dos etapas con máquinas paralelas, pero no idénticas en la primera etapa, y no relacionadas en la segunda.

Los tiempos de procesos se sabe que son diferentes para cada máquina. Se realizará el estudio posterior combinando diferentes números de máquinas en cada una de las etapas, así como el número total de trabajos y componentes a procesar.

Para comenzar, se describen a continuación los detalles del problema: entorno, funciones objetivos y restricciones para tener en cuenta, así como una descripción formal de manera matemática de ambos objetivos, a calcular mediante la notación de Graham et al. (1979), $\alpha / \beta / \gamma$.

En el último punto de este apartado, se realiza un ejemplo para una instancia de dimensiones muy reducidas para cada regla de asignación en la segunda etapa. Los resultados finales serán diferentes en función del criterio empleado, FAM o ECT.

3.1 Introducción al problema

El problema se desarrolla en un entorno *flowshop* de dos etapas, donde las piezas de cada trabajo son realizadas por las m_is máquinas de la primera etapa (cada máquina fabrica un componente del producto) y ensambladas en las at_i máquinas de la segunda etapa. Al emplearse tan sólo una máquina de las disponibles en ensamblaje es necesario definir un criterio de asignación para la selección de la máquina donde se unan las componentes, en nuestro caso, FAM (*First Available Machine*) y ECT (*Earliest Completion Time*).

Los objetivos a estudiar van a ser *total completion time* ($\sum C_j$) y *makespan* (C_{max}). Ambos se obtienen a partir del vector de finalización de los trabajos, C_{ij}.

- $\sum C_j$: Será la suma de todas las componentes del vector de finalización, ya que representa el tiempo total de terminación de todos los trabajos al completo.
- C_{max}: Determina el tiempo máximo de finalización del último trabajo ensamblado. De esta manera, a efectos matemáticos, se trata del máximo del vector de tiempos de finalización.

Ambos objetivos están por tanto relacionados. Aunque en la programación de operaciones existen multitud de objetivos a optimizar, estos dos son los más relevantes a niveles generales, ya que si se reducen los tiempos de finalización la producción será en la mayoría de los casos más rentable, pudiendo cumplir a la vez objetivos relacionados, como, por ejemplo, el de tardiness (retraso en las

fechas de entrega).

Los tiempos de procesos en las máquinas son diferentes entre sí. Por lo que hay que distinguir entre los de la etapa de fabricación de componentes (p_{ij}) y los tiempos de ensamblaje (at_{ij}).

A continuación, se detallan el resto de las consideraciones para tener en cuenta a la hora de realizar la programación de la obtención de dichos objetivos:

- En la primera etapa cada máquina es capaz de procesar a la vez el mismo trabajo sin posibilidad de interrupción, es decir, cuando comienza a procesarse un trabajo en la primera de las máquinas de la etapa, ese trabajo debe ser completado en cada una de ellas sin excepción.
- Sólo cuando el trabajo ha sido procesado por cada una de las máquinas de la primera etapa, es posible que pase a la segunda etapa. Siendo ese tiempo de finalización el inicial de la segunda etapa.
- No existen consideraciones previas sobre la prioridad en la finalización de los trabajos. A priori, no se tendrán en cuenta fechas de entrega, y las secuencias de procesamiento se modificarán con diferentes reglas de despacho hasta obtener una solución mejor que todas para el problema. Por tanto, la mejor solución se obtendrá tras la realización de una multitud de ensayos y nunca será el óptimo global del problema. Esto sucede al estar aplicando reglas de despacho, método aproximado.
- Las máquinas no tienen los mismos tiempos de proceso en ninguna de las etapas, puede variar el número y el tiempo de procesamiento, ya que en la primera etapa son paralelas, pero no idénticas DP_i y en la segunda no relacionadas R_i, siendo *i* el número de máquinas en cada etapa.
- El buffer de las máquinas se va a suponer infinito. Debido a la complejidad del problema en sí, no se va a tener en cuenta los tiempos de reposición de trabajos en cada máquina. El *buffer*, se refiere al "almacén" de cada máquina, es decir, lugar previo donde se preparan los trabajos que va a ser procesados. Al suponerse infinito, no se tendrá en cuenta los tiempos de desplazamiento de los materiales necesarios porque se supone que se encuentran ya preparados.
- Las fechas de llegada (r_j) se toman nulas, es decir, se supone que todo material necesario se va a encontrar en las instalaciones en su totalidad al comentar el programa. A su vez, los *set-up* de cada máquina y trabajo se definen nulos para todo el problema, ya que no se tiene en cuenta ningún tiempo de preparación previo.

Para finalizar la descripción de este problema, se muestra la notación formal del mismo para cada uno de los dos objetivos a estudiar:

- *Total completion time*: $\sum C_j$

$$DP_m \rightarrow R_m||\sum C_j \quad [3.1]$$

- *Makespan*: C_{max}

$$DP_m \rightarrow R_m||C_{max} \quad [3.2]$$

3.2 Ejemplo para una instancia pequeña

En este apartado se va a resolver un ejemplo concreto para ilustrar completamente el objetivo de nuestro proyecto. Como se ha indicado previamente, se trata del análisis de comportamiento de nuestro problema en función de la aplicación de diferentes reglas de despacho, según los objetivos de Cmax (*makespan*) y $\sum C_j$ (*total completion time*).

3.2.1 Datos de partida

En este ejemplo, se ha proporcionado una matriz de tiempos de proceso que recoge el tiempo de cada uno de los cinco trabajos en las cinco máquinas. En la primera etapa, habrá dos máquinas en paralelo, pero no idénticas (todos los trabajos deben pasar por todas las máquinas) y en la segunda etapa, dos máquinas no relacionadas.

La secuencia de trabajos del ejemplo es la mostrada a la derecha (1, 3, 2, 0, 4). En Python, los vectores serán iniciados en 0 en vez de en 1, es por ello por lo que ya lo tratamos con esa nomenclatura.

ETAPA 1		ETAPA 2	
m1	m2	at1	at2
3	4	3	1
1	3	2	7
4	4	5	1
2	1	8	2
5	2	3	1

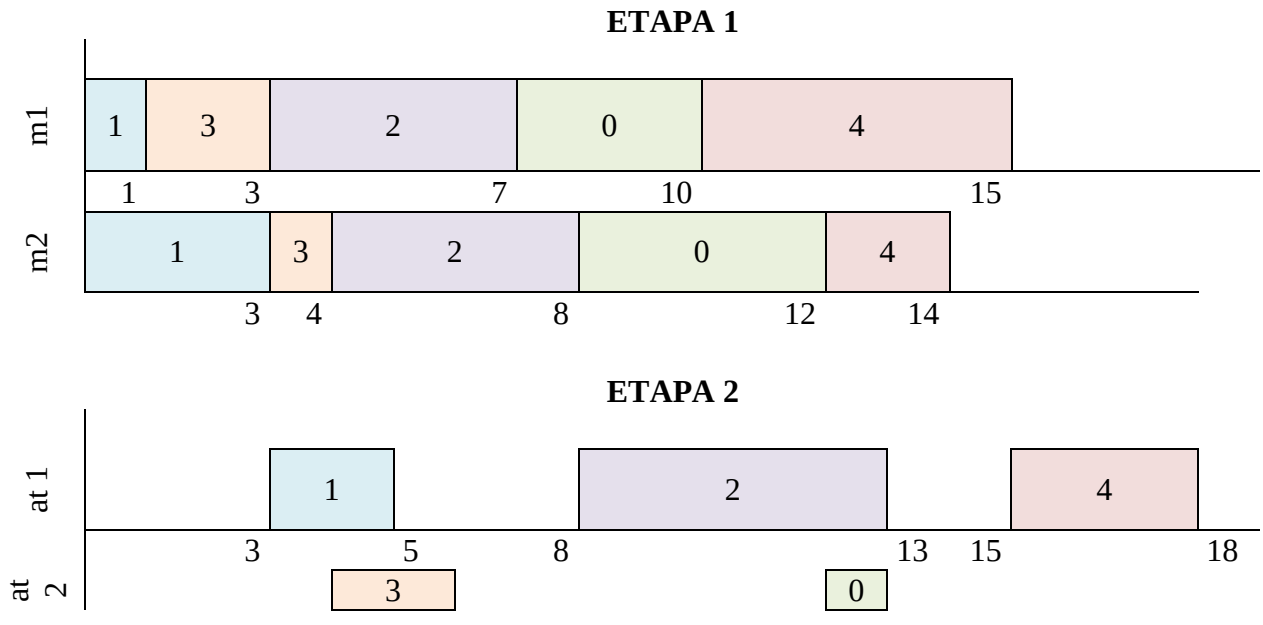
SECUENCIA DE TRABAJOS				
1	3	2	0	4

Tabla 3-1: Tabla de datos tiempos de proceso para cada etapa y secuencia de trabajos.
[Fuente: Elaboración propia]

A continuación, se detalla el diagrama de Gantt para ambos objetivos según FAM o ECT, según corresponda como regla de asignación en el momento de pasar a la segunda etapa. La etapa primera, será igual para ambos objetivos, ya que todos los trabajos deben procesarse en ese orden (secuencia) en todas las máquinas, de manera que el momento en el que se pasa el trabajo a la etapa dos, será el último tiempo de finalización en la etapa previa.

3.2.2 Problema con regla de asignación FAM

Cuando el trabajo pasa a la segunda etapa, se asigna a la primera de ellas que se encuentre disponible, sin tener en cuenta el tiempo de proceso en la segunda etapa. Se pasa a la siguiente máquina si se encuentra ocupada, asignándose a la máquina con menor índice en caso de empate.



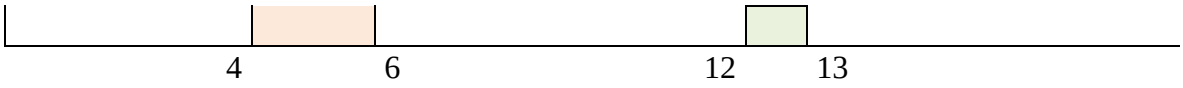


Figura 3-1: Diagrama de ejemplo, programación del problema con FAM. *[Fuente: Elaboración propia]*

3.2.3 Problema con regla de asignación ECT

En este caso, el trabajo será asignado a la máquina cuyo tiempo de proceso sea más pequeño. Teniendo en cuenta todas las máquinas, incluso las que se encuentren ocupadas en el instante de finalización en la etapa previa. En caso de empate se asigna a la que tenga menor índice *i*.

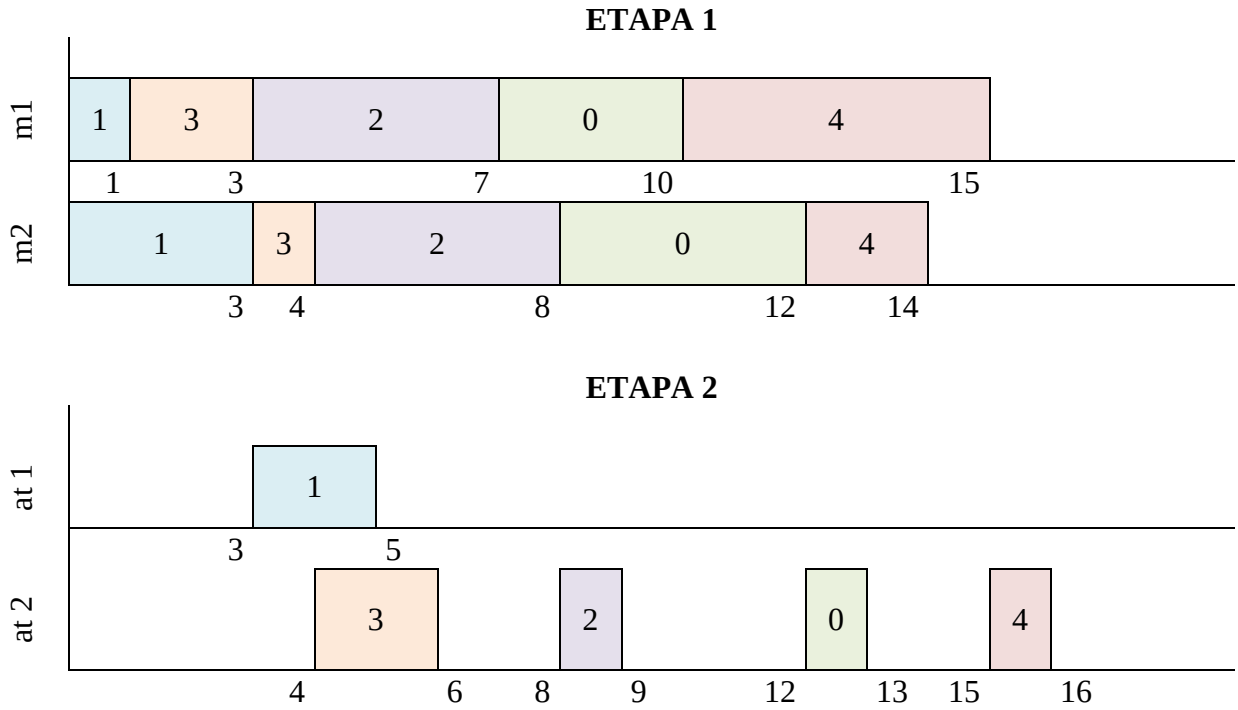


Figura 3-2: Diagrama de ejemplo, programación del problema con ECT. *[Fuente: Elaboración propia]*

3.2.4 Resultados

Los resultados, en este caso, son los que corresponden a los objetivos de nuestro problema $\sum C_j$ y Cmax. Para ello, hay que atender al vector final de tiempos de finalización de cada trabajo en este caso:

- FAM → (13,5,13,6,18)
- ECT → (13,5,9,6,16)

FAM		ECT	
C_{max}	$\sum C_j$	C_{max}	$\sum C_j$
18	55	16	49

Tabla 3-2: Resultados para cada objetivo con reglas de asignación FAM y ECT *[Fuente: Elaboración propia]*

Se observa que para ECT ambos objetivos son mejores, algo que tiene sentido ya que son fórmulas donde los elementos fundamentales son los tiempos de finalización, y ECT, reduce siempre esos tiempos debido al tipo de asignación. Al tratarse de un problema con una dimensión reducida, se observa también que la diferencia entre ambos no es muy elevada.

4 METODOLOGÍA

"La mejor estructura no garantizará los resultados ni el rendimiento, pero la estructura equivocada es una garantía de fracaso.".

- Drucker, Peter–

Tras definir los conceptos básicos de la programación de la producción aplicados a nuestro problema, así como la definición detallada del mismo, en este apartado procedemos a desarrollar la metodología empleada en el proyecto.

Se comienza proporcionando información sobre el lenguaje de programación empleado y el porqué de la elección de ese entorno de trabajo. A continuación, el método de resolución aplicado y finalmente la explicación de los objetivos a calcular.

4.1 Lenguaje de programación

Un lenguaje de programación es empleado para comunicarse y desarrollar programas de software, aplicaciones, páginas webs, scripts u otros conjuntos de instrucciones, para que sean ejecutadas por los ordenadores. Existen muchos tipos de lenguajes con características y funciones propias. Según el objetivo del proyecto, y la destreza del programador en la ejecución de éste, pueden ser más ventajosos algunos de ellos.

En el enlace web "index TIOBE", se puede consultar ese índice que nos marca en tiempo actual cuales, con los lenguajes más empleados, según estadísticas en todo el mundo. Actualmente, se enumeran los lenguajes más significativos en el siguiente orden:

Jun 2022	Jun 2021	Change	Programming Language	Ratings	Change
1	2	^	Python	12.20%	+0.35%
2	1	v	C	11.91%	-0.64%
3	3		Java	10.47%	-1.07%
4	4		C++	9.63%	+2.26%
5	5		C#	6.12%	+1.79%

Figura 4-1: Consulta index TIOBE fecha 12-06-2022 *[Fuente: Index TIOBE, 2022]*

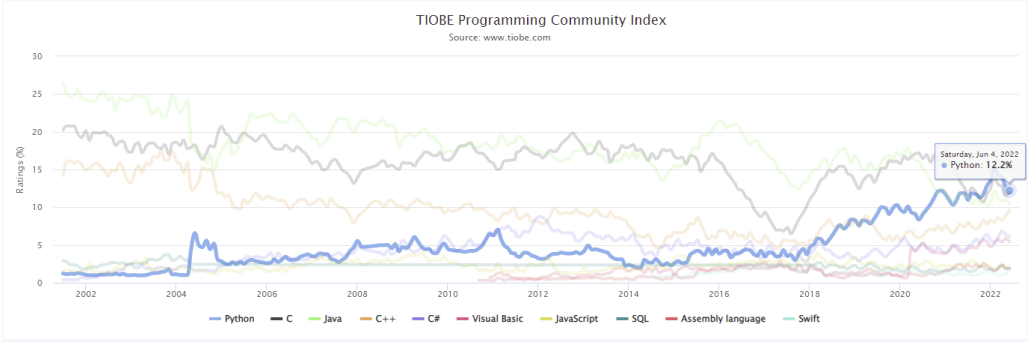


Figura 4-2: Consulta index TIOBE posición de Pyhton respecto a otros lenguajes *[Fuente: Index TIOBE, 2022]*

A continuación, se detalla en profundidad Python, ya que es el empleado en este TFG.

Figura 4-3: Icono de lenguaje Python *[Fuente: Rockbotic, 2018]*

Python, se trata de un lenguaje de programación interpretada y multiplataforma, creado por Guido Van Rossum a finales de los años 80. El objetivo de su creación fue realizar una alternativa a lenguajes básicos orientado a crear prototipos rápidos. Esta motivado, principalmente, por la cantidad de librerías de fácil acceso y velocidad a la hora de programar, debido a la sencillez de su interpretación.

El lenguaje ofrece muchas facilidades al programador por su sintaxis sencilla. Este hecho, hace que sea sencillo de aprender y fomenta por tanto la productividad. Es muy legible y los módulos están muy ordenados para que funcione, por tanto, es muy sencillo de interpretar. Actualmente, es el más usado a nivel mundial. Eso hace que los usuarios participen activamente en el desarrollo del lenguaje y que exista una gran amplia facilidad de búsqueda de soluciones mientras se desarrollan los programas.

Es un lenguaje interpretado, es decir, se ejecuta directamente instrucción a instrucción. Por ello, se precisa de un intérprete que lea la instrucción en tiempo real y la ejecute.

Un intérprete es un programa informático que actúa como interfaz. Las líneas de código deben traducirse a los correspondientes comandos legibles por máquina, y se envía directamente al procesador lo que hace que aumente el tiempo de ejecución. Algo positivo es que, de esta manera, una línea de código problemática se detecta inmediatamente, después de ocurrir el fallo, sin necesidad de esperar a finalizar el código para compilar. Eso hace que la depuración sea rápida e intuitiva. En este trabajo se trabaja con el intérprete Spyder.

Figura 4-4: Icono intérprete Spyder *[Fuente: Rockbotic, 2018]*

Se ha decidido desarrollar nuestra programación en lenguaje python por varios motivos. Es un lenguaje intuitivo y de fácil sintaxis. Por otro lado, empleando las librerías adecuadas, las cuales son fáciles de usar ya que tan sólo debes llamarlas al inicio del código, las operaciones matemáticas básicas se reducen a tan sólo un comando. Además, existen multitud de herramientas de interacción fundamental para recorres, vectores y matrices, lo cual será muy útil en nuestro proyecto.

La sintaxis de este lenguaje permite, además, reducir en gran medida la complejidad del código. Por otra parte, es posible trabajar con tuplas y listas. Estos elementos facilitarán el manejo de datos y soluciones. Las tuplas, por ejemplo, hacen que, en una misma operación, se pueda ordenar por tiempos de proceso, sin perder los indicadores que reflejan los trabajos a los que corresponde. Las listas permitirán, entre otras funcionalidades, leer matrices de distintas dimensiones, y guardarlas en las propias listas, como si se tratará de un vector. Optimizará por tanto la ejecución de nuestro proyecto. Además, se ha empleado el intérprete Spyder ya que es gratuito, y al ser gestionado por python oficial, tiene acceso a muchas librerías ya instaladas, y su manejo es de fácil consulta en web

debido a la gran comunidad que lo usa.

4.2 Estructura de ejecución del programa (metodología)

En este apartado se detalla, en forma de diagrama de flujo, la estructura que debe seguir nuestro código, de manera que se resuma de forma clara, los módulos importantes del código que a continuación se explica, y que, en la parte final de Anexos, se puede consultar en su totalidad.

1. Generación de instancias.
2. Lectura de instancias.
3. Indicadores.
4. Johnson y criterios de ordenación.
5. Reglas de despacho.
6. Obtención de secuencia.
7. Funciones objetivo según regla de asignación, FAM y ECT.
8. Guardar resultados en la hoja de cálculo según objetivo

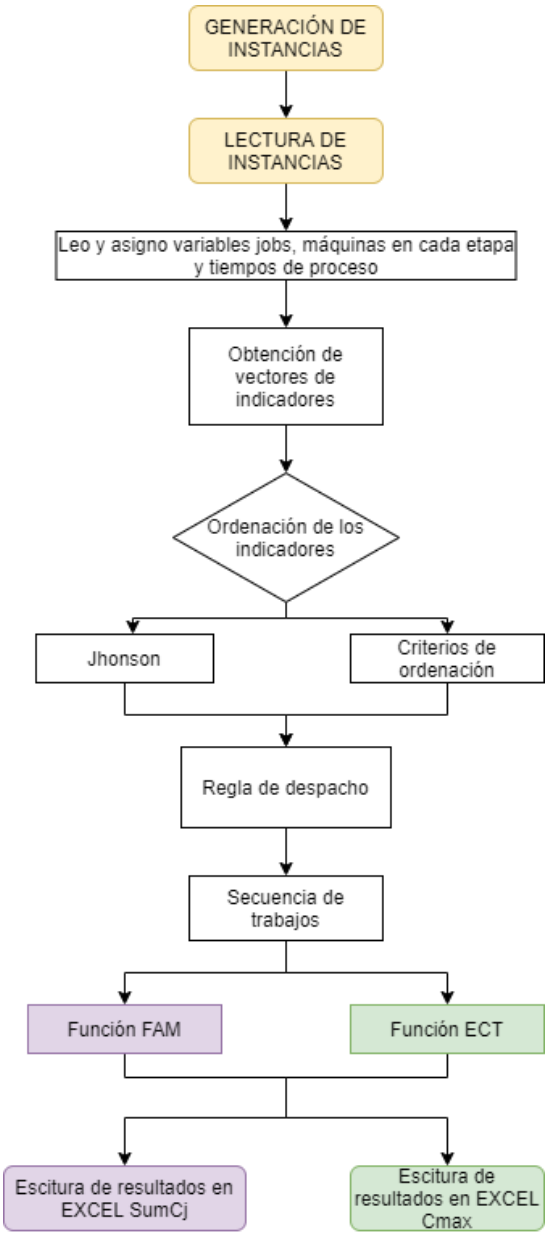


Figura 4.5: Estructura de ejecución del programa *[Fuente: Elaboración propia]*

4.3 Métodos de resolución

Existen dos métodos de resolución de la producción, los aproximados y los exactos.

- Exactos: Proporcionan una solución óptima. Son aplicables a problemas polinomiales y a algunos no polinomiales pero que han sido demostrados de manera matemática. A su vez, se pueden dividir en algoritmos constructivos o enumerativos.
  - Constructivos: Resuelve problemas polinomiales en un tiempo reducido, aunque el número de la instancia sea elevado. Aun así, la realidad es que existen muy pocos problemas de este tipo en la programación de la producción.
  - Enumerativos: Algoritmos no polinomiales que obtienen el óptimo del problema inspeccionando todo el espacio, es aplicable tan solo para instancias pequeñas ya que los tiempos de ejecución son elevados. Algunos ejemplos son la programación matemática mediante modelado y resolución con solver, Branch and Bound o la programación dinámica.

- Aproximados: No proporcionan el óptimo, pero si una solución factible. La eficiencia de estos algoritmos se obtiene al aplicar un gran número de ensayos. Dentro de esta tipología de algoritmos se encuentra la metaheurística y la heurística constructiva.
    - Metaheurística: Optimización local alrededor de una solución inicial. A medida que se va ejecutando el algoritmo se va mejorando la solución y se detiene cuando se cumple algún criterio de parada diseñado por el usuario, por ejemplo, numero de iteraciones, porcentaje de error o tiempo de ejecución.
    - Heurística constructiva: No se parte de una secuencia inicial, ésta es construida mediante la adición de trabajos y ordenación de estos según criterios establecidos. Un ejemplo de estos métodos son las reglas de despachos las cuales aplicaremos en el proyecto.

Para obtener dichas reglas en nuestro problema, se pasa a detallar la metodología aplicada:

1. Construcción de indicadores: Vectores que agrupen los datos por trabajos realizando operaciones matemáticas en la matriz de tiempos de procesos en ambas etapas.
2. Combinación de los indicadores: En algunas ocasiones se combinarán varios cálculos de indicadores para obtener diferentes reglas según la combinación entre ellos.
3. Ordenación de los vectores de indicadores: Por último, se ordenarán dichos vectores obteniendo finalmente los índices de dichos vectores, de esta manera obtendremos la secuencia de trabajos a procesar. La ordenación final se realizará mediante criterios de ordenación establecidos (SPT, LPT, Hill, Valley, HI/HILO, HI/LOHI, LO/HILO, LO/LOHI) o mediante la ejecución del algoritmo de Johnson.

Esta metodología ha sido tomada como propuesta de los artículos propuestos por Komaki & Kayvanfar (2015), Lee, Cheng, Lin (1993) y Lee, Cheng, Chou (2006). En ellos se propone la resolución mediante dichas reglas de despacho para el objetivo de *makespan* (C_{max}). En este proyecto, también se realizará el análisis para el objetivo de *total completion time*, de esta manera se probará su eficacia al aplicarlo a objetivos diferentes al propuesto.

A continuación, se detalla el cálculo de los vectores indicadores, junto a las expresiones matemáticas que los definen.

4.3.1 Indicadores

Se definen una serie de indicadores que, posteriormente, van a ser empleados para obtener las reglas de despacho. Estos indicadores, se combinarán en algunos casos, generando siempre un vector que luego será ordenado, según diferentes criterios, creando las secuencias de trabajo.

En Python, los vectores indicadores, se forman como *arrays* (vectores) aunque, posteriormente, en los criterios de ordenación, o al aplicar el algoritmo de Johnson, se tratarán como tuplas, guardando el valor de dicho indicador y el índice donde se encuentra en el vector inicial, ya que la secuencia que necesitamos es la que ordena los trabajos, no los indicadores.

En todos los casos, cuando se refiera a los tiempos de proceso de la primera etapa se denominan p_{ij} y los de la segunda etapa at_{ij} (*i* máquinas y *j* trabajos).

Los indicadores que se han necesitado programar son los siguientes:

- Máximo tiempo de proceso de cada trabajo: Máximo tiempo de proceso de cada trabajo *j* en cada máquina *i* de la primera etapa:

$$Max\ P_{ij} \quad [4.1]$$

- Código Python:

```
def max_p_ij (tp, et):
    return [np.max(j[0:et]) for j in tp]
```

- Promedio tiempos de proceso en la segunda etapa: Promedio de tiempos en las máquinas de la segunda etapa para cada trabajo. Se calcula la media de los valores de los tiempos de proceso de la segunda etapa para cada trabajo, es decir, sumatorio de tiempos de proceso de cada máquina *i* dividido entre el número total de máquinas en la segunda etapa para cada trabajo *j*.

$$\frac{\sum at_{ij}}{m_2} \quad [4.2]$$

- Código Python:

```
def at_j_m2(tp, et):
    return [np.mean(j[et-1::]) for j in tp]
```

- Tiempo de proceso en la máquina más cargada: Tiempos de proceso en la máquina *i**, siendo esa la máquina más cargada para cada trabajo *j*. Se suman todos los tiempos de todos los trabajos en la primera etapa, en cada máquina *i*. La máquina cuya suma sea la mayor, será la máquina que conformará el vector de indicadores en este caso, ya que se tratará de la que más cargada esté. Las componentes de dicho indicador serán por tantos los tiempos de proceso de esa máquina *i**.

$$P_{i*j} \quad [4.3]$$

- Código Python:

```
def p_ij(tp, et)
   return tp[:,np.argmax(np.sum(tp[:,0:et], axis = 0))]
```

- Suma de tiempos de proceso: Suma de los tiempos de proceso de las máquinas *i* para cada trabajo *j*.

$$\sum P_{ij} \quad [4.4]$$

- Código Python:

```
def sum_p_ij(tp, et):
    return [np.sum(j[0:et]) for j in tp]
```

- Promedio tiempos de proceso en la primera etapa: Promedio de tiempos en las máquinas de

la primera etapa, para cada trabajo. Se calcula la media de los valores de los tiempos de proceso de la primera etapa para cada trabajo, es decir, sumatorio de tiempos de proceso de cada máquina *i* dividido entre el número total de máquinas en la primera etapa para cada trabajo *j*.

$$\frac{\sum P_{ij}}{m_1} \quad [4.5]$$

- Código Python:

```
def sumpromd_p_ij(tp, et):
    return [np.mean(j[0:et]) for j in tp]
```

Para crear las reglas de despacho, a continuación, se detalla la relación entre ellos, en algunos casos se necesitará tan solo de un indicador (o combinación de varios en forma de suma) y en otros se precisará de dos indicadores, o bien, dos combinaciones de los mismos.

4.3.2 Combinación de indicadores

Se recuerda que las reglas de despacho tratan de métodos que nos permite definir la prioridad en la ejecución se los trabajos generando una secuencia según combinaciones matemáticas establecidas.

En nuestro proyecto se van a construir ordenando los vectores indicadores definidos con anterioridad en este punto. En las siguientes tablas se detallan las combinaciones matemáticas de cada uno de los indicadores.

Además, el orden ha sido definido con dos métodos diferentes, el primero de ellos aplicando algoritmo Johnson, por lo que se precisa de dos vectores indicadores, y el segundo método es la aplicación del resto de ordenaciones (SPT, LPT, Valley, Hill, HI/LOHI, HI/HILO, LO/LOHI, LO/HILO).

DR nº	Indicador 1	Indicador 2
1	$Max\ P_{ij}$	$\frac{\sum at_{ij}}{m_2}$
2	P_{i*j}	$\frac{\sum at_{ij}}{m_2}$
3	$\sum P_{ij} + \frac{\sum at_{ij}}{m_2}$	$\frac{\sum at_{ij}}{m_2}$
4	$\frac{\sum P_{ij}}{m_1}$	$\frac{\sum at_{ij}}{m_2}$

10	$Max\ P_{ij}$	$Max\ P_{ij} + \frac{\sum at_{ij}}{m_2}$
11	$\frac{\sum P_{ij}}{m_1}$	$Max\ P_{ij} + \frac{\sum at_{ij}}{m_2}$
12	P_{i*j}	$Max\ P_{ij} + \frac{\sum at_{ij}}{m_2}$
13	$Max\ P_{ij}$	$\frac{\sum P_{ij}}{m_1} + \frac{\sum at_{ij}}{m_2}$
14	$\frac{\sum P_{ij}}{m_1}$	$\frac{\sum P_{ij}}{m_1} + \frac{\sum at_{ij}}{m_2}$
15	P_{i*j}	$\frac{\sum P_{ij}}{m_1} + \frac{\sum at_{ij}}{m_2}$
16	$Max\ P_{ij}$	$P_{i*j} + \frac{\sum at_{ij}}{m_2}$
17	$\frac{\sum P_{ij}}{m_1}$	$P_{i*j} + \frac{\sum at_{ij}}{m_2}$
18	P_{i*j}	$P_{i*j} + \frac{\sum at_{ij}}{m_2}$

Tabla 4-1: Reglas de despacho con Algoritmo de Johnson *[Fuente: Elaboración propia]*

A continuación, se detallan las ocho reglas de ordenación, que se aplicarán cuando no se realice con el algoritmo de Johnson. En este caso, sólo se le pasa un vector de indicador ya que lo que se realiza ahora es tan sólo una ordenación de un vector.

Ese vector trata de un indicador calculado previamente, o bien de una combinación de varios indicadores. Para calcular esa combinación tan solo hay que sumar dichos vectores por componentes.

En la siguiente tabla se muestra la definición de cada regla de despacho:

DR nº	**Indicador**
5	$\frac{\sum P_{ij}}{m_1}$
6	P_{i*j}
7	$Max\ P_{ij} + \frac{\sum at_{ij}}{m_2}$
8	$\frac{\sum P_{ij}}{m_1} + \frac{\sum at_{ij}}{m_2}$

9	$P_{i*j} + \frac{\sum at_{ij}}{m_2}$
19	$Max\ P_{ij}$
20	$\frac{\sum at_{ij}}{m_2}$
21	$Max\ P_{ij} + \frac{\sum at_{ij}}{m_2}$

Tabla 4.2: Indicadores a ordenar según criterios de ordenación *[Elaboración propia]*

4.3.3 Criterios de ordenación

En primer lugar, se detalla la programación del Algoritmo de Johnson, y a continuación, el resto de los criterios de ordenación empleados para la obtención de las diferentes reglas de despacho:

- Algoritmo Johnson: Se trata de un método exacto para la programación de la producción aplicado históricamente a entorno de *flowshop*. El método consta de los siguientes pasos:

1. Establecer dos vectores que tras la condición de comparación recojan los valores (J1 y J2)
2. Si para un trabajo, la componente del indicador 1 es menos que la del indicador dos para ese trabajo, el valor se añade al vector J1.
3. En caso contrario, se añade al vector J2.
4. Una vez realizado el reparto en ambos vectores se ordenan, J1 de menor a mayor (SPT) y J2 de mayor a menor (LPT).
5. Finalmente, ambos se unen siendo ese el orden de los trabajos a realizar, secuencia de nuestro problema.

- Código Python: Ha sido interesante el trabajo con tuplas, ordenando según la componente [1] de la tupla que es donde se encuentran recogidos los valores de los indicadores mientras que en la posición [0] se encuentra el valor del índice (trabajo).

```
def algo_johnson(ind_1, ind_2):
 j1 = []
 j2 = []
 for pos,value in enumerate(ind_1):
     if value < ind_2[pos]:
         j1.append((pos, value))
     else:
         j2.append((pos, ind_2[pos]))
 j1 = sorted(j1, key = lambda data: data[1])
 j2 = sorted(j2, key = lambda data: data[1], reverse=True)
```

```
    return j1 + j2
```

- Creciente (SPT): Orden creciente (de menor a mayor) de un vector de indicadores.
  - Código Python: Función de Python sorted (automáticamente ordena de menor a mayor) y especifico que ordene vector [1] ya que quiero que ordene por la componente 1 de las tuplas.

```
def creciente(vector):
    return [[index,value] for index, value in sorted(enumerate(vector), key=lambda x: x[1])]
```

(12 21 4 8 9 7) → (4 7 8 9 12 21)

- Decreciente (LPT): Orden decreciente (de mayor a menor) de un vector de indicadores.
  - Código Python: Función de Python sorted + reverse = True (automáticamente ordena de manera inversa a la función sorted explicada anteriormente) y especifico que ordene vector [1] ya que quiero que ordene por la componente 1 de las tuplas.

```
def decreciente(vector):
    return [[index,value] for index, value in sorted(enumerate(vector), reverse=True, key=lambda x: x[1])]
```

(12 21 4 8 9 7) → (21 12 9 8 7 4)

- Colina (Hill): Se compone de una parte creciente, parte desde un mínimo y culmina en un máximo. La segunda parte del vector es decreciente, partiendo del siguiente máximo hasta un mínimo. En primer lugar, se ordena el vector indicador de manera creciente y se va completando la secuencia insertando de manera alternada.
  - Código Python: Se le facilita a la función el vector indicadores calculados previamente, según regla de despacho. Se necesita ordenarlos de menor a mayor, por eso lo primero será llamar a la función creciente (ordena en SPT y genera la tupla). Se ordena, de manera que, desde el inicio hasta la mitad, se ordene de menor a mayor, y luego viceversa. Se divide el vector SPT en valores pares e impares, según posición. Finalmente, a los impares se les da la vuelta con la función de Python reversed para que se ordenen por "LPT".

En este caso, se unen los subvectores en el orden: pares + reversed (impares)

```
def hill(vector):
    sec_spt=creciente(vector)
    pares = []
    impares = []
    for index, j in enumerate(sec_spt):
        if index % 2:
            impares.append(j)
        else:
```

```
            pares.append(j)
    return pares + list(reversed(impares))
```

(12 21 4 8 9 7) → (4 8 12 21 9 7)

- Valle (Valley): Se compone de una parte decreciente, parte desde un máximo y culmina en un mínimo. La segunda parte del vector es ahora creciente, partiendo del siguiente mínimo hasta un máximo, justo al contrario que la ordenación en colina. En primer lugar, se ordena el vector indicador de manera creciente y se va completando la secuencia insertando de manera alternada.
  - Código Python: Se le pasa el vector indicadores calculado. Se necesita ordenarlos de menor a mayor; por eso, lo primero será ejecutar la función creciente (ordena en SPT y genera la tupla). Se ordena de manera que, desde el inicio hasta la mitad, se coloque de mayor a menor y luego viceversa. Se divide el vector SPT en valores pares e impares, según posición. Finalmente, a los impares se les da la vuelta con la función de Python reversed, para que se ordenen por "LPT".

En este caso, se unen los subvectores en el orden: reversed (impares) + pares. Cambia la concatenación de ambos vectores, ordenados de la misma manera que en Hill para obtener el efecto contrario.

```
def valley(vector):
    sec_spt=creciente(vector)
    pares = []
    impares = []
    for index, j in enumerate(sec_spt):
        if index % 2:
            impares.append(j)
        else:
            pares.append(j)
    return list(reversed(impares)) + pares
```

(12 21 4 8 9 7) → (21 9 7 4 8 12)

- HI/HILO: Se obtiene a partir de un vector de indicadores ordenados de manera creciente de modo que, se intercala un valor bajo (el menor) seguido del más bajo. Hasta recorrer todo el vector de indicadores.
  - Código Python: Se ordena, partiendo de un vector ordenado con SPT (se comienza los valores más bajo LOW), por tanto, se llama a la función: creciente. A continuación, se va recorriendo la secuencia SPT de manera alterna guardando el primer valor, luego el último y así se obtiene un vector con valor bajos y altos consecutivos, empezando por el máximo.

```
def hi_hilo(vector):
    sec_spt=creciente(vector)
```

```
    return [x for x in
itertools.chain.from_iterable(itertools.zip_longest(list(reversed(sec_spt)),s
ec_spt)) if x][0:len(sec_spt)]
```

(4 7 8 9 12 21) → (4 21 7 12 8 9)

Secuencia SPT HI/HILO

- HI/LOHI: Se obtiene a partir de un vector de indicadores ordenados de manera decreciente de modo que, se intercala un valor alto seguido del más bajo. Hasta recorrer todo el vector.
  - Código Python: Se ordena partiendo de un vector ordenado con LPT, por tanto, se llama a la función: decreciente. A continuación, se va recorriendo la secuencia inicial LPT de manera alterna guardando el primer valor, luego el último y así se obtiene un vector con valor altos y bajos consecutivos empezando por el máximo.

```
def hi_lohi(vector):
    sec_lpt=decreciente(vector)
    return [x for x in
itertools.chain.from_iterable(itertools.zip_longest(list(reversed(sec_lpt)),
sec_lpt)) if x][0:len(sec_lpt)]
```

(21 12 9 8 7 4) → (21 4 12 7 9 8)

Secuencia LPT HI/LOHI

- LO/HILO: Trata de la inversa a la definida anteriormente como HI/HILO.
  - Código Python: Se llama a la función HI/HILO y con reversed se modifica el orden completo del vector.

```
def lo_hilo(vector):
    sec_spt=creciente(vector)
    return list(reversed(hi_hilo(sec_spt)))
```

(4 21 7 12 8 9) → (9 8 12 7 21 4)

Secuencia HI/HILO Inversa de HI/HILO

- LO/LOHI: Será ahora la inversa a la definida anteriormente como HI/LOHI.
  - Código Python: Se llama a la función HI/LOHI y con reversed, se modifica el orden completo del vector.

```
def lo_lohi(vector):
    sec_lpt=decreciente(vector)
    return list(reversed(hi_lohi(sec_lpt)))
```

(21 4 12 7 9 8) → (8 9 7 12 4 21)

Secuencia HI/LOHI Inversa de HI/LOHI

4.4 Programación de las funciones objetivo

En nuestro problema se van a calcular dos objetivos, a su vez cada uno de ellos con dos reglas de asignación para la segunda etapa. Los objetivos son:

- *Makespan* (C_{max}): Tiempo máximo de finalización del último trabajo.
- *Total Completion Time* ($\sum C_j$): Suma de tiempos de finalización de cada uno de los trabajos.

4.4.1 Con criterio de asignación FAM

La primera función objetivo se hará con regla de asignación FAM, en el paso a la segunda etapa. Se recuerda, que FAM significa primera máquina disponible.

A la función, es necesario proporcionarle una matriz de tiempos de proceso, el número de máquinas en cada etapa y una secuencia inicial que será, en cada caso, la devuelta por cada regla de despacho.

- Etapa 1: Común para ambos casos (FAM y ECT):

Se suman directamente todos los tiempos de proceso de manera ordenada, según secuencia inicial para cada una de las máquinas que pertenecen a la etapa 1, ya que, en nuestro problema, en esta etapa, todos los trabajos deben ser procesados en todas las máquinas.

Ese valor obtenido se guarda en un vector llamado m_{max}, que guarda el valor máximo de ese trabajo en la etapa, ya que hasta que el trabajo no acabe en la última de las máquinas donde se debe procesar, no pasará a la etapa dos.

- Etapa 2:

Mediante un bucle, se compara por orden, la primera máquina en la que el tiempo m_{max} obtenido, en el apartado anterior para ese trabajo, es mayor que el tiempo actual de finalización de esa máquina. De esta manera, solo se asignará si existe el hueco disponible, si no pasará a comparar lo mismo, en la siguiente máquina.

Así, en el caso de llegar a la última de las máquinas y no ha obtenido hueco libre, se asigna a la primera, al ser la de menor índice. Eso se controla con la variable auxiliar *nmach*.

```
def FAM(tp, et_1, et_2, sec):
    m_1 = np.zeros((et_1, len(sec)))
    auxet_1 = []
    for index, order in enumerate(sec):
        auxet_1.append(tp[order,0:et_1])
        if index == 0:
            m_1[:,order] = auxet_1[index]
        else:
            m_1[:,order] = [sum(i) for i in zip(*auxet_1)]
    m_max = np.max(m_1, axis = 0)
    m_2 = np.zeros((et_2, len(sec)))
    at_mach = []
    nmach = 0
    for index, order in enumerate(sec):
```

```
        at_mach.append(tp[order,et_2-1::])
        for nrow, row in enumerate(m_2):
            if m_max[order] >= np.max(row):
                m_2[nmach,order] = at_mach[index][nrow] + m_max[order]
                nmach +=1
                break
        if nmach == et_2: nmach = 0
    cij = np.sum(m_2, axis = 0)
    cmax = np.max(cij)
          sumcij = np.sum(cij)

    return cmax, sumcij
```

Los valores que se devuelven son C_{max} y $\sum C_j$ para esa instancia y regla de despacho.

4.4.2 Con criterio de asignación ECT

La siguiente función objetivo se hará con regla de asignación ECT (menor tiempo de finalización) en el paso a la segunda etapa.

A la función es necesario proporcionarle los mismos datos que a la anterior: matriz de tiempos de proceso, el número de máquinas en cada etapa y una secuencia inicial que será en cada caso la devuelta por cada regla de despacho.

- Etapa 1: Común para ambos casos (FAM y ECT):

- Se suman directamente todos los tiempos de proceso de manera ordenada según secuencia inicial para cada una de las máquinas que pertenecen a la etapa 1 ya que en nuestro problema en esta etapa todos los trabajos deben ser procesados en todas las máquinas.
- Ese valor obtenido se guarda en un vector llamado m_max que guarda el valor máximo de ese trabajo en la etapa ya que hasta que el trabajo no acabe en la última de las máquinas donde se debe procesar no pasara a la etapa dos.

- Etapa 2:

- En este caso se asigna a todas las máquinas, cada una con su tiempo de proceso, y se guarda en un vector auxiliar. Se calcula el mínimo de ese vector de manera que esa sea la máquina a la que se asocie el trabajo.
- Posteriormente se guardará sólo ese trabajo y se restaura el vector auxiliar para realizar el mismo proceso con todos los trabajos.

```
def ECT(tp, et_1, et_2, sec):
    m_1 = np.zeros((et_1, len(sec)))
    auxet_1 = []
    for index, order in enumerate(sec):
        auxet_1.append(tp[order,0:et_1])
        if index == 0:
            m_1[:,order] = auxet_1[index]
        else:
            m_1[:,order] = [sum(i) for i in zip(*auxet_1)]
    m_max = np.max(m_1, axis = 0)
    m_2 = np.zeros((et_2, len(sec)))
    at_mach = []
    nmach=0
    for index, order in enumerate(sec):
```

```
            at_mach.append(tp[order,et_1::])
            aux = []
            for nrow, row in enumerate(m_2):
                if m_max[order] >= np.max(row):
                    aux.append(at_mach[index][nrow] + m_max[order])
                else:
                    aux.append(at_mach[index][nrow] + np.max(row))

                nmach=nmach+1
            m_2[np.argmin(aux), order] = np.min(aux)
            if nmach == et_2: nmach = 0

        cij = np.sum(m_2, axis = 0)
        cmax = np.max(cij)
        sumcij = np.sum(cij)

        return cmax, sumcij
```

Los valores que se devuelven son C_{max} y $\sum C_j$ para esa instancia y regla de despacho concreta. Dichos valores son escritos en cada iteración en el documento de Excel destinado para almacenar todos los resultados obtenidos de cada iteración.

5 ANÁLISIS DE RESULTADOS

"El primer paso es establecer que algo es posible;
Entonces la probabilidad ocurrirá".

- Elon Musk -

En este apartado, se detallan los resultados, de la ejecución del proyecto. Se desarrolla la generación de las instancias, como se calculará el indicador ARPD y por último un análisis de los resultados obtenidos.

5.1 Generación de instancias

Las instancias se generan con combinaciones de datos. Se han generado cinco instancias diferentes para cada combinación. Los tiempos de proceso de la primera etapa han seguido una distribución U [1,100] mientras que en la segunda U [1, et_2*100], siendo et_2 el número de máquinas en la segunda etapa.

Los niveles de los parámetros principales de las combinaciones son:

- Jobs = {50,100,150,200,250,300}
- Máquinas primera etapa = {2,4,6,8}
- Máquinas segunda etapa = {2,3,4}

De esta manera se han generado en total, 360 instancias diferentes, las cuales serán ejecutadas con todas las reglas de despacho programadas. (Nº de instancias = 5 instancias · 6 (rango de jobs) · 4 (rango de máquinas en la et_1) · 3 (rango de máquinas en la et_2).

5.2 Indicador ARPD

La comparación de resultados, con reglas de despacho, se realiza mediante el cálculo de indicador ARPD (Average Relative Percentage Difference) que mide el porcentaje de desviación de cada solución, con respecto al mínimo de todas ellas. En primer lugar, se obtiene dicho valor para todas las instancias, y posteriormente, se calcula la desviación de cada solución mediante la siguiente fórmula:

$$ARPD_{ij} = \frac{Solución\ DR_{ij} - Min(DR_{ij})}{Min(DR_{ij})} \qquad [5.1]$$

A continuación, se calcula para cada regla de despacho, el promedio de los valores de ARPD.

$$\overline{ARPD_i} = \frac{\sum ARPD_{ij}}{Nº\ de\ instancias} \qquad [5.2]$$

Siendo *i*, cada regla de despacho, y *j*, cada instancia ejecutada para esa regla de despacho.

5.3 Análisis de resultados

Tras el cálculo del indicador ARPD, se han realizado dos tablas a modo de resumen agrupando cada promedio obtenido, para cada una de las reglas de despacho programadas. Se han dividido inicialmente según el objetivo a calcular, *total completion time* o *makespan*. Además, en la parte final de este apartado, se realiza también un análisis de las soluciones obtenidas, según el criterio de asignación empleado en la segunda etapa (FAM o ECT).

5.3.1 Resultados para *Makespan* (C_{max})

C_{max}							
DR	APRD	**DR**	APRD	**DR**	APRD	**DR**	APRD
DR1_ECT	0,323%	**DR6_8_ECT**	1,835%	**DR9_3_ECT**	2,160%	**DR19_5_FAM**	2,719%
DR1_FAM	1,222%	**DR6_8_FAM**	2,613%	**DR9_3_FAM**	1,899%	**DR19_6_ECT**	1,471%
DR2_ECT	0,385%	**DR7_1_ECT**	4,795%	**DR9_4_ECT**	3,010%	**DR19_6_FAM**	2,633%
DR2_FAM	1,172%	**DR7_1_FAM**	3,395%	**DR9_4_FAM**	3,279%	**DR19_7_ECT**	1,635%
DR3_ECT	**0,210%**	**DR7_2_ECT**	0,664%	**DR9_5_ECT**	1,482%	**DR19_7_FAM**	2,531%
DR3_FAM	1,093%	**DR7_2_FAM**	1,626%	**DR9_5_FAM**	2,408%	**DR19_8_ECT**	1,764%
DR4_ECT	0,618%	**DR7_3_ECT**	0,692%	**DR9_6_ECT**	1,422%	**DR19_8_FAM**	2,697%
DR4_FAM	1,619%	**DR7_3_FAM**	1,662%	**DR9_6_FAM**	2,400%	**DR20_1_ECT**	6,607%
DR5_1_ECT	1,309%	**DR7_4_ECT**	3,740%	**DR9_7_ECT**	1,813%	**DR20_1_FAM**	3,357%
DR5_1_FAM	2,465%	**DR7_4_FAM**	3,302%	**DR9_7_FAM**	2,693%	**DR20_2_ECT**	**0,210%**
DR5_2_ECT	4,451%	**DR7_5_ECT**	1,315%	**DR9_8_ECT**	1,726%	**DR20_2_FAM**	1,093%
DR5_2_FAM	2,654%	**DR7_5_FAM**	2,571%	**DR9_8_FAM**	2,564%	**DR20_3_ECT**	0,324%
DR5_3_ECT	3,338%	**DR7_6_ECT**	1,293%	**DR10_ECT**	1,566%	**DR20_3_FAM**	1,329%
DR5_3_FAM	2,675%	**DR7_6_FAM**	2,456%	**DR10_FAM**	2,610%	**DR20_4_ECT**	4,803%
DR5_4_ECT	1,491%	**DR7_7_ECT**	1,793%	**DR11_ECT**	1,309%	**DR20_4_FAM**	3,268%
DR5_4_FAM	2,474%	**DR7_7_FAM**	2,558%	**DR11_FAM**	2,465%	**DR20_5_ECT**	1,256%
DR5_5_ECT	1,712%	**DR7_8_ECT**	1,997%	**DR12_ECT**	1,333%	**DR20_5_FAM**	2,542%
DR5_5_FAM	2,542%	**DR7_8_FAM**	2,600%	**DR12_FAM**	2,398%	**DR20_6_ECT**	1,229%
DR5_6_ECT	1,748%	**DR8_1_ECT**	3,819%	**DR13_ECT**	1,203%	**DR20_6_FAM**	2,618%
DR5_6_FAM	2,588%	**DR8_1_FAM**	3,339%	**DR13_FAM**	2,253%	**DR20_7_ECT**	1,794%
DR5_7_ECT	1,758%	**DR8_2_ECT**	0,494%	**DR14_ECT**	1,309%	**DR20_7_FAM**	2,654%
DR5_7_FAM	2,678%	**DR8_2_FAM**	1,407%	**DR14_FAM**	2,465%	**DR20_8_ECT**	1,903%
DR5_8_ECT	1,750%	**DR8_3_ECT**	0,610%	**DR15_ECT**	1,170%	**DR20_8_FAM**	2,656%
DR5_8_FAM	2,644%	**DR8_3_FAM**	1,591%	**DR15_FAM**	2,185%	**DR21_1_ECT**	4,795%
DR6_1_ECT	1,333%	**DR8_4_ECT**	3,333%	**DR16_ECT**	2,568%	**DR21_1_FAM**	3,395%
DR6_1_FAM	2,398%	**DR8_4_FAM**	3,296%	**DR16_FAM**	2,279%	**DR21_2_ECT**	0,664%
DR6_2_ECT	**11,140%**	**DR8_5_ECT**	1,476%	**DR17_ECT**	1,305%	**DR21_2_FAM**	1,626%
DR6_2_FAM	2,483%	**DR8_5_FAM**	2,553%	**DR17_FAM**	2,459%	**DR21_3_ECT**	0,692%
DR6_3_ECT	6,475%	**DR8_6_ECT**	1,487%	**DR18_ECT**	1,333%	**DR21_3_FAM**	1,662%
DR6_3_FAM	2,633%	**DR8_6_FAM**	2,410%	**DR18_FAM**	2,398%	**DR21_4_ECT**	3,740%
DR6_4_ECT	1,365%	**DR8_7_ECT**	1,844%	**DR19_1_ECT**	1,566%	**DR21_4_FAM**	3,302%
DR6_4_FAM	2,629%	**DR8_7_FAM**	2,588%	**DR19_1_FAM**	2,610%	**DR21_5_ECT**	1,315%
DR6_5_ECT	1,629%	**DR8_8_ECT**	1,914%	**DR19_2_ECT**	3,911%	**DR21_5_FAM**	2,571%
DR6_5_FAM	2,648%	**DR8_8_FAM**	2,612%	**DR19_2_FAM**	2,611%	**DR21_6_ECT**	1,293%
DR6_6_ECT	1,646%	**DR9_1_ECT**	3,567%	**DR19_3_ECT**	2,986%	**DR21_6_FAM**	2,456%
DR6_6_FAM	2,560%	**DR9_1_FAM**	3,404%	**DR19_3_FAM**	2,507%	**DR21_7_ECT**	1,793%
DR6_7_ECT	1,736%	**DR9_2_ECT**	3,208%	**DR19_4_ECT**	1,639%	**DR21_7_FAM**	2,558%
DR6_7_FAM	2,623%	**DR9_2_FAM**	1,761%	**DR19_4_FAM**	2,620%	**DR21_8_ECT**	1,997%
				DR19_5_ECT	1,525%	**DR21_8_FAM**	2,600%

Tabla 5-1: Tabla recogida de resultados finales para C_{max} *[Fuente: elaboración propia]*

En la tabla anterior se recogen todos los resultados obtenidos del indicador ARPD para cada regla de despacho. La notación empleada recoge, el número de cada regla de despacho, detalladas en las Tablas 4.1 y 4.2, así como el criterio de asignación empleado en la segunda etapa, FAM o ECT. En el caso de las reglas de despacho obtenidas aplicando los ocho criterios de ordenación, y no el algoritmo Johnson, vienen seguidas de un número entre el uno y el ocho que las distingue, a saber, la correspondencia de cada una:

SPT → DRx_1_x

LPT → DRx_2_x

VALLEY → DRx_3_x

HILL → DRx_4_x

HI/HILO → DRx_5_x

HI/LOHI → DRx_6_x

LO/HILO → DRx_7_x

LO/LOHI → DRx_8_x

En esta tabla se aprecia que los valores de la desviación son muy pequeños, el máximo es tan sólo de 11,140%, mientras que en la tabla que recoge el mejor resultado de *makespan*, estamos hablando de 0,210%. Demuestra que, para este objetivo, los métodos aproximados de reglas de despacho son mejores.

RESULTADOS Cmax	
MÍNIMO	MÁXIMO
0,210%	11,140%
DR3_ECT	**DR6_2_ECT**
DR20_2_ECT	

Tabla 5-2: Mejor y peor regla de despacho aplicada para C_{max} *[Fuente: elaboración propia]*

- Las reglas de despacho (DR) con las que se ha obtenido un mejor valor de la función objetivo y, por tanto, un valor más bajo de ARPD en este caso han sido dos, la 3 (Johnson) y la 20 con LPT.
- La **DR3_ECT** precisa de dos indicadores para aplicar el algoritmo Johnson, son:

$$\sum P_{ij} + \frac{\sum at_{ij}}{m_2} \text{ y } \frac{\sum at_{ij}}{m_2} \qquad [5.5]$$

- Por otro lado, se ha obtenido el mismo valor con **DR20_2_ECT:** $\frac{\sum at_{ij}}{m_2}$ y criterio de ordenación LPT. En este caso, no debe empeorar tanto el resultado la ordenación, ya que lo que se contabiliza, es el tiempo final, no cuando terminen cada uno de los trabajos de manera individual.

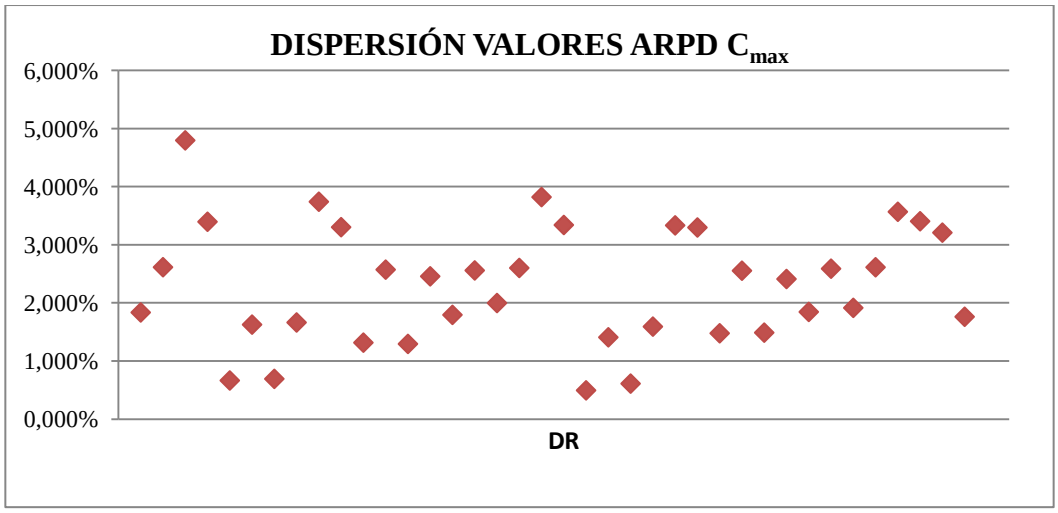
- El valor más alto del indicador ARPD para el objetivo de C_{max} se obtiene con la regla de despacho **DR6_2_ECT**. Es la misma regla de despacho que la que se ha observado más desfavorable también para el objetivo anterior, $\sum C_j$.

Se observa que la regla de despacho que emplea tan sólo los tiempos de proceso de la máquina más cargada no es una buena manera de calcular el orden de ejecución de los trabajos. Aunque ahora el criterio de ordenación no es tan determinante, si lo es el tiempo total de finalización de los trabajos. Por ese motivo, por ejemplo, empleando LPT da la mejor y peor solución, porque no se contabiliza de manera individual cuando finaliza cada uno de los trabajos.

$$P_{i*j} \ con \text{ LPT} \quad [5.6]$$

Hay dos valores mínimos, los cuales muestran que para C_{max} el método de aplicación de las reglas de despacho es bueno, ya que, además, la diferencia total entre ambos es sólo del 11.140%.

Aquí ECT es el mejor resultado en ambos mínimos, si se observan los valores de la Tabla 5-3, además, se comprueba que es así en la mayoría de los casos.



Gráfica 5-1: Tendencia en la dispersión de ARPD C_{max} *[Fuente: elaboración propia]*

En este caso, la dispersión entre los resultados es menor, aun así, aunque los porcentajes hayan disminuido en gran medida, la diferencia entre ellos sigue siendo elevada en algunos casos, pero al ser valores más pequeños, influyen menos en la validez del método.

Si bien es cierto, que, en este objetivo, hay algunos valores que distorsionan el resultado, ya que, como se comentó anteriormente, si se eliminan las pocas soluciones que se salen del rango de valores muy elevados, los resultados serían mucho mejores. Es interesante, excluir para el estudio, las que se alejen mucho del mínimo valor, y tomar por tanto como bueno este método de resolución, mediante reglas de despacho, como válido para el cálculo del objetivo C_{max}.

Las reglas de despacho que no se deben tener en cuenta como válidas, para este método, son las que dan un ARPD más elevado:

DR6_2_ECT	11,140%
DR6_3_ECT	6,475%
DR20_1_ECT	6,607%

La regla de despacho 6, hace referencia a la ecuación, 4.3, que emplea los tiempos de la máquina más cargada. Sin embargo, la otra peor solución es la 20, que corresponde al indicador 4.2, promedio de tiempos de proceso en la segunda etapa.

Según los artículos en los que se ha basado este proyecto, el método sugerido de resolución, mediante reglas de despacho, se propone para el objetivo de *makespan*. Para dicho objetivo ha quedado comprobado que el programa es capaz de proporcionar una solución que minimiza el tiempo máximo de finalización para cualquier número de trabajos y de máquinas en cada etapa.

A continuación, se realiza el cálculo para otro objetivo, *total completion time*, y de esta manera se analizará si el método aproximado empleado es factible para su uso en el cálculo de otros objetivos.

5.3.2 Resultados para *Total Completion Time* ($\sum C_j$)

$\sum C_j$							
DR	APRD	**DR**	APRD	DR	APRD	DR	APRD
DR1_ECT	41,345%	**DR6_8_ECT**	48,975%	**DR9_3_ECT**	54,233%	**DR19_5_FAM**	14,783%
DR1_FAM	15,954%	**DR6_8_FAM**	15,374%	**DR9_3_FAM**	14,712%	**DR19_6_ECT**	44,691%
DR2_ECT	45,728%	**DR7_1_ECT**	38,917%	**DR9_4_ECT**	52,900%	**DR19_6_FAM**	14,139%
DR2_FAM	13,051%	**DR7_1_FAM**	5,246%	**DR9_4_FAM**	16,386%	**DR19_7_ECT**	48,754%
DR3_ECT	55,614%	**DR7_2_ECT**	60,484%	**DR9_5_ECT**	47,790%	**DR19_7_FAM**	15,141%
DR3_FAM	25,526%	**DR7_2_FAM**	26,757%	**DR9_5_FAM**	17,312%	**DR19_8_ECT**	48,673%
DR4_ECT	29,406%	**DR7_3_ECT**	48,844%	**DR9_6_ECT**	47,476%	**DR19_8_FAM**	15,114%
DR4_FAM	8,005%	**DR7_3_FAM**	14,726%	**DR9_6_FAM**	16,886%	**DR20_1_ECT**	46,110%
DR5_1_ECT	27,756%	**DR7_4_ECT**	49,103%	**DR9_7_ECT**	48,812%	**DR20_1_FAM**	5,757%
DR5_1_FAM	6,546%	**DR7_4_FAM**	16,768%	**DR9_7_FAM**	15,371%	**DR20_2_ECT**	55,614%
DR5_2_ECT	70,926%	**DR7_5_ECT**	47,205%	**DR9_8_ECT**	48,582%	**DR20_2_FAM**	25,526%
DR5_2_FAM	19,662%	**DR7_5_FAM**	16,787%	**DR9_8_FAM**	14,525%	**DR20_3_ECT**	50,262%
DR5_3_ECT	48,687%	**DR7_6_ECT**	46,908%	**DR10_ECT**	32,908%	**DR20_3_FAM**	13,956%
DR5_3_FAM	13,581%	**DR7_6_FAM**	16,100%	**DR10_FAM**	8,316%	**DR20_4_ECT**	49,624%
DR5_4_ECT	50,528%	**DR7_7_ECT**	48,638%	**DR11_ECT**	27,756%	**DR20_4_FAM**	17,096%
DR5_4_FAM	12,475%	**DR7_7_FAM**	15,118%	**DR11_FAM**	6,546%	**DR20_5_ECT**	48,674%
DR5_5_ECT	48,079%	**DR7_8_ECT**	48,821%	**DR12_ECT**	44,103%	**DR20_5_FAM**	16,903%
DR5_5_FAM	15,346%	**DR7_8_FAM**	15,350%	**DR12_FAM**	11,998%	**DR20_6_ECT**	48,553%
DR5_6_ECT	47,567%	**DR8_1_ECT**	37,455%	**DR13_ECT**	34,092%	**DR20_6_FAM**	16,580%
DR5_6_FAM	15,443%	**DR8_1_FAM**	**5,223%**	**DR13_FAM**	8,737%	**DR20_7_ECT**	48,471%
DR5_7_ECT	48,627%	**DR8_2_ECT**	61,737%	**DR14_ECT**	27,756%	**DR20_7_FAM**	14,524%
DR5_7_FAM	15,171%	**DR8_2_FAM**	26,400%	**DR14_FAM**	6,546%	**DR20_8_ECT**	48,474%
DR5_8_ECT	48,766%	**DR8_3_ECT**	48,798%	**DR15_ECT**	44,254%	**DR20_8_FAM**	14,493%
DR5_8_FAM	14,720%	**DR8_3_FAM**	14,634%	**DR15_FAM**	11,972%	**DR21_1_ECT**	38,917%
DR6_1_ECT	44,103%	**DR8_4_ECT**	49,181%	**DR16_ECT**	39,715%	**DR21_1_FAM**	5,246%
DR6_1_FAM	11,998%	**DR8_4_FAM**	16,904%	**DR16_FAM**	8,151%	**DR21_2_ECT**	60,484%
DR6_2_ECT	**93,353%**	**DR8_5_ECT**	48,554%	**DR17_ECT**	27,964%	**DR21_2_FAM**	26,757%
DR6_2_FAM	19,061%	**DR8_5_FAM**	16,011%	**DR17_FAM**	6,626%	**DR21_3_ECT**	48,844%
DR6_3_ECT	58,022%	**DR8_6_ECT**	48,280%	**DR18_ECT**	44,103%	**DR21_3_FAM**	14,726%
DR6_3_FAM	11,712%	**DR8_6_FAM**	16,024%	**DR18_FAM**	11,998%	**DR21_4_ECT**	49,103%
DR6_4_ECT	58,080%	**DR8_7_ECT**	48,730%	**DR19_1_ECT**	32,908%	**DR21_4_FAM**	16,768%
DR6_4_FAM	12,379%	**DR8_7_FAM**	14,847%	**DR19_1_FAM**	8,316%	**DR21_5_ECT**	47,205%
DR6_5_ECT	47,279%	**DR8_8_ECT**	48,827%	**DR19_2_ECT**	65,153%	**DR21_5_FAM**	16,787%
DR6_5_FAM	18,016%	**DR8_8_FAM**	14,346%	**DR19_2_FAM**	19,710%	**DR21_6_ECT**	46,908%
DR6_6_ECT	46,932%	**DR9_1_ECT**	41,867%	**DR19_3_ECT**	48,127%	**DR21_6_FAM**	16,100%
DR6_6_FAM	16,635%	**DR9_1_FAM**	7,309%	**DR19_3_FAM**	13,963%	**DR21_7_ECT**	48,638%
DR6_7_ECT	48,997%	**DR9_2_ECT**	79,827%	**DR19_4_ECT**	50,514%	**DR21_7_FAM**	15,118%
DR6_7_FAM	14,829%	**DR9_2_FAM**	26,565%	**DR19_4_FAM**	14,072%	**DR21_8_ECT**	48,821%
				DR19_5_ECT	45,060%	**DR21_8_FAM**	15,350%

Tabla 5-3: Tabla recogida de resultados finales para $\sum C_j$ *[Fuente: elaboración propia]*

En la Tabla 5-4 se observan todos los resultados obtenidos, para el objetivo de *total completion time*, tanto para el criterio de asignación FAM como ECT. La notación es la misma que la explicada en el apartado anterior para el objetivo de *makespan*.

Se observa, de manera general, que los resultados más favorables, se obtienen, cuando se emplea el criterio de asignación FAM en la segunda etapa.

Aun así, en la mayoría de los casos, los resultados son muy elevados y dispares para este objetivo. Según los datos obtenidos, se demuestra que, para el objetivo de *total completion time*, el método de las reglas de despacho no es una buena manera de resolver el problema.

RESULTADOS $\sum C_j$	
MÍNIMO	MÁXIMO
5,223%	93,353%
DR8_1_FAM	**DR6_2_ECT**

Tabla 5-4: Mejor y peor regla de despacho aplicada para $\sum C_j$ *[Fuente: elaboración propia]*

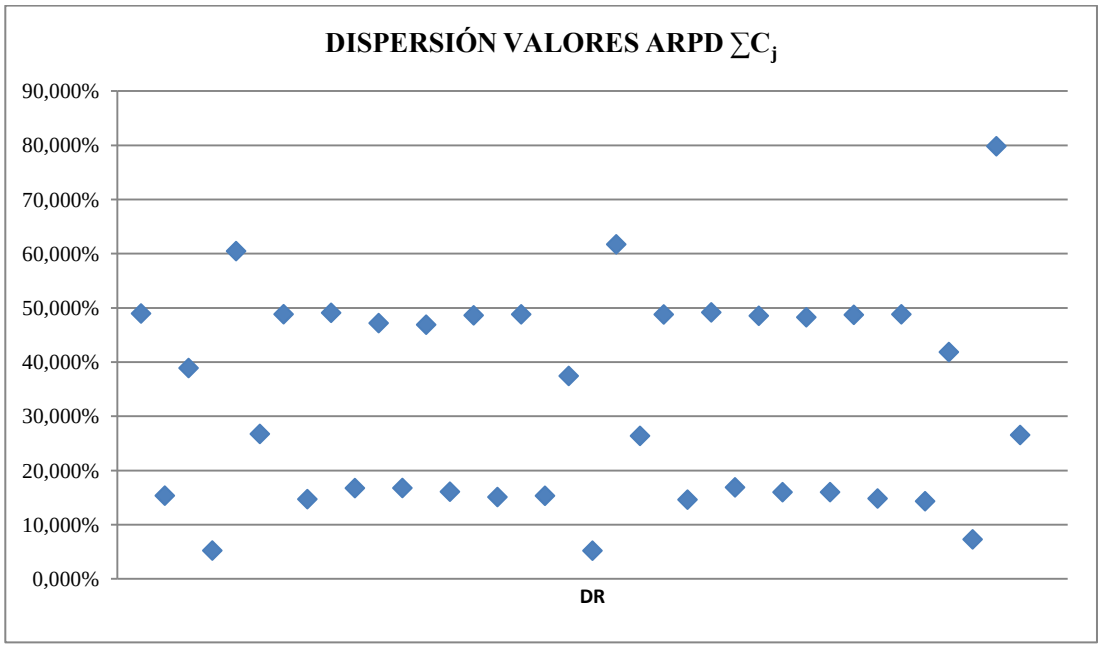
- La mejor solución la proporciona la regla de despacho (**DR8_1_FAM**) es la que combina el indicador ocho, con el primer criterio de ordenación (SPT). Ordena de menor a mayor, tiene sentido que esta regla de despacho proporcione un buen valor ya que tiene en $\sum C_j$ afecta el tiempo de finalización de cada uno de los trabajos, no sólo el total. Es por ello, por lo que al ordenar de menor a mayor los tiempos, serán más reducidos generalmente.

$$\frac{\sum \mathrm{P_{ij}}}{\mathrm{m_1}} + \frac{\sum at_{ij}}{m_2} \quad \text{con SPT [5.3]}$$

- Análisis de la peor regla de despacho (**DR6_2_ECT)**: Es la regla de despacho que más *total completion time* obtiene, ya que utiliza, los tiempos de proceso de la máquina más cargada y además los ordena por LPT (de mayor a menor). Dicha ordenación hace que el tiempo se vaya acumulando, obteniendo por tanto el peor valor para $\sum C_j$. También se observa, que, en el resto de los resultados, el criterio de ordenación 2 (LPT) y con asignación (ECT) obtiene siempre los valores más altos de sus combinaciones.

$$\mathrm{P_{i*j}} \text{ con LPT [5.4]}$$

A modo de resumen, en la Tabla anterior, 5-2, se hace referencia a la mejor y peor solución obtenida. La diferencia entre ellas es muy elevada, lo que resta fiabilidad al método de resolución. Además, se observa que la mejor solución es para FAM y la peor para ECT. Eso mismo es lo que sucede en el resto de las soluciones ya que se observa que para FAM, en la mayoría de los casos, los valores son menores que para ECT.



Gráfica 5-2: Tendencia en la dispersión de ARPD $\sum C_j$ *[Fuente: elaboración propia]*

Se ha representado la dispersión de los valores, para una muestra de resultados en la Gráfica anterior, 5-1. Se puede observar que es muy elevada, ya que los valores, además de ser un porcentaje muy alto, entre ellos son muy diferentes. Aún eliminando los casos extremos, la diferencia entre ellos sigue siendo muy elevada.

Tras analizar los resultados obtenidos se llega a la conclusión de que, para este objetivo, el método de resolución aplicado carece de fiabilidad. Se demuestra tanto en el elevado valor del indicador ARPD como en que las soluciones con FAM proporcionan un valor menor a las de ECT, ese hecho no tiene sentido de manera teórica puesto que, al minimizar los tiempos de cada trabajo, el tiempo final también sería menor.

Tal y como se comentó en la metodología, la aplicación de reglas de despacho era un método aproximado sugerido para el cálculo del *makespan*, y la aplicación de dichas reglas para la minimizar el objetivo $\sum C_j$ no devuelve buenos resultados, por lo que sería recomendable diseñar otras reglas de despacho

5.3.3 Comparación según criterio de asignación

A continuación, se comparan los resultados obtenidos, según el criterio de asignación, estudiando un caso concreto de regla de despacho. En ella, se recogen los valores del indicador ARPD de la regla de despacho 8, según los criterios de ordenación SPT, LPT, VALLEY, HILL, HI/HILO, HI/LOHI, LO/HILO, LO/LOHI. Se ha elegido la regla de despacho 8, porque trabaja con tiempos de proceso en ambas máquinas, y a la hora de realizar el análisis, puede ser más interesante que otra, que solo tenga en cuenta el de una de las dos, además al ser el promedio de ambas va a ser muy objetivo.

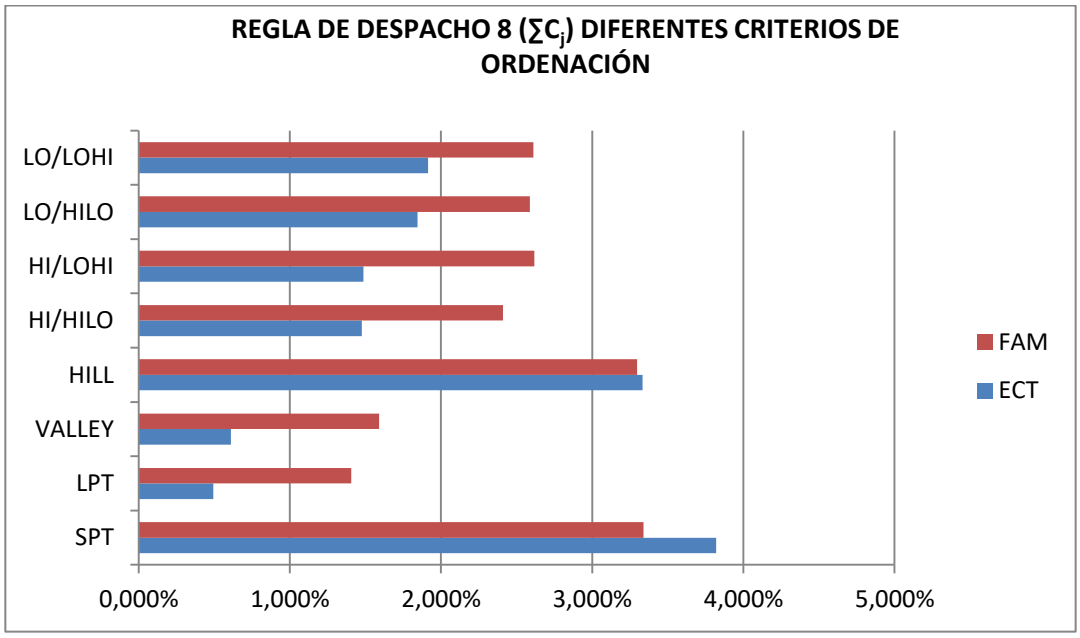
A continuación, se realiza dicho estudio para *makespan* (C_{max}), ya que en el apartado anterior se ha llegado a la conclusión de que para el objetivo de *total completion time*, el cálculo de la función objetivo mediante el método de reglas de despacho no era eficiente.

C_{max}		
DR_8	**ECT**	**FAM**
SPT	3,819%	3,339%
LPT	0,494%	1,407%
VALLEY	0,610%	1,591%
HILL	3,333%	3,296%
HI/HILO	1,476%	2,410%
HI/LOHI	1,487%	2,618%
LO/HILO	1,844%	2,588%
LO/LOHI	1,914%	2,612%

Tabla 5-5: Resultados APRD para la DR 8 dividido en ECT y FAM *[Fuente: elaboración propia]*

En este caso, no está tan claro como para el caso anterior, ya que, las diferencias son menores, aunque es mejor para ECT, en la mayoría de los casos. Al tratarse de un objetivo global, ya que, se trata, de minimizar el tiempo máximo de finalización de todos los trabajos, tiene sentido, que la asignación ECT de mejor resultado, puesto que, en todos los trabajos, va a asignarse a la máquina donde menos tarde en procesarse, y por ello, el tiempo final será menor.

Se muestra a continuación los resultados analizados en forma de gráfico de barras, para una mejor visualización de la solución.



Gráfica 5-3: ARPD (FAM y ECT) para los criterios de ordenación en DR 8 y C_{max} *[Fuente: elaboración propia]*

6 CONCLUSIONES

En este apartado se detallan las principales conclusiones obtenidas, tras la realización de este proyecto, así como futuras mejoras aplicables al problema inicial planteado, para que se asemeje más a la realidad de la producción en un proceso de ensamblado.

Las conclusiones han estado basadas en los resultados obtenidos, los cuales han sido desarrollados en el apartado anterior, "5. Análisis de resultados". Dichos valores, han sido calculados, tras ejecutar el programa, desarrollado en lenguaje Python, el cual se puede consultar en el apartado de Anexos, junto al conjunto de instancias ejecutadas.

6.1 Conclusiones

El objetivo de este proyecto ha sido la resolución de un problema de ensamblado, dividido en dos etapas, mediante la aplicación de reglas de despacho. Tras analizar todos los resultados, para cada uno de los objetivos estudiados, *total completion time* y *makespan*. De esta manera, se ha podido medir la eficiencia de la aplicación de dicho método al problema.

Tras ejecutar las 360 instancias, para ambos objetivos, y según los dos criterios de asignación diferentes en la segunda etapa, FAM o ECT. Se ha llegado a la conclusión principal, de que el método, es eficaz en el objetivo de C_{max}, tal y como lo sugerían los artículos consultados inicialmente y comentados en el apartado de metodología. Dicha eficiencia ha sido medida con el cálculo del indicador ARPD, el cual mide la desviación entre las soluciones obtenidas y el mínimo de éstas.

Los valores obtenidos en general, para ambos objetivos, son muy diferentes. Esto es debido, a que el número de trabajos es muy elevado, y en el objetivo $\sum C_j$ se suman todos los tiempos de finalización. El error, por tanto, en dichos casos aumenta, al aumentar el orden de los resultados del problema. Es una de las principales causas, por la que el indicador ARPD, obtiene unos valores tan elevados para dicho objetivo. Nuestro problema, va a poder ser estudiado correctamente, por tanto, con el objetivo de C_{max} lo cual hace que tengamos un control de la producción de manera global y no individual de cada trabajo, ya que el objetivo para el que mejor funciona el método programado es para el C_{max}.

Debido a la alta dispersión del valor de ARPD para $\sum C_j$, los resultados no proporcionan una información que evidencie la mejor programación para dicho objetivo. Sin embargo, sí se obtiene una solución, que resulta concluyente en el caso del objetivo C_{max}. Quiere decir, que no se ha obtenido una solución concluyente para cada trabajo de manera individual, pero sí de la producción global.

El final de este proyecto por tanto culmina con la obtención de un método de resolución factible y eficaz, que proporciona de manera corroborada, una mejor solución para nuestro problema de ensamblado con el objetivo de *makespan* (C_{max}) y es, programar las operaciones, aplicando la regla de despacho que ordene los trabajos según la DR 3 y que se siga el criterio de asignación ECT en la segunda etapa. De esta manera, los tiempos de producción serán mínimos, y, por tanto, los costes de producción de la empresa reducidos de manera general.

6.2 Mejoras futuras del proyecto

En cuanto a las posibles mejoras aplicables al proyecto en un futuro, se encuentra principalmente el estudio para otros objetivos. El proyecto, tan sólo se ha centrado, en la minimización de tiempos de finalización, tanto en suma de cada trabajo, como en el máximo de todos ellos. Al introducir el

cálculo de otros objetivos, se pueden aplicar otras reglas de despacho diferente en función de los objetivos, lo cual otorga mayor generalización del problema.

Bien es cierto, que, en la mayoría de los casos, al reducir los tiempos de finalizar los trabajos, se ven reducidos los costes de manera general. Es por eso, por lo que los principales objetivos que se estudian son los relacionados con dichos tiempos.

Se propone, por tanto, la continuación de este estudio, con diferentes objetivos relacionados con fechas y tiempos (fechas de entrega, fechas de llegada de material, tiempos de set-up o bien buffers distintos de infinito). De esta manera, el problema será mucho más asemejado a la realidad, ya que en las empresas todos esos factores son condicionantes.

Por otro lado, al no haber obtenido resultados concluyentes para el objetivo de *total completion time*, se sugiere el cálculo de dichas soluciones mediante otros métodos de resolución diferentes a las reglas de despacho elegidas.

7 REFERENCIAS

Framinan et al. (2019): *Deterministic assembly scheduling problems: A review and classification of concurrent-type scheduling models and solution procedures*. European Journal of Operational Research. Volume 273, Issue 2, Pages 401-417. https://www.sciencedirect.com/science/article/pii/S0377221718303369?via%3Dihub

Navaei et al. (2013): *Two-stage flow-shop scheduling problem with non-identical second stage assembly machines*. The International Journal of Advanced Manufacturing Technology volume 69, pages 2215–2226. https://link.springer.com/article/10.1007/s00170-013-5187-3

Komaki & Kayvanfar (2015): *Optimizer algorithm for the 2-stage assembly flowshop scheduling problem.* Journal of Computational Science. Volume 8, Pages 109-120. https://www.sciencedirect.com/science/article/abs/pii/S1877750315000381

Graham et al. (1979): *Optimization and Approximation in deterministic Sequencing and Scheduling.* Annals of Discrete Mathematics. Volume 5, Pages 287-326. https://www.sciencedirect.com/science/article/abs/pii/S016750600870356X

Lee, Cheng, Lin (1993): *Minimizing the makespan in the 3-machine assembly-type flowshop scheduling problem.* Management Science. Vol. 39, No. 5, pp. 616-625 (10 pages). https://www.jstor.org/stable/2632599

Lin, Cheng, Chou (2006): *Scheduling in an assembly-type production chain with batch transfer*. Omega, 35*(2), 143-151*.

Jatinder N. D. Gupta (1988): *Two-Stage, Hybrid Flowshop Scheduling Problem.* The Journal of the Operational Research Society. Vol. 39, No. 4, pp. 359-364 (6 pages) https://www.jstor.org/stable/2582115.

Yonglin Li y Zhenhua Dai (2020*): A two-stage flow-shop scheduling problem with incompatible job families and limited waiting time*. Engineering Optimization Volume 52, Issue 3. https://www.tandfonline.com/doi/abs/10.1080/0305215X.2019.1593974?journalCode=geno20

8 ANEXOS

En este apartado de Anexos se refleja el enlace de acceso a la carpeta donde es posible consultar:

- El código .py al completo.
- Excel con todos los resultados obtenidos y en las gráficas que se ha basado el análisis de resultados. Obtención de gráficas representativas.
- Los ficheros con las trescientas sesenta instancias ejecutadas.

https://drive.google.com/drive/folders/1AnslFYRd1_PY7SqjrngW_1ebnNW-wSi7?usp=sharing