Universidad de Valladolid

UNIVERSIDAD DE VALLADOLID

ESCUELA DE INGENIERIAS INDUSTRIALES

GRADO EN INGENIERÍA EN ORGANIZACIÓN INDUSTRIAL

HEURÍSTICAS CONSTRUCTIVAS PARA LA PROGRAMACIÓN DE TAREAS EN TALLERES FLEXIBLES.

Autor:

Fraile Pérez, Javier

Tutor:

Araúzo Araúzo, J. Alberto

Dpto. de Organización de Empresas y C.I.M.

Valladolid, mayo de 2024

Quiero agradecer, en primer lugar, a mi tutor, por el apoyo brindado. En segundo lugar, a mi familia, en especial a mis padres y abuelos, que siempre han creído en mí. Sin olvidarme de mi peludo Curro, por la compañía que me ha hecho en estos meses de trabajo. Por último, a mi grupo de amigos, que siempre están ahí cuando les necesitas.

Resumen:

La programación de operaciones en talleres flexibles es un problema de gran interés industrial. Minimizar el tiempo de procesamientos de los trabajos juega un papel clave. Existen numerosas técnicas para realizar esta tarea, entre las que se pueden destacar los métodos constructivos y las metaheurísticas.

Para afrontar el problema del Job Shop Flexible, los algoritmos encontrados en la literatura lo resuelven en dos pasos: el de asignación de las máquinas a las operaciones y el de su posterior temporización; lo que se denomina planificación separada. Se quiere resolver el problema unificando estos dos pasos en uno solo, denominando esta forma de resolución como planificación conjunta.

Se ha diseñado un algoritmo para afrontar el desarrollo de la programación de tareas en talleres flexibles utilizando para ello métodos constructivos combinando diversos tipos de planificación, esquemas de construcción y heurísticas.

Posteriormente se realiza una comparación exhaustiva con los resultados obtenidos entre las diferentes combinaciones mencionadas para hallar la mejor combinación posible.

Palabras clave:

Programación de operaciones, Job Shop Flexible, Métodos constructivos, Heurísticas, Dirección de operaciones.

Abstract:

Programming operations in flexible workshops is a problem of great industrial interest. Minimizing job processing time is a key role. There are several techniques to undertake this task, including constructive methods and metaheuristics can be highlighted.

To address the Flexible Job Shop problem, algorithms found in the literature solve it in two steps: machine assignment to operations and subsequent scheduling; known as separate scheduling. The aim is to solve the problem by unifying these two steps into one, calling this form of resolution joint scheduling.

An algorithm has been designed to tackle the development of task scheduling in flexible workshops using constructive methods with diferents types of planning, construction schemes, and heuristics.

Subsequently, a comprehensive comparison can be made among the the results obtained from the different mentioned combinations in order to find the best possible option.

Key words:

Scheduling, Flexible Job Shop, Constructive methods, Heuristics, Operations Management.

ÍNDICE

Índice de ilustraciones

Índice de tablas

1 INTRODUCCIÓN

1.1 Antecedentes y justificación

La gestión eficiente de talleres en el contexto de la dirección de operaciones es un desafío crítico en numerosas industrias. La complejidad aumenta considerablemente cuando nos adentramos en el ámbito del *Job Shop Flexible*, donde la asignación de recursos a tareas específicas debe ser adaptable y flexible. A pesar de los avances tecnológicos, la optimización de la planificación en talleres sigue siendo un campo de investigación activo y relevante en la actualidad. La necesidad de encontrar soluciones efectivas para la asignación de recursos, la minimización de tiempos de ejecución y la maximización de la eficiencia operativa sigue siendo una prioridad para muchas organizaciones.

Además, en un entorno empresarial cada vez más dinámico y competitivo, la capacidad de adaptarse rápidamente a cambios en la demanda del mercado y en las condiciones operativas es esencial para mantener la ventaja competitiva. En este sentido, la gestión eficiente de talleres, especialmente en entornos de *Job Shop Flexible*, se convierte en un factor crítico para el éxito empresarial.

La investigación en este campo no solo contribuye a mejorar la eficiencia operativa de las organizaciones, sino que también tiene un impacto significativo en aspectos como la satisfacción del cliente, la calidad del producto y la optimización de recursos. Al abordar los desafíos específicos asociados con la asignación dinámica de recursos en talleres, se pueden identificar oportunidades para mejorar los procesos, reducir costos y aumentar la productividad.

Se ha observado que existen numerosas investigaciones que han explorado y propuesto soluciones utilizando la planificación separada de la producción, destacando la eficacia de estos métodos en ciertos contextos. Sin embargo, se ha notado una falta de artículos de relevancia que aborden específicamente la planificación conjunta en el contexto del *Job Shop Flexible*. Por lo tanto, este estudio se propone llenar este vacío en la literatura académica al investigar y evaluar críticamente la viabilidad y eficacia de la planificación conjunta en comparación con la planificación separada, proporcionando así una visión más completa y equilibrada de los dos enfoques de planificación disponibles para abordar este problema.

1.2 Objetivos

El propósito principal de este proyecto radica en la exploración y comparación de algoritmos de planificación en el ámbito del Job Shop Flexible. Esta labor reviste una importancia fundamental debido a la creciente complejidad de los entornos industriales modernos. En estos entornos, la optimización en la asignación de recursos y la gestión eficiente de los talleres son aspectos cruciales para preservar la competitividad y la rentabilidad de las organizaciones.

La comparación se centra en dos enfoques de planificación dentro del entorno flexible: la planificación conjunta, que implica la secuenciación de operaciones a medida que son asignadas a máquinas, y la planificación separada, que inicialmente asigna cada operación a una sola máquina para luego secuenciarlas.

Además, se proponen diversas heurísticas y estrategias de asignación de operaciones a máquinas, lo que permite realizar una comparación detallada y un análisis contrastado de los resultados obtenidos. Este enfoque integral proporciona una visión completa y profunda de las diferentes metodologías de planificación y su impacto en la eficiencia y productividad de los procesos industriales.

1.3 Contenido de la memoria

En la sección de "Programación de Operaciones", se abordan los fundamentos teóricos de la programación de operaciones, con énfasis en el problema del Job Shop Scheduling (JSP) y haciendo una introducción al Flexible Job Shop Scheduling Problem (FJSP). A continuación, en "Métodos FJSP", se revisan los enfoques de planificación, incluyendo los esquemas de Giffler y Thompson, así como heurísticas simples, ofreciendo una perspectiva de métodos desarrollados por otros autores relevantes en este campo.

Posteriormente, en el apartado de "Lenguaje de Programación", se explica la elección del lenguaje utilizado para implementar los algoritmos, destacando sus ventajas para abordar los requisitos técnicos del proyecto. Luego, en "Programa", se detalla la implementación del algoritmo diseñado, su estructura, funcionamiento y las consideraciones técnicas relevantes que fueron tenidas en cuenta durante su desarrollo.

En el siguiente apartado de "Test" se presentan los resultados de las pruebas realizadas, analizando tanto la eficacia como la eficiencia de los enfoques de planificación evaluados. Continuando con el estudio, en la sección de estudio económico se recogen todos los datos económicos del proyecto, desagregando los costes totales involucrados en el desarrollo e implementación de los métodos de planificación.

Finalmente, en "Conclusiones" se resumen los hallazgos clave del estudio, identificando las limitaciones encontradas y proponiendo posibles direcciones futuras de investigación en el ámbito de la planificación en el Job Shop Flexible.

2 PROGRAMACIÓN DE OPERACIONES

2.1 Introducción a la programación de operaciones

La programación de operaciones es una parte muy importante de la gestión de operaciones, en la cual se planifican y organizan una serie de actividades que requieren de un conjunto de recursos limitados para poder llevarse a cabo. La programación de operaciones trata de producir bienes o servicios dentro de una empresa de la manera más eficaz posible. Para ello asigna estos recursos, como, por ejemplo, mano de obra o maquinaria, a los pedidos que han de realizarse dentro de una empresa a lo largo del tiempo.

La programación de operaciones desempeña un papel fundamental en la optimización de los procesos de fabricación. Proporciona una visión clara y detallada del sistema de producción, lo que resulta crucial a la hora de tomar decisiones; así como flexibilidad para adaptar el programa a las condiciones cambiantes, anticipando posibles variaciones en los elementos críticos. Asimismo, actúa como una herramienta de control, ya que el programa sirve como referencia para evaluar el rendimiento real en comparación con el planificado, lo que subraya la importancia de desarrollar programas realistas y precisos.

Sin embargo, la programación de operaciones conlleva grandes desafíos debido a la complejidad implícita en los cálculos que involucra. Los problemas de programación de operaciones suelen ser de naturaleza combinatoria[1], lo que dificulta la búsqueda de soluciones óptimas para problemas de gran escala en un tiempo razonable. La asignación de múltiples tareas a varios recursos puede conllevar dependencias complicadas entre ellos. El gran número de restricciones, ya sean de carácter físico o

[1] Tipo de problema en el que la resolución implica la búsqueda de una combinación óptima, o la selección de una combinación específica, de entre un conjunto finito de elementos.

temporal, representa otro desafío importante en la programación de operaciones. Estas restricciones pueden variar desde limitaciones funcionales de las máquinas y herramientas, hasta restricciones de tiempo, como fechas de finalización y duraciones específicas de operaciones.

En el entorno de fabricación, la programación de operaciones se enfoca en la asignación efectiva de recursos, como máquinas, herramientas y personal, a las tareas específicas, estableciendo fechas de inicio para lograr un funcionamiento óptimo del sistema de producción. Se sitúa en el nivel más bajo de planificación, correspondiente a la planificación a corto plazo, y actúa como una etapa preparatoria crucial en el proceso de planificación de la producción en una empresa. El resultado final de este proceso es la asignación eficiente de recursos y la creación de un calendario de actividades que guía la ejecución de las tareas de fabricación (Villahoz, 2013).

Existen gran número de herramientas asociadas a esta metodología, la cual nos permite llevar a cabo la gestión de operaciones de una manera óptima y ordenada. A lo largo de este capítulo se mostrarán conceptos a cerca de diagramas de *Gantt* y teoría de grafos reticulares. Estos nos permiten obtener de forma gráfica, la secuenciación de la operaciones o tareas de manera rápida. La forma de generar estos diagramas es muy amplia, y ha ido mejorando constantemente a lo largo de los años, desde los primeros diagramas de *Gantt* inventados por Henry *Gantt* a finales del siglo XIX.

2.2 Conceptos básicos

2.2.1 Planificación

La planificación es la actividad que consiste en la asignación de unos recursos limitados para alcanzar un objetivo final. Es un proceso principalmente analítico.

La propia palabra “planificación” nos revela que es una actividad que debe realizarse en primer lugar, antes de realizar cualquier otro tipo de tarea. De esta manera reduciremos riesgos y con gran seguridad, mejoraremos los resultados finalmente obtenidos.

2.2.2 Programación

La programación se refiere al proceso de determinar la secuencia y asignación de recursos para llevar a cabo las diferentes actividades y tareas necesarias para lograr los objetivos operativos de una organización.

Este proceso implica tomar decisiones sobre qué actividades se realizarán, cuándo se llevarán a cabo y qué recursos específicos se asignarán a cada actividad.

La programación es la herramienta ejecutora de la planificación, es decir, mientras la planificación establece los objetivos, metas y estrategias generales, la programación se encarga de implementar esas estrategias en acciones concretas y detalladas.

2.2.3 Niveles de la planificación

Existen tres niveles de decisión en la planificación de operaciones, cada uno de ellos con sus propias responsabilidades y enfoques:

Nivel estratégico: se toman decisiones de alto nivel que afectan a toda la organización. Se centra en la planificación a largo plazo y en la formulación de objetivos y estrategias globales. Normalmente está asociado a juntas directivas de la organización.

Nivel táctico: se traducen las estrategias y objetivos generales en planes y acciones concretas. A este nivel encontramos el desarrollo de planes de operaciones, gestión de materiales y de stocks, programación maestra de las operaciones.

Nivel operativo: se ejecutan las tareas diarias o el corto plazo y se supervisan las actividades cotidianas. A este nivel se encuentra la gestión de inventarios y otras actividades que aseguran que las operaciones de la organización funcionen de manera eficiente y efectiva.

2.3 Tipos de procesos o configuraciones productivas

Existen variaciones respecto a la configuración de los distintos tipos de procesos. La primera de todas fue propuesta por Woodward en el año 1965. En esta se distinguían dos clasificaciones: entre la producción entre pequeños y grandes lotes; y entre procesos continuos y la producción en serie. El problema de esta clasificación radica en la subjetividad a la hora de etiquetar a un lote como “pequeño” o “grande”.

La clasificación de procesos productivos, en función de la continuidad en la obtención del producto que se elige, es la siguiente (Machuca, 1998):

Por proyectos: cuando se obtiene un solo o pocos productos derivados de un largo periodo de fabricación. Como por ejemplo la construcción de un crucero.

Por lotes: cuando se obtienen productos diferentes en las mismas instalaciones. Se lleva a cabo en diversos sectores industriales, los más comunes son el manufacturero, el alimenticio, el farmacéutico o el metalúrgico, entre otros. Por ejemplo, una fábrica de engranajes, en la cual existen diferentes modelos.

Continua: se obtiene un único producto en la misma instalación. Como, por ejemplo, la producción de papel.

2.3.1 Configuración productiva por proyectos

La configuración por proyectos suele estar compuesta de gran cantidad de *inputs*[2]. Normalmente son trasladados al lugar donde se elabora el producto o genera el servicio. Todas las actividades que conforman el proyecto se controlan conjuntamente por un equipo de coordinación, atendiendo de forma especial a la duración total del proyecto y los recursos necesarios. Esto establece unas relaciones de precedencia entre las distintas tareas y una asignación de recursos a cada una de estas tareas que tiene en cuenta las duraciones parciales de estas, los costes de los retrasos, etc. La función principal de la persona que lleva a cabo la gestión productiva de un proyecto se puede resumir en la coordinación de un gran número de actividades interrelacionadas entre sí, de forma que se minimice el coste de los recursos y el tiempo total del proyecto.

2.3.2 Configuración productiva por lotes

La característica más identificativa de esta configuración es que utiliza las mismas instalaciones para la obtención de productos diferenciados. Esta característica es compartida por todos los tipos de configuración por lotes que veremos a continuación, pero en función del tamaño de lotes, la homogeneidad de los productos fabricados o las características de los procesos llevados a cabo se pueden definir dos distinciones:

2.3.2.1 Configuraciones Job Shop

En este tipo de configuración, se encuentran una serie de lotes de pequeño tamaño y que difieren en gran medida del resto de lotes. Por lo que en ocasiones son producidos a medida y carecen o tienen muy poca estandarización. Al no haber máquinas muy especializadas, los costes variables suelen tender a ser elevados; mientras que los costes fijos, en general, son bajos.

La disposición de las plantas dedicadas a este tipo de producción suele estar agrupadas en talleres o centros de trabajo. Estos centros suelen ser versátiles, por lo que en un mismo centro pueden llevarse a cabo operaciones distintas. En una planta industrial, pueden estar produciéndose en un mismo instante de tiempo, operaciones pertenecientes a la fabricación de lotes distintos, cada uno de ellos de un tamaño. Como

[2] son los elementos fundamentales que se requieren para planificar, desarrollar y gestionar proyectos específicos.

cada lote puede encontrarse en diferentes etapas del proceso, resulta difícil utilizar un calendario fijo para la secuenciación de operaciones que se adapte a las necesidades.

La programación de operaciones juega un papel crucial en la configuración del *Job Shop*. Dentro de este tipo de configuración, puede haber dos distinciones:

2.3.2.1.1 Configuración a medida o de talleres

La obtención del producto final requiere de un número reducido de operaciones poco especializadas. Por este motivo pueden ser realizadas por el mismo trabajador o un grupo de trabajadores que se encargan del proceso productivo al completo; empleando los centros de trabajos pertinentes en su proceso. La cantidad de productos finales que se pueden obtener es casi infinita, únicamente limitada por las capacidades productivas de la empresa y lo innovadores que sean los clientes.

Es frecuente que el tipo de proceso a medida tenga muy baja sofisticación tecnológica y la automatización del proceso productivo sea muy baja o nula. Lo cual permite adaptarse a los requerimientos de los clientes.

2.3.2.1.2 Configuración en *batch*

En esta variante del tipo de configuración de un *Job Shop*, la obtención del producto final requiere de un mayor número de operaciones y, además, son más especializadas. Por lo que resulta complicado al operario o grupo de operarios dominar todo el proceso productivo del pedido. Es frecuente que uno o varios operarios se especialicen en las operaciones dentro de un centro de trabajo en concreto.

Como cada operario se encuentra en un solo centro de trabajo, cuando una operación es terminada y el producto semiterminado necesita ser tratado en otro centro de trabajo, se utilizan almacenes intermedios en los que el producto espera a que el próximo centro de trabajo en el que necesita ser tratado, quede libre.

En menor medida que la configuración a medida o de talleres, la configuración *batch* sigue siendo muy flexible, aunque ya no es “a medida”. Sigue habiendo una baja repetitividad de las operaciones y el tamaño de los lotes suele ser mayor.

En ambos casos la esencia es la misma, el problema de la programación de operaciones**.** Un pedido necesita ser tratado en diferentes centros de trabajo por distintas operaciones (ya sean ejecutadas por los mismos o distintos trabajadores). En los centros de trabajo puede haber colas de espera, en caso de que un pedido llegue a un centro de trabajo y este esté ocupado por otro. Puede haber tiempos ociosos, en caso de que el tratamiento del pedido dentro del centro termine antes de que llegue otro. Ambas situaciones son problemáticas y aumentan los costos.

En la configuración a medida será común la infrautilización de los centros de trabajo. Siendo más fácil cumplir con las fechas de entrega. En el caso de la configuración *batch*, los procesos de obtención tienen una mayor complejidad, debido principalmente al elevado número de operaciones y centros de trabajo; aumentando así la dimensión del problema.

A la hora de resolver estos problemas en este trabajo, no haremos distinción entre los dos tipos de configuración mencionados. Trataremos el problema conjunto y lo denominaremos configuración *Job Shop*.

2.3.2.2 Configuraciones en línea

Se trata de un tipo de fabricación en la que los lotes son de gran tamaño y lo productos son homogéneos, con poca o nula variedad de opciones. En esta configuración, las instalaciones se organizan de manera secuencial, con máquinas especializadas dispuestas en línea una tras otra. Esto es posible gracias a que todos los productos necesitan la misma secuenciación, o las variaciones son reducidas dentro del proceso productivo.

La maquinaria utilizada es altamente especializada y automatizada, lo que implica una inversión de capital significativamente alta (altos costes fijos). Sin embargo, esta inversión se justifica por la mayor eficiencia y homogeneidad en los procesos en comparación con configuraciones como el Job-Shop. A pesar de la especialización, las máquinas aún deben ser adaptables para llevar a cabo operaciones muy similares, pero no idénticas, lo que permite cierta versatilidad en la producción en comparación con la configuración continua, en la que las instalaciones están diseñadas para realizar una sola operación específica.

Los trabajadores tienden a estar altamente especializados en tareas específicas. Esta especialización conlleva menores costos variables gracias a las economías de escala, aunque se pierde flexibilidad. Para sacar rendimiento, requiere un alto nivel de utilización de las instalaciones y se enfoca en la producción de grandes volúmenes.

2.3.3 Configuración productiva continua

En lugar de fabricar por lotes, tenemos un flujo continuo de producción. Por lo que ya no habrá tiempos ociosos ni esperas. Se estarán ejecutando las mismas operaciones a lo largo del tiempo, en las mismas máquinas y obtendremos el mismo producto final de manera constante.

Los trabajadores realizan siempre la misma operación y están muy especializados. Existe un patrón de secuenciación fijo. Las distintas operaciones para obtener el producto final siempre siguen el mismo orden.

2.4 Restricciones en el problema de *Scheduling*

Existen distintos tipos de problemas de *Scheduling*, clasificables en función de la notación α|β|γ (Durasevic, 2020). Donde α representa el problema relacionado con los recursos, en el presente caso de estudio, las máquinas. B representa las restricciones a las que está sujeto el problema y en qué condiciones se opera. Por último, γ denota la función objetivo a minimizar o maximizar.

2.4.1 Restricciones de recursos (α)

Existen distintas configuraciones de los talleres que pueden surgir en el contexto de la programación de trabajos en una planta de fabricación. Cada uno de estos entornos tiene características únicas que afectan la complejidad y la eficiencia de la programación de trabajos (Araúzo, 2022).

En los modelos O, P, Q y R cada trabajo consiste en una única operación que puede ser realizada en una sola o varias máquinas:

- **Máquina única (O):** implica un único recurso de procesamiento y un flujo secuencial de trabajos a través de esta máquina.
- **Máquinas paralelas idénticas (Pm):** involucran múltiples máquinas idénticas que pueden procesar trabajos simultáneamente, lo que permite una mayor capacidad de producción.
- **Máquinas paralelas uniformes (Qm):** conjunto de m máquinas en paralelo, cada una de ellas tiene una eficiencia distinta. Todos los trabajos tienen el mismo tiempo de procesamiento en cada máquina.
- **Máquinas paralelas no relacionadas (Rm):** consiste en m máquinas en paralelo. El rendimiento de cada una de las máquinas depende de la máquina y del trabajo.

En los modelos presentados a continuación, cada trabajo está compuesto por un conjunto de operaciones. Estas pueden realizarse en una sola o varias máquinas y, además, en algunos casos deberán seguir un orden determinado, mientras que en otros no:

- **Flow Shop (FSP):** se refiere a un entorno donde todos los trabajos pasan a través de una secuencia predeterminada de máquinas. Todos los trabajos han de pasar por todas las máquinas en el mismo orden.
- **Flexible Flow Shop (FFSP):** es una generalización del Flow Shop; en lugar de m máquinas tenemos c centros de trabajo. En cada centro de trabajo tenemos un número determinado de máquinas idénticas. Cada trabajo puede seguir una ruta

específica a través de las etapas, lo que permite cierta flexibilidad y adaptabilidad en el proceso de producción.

- ***Job Shop* (JSP):** consiste en m máquinas, todas ellas en serie. Cada trabajo tiene su propia ruta específica a seguir.
- **Flexible *Job Shop* (FJSP):** es una generalización del *Job Shop*, pero existen c centros de trabajo; cada uno de ellos con un conjunto de máquinas iguales. Cada trabajo debe ser procesado en cualquiera de las máquinas de cada centro de trabajo, pero solo en una de ellas.
- **Open Shop (OSP):** consiste en m máquinas en serie. Cada trabajo debe pasar por cada una de las máquinas, aunque el tiempo de procesamiento en alguna máquina pueda ser cero. No existen relaciones de precedencia entre las operaciones.

2.4.2 Restricciones del proceso (β)

Algunas de las posibles restricciones del proceso que modificarán el problema son:

- **Fecha de lanzamiento (r):** cuando aparezca esta restricción, la operación j del trabajo i no podrá empezar antes de una fecha r_{ij} .
- **Interrupción (prmp)**: cuando una operación tiene esta variación, nos indica que no es necesario completar una operación una vez ha comenzado. El agente encargado de la planificación en la planta puede interrumpir el procesamiento de un trabajo en cualquier instante de tiempo y comenzar la ejecución de otro trabajo diferente en esa misma máquina.
- **Recirculación (rcrc)**: indica que un mismo trabajo puede pasar por la misma máquina en más de una ocasión.
- **Restricción de precedencia (prec)**: denota que algunos trabajos no pueden comenzar hasta que otros trabajos sean finalizados, es decir, existen relaciones de precedencia entre algunos trabajos. Existen distintas relaciones de precedencia:
- **Prec**: existe algún tipo de precedencia entre trabajos que no estén ya definidas previamente.
- ***Chain***: un trabajo puede tener como máximo un trabajo sucesor y un trabajo precedente.
- ***Intree***: un trabajo puede tener más de un trabajo precedente.
- ***Outtree***: un trabajo puede tener a varios trabajos sucesores.
- **Tiempos de preparación dependientes de la secuenciación (i):** el tiempo de procesamiento de una operación depende de la operación que ha sido programada previamente en esa misma máquina. Es el caso de algunas operaciones determinadas que son costosas de desmontar de la máquina antes de pasar a otra.

- **Tiempos de *setup* (s):** denota un tiempo adicional cuando un trabajo acaba de ser procesado en su totalidad en una máquina; representa el tiempo que tarda una máquina en ser preparada para procesar un nuevo trabajo.
- **Permutacional (prmu):** esta restricción ocurre en el entorno Flow shop. Indica que todos los trabajos deben seguir el mismo orden en todas las máquinas.
- **Paradas de máquinas (brkdwn):** indica que las máquinas no van a estar siempre disponibles a lo largo del tiempo. Habrá momentos, como en reparaciones o mantenimientos, en los cuales la máquina no va a poder procesar ninguna operación. Estas paradas pueden ser programadas o no programadas, en caso de avería inesperada.
- **Familias de trabajos (fmls):** es usada en aquellos casos en los que los trabajos se agrupan en un total de F familias de trabajo. Cuando las operaciones de los trabajos pertenecientes a la misma familia son procesados unas después de otras, no existe ningún tiempo adicional de preparación. Mientras que, en aquellos casos en los que las operaciones adyacentes en el tiempo pertenecen a trabajo de distintas familias, se debe añadir un tiempo extra de preparación.
- **Restricciones en el número de trabajos (nbr):** esta restricción impone un número máximo de trabajos que pueden ser procesados en un determinado intervalo de tiempo.
- **Restricciones de número máximo de operaciones de un trabajo (n):** como su propio nombre indica, existe un número máximo de operaciones por trabajo. Este número máximo es común para todos los trabajos.
- **Fechas límite (d):** cada trabajo ha de ser entregado en un plazo determinado, por lo que existen fechas máximas de entrega para cada uno de ellos.

2.4.3 Función objetivo (γ)

Las funciones objetivo a optimizar son función de los tiempos de finalización de las operaciones que conforman nuestro programa de producción. Estas son algunas de las más comúnmente usadas:

- ***Makespan* (Z_{max}):** es el tiempo que transcurre desde que entra el primer trabajo en la planta hasta que el último trabajo de todos es procesado completamente.

$$Z_{max} = max_i(C_i) \ [1]$$

- ***Maximun Flowtime* (F_{max}):** tiempo máximo transcurrido en cualquiera de los trabajos.

$$F_{max} = max_i(F_i) \ [2]$$

- ***Maximun tardiness*** **(T_{max}):** es el retraso máximo llevado a cabo en cualquiera de los trabajos.

$$T_{max} = max_i(T_i) \text{ [3]}$$

- ***Total Weighted Earliness and Tardiness*** **(TWET):** es la suma ponderada de los retrasos y los adelantos. Ponderación adelanto (w'_i)**;** ponderación retraso (w''_i)**.**

$$TWET = \sum_i w'_i E_i + w''_i T_i \text{ [4]}$$

- ***Total Flowtime*** **(Ft)**: es la suma de todos los tiempos fin de todos los trabajos.

$$Ft = \sum_i F_i \text{ [5]}$$

- ***Weighted Flow Time*** **(WFT):** suma ponderada del tiempo de flujo de jada trabajo. Cada trabajo tiene un peso.

$$WFT = \sum_i w_i T_i \text{ [6]}$$

2.5 Problema Job Shop *Scheduling*

Este apartado se centra en la programación de máquinas y de talleres. Podemos diferenciar dos problemas:

Machine Scheduling Problem (Programación de Máquinas): Se refiere a la asignación eficiente de tareas o trabajos a máquinas en un entorno de producción. El objetivo principal es optimizar la utilización de las máquinas, minimizar el tiempo de inactividad y maximizar la eficiencia de la producción. Este enfoque es crucial en entornos donde múltiples tareas o trabajos deben ser completados en un conjunto limitado de máquinas, como en fábricas y plantas de fabricación.

Job Shop Scheduling Problem (Programación de Taller): Se refiere a la planificación de la secuencia de operaciones en un taller de producción donde varias tareas o trabajos deben pasar a través de una serie de máquinas en un orden específico. El objetivo es planificar y programar estas operaciones de manera que se minimice el tiempo total de producción y se maximice la eficiencia general del proceso de fabricación.

2.5.1 Programación de operaciones en entornos *Job Shop*

Ya se explica en apartados anteriores, (2.3.2.1) en qué consiste la configuración *Job Shop*, pero ahora se va a ver en detalle el modelo (Pinedo, 2009).

En este tipo de problemas, hay n trabajos y cada trabajo visita un número de máquinas siguiendo una ruta predeterminada. Cada vez que un trabajo es tratado en una máquina recibe el nombre de "operación". De esta manera, un trabajo está formado por un conjunto de operaciones.

En algunos modelos, un trabajo puede visitar una máquina determinada como máximo una vez, y en otros modelos un trabajo puede visitar cada máquina más de una vez. En este último caso, decimos que el modelo del taller está sujeto a recirculación.

Una generalización del *Job Shop* es el *Job Shop Flexible*, en el cual un taller está formado por varios centros de trabajo distintos (CT). La operación i de un trabajo, puede ser procesada en cualquiera de las máquinas que se encuentran en el mismo CT. Es importante destacar que no tiene por qué tener el mismo tiempo de procesamiento para una misma operación, todas las maquinas del mismo CT. Se pueden asignar una o varias máquinas para realizar las operaciones para cada tarea, y la asignación de máquinas debe considerar las capacidades y habilidades de estas para lograr la eficiencia y calidad en la producción.

2.5.2 Caracterización de los programas

En la dirección de operaciones, los problemas *Job Shop*, han sido estudiados de manera exhaustiva a lo largo de los años, estudiando la factibilidad del problema, la complejidad y las formas de abordarlo; (Conway, 1967) (Baker, 1974) (Kan, 2012) (French, 1982). Los problemas tipo *Job Shop* pueden ser clasificados en semiactivos, activos y sin tiempos muertos.

Hay variaciones respecto a la clasificación de estos problemas en función de cada autor. La más común y a la que se va a hacer referencia es la propuesta por Baker (1974). Esta clasificación aparte de ser utilizada en los JSP puede ser usada de forma global en todos los tipos de problemas de programación de proyectos con restricciones (RCPSP). (Arno Sprecher, 1995).

Antes de describir los diferentes tipos de programas existentes, es importante establecer algunas definiciones fundamentales que facilitarán la comprensión.

- Un programa factible es una solución que cumple con todas las restricciones de recursos y órdenes de precedencia.
- Un desplazamiento local a la izquierda de una operación j, 1<j<O, dentro de un horario factible S, es desplazar una operación hacia una posición anterior en el diagrama de *Gantt*, manteniendo el orden secuencial de las operaciones.

- Un desplazamiento global a la izquierda de una operación j, 1<j<O, dentro de un programa factible S, es una traslación hacia la izquierda de la actividad j que no se puede obtener mediante una traslación hacia la izquierda local. Es decir, se adelanta el comienzo y fin de una operación j permitiendo alterar el orden de operaciones.

Dependiendo de si en un programa se pueden adelantar o no sus operaciones, se pueden clasificar en:

- Un **programa semiactivo** es aquel en el que a ninguna de las operaciones j se la puede aplicar un desplazamiento local a la izquierda (Ilustración 1). Según la definición dada por Baker en 1974: "En un programa semiactivo, el tiempo de inicio de una operación en particular está condicionado por el procesamiento de otro trabajo en la misma máquina o por el procesamiento de la operación inmediatamente anterior en una máquina diferente".

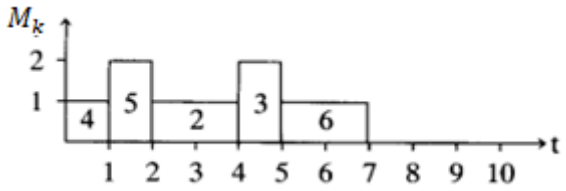

Ilustración 1. Programa semiactivo

Un programa factible (Ilustración 2) se puede transformar en un programa semiactivo mediante una serie de desplazamientos locales hacia la izquierda. Es importante destacar que en problemas *Job Shop Flexible*, hay más de un programa semiactivo derivado de un programa factible en la mayoría de los casos.

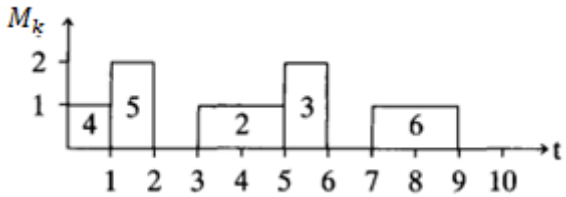

Ilustración 2. Programa factible

- Un **programa activo** (Ilustración 3) es aquel en el que ya no puede realizarse ningún desplazamiento global (y por lo tanto tampoco local) a la izquierda.

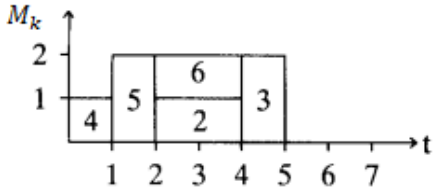

Ilustración 3. Programa activo

- Un **programa sin tiempos muertos** es aquel en el que las máquinas no tienen intervalos de tiempo en los que están paradas mientras siga habiendo operaciones en cola.

- Un **programa con tiempos muertos** es aquel que existen intervalos de tiempo en los que las máquinas están paradas.

Todos los programas semiactivos son programas con tiempos muertos, pero no todos los programas con tiempos muertos son semiactivos, tal y como se muestra en la Ilustración 4.

Los programas activos pueden ser programas con y sin tiempos muertos. Pero todos los programas sin tiempos muertos son programas activos.

Ilustración 4. Caracterización programas (Araúzo, 2022)

2.5.3 Complejidad del problema

Dentro del ámbito de la complejidad informática se encuentran diferentes tipos de problemas que se agrupan en conjuntos o categorías de complejidad. Una categoría de complejidad es un conjunto de problemas de decisión relacionados. Un problema de decisión es aquel en el que las posibles soluciones se reducen a sí o no.

Muchos problemas pueden transformarse en un problema de decisión equivalente. Los problemas pertenecientes a la clase P son aquellos que pueden resolverse en un tiempo polinómico mediante una máquina de Turing determinista. Por otro lado, en los problemas de la clase NP no se puede garantizar encontrar la solución óptima en un tiempo polinómico, pero sí verificar una solución.

Como se muestra en la Ilustración 5, existen dos posibles clasificaciones, en función de si P=NP o P≠NP:

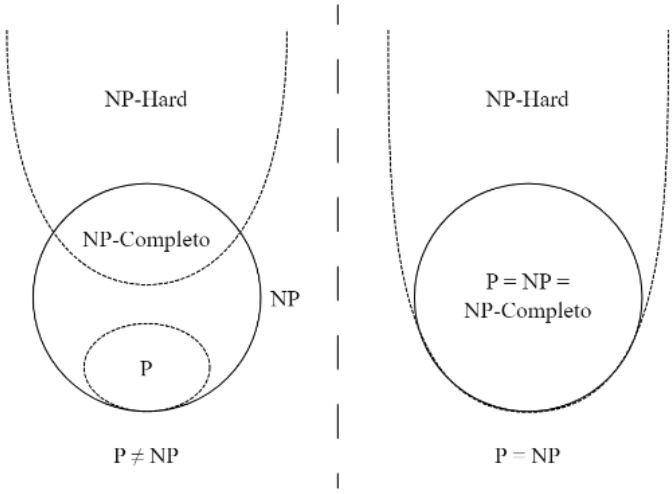

Ilustración 5. Posibles clasificaciones de la complejidad computacional

Es importante notar que cualquier algoritmo determinista puede utilizarse en la verificación de un algoritmo no determinista. La relación entre estas clases es crucial, ya que se ha demostrado que P es al menos un subconjunto de NP. (Crespo, 2022)

Además, en el ámbito de los problemas NP, se encuentran los problemas NP-Completos. Un problema de decisión es considerado NP-Completo si pertenece a la clase NP y es al menos tan difícil como cualquier otro problema en la misma clase. Aunque no se ha demostrado, se cree que los problemas NP-Completos son los más difíciles en la clase NP y probablemente no formen parte de la clase P.

En el caso de los problemas de optimización, se habla de la clase de problemas NP-duros, la cual se corresponde con la NP-completa.

La dimensionalidad del problema del JSP viene dada por n x o, siendo n el número de trabajos y o el número de máquinas, de tal manera que se tienen al menos (n!)*o posibles soluciones generándose una explosión combinatoria al crecer n y o. El crecimiento exponencial del número de posibles soluciones hace que este problema sea reconocido como NP-duro.

2.5.4 Métodos de resolución *Job Shop Scheduling*

El problema de la programación de la producción en un entorno de taller de trabajo (*Job Shop Scheduling*) es un desafío complejo que implica la asignación de unas determinadas operaciones que conforman trabajos, a las máquinas del taller. El objetivo final es optimizar las diferentes funciones objetivo, vistas anteriormente. Existen diversas maneras de abordar este problema. A continuación, se describirá algunas de las técnicas más comunes:

2.5.4.1 Modelo de planificación separada

La planificación separada consiste en la descomposición en subproblemas interrelacionados de forma que al resolver los subproblemas se obtiene la solución al problema original. Este enfoque tiene como objetivo reducir la complejidad del manejo del problema y mejorar la eficiencia de procesamiento, al mismo tiempo que requiere menos recursos computacionales en comparación con la resolución del problema en su totalidad (Juan Carlos Osorio*, 2007).

El modelo de la planificación separada de la producción reconoce la existencia de diferentes niveles o etapas de toma de decisiones dentro del sistema de gestión de la producción. Cada nivel aborda problemas específicos en relación con el tiempo y los recursos. El objetivo principal de este enfoque es lograr la coherencia y la sinergia entre los subproblemas presentados.

Los primeros estudios que utilizan este enfoque datan de las investigaciones llevadas a cabo por Hax y Meal (Meal, 1973), en los cuales hay cuatro niveles de planificación:

- Asignación de productos a plantas a largo plazo (mediante programación entera mixta).
- Planificación de inventario periódico (mediante programación lineal).
- Creación de horarios detallados para cada familia de productos (usando métodos estándar de control de inventario).
- Cálculo de las cantidades individuales para cada artículo.

Luego, en 1995 (Schneeweis, 1995) presentó una estructura sólida de los modelos de planificación separada, integrándolos en los modelos de toma de decisiones distribuidas y de soporte. Años más tarde, se realizaron contribuciones significativas, abordando la

consideración de múltiples productos, la optimización de costos de inventario y la integración de restricciones de capacidad en la planificación de la producción.

En su posterior trabajo, Schneeweiss amplió aún más los elementos clave del sistema de planificación separada, centrándose en el concepto de anticipación como un pilar fundamental en la planificación.

2.5.4.2 Algoritmos constructivos y de mejora

Para encontrar soluciones óptimas al problema *Job Shop*, se han utilizado a lo largo de los años métodos de ramificación y acotación (E Ignall, 1965). Años más tarde, en (Szwarc, 1983) se analizó las propiedades de este problema y definió una clase de casos resolubles de manera eficiente. A pesar de esto, debido a la NP-complejidad del problema, es muy difícil encontrar algoritmos eficientes que puedan resolver de manera óptima el caso general. A partir de este hecho, se han centrado los focos de investigación en el campo de métodos heurísticos. La mayoría de las heurísticas se dividen en dos tipos: heurísticas constructivas, como (C Wang, 1997) y heurísticas de mejora, por ejemplo (Ho, 1995).

Las heurísticas constructivas y las heurísticas de mejora son dos enfoques distintos utilizados en la resolución de problemas complejos.

Las **heurísticas constructivas** se centran en construir soluciones factibles paso a paso, comenzando desde una solución vacía y agregando gradualmente componentes para construir una solución completa. Además, toman decisiones óptimas en cada paso sin considerar el impacto total en el resultado final, aunque se pueden agregar modificaciones que tengan en cuenta cómo la elección en cada iteración afecta a futuras iteraciones.

Computacionalmente, las heurísticas constructivas tienden a ser más rápidas en términos de tiempo de ejecución, ya que no requieren un procesamiento intensivo y se centran en construir soluciones viables.

Las soluciones obtenidas a partir de las heurísticas constructivas no se acercan a las soluciones óptimas; pero tienden a ser aceptables y pueden servir como puntos de partida para métodos de mejora.

Dentro de los métodos heurísticos constructivos, se encuentran dos partes bien diferenciadas. La primera de ellas, denominada esquema de construcción del programa, se centra en la forma de elegir qué operaciones son elegibles. En función del esquema de construcción del programa, se obtendrán diferentes tipos de programa, como por ejemplo semiactivos, activos, con o sin tiempos muertos, etc.

La segunda parte, denominada heurística, consiste en seleccionar la operación a programar dentro de las elegibles.

Las **heurísticas de mejora** se utilizan para refinar y mejorar las soluciones existentes. En la primera iteración se parte de una solución completa y a medida que aumenta el número de iteraciones se va mejorando la solución. Estas heurísticas exploran el espacio de soluciones a partir de una solución dada y realizan cambios iterativos para mejorar la solución actual, con la esperanza de alcanzar un óptimo local o incluso global.

Respecto a la complejidad computacional, las heurísticas de mejora suelen ser por lo general más costosas, ya que implican un análisis más profundo de las soluciones y requieren un número mayor de iteraciones para lograr mejoras significativas.

Estas heurísticas pueden producir soluciones de mejor calidad en comparación con las heurísticas constructivas, ya que se centran en optimizar soluciones existentes y tienen como objetivo mejorar la calidad de la solución inicial.

2.5.4.3 Programación lineal

La programación lineal en un entorno *Job Shop* implica la formulación de una función objetivo y un conjunto de restricciones que representan las limitaciones y las secuencias de operaciones requeridas. En el ámbito de la programación de la producción, se suele referir a los recursos y tareas como máquinas y trabajos respectivamente, mientras que la métrica de rendimiento más comúnmente utilizada es el tiempo de finalización de los trabajos.

El problema *Job Shop* ha sido ampliamente investigado desde principios de la década de 1950, pero no es hasta principios de los 60, cuando surgen los primeros modelos matemáticos aplicados al sector.

Los enfoques matemáticos, como la programación lineal entera mixta (MILP), se utilizan frecuentemente para abordar estos problemas de programación. Los modelos MILP han demostrado ser eficaces en la resolución de problemas de programación, proporcionando formulaciones precisas y soluciones cercanas a los óptimos en entornos de programación complejos.

Si combinamos la programación lineal con heurísticas como algoritmos genéticos, búsqueda tabú, recocido simulado y enfoques híbridos; podemos mejorar las soluciones obtenidas, llegando incluso en ocasiones a los óptimos globales.

El gran investigador J.F. Shapiro, presentó modelos de programación matemática y métodos de solución que se han aplicado a varios tipos de problemas de planificación y programación de producción. C.H. Pan ha proporcionado una revisión y comparación de formulaciones MILP para problemas JSP, FSP y FS prmu.

Sin duda, uno de los avances más revolucionarios en el sector de la dirección de operaciones aplicada a talleres fue el propuesto por Brandimarte (Brandimarte, 1999); el cual propone un modelo de programación flexible y multiobjetivo. La flexibilidad de plan de proceso implica la capacidad de ajustar los planes de proceso en función de las

condiciones cambiantes y las demandas del entorno. El enfoque multiobjetivo implica la consideración de múltiples criterios en la toma de decisiones, lo cual hace que en lugar de centrarnos en un único objetivo como podría ser el *makespan*; el programa tenga en cuenta varios objetivos simultáneamente.

Low y Wu han abordado el FJSP con el objetivo de minimizar la tardanza total, teniendo en cuenta el tiempo de configuración. El problema presentado como un grafo disyuntivo extendido se ha formulado como un modelo de programación entera mixta y luego se han linealizado términos cuadráticos en las restricciones. (Özgüven)

I.C. Choi y D.S. Cho han presentado un modelo MILP para el FJSP con configuraciones dependientes de la secuencia, junto con un algoritmo de búsqueda local que utiliza una regla de despacho para obtener un límite superior en el tiempo de ejecución de un subproblema.

Estos son algunos de los investigadores y matemáticos que asentaron las bases de la programación lineal en la programación de talleres, pero en la actualidad se continúa avanzando en este sector, y resulta prácticamente imposible recopilar de manera representativa todos los programas propuestos debido al gran número que hay y a el carácter único de cada uno de ellos.

2.6 Representación gráfica del problema

La planificación y control de proyectos comenzó a ganar importancia después de la Segunda Guerra Mundial, cuando se difundió el diagrama de *Gantt*. Hasta finales de 1950 era la única herramienta de la que se disponía; fue utilizada en proyectos de las Fuerzas Especiales Marítimas de Estados Unidos de América para la construcción de submarinos atómicos armados con proyectiles. Proyecto el cual se le estimaba una duración de 5 años.

En 1958 se publica un ensayo sobre el *Programme Evaluation and Review Technique* (PERT). Consiguiendo una estimación de tres años para el proyecto de las Fuerzas Marinas citado anteriormente.

También en 1958, meses más tarde, el equipo de Du Pont; dirigido por J. E. Kelley y M. R. Walker crean la técnica denominada *Critical Path Method* (CPM, método del camino crítico), el cual permite determinar el tiempo más corto posible para completar un proyecto específico y resaltar las actividades que no pueden retrasarse sin retrasar el proyecto en su conjunto (Hofmann, 1993).

La diferencia entre el PERT y el CPM es que el primero se centra en aspectos temporales basándose en estimaciones probabilísticas; mientras que el CPM funciona con duraciones deterministas. Hoy en día, en la mayoría de las situaciones se utiliza una combinación de ambos métodos.

De forma paralela surge el método de los potenciales, conocido hoy en día como diagrama de Roy. El cual fue desarrollado por Walter A. Shewhart en la década de 1920 y luego modificado y popularizado por su compañero y amigo, Albert F. H. S. Roy en 1956. Se diferencia del PERT y del PM en que tiene mayor flexibilidad para simular las interrelaciones entre las distintas actividades del proyecto.

Para comenzar a construir cualquier diagrama o grafo de los citados anteriormente, debemos partir de dos formatos posibles, o la matriz de encadenamientos (Tabla 1), o la tabla de precedencias (Tabla 2). A continuación, se muestra un ejemplo gráfico de ambos tipos.

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

Tabla 1. Ejemplo de una matriz de encadenamientos

Actividades	Precedentes
A	
B	
C	A, B
D	A
E	A
F	D

Tabla 2. Ejemplo tabla de precedencias

2.6.1 Diagramas de *Gantt*

El diagrama de *Gantt* (Ilustración 6) es una representación gráfica de la secuenciación de las operaciones a lo largo del tiempo. Consta de dos ejes, en el de las abscisas se representa el tiempo y en el de ordenadas el conjunto de actividades a desarrollar. (Pastor, 2011).

Fue desarrollado por L. *Gantt* en 1910. Es la representación más utilizada hoy en día debido a su fácil interpretabilidad y facilidad de uso. Se caracteriza por:

- Las operaciones se representan por segmentos horizontales cuya longitud es proporcional a su duración.

- La posición del segmento respecto al eje horizontal representa el intervalo de tiempo de ejecución de la tarea.

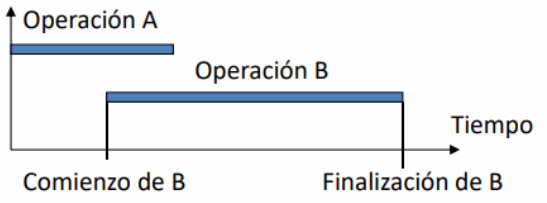

Ilustración 6. Ejemplo Diagrama de Gantt (Marta Posada, 2020).

Existen dos formas de programar un diagrama de *Gantt*:

Programación hacia adelante implica la programación en función del orden de las relaciones de precedencia. Las actividades se programan solo después de que todas sus tareas predecesoras han sido programadas. Este método se ajusta al inicio de las operaciones en la fecha más temprana posible.

Por otro lado, la programación hacia atrás implica la programación siguiendo el orden inverso determinado por las relaciones de precedencia. Las actividades se programan solo después de que todas sus sucesoras han sido programadas. Este enfoque se ajusta al final de las operaciones en la fecha más tardía posible.

2.6.2 Grafo PERT

En este método se comienza descomponiendo el proyecto en actividades. Si hacemos el paralelismo aplicándolo al campo de la dirección de operaciones, estas actividades serían las operaciones realizadas en las máquinas. Se debe tener la duración de cada actividad.

Un concepto esencial son los sucesos, representan un instante de tiempo. No consume recursos, solamente indica el principio o fin de una actividad.

En el grafo PERT, los vértices representas los sucesos y los arcos las actividades (Ilustración 7). El grafo solo tendrá un suceso inicial y un suceso final. Todas las actividades, excepto las que salen del suceso inicial o llegan al suceso final, tienen al menos una actividad precedente y una actividad sucesora. Toda actividad “ijk” llegará a un suceso de orden superior al del que sale. No puede haber dos actividades que, partiendo del mismo suceso, vayan al mismo suceso.

A veces el cumplimiento obligatorio de estas reglas hace que no se puedan dar algunas situaciones factibles, es aquí cuando se usan actividades ficticias (representadas mediante trazo discontinuo).

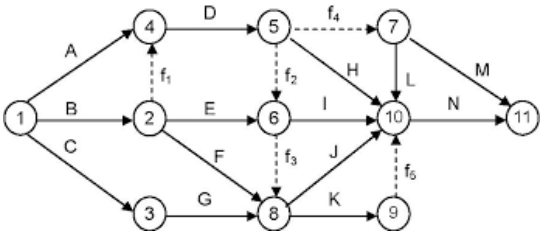

Ilustración 7. Ejemplo grafo PERT

2.6.3 Grafo ROY

El grafo ROY difiere principalmente del PERT en que en esta técnica los nodos representan las actividades y los arcos únicamente representan exclusivamente la interrelación de los nodos, tal y como refleja la Ilustración 8.

Consta de un nodo inicial de duración nula, al que seguirán aquellas actividades que no tienen ningún precedente. A continuación, se irán añadiendo el resto de las actividades, respetando las relaciones de precedencia establecidas en la matriz de encadenamientos o la tabla de precedencias hasta llegar al nodo final.

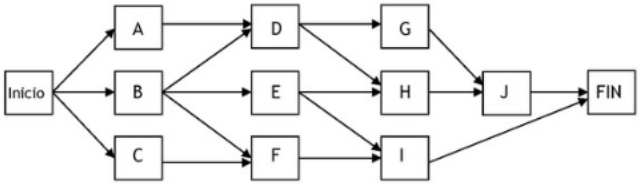

Ilustración 8. Ejemplo grafo ROY

3 MÉTODOS DE RESOLUCIÓN

3.1 Introducción

El problema *Job Shop Flexible* (FJSP) es una generalización del *Job Shop* clásico (JSP). En el FJSP las operaciones pueden ser procesadas en más de una máquina, es decir, se tienen distintas alternativas a la hora de procesar una operación. Por lo que el FJSP es más costoso de resolver que el JSP clásico.

En la literatura, se proponen dos maneras para resolver los problemas FJSP (aunque existe un tercero, los métodos de descomposición, como se verá más adelante):

- El enfoque de los **modelos matemáticos formulados mediante programación Entera Mixta**, los cuales tienen capacidades computacionales limitadas en cuanto al tamaño de los problemas, a pesar de la capacidad computacional de los sistemas informáticos modernos.
- **Enfoques heurísticos y metaheurísticos**. Para la resolución del problema mediante algoritmos heurísticos, se emplean dos enfoques. El enfoque separado, en el cual se resuelven dos subproblemas que consisten en la asignación de las máquinas a las operaciones y en la secuenciación de las operaciones; y el enfoque conjunto, en el cual estos subproblemas son resueltos de manera simultánea.

Debido a la gran complejidad y tamaño de los problemas FJSP, inducen a ser resueltos ayudándonos de métodos heurísticos, los cuales aportan soluciones razonablemente buenas teniendo en cuenta el bajo tiempo de procesamiento. Normalmente, las heurísticas no ofrecen soluciones que garanticen estar cerca del óptimo, pero suelen ser eficaces en la mayoría de los casos.

Por otro lado, tenemos la resolución de estos problemas mediante metaheurísticas. Las metaheurísticas (Sörensen, 2013) son métodos de resolución de propósito general que

ofrecen un marco genérico para desarrollar enfoques heurísticos específicos. Estas estrategias destacan como una de las mejores aproximaciones para abordar problemas de optimización combinatoria. A diferencia de las heurísticas tradicionales, las metaheurísticas no son algoritmos específicos, sino más bien unas reglas básicas para la construcción de algoritmos que no dependen del problema en cuestión. Además, las metaheurísticas no están limitadas a una disciplina particular, ya que combinan conceptos de diversos campos como la genética, la biología, las matemáticas, la física o la neurología.

Una de las ventajas clave de las metaheurísticas es su aplicabilidad general, brindando buenos resultados prácticos y siendo relativamente fáciles de implementar y paralelizar. Sin embargo, también presentan desventajas, ya que, al igual que las heurísticas, son algoritmos aproximados y no exactos, y en algunos casos, carecen de un sólido respaldo teórico. A pesar de estas limitaciones, las metaheurísticas se han convertido en herramientas valiosas en la búsqueda de soluciones eficientes para problemas complejos de optimización, destacando por su flexibilidad y capacidad para abordar una amplia gama de contextos y disciplinas.

3.2 Notación

Los problemas de *Scheduling*, en general, están formados por un conjunto de trabajos ("n" trabajos) que han de ser procesados en una o más máquinas ("m" máquinas). Estos trabajos están compuestos por una serie de operaciones ("o" operaciones) que deben ser procesadas en determinadas máquinas.

La resolución del problema se basa en la asignación de estas máquinas a las operaciones de los trabajos a lo largo del tiempo.

El subíndice "k" se emplea para especificar una máquina en concreto. El subíndice "i" se usa para denotar a qué trabajo nos referimos. Por último, el subíndice "j" se usa para referirnos a la operación concreta del trabajo "i". Adicionalmente, en alguno de los problemas que se mostrarán más adelante, contaremos con "l" centros de trabajo, y cada uno de ellos se denota con el subíndice "t".

En ocasiones no será necesario referirnos a ninguna máquina con el índice "k", ya que todas las máquinas son exactamente iguales y tardan el mismo tiempo en procesar una operación.

En los diferentes problemas de *Scheduling*, existen todas o solo algunas de las siguientes características:

- **Tiempo de procesamiento o *processing time* (p_{ijk}):** tiempo que tarada la operación j del trabajo i en ser procesada en la máquina k.

- **Tiempo de disponibilidad o *release date* (r_{ij}):** instante de tiempo en el que la operación j del trabajo i puede ser programada.
- **Fecha límite o *due date* (d_i):** instante de tiempo en el que, si no se ha completado el trabajo i, existe en retardo en el trabajo.
- **Tiempo comienzo o *begining time* (b_{ijk}):** instante de tiempo en el que la operación j del trabajo i comienza a ser ejecutada en la máquina k. B_i será el instante de tiempo en el que comienza el trabajo i, $B_i = b_{i0k}$
- **Tiempo fin operación o *completion time* (c_{ijk}):** instante de tiempo en el que la operación j del trabajo i termina de ser procesada en la máquina k. C_i será el instante de tiempo en el que el trabajo i es finalizado en su totalidad, $C_i = c_{iok}$, siendo "o" el número de operaciones del trabajo "i".

$$c_{ijk} = b_{ijk} + p_{ijk} \text{ [7]}$$

- **Peso o *weight* (w_i):** importancia de un trabajo j. A mayor peso, más prioritario será ese trabajo.
- **Tiempo de retardo o *tardiness* (T_i):** tiempo transcurrido desde el instante de tiempo en el que el trabajo debería haber concluido, hasta el instante en el que realmente es procesado completamente.

$$T_i = \max(C_i - d_i, 0) \text{ [8]}$$

- **Tiempo de adelanto o *earliness* (E_i):** tiempo transcurrido desde el instante de tiempo en el que el trabajo termina, hasta el momento en el que debería haber terminado.

$$E_i = \max(-(C_i - d_i), 0) \text{ [9]}$$

- **Tiempo de flujo o *Flowtime* (F_i):** tiempo que ha estado el trabajo i en el sistema.

$$F_i = C_i - r_{i0} \text{ [10]}$$

- **Tiempo restante o *remaining time* (RT_{ij}):** tiempo de procesamiento total que queda por procesar a parir de la operación actual.

$$RT_{ij} = \sum_{i}^{n} p_{ij} \text{ [11]}$$

3.3 Esquema de construcción del programa

En los métodos heurísticos constructivos distinguimos dos aspectos, la primera es el esquema de construcción del programa, el cual afecta a la forma de seleccionar las operaciones elegibles. La segunda, es la heurística, la cual selecciona la operación a procesar entre las elegibles.

La finalidad del esquema de construcción de los programas es obtener las operaciones elegibles. Posteriormente, sobre estas operaciones elegibles se trabajará para obtener la operación que se va a procesar finalmente.

Los esquemas de construcción de programas más utilizadas son del tipo hacia adelante, se comienza programando operaciones sin precedente. Una vez se hayan programado una operación, se desbloquea su sucesora y esta entra dentro de las programables.

Las dos técnicas destacadas dentro de los esquemas de construcción hacia adelante son:

3.3.1 Método de Lanzamiento

Lo primero a explicar de este método es que selecciona, dentro del conjunto de elegibles, aquellas que antes pueden comenzar. Se denomina “de lanzamiento” debido a la metodología seguida; la cual hace un símil al lanzamiento de las operaciones en los talleres que carecen de cualquier tipo de programa: en cuanto una máquina queda libre, se selecciona una operación de las que hay en cola.

Al usar el esquema de construcción de lanzamiento, se obtienen programas sin tiempos muertos; lo cual no garantiza que entre estos programas esté el óptimo.

Para aplicar el método de lanzamiento primero se tiene que dividir el conjunto de operaciones programables en “m” subconjuntos ($E_1, E_2, \dots, E_m$), siendo esta el número de máquinas de las que dispone el taller. Cada subconjunto estará formado por todas las operaciones programables de esa máquina (Araúzo, 2022).

Para cada máquina k, se tiene f_k, que representa la fecha en la que la máquina k está disponible para poder ser utilizada.

Para cada operación j del trabajo i se tiene r_{ij}, la cual indica la fecha de disponibilidad de la operación O_{ij}. Esta se calcula de la siguiente manera:

$$r_{ij} \begin{cases} r_i & si\ \ j = 1 \\ c_{i\,j-1} & si\ j > 1 \end{cases} \quad [12]$$

Es decir, si es la primera operación del trabajo se tendrá en cuenta únicamente la fecha de disponibilidad de la máquina donde se deba procesar O_{ij}. Por otro lado, si el índice de la operación j es superior, se tendrá en cuenta la fecha fin de operación predecesora.

La fecha más temprana en la que la operación ij se denota como rp_{ij}.

$$rp_{ij} = max\left\{r_{ij}, f_{k_{ij}}\right\} \text{ [13]}$$

Una vez se calcula rp_{ij}, se puede aplicar el esquema de construcción del programa de lanzamiento en concreto. En cada iteración se selecciona la máquina que antes puede comenzar, para ello se calcula la fecha más temprana en la que una operación puede comenzar en la máquina (fp_k).

$$fp_k \begin{cases} \min\limits_{ij \in E_k} \{rp_{ij}\} & si\ E_k \neq \emptyset \\ \infty & si\ E_k = \emptyset \end{cases} \text{ [14]}$$

Una vez se calcula los fp_k para cada una de las k máquinas, se selecciona aquella que tiene el menor de ellos.

$$fp^{min} = \min_k\{fp_k\} \rightarrow k_{lanz} = k\ \ tal\ que\ fp_k = fp^{min} \text{ [15]}$$

Una vez se tiene k_{lanz}, se elabora el conjunto de operaciones elegibles (E^*); para ello se toman aquellas que pueden comenzar en fp^{min}.

$$E^* = \left\{ij / ij \in E_{k_{lanz}}\ y\ r_{ij} \leq fp^{min}\right\} \text{ [16]}$$

3.3.2 Método de *Giffler y Thompson*

La clave de este método radica en seleccionar dentro del conjunto de operaciones programables aquellas pertenecientes a la máquina que antes puede terminar, y dentro de esas, aquellas que pueden empezar antes de esa fecha de finalización.

Al usar el esquema de construcción de *Giffler y Thompson*, se obtienen programas activos únicamente, esto hace que entre estos programas esté el óptimo; como se vio anteriormente.

Para calcular rp_{ij} se sigue exactamente el mismo procedimiento que en el método de lanzamiento.

$$rp_{ij} = max\left\{r_{ij}, f_{k_{ij}}\right\} \text{ [17]}$$

En cada iteración se selecciona la máquina que antes puede finalizar, para ello se calcula la fecha más temprana en la que una operación puede finalizar en la máquina k (ff_k).

$$ff_k \begin{cases} \min\limits_{ij \in E_k} \{rp_{ij} + d_{ij}\} & si\ E_k \neq \emptyset \\ \infty & si\ E_k = \emptyset \end{cases} \text{ [18]}$$

Una vez se calcula los ff_k para cada una de las k máquinas, se selecciona aquella que tiene el menor de ellos.

$$ff^{min} = \min_{k}\{ff_k\} \rightarrow k_{G\&T} = k \ \ tal\ que\ ff_k = ff^{min} \ [19]$$

Una vez se ha obtenido la máquina $k_{G\&T}$, se elabora el conjunto de operaciones elegibles (E^*); para ello se toman aquellas que pueden comenzar en fp^{min}.

$$E^* = \{ij/ij \in E_{k_{G\&T}}\ y\ r_{ij} \leq ff^{min}\} \ [20]$$

3.3.3 Métodos de inserción

A diferencia de las estrategias de Lanzamiento y *Giffler y Thompson*, donde la incorporación de nuevas operaciones se realiza siguiendo una secuencia detrás de las ya programadas, los algoritmos de inserción permiten introducir operaciones intercaladas entre las ya planificadas.

Estos algoritmos comienzan programando las operaciones de un solo trabajo, como por ejemplo la de mayor tiempo total de procesamiento. Después de programar el primer trabajo, se procede a programar el resto de las operaciones siguiendo un procedimiento específico: primero se elige una operación programable, generalmente la de mayor tiempo de procesamiento, luego se selecciona la máquina correspondiente y se prueba a agregar la operación en todas las posiciones posibles (desde la primera hasta la última operación). A continuación, se evalúa una función heurística para cada posible solución, como el tiempo más tardío en el que concluye la operación programada que finaliza más tarde. Por último, se programa la operación en la posición que devuelve el mejor valor según la función heurística.

Estos métodos suelen ofrecer resultados superiores a los algoritmos basados en reglas de lanzamiento, aunque requieren mayores recursos computacionales.

3.3.4 Método de Cuello de Botella

En entornos productivos, es común encontrar un recurso clave conocido como "cuello de botella" o “máquina crítica”, cuya saturación supera a la del resto de los recursos. La gestión adecuada de estos cuellos de botella es esencial para maximizar la eficiencia del sistema. Los métodos de programación que se centran en los cuellos de botella se enfocan en la identificación y gestión de estos recursos críticos. Inicialmente, se identifica el recurso que actúa como cuello de botella, y luego se establece un programa parcial específico para dicho recurso. Este programa define prioridades de producción que influyen en la actividad de los demás recursos del sistema. Aunque este enfoque simplifica el sistema alrededor de un único recurso, presenta un desafío significativo: la identificación de un cuello de botella absoluto. En la mayoría de los casos, la naturaleza del cuello de botella depende del programa de producción, y en entornos dinámicos, estos cuellos de botella pueden cambiar continuamente.

3.4 Heurísticas

En el ámbito de la dirección de operaciones, se define el concepto de heurística de la siguiente forma:

"Se califica de heurístico a un procedimiento para el que se tiene un alto grado de confianza en que encuentra soluciones de alta calidad con un coste computacional razonable, aunque no se garantice su optimalidad o su factibilidad, e incluso, en algunos casos, no se llegue a establecer lo cerca que se está de dicha situación. Se usa el calificativo heurístico en contraposición a exacto (Melian, 2003)."

Encontrar una secuenciación de las operaciones de los trabajos óptima, que permita maximizar el rendimiento de los recursos que disponemos y minimice el *makespan*[3] , es un problema que perdura a lo largo de los años. Hoy en día, continúa la investigación en este campo, en busca de heurísticas que permitan optimizar el proceso productivo.

Existen gran número de reglas mediante las cuales se selecciona la operación a programar entre el conjunto de operaciones elegibles. Las heurísticas que se citan más adelante son de baja complejidad, que se utilizan para obtener soluciones aceptables. Estas heurísticas pueden usarse para obtener soluciones a partir de las cuales trabajar con metaheurísticas de mayor complejidad.

- FIFO (*First In First Out*): se selecciona la operación que antes ha llegado a la máquina (operación con menor r_{ijk}). Se utiliza cuando se quiere minimizar el *Maximun Flowtime* (F_{max}).
- SPT (*Shortest Processing Time*): se selecciona la operación entre el conjunto de elegibles que tiene menor duración (d_{ijk}). Se utiliza cuando se quiere minimizar el Total *Flowtime* (Ft) y el *Maximun Flowtime* (F_{max}).
- LPT (*Longest Processing Time*): selecciona la operación más larga (la de mayor d_{ijk}). LPT puede ser más adecuado para sistemas donde se busca evitar cuellos de botella y maximizar la utilización de los recursos
- SRPT (*Shortest Remaining Processing Time*): selecciona la operación que tiene por delante el tiempo de procesamiento más corto (RT_{ij} menor). Busca minimizar el Total *Flowtime* (Ft) y el *Maximun Flowtime* (F_{max}).
- LRPT (*Longest Remaining Processing Time*): selecciona la operación que tiene por delante el tiempo de procesamiento más largo (RT_{ij} mayor). Busca minimizar *Makespan* (Z_{max}).

$$RT_{ij} = \sum_{j+1}^{o_j} d_{i,j+1} \quad [21]$$

[3] duración total desde el inicio del primer trabajo o proceso hasta la finalización del último, dentro de un entorno de producción determinado.

- EDD (*Earliest Due Date*): puede ser utilizada únicamente en aquellos problemas con fecha límite de entrega para los trabajos (D_i). Consiste en elegir la operación perteneciente al trabajo con menor fecha de entrega de entre el conjunto de elegibles.
- WINQ (*Work In Next Queue*): se selecciona la operación O_{ijk} con menor cantidad de tiempo de procesamiento pendiente en la máquina donde se realizará la próxima operación ($M_{i,j+1}$) del mismo trabajo. Si ij es la última operación del trabajo, el tiempo de procesamiento pendiente será 0.
- LNRO (*Largest Number of Remaining Operations*): esta heurística selecciona la operación O_{ijk} del conjunto de elegibles cuyo trabajo tiene mayor número de operaciones pendientes.
- COEF: consiste en seleccionar la operación O_{ij} que mayor COEF tenga. Para ello se trata de elegir la que mayor tiempo restante tiene (RT), menor duración (d_{ijk}) y menor tiempo muerto ($rp_{ij} - fp_k$). En el entorno flexible, el tiempo restante se calcula como la suma de todos los tiempos de procesamiento de las operaciones sucesoras de la máquina con menor duración entre el conjunto de alternativas (S.A, 1989).

$$COEF_{ijk} = \frac{RT_{ij}}{d_{ijk} + (rp_{ijk} - fp_k)} \quad [22]$$

- *RANDOM*: consiste en seleccionar de forma aleatoria la operación que se incluirá en la solución. Hay diversas variantes de este enfoque, como asignar diferentes probabilidades a las operaciones para ser seleccionadas o reducir el conjunto de operaciones candidatas mediante criterios heurísticos.
  Aunque estos métodos no suelen proporcionar soluciones óptimas, a veces resultan útiles para generar diversos puntos de inicio.
- CR (*Critical Ratio*): da preferencia a las tareas con menor razón crítica, siendo esta el cociente entre el tiempo disponible para realizar la orden (fecha límite (d_i) menos fecha actual) y el plazo estimado para finalizar la orden o tiempo restante, *remaining time* (RT_{ij}).

A continuación, se muestran algunas de los métodos que ofrecen mejores rendimientos o que han servido de base para poder seguir avanzando en este campo.

3.5 Asignación máquinas a operaciones

A la hora de resolver el subproblema de asignación de las máquinas a las operaciones (primer nivel), se dispone de un gran número de posibilidades. Estas varían desde la más simple, la cual consiste en asignar, entre las diferentes posibilidades, la máquina que

menor tiempo de procesamiento tarda en realizar la operación; a otras de mayor complejidad, las cuales tienen en cuenta la carga de trabajo de las máquinas. Uno de los modelos de mayor impacto en este ámbito es el propuesto en (Li-Ning Xing *, 2009).

En este modelo, el subsistema de asignación de operaciones a máquinas consta de tres partes: la recolección de datos, la asignación de operaciones y la optimización de la asignación. El flujo de cálculo básico del subsistema se muestra en la Ilustración 10.

En la recopilación de datos, la función se encarga de leer los datos de entrada desde un archivo especificado por FileName.

En la asignación de operaciones, se calcula el número total de operaciones que se deben asignar a las máquinas. Luego se genera un número aleatorio entre 1 y el número total de máquinas para cada una de las operaciones. Quedando así asignada una máquina a cada operación.

En la optimización de la asignación se realiza un proceso iterativo para optimizar la asignación de operaciones. Se recorren todas las máquinas y para cada una, se recorren todas las operaciones. Se ajusta la operación actual para ser asignada a otras máquinas diferentes a la original, y se busca la mejor manera de hacer este ajuste para mejorar el rendimiento de la asignación actual. Si se encuentra un ajuste que mejora el rendimiento, se ejecuta dicho ajuste.

Cabe destacar que se evalúa cada ajuste considerando el rendimiento de la asignación obtenida después de aplicar el ajuste. Este rendimiento (Fit), se calcula tal y como muestra la Ilustración 9, donde $F_2(\alpha)$ representa el balance total de carga entre las máquinas en la asignación α y $F_3(\alpha)$ la carga crítica de la máquina con mayor carga en la asignación α. Como puede observarse en la Ilustración 9, se da mucho más peso a la carga crítica que al balance total.

$$Fit(\alpha) = 0.2 \times F_2(\alpha) + 0.8 \times F_3(\alpha)$$

Ilustración 9. Cálculo del rendimiento

```
% Part one: data collecting
Reading_Input_Data (FileName);

% Part two: operation assignment
Total_Operation_Account = Task_Num * Operation_Num;          % calculate the total operation account.
                % Task_Num denotes task account; Operation_Num denotes the operation account in each job.
for ii = 1: Total_Operation_Account
    RandP = fix (rand (1, 1)) * Machine_Num;                 % generate a random positive integer.
                                                             %Machine_Num denotes the machine account.
    Assign the ii^th operation to the machine RandP;
end

% Part three: assignment optimization
% Let Assignment denotes the initial assignment achieved by part two.
% Let Num denotes the maximal operation number among all machines in the Assignment.
for Iter=1 : Max_Iter                % Max_Iter denotes the maximal iterative times.
    for ii=1 : Machine_Num           % Machine_Num denotes the account of all machines.
        for jj=1 : Num
            Oper = the jj^th operation in the ii^th machine;
            Adjust Oper to the other machines (all machines except the ii^th machine)①;
            Find the best adjustment which has the best performance②;
            if (This best adjustment can improve performance of the giving assignment)
                Execute this adjustment;
            end
        end
    end
end
①adjust Oper from Machine A to Machine B, it means that cancel Oper form the assigned operation set of machine A, and add Oper to
  the assigned operation set of machine B.
②please note, we evaluate each adjustment through evaluating the achieved assignment after this giving adjustment.
```

*Ilustración 10. Subsistema para el cálculo del flujo, asignación de operaciones a máquinas (Li-Ning Xing *, 2009).*

3.6 Planificación separada mediante programación lineal

La programación lineal es un enfoque matemático que busca optimizar una función objetivo sujeta a un conjunto de restricciones lineales. Es común el uso de programación lineal para resolver problemas mediante asignación separada.

En el modelo que se presenta a continuación, (Juan Carlos Osorio*, 2007) se tiene un modelo matemático subdividido en niveles. Estos modelos a menudo se conocen como "monolíticos" y se destacan por incorporar una modelación matemática considerablemente compleja, no solo en su formulación, sino principalmente en el proceso de encontrar la solución. El planteamiento del enfoque separado se presenta como una alternativa sustancial para abordar el problema del *Job Shop Flexible*.

El modelo busca resolver el problema mediante la definición de dos niveles, cada uno de los cuales tiene asociado diferentes problemas de toma de decisiones. Pero antes de centrarse en estos dos niveles se citan una serie de consideraciones a tener en cuenta:

- Cada uno de los “i” trabajos están disponibles para empezar a ser procesados en el instante t=0.

- Todas las máquinas están libres y pueden comenzar a procesar operaciones en t=0.
- No se permite la interrupción de las operaciones en las máquinas. Una vez inicia una operación en una máquina ésta solamente queda disponible hasta que la operación finaliza.
- Existe recirculación.
- Hasta que una operación no se haya procesado totalmente, esta no se podrá considerar libre para realizar ningún otro trabajo.
- Todos los trabajos tienen la misma prioridad.
- Una vez se procesa una operación en cualquier máquina, ésta automáticamente queda disponible para recibir la siguiente operación. No existen tiempos de alistamiento.
- **Todas las máquinas pueden realizar todas las operaciones** (flexibilidad total), pero con tiempos de procesamiento distintos.
- El tiempo total de procesamiento para un trabajo "I" es la suma del tiempo de proceso de cada una de las operaciones que lo componen.
- Existe restricción de secuencia en las operaciones de cada uno de los trabajos. La operación O_{i2} solamente podrá comenzar a ser procesada una vez finalice completamente la operación O_{i1}.

3.6.1.1 Nivel superior

En este nivel, el problema se centra en asignar los "n" trabajos a los "t" centros de trabajo de manera que se minimice la suma de los tiempos de ejecución. Además, se busca un equilibrio en la carga de los centros de trabajo para evitar la sobrecarga de uno solo. Si se diese la situación de tener un centro sobrecargado, aumentaría el *makespan* considerablemente. Por lo tanto, el objetivo en este nivel es lograr que todos los centros tengan una carga similar para garantizar tiempos de terminación equitativos y, por ende, un mejor valor de *makespan*.

En este nivel se encuentran dos agregaciones:

Agregación de las operaciones en trabajos; se considera que el tiempo de procesamiento total de un trabajo es el equivalente a la suma de los tiempos de procesamiento de todas las operaciones que lo conforman, sin tener en cuenta tiempos de alistamiento ni paradas en el proceso.

Agregación de máquinas en centros de trabajo; se agregan máquinas conforme a los valores máximo y mínimo de los tiempos de ejecución de los trabajos en cada una de ellas. La propuesta es formar grupos en un mismo centro de trabajo que combinen máquinas con valores mínimos y máximos, es decir, un emparejamiento selectivo. Esto se busca para lograr una distribución equitativa de la variación, permitiendo que, al asignar los trabajos, las operaciones tengan la oportunidad de ejecutarse en tiempos breves en cualquiera de los centros de trabajo. Esta agregación no solo incorpora las

máquinas, sino también las operaciones dentro de los trabajos. De hecho, es a partir del resultado de esta agrupación que se configuran los centros de trabajo.

3.6.1.1.1 Modelo de programación lineal para el nivel superior

Tenemos la siguiente función, donde Z es el *makespan*:

$$Z_{max} = Max \sum_{i=1}^{n} \sum_{t=1}^{l} P_{ti} * X_{ti} \ [23]$$

Donde P_{ti} son los tiempos de procesamiento del trabajo i en el centro de trabajo t. X_{ti} es una variable binaria, que toma el valor 1 si el trabajo i se realiza en el centro de trabajo t; en caso contrario tiene el valor de 0.

Por lo que se busca minimizar la función Z; para ello se construye el siguiente modelo que cuenta con una serie de restricciones:

$Min\ Z$

$s.a$:

$$Z \geq \sum_{i=1}^{n} P_{ti} * X_{ti} \qquad \forall t/t = 1 \dots l \ \ [24]$$

$$\sum_{t=1}^{l} X_{ti} = 1 \qquad \forall \frac{i}{i} = 1 \dots n \ [25]$$

$$X_{ti} \begin{cases} 1 & si\ i \in t \\ 0 & si\ i \notin t \end{cases} \ [26]$$

EL modelo busca minimizar el *makespan* dentro del taller. Para que el programa cumpla con un conjunto de restricciones lógicas, se deben tener en cuenta una serie ecuaciones. La ecuación número [24] hace que se tenga un balance entre los centros de trabajo y la [25] garantiza que cada trabajo sea asignado a un único centro.

Una vez se tienen asignados los trabajos a los centros de trabajo, se puede comenzar con el segundo nivel.

3.6.1.2 Nivel inferior

En este nivel se tratan los centros de trabajo a nivel de máquinas; y los trabajos son tratados como un conjunto de operaciones. Una vez se realiza esta desagregación, se obtienen L subproblemas que consisten en la asignación de las operaciones que forman los trabajos (asignados en el nivel superior) a las máquinas que se encuentran en el

centro mismo de trabajo que las operaciones. Una vez están asignadas las operaciones a las máquinas, se procede a secuenciar estas operaciones.

3.6.1.2.1 Modelo de programación lineal para el nivel inferior

Se trata de minimizar el *makespan* (Z), siendo este:

$$Z_{max} = Max \sum_{i=1}^{n} \sum_{j=1}^{o} \sum_{k=1}^{m} p_{ijk}\, x_{ijk} \quad [27]$$

Donde p_{ijk} es el tiempo que tarda la operación j del trabajo i en ser procesada en a máquina k. x_{ijk} es la variable de decisión que representa si la operación j del trabajo i se lleva a cabo en la máquina k o no.

Se tienen el siguiente modelo sujeto a una serie de restricciones:

$Min\ Z$ [28]

$s.a$:

$$Z \geq \sum_{i=1}^{n} \sum_{j=1}^{o} P_{ijk} * X_{ijk} \qquad \forall k/k = 1 \dots m \quad [29]$$

$$\sum_{k=1}^{m} X_{ijk} = 1 \qquad \forall j/j = 1 \dots o, \forall i/i = 1 \dots n \quad [30]$$

$$X_{ijk} \begin{cases} 1 & si\ O_{ijk}\ se\ realiza\ en\ M_k \\ 0 & de\ lo\ contrario \end{cases} [31]$$

La ecuación [29] garantiza que se tenga un balance de carga entre las máquinas y además que se tenga un *makespan* mínimo. La ecuación [30] garantiza que todas las operaciones estén asignadas a alguna máquina y solo a una de ellas.

Al finalizar con el nivel inferior, se tienen como resultado la asignación de los trabajos a los centros de trabajo, las operaciones a los trabajos, y las máquinas a las operaciones.

3.7 Métodos de descomposición

Una alternativa para abordar problemas de optimización de gran escala implica subdividirlos en varios subproblemas más pequeños. Estos subproblemas se pueden resolver de manera más sencilla, y una vez obtenidas sus soluciones, se integran para alcanzar una solución al problema inicial. En general, estos métodos no logran el óptimo de forma inmediata, sino que resulta de un proceso iterativo. En el contexto actual, existen tres métodos específicos propuestos para afrontar este tipo de problemas: la descomposición de Dantzing-Wolfe, la descomposición de Benders y la Relajación Lagrangiana.

3.7.1 Descomposición de Dantzing-Wolfe

La descomposición de Dantzing-Wolfe es una técnica utilizada en problemas matemáticos de programación lineal. La idea detrás de esta técnica es dividir un problema grande en partes más pequeñas y manejables para facilitar su resolución.

En primer lugar, se debe hablar de dos conceptos clave: el teorema de Caratheodory y la teoría de la dualidad. El teorema de Caratheodory establece que las soluciones factibles de un problema pueden expresarse como combinaciones de puntos extremos y direcciones extremas. La descomposición de Dantzing-Wolfe aprovecha este teorema para reformular el problema en un conjunto de problemas más pequeños llamados "subproblemas" y un problema principal llamado "problema maestro".

La idea es simplificar la resolución del problema maestro al reducir el número de restricciones. Sin embargo, construir este problema maestro requiere conocer todos los puntos extremos y direcciones extremas de cada conjunto de restricciones, lo cual puede resultar un gran desafío.

La descomposición de Dantzing-Wolfe no solo se aplica a problemas de programación lineal estándar, sino que también puede adaptarse para resolver problemas más complejos, como el *Job Shop Scheduling Problem*.

La resolución del JSP utilizando la descomposición de Dantzing-Wolfe implica resolver subproblemas asociados a cada orden de operaciones, obtener soluciones globales mediante el problema maestro y ajustar estas soluciones basándose en la información dual. Este enfoque iterativo ha sido estudiado en profundidad y se ha demostrado efectivo para problemas del tipo "*Combinatorial Allocation Problem*" (CAP), tipo del que el JSP forma parte.

3.7.2 Descomposición de Benders

La descomposición de Benders es una metodología alternativa a la descomposición de Dantzing-Wolfe en la programación lineal. A diferencia de la Dantzing-Wolfe, Benders sigue un enfoque opuesto. En lugar de obtener soluciones duales mediante la resolución de subproblemas, Benders propone soluciones a través de la resolución del problema maestro dual.

El proceso de Benders se presenta como un procedimiento iterativo, donde la resolución del problema maestro dual proporciona soluciones, y los subproblemas generan soluciones duales que son utilizadas por el problema maestro para proponer nuevas soluciones. Al igual que la Descomposición de Dantzing-Wolfe, la descomposición de Benders ha demostrado ser aplicable al JSP.

3.7.3 Relajación Lagrangiana

La Relajación Lagrangiana es una técnica utilizada en problemas de optimización para simplificar restricciones. Se considera una "relajación" cuando el conjunto de soluciones factibles de un problema original A está contenido en el conjunto de soluciones factibles de un problema B, y la función objetivo de B es menor o igual a la de A. Esta técnica se puede aplicar a problemas de optimización entera y mixtos, siendo particularmente exitosa en la programación *Job Shop*.

En el proceso de Relajación Lagrangiana, se sustituye un conjunto de restricciones por una combinación lineal de ellas, incorporándola en la función objetivo y penalizando la violación de estas restricciones. Implica simplificar restricciones para hacer el problema más fácil de resolver. Presentada como un problema matemático, su aplicabilidad se extiende tanto a problemas lineales como no lineales. Al "relajar" ciertas restricciones, se vuelve el problema más manejable. La resolución de este problema simplificado se realiza de forma iterativa, paso a paso. Este proceso nos lleva a una solución que, aunque puede no ser óptima, es más fácil de obtener y proporciona información útil sobre el problema original.

3.8 Métodos de mejora o metaheurísticas

Bajo la categoría de "métodos de mejora", también conocidos como "búsqueda por entornos" o metaheurísticas, se engloba la mayoría de los enfoques propuestos en las últimas décadas. Estos métodos se fundamentan en la búsqueda de soluciones refinadas mediante pequeñas modificaciones, denominadas "movimientos básicos", aplicadas a una solución existente (Mattfeld, 1995). La implementación de estos métodos requiere

una solución inicial, obtenida mediante cualquier técnica constructiva, a partir de la cual se inicia la búsqueda.

Un "movimiento básico" implica la reorganización de la secuencia de actividades programadas en cada máquina. Existen diversas alternativas para realizar estos movimientos básicos, como intercambiar el orden de dos operaciones consecutivas o colocar una operación al principio o al final de la secuencia.

La definición del movimiento básico determina el entorno o vecindario de una solución, siendo el conjunto de soluciones alcanzables a través de todos los movimientos básicos según la definición establecida.

La selección de una solución dentro del entorno para continuar la búsqueda distingue los diversos métodos disponibles. Inicialmente, podría parecer lógico elegir la mejor solución posible y, si no mejora la solución original, dar por concluida la búsqueda. Sin embargo, este enfoque, denominado "mejora iterativa", podría limitar la búsqueda a óptimos locales, distantes del óptimo global del problema.

El JSP implica programar un conjunto de operaciones con tiempos de procesamiento conocidos en m máquinas. Un grupo de “o” operaciones forma un trabajo, y hay “n” trabajos definidos. El objetivo es encontrar una permutación de las operaciones que minimice el *makespan*.

En un entorno de Taller Abierto, no existe un orden de procesamiento tecnológico prescrito entre las operaciones de un trabajo, lo que permite hacer todo tipo de permutaciones. En cambio, en un entorno de *Job Shop*, las operaciones de cada trabajo deben pasar por las máquinas en un orden determinado. (Christian Bierwirth, 1996)

Para evitar representar soluciones no factibles en problemas JSP, se propone un esquema de permutación ligeramente modificado. En lugar de hacer permutaciones entre las o operaciones se hacen permutaciones entre los n trabajos; lo cual asegura el orden lógico de las operaciones. Por ejemplo, en lugar de intercambiar el orden de la operación $O_{4,6}$ por la $O_{3,5}$, se intercambiaría el trabajo J_4 por el J_3 y se harían las operaciones que procedieran en cada uno de ellos (la inmediatamente posterior a la ya procesada de cada trabajo).

3.8.1 Metaheurísticas basadas en poblaciones

Las metaheurísticas basadas en poblaciones forman parte de los algoritmos de optimización, las cuales difieren significativamente de las estrategias de búsqueda tradicionales. En lugar de partir de una única solución y modificarla iterativamente (como en las metaheurísticas de trayectoria), estas técnicas se fundamentan en la gestión y evolución de un conjunto diverso de soluciones, denominado población. Esta población se somete a procesos de modificación y combinación en cada iteración, dando

lugar a una dinámica evolutiva que busca mejorar colectivamente el rendimiento del conjunto (Heredero, 2022).

La esencia de estas metaheurísticas radica en la capacidad de aprovechar la "información de grupo", donde la interacción y combinación de múltiples soluciones contribuyen a explorar de manera más eficiente el espacio de búsqueda.

A medida que evolucionan, las soluciones menos prometedoras pueden ser reemplazadas por otras más prometedoras, fomentando la convergencia hacia regiones de alta calidad en el espacio de soluciones.

Este enfoque poblacional no solo permite una mayor diversidad en la búsqueda, sino que también ofrece la flexibilidad de adaptarse a una variedad de problemas de optimización. La convergencia hacia soluciones óptimas y la capacidad de escapar de óptimos locales son características clave de estas metaheurísticas (Ilustración 11. Búsqueda basada en poblaciones.

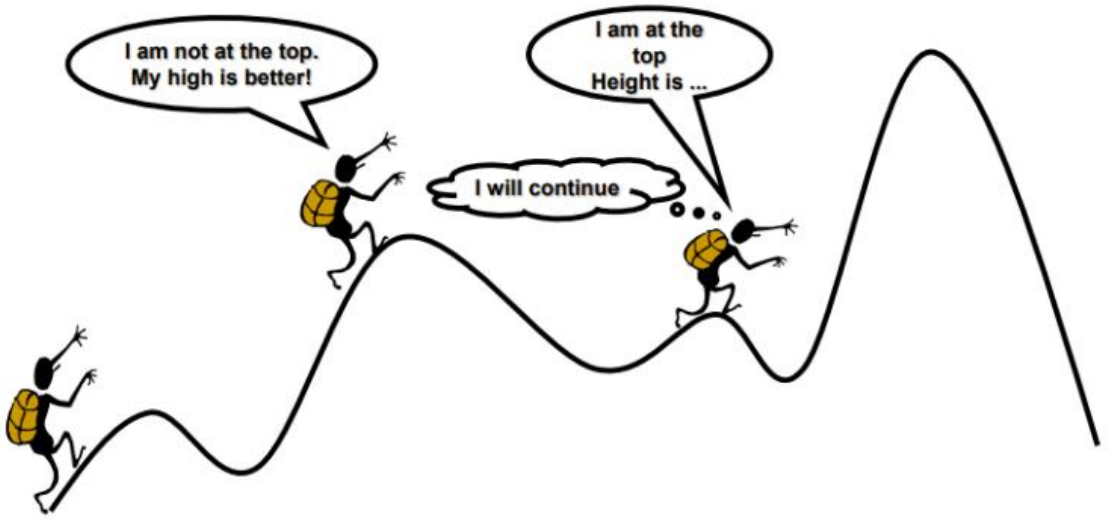

Ilustración 11. Búsqueda basada en poblaciones (II, 2022)

3.8.1.1 Algoritmo genético

Los algoritmos genéticos son un tipo de metaheurística que se basan en principios inspirados en la evolución biológica. Utilizan conceptos como selección, cruce y mutación para guiar la búsqueda en el espacio de soluciones y encontrar soluciones aproximadas a problemas complejos de optimización.

Dado que los algoritmos genéticos no se centran en la construcción secuencial de programas o en la selección de operaciones específicas, sino más bien en la búsqueda de soluciones óptimas en un espacio de búsqueda, su papel se acerca más a la heurística que al esquema de construcción del programa. Sin embargo, es importante tener en cuenta que los algoritmos genéticos pueden ser utilizados como una herramienta

poderosa en combinación con esquemas de construcción de programas y otras heurísticas para abordar problemas complejos de optimización en diversas áreas, incluida la programación de la producción.

La metaheurística del algoritmo genético ha demostrado ser eficaz en la resolución de problemas complejos de optimización, incluido el FJSP.

A continuación, se presenta la metaheurística propuesta por (F. Pezzellaa, 2008) aplicada a FJSP que mejora algunas estrategias ya conocidas en la literatura y las combina para encontrar los mejores criterios en cada paso del algoritmo. Este algoritmo se basa en el [Modelo de planificación separada] visto con anterioridad y sigue los pasos reflejados en la Ilustración 12.

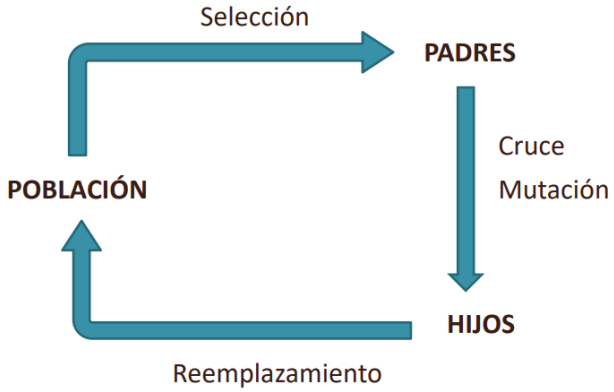

Ilustración 12. Fases algoritmo genético

3.8.1.1.1 Codificación

Los cromosomas contienen la información de que máquina va asignada a cada operación. El orden de estos cromosomas representa el orden en el que se realizan las operaciones. La combinación de estos dos tipos de codificación representa la solución final.

Para representar las soluciones se utiliza un *string*[4] e iría representada de la manera representada en la Ilustración 13.

(1,1,3)	(1,2,1)	(2,1,3)	(2,2,4)	(3,1,3)	(1,3,2)	(3,2,1)	(2,3,4)

Ilustración 13. Representación de un cromosoma

La longitud del *string* se corresponde con el número de operaciones total.

El primer número de cada porción del *string* representa el trabajo "i"; el segundo número a la operación "j"; y, por último, "k" representa en qué máquina se realiza la operación O_{ij}, tal y como se muestra en la Ilustración 14.

[4] Es un tipo de dato que contiene una secuencia de elementos y se utiliza para representar texto.

$$S = (O_{11}, M_3), (O_{12}, M_1), (O_{21}, M_3), (O_{22}, M_4), (O_{31}, M_3), (O_{13}, M_2), (O_{32}, M_1), (O_{23}, M_4)$$

Ilustración 14. Representación lógica de una posible solución

3.8.1.1.2 Población inicial

En el **primer nivel**, los cromosomas iniciales son obtenidos a partir de mínimo global y permutación aleatoria de máquina y trabajo. (Asignación de máquinas a operaciones)

	M1	M2	M3	M4
O11	7	5	8	4
O12	3	3	2	7
O13	9	11	3	1
O21	8	4	7	10
O22	7	2	6	9
O31	4	4	5	8
O32	6	12	9	7

	M1	M2	M3	M4
O11	7	5	8	5
O12	3	3	2	8
O13	9	11	3	1
O21	8	4	7	11
O22	7	2	6	10
O31	4	4	5	9
O32	6	12	9	8

	M4	M1	M2	M3
O21	11	8	4	7
O22	10	7	2	6
O31	9	4	4	5
O32	8	6	12	9
O11	5	7	5	8
O12	8	3	3	2
O13	1	9	11	3

Ilustración 15. Representación de una iteración para generar la solución inicial.

La Ilustración 15 representa una iteración, este proceso se repetiría tantas veces como operaciones tenga. En el resultado final, tras "o" iteraciones, tendríamos todas las operaciones asignadas a una máquina.

Después de asignar la operación "ij" a la máquina "k" a partir de mínimo global, debemos actualizar los tiempos en los que pueden comenzar el resto de las operaciones en la máquina que se acaba de asignar.

En el ejemplo de la Ilustración 15, el mínimo global es 1 unidad de tiempo; por lo que se asigna O13 a M4 y al resto de operaciones de M4 se les suma el tiempo de procesamiento de la operación elegida.

Es importante remarcar que después de cada iteración se debe permutar los trabajos (y no las operaciones) y las máquinas. Esto se hace porque para encontrar el mínimo global ejecutamos un barrido de arriba hacia abajo y de izquierda a derecha. En caso de encontrar más de un tiempo de procesamiento mínimo, nos quedaremos con el localizado en primer lugar.

En el **segundo nivel,** una vez se hayan establecido las asignaciones, debemos determinar cómo secuenciar las operaciones en las máquinas.

En este algoritmo genético, la secuenciación de las asignaciones iniciales se obtiene mediante una combinación de tres reglas de despacho:

- *Randomly select a job (Random)*, la cual añade un componente aleatorio a la elección del trabajo.

- *Most Work Remaining* (MWR), en la que se elige la operación que tiene tiempo de procesamiento total restante por delante en comparación con las demás operaciones disponibles.
- *Most number of Operations Remaining* (MOR), en la que se elige la operación que tiene mayor número de operaciones restante por delante en comparación con las demás operaciones disponibles.

Los porcentajes de influencia de cada uno de estos tres métodos de secuenciación de las operaciones puede variar a medida que se ejecutan más iteraciones. Por ejemplo, en las primeras iteraciones se le puede dar más peso al componente aleatorio para explorar un espacio de soluciones más amplio.

3.8.1.1.3 Evaluación aptitud

Para comparar las distintas soluciones obtenidas, este algoritmo se fija en el makespan de cada cromosoma.

3.8.1.1.4 Selección padres

En un algoritmo genético, los "padres" son los individuos seleccionados entre un conjunto de soluciones para crear la siguiente generación de individuos a través de procesos de reproducción, como el cruce y la mutación. La selección de los padres es un paso crucial, ya que los nuevos individuos, conocidos como “hijos”, heredarán parte de las características de sus padres, lo que influye en la diversidad y calidad de la población en las generaciones futuras.

En el algoritmo propuesto por (F. Pezzellaa, 2008), en cada iteración se seleccionan a los padres mediante la combinación de tres reglas de decisión:

- *Binary tournament*, en la que se eligen a dos padres aleatoriamente entre el conjunto de cromosomas del que disponemos. El padre que tenga la mejor solución será seleccionado como progenitor.
- *n-Size tournament*, es similar al *binary tournament*, pero en lugar de seleccionar a dos candidatos a padres al azar, se seleccionan “n”, habiendo fijado n anteriormente.
- *Linear ranking*, se ordenan los cromosomas de peor a mejor solución obtenida, siendo S_1 el peor cromosoma obtenido y S_n el mejor.

$$p_i = \frac{2 * r_i}{N(N+1)}; \quad i = 1, \dots, N; \quad r_i \in \{1, \dots, \mathrm{N}\}\ [32]$$

Siendo p_i la probabilidad de elegir al cromosoma situado en el puesto r_i.

3.8.1.1.5 Generación hijos

Una vez se han seleccionado los cromosomas padres, se aplican operadores genéticos de cruce y mutación para crear nuevos cromosomas denominados hijos. El operador de cruce se aplica a pares de cromosomas, mientras que el operador de mutación se aplica a cromosomas individuales.

Se distinguen dos tipos de operadores genéticos:

Los **operadores de asignación** únicamente modifican qué máquinas van asignadas a cada operación, es decir, la forma en la que las operaciones van ordenadas se mantiene en los hijos. El operador de cruce en asignación genera la descendencia intercambiando la asignación de un subconjunto de operaciones entre los dos padres. La mutación de asignación únicamente intercambia la asignación de una sola operación de un padre. En ambos casos, las operaciones a intercambiar se eligen arbitrariamente. En la mutación inteligente, seleccionamos una operación en la máquina con la carga de trabajo máxima y la asignamos a la máquina con la carga de trabajo mínima, siempre y cuando sea compatible.

Los **operadores de secuenciación** solo cambian el orden de las operaciones en los cromosomas parentales, mientras que la asignación de operaciones a máquinas se conserva en los hijos. Al aplicar los operadores de secuenciación, debemos respetar las restricciones de precedencia entre las operaciones del mismo trabajo. Adoptar un algoritmo corrector para modificar la descendencia no viable es muy costoso computacionalmente, por lo que es preferible diseñar operadores de modo que no se violen las restricciones de precedencia.

La fase de generación de hijos finaliza en el momento en el que se alcanza el número máximo de individuos en el grupo de apareamiento, es decir, se encuentra una nueva generación.

El algoritmo finaliza cuando se alcanza un número máximo de generaciones. El mejor cromosoma hijo se proporciona como salida.

3.8.1.2 Algoritmo de la colonia de hormigas (ACO)

El algoritmo de optimización de colonias de hormigas (ACO) fue propuesto por el científico milanés Marco Dorigo en 1992 (Dorigo, 1992). Como muchos de los algoritmos genéticos, este algoritmo encuentra su origen en la misma naturaleza, en este caso en el comportamiento de las hormigas. Las hormigas son capaces de liberar unas sustancias químicas denominadas feromonas; gracias a esta sustancia las hormigas son capaces de elegir el camino más corto que han encontrado otras previamente.

El comportamiento en plena naturaleza de las hormigas es el siguiente: cuando una hormiga llega a un punto del que sale más de un camino tiene que tomar la decisión de por cuál de ellos ir. Inicialmente desconoce cuál de ellos es el mejor, por lo que las

hormigas se empezaran a repartir de forma equitativa entre ambas opciones. Esto implica que, bajo la premisa de que todas las hormigas se desplazan a la misma velocidad y salen equitativamente por cada camino a intervalos regulares, el camino más corto acumulará una mayor cantidad de feromonas con el tiempo. En consecuencia, con el paso del tiempo, naturalmente se tenderá a pasar por alto el camino más largo para las futuras hormigas, ya que la cantidad de feromonas en el camino corto será significativamente mayor.

En los estudios del algoritmo ACO, diversos expertos han propuesto numerosos algoritmos mejorados en los últimos años. Estos incluyen enfoques continuos modificados, algoritmos multi-objetivo, algoritmos co-evolutivos, algoritmos autocontrolados, y otras variantes híbridas. Aunque estos métodos han mejorado la resolución de problemas de optimización complejos, como el FJSP, aún enfrentan desafíos como la convergencia lenta y la propensión a caer en valores óptimos locales. El trabajo propuesto por (Wu Deng, 2019) incorpora estrategias de multipoblación, mecanismos de coevolución y estrategias de actualización y difusión de feromonas para lograr un rendimiento de optimización superior.

El algoritmo ACO consta de varias iteraciones en las cuales las hormigas construyen soluciones completas utilizando experiencias previas de poblaciones anteriores. Las experiencias se representan mediante un rastro de feromonas depositado en los elementos de la solución. La regla de actualización de feromonas se divide en reglas de transición y actualización de feromonas.

En la regla de transición, cada hormiga se mueve de un estado a otro de manera iterativa, seleccionando su próximo estado con una probabilidad determinada por la concentración de feromonas y la longitud del recorrido. La actualización de feromonas incluye la actualización local y global, afectando las soluciones "buenas" y "malas". Además, se introduce un mecanismo de difusión de feromonas para ampliar el rango de búsqueda.

Para mejorar la calidad de las soluciones, se propone el algoritmo ICMPACO, que incorpora estrategias de multipoblación, coevolución, actualización y difusión de feromonas. La estrategia de multipoblación divide las hormigas en élite y comunes para mejorar la tasa de convergencia y evitar óptimos locales. La estrategia de actualización de feromonas mejora la capacidad de optimización, y el mecanismo de difusión de feromonas libera gradualmente feromonas en regiones adyacentes. La coevolución intercambia información entre subpoblaciones, mejorando el rendimiento de la optimización.

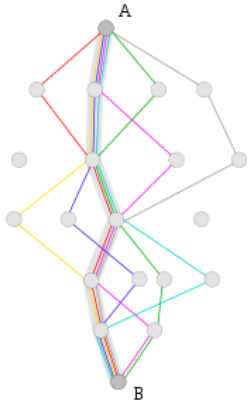

Ilustración 16. Representación gráfica ACO

El algoritmo de colonia de hormigas propuesto (Dorigo, 1992) consta de los siguientes pasos:

3.8.1.2.1 Inicialización

1. Se selecciona el número de hormigas a emplear.
2. Se inicializa el número de feromonas de cada camino (τ_{ij}) a un valor pequeño e igual, por ejemplo 0,1.
3. Para cada uno de los caminos se ha de conocer la longitud o coste del mismo, siendo 1/(longitud ij) lo que se denomina visibilidad del camino (η_{ij}).
4. Se definen parámetros de uso a lo largo de todo el algoritmo, siendo alfa (α) la importancia relativa de la cantidad de feromonas y beta (β) y la visibilidad. La tasa de evaporación (ρ) es el parámetro que marcará el ritmo de desaparición de las feromonas y el último parámetro será la tasa de aprendizaje (Q).

3.8.1.2.2 Construcción de soluciones

A cada una de las hormigas se le asigna un nodo de partida, pudiéndose determinar de forma aleatoria para cada una o igual para todas.

Desde el nodo de partida (i) de la primera hormiga se aplica la siguiente fórmula teniendo como destino cualquiera de los nodos (j) adyacentes.

$$P_{ij} = \frac{(\tau_{ij})^2 * (\eta_{ij})^2}{\sum(\tau_{ij})^2 * (\eta_{ij})^2} \quad [33]$$

La fórmula [33] representa el cálculo del producto del número de feromonas existentes en un camino por su visibilidad dividido entre el sumatorio de esa misma cuenta para todas las opciones posibles (se realiza una ponderación del valor de cada una de las opciones).

Por ejemplo, si partes de la ciudad A y tienes tres ciudades (B, C y D) a las que puedes ir; se tendrá que calcular 3 probabilidades distintas en las que el denominador de la fórmula será siempre el mismo valor.

1 Una vez hallada la probabilidad ij se empleará un número aleatorio y una ruleta con sectores proporcionales para decidir cuál va a ser el camino que va a tomar la hormiga, de esa forma se decidirá cuál es el próximo nodo del recorrido.
2 Una vez que se ha llegado al siguiente nodo habrá que replicar el proceso desde este. El proceso deberá replicarse en tantos nodos como sea necesario hasta que se llegue al nodo final estipulado o se cumpla el objetivo marcado.
3 Habiéndose llegado al final del camino, habrá que calcular el coste del recorrido seguido por la hormiga 1.
4 Una vez se tiene el coste total de la hormiga 1 se repetirá el paso de la construcción de soluciones tantas veces como hormigas se haya estipulado.

3.8.1.2.3 Actualización de feromonas

Una vez que todas las hormigas han generado una solución es el momento de actualizar el número de feromonas de cada camino. La fórmula para actualizar el número de feromonas se empleará para todos los caminos, aunque no haya pasado ninguna hormiga por este.

$$\tau_{ij}{}^{*} = (1-\rho) * \tau_{ij} + \sum \Delta\tau_{ij} \; [34]$$

Como puede verse en la ecuación [34], el primer término va a ser general para cualquier camino, ya que sencillamente representa la desaparición natural de las feromonas de un camino. El segundo término consiste en el sumatorio de las feromonas adicionadas por las hormigas en ese camino.

El termino $\Delta\tau_{ij}$ representa las feromonas generadas por una única hormiga, siendo un valor no común para las hormigas, este se calculará de la siguiente manera:

- Si la hormiga x no atraviesa dicho camino su $\Delta\tau_{ij}$ será igual a 0.
- Si la hormiga x si atraviesa dicho camino sus feromonas tomarán el valor indicado en la ecuación [35]

$$\Delta\tau_{ij} = \frac{Q}{Coste\ Total\ Camino\ Hormiga\ X} \; [35]$$

Una vez se han calculado las nuevas feromonas correspondientes a cada camino se termina este paso.

3.8.1.2.4 Convergencia y solución final:

El algoritmo continúa repitiendo los pasos de construcción y actualización de feromonas hasta que se alcanza un criterio de parada, por ejemplo, un número máximo de iteraciones o una mejora mínima en la mejor solución encontrada.

La solución final se obtiene tomando el camino que haya sido recorrido por la hormiga que haya encontrado la mejor solución durante la búsqueda.

3.8.1.2.5 ACO en FJSP

En el contexto del FJSP, el objetivo es optimizar la asignación de operaciones a máquinas, teniendo en cuenta restricciones como tiempos de procesamiento, restricciones de secuencia y la capacidad de las máquinas. El ACO aborda este problema utilizando una colonia de hormigas virtuales que construyen soluciones iterativamente.

La aplicación del ACO al FJSP ha demostrado ser efectiva para encontrar soluciones de alta calidad en un tiempo razonable. Al adaptar las reglas de transición y la función objetivo a las características específicas del FJSP, el ACO puede abordar de manera eficiente problemas de programación de talleres flexibles, contribuyendo a la mejora de la eficiencia y la utilización de los recursos en entornos de fabricación complejos.

En la fase de la construcción de soluciones la elección que hace cada hormiga de uno de los posibles caminos representa la asignación de cada una de las operaciones a las diversas posibles máquinas que estas tienen.

Una vez están todas las operaciones asignadas a las máquinas, se procede con la secuenciación de las operaciones. Como tenemos claramente la fase de asignación y la secuenciación, la forma de resolver el problema es mediante el modelo de planificación separada visto con anterioridad. En esta fase cada hormiga construirá una posible solución, siempre respetando las restricciones de precedencia que llevan las operaciones en su naturaleza.

3.8.2 Metaheurísticas por trayectorias

Las metaheurísticas de trayectorias representan una perspectiva dinámica y evolutiva en la exploración del espacio de soluciones. Estas estrategias comienzan con una solución inicial y, a lo largo de iteraciones sucesivas, realizan un análisis minucioso de soluciones vecinas dentro de su entorno inmediato. La clave radica en la capacidad de evaluar y comparar estas soluciones cercanas, con el objetivo de actualizar la solución actual hacia estados más prometedores (Heredero, 2022).

La exploración se centra en ajustes incrementales y modificaciones de la solución actual, permitiendo un análisis detallado de las vecindades. Las metaheurísticas de trayectorias

son especialmente efectivas en problemas donde se busca optimizar soluciones en un espacio continuo, proporcionando buenos resultados en situaciones donde las soluciones cercanas comparten similitudes significativas (Ilustración 17).

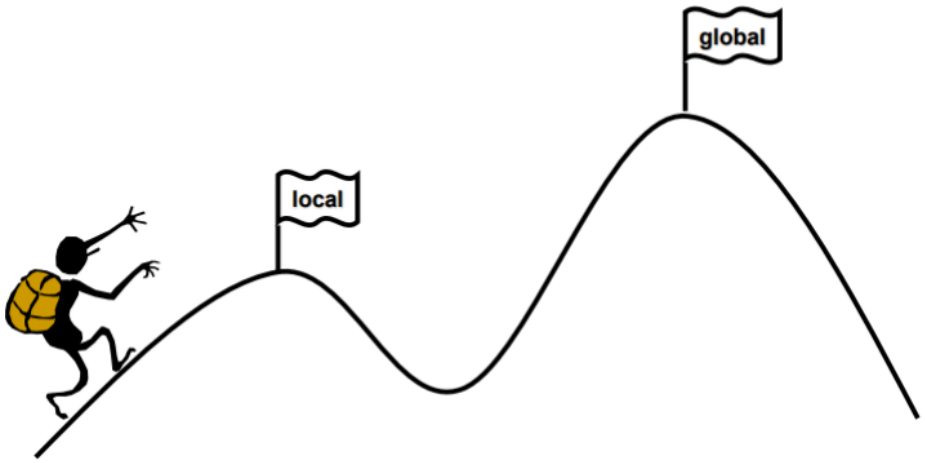

Ilustración 17. Metaheurísticas basadas en trayectorias (II, 2022)

3.8.2.1 Recocido Simulado

Los algoritmos de recocido simulado se inspiran en la metalurgia, específicamente en el proceso de recocido. El recocido es una técnica que transforma sólidos de alta energía en sólidos de baja energía mediante un tratamiento térmico que implica un calentamiento seguido de un enfriamiento lento.

Durante el enfriamiento, las partículas del sólido se reordenan y alcanzan un estado de menor energía. La velocidad de enfriamiento es crucial, ya que un enfriamiento rápido podría impedir el reordenamiento y llevar a un estado final no deseado.

Los algoritmos de recocido simulado imitan este proceso metalúrgico (Yamada, 1996). En este contexto, los estados del sólido se corresponden con las posibles soluciones del problema, y la energía representa la calidad de la solución.

La temperatura se traduce en un parámetro que controla la probabilidad de aceptar soluciones de menor calidad que la solución actual. Si la temperatura es alta, se aceptarán más fácilmente soluciones de peor calidad; si es baja, será menos probable.

El inicio de la búsqueda se realiza a "altas temperaturas", lo que permite explorar un amplio rango de soluciones. A medida que avanza el algoritmo, la temperatura disminuye, de modo que al final, cuando las temperaturas son bajas, solo se consideran mejoras pequeñas en la solución. El control de esta "temperatura" (enfriamiento) establece las condiciones de búsqueda en cada etapa del proceso.

3.8.2.2 Búsqueda Tabú

La metaheurística de búsqueda tabú, una evolución significativa de la de recocido simulado, aborda con eficacia la limitación de este último método al incorporar una

memoria que registra las soluciones previamente exploradas. Este enfoque surge como una respuesta estratégica para evitar la repetición de soluciones y la formación de bucles, proporcionando un mayor control y diversificación en la búsqueda de soluciones óptimas (A. Amuthan and K. Deepa Thilak, 2016).

La esencia de la búsqueda tabú radica en la creación de una lista, conocida como lista tabú, donde se almacenan los movimientos ya realizados y se prohíbe su repetición. Al partir de una solución inicial, el algoritmo se embarca en una serie de movimientos que buscan mejorar la solución actual. Cada paso implica la elección del movimiento que genere una mejora significativa en la solución, y en caso de no existir tal movimiento, se acepta el menos desfavorable. La novedad reside en que cualquier movimiento presente en la lista tabú queda vetado, incluso si pudiera conducir a una mejora inmediata, con el objetivo de evitar la repetición de soluciones y explorar regiones no visitadas del espacio de búsqueda.

Este enfoque de trayectorias de búsqueda tabú se ha destacado por su capacidad para sortear óptimos locales y mantener una exploración robusta y diversificada del espacio de soluciones.

La combinación de exploración intensiva y diversificación controlada lo convierte en una herramienta valiosa para la optimización en problemas complejos y de gran escala.

La efectividad de la búsqueda tabú radica en su habilidad para equilibrar la intensificación y la diversificación, proporcionando soluciones de alta calidad y evitando la caída en mínimos locales.

3.8.3 Heurística de Cuello de Botella Variable

La heurística del cuello de botella variable tiene una orientación específica hacia la optimización de recursos clave en sistemas de producción y no se ajusta fácilmente a la categorización convencional de metaheurísticas de trayectoria o de población.

Uno de los procedimientos más exitosos para minimizar el tiempo total de ejecución en un taller de trabajo es la heurística de Cuello de Botella Variable. (Pinedo, 2009), la cual difiere de la heurística del Cuello de Botella en que esta se actualiza dinámicamente en cada iteración. Esta metaheurística, se centra en identificar la máquina que más afecta el cronograma de operaciones en cada iteración, conocida como la "máquina crítica". Se busca minimizar el tiempo total haciendo ajustes continuos en el cronograma en función de las interrupciones y demoras asociadas con la máquina crítica identificada. Además, se lleva a cabo una resecuenciación adicional al final de cada iteración para optimizar aún más el tiempo total y garantizar una programación más eficiente y efectiva de las operaciones. Este enfoque permite una adaptación dinámica para abordar las interrupciones y los problemas que podrían surgir durante la programación de operaciones en un taller de producción.

4 SELECCIÓN DE LA HERRAMIENTA

4.1 Introducción

En el estudio de alternativas y viabilidad para abordar el problema de programación flexible (FJSP), se consideraron diversas opciones, entre las cuales destacaron C++, Java y Python. La elección del lenguaje de programación para esta tarea es fundamental, ya que impacta directamente en la eficiencia, la facilidad de implementación y el éxito general del proyecto.

La alternativa de implementación en C++ se presentó como una opción atractiva, especialmente por su rendimiento elevado, velocidad en cálculos complejos y la base de conocimiento desde la que se partía. Sin embargo, esta elección también conlleva limitaciones, ya que, para exprimir el verdadero potencial de este lenguaje, se necesita un muy profundo conocimiento, además los cálculos requeridos por el programa no suponen de gran carga computacional.

Por otro lado, la alternativa de Java también se consideró, destacando sus capacidades de rendimiento y su versatilidad. Sin embargo, se identificó como un inconveniente significativo el desconocimiento total del autor sobre este lenguaje. La necesidad de un periodo de formación prolongado podría retrasar el desarrollo del proyecto y afectar la eficiencia en la resolución del Flexible *Job Shop Scheduling* Problem.

En medio de este proceso de toma de decisiones, la opción de Python emergió como una gran alternativa. Aunque inicialmente se encontraba en desventaja debido a la falta de familiaridad del autor con el lenguaje, su naturaleza amigable y legible, junto con una amplia variedad de bibliotecas y *frameworks* disponibles, lo posicionaron como un candidato atractivo.

A través de un análisis detenido de las ventajas y desventajas de cada alternativa, se tomó la decisión de optar por Python, una elección que implicó superar ciertos

obstáculos, como el profundo aprendizaje de esta poderosa herramienta, pero que finalmente se reveló como la opción más flexible. Python es un lenguaje accesible, permitiendo una rápida asimilación de conceptos y una implementación eficiente del Flexible *Job Shop Scheduling* Problem.

En resumen, la elección de Python para abordar el Flexible *Job Shop Scheduling Problem* fue el resultado de un proceso de evaluación y adaptación. A pesar de las consideraciones iniciales en contra, la decisión de aprovechar la versatilidad de Python permitió no solo superar las limitaciones de conocimiento, sino también ofrecer una solución eficiente y robusta a un problema complejo de programación. La flexibilidad de Python se convirtió en la clave para el éxito, destacando la importancia de no solo considerar las ventajas y desventajas teóricas, sino también la capacidad de adaptación y aprendizaje del autor.

4.2 Python

En el ámbito de la programación, Python destaca como un lenguaje de gran potencia y accesibilidad. Su sintaxis y su enfoque en la programación orientada a objetos, respaldados por estructuras de datos eficientes a nivel superior y un sistema de tipado dinámico, lo convierten en una elección idónea para el scripting y el desarrollo ágil de aplicaciones en diversas áreas y plataformas. Es de gran importancia que el programa esté orientado a objetos ya que para la resolución del FJSP se utilizan estructuras de datos complejas.

La naturaleza interpretada de Python, unida a la disponibilidad gratuita del intérprete y la extensa biblioteca estándar, facilita su uso y distribución en una amplia gama de plataformas. Tanto el intérprete como la librería estándar están accesibles en código fuente y forma binaria a través de la Web de Python, permitiendo su distribución sin restricciones. Además, la plataforma proporciona acceso a módulos de terceros, programas, herramientas y documentación adicional.

La modularidad de Python facilita la reutilización de código, y su amplia colección de módulos estándar brinda una base sólida para el desarrollo. Al ser interpretado, Python ahorra tiempo de desarrollo al eliminar la necesidad de compilar y enlazar. Esta modularidad hace que se facilite la resolución del problema, implementando librerías de gráficos que pueden ser aprovechadas, como por ejemplo es el caso de la biblioteca Bokeh utilizada para crear el diagrama de Gantt.

Su sintaxis intuitiva, que incluye la agrupación de instrucciones mediante indentación, contribuye a programas más concisos y legibles.

Python se distingue por su capacidad de extensión mediante la implementación de funciones y tipos de datos en lenguajes como C o C++. Esta versatilidad lo convierte en una elección adecuada para la ampliación de aplicaciones modificables.

Python ofrece una alternativa ágil al ciclo de escribir/compilar/probar/recompilar. Su facilidad de extensión y su disponibilidad en sistemas operativos como Windows, macOS y Unix lo han convertido en una elección eficiente para acelerar el desarrollo.

Las siguientes explicaciones no pretenden cubrir cada una de las características del lenguaje, ni siquiera las más utilizadas. En su lugar, van a explicarse las sentencias utilizadas a lo largo del programa para establecer la programación de operaciones de un problema FJSP.

5 IMPLEMENTACIÓN DE MÉTODOS DE RESOLUCIÓN

5.1 Introducción

A continuación, se muestra un programa de elaboración propia diseñado en Python que es capaz de abordar de manera eficiente el problema de la planificación de operaciones en sistemas de producción FJSP.

Este programa, lee datos de un archivo de texto que contiene los datos del problema y genera un diagrama de *Gantt* como salida. Con esta solución, se ofrece una interfaz accesible y eficiente para analizar y visualizar la solución del problema. Además, el programa proporciona una serie de datos como *makespan, flowtime* o porcentajes de asignación de cada una de las máquinas para realizar un análisis, contribuyendo a la mejora de la toma de decisiones en entornos de producción complejos.

La forma en que se resuelve el problema tiene en cuenta las siguientes consideraciones:

- Todos los trabajos están disponibles para comenzar a ser procesados en el instante t = 0, es decir la primera operación de cada trabajo puede comenzar a ser procesada desde un inicio.
- Todas las máquinas se encuentran listas en t = 0.
- No se permite la interrupción de las operaciones en una máquina, esta queda ocupada hasta que la operación finaliza.
- Se permite la recirculación, lo que significa que un trabajo puede visitar una máquina en más de una ocasión. Como se trabaja con el problema flexible, podría llegar a darse el caso en el que un trabajo pueda tener al menos una de las alternativas de las operaciones en la misma máquina para varias operaciones del trabajo.

- Hasta que una operación no haya terminado de ser procesada, la máquina en la que se esté realizando dicha operación no se considerará disponible para ningún otro trabajo.
- No existen centros de trabajo, una operación puede ser procesada en una o varias máquinas; pero estas máquinas no están próximas físicamente. Se debe tener en cuenta en el fichero.txt la manera en la que las máquinas pueden estar ordenadas.
- Todos los trabajos tienen la misma prioridad dentro del sistema y no existen fechas límite de entrega para ningún trabajo, por lo que no se contemplan las heurísticas que trabajan con retrasos y adelantos.
- Una vez que finaliza el proceso de una operación en cualquier máquina, esta automáticamente queda disponible para recibir la siguiente operación.
- Los tiempos de procesamiento de las operaciones en las máquinas son conocidos y determinísticos. Una operación puede ser procesada en más de una máquina con tiempos de procesamientos distintos para cada una de ellas.
- El tiempo total de proceso para un trabajo J es la suma del tiempo de proceso de cada una de las operaciones que lo componen. No se consideran alistamientos ni tiempos de parada.
- Existe restricción de secuencia en las operaciones de cada uno de los "n" trabajos, es decir, que la operación o_{i2} solo puede comenzar a procesarse una vez la operación o_{i1} haya finalizado completamente.
- No se consideran tiempos de transporte para los trabajos entre una y otra máquina.

5.2 Programa FJSP

5.2.1 Definición de clases

Para comenzar con el programa se realiza la definición de las clases (Ilustración 18), divididas en "Modo, "Operación" y "Trabajo". A continuación, se profundiza en cada una de ellas:

La clase trabajo está formada por el atributo operaciones. Representa los n trabajos a procesar dentro del programa.

La clase operación tiene el atributo modos y representa las operaciones a realizar de cada uno de los trabajos.

Por último, la clase modo representa cada una de las alternativas donde se puede realizar una operación; esto ocurre porque se trabaja con problemas flexibles.

La clase modo contiene los siguientes atributos: “máquina” y “duración”, como su propio nombre indica, representa la máquina y el tiempo de procesamiento donde se procesa el modo. Los atributos "r", "b" y "c" representan el momento en el tiempo en el cual el modo puede comenzar a ser procesado, el momento en el cual realmente comienza a ser procesado, y el momento en el cual ha sido completamente procesado, respectivamente.

```python
#Creación de clases

import numpy as np
import random

class Modo:
    def __init__(self, maquina, duracion, r=None, b=None, c=0):
        self.maquina = maquina
        self.duracion = duracion
        self.r = r
        self.b = b
        self.c = c

class Operacion:
    def __init__(self):
        self.modos = []

class Trabajo:
    def __init__(self):
        self.operaciones = []
```

Ilustración 18. Función definición de clases

5.2.2 Función lectura txt

La función lectura se utiliza para leer archivos de texto desde un archivo de texto “.txt”. La función utiliza el contexto *with* para abrir el archivo "xxxxxxx.txt" en modo lectura ('r'). El uso del *with* garantiza que el archivo se cierre correctamente después de su uso, incluso si ocurre una excepción durante la ejecución del bloque de código.

Un ejemplo de archivo txt sería el propuesto en (Brandimarte, 1999), el cual se muestra en la Ilustración 19. Bramdimarte mk04.txt ejemplo.

Format

- First line: `<number of jobs> <number of machines>`
- Then one line per job: `<number of operations>` and then, for each operation, `<number of machines for this operation>` and for each machine, a pair `<machine> <processing time>`.
- Machine index starts at 0.

```
15 8
8 1 0 6 2 0 6 6 9 2 5 7 2 1 2 3 2 6 5 3 0 8 2 9 7 9 3 1 3 3 8 2 2 2 4 5 5 7 2 5 1 3 7
7 1 5 1 2 5 1 3 7 1 0 6 2 5 7 2 1 3 1 3 3 8 2 2 1 5 2 1 6 2
6 1 5 1 3 1 3 3 8 2 2 3 2 2 6 1 3 4 2 3 2 6 5 2 0 7 2 7 2 3 4 2 1
5 1 6 2 1 0 6 2 0 6 6 9 2 5 7 2 1 2 3 5 4 7
7 1 6 2 2 0 6 6 9 2 3 4 2 1 3 0 8 2 9 7 9 2 0 7 2 7 3 1 3 3 8 2 2 2 3 5 4 7
9 1 5 2 2 3 4 2 1 3 2 2 6 1 3 4 2 5 1 3 7 2 3 5 4 7 3 0 8 2 9 7 9 2 0 7 2 7 1 5 1 2 0 6 6 9
5 2 4 5 5 7 2 0 7 2 7 2 5 1 3 7 1 5 2 2 5 7 2 1
6 2 3 5 4 7 2 4 5 5 7 3 1 3 3 8 2 2 1 5 2 1 5 1 2 0 6 6 9
9 1 0 6 2 0 6 6 9 2 3 4 2 1 3 0 8 2 9 7 9 2 3 2 6 5 2 5 1 3 7 1 6 2 2 0 7 2 7 3 1 3 3 8 2 2
5 2 4 5 5 7 1 0 6 1 6 2 2 3 5 4 7 2 0 6 6 9
4 3 0 8 2 9 7 9 1 0 6 3 1 3 3 8 2 2 2 3 2 6 5
6 2 3 2 6 5 1 5 1 1 0 6 2 0 7 2 7 3 0 8 2 9 7 9 1 6 2
4 1 5 2 2 5 7 2 1 2 5 1 3 7 2 4 5 5 7
3 2 4 5 5 7 1 5 1 2 3 2 6 5
6 2 3 5 4 7 1 6 2 3 0 8 2 9 7 9 3 1 3 3 8 2 2 3 2 2 6 1 3 4 1 0 6
```

Ilustración 19. Bramdimarte mk04.txt ejemplo

La Ilustración 20 muestra cómo se lee la primera línea del archivo, que contiene dos números separados por espacio: el número de trabajos y el número de máquinas, respectivamente. La función *split* () divide la línea en palabras y *map* (*int*, ...) convierte esas palabras en números enteros.

A continuación, se itera sobre el número de trabajos y se inicia un bucle para cada trabajo. La línea actual se divide en números enteros y se extrae el primer número de cada línea, el cual indica el número de operaciones de cada trabajo.

Para cada trabajo, se itera sobre el número de operaciones. Se inicializa un objeto Operacion y se llena con objetos Modo. Cada modo representa una operación en una máquina específica con una duración específica.

El bucle que itera las posiciones de cada línea se utiliza para diferenciar qué números son duraciones de máquinas, cuáles son la maquina donde se puede procesar el modo y cuales indican el número de modos de cada una de las operaciones de la línea. Por ejemplo, la línea con los números 5 1 6 2 1 0 6 2 0 6 6 9 2 5 7 2 1 2 3 5 4 7, indicaría que el trabajo estaría compuesto por 5 operaciones, la primera operación se compone de un modo que se realiza en la máquina 6 y tiene una duración de 2 unidades de tiempo. Los números sombreados representan los números de modos de cada operación.

La función retorna una tupla que contiene la lista de trabajos, el número de trabajos, el número de máquinas y el número de operaciones.

```python
#Lee el archivo txt

def lectura():
    with open('mk04.txt', 'r') as archivo:
        num_trabajos, num_maquinas = map(int, archivo.readline().split())
        trabajos = []
        for i in range(num_trabajos):
            linea = [int(num) for num in archivo.readline().split()]
            num_operaciones = linea[0]
            pos=1
            trabajo = Trabajo()
            for j in range(num_operaciones):
                operacion = Operacion()
                for k in range(linea[pos]):
                    maquina = linea[pos + 2*k + 1]
                    duracion = linea[pos + 2*k + 2]
                    modo = Modo(maquina, duracion)
                    operacion.modos.append(modo)
                pos=pos+2*linea[pos]+1
                trabajo.operaciones.append(operacion)
            trabajos.append(trabajo)
        return trabajos, num_trabajos, num_maquinas, num_operaciones
```

Ilustración 20. Función lectura de un txt

5.2.3 Función inicialización de variables

La función mostrada en la Ilustración 21 representa la inicialización de las variables.

En primer lugar, se utiliza la librería *NumPy* para crear dos vectores, S y P. El vector S representa las operaciones programables, donde cada posición corresponde a un trabajo específico. En este punto, el vector P se inicializa vacío, ya que se utilizará posteriormente para almacenar las operaciones que ya han sido realizadas.

Para completar la inicialización de S, se recorre cada trabajo y se asigna la primera operación de cada uno al correspondiente índice en el vector S. Este paso es fundamental, ya que establece la secuencia inicial de operaciones programables para cada trabajo.

Además, se inicializa el atributo r de cada modo de operación en S con el valor 0. Este atributo representa el instante de tiempo en el cual una operación puede comenzar a ser procesada. Este detalle es esencial para garantizar la coherencia temporal en la planificación de operaciones.

Por último, se crea el vector P como un array *NumPy* vacío con tipo de datos Operacion. Este vector se utilizará posteriormente para rastrear las operaciones que ya han sido completadas durante la ejecución del programa.

```
#Inicializar vectores S P. S es el vector de operaciones programables donde la posicion 0 corresponde al trabajo 0,
#la posicion 1 al trabajo 1,...
#El vector P es el vector donde se van guardando las operaciones que ya se han realizado. En la inicialización va vacío

def inicializacion(num_trabajos, num_operaciones):
    import numpy as np
    S = np.empty(num_trabajos, dtype=Operacion)

    for i in range(num_trabajos):
        S[i] = trabajos[i].operaciones[0]

    for operacion in S:
        for modo in operacion.modos:
            modo.r = 0

    P = np.empty (0,dtype=Operacion)

    return S, P
```

Ilustración 21. Función inicialización variables

5.2.4 Función método de construcción

A la hora de seleccionar una máquina se pueden utilizar dos esquemas de construcción del programa, el método de *Giffler y Thompson* y el de Lanzamiento, ambos vistos con anterioridad. Estos esquemas de construcción permiten determinar en qué máquina se va a procesar la próxima operación.

La función del esquema de construcción de Lanzamiento mostrada en la Ilustración 22 se encarga de seleccionar la máquina asociada a la operación que tiene el tiempo mínimo de inicio.

Primero, se inicializa una variable menor_r_duracion con un valor infinito, y se crea una lista vacía llamada operaciones_minimas. Estas estructuras de datos se utilizan para rastrear la operación y el modo que tienen el tiempo mínimo de inicio.

Luego, se realiza un bucle a través de las operaciones y modos en el vector de operaciones programables S y se guardan los modos de menor duración en la lista operaciones_mínimas. Si se encuentra un modo estrictamente menos se debe reiniciar la lista. Esto se hace para que en caso de haber varios modos con la misma duración se guarden todos ellos para posteriormente elegir uno al azar.

Finalmente, se determina en que máquina se realiza la operación que antes puede comenzar y se guarda en la variable MAQ.

```python
#Devuelve la maquina de la operacion que tiene el tiempo minimo,

#Método Lanzamiento

def elegir_maquina_Lanz(S):
    menor_r_duracion = float('inf')
    operacione_minima = None
    for operacion in S:
        for modo in operacion.modos:
            r_duracion = modo.r
            #if r_duracion < menor_r_duracion: -> si ponemos esto se quedara con el pimer menor modo encontrado
            if r_duracion <= menor_r_duracion:
                menor_r_duracion = r_duracion
                operacion_minima = operacion

    FT = Trabajo()
    FT.operaciones.append(operacion_minima)
    MAQ = FT.operaciones[0].modos[0].maquina

    return MAQ
```

Ilustración 22. Función método constructivo Lanzamiento

De forma similar se realiza el esquema de construcción de *Giffler y Thompson*, con la modificación de que en cada iteración se va guardando la operación que antes puede terminar en la lista operaciones_mínimas. Para ello se suma al instante de tiempo en el que la operación puede comenzar la duración de la propia operación en esa máquina (Ilustración 23).

```python
#Devuelve la maquina de la operacion que tiene el tiempo minimo,
#Método G y T

def elegir_maquina_G_y_T(S):
    menor_r_duracion = float('inf')
    operacion_minima = None
    for operacion in S:
        for modo in operacion.modos:
            r_duracion = modo.r + modo.duracion
            if r_duracion < menor_r_duracion:
                menor_r_duracion = r_duracion
                operacion_minima = operacion

    FT = Trabajo()
    FT.operaciones.append(operacion_minima)
    MAQ = FT.operaciones[0].modos[0].maquina

    return MAQ
```

Ilustración 23. Función esquema de construcción de Giffler y Thompson

5.2.5 Función operaciones elegibles

La función de la Ilustración 24 se encarga de guardar en un vector todos aquellos modos que se procesan en la máquina seleccionada en la función anterior.

Se utiliza un bucle anidado para recorrer cada operación en el vector S y luego explorar los modos asociados a cada operación. Se verifica si el modo actual pertenece a la máquina MAQ. En caso afirmativo, se añade ese modo a la lista modos_en_MAQ. Esto garantiza que solo se seleccionen los modos que están destinados a ser procesados en la máquina específica MAQ.

```
# modos_en_MAQ[0] contendrá los modos de S que se procesan en la máquina MAQ

def dividir_por_maquina(S, num_maquinas, MAQ):
    # Inicializa una lista vacía para los modos que se hacen en la máquina MAQ
    modos_en_MAQ = []

    # Itera a través de las operaciones en S
    for operacion in S:
        for modo in operacion.modos:
            if modo.maquina == MAQ:
                modos_en_MAQ.append(modo)

    return modos_en_MAQ
```

Ilustración 24. Función modos en MAQ

5.2.6 Heurísticas

A continuación, hay una serie de funciones que representan algunas de las heurísticas mencionadas con anterioridad. Todas ellas eligen el modo entre el conjunto de elegibles (vector modos_en_MAQ) siguiendo diferentes criterios.

La función de la Ilustración 25 implementa la heurística SPT (*Shortest Processing Time*). La función toma como entrada la lista de los modos de las operaciones que se ejecutan en la máquina MAQ (modos_en_MAQ) y devuelve el modo de operación que tiene la menor duración entre estos modos.

```python
#Nos dice el modo de la MAQ que se va a procesar
#Heurística SPT, se elige la operación con menor duración.

def encontrar_modo_mas_corto(modos_en_MAQ):
    duracion_minima = float('inf')
    modo_minimo = None

    for modo in modos_en_MAQ:
        if modo.duracion < duracion_minima:
            duracion_minima = modo.duracion
            modo_minimo = modo

    return modo_minimo
```

Ilustración 25. Función Heurística SPT

La función de la Ilustración 26Ilustración 25 implementa la heurística LPT (*Longest* Processing *Time*), la cual toma como entrada la lista de los modos de las operaciones que se ejecutan en la máquina MAQ (modos_en_MAQ) y devuelve el modo de la operación que tiene la duración más larga entre estos modos.

```python
#Nos dice el modo de la MAQ que se va a procesar
#Heurística LPT, se elige la operación con mayor duración.

def encontrar_modo_mas_largo(modos_en_MAQ):
    duracion_maxima = float('-inf')
    modo_mas_largo = None

    for modo in modos_en_MAQ:
        if modo.duracion > duracion_maxima:
            duracion_maxima = modo.duracion
            modo_mas_largo = modo

    return modo_mas_largo
```

Ilustración 26. Función Heurística LPT

La función definida en la Ilustración 27 implementa la heurística FIFO (*First In First Out*). Las operaciones a medida que van estando disponibles se van añadiendo en la última posición del vector S como muestra la función de la Ilustración 27. Por lo que la lista de los modos en la máquina MAQ van a estar ordenados de más antiguos a más nuevos. La función que implementa la heurística FIFO únicamente procesa el modo que se encuentra en la primera posición de la lista.

```python
#Nos dice el modo de la MAQ que se va a procesar
#Heurística FIFO, la primera operación que llegue es la que se procesa

def encontrar_modo_fifo(modos_en_MAQ):
    if modos_en_MAQ:
        return modos_en_MAQ[0]
    else:
        return None
```

Ilustración 27. Función Heurística FIFO

La función de la Ilustración 28 representa la función de máximo coeficiente (COEF). Esta función selecciona el modo entre el conjunto de modos que se procesan en MAQ que tiene mayor COEF. Esta regla compuesta elige el modo que tiene mayor tiempo de procesamiento restante (RT_{ij}), menor tiempo de procesamiento (d_{ij}) y menor tiempo muerto ($rp_{ij} - fp_{m_{ij}}$). El coeficiente, tal y como se explica anteriormente, se calcula de la siguiente manera:

$$COEF_{ij} = \frac{RT_{ij}}{d_{ij} + (rp_{ij} - fp_{m_{ij}})} \quad [36]$$

En primer lugar, la función inicializa la variable coef_maximo y modo_maximo.

Se itera sobre cada modo en el conjunto de modos de una máquina específica para obtener el número de operación y al trabajo al que pertenece el modo actual.

Luego se calcula el tiempo restante (RT) sumando las duraciones de los modos restantes en todas las operaciones restantes del trabajo actual a partir de la operación actual.

Para realizar el cálculo del tiempo muerto se debe calcular rp y fp. Para calcular fp se busca dentro del vector de operaciones procesadas P la operación inmediatamente anterior y el modo que ha sido procesado de esa operación obtenemos la fecha fin de procesamiento (c).

En la obtención de rp buscamos la fecha de disponibilidad del modo (r).

Se calcula el COEF de todos los modos y la función devuelve el modo con el coeficiente máximo encontrado durante la iteración.

```python
def encontrar_modo_max_coef(modos_en_MAQ, P, trabajos):
    coef_maximo = float('-inf')
    modo_maximo = None

    for modo in modos_en_MAQ:
        operacion_pertenece = modo.operacion_pertenece
        trabajo_pertenece = modo.trabajo_pertenece

        trabajo = next((t for t in trabajos if any(md.trabajo_pertenece == trabajo_pertenece
                                                   for op in t.operaciones for md in op.modos)), None)

        rt = sum(md.duracion for op in trabajo.operaciones[operacion_pertenece - 1:] for md in op.modos)

        fp = next((modo.c for op in P if any(modo.b != float('inf') and modo.c != float('inf')
                                             and modo.trabajo_pertenece == trabajo_pertenece for modo in op.modos)), 0)

        rp = modo.r


        if modo.duracion + (rp - fp) != 0:
            coef = rt / (modo.duracion + (rp - fp))
        else:
            # Asignar un valor específico o manejar el caso de división por cero según tus necesidades
            coef = float('inf')  # o cualquier otro valor adecuado


        # Actualizar el máximo
        if coef > coef_maximo:
            coef_maximo = coef
            modo_maximo = modo

    return modo_maximo
```

Ilustración 28. Heurística max COEF

5.2.7 Función procesar modos

La función encargada de procesar los modos juega un papel clave dentro del programa (Ilustración 29). En primer lugar, se calcula max_c que representa el instante de tiempo en el que la máquina en la que debe procesarse el modo_mínimo queda disponible. A continuación, se actualiza el tiempo de inicio (b) y el tiempo fin (c) del modo procesado.

Además, esta función también es la encargada de actualizar todos los modos de la operación sucesora del modo_minimo. Para ello se busca la operación a la que pertenece el modo_minimo y se identifica la operación sucesora. El tiempo en el que pueden comenzar a ser procesados los modos de la operación sucesora (r), se actualiza con el tiempo de finalización (c) del modo_minimo. Esto asegura una transición adecuada entre operaciones consecutivas.

Para finalizar, deben actualizarse todos los modos pertenecientes a la operación que contiene modo_minimo para que estos ya no puedan ser elegidos, ya que dentro de una operación únicamente puede procesarse uno de los modos.

Para ello, se identifica la operación a la que pertenece el modo_minimo; luego, para todos los modos en esta operación, excepto el modo_minimo, se actualizan sus tiempos de inicio (b) y finalización (c) a infinito.

Esto indica que estos modos no están programados para ser procesados, ya que el modo_minimo se ha seleccionado y planificado.

```
def procesar_modo(modos_en_MAQ, modo_minimo, trabajos, S, P, MAQ):
    #modo_minimo debe procesarse, es decir; debemos cambiar la b y la c de ese modo.
    max_c = float('-inf')
    for operacion in P:
        for modo in operacion.modos:
             if modo.maquina == MAQ and modo.c != float('inf') and modo.c > max_c:
                max_c = modo.c
    modo_minimo.b = max(modo_minimo.r, max_c)
    modo_minimo.c = modo_minimo.b + modo_minimo.duracion

    #actualizar modo r de la operacion sucesora
    for trabajo in trabajos:
        for i, operacion in enumerate(trabajo.operaciones):
            if modo_minimo in operacion.modos:  # Encuentra la operación que contiene modo_minimo
                if i + 1 < len(trabajo.operaciones):  # Verifica si hay una operación sucesora
                    operacion_sucesora = trabajo.operaciones[i + 1]
                    for modo_sucesor in operacion_sucesora.modos:
                        modo_sucesor.r = modo_minimo.c

    #actuallizar el resto de modos de la operacion a la que pertenece modo_minimo
    operacion_pertinente = None
    for trabajo in trabajos:
        for operacion in trabajo.operaciones:
            if modo_minimo in operacion.modos:
                operacion_pertinente = operacion
                break
        if operacion_pertinente is not None:
            break
    for modo in operacion_pertinente.modos:
        if modo != modo_minimo:
            modo.b = float('inf')
            modo.c = float('inf')
```

Ilustración 29. Función procesar modos

5.2.8 Función mover operación

La función mover_operacion (Ilustración 30. Función mover operaciones) tiene como objetivo gestionar el movimiento de operaciones entre los vectores S (operaciones programables) y P (operaciones ya realizadas).

Primero se busca la operación que contiene el modo_minimo (operacion_contenedora) y, si existe, se identifica la operación sucesora (operacion_sucesora). Si la operación que ha sido procesada está presente en el vector S, se elimina de S y se agrega a P.

Esto refleja que la operación ha sido programada y realizada. Si hay una operación sucesora, se agrega a S. Esto indica que la operación sucesora ahora está disponible para su programación.

```python
def mover_operacion(S, P, modo_minimo, trabajos):
    operacion_contenedora = None
    operacion_sucesora = None

    # Encuentra la operación que contiene modo_minimo y la operación sucesora
    for trabajo in trabajos:
        for i, operacion in enumerate(trabajo.operaciones):
            if modo_minimo in operacion.modos:
                operacion_contenedora = operacion
                if i + 1 < len(trabajo.operaciones):
                    operacion_sucesora = trabajo.operaciones[i + 1]
                break

    # Mueve la operación que contiene a modo_minimo de S a P
    if operacion_contenedora in S:
        S = np.delete(S, np.where(S == operacion_contenedora))
        P = np.append(P, operacion_contenedora)

    # Agrega la operación sucesora a S
    if operacion_sucesora is not None:
        S = np.append(S, operacion_sucesora)

    return S, P
```

Ilustración 30. Función mover operaciones

5.2.9 Función generar *Gantt*

La función de la Ilustración 31 se encarga de generar un diagrama de *Gantt* utilizando la biblioteca Bokeh en Python. El diagrama de *Gantt* representa cada una de las operaciones de los trabajos, la máquina donde se procesa dicha operación y el instante de inicio y fin de procesamiento.

En la inicialización de listas se crean varias listas vacías (machines, starts, ends, colors, operations, works) que se utilizarán para almacenar información sobre los modos elegidos de las operaciones que se visualizarán en el diagrama de *Gantt*.

La función itera sobre las operaciones en el vector P y los modos dentro de cada operación. Comprueba si el modo es el seleccionado entre el conjunto de modos de cada operación (modo.b y modo.c no son infinito), y si es así, agrega información relevante a las listas creadas anteriormente.

También se crea la fuente de datos para conectar los datos que se desean visualizar con los elementos del gráfico, facilitando así la actualización dinámica de la visualización cuando cambian estos datos.

Se configura la leyenda iterando sobre los trabajos y asigna un color distinto a cada uno.

Para finalmente hacer la representación gráfica de las operaciones en el diagrama de *Gantt* se utiliza p.hbar para agregar barras horizontales al gráfico. Cada barra representa una operación y su posición y tamaño están determinados por la información proporcionada en las listas (machines, starts, ends, colors).

Por último, se utiliza show(p) para mostrar el diagrama de *Gantt*.

```python
from bokeh.plotting import figure, show
from bokeh.models import ColumnDataSource
from bokeh.palettes import Category20

def crear_diagrama_gantt(P, num_maquinas):
    machines = []
    starts = []
    ends = []
    colors = []
    operations = []
    works = []

    for i, operacion in enumerate(P):
        for modo in operacion.modos:
            if modo.b != float('inf') and modo.c != float('inf'):
                machines.append(f"Máquina {modo.maquina + 1}")
                starts.append(modo.b)
                ends.append(modo.c)
                colors.append(Category20[20][modo.trabajo_pertenece % 20])  # Color según el trabajo
                operations.append(f"Operación {i}")
                works.append(f"Trabajo {modo.trabajo_pertenece}")

    # Fuente para el Gantt
    source = ColumnDataSource(data=dict(
        machines=machines,
        starts=starts,
        ends=ends,
        color=colors,
        operations=operations,
        works=works
    ))

    p = figure(y_range=list(set(machines)), plot_width=800, plot_height=400, title="Diagrama de Gantt")

    legend_items = []
    for i, work in enumerate(set(works)):
        legend_items.append((work, [p.rect(x=0, y=0, width=0, height=0, color=Category20[20][i % 20])]))

    p.legend.items = legend_items
    p.hbar(y='machines', left='starts', right='ends', height=0.4, color='color', source=source, legend_field='works')
    show(p)
```

Ilustración 31. Función diagrama de Gantt

5.2.10 Función programa por niveles menor duración

La función de la Ilustración 32 se utilizará únicamente cuando queramos resolver el problema mediante planificación separada. Esta función debe ser llamada al principio del programa, después de haber leído el fichero y haberlo guardado en el vector trabajos. El objetivo de esta función es que cada operación tenga un único modo mediante una asignación previa, la cual únicamente guarda el modo de menor duración de cada una de las operaciones con su correspondiente máquina.

```python
def seleccionar_modo_menor_duracion(trabajos):
    for trabajo in trabajos:
        for operacion in trabajo.operaciones:
            # Encontrar el modo con la duración mínima
            min_duracion = float('inf')
            selected_modo = None
            for modo in operacion.modos:
                if modo.duracion < min_duracion:
                    min_duracion = modo.duracion
                    selected_modo = modo

            # Configurar las propiedades de Modo
            selected_modo.r = 0
            operacion.modos = [selected_modo]
```

Ilustración 32. Función programa planificación separada de menor duración

5.2.11 Función programa planificación separada distribuyendo carga

La función de la Ilustración 33 se encarga de asignar las operaciones de trabajos a las máquinas de manera que se distribuya equitativamente la carga de trabajo. Primero, calcula la duración mínima de cada operación y la carga inicial de cada máquina. Luego, determina la capacidad de cada máquina (que es la misma para todas) y se calcula sumando todos los tiempos de procesamiento de todas las operaciones (para ello se elige el modo de menor duración) y dividiendo el tiempo total entre el número de máquinas existentes. Su puede multiplicar por un factor k para aumentar el margen de las capacidades de las máquinas y que la máquina más efectiva tenga algo más de asignación que el resto.

En un bucle, la función busca reasignar las operaciones que causan sobrecarga en ciertas máquinas a aquellas menos cargadas. Este proceso se repite hasta que se cumpla la capacidad disponible en las máquinas o hasta alcanzar un máximo de 20 iteraciones, lo que evita bucles infinitos o excesivamente largos.

Para cada iteración, la función identifica la máquina más sobrecargada y la menos cargada. Luego, busca la operación que contribuye más a la sobrecarga y la reasigna a la máquina menos cargada compatible. Después de cada reasignación, se actualiza la carga de las máquinas y se continúa el proceso.

Finalmente, la función actualiza los modos asignados en los objetos de trabajo y devuelve los trabajos actualizados con las nuevas asignaciones de operaciones a máquinas. Por lo que cada operación tendrá como resultado final un único modo.

```python
def reasignacion_metodo_distribuido(trabajos, num_maquinas):
    duraciones_minimas = []
    carga_maquinas = [0] * num_maquinas

    # Calcula la duración mínima de cada operación y la carga inicial de cada máquina
    for trabajo in trabajos:
        for operacion in trabajo.operaciones:
            duraciones = [modo.duracion for modo in operacion.modos]
            duracion_minima = min(duraciones)
            duraciones_minimas.append(duracion_minima)
            # Asigna el modo de menor duración a la operación en la primera iteración
            operacion.modo_asignado = min(operacion.modos, key=lambda x: x.duracion)
            carga_maquinas[operacion.modo_asignado.maquina] += operacion.modo_asignado.duracion

    # Calcula la capacidad disponible en cada máquina
    total_duracion_minima = sum(duraciones_minimas)
    capacidad_disponible = (total_duracion_minima * 1.4) / num_maquinas

    #print("La capacidad máxima de cada máquina es:", capacidad_disponible)

    # Realiza la reasignación hasta que se cumpla la capacidad disponible o se alcancen 150 iteraciones
    iteracion = 1
    while max(carga_maquinas) > capacidad_disponible and iteracion <= 20:

        maquina_sobrecargada = carga_maquinas.index(max(carga_maquinas))
        maquina_menos_cargada = carga_maquinas.index(min(carga_maquinas))

        # Busca el modo asignado de mayor duración en la máquina sobrecargada y encuentra un modo compatible en la máquina menos cargada
        for trabajo in trabajos:
            for operacion in trabajo.operaciones:
                if operacion.modo_asignado.maquina == maquina_sobrecargada:
                    modos_compatibles = [modo for modo in operacion.modos if modo.maquina == maquina_menos_cargada]
                    if modos_compatibles:
                        # Encuentra el modo de mayor duración
                        modo_max_duracion = max(operacion.modos, key=lambda x: x.duracion)
                        # Actualiza el modo asignado de la operación
                        operacion.modo_asignado = min(modos_compatibles, key=lambda x: x.duracion)
                        # Actualiza la carga de las máquinas
                        carga_maquinas[maquina_sobrecargada] -= modo_max_duracion.duracion
                        carga_maquinas[operacion.modo_asignado.maquina] += modo_max_duracion.duracion
                        break  # Sale del bucle una vez que se realiza la reasignación
        iteracion += 1
    print(f"Cargas de máquinas final en la iteración {iteracion}: {carga_maquinas}")
    # Actualiza los modos asignados en los objetos de trabajo
    for trabajo in trabajos:
        for operacion in trabajo.operaciones:
            selected_modo = operacion.modo_asignado
            selected_modo.r = 0  # Restablece el valor de r
            operacion.modos = [selected_modo]  # Conserva solo el modo seleccionado
```

Ilustración 33. Función programa separado con distribución de carga

5.2.12 Función porcentajes asignación de las máquinas por número de operaciones realizadas

La función calcular_porcentajes_asignacion (Ilustración 35) calcula los porcentajes de asignación de tareas a cada máquina en función de los modos válidos presentes en el vector P. Comienza inicializando un diccionario donde cada clave representa una máquina y su valor se establece en 0. Luego, recorre el vector P y sus modos, verificando si el modo es válido mediante la comprobación de los valores de b y c. Para cada modo válido, se incrementa el contador de la máquina correspondiente en el diccionario.

Después, se calcula el total de modos asignados sumando los valores del diccionario. Finalmente, se calcula y muestra el porcentaje de asignación para cada máquina en función de su frecuencia de asignación y el total de modos asignados.

Esta función es útil para analizar la distribución de la carga de trabajo en un entorno de producción, identificando qué máquinas están más activas en el proceso de producción.

A continuación, en la Ilustración 34 puede observarse cómo se actualiza la carga a medida que avanzan las iteraciones en la función para reasignar las cargas. En este ejemplo las máquinas 3, 5 y 6 solo podían realizar una o dos operaciones; para llevar el ejemplo al extremo. Se observa que a medida que avanzan las iteraciones, como no se puede cumplir que todas las máquinas queden por abajo de la capacidad, se seguirán haciendo iteraciones hasta llegar al segundo criterio de parada (150 iteraciones como máximo).

```
Iteración 1:
Cargas de máquinas después de la iteración 1: [124, 30, 2, 6, 3, 4]
Iteración 2:
Cargas de máquinas después de la iteración 2: [118, 30, 2, 12, 3, 4]
Iteración 3:
Cargas de máquinas después de la iteración 3: [113, 30, 2, 17, 3, 4]
Iteración 4:
Cargas de máquinas después de la iteración 4: [108, 30, 2, 22, 3, 4]
Iteración 5:
Cargas de máquinas después de la iteración 5: [103, 30, 2, 27, 3, 4]
Iteración 6:
Cargas de máquinas después de la iteración 6: [99, 30, 2, 31, 3, 4]
Iteración 7:
Cargas de máquinas después de la iteración 7: [95, 30, 2, 35, 3, 4]
Iteración 8:
Cargas de máquinas después de la iteración 8: [91, 30, 2, 39, 3, 4]
Iteración 9:
Cargas de máquinas después de la iteración 9: [87, 30, 2, 43, 3, 4]
Iteración 10:
Cargas de máquinas después de la iteración 10: [83, 30, 2, 47, 3, 4]
Iteración 11:
Cargas de máquinas después de la iteración 11: [79, 30, 2, 51, 3, 4]
Iteración 12:
Cargas de máquinas después de la iteración 12: [75, 30, 2, 55, 3, 4]
```

Ilustración 34. Ejemplo con la distribución de carga entre las máquinas

```python
def calcular_porcentajes_asignacion(P, num_maquinas):
    frecuencia_maquinas = {maquina: 0 for maquina in range(num_maquinas)}
    for operacion in P:
        for modo in operacion.modos:
            if modo.b != float('inf') and modo.c != float('inf'):
                maquina = modo.maquina
                if maquina in frecuencia_maquinas:
                    frecuencia_maquinas[maquina] += 1
                else:
                    frecuencia_maquinas[maquina] = 1
    total_modos_asignados = sum(frecuencia_maquinas.values())

    print("Porcentaje de asignación a cada máquina:")
    for maquina, frecuencia in frecuencia_maquinas.items():
        porcentaje = (frecuencia / total_modos_asignados) * 100
        print(f"Máquina {maquina+1}: {porcentaje:.2f}%")
```

Ilustración 35. Porcentajes de asignación máquinas por número de operaciones realizado

5.2.13 Función porcentajes asignación de las máquinas por tiempo ocupado.

La función encargada de calcular los porcentajes de asignación de las máquinas en base a los tiempos de procesamiento de las operaciones que realiza cada una de ellas se observa en la Ilustración 36.

En primer lugar, se crea un diccionario para rastrear la duración asignada a cada máquina. Luego, itera sobre las operaciones y sus modos, acumulando la duración asignada a cada máquina. Después de calcular la duración total asignada, calcula y muestra el porcentaje de asignación para cada máquina.

```
def calcular_porcentajes_asignacion_duración(P, num_maquinas):
    duracion_maquinas = {maquina: 0 for maquina in range(num_maquinas)}
    for operacion in P:
        for modo in operacion.modos:
            if modo.b != float('inf') and modo.c != float('inf'):
                maquina = modo.maquina
                duracion_maquinas[maquina] += modo.duracion
    duracion_total_asignada = sum(duracion_maquinas.values())

    print("Porcentaje de asignación a cada máquina por duración de operaciones:")
    for maquina, duracion in duracion_maquinas.items():
        porcentaje = (duracion / duracion_total_asignada) * 100
        print(f"Máquina {maquina+1}: {porcentaje:.2f}%")
```

Ilustración 36. Función porcentaje asignación máquinas por tiempo ocupado

5.2.14 Llamada de funciones

El código de la Ilustración 37. Llamada de funciones se encarga de la llamada de todas las funciones y su interrelación. El criterio de parada de este algoritmo es cuando el vector de operaciones programables (S) esté vacío.

En función de si se realiza o no una planificación separada con o sin distribución o una planificación conjunta se ejecutarán o no sus respectivas funciones.

```
trabajos, num_trabajos, num_maquinas, num_operaciones = lectura("mk01.txt")
#reasignacion_metodo_distribuido(trabajos, num_maquinas)
seleccionar_modo_menor_duracion(trabajos)
S, P = inicializacion(num_trabajos, num_operaciones)

while len(S) > 0:  # Verifica si la longitud de S es mayor que cero
    MAQ = elegir_maquina_G_y_T(S)#puede ser elegir_maquina_G_y_T o elegir_maquina_Lanz
    modos_en_MAQ = dividir_por_maquina(S, num_maquinas, MAQ)
    modo_minimo = encontrar_modo_coef(modos_en_MAQ,P,trabajos)
    procesar_modo(modos_en_MAQ, modo_minimo, trabajos, S, P, MAQ)
    S, P = mover_operacion(S, P, modo_minimo, trabajos)
crear_diagrama_gantt(P, num_maquinas)
obtener_makespan(P)
```

Ilustración 37. Llamada de funciones

Además, para realizar el análisis de los datos como porcentajes de asignación o *flowtimes*, se hará la llamada de las correspondientes funciones de la siguiente manera (Ilustración 38).

```
calcular_porcentajes_asignacion_duración(P, num_maquinas)
```

```
Porcentaje de asignación a cada máquina por duración de operaciones:
Máquina 1: 9.80%
Máquina 2: 45.75%
Máquina 3: 26.14%
Máquina 4: 7.84%
Máquina 5: 0.65%
Máquina 6: 9.80%
```

```
calcular_porcentajes_asignacion(P, num_maquinas)
```

```
Porcentaje de asignación a cada máquina por numero de operaciones:
Máquina 1: 27.27%
Máquina 2: 10.91%
Máquina 3: 10.91%
Máquina 4: 12.73%
Máquina 5: 9.09%
Máquina 6: 29.09%
```

```
flowtime_trabajos = flowtime_de_cada_trabajo(P, trabajos)
print(flowtime_trabajos)
flowtime_medio = np.mean(flowtime_trabajos)
print(flowtime_medio)
```

```
[79, 77, 53, 46, 67, 56, 47, 24, 64, 73]
58.6
```

Ilustración 38. Más llamadas de funciones

6 PRUEBAS DE MÉTODOS IMPLEMENTADOS

6.1 Introducción

La optimización de la programación de operaciones es un desafío clave en la gestión eficiente de sistemas de producción. En este contexto, se han aplicado diversas heurísticas para minimizar el tiempo total de procesamiento. En el presente estudio, se han evaluado dos enfoques distintos: el esquema de construcción de *Giffler y Thompson* y el método de Lanzamiento. A través de la implementación de seis heurísticas, incluyendo SPT, LPT, FIFO, LRPT, SRPT y COEF, se han obtenido los resultados mostrados más adelante, los cuales ofrecen una visión detallada del rendimiento relativo de cada estrategia en ambos métodos.

Los siguientes resultados se han obtenido para el fichero de datos “mk01”, el cual es un problema propuesto en (Brandimarte, 1999); en el que se dispone de un total de 10 trabajos y un total de 6 máquinas. Para cada trabajo se puede tener entre 5 y 7 operaciones. Cada una de las operaciones puede ser programada en un máximo de tres variantes distintas de máquinas, cada una de ellas con tiempos de procesamiento diferentes, oscilando estos entre 1 y 7 unidades de tiempo.

En la Ilustración 39 se muestra un conjunto de enunciados con diferentes números de trabajos, número de máquinas, números de operaciones o tiempos de procesamiento.

Problem instances for the minimum makespan problem.

	njob	nmac	nop	meq	proc
mk1	10	6	5 – 7	3	1 – 7
mk2	10	6	5 – 7	6	1 – 7
mk3	15	8	10 – 10	5	1 – 20
mk4	15	8	3 – 10	3	1 – 10
mk5	15	4	5 – 10	2	5 – 10
mk6	10	15	15 – 15	5	1 – 10
mk7	20	5	5 – 5	5	1 – 20
mk8	20	10	10 – 5	2	5 – 20
mk9	20	10	10 – 15	5	5 – 20
mk10	20	15	10 – 15	5	5 – 20
mk11	30	5	5 – 8	2	10 – 30
mk12	30	10	5 – 10	2	10 – 30
mk13	30	10	5 – 10	5	10 – 30
mk14	30	15	8 – 12	2	10 – 30
mk15	30	15	8 – 12	5	10 – 30

njob : number of jobs;
nmac : number of machines;
nop : minimum and maximum number of operations per job;
meq : maximum number of equivalent machines per operation;
proc : minimum and maximum processing time per operation.

Ilustración 39. Ficheros de problemas FJSP Bramdimarte

6.2 Planificación conjunta

En este tipo de planificación, cada vez que una operación es procesada, todos los modos de la operación sucesora se graban en el vector de modos programables “S”. Esto hace que en cada iteración se vayan teniendo en cuenta todos los modos, cada uno en su respectiva máquina.

6.2.1 *Giffler y Thompson*

Mediante el método de *Giffler y Thompson*, en cada una de las iteraciones se elige la máquina que contiene las operaciones programables que pueden finalizar en la fecha más temprana. Una vez se ha seleccionado la máquina, para elegir la operación programable de esa máquina que finalmente se va a procesar, se pueden seguir diferentes heurísticas. A continuación, se muestran algunas con las que se han realizado ensayos:

6.2.1.1 SPT

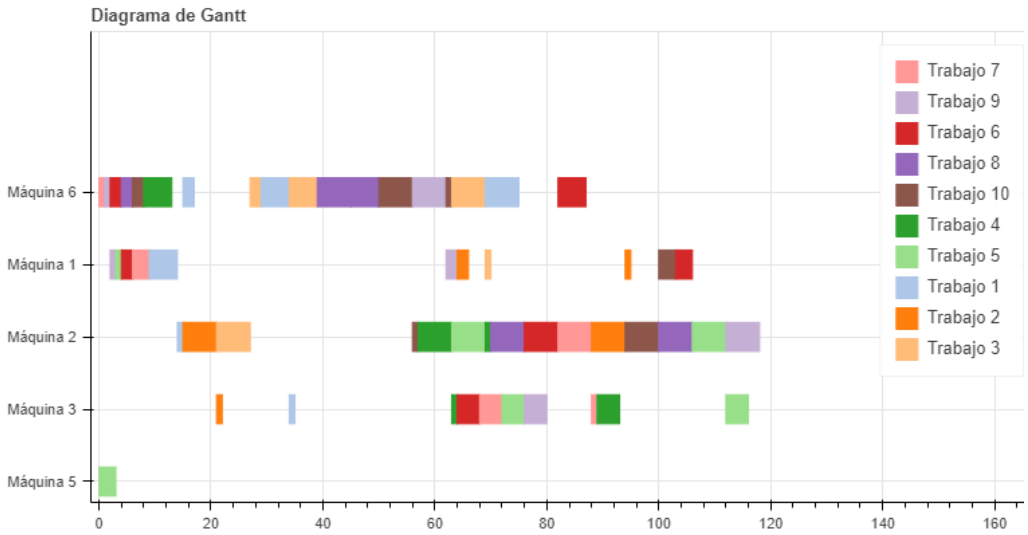

Ilustración 40. Planificación conjunta, Giffler y Thompson, SPT

La Ilustración 40 representa el diagrama de *Gantt* para la combinación del esquema de construcción de *Giffler y Thompson* y la heurística de SPT. La última operación del último trabajo termina de ser procesada en el instante de tiempo $\boldsymbol{Z_{max}} = 118$.

6.2.1.2 LPT

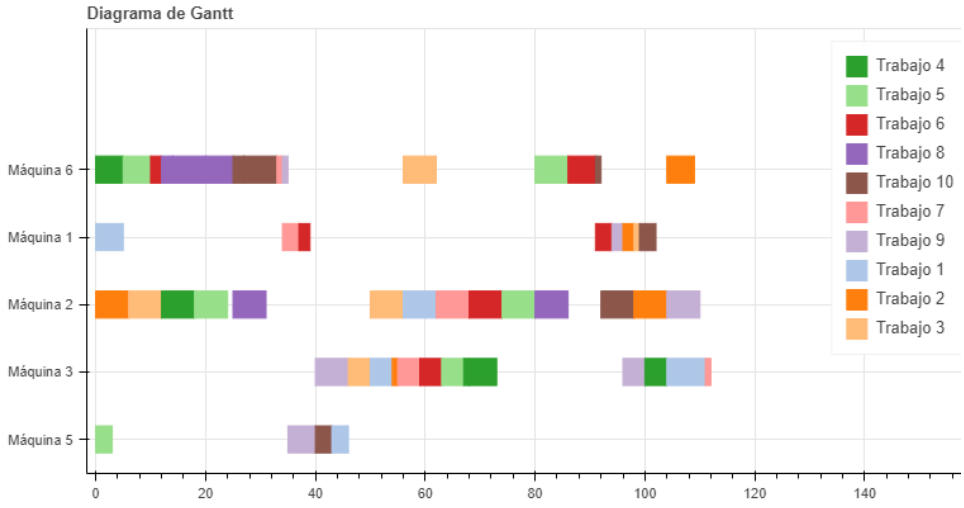

Ilustración 41. Planificación conjunta, Giffler y Thompson, LPT

Mediante la combinación del esquema de construcción de *Giffler y Thompson* y la heurística de LPT, la última operación del último trabajo termina de ser procesada en el instante de tiempo $\boldsymbol{Z_{max}} = 112$. Se ve reflejada la secuenciación de las operaciones en la Ilustración 41.

6.2.1.3 FIFO

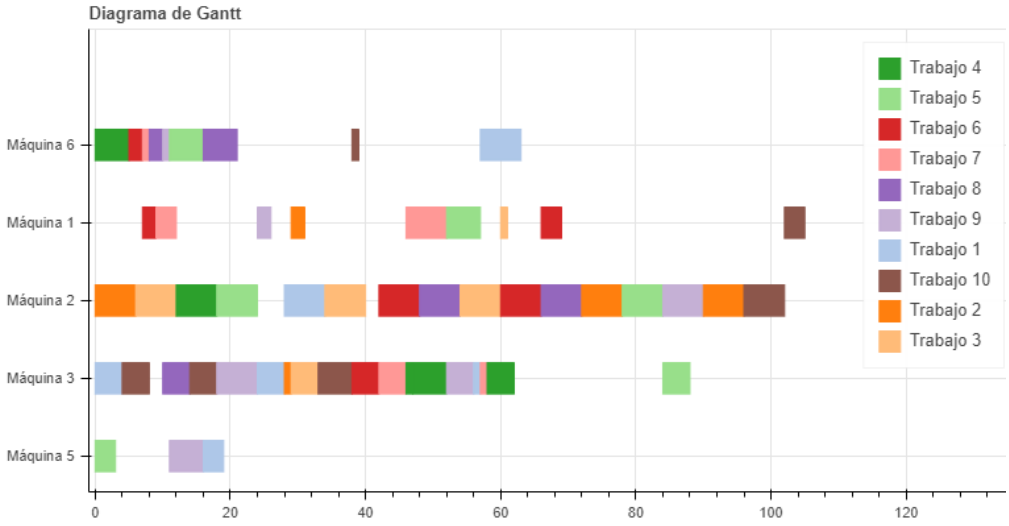

Ilustración 42. Planificación conjunta, Giffler y Thompson, FIFO

Mediante la combinación del esquema de construcción de *Giffler y Thompson* y la heurística de FIFO, la última operación del último trabajo termina de ser procesada en el instante de tiempo $\boldsymbol{Z_{max}} = 105$ (Ilustración 42).

6.2.1.4 LRPT

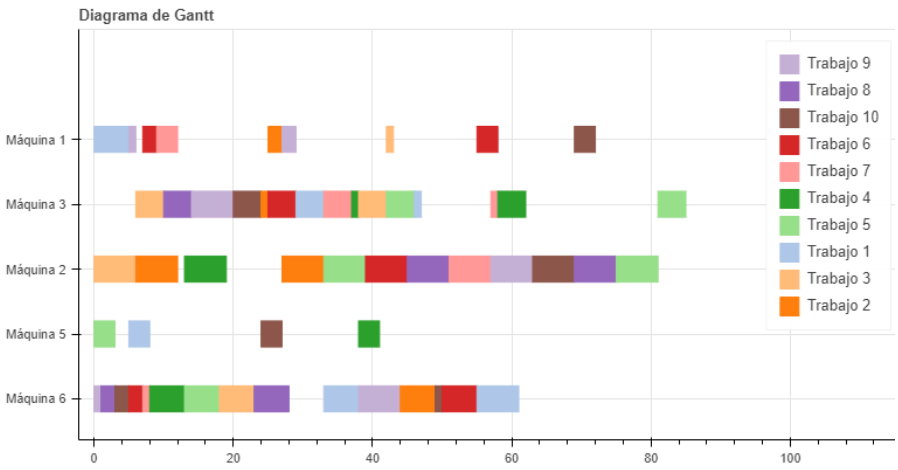

Ilustración 43. Planificación conjunta, Giffler y Thompson, LRPT

El diagrama de *Gantt* de la Ilustración 43 representa la combinación del esquema de construcción de *Giffler y Thompson* y la heurística de LRPT. La última operación del último trabajo termina de ser procesada en el instante de tiempo $\boldsymbol{Z_{max}} = 85$.

6.2.1.5 SRPT

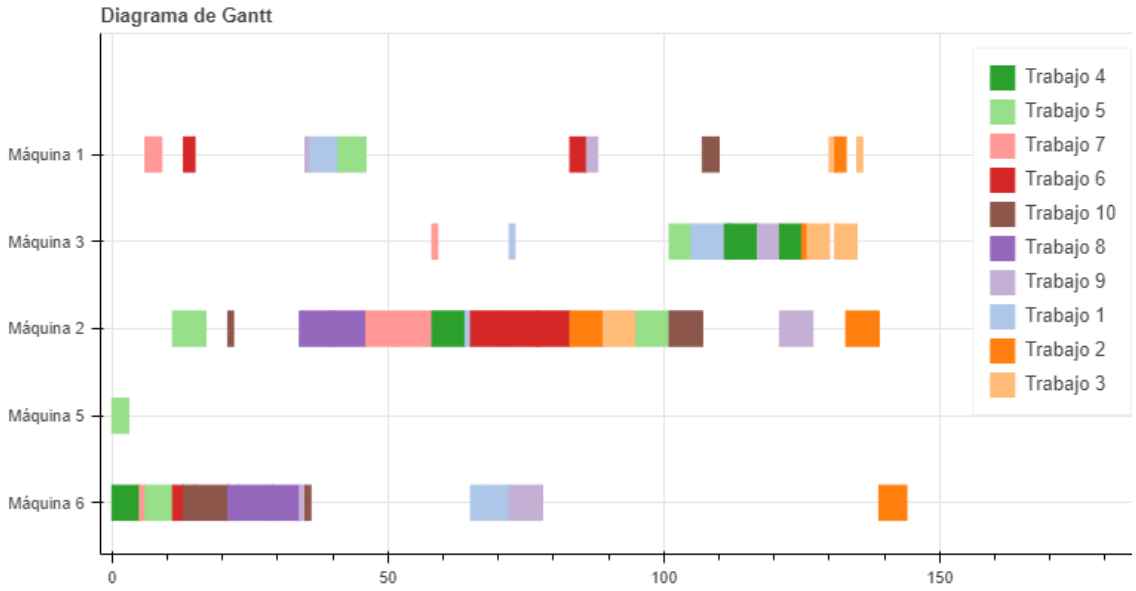

Ilustración 44. Planificación conjunta, Giffler y Thompson, SRPT

La Ilustración 44 representa la combinación del esquema de construcción de *Giffler y Thompson* y la heurística de SRPT. La última operación del último trabajo termina de ser procesada en el instante de tiempo $\boldsymbol{Z_{max}} = 144$.

6.2.1.6 COEF

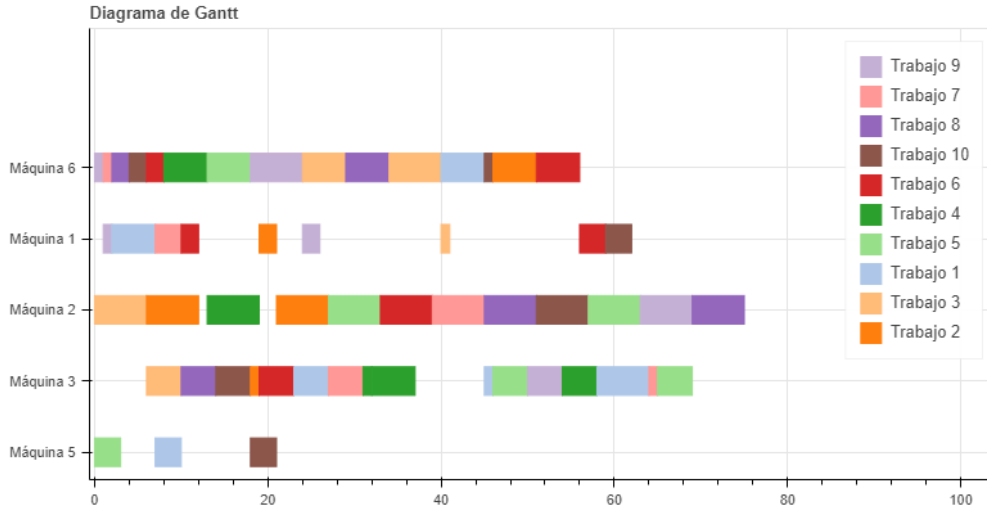

Ilustración 45. Planificación conjunta, Giffler y Thompson, COEF

Mediante la combinación del esquema de construcción de *Giffler y Thompson* y la heurística de COEF, la última operación del último trabajo termina de ser procesada en el instante de tiempo $\boldsymbol{Z_{max}} = 118$ (Ilustración 45).

6.2.2 Lanzamiento

Mediante el método de Lanzamiento, en cada una de las iteraciones se va eligiendo la máquina que contiene las operaciones programables que pueden comenzar en la fecha más temprana. Una vez se tiene la máquina que va a procesar la operación, para elegir la operación programable pueden seguir diferentes heurísticas. A continuación, se muestran algunas con las que se han realizado diferentes pruebas:

6.2.2.1 SPT

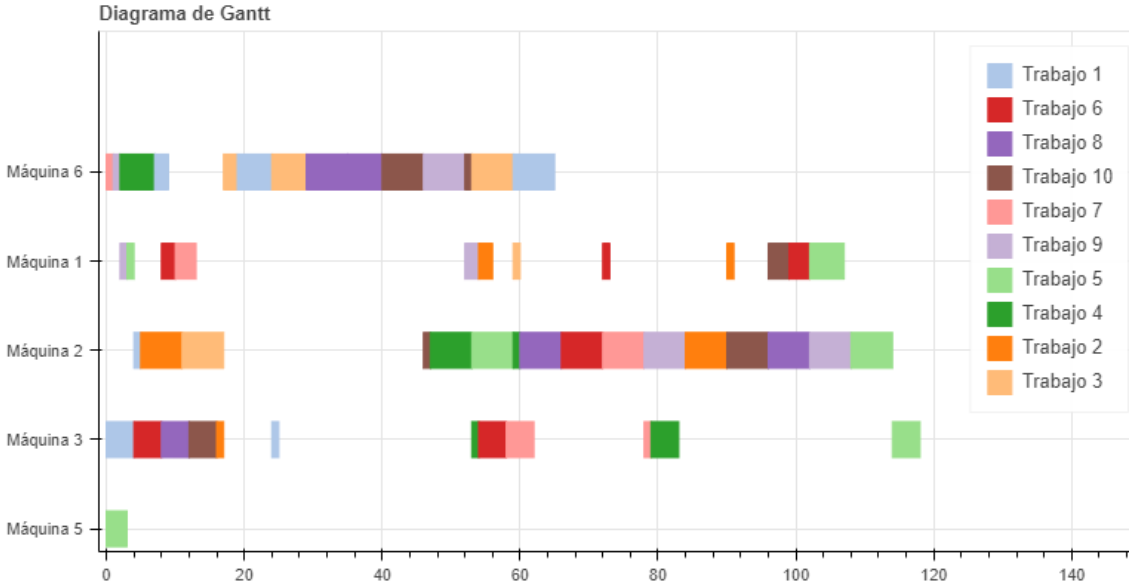

Ilustración 46. Planificación conjunta, Lanzamiento, SPT

En la Ilustración 46 se observa el diagrama de *Gantt* para la combinación del esquema de construcción de Lanzamiento y la heurística de SPT. La última operación del último trabajo termina de ser procesada en el instante de tiempo $\boldsymbol{Z_{max}} = 118$.

6.2.2.2 LPT

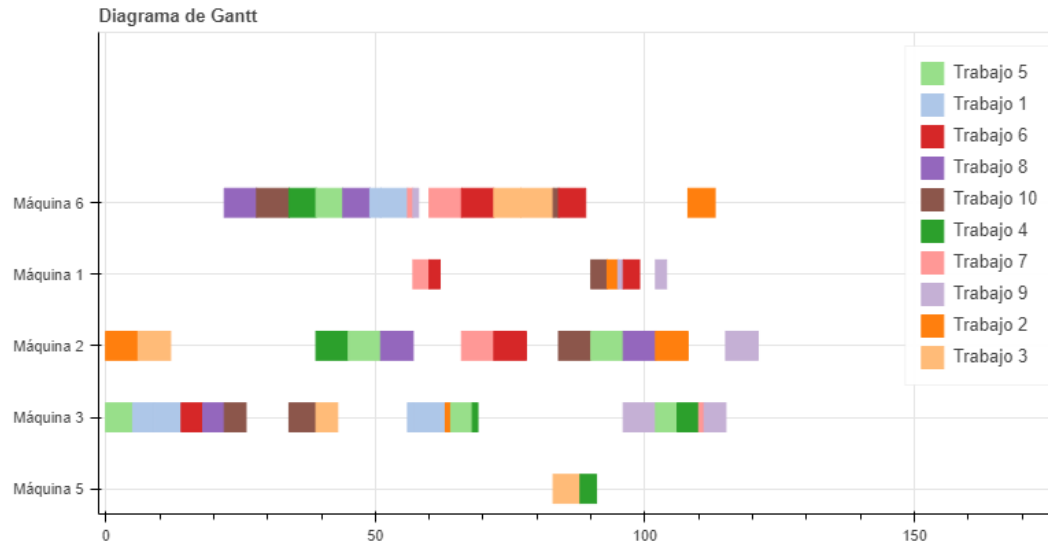

Ilustración 47. Planificación conjunta, Lanzamiento, LPT

Mediante la combinación del esquema de construcción de Lanzamiento y la heurística de LPT, se obtiene un *makespan* $\boldsymbol{Z_{max}} = 121$ (Ilustración 47).

6.2.2.3 FIFO

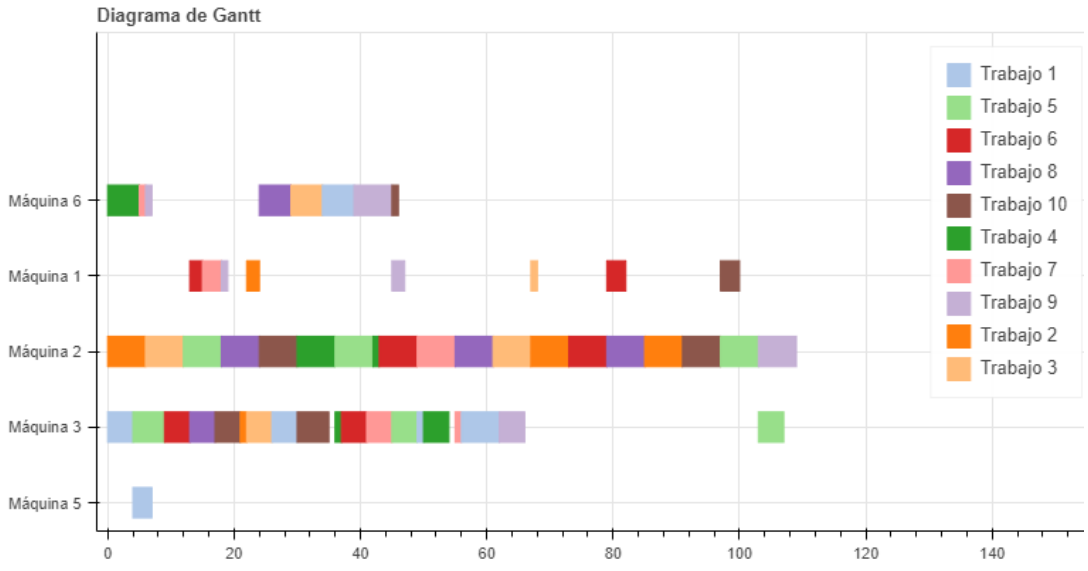

Ilustración 48. Planificación conjunta, Lanzamiento, FIFO

Mediante la combinación del esquema de construcción de Lanzamiento y la heurística de FIFO, se obtiene un *makespan* $\boldsymbol{Z_{max}} = 109$ (Ilustración 48).

6.2.2.4 LRPT

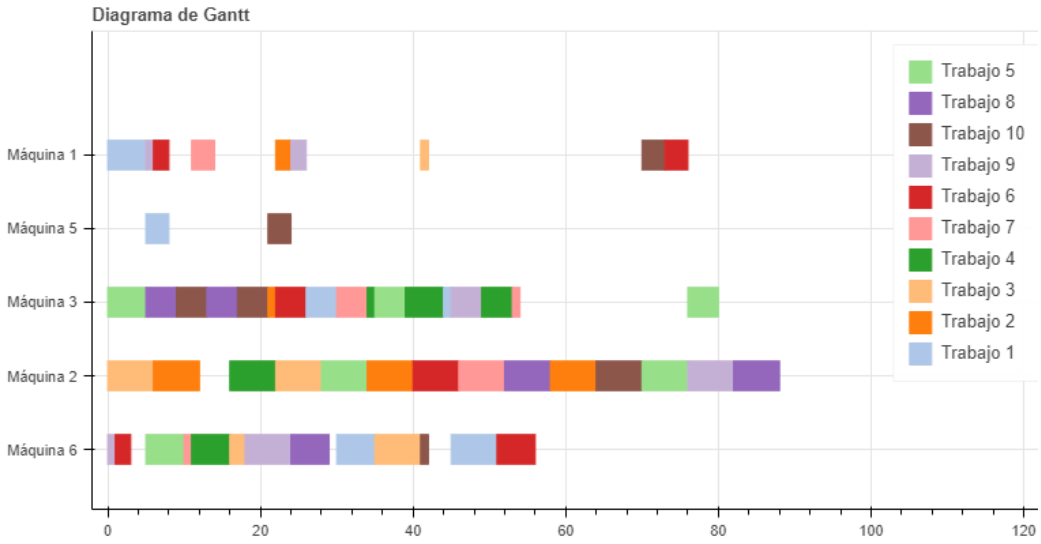

Ilustración 49. Planificación conjunta, Lanzamiento, LRPT

La Ilustración 49 representa el diagrama de *Gantt* para la combinación del esquema de construcción de Lanzamiento y la heurística de LRPT. Se obtiene un *makespan* $\boldsymbol{Z_{max}} = 88$.

6.2.2.5 SRPT

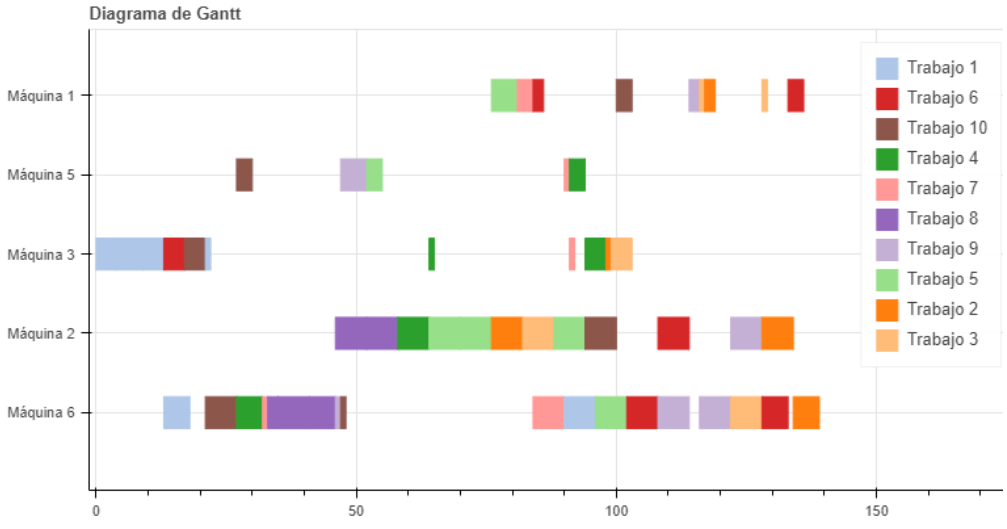

Ilustración 50. Planificación conjunta, Lanzamiento, SRPT

Mediante la combinación del esquema de construcción de Lanzamiento y la heurística de SRPT, la última operación del último trabajo termina de ser procesada en el instante de tiempo $\boldsymbol{Z_{max}} = 139$ (Ilustración 50).

6.2.2.6 COEF

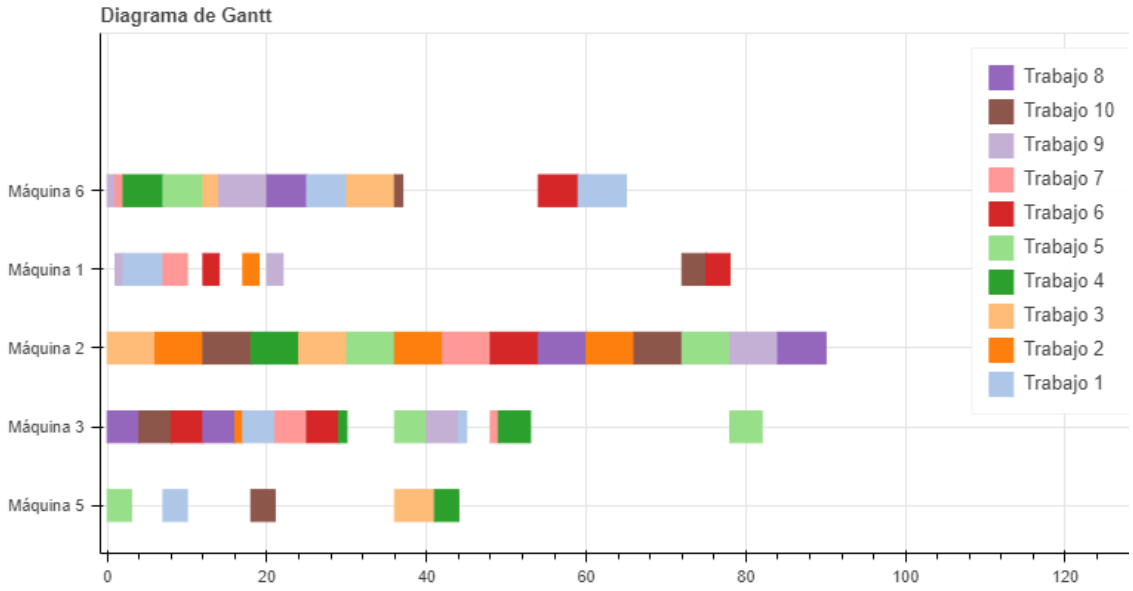

Ilustración 51. Planificación conjunta, Lanzamiento, COEF

Mediante la combinación del esquema de construcción de Lanzamiento y la heurística de COEF, se obtiene un *makespan* $\boldsymbol{Z_{max}} = 90$ (Ilustración 51).

6.3 Planificación separada, menor duración

En el caso de la planificación separada de la producción, se ha establecido una regla que se aplica al comienzo del programa y que consiste en quedarse con un único modo para cada una de las operaciones. Es decir, el problema se resuelve en dos niveles; primero se asigna la máquina a la operación donde se va a procesar y luego se realiza la secuenciación de todas las operaciones.

Hay infinitas reglas de gran complejidad para realizar la asignación del primer nivel, pero la elegida para abordar el problema en este caso consiste en quedarse con el modo que tiene menor tiempo de procesamiento para cada una de las operaciones.

6.3.1 *Giffler y Thompson*

Como se explicó anteriormente, mediante el método de *Giffler y Thompson*, en cada una de las iteraciones vamos eligiendo la máquina que contiene las operaciones programables que pueden finalizar en la fecha más temprana.

A continuación, se muestran los resultados obtenidos representados en un diagrama de *Gantt* cuando es aplicado dicho método:

6.3.1.1 SPT

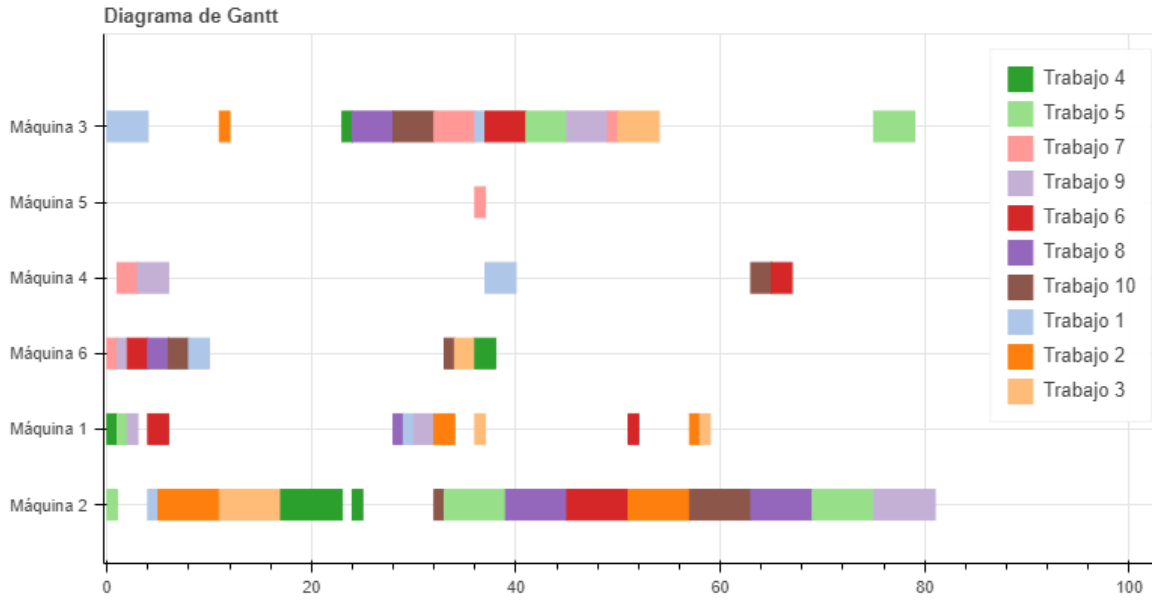

Ilustración 52. Planificación separada, Giffler y Thompson, SPT

La Ilustración 52 representa el diagrama de *Gantt* para la combinación del esquema de construcción de *Giffler y Thompson* y la heurística de SPT. La última operación del último trabajo termina de ser procesada en el instante de tiempo $\boldsymbol{Z_{max}} = 81$.

6.3.1.2 LPT

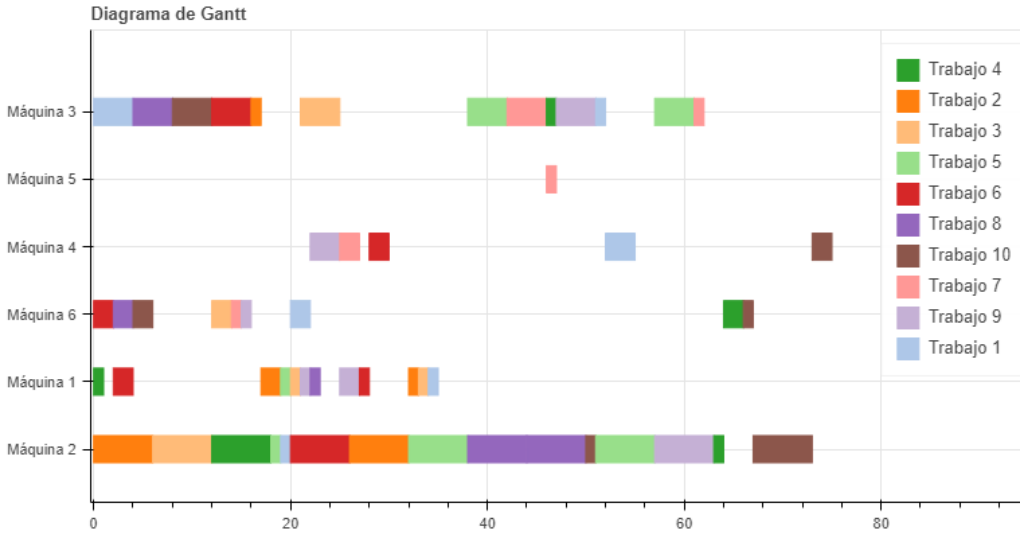

Ilustración 53. Planificación separada, Giffler y Thompson, LPT

La Ilustración 53 representa el diagrama de *Gantt* para la combinación del esquema de construcción de *Giffler y Thompson* y la heurística de LPT. Para esta combinación se obtiene un *makespan* $\boldsymbol{Z_{max}} = 75$.

6.3.1.3 FIFO

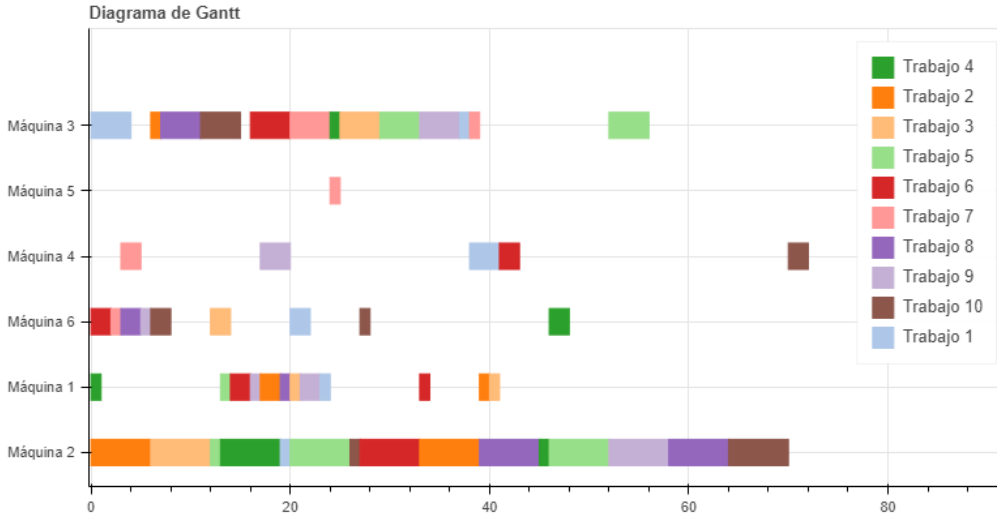

Ilustración 54. Planificación separada, Giffler y Thompson, FIFO

La Ilustración 54 representa el diagrama de *Gantt* para la combinación del esquema de construcción de *Giffler y Thompson* y la heurística de SPT. La última operación del último trabajo termina de ser procesada en el instante de tiempo $\boldsymbol{Z_{max}} = 72$.

6.3.1.4 LRPT

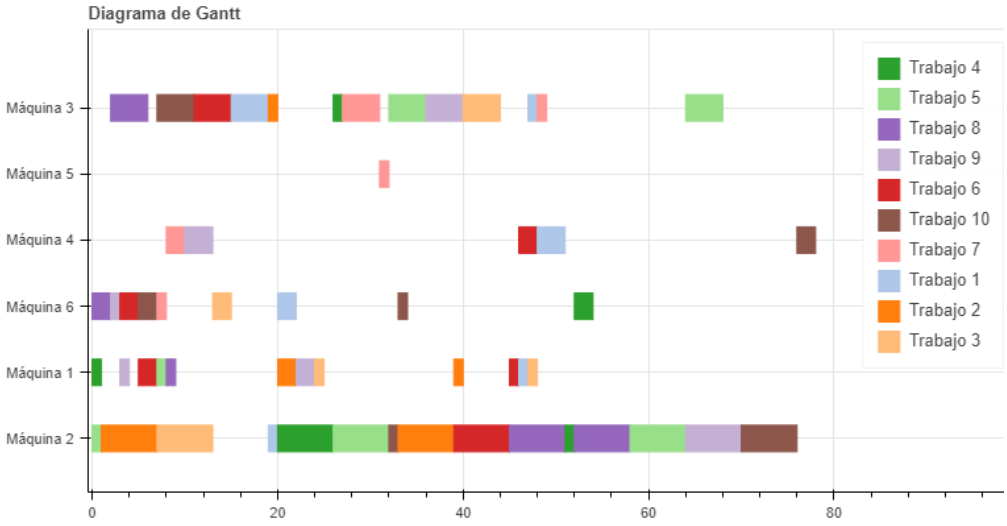

Ilustración 55. Planificación separada, Giffler y Thompson, LRPT

En la Ilustración 55 se muestra el diagrama de *Gantt* para la planificación separada de la producción mediante el esquema de construcción de *Giffler y Thompson* y la heurística de LRPT. Se obtiene un *makespan* $\boldsymbol{Z_{max}} = 78$.

6.3.1.5 SRPT

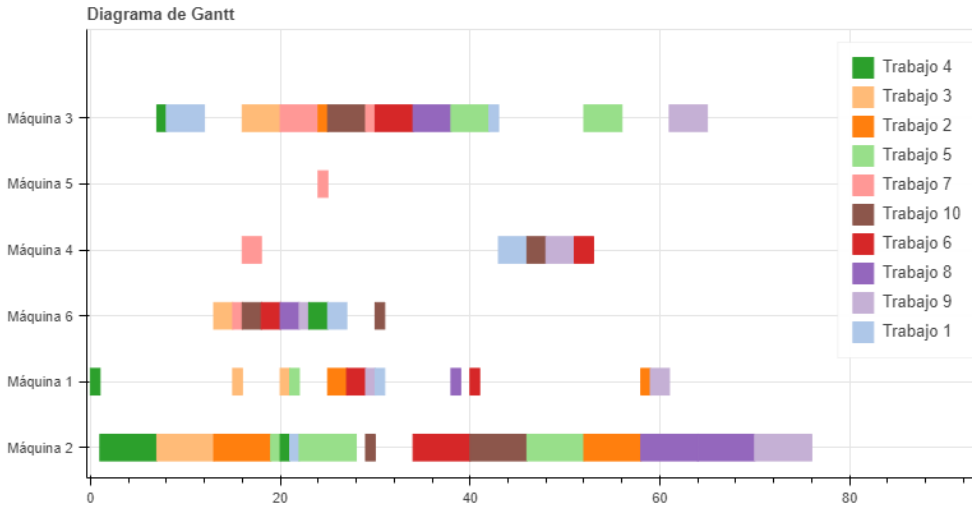

Ilustración 56. Planificación separada, Giffler y Thompson, SRPT

La Ilustración 56 representa el diagrama de *Gantt* para la combinación del esquema de construcción de *Giffler y Thompson* y la heurística de SPT. La última operación del último trabajo termina de ser procesada en el instante de tiempo $\boldsymbol{Z_{max}} = 76$.

6.3.1.6 COEF

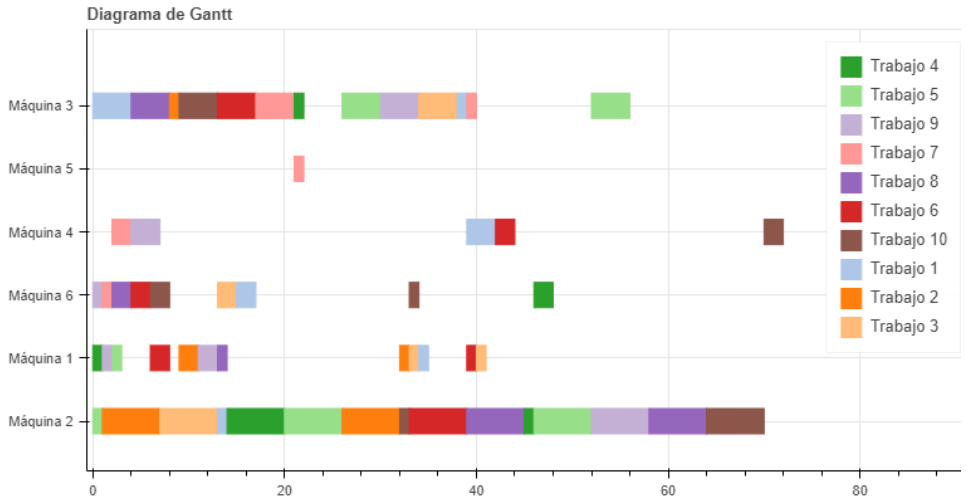

Ilustración 57. Planificación separada, Giffler y Thompson, COEF

En la Ilustración 57 se muestra el diagrama de *Gantt* para la planificación separada de la producción mediante el esquema de construcción de *Giffler y Thompson* y la heurística de LRPT. Se obtiene un *makespan* $\boldsymbol{Z_{max}} = 72$.

6.3.2 Lanzamiento

Tal y como se explica anteriormente, en el método de Lanzamiento, en cada una de las iteraciones se va eligiendo la máquina que contiene las operaciones programables que pueden comenzar en la fecha más temprana.

6.3.2.1 SPT

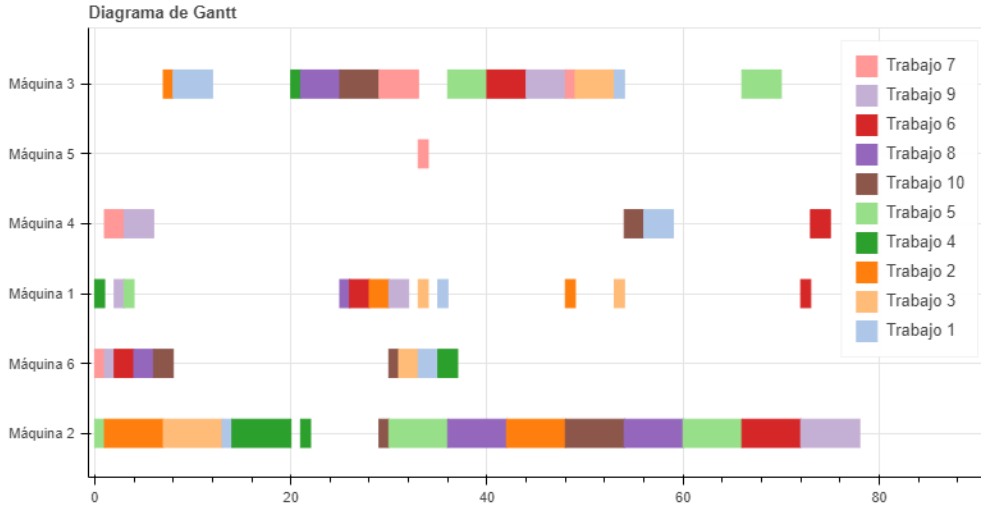

Ilustración 58. Planificación separada, Lanzamiento, SPT

La Ilustración 58 representa el diagrama de *Gantt* para la combinación del esquema de construcción de Lanzamiento y la heurística de SPT. La última operación del último trabajo termina de ser procesada en el instante de tiempo $\boldsymbol{Z_{max}} = 78$.

6.3.2.2 LPT

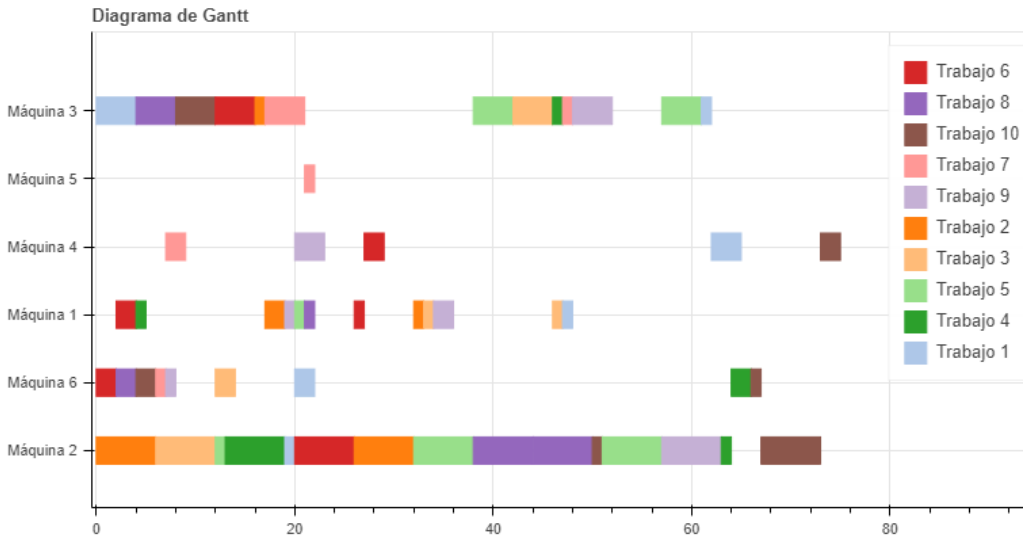

Ilustración 59. Planificación separada, Lanzamiento, LPT

La Ilustración 59 representa el diagrama de *Gantt* para la combinación del esquema de construcción de Lanzamiento y la heurística de SPT. La última operación del último trabajo termina de ser procesada en el instante de tiempo $\boldsymbol{Z_{max}} = 75$.

6.3.2.3 FIFO

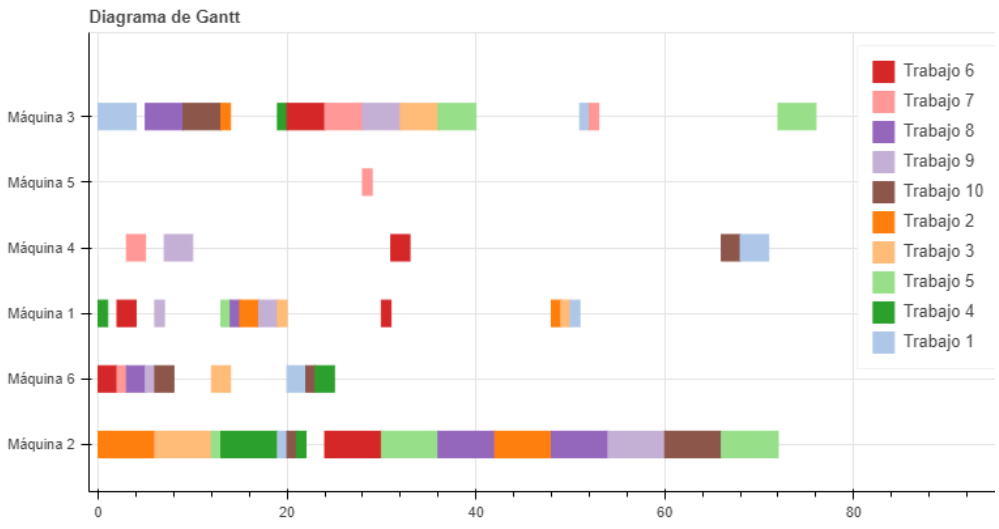

Ilustración 60. Planificación separada, Lanzamiento, FIFO

Mediante la combinación del esquema de construcción de Lanzamiento y la heurística de FIFO, la última operación del último trabajo termina de ser procesada en el instante de tiempo $\boldsymbol{Z_{max}} = 76$. (Ilustración 60)

6.3.2.4 LRPT

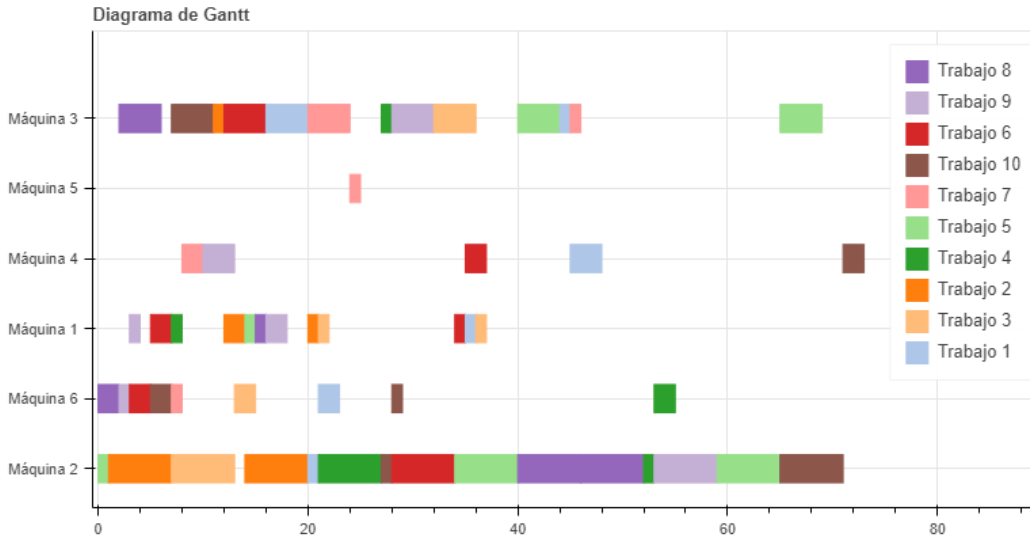

Ilustración 61. Planificación separada, Lanzamiento, LRPT

La Ilustración 61 representa el diagrama de *Gantt* para la combinación del esquema de construcción de Lanzamiento y la heurística de LRPT. Se obtiene un *makespan* $\boldsymbol{Z_{max}} = 73$.

6.3.2.5 SRPT

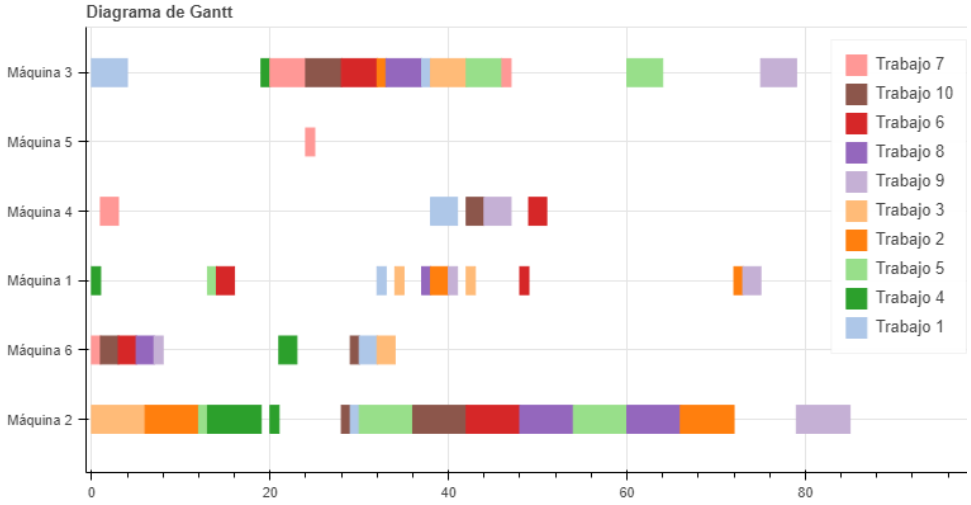

Ilustración 62. Planificación separada, Lanzamiento, SRPT

Mediante la combinación del esquema de construcción de Lanzamiento y la heurística de SRPT, la última operación del último trabajo termina de ser procesada en el instante de tiempo $\boldsymbol{Z_{max}} = 85$. (Ilustración 62).

6.3.2.6 COEF

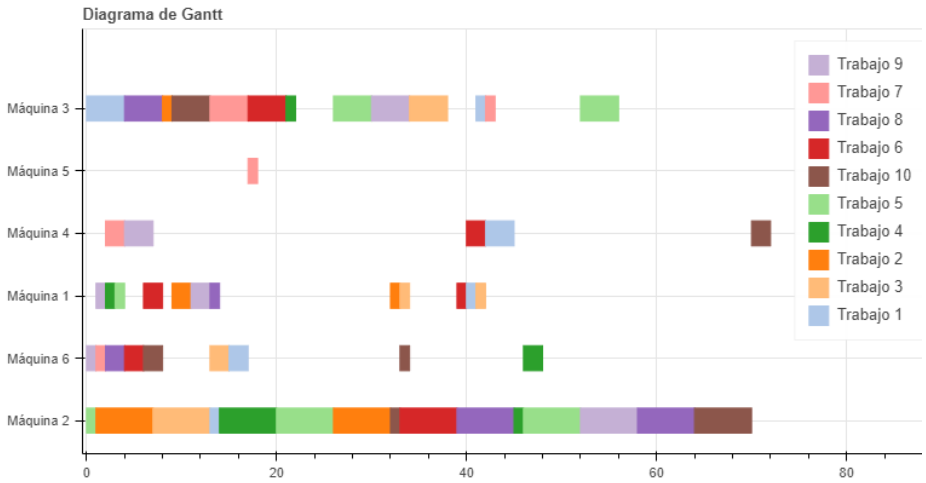

Ilustración 63. Planificación separada, Lanzamiento, COEF

Mediante la combinación del esquema de construcción de Lanzamiento y la heurística de COEF Se obtiene un *makespan* $\boldsymbol{Z_{max}} = 72$ (Ilustración 63).

6.4 Planificación separada con distribución de carga

En ocasiones, cuando una máquina tiene rendimientos superiores respecto a otras, la planificación separada asignando las operaciones a las máquinas únicamente fijándose en el tiempo de procesamiento de cada modo ofrece rendimientos muy bajos. Para solucionar este problema se crea un método que tiene en cuenta las cargas en las máquinas. Si una máquina, en la asignación en la que cada operación procesa el modo de menor duración, esta sobrecargada, se reparte la carga entre las máquinas con una menor. El criterio que se sigue es: en primer lugar, se identifica la operación que mayor carga aporta a la máquina sobrecargada. Si esta operación puede procesarse en la máquina que menor carga tiene asignada, se modifica la asignación. En caso contrario se sigue buscando en la segunda máquina con menor asignación y se realiza la misma comprobación. Este proceso se repite hasta que todas las máquinas estén por debajo del límite considerado sobrecarga. Este umbral es denominado como capacidad; la cual es la misma para todas las máquinas y se calcula como la suma de todos los tiempos de procesamiento de todas las operaciones y trabajos; multiplicado por un factor superior a uno; entre el número de máquinas que se dispone. Para escoger el tiempo de procesamiento de cada operación, se elige el modo de menor duración siempre. El factor indicado anteriormente, cuanto mayor es, más heterogeneidad de carga se permite en la reasignación de las máquinas a las operaciones, se recomienda un factor superior a 1,3 e inferior a 2.

6.4.1 *Giffler y Thompson*

Tal y como se explica anteriormente, en el método de *Giffler y Thompson*, en cada una de las iteraciones se va eligiendo la máquina que contiene las operaciones programables que antes pueden finalizar.

A través de la planificación separada con distribución de carga de la producción y el esquema de construcción de *Giffler y Thompson*, se han obtenido los resultados mostrados a continuación para cada una de las heurísticas implementadas. Es importante destacar que la constante que sirve para determinar las capacidades de las máquinas toma el valor fijo de 1.4; la variación de esta puede modificar radicalmente los resultados obtenidos.

6.4.1.1 SPT

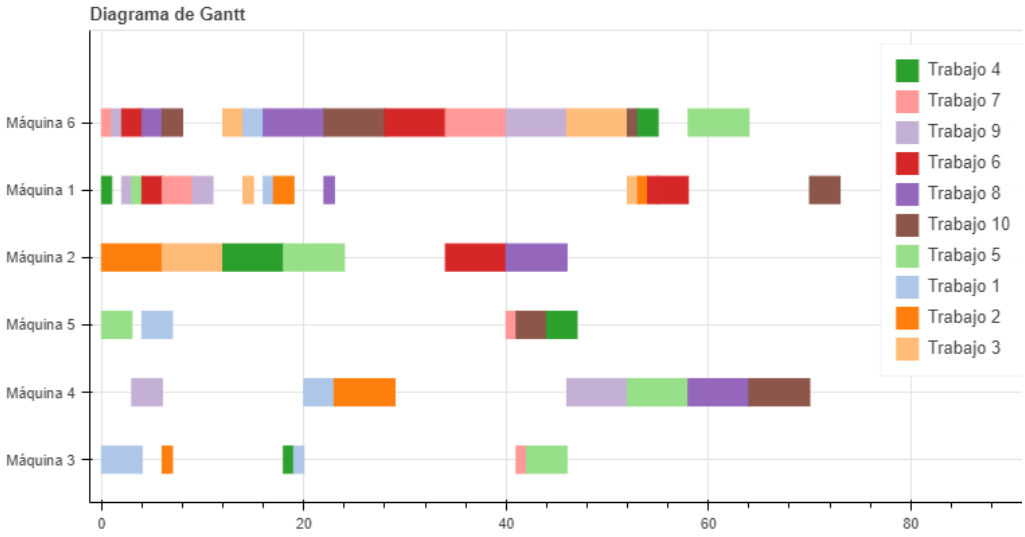

Ilustración 64. Planificación separada distribuida, Giffler y Thompson, SPT

La solución obtenida al resolver el problema con la heurística de SPT puede observarse en la, obteniendo un *makespan* de $\boldsymbol{Z_{max}} = 73$.

6.4.1.2 LPT

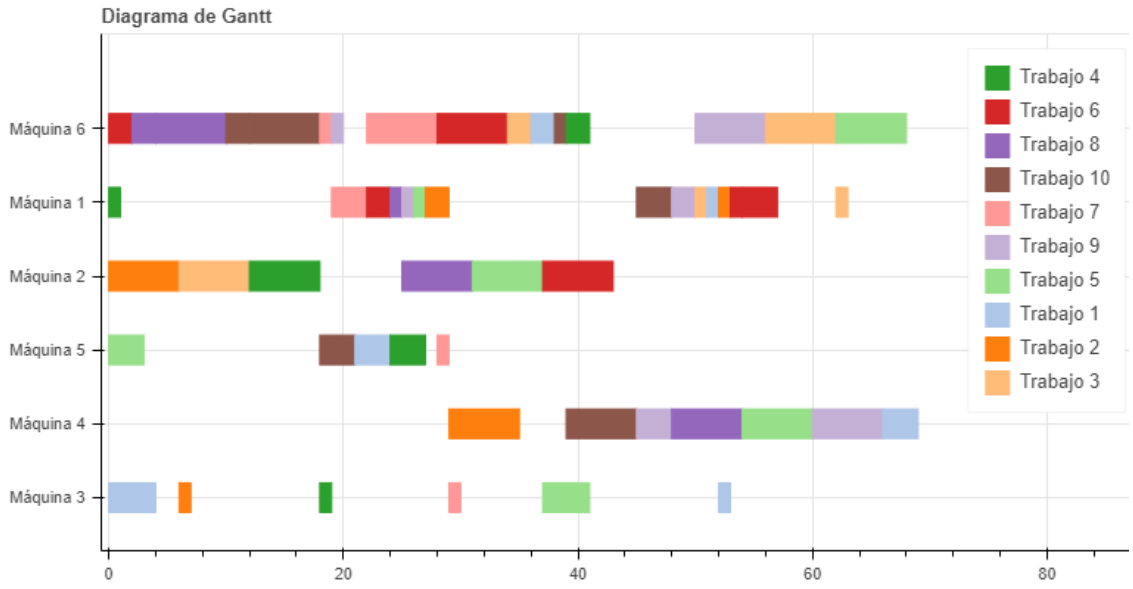

Ilustración 65. Planificación separada distribuida, Giffler y Thompson, LPT

El *makespan* obtenido utilizando la heurística de LPT es $\boldsymbol{Z_{max}} = 69$ (Ilustración 65).

6.4.1.3 FIFO

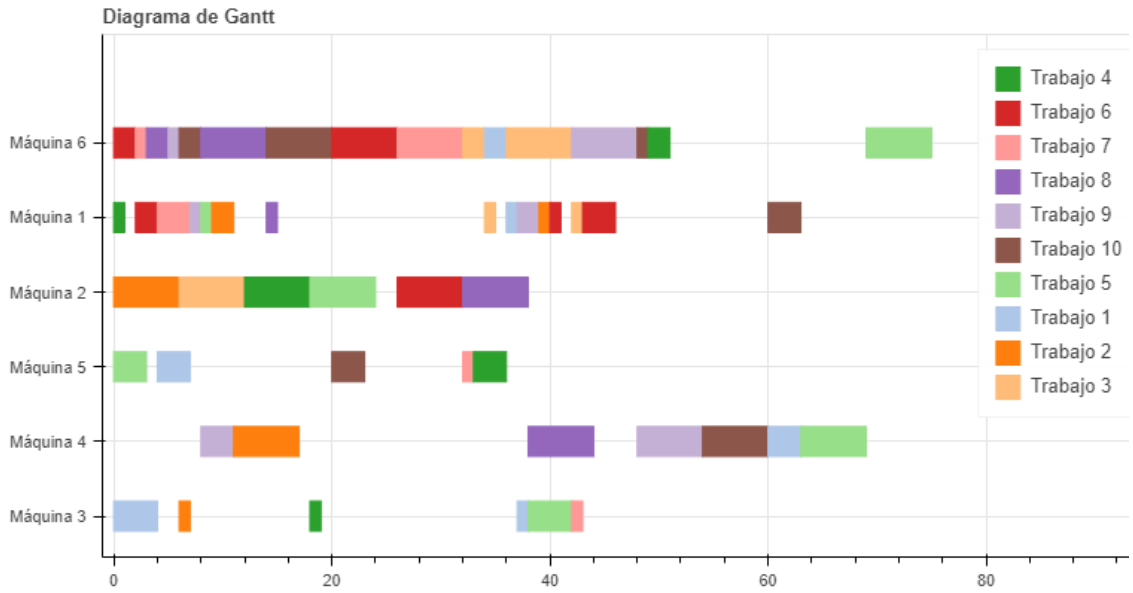

Ilustración 66. Planificación separada distribuida, Giffler y Thompson, FIFO

Si combinamos la planificación separada con distribución de carga, el método de construcción de Giffler y Thompson y la heurística FIFO, se obtiene un *makespan* de $\boldsymbol{Z_{max}} = 75$ (Ilustración 66).

6.4.1.4 LRPT

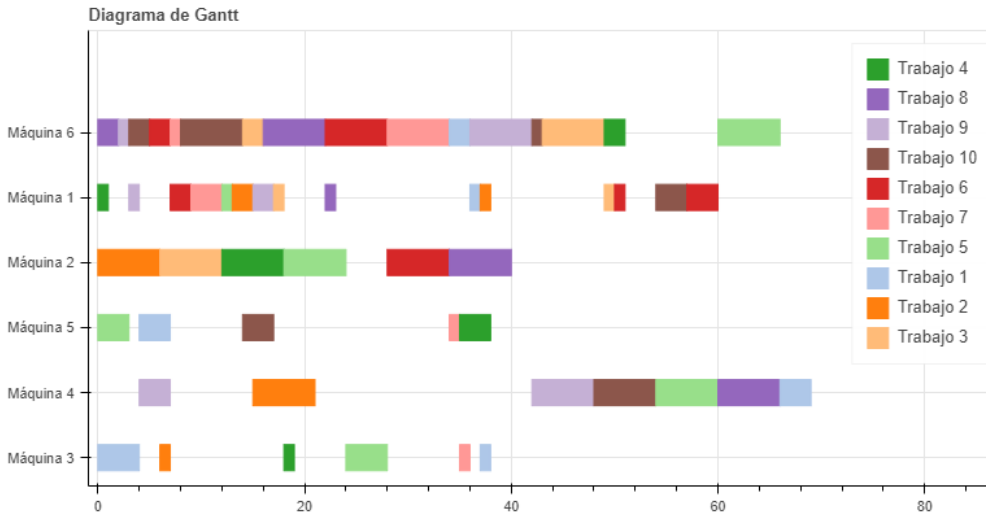

Ilustración 67. Planificación separada distribuida, Giffler y Thompson, LRPT

Para la heurística de LRPT, se obtiene un makespan $\boldsymbol{Z_{max}} = 69$ (Ilustración 67).

6.4.1.5 SRPT

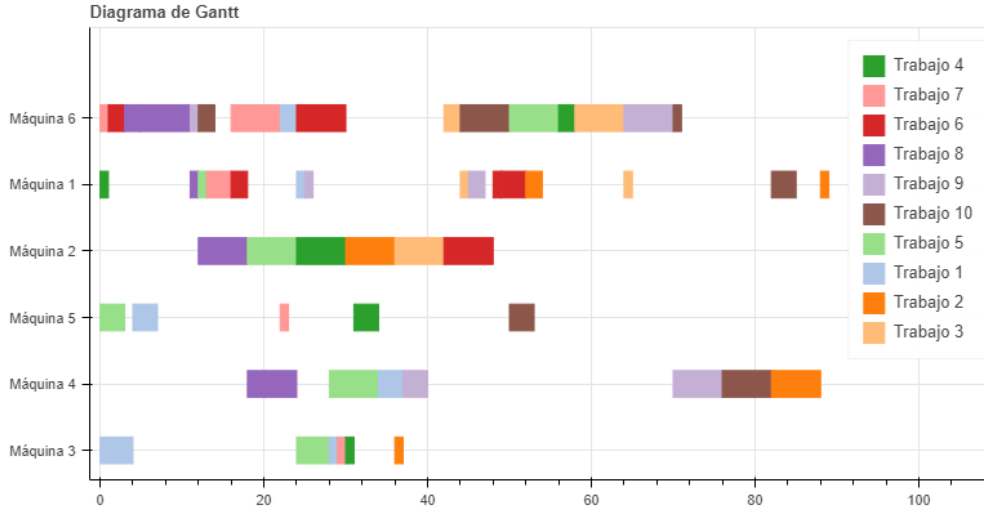

Ilustración 68. Planificación separada distribuida, Giffler y Thompson, SRPT

Si combinamos la planificación separada con distribución de carga, el método de construcción de Giffler y Thompson y la heurística SRPT, se obtiene un *makespan* de $\boldsymbol{Z_{max}} = 89$ (Ilustración 68).

6.4.1.6 COEF

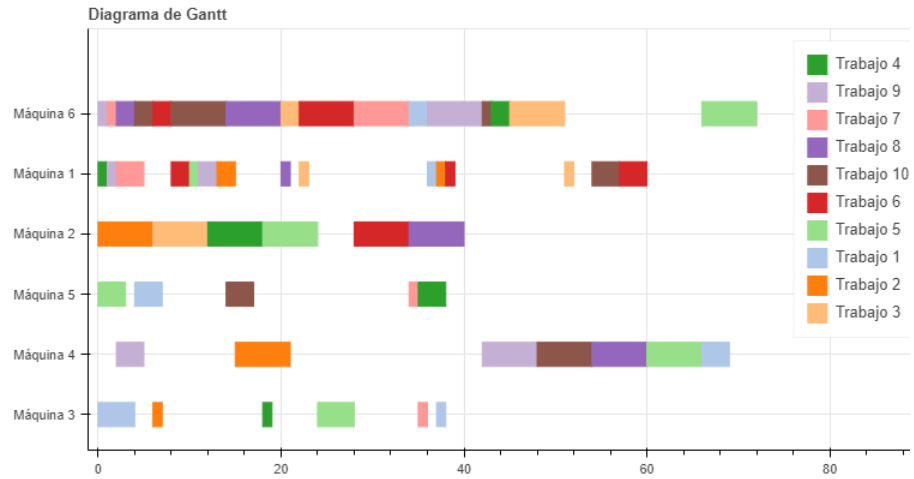

Ilustración 69. Planificación separada distribuida, Giffler y Thompson, COEF

La solución obtenida al resolver el problema con la heurística de COEF puede observarse en la Ilustración 69, obteniendo un *makespan* de $\boldsymbol{Z_{max}} = 72$.

6.4.2 Lanzamiento

En el método de Lanzamiento, en cada una de las iteraciones se va eligiendo la máquina que contiene las operaciones programables que pueden comenzar en la fecha más temprana. Mediante la planificación separada con distribución de carga y el esquema de construcción de Lanzamiento, se obtienen los siguientes resultados para cada una de las heurísticas:

6.4.2.1 SPT

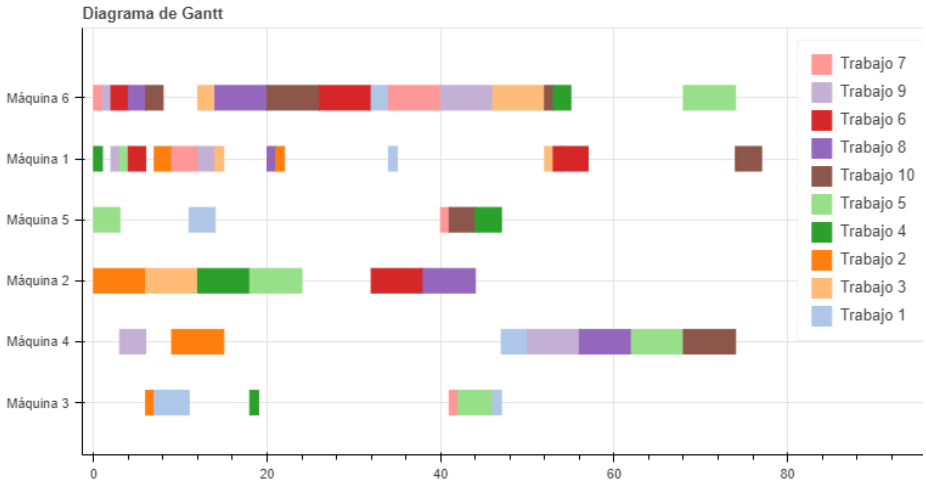

Ilustración 70. Planificación separada distribuida, Lanzamiento, SPT

Al utilizar la heurística de SPT, combinada con la planificación separada de la producción con carga distribuida y el método constructivo de Lanzamiento se obtiene un *makespan* $\boldsymbol{Z_{max}} = 77$ (Ilustración 70).

6.4.2.2 LPT

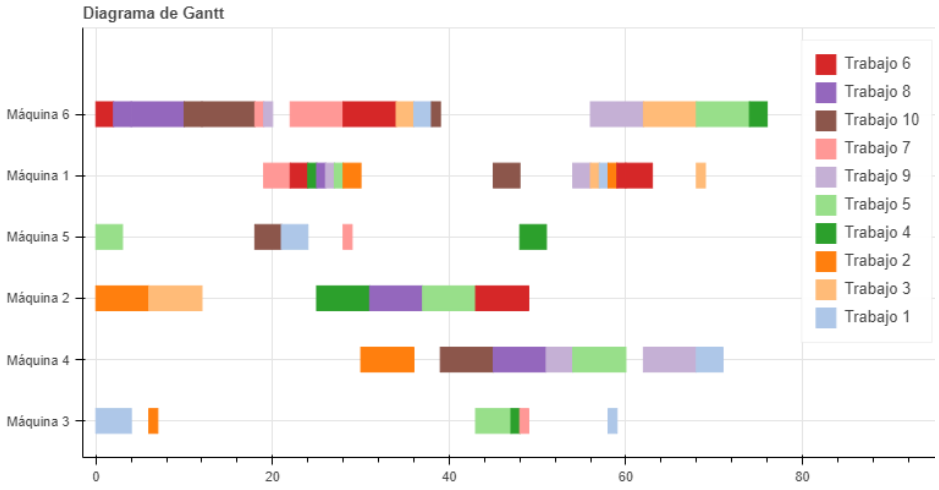

Ilustración 71. Planificación separada distribuida, Lanzamiento, LPT

En la Ilustración 71 puede observarse la solución obtenida usando la planificación separada de la producción con carga distribuida, el método constructivo de Lanzamiento y la heurística de LPT. El *makespan* obtenido es $\boldsymbol{Z_{max}} = 76$.

6.4.2.3 FIFO

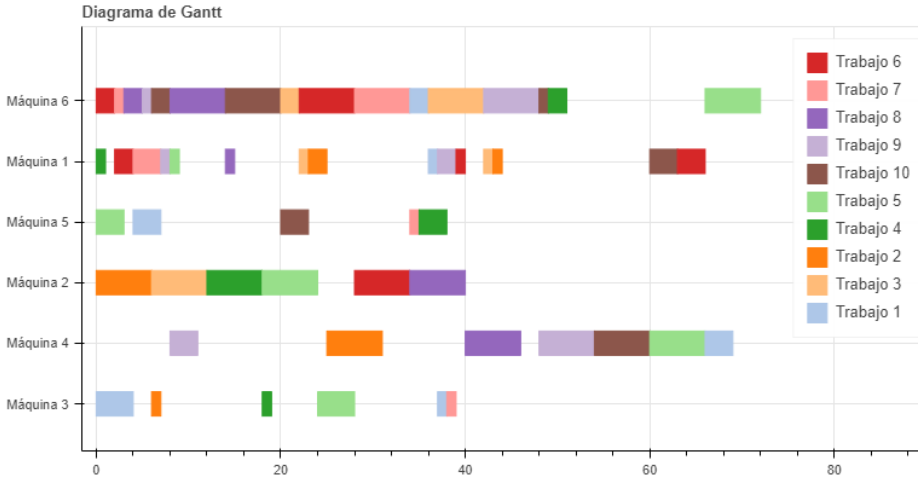

Ilustración 72. Planificación separada distribuida, Lanzamiento, FIFO

En la Ilustración 72 puede observarse la solución obtenida usando la planificación separada de la producción con carga distribuida, el método constructivo de Lanzamiento y la heurística de FIFO. El *makespan* obtenido es $\boldsymbol{Z_{max}} = 72$

6.4.2.4 LRPT

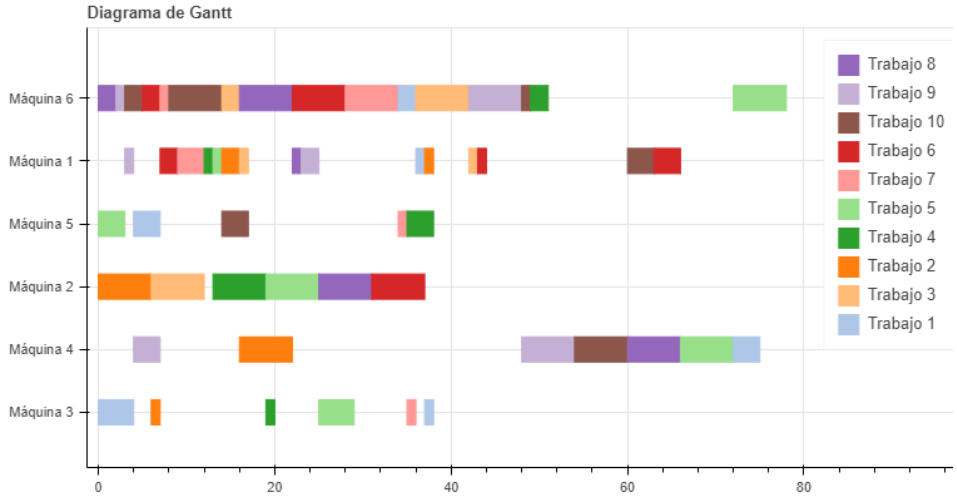

Ilustración 73. Planificación separada distribuida, Lanzamiento, LRPT

Al utilizar la heurística de LRPT, combinada con la planificación separada de la producción con carga distribuida y el método constructivo de Lanzamiento se obtiene un *makespan* $\boldsymbol{Z_{max}} = 78$ (Ilustración 73).

6.4.2.5 SRPT

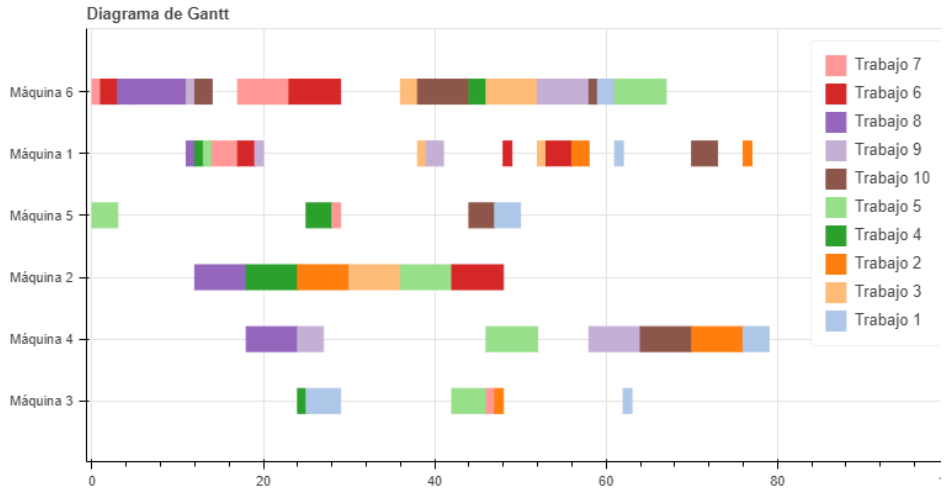

Ilustración 74. Planificación separada distribuida, Lanzamiento, SRPT

Si utilizamos la heurística de SRP en su lugar, el makespan obtenido es $\boldsymbol{Z_{max}} = 79$ (Ilustración 74).

6.4.2.6 COEF

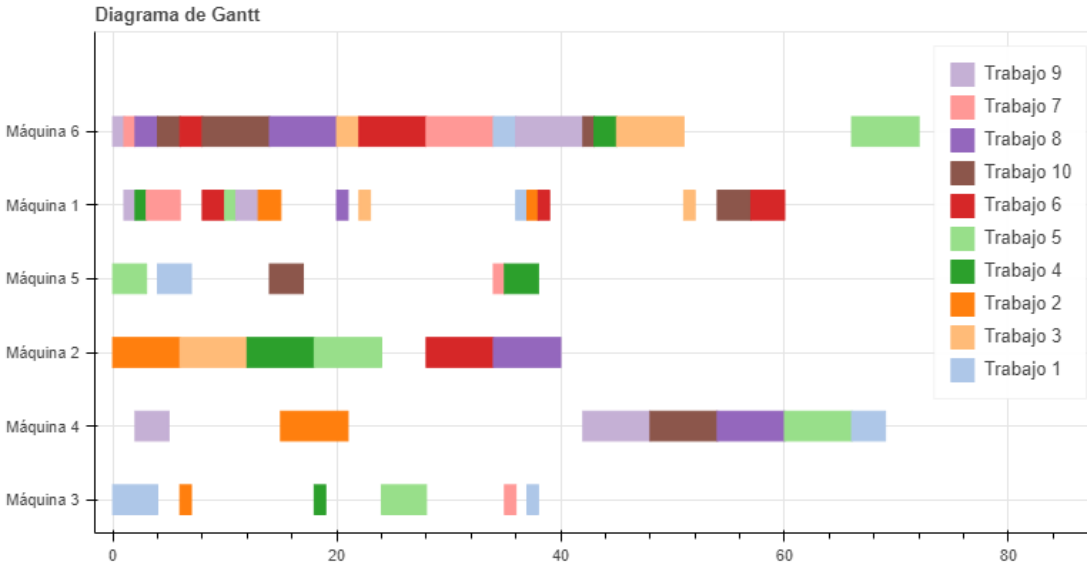

Ilustración 75. Planificación separada distribuida, Lanzamiento, COEF

En la Ilustración 75 puede observarse la solución obtenida usando la planificación separada de la producción con carga distribuida, el método constructivo de Lanzamiento y la heurística de COEF. El *makespan* obtenido es $\boldsymbol{Z_{max}} = 72$

6.5 Comparación resultados

La comparación de resultados entre diferentes heurísticas y métodos de construcción se convierte en un aspecto crucial para determinar la eficacia y eficiencia de dichas estrategias. Este análisis permite evaluar cómo cada enfoque aborda y resuelve los desafíos planteados, proporcionando una visión general de su rendimiento.

Al explorar las diversas heurísticas y métodos, se pueden identificar patrones y tendencias que ayudan a entender mejor las fortalezas y debilidades de cada enfoque. Además, este proceso facilita la selección de la estrategia más adecuada para un problema específico, optimizando así los recursos y mejorando la toma de decisiones.

6.5.1 Makespan

Si se analizan los *makespan*s obtenidos que pueden observarse en la Tabla 3, se obtienen las siguientes conclusiones:

- El menor *makespan* obtenido se consigue con la combinación de planificación separada con carga distribuida de la producción, la heurística de LRPT y el esquema de construcción de lanzamiento o *Giffler y Thompson*.
- Los resultados obtenidos, de forma general, son mejores mediante el esquema de planificación separada y planificación separada distribuida que si lo realizamos con el de planificación conjunta.
- La selección de una heurística correcta afecta en mayor medida a la planificación conjunta de la producción. Los *makespan*s obtenidos mediante planificación separada y planificación separada distribuida son más homogéneos, debido a la reducción del número de modos realizada en primera instancia.
- De forma general, las dos heurísticas que mejores resultados obtienen son la de COEF y la de LRPT.

Makespan	Conjunta		Separada		Separada distribuida	
	G y T	Lanz	G y T	Lanz	G y T	Lanz
SPT	118	118	81	78	73	77
LPT	112	121	75	75	69	76
FIFO	105	109	72	76	75	72
LRPT	85	88	78	73	69	78
SRPT	144	139	76	85	89	79
COEF	118	90	72	72	72	72

Tabla 3. Comparación makespan heurísticas Zmax MK01.

A continuación, se va a analizar el método de optimización y el esquema de construcción óptimo para ser combinado junto la heurística COEF para cada tipo de problema. Se tendrán en cuenta factores como el número total de trabajos, de máquinas o de operaciones; así como el número máximo de alternativas (modos) a la hora de procesar una operación y la variabilidad en los tiempos de procesamiento (Tabla 4).

PROB.	Nº TRAB.	Nº MÁQ	Nº OPER.	Nº MÁX ALTER.	TIEMPO PROCES.	Resultado (COEF)					
						CONJUNTA		SEPARADA		SEPAR. DISTRIB.	
						G y T	L	G y T	L	G y T	L
MK01	10	6	5 a 7	3	1 a 7	75	90	72	72	72	72
MK02	10	6	5 a 7	6	1 a 7	66	72	45	45	55	59
MK03	15	8	10	5	1 a 20	410	445	330	330	250	255
MK04	15	8	3 a 10	3	1 a 10	160	166	190	190	190	190
MK05	15	4	5 a 10	2	5 a 10	318	284	239	239	212	212
MK06	10	15	15	5	1 a 10	269	275	110	114	113	115
MK07	20	5	5	5	1 a 20	315	435	217	223	255	261
MK08	20	10	5 a 10	2	5 a 20	662	702	627	723	627	723
MK09	20	10	10 a 15	5	5 a 20	601	631	477	474	466	485
MK10	20	15	10 a 15	5	5 a 20	665	801	364	379	354	379
MK11	30	5	5 a 8	2	10 a 30	992	861	905	905	767	767
MK12	30	10	5 a 10	2	10 a 30	724	744	819	819	819	819
MK13	30	10	5 a 10	5	10 a 30	1102	1072	707	729	707	729
MK14	30	15	8 a 12	2	10 a 30	1451	1520	1111	1111	1111	1111
MK15	30	15	8 a 12	5	10 a 30	872	945	741	716	673	648

Tabla 4. Resultados Bramdimarte COEF

Cuando la heurística aplicada es la de COEF, si se comparan los esquemas de construcción, puede observarse que, en la mayoría de los problemas resueltos, la mejor opción es *Giffler y Thompson* frente a Lanzamiento. Las diferencias son más notorias en planificación conjunta, ya que, al resolver el problema mediante planificación separada, se elimina gran número de alternativas posibles y el problema queda simplificado (Tabla 4).

PROB.	Nº TRAB.	Nº MÁQ	Nº OPER.	Nº MÁX ALTER.	TIEMPO PROCES.	Resultado (LRPT)					
						CONJUNTA		SEPARADA		SEPAR. DISTRIB.	
						G y T	L	G y T	L	G y T	L
MK01	10	6	5 a 7	3	1 a 7	85	88	78	73	69	78
MK02	10	6	5 a 7	6	1 a 7	74	90	45	45	55	59
MK03	15	8	10	5	1 a 20	487	481	335	330	264	264
MK04	15	8	3 a 10	3	1 a 10	151	173	188	190	188	190
MK05	15	4	5 a 10	2	5 a 10	298	308	239	239	216	222
MK06	10	15	15	5	1 a 10	276	282	116	115	113	128
MK07	20	5	5	5	1 a 20	429	477	217	220	255	255
MK08	20	10	5 a 10	2	5 a 20	677	731	668	707	668	707
MK09	20	10	10 a 15	5	5 a 20	682	688	532	521	526	524
MK10	20	15	10 a 15	5	5 a 20	734	721	351	379	351	379
MK11	30	5	5 a 8	2	10 a 30	941	970	940	940	806	788
MK12	30	10	5 a 10	2	10 a 30	798	909	757	756	757	756
MK13	30	10	5 a 10	5	10 a 30	1550	1246	777	810	777	810
MK14	30	15	8 a 12	2	10 a 30	1468	1548	1151	1127	1151	1127
MK15	30	15	8 a 12	5	10 a 30	860	1184	720	746	656	650

Tabla 5. Resultados Brandimarte LRPT

En la Tabla 5, se observan las soluciones para cada uno de los tipos de planificación y esquemas de construcción para la heurística de LRPT ante todos los problemas propuestos en Brandimarte (Brandimarte, 1999). Se observa que en la mayoría de los casos la planificación por niveles distribuida y la planificación por niveles sigue siento la mejor opción. En cuanto al esquema de construcción, la mejor opción es *Giffler y Thompson* en prácticamente todos los problemas. Cuando el número de alternativas para realizar una operación no es muy numeroso, la planificación simultánea ha de tenerse en cuenta, ya que ofrece buenos rendimientos. A medida que aumenta la cantidad de alternativas, la planificación separada (y la distribuida) es notablemente mejor.

Si se compara la Tabla 4 y la Tabla 5, pueden apreciarse ligeras diferencias respecto al valor del *makespan* entre la heurística de LRPT y COEF. De forma general, la heurística COEF obtiene mejores resultados en la mayoría de los problemas propuestos.

Para la planificación conjunta, es importante destacar que los resultados de *makespan* obtenidos mediante la heurística de LRPT y COEF en combinación con el esquema de construcción de *Giffler y Thompson* son los que más destacan frente al resto de heurísticas (Ilustración 76 e Ilustración 77).

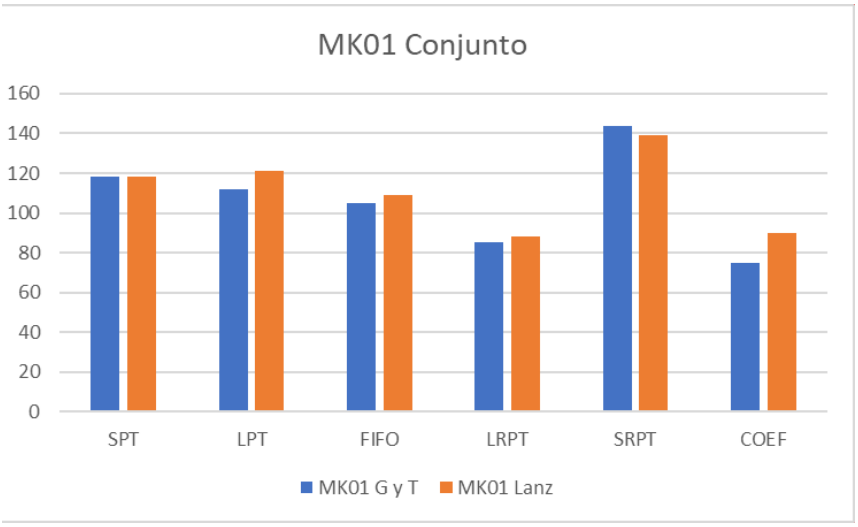

Ilustración 76. Resultados G y T y Lanz para MK01

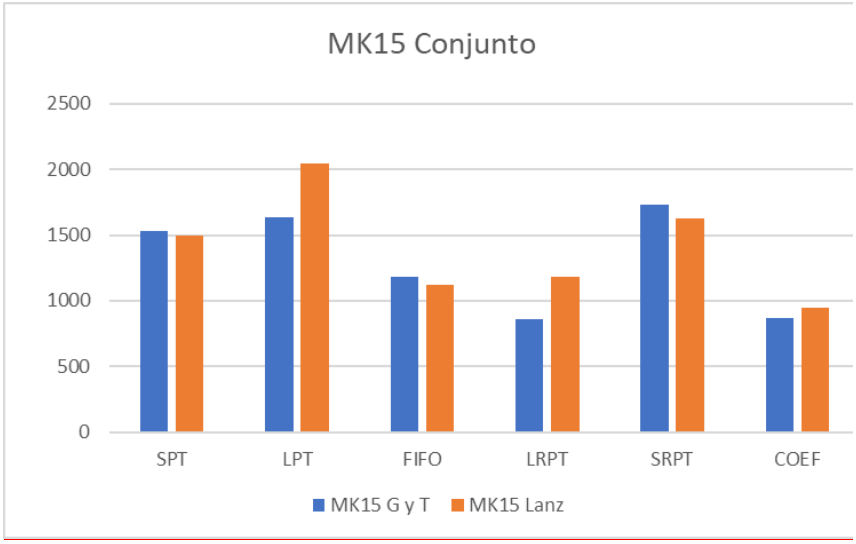

Ilustración 77. Resultados G y T y Lanz para MK15

Es de gran importancia destacar el siguiente caso: cuando la carga entre las máquinas es uniforme y no existe una máquina con mejores rendimientos para la mayoría de las operaciones, el mejor modelo es el de la planificación separada de la producción.

Cuando existe una máquina mucho más efectiva en comparación con el resto, este modelo no serviría de nada, ya que en la asignación del primer nivel esta máquina estaría superpoblada. En la Ilustración 78 puede verse cómo la máquina 1, que tiene el menor tiempo de procesamiento en la mayoría de los casos para las distintas variantes de una operación, queda sobreasignada, obteniéndose un *makespan* $Z_{max} = 133$.

Por el contrario, la Ilustración 79, que utiliza la planificación conjunta, evita este problema y se obtiene un notable mejor *makespan* $Z_{max} = 89$.

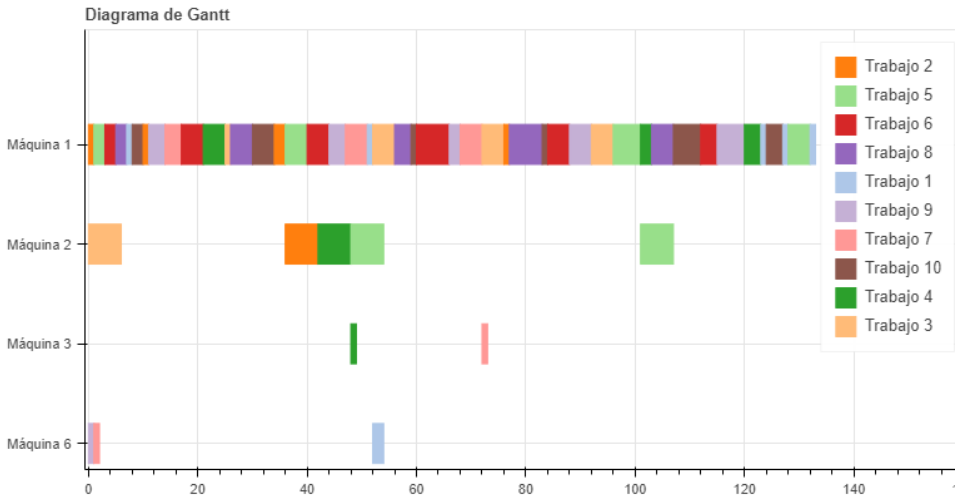

Ilustración 78. Planificación separada, Giffler y Thompson, COEF, mk01_maq_superpoblada

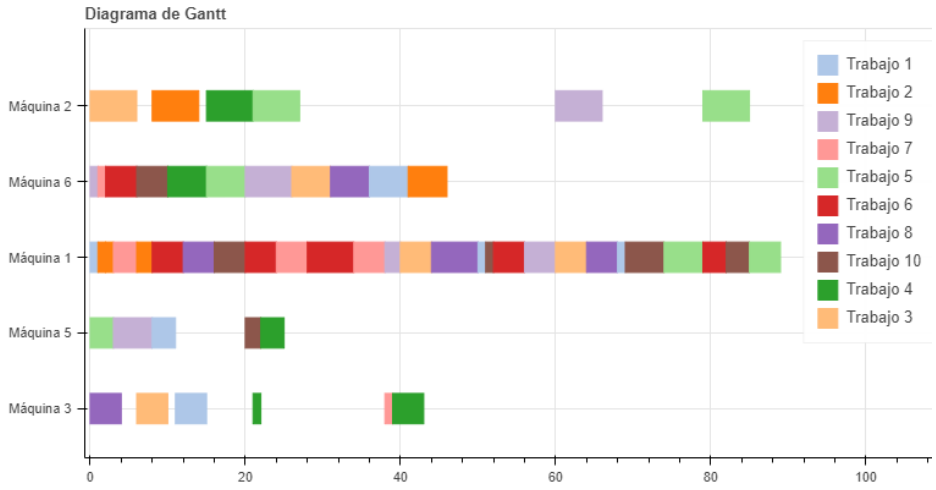

Ilustración 79. Planificación conjunta, Giffler y Thompson, COEF mk01_maq_superpoblada

La tendencia que puede observarse en la Tabla 4 y la Tabla 5 es que, a mayor número de alternativas máximo, es mejor utilizar la planificación separada. En aquellos casos en los que en el taller se disponga de una o dos máquinas con rendimientos muy superiores frente al resto de máquinas, es necesario utilizar planificación conjunta de la producción.

Para solucionar este problema, ya que los rendimientos de la planificación conjunta son notablemente peores, se diseñó el modelo de la planificación separada de la producción con carga distribuida. Si se resuelve el problema en el que una máquina tiene rendimientos muy superiores respecto al resto de máquinas del taller con este modelo, se obtienen las siguientes soluciones (Ilustración 80 e Ilustración 81).

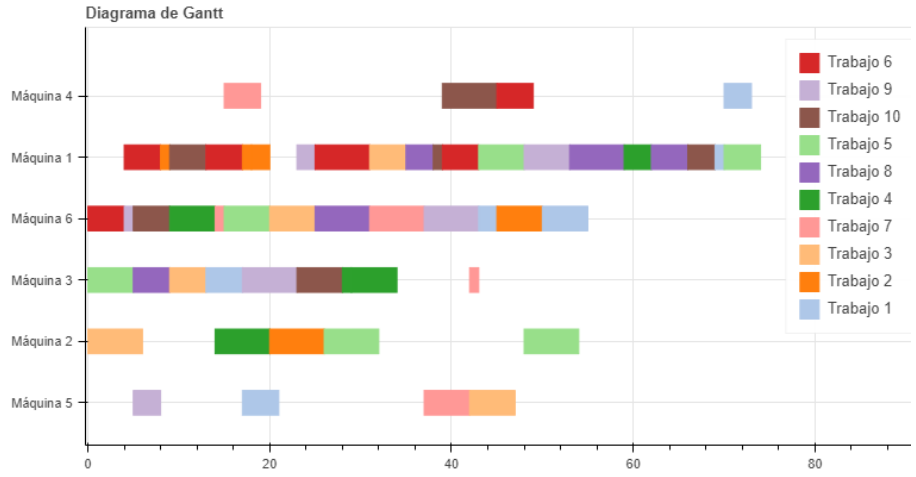

Ilustración 80. Planificación separada con carga distribuida, Lanz, LRPT

Cuando se combina el método constructivo de lanzamiento junto a la heurística de LRPT (Ilustración 80), se obtiene un *makespan* de $Z_{max} = 74$. Este resultado es infinitamente mejor que cuando no se distribuye la carga y únicamente se asigna el mejor modo a cada operación. Este makespan obtenido también es algo mejor que cuando se utiliza el modelo de planificación conjunto.

Si, por ejemplo, se cambia el método constructivo y la heurística; tal y como se observa en la Ilustración 81, el *makespan* obtenido es $Z_{max} = 77$. Estos cambios son ligeros, pero puede notarse que siempre que se use la planificación separada con carga distribuida los rendimientos son buenos.

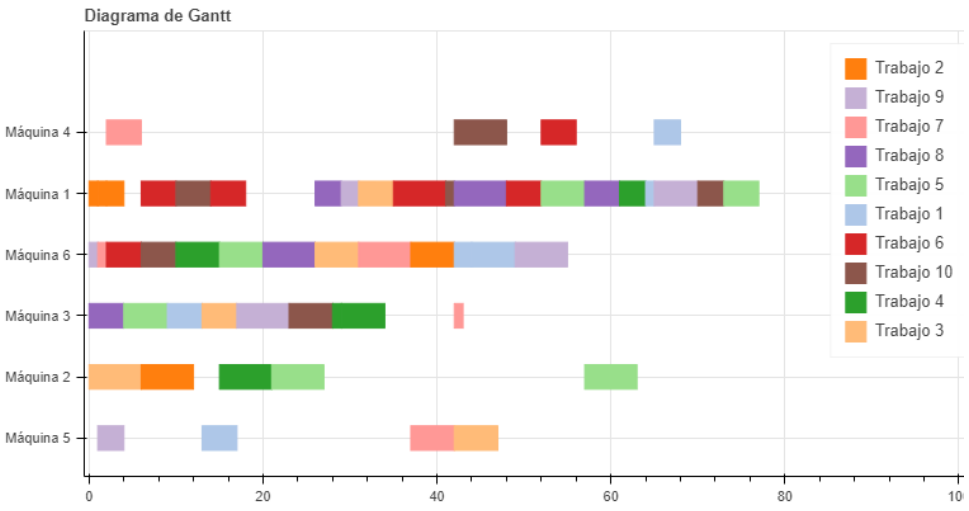

Ilustración 81. Planificación separada con carga distribuida, G y T, COEF

6.5.2 *Flowtime*

El algoritmo elaborado no tiene en cuentas tiempos de entrega ni retrasos, por lo que el aspecto más importante para comparar resultados es el *makespan*. Pero el *Flowtime* (Ft) es otro aspecto que analizar, ya que puede afectar a costos asociados, como el almacenamiento, el manejo de los inventarios o el tiempo ocioso de la maquinaria.

Flowtime	Conjunta		Separada	
	G y T	Lanz	G y T	Lanz
SPT	[75, 95, 70, 93, 116, 106, 89, 106, 118, 103] -> 97,1	[65, 91, 60, 83, 118, 102, 79, 102, 108, 99] -> 90,7	[40, 58, 59, 38, 79, 67, 50, 69, 81, 65] -> 60,6	[59, 49, 54, 37, 70, 75, 49, 60, 78, 56] -> 58,7
LPT	[111, 109, 99, 104, 86, 94, 112, 86, 110, 102] -> 101,3	[63, 113, 88, 110, 106, 99, 111, 102, 121, 93] -> 100,6	[55, 33, 34, 66, 61, 30, 62, 50, 63, 75] -> 52,9	[65, 33, 47, 66, 61, 29, 48, 50, 63, 75] -> 53.7
FIFO	[63, 96, 61, 62, 88, 69, 58, 72, 90, 105] -> 76,4	[62, 91, 68, 54, 107, 82, 56, 85, 109, 100] -> 81,4	[41, 40, 41, 48, 56, 43, 39, 64, 58, 72] -> 50,2	[71, 49, 50, 25, 76, 33, 53, 54, 60, 68] -> 53,9
LRPT	[61, 49, 43, 62, 85, 58, 58, 75, 63, 72] -> 62,2	[51, 64, 42, 53, 80, 76, 54, 88, 82, 73] -> 66,3	[51, 40, 48, 54, 68, 48, 49, 58, 70, 78] -> 56,4	[48, 21, 37, 55, 69, 37, 46, 52, 59, 73] -> 49,7
SRPT	[111, 144, 136, 125, 105, 86, 59, 46, 127, 110] -> 104,9	[96, 139, 129, 98, 102, 136, 92, 58, 128, 103] -> 108,1	[46, 59, 21, 25, 56, 53, 30, 70, 76, 48] -> 48,8	[41, 73, 43, 23, 64, 51, 47, 66, 85, 44] -> 53,7
COEF	[64, 51, 41, 58, 69, 59, 65, 75, 69, 62] -> 61,3	[65, 66, 41, 53, 82, 78, 49, 90, 84, 75] -> 68,3	[42, 33, 41, 48, 56, 44, 40, 64, 58, 72] -> 49,8	[45, 33, 42, 48, 56, 42, 43, 64, 58, 72] -> 50,3

Tabla 6. Flowtimes planificación conjunta y separada, mk01

- Si queremos minimizar el *Flowtime* (F_{max}), la mejor opción sería la heurística FIFO, COEF y LRPT. Entendiendo por F_{max} la media de los *Flowtime*s de todos los trabajos.
- Al escoger la heurística SRPT se consiguen *Flowtime*s mínimos para alguno de los trabajos, pero como el *makespan* para esta heurística es bastante amplio, las medias del *Flowtime* obtenidas son superiores.
- En condiciones de equilibrio entre el rendimiento de las diferentes máquinas, la mejor opción para minimizar el *Flowtime* es la planificación separada.
- No hay una clara diferencia entre los métodos constructivos de *Giffler y Thompson* y Lanzamiento.

Si se analizan los *Flowtimes* para el modelo de planificación separada con carga distribuida, los flowtimes obtenidos son muy similares a los obtenidos cuando la carga no se distribuye, aunque ligeramente peores; siempre y cuando no existan máquinas con muy buenos rendimientos respecto al resto. A continuación, se muestran algunos *Flowtimes* para este modelo (Tabla 7. Flowtimes planificación separada distribuida, mk01).

Flowtime	Separada con carga distribuida	
	G y T	Lanz
SRPT	[37, 89, 65, 58, 56, 52, 30, 24, 76, 85] -> 57.2	[79, 77, 53, 46, 67, 56, 47, 24, 64, 73] -> 58.6
LRPT	[69, 38, 50, 51, 66, 60, 36, 66, 48, 57] -> 54.1	[75, 38, 43, 51, 78, 66, 36, 66, 54, 63] -> 57.0
COEF	[69, 38, 52, 45, 72, 60, 36, 60, 48, 57] -> 53.7	[69, 38, 52, 45, 72, 60, 36, 60, 48, 57] -> 53.7
FIFO	[63, 40, 43, 51, 75, 46, 43, 44, 54, 63] -> 52.2	[69, 44, 43, 51, 72, 66, 39, 46, 54, 63] -> 54.7

Tabla 7. Flowtimes planificación separada distribuida, mk01

6.5.3 Asignación máquinas

El porcentaje de tiempo en el que cada máquina está ocupada es otro factor a tener en cuenta. Ya que cuanto mayor disparidad exista entre ellas; mayor tasa de averías tendrán aquellas que son más utilizadas frente a las que no.

A continuación, se muestra brevemente la carga asignada a cada una de las máquinas cuando se utilizan los distintos modelos de planificación, métodos constructivos y heurísticas (Tabla 8).

Heurística	Conjunta		Separada		Separada distribuida	
	G y T	Lanz	G y T	Lanz	G y T	Lanz
LRPT	Máquina 1: 10.38% Máquina 2: 33.96% Máquina 3: 23.58% Máquina 4: 0.00% Máquina 5: 5.66% Máquina 6: 26.42%	Máquina 1: 10.00% Máquina 2: 38.18% Máquina 3: 26.36% Máquina 4: 0.00% Máquina 5: 2.73% Máquina 6: 22.73%	Máquina 1: 9.80% Máquina 2: 45.75% Máquina 3: 26.14% Máquina 4: 7.84% Máquina 5: 0.65% Máquina 6: 9.80%	Máquina 1: 9.80% Máquina 2: 45.75% Máquina 3: 26.14% Máquina 4: 7.84% Máquina 5: 0.65% Máquina 6: 9.80%	Máquina 1: 13.48% Máquina 2: 20.22% Máquina 3: 6.74% Máquina 4: 20.22% Máquina 5: 7.30% Máquina 6: 32.02%	Máquina 1: 13.48% Máquina 2: 20.22% Máquina 3: 6.74% Máquina 4: 20.22% Máquina 5: 7.30% Máquina 6: 32.02%

COEF	Máquina 1: 10.28% Máquina 2: 33.64% Máquina 3: 25.70% Máquina 4: 0.00% Máquina 5: 4.21% Máquina 6: 26.17%	Máquina 1: 9.38% Máquina 2: 40.18% Máquina 3: 21.43% Máquina 4: 0.00% Máquina 5: 7.59% Máquina 6: 21.43%	Máquina 1: 9.80% Máquina 2: 45.75% Máquina 3: 26.14% Máquina 4: 7.84% Máquina 5: 0.65% Máquina 6: 9.80%	Máquina 1: 9.80% Máquina 2: 45.75% Máquina 3: 26.14% Máquina 4: 7.84% Máquina 5: 0.65% Máquina 6: 9.80%	Máquina 1: 13.48% Máquina 2: 20.22% Máquina 3: 6.74% Máquina 4: 20.22% Máquina 5: 7.30% Máquina 6: 32.02%	Máquina 1: 13.48% Máquina 2: 20.22% Máquina 3: 6.74% Máquina 4: 20.22% Máquina 5: 7.30% Máquina 6: 32.02%

Tabla 8. Porcentaje asignación máquinas, mk01

Se observa claramente que la mayor distribución de carga se obtiene con la planificación separada de la producción con carga distribuida. Es importante resaltar que cuando la planificación es conjunta, dependiendo del problema la carga podría quedar más o menos distribuida, pero siempre de forma controlada, mientras que la planificación separada sin distribución de carga no sigue ningún criterio para repartir la asignación de las máquinas y podría quedar demasiado irregular.

7 ESTUDIO ECONÓMICO

7.1 Introducción

El estudio económico desempeña un papel crucial en la evaluación integral de cualquier proyecto técnico, ya que determina su viabilidad económica y práctica.

En este capítulo, se detallará el coste económico asociado a las diferentes etapas del proyecto. Se describirán las distintas fases consideradas en los cálculos económicos, seguido por una explicación detallada de cómo se calcularon los costes de personal, amortización, material y los gastos generales. Finalmente, se presentarán los costes totales y se establecerá el precio de venta del TFG.

7.2 Fases de desarrollo

A continuación, se hace un desglose de las distintas fases del presente trabajo. Principalmente el trabajo se divide en tres partes bien diferenciadas; cada una de las cuales a su vez consta de un conjunto de actividades.

Recopilación información:

En esta fase inicial del proyecto, se llevó a cabo una inmersión en el campo de la programación de talleres y en el problema específico a abordar: el Job Shop Flexible (FJSP). Este proceso implicó la búsqueda y estudio de la literatura existente, lo que permitió identificar un amplio campo de trabajo para desarrollar un trabajo fin de grado significativo y enriquecedor.

Durante esta exploración, se recopiló información relevante sobre el tipo de problema del que se iba a realizar el estudio, la caracterización de este y la diversidad de formas para resolver este problema, donde se incluyen un conjunto de heurísticas y metaheurísticas con distintos métodos y esquemas de construcción.

Elaboración del programa:

En esta segunda fase, en primer lugar, se realizó un estudio de la mejor alternativa posible para la realización del programa que resolviera el problema FJSP. Tras un análisis de las ventajas y desventajas entre las diferentes alternativas, se optó por la utilización del Python para abordar el problema.

A continuación, se comenzó con el aprendizaje de este lenguaje desde una base muy baja. Pero a esta fase no se le dedicó un gran número de horas, debido a que preferí ir aprendiendo a resolver los problemas que aparecían a la hora de elaborar el programa, a medida que iban surgiendo.

Por último, se abordó la realización del programa; la fase que mayor número de horas a supuesto; pues esta incluye tanto la realización del algoritmo a papel, como la programación de este.

Análisis de resultados:

Una vez el programa había sido terminado, se comenzó con el análisis y comparación de los resultados obtenidos. En esta fase, también hubo que realizar pequeñas modificaciones en el programa para poder obtener resultados de manera más cómoda y clara.

7.3 Cálculo de costes

Para realizar adecuadamente este estudio económico, es necesario calcular las horas efectivas trabajadas y transformarlas en el salario bruto de un ingeniero. Este cálculo implica estimar los días efectivos de trabajo.

Para calcular el número de horas empleadas, se va a suponer una media de hora y media de trabajo al día durante 6 meses, incluyendo festivos y fines de semana; que es el tiempo que ha sido dedicado a la realización del presente trabajo. Lo que da como resultado un total de 270 horas de trabajo.

Si se aproxima este número de horas a la jornada laboral de 8h, teniendo en cuenta festivos, fines de semana y pausas durante la jornada laboral, entre otros aspectos; se obtiene el número de meses necesarios para realizar este estudio; tal y como muestra la Tabla 9.

Nº días de trabajo por mes	**22d**
-días festivos	-1d
-días vacaciones prorrateados	-2d
-media de días perdidos por enfermedad mensual	-1d
Total, días efectivos mensuales	18d
Total horas teóricas	144h
-horas descanso durante la jornada (20 min al día)	-6h
Total horas efectivas mensuales	138h

Tabla 9. Horas efectivas mensuales de un trabajador.

7.3.1 Costes de personal

En el presente trabajo vamos a suponer que los costes de personal (también llamados costes salariales) son los correspondientes al salario de un ingeniero de organización industrial.

Si se considera que el número de horas necesario para realizar el presente trabajo ha sido de 270 horas y que el número efectivo de horas mensuales de un trabajador medio es de 138 horas; se obtiene que prácticamente se necesitaría el coste salarial de dos meses.

Sueldo bruto anual medio	**28000€**
Sueldo bruto 2 meses	4666,67€

Tabla 10. Coste salarial

A continuación, se va a asignar a cada etapa o actividad el número de horas dedicadas y así, se calcula el coste por actividad. Permitiendo calcular el porcentaje de cada actividad sobre el total.

Actividad	Horas necesarias	Coste por actividad	% sobre el total
Recopilación información	98	1693,83	36,30%
Elaboración programa	129	2229,63	47,78%
Análisis de resultados	43	743,21	15,92%
Total	270	4666,67	100%

Tabla 11. Coste salarial desagregado

7.3.2 Costes de amortización

En esta sección, se llevará a cabo el cálculo de las amortizaciones de los equipos empleados en la realización de este trabajo de fin de grado. Para llevar a cabo esta tarea, se utilizó un ordenador personal junto con un conjunto de periféricos y licencias. El periodo de amortización se especifica en la Tabla 12, así como el coste total para el periodo utilizado de dicho equipo.

Concepto	Coste (€)	Periodo amortización	Coste 2 meses (€)
Ordenador AMD Ryzen 7 5800X/32GB/1TB SSD/RTX 4060 Ti	1349,00	8 años	28,10
Monitor MSI G2722 27”	230,00	8 años	4,79
Monitor auxiliar HP	60,00	8 años	1,25
Office 365	99,00	1 año	16,50
Licencia Python	0,00	1 año	0,00
Periféricos	210,00	5 años	7,00
Total	2029,00		57,64

Tabla 12. Coste y amortización requeridas

7.3.3 Costes generales

Los costes generales, también conocidos como indirectos, vienen representados en la Tabla 13.

Gastos generales	Coste (€, 2 meses)
Electricidad	143
Internet	61
Teléfono	40
Agua	25
Total	269

Tabla 13. Costes indirectos.

7.3.4 Costes totales

La evaluación económica de este proyecto de fin de grado se compone de la suma de los costes individuales identificados en secciones previas. A continuación, se presenta la Tabla 14, que muestra estos costes desglosados, junto con el porcentaje que cada partida representa respecto al total de los costes.

Concepto	Coste (€)	% sobre el total
Coste de personal	4666,67	93,46%
Costes de amortización	57,64	1.15%
Costes indirectos	269,00	5,39%
Coste total	**4993,31**	**100%**

Tabla 14. Coste total proyecto

8 CONCLUSIONES

La programación de tareas en talleres flexibles es un campo de estudio en el que se han aplicado en la gran mayoría de ocasiones metaheurísticas para ser resueltos; obteniendo resultados de gran calidad.

En el presente trabajo, se ha tratado de resolver el problema mediante métodos heurísticos constructivos. Los cuales requieren mucha menos carga computacional para obtener resultados razonablemente buenos.

Como es obvio, con los métodos constructivos se obtienen peores soluciones que con las metaheurísticas, como podrían ser las de poblaciones o trayectorias. Pero estos, debido al bajo coste computacional que requieren podrían ser utilizados como punto de partida sobre el que aplicar estas metaheurísticas.

En la planificación conjunta, cuando se combina con el esquema de construcción de Giffler y Thompson y heurísticas como LRPT o COEF, las soluciones obtenidas son aceptables, pudiéndose adaptar a todo tipo de problema. Esto es posible, ya que a medida que se va asignando una máquina a una operación, se va temporizando esta operación y analiza el escenario del problema en su conjunto en cada iteración.

En la planificación separada, como todas las operaciones se asignan a las máquinas a la vez para posteriormente ir temporizando, se pierde esta visión global del problema. Por lo que este tipo de planificación no se adapta igual de bien a todo tipo de situaciones.

Por regla general, con la planificación separada se obtienen mejores rendimientos, tanto en makespan como en flowtime; exceptuando el caso en el que algunas máquinas tienen mejores rendimientos que otras. En la asignación de las máquinas a las operaciones, se realizan sobreasignaciones a alguna de las máquinas. Para resolver este problema se diseña la planificación separada con distribución de carga; la cual evalúa la carga de trabajo de cada máquina tras una asignación inicial y se redistribuye la carga, homogeneizándola.

Otro inconveniente de la planificación separada es que se debe tener claro en el taller todos los trabajos que van a ser realizados; ya que se necesita asignar todas las máquinas a las operaciones de golpe; por la que hay que saber de antemano esta información. En la planificación conjunta, esto no ocurre. Si por cualquier motivo, un trabajo llega al taller de manera imprevista, con este tipo de planificación; como no se necesita hacer una asignación de todas las máquinas a las operaciones; se puede insertar dentro del problema algún trabajo adicional y el programa seguiría funcionando correctamente (siempre y cuando se eliminen del problema todas las operaciones que ya han sido procesadas).

Debido al bajo coste computacional que conlleva la resolución del Problema Job Shop Flexible mediante métodos constructivos, se puede resolver el problema con diversas combinaciones y una vez guardados los *makespans*, *flowtimes* y porcentajes de asignación de cada una, quedarse con la que más convenga en cada situación.

Bibliografía

A. Amuthan and K. Deepa Thilak. (2016). Survey on Tabu Search meta-heuristic optimization,. *International Conference on Signal Processing, Communication, Power and Embedded System (SCOPES)*, (págs. 1539-1543). Paralakhemundi, India.

Araúzo, J. A. (2022). *Dirección de operaciones.* Obtenido de Campus virtual UVA: https://campusvirtual.uva.es/

Arno Sprecher, R. K. (1995). Semi-active, active, and non-delay schedules. *European Journal of Operational Research*, 94-102.

Baker, K. R. (1974). Introduction to sequencing and scheduling.

Brandimarte, P. (1999). Exploiting process plan flexibility in production scheduling: a multi-objective approach. *Eur. J. Oper. Res.*, 59-71.

C Wang, C. C.-M. (1997). Heuristic approaches for n/m/F/∑Ci scheduling problems. *European Journal of Operational Research*, 639-644.

Christian Bierwirth, D. C. (1996). On permutation representations for scheduling problems. *PPSN IV: International Conference on Evolutionary Computation* (págs. 310-318). Berlin, Germany,: Springer Berlin Heidelberg.

Conway, R. W. (1967). Theory of scheduling Addison.

Crespo, C. R. (2022). *Resolución del Job Shop Scheduling Problem mediante reglas de prioridad.* Universidad de Oviedo.

Dorigo, M. &. (1992). An Investigation of some Properties of an" Ant Algorithm".

Durasevic, M. (2020). *Automatic design of dispatching rules for static scheduling conditions.* Londres.

E Ignall, L. S. (1965). Application of the branch and bound technique to some flowshop scheduling problem. *Operations Research*, 405-412.

F. Pezzellaa, ∗. G. (2008). A genetic algorithm for the Flexible Job-shop Scheduling Problem. *Computers & Operations Research 35*, 3202-3212.

French, S. (1982). Sequencing and scheduling. An Introduction to the Mathematics of the Job-shop.

Heredero, J. D. (2022). Métodos heurísticos y metaheurísticos, MÉTODOS CUANTITATIVOS EN INGENIERÍA DE ORGANIZACIÓN II, Escuela de Ingenierías Industriales (UVa). Valladolid.

Ho, J. (1995). Flowshop sequencing with mean flow time objective. *European Journal of Operational Research*, 571-578.

Hofmann, P. A. (1993). Critical path method: an important tool for coordinating clinical care. *The Joint Commission journal on quality improvement,*, 235-246.

II, M. C. (2022). Métodos heurísticos y metaheurísticos. Valladolid.

Juan Carlos Osorio*, T. G. (2007). Planificación jerárquica de la producción en un. Cali.

Kacem I, H. S. (2002). Approach by localization and multiobjective evolutionary optimization for flexible job-shop scheduling problems. *IEEE Transactions on Systems*, 1-13.

Kan, A. R. (2012). Machine scheduling problems: classification, complexity and computations.

Li-Ning Xing *, Y.-W. C.-W. (2009). Multi-objective flexible job shop schedule: Design and evaluation by. *Applied Soft Computing 9* , 362-293.

Machuca, D. (1998). *Direcciónde operaciones: aspectos tácticos y operativos en la producción y los servicios.* MacGraw-Hill.

Marta Posada. (2020). *Planificación, programación y control de proyectos*. Obtenido de Campus virtual UVA: https://campusvirtual.uva.es/

Mattfeld, D. (1995). Effective neighborhood functions for the flexible Job Shop Scheduling Ploblems. *Proc. of 6th Intern. Conf. of Genetic Algorithms*.

Meal, A. C. (1973). *HIERARCHICAL INTEGRATION OF PRODUCTION PLANNING.*

Melian, B. P. (2003). Metaheuristicas: una vision global. *Revista Iberoamericana* , 7-28.

Özgüven, C. (s.f.). Mathematical models for job-shop scheduling problems with routing and process plan flexibility. *Applied Mathematical Modelling*, 1539-1541.

Pastor, R. T. (2011). Planificación y programación de operaciones. 10-13.

Pinedo, M. L. (2009). *Planning and Scheduling in Manufacturing and Services.*

S.A, M. (1989). *Planificación y programación de la producción.* Barcelona: Companys-Pascual R.

Schneeweis, D. M. (1995). Photovoltage of rods and cones in the macaque retina. *Science, 268(5213)*, 1053-1056.

Sörensen, K. &. (2013). Metaheuristics. En *Encyclopedia of operations research and management science* (págs. 960-970).

Szwarc, W. (1983). The flow-shop problem with mean completion time criterion. *AIIE Transactions, 15(2),*, 172-176.

Villahoz, J. J. (2013). *Análisis de la relajación lagrangiana como método de programación de talleres flexibles en un entorno multiagente.* Universidad de Burgos.

Wu Deng, J. X. (2019). *An Improved Ant Colony Optimization Algorithm Based on Hybrid Strategies for Scheduling Problem.*

Yamada, T. &. (1996). Job-shop scheduling by simulated annealing combined with deterministic local search. *Meta-heuristics: Theory and applications*, 237-248.