Trabajo Fin de Grado
Grado en Ingeniería de Tecnologías Industriales

Modelado y Optimización de Líneas de Ensamblaje Low-Volume Mixtas

Autor: Carlos De la Cruz Bailén

Tutor: Jose Manuel Framiñán Torres

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

Sevilla, 2018

Trabajo Fin de Grado
Grado en Ingeniería de Tecnologías Industriales

Modelado y Optimización de Líneas de Ensamblaje Low-Volume Mixtas

Autor:
Carlos De la Cruz Bailén

Tutor:
Jose Manuel Framiñán Torres
Profesor Titular

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

Sevilla, 2018

Trabajo Fin de Grado: Modelado y Optimización de Líneas de Ensamblaje Low-Volume Mixtas

Autor: Carlos De la Cruz Bailén
Tutor: Jose Manuel Framiñán Torres

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

Presidente:

Vocal/es:

Secretario:

acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

Agradecimientos

A mi familia por apoyarme siempre.
A mi tutor D.Jose Manuel Framiñán Torres por su ayuda y atención en todo momento.
Sevilla, 2018

Resumen

En el presente trabajo fin de grado, se considerarán los problemas de planificación de producción que engloban las líneas de ensamblaje de bajo volumen para modelos mixtos enfocadas al sector aeronáutico. Para ello, se propondrá y desarrollará un modelo de programación lineal para aplicarlo a un caso práctico de este sector que permita la reducción de costes asociados a mantenimiento de inventario debido al procesamiento de componentes de gran volumen, así como la reducción de costes laborales originados por la gran cantidad de mano de obra que caracterizan a estos tipos de línea. Para el análisis de su aplicabilidad y desempeño, los experimentos computacionales se generarán en lenguaje Python para posteriormente analizar los resultados obtenidos y las distintas ineficiencias que pueden generarse a lo largo del flujo de valor.

Abstract

In the present final degree project, the production planning problems for mixed-model low-volumen assembly lines focused on the aeronautical sector will be considered. A linear programming model will be proposed and developed in order to apply it to a case study from this sector with the purpose of reducing inventatory holding costs due to the huge-sized jobs production as well as the reduction of total labour costs caused by the large amounts of tasks to be manually performed that characterizes this kind of lines. For the analysis of its applicability and performance, the computational experiments will be generated in Python in order to analyze afterwards the obtained results and the different inefficiencies that can be generated along the value chain.

Índice

Resumen III
Abstract V

Índice de Figuras IX
Índice de Tablas XI

1 Introducción 1
1.1 Introducción y objeto del proyecto 1
1.2 Sumario sobre la estructura del proyecto 2

2 Antecedentes 3
2.1 Introducción a las Líneas de Ensamblaje Low-Volume Mixtas 3
2.1.1 Introducción 3
2.1.2 Características principales de las Líneas de Ensamblaje Low-Volume Mixtas 5
2.2 Formalización del problema 6
2.2.1 Descripción del problema 6
2.2.2 Objetivo del problema 8
2.3 Descripción del caso práctico 10

3 Metodología 13
3.1 Introducción a la programación lineal 13
3.1.1 Bases de la programación lineal 13
3.1.2 Programación Multi-Objetivo 15
3.2 Bases del modelado matemático 18
3.2.1 Concepto de modelo 18
3.2.2 Desarrollo del proceso de modelado 19
3.3 Introducción a la optimización en lenguaje Python 20
3.3.1 Introducción a Pyomo 21
3.3.2 Modelado computacional y comandos usuales 21

4 Modelado y aplicación al caso práctico 25
4.1 Modelo matemático 25
4.1.1 Formulación matemática 25
4.1.2 Análisis del modelo matemático 26
4.2 Modelado computacional 32
4.2.1 Bibliotecas externas implementadas 32

4.2.2 Modelado de índices y conjuntos 32
4.2.3 Modelado de parámetros y variables 42
4.2.4 Modelado de la función objetivo y restricciones 44

5 Resultados 49
5.1 Resultados computacionales 49
5.2 Análisis experimental 54
5.2.1 Análisis de requisitos laborales 54
5.2.2 Análisis multi-objetivo 56
5.2.3 Análisis de la eficiencia laboral 59

6 Extensiones al modelo matemático 63
6.1 Introducción y descripción de la problemática 63
6.2 Propósito y objetivos del problema 65
6.3 Formulación matemática 68
6.4 Aplicaciones prácticas 72

7 Conclusiones 75

Apéndice A Codifiación en Python 77
A.1 Ejemplo multi-objetivo: Método de las restricciones 77
A.2 Ejemplo introducción a Pyomo 78
A.3 Caso práctico 78
A.4 Problema monoestacional 85
A.5 Modelo adicional. Ejemplo práctico 90
Bibliografía 93

Índice de Figuras

2.1 Ejemplo de un diagrama de precedenica para 7 actividades [Fuente: Elaboración propia] 4
2.2 Representación de una línea de modelo único junto con otra línea mixta [Fuente: Elaboración propia] 4
2.3 Fotografía del modelo C-295 AEW (a la izquierda) y el modelo C-295 MSA (a la derecha) [Fuente: [4], [5]] 5
2.4 Disposición de las estaciones de una línea de ensamblaje [Fuente: Elaboración propia] 7
2.5 Diagrama de Gantt del primer caso [Fuente: Elaboración propia] 9
2.6 Diagrama de Gantt del segundo caso [Fuente: Elaboración propia] 9
2.7 Estación 60 de ensamblaje en la planta de Airbus-San Pablo Sur [Fuente: [2]] 10
2.8 Disposición de la estructura de la línea de ensamblaje para el caso práctico [Fuente: Elaboración propia] 11

3.1 Representación gráfica de la región admisible y solución óptima del problema.[Fuente:[24]] 14
3.2 Ejemplo ilustrativo de la optimalidad de Pareto [Fuente: Elaboración propia] 16
3.3 Representación gráfica de la frontera de Pareto mediante el método de las restricciones [Fuente: Elaboración propia] 18
3.4 Relación entre un proceso real y un proceso modelado [Fuente: [33]] 19
3.5 Diagrama de flujo del proceso de modelado [Fuente: Elaboración propia] 20

4.1 Disposición de las etaciones y los órdenes de precedencia entre trabajos del problema biestacional [Fuente: Elaboración propia] 27
4.2 Diagrama de Gantt del problema biestacional para el análisis matemático [Fuente: Elaboración propia] 28
4.3 Representación gráfica de la familia de conjuntos indexados de P_i [Fuente: Elaboración propia] 30
4.4 Diagrama de precedencia y su matriz de precedencia asociada para un ejemplo de 3 estaciones [Fuente: Elaboración propia] 36

5.1 Diagrama de Gantt resultante del caso práctico [Fuente: Elaboración propia] 51
5.2 Variaciones de los requisitos laborales por cada estación y modelo de la linea de ensamblaje [Fuente: Elaboración propia] 54
5.3 Gráfico de la variación de la eficiencia laboral frente a los requisitos laborales totales [Fuente: Elaboración propia] 55
5.4 Representación gráfica del frente de Pareto [Fuente: Elaboración propia] 59
5.5 Gráfico variación del número operarios ociosos frente al número de operarios totales por estación [Fuente: Elaboración propia] 60
5.6 Gráfico variación del Lead Time frente al número de operarios totales por estación [Fuente: Elaboración propia] 60

5.7 Gráfico de la variación de la eficiencia laboral frente al número total de operarios asignados [Fuente: Elaboración propia] 62

6.1 Línea de ensamblaje compuesta por 2m+1 estaciones [Fuente: [12]] 63
6.2 Definición del tiempo de flujo de un trabajo i para un modelo j [Fuente: [12]] 64
6.3 Datos requeridos por cada estación [Fuente: Elaboración propia] 66
6.4 Representación del flujo de tiempo para el primer caso [Fuente: Elaboración propia] 67
6.5 Representación del flujo de tiempo para el segundo caso [Fuente: Elaboración propia] 67
6.6 Diagrama de Gantt ejemplo práctico [Fuente: Elaboración propia] 74

Índice de Tablas

2.1 Características que diferencian una línea Low-Volume a una línea High-Volume. [Fuente: [38]] 6
2.2 Datos que conforman una línea de 2 estaciones [Fuente: Elaboración propia] 8
2.3 Datos del modelo 1 para cada una de las estaciones [Fuente: Elaboración propia] 11
2.4 Datos del modelo 2 para cada una de las estaciones [Fuente: Elaboración propia] 12

3.1 Resultados experimentales del problema mediante el método de las restricciones [Fuente: Elaboración Propia] 17
3.2 Definición las clases más importantes utilizadas en Pyomo [Fuente: Elaboración propia] 21

4.1 Datos para la línea de ensamblaje del problema biestacional para el análisis matemático [Fuente: Elaboración propia] 28
4.2 Tabla Excel que muestra los datos de la estructura de la línea [Fuente: Elaboración propia] 33
4.3 Tabla Extracto Excel para la codificación de la matriz estaciones [Fuente: Elaboración propia] 35
4.4 Tabla Excel que almacena los datos de precedencia de la línea de ensamblaje [Fuente: Elaboración propia] 37
4.5 Extracto de una hoja Excel que almacena los distintos parámetros dato [Fuente: Elaboración propia] 43

5.1 Resultados Experimentales del caso práctico [Fuente: Elaboración propia] 50
5.2 Resultados experimentales frente a una variación de los requisitos laborales totales [Fuente: Elaboración propia] 55
5.3 Datos del problema monoestacional con libre asignación de operarios totales [Fuente: Elaboración propia] 57
5.4 Resultados experimentales del problema monoestacional obtenidos para conformar la matriz de pago [Fuente: Elaboración propia] 57
5.5 Resultados experimentales del problema monoestacional para el conformado del frente de Pareto [Fuente: Elaboración propia] 58
5.6 Resultados experimentales obtenidos para el estudio de la eficiencia laboral [Fuente: Elaboración propia] 61

6.1 Datos ejemplo práctico por cada estación [Fuente: Elaboración propia] 72
6.2 Operarios disponibles por periodo [Fuente: Elaboración propia] 72
6.3 Resultados experimentales ejemplo práctico [Fuente: Elaboración propia] 73

1 Introducción

1.1 Introducción y objeto del proyecto

El proyecto que se presenta considera los problemas de planificación de la producción de las líneas de ensamblaje Low-Volume (bajo volumen de producción) Mixtas, donde se resolverá un problema real de una empresa del sector aeronáutico. Se tomará como referente este sector debido a su caracterización por la alta mano de obra requerida y el bajo volumen de producción de actividades sobre productos de grandes dimensiones.

Las líneas de ensamblaje Low-Volume Mixtas deben adaptarse a un mercado exigente con altos costes de mantenimiento de inventario debido a los componentes de grandes dimensiones y costes de mano de obra, donde mayoritariamente, existe una escasa automatización en la producción siendo requerida de forma indispensable la mano de obra laboral. Teniendo esto en cuenta, el presente proyecto busca la reducción de los costes asociados a la duración de la vida del proyecto traduciéndose en costes de mantenimiento de inventario, la reducción del número de operarios ociosos a lo largo de la línea, así como los costes totales laborales. Para ello, se propone y desarrolla un modelo de programación lineal que se resolverá y analizará en profundidad, además de un modelo adicional que haremos referencia al final del documento debido a su gran potencial.

El modelo desarrollado en el proyecto se enfoca en la planificación a nivel estacional, donde cada estación de la línea de ensamblaje tiene asignado un número disponible de operarios caracterizados por tres especialidades diferentes además de las actividades que componen cada una de ellas con sus órdenes de precedencia (con una secuencia de trabajo preestablecida). El objeto del modelo es buscar la asignación de los operarios atendiendo a las demandas de cada trabajo para su procesamiento, así como los tiempos de inicio de cada trabajo que consigan como objetivo la minimización del Lead Time (definido como el tiempo que transcurre desde que se inicia el proceso hasta que termina produciéndose el último modelo de la línea de ensamblaje) de la ventana de planificación para todos los modelos que transcurren por la línea, además del número de operarios ociosos totales generados. El modelo adicional se caracteriza por permitir la libre asignación de operarios dado un número disponible de operarios por periodo y estación. Con este modelo, se busca la minimización de los costes de mantenimiento de inventario y los costes laborales totales.

Así, en el proyecto se aplican las metodologías y modelos desarrollados a un caso práctico del sector aeronáutico mediante experimentos computacionales modelados en Python que demuestran el gran impacto de la programación lineal en problemas de planificación de la producción para líneas de ensamblaje Low-Volume Mixtas. Por último, se propone un análisis de los resultados experimentales obtenidos, para, de esta forma, poder valorar así las sinergias y correlaciones entre

las distintas variables que componen una línea de ensamblaje en busca de posibles ineficiencias que se puedan generar.

1.2 Sumario sobre la estructura del proyecto

El documento se desarrollará en seis capítulos y un apéndice que contiene los distintos códigos en Python. A continuación, se procederá a describir cada uno de los puntos que lo componen para facilitar la identificación de los mismos:

En primer lugar, el capítulo 1 "Introducción" presenta el objetivo general del proyecto. Posteriormente, se describen y resumen los distintos capítulos que componen el documento a modo de sumario.

El capítulo 2 "Antecedentes" expone una introducción a las líneas de ensamblaje Low-Volumen Mixtas junto con las características principales que las definen. La descripción de la problemática identificada en este tipo de líneas, así como el propósito que se busca para su mejora se describen también en este punto, finalizando, por último, con la descripción del caso práctico en el sector aeronáutico con el que trabajaremos para la aplicación de los métodos y modelos desarrollados a lo largo del documento.

El capítulo 3 "Metodología" introduce las bases teóricas que se van a aplicar a nuestro caso práctico. Entre ellas, la programación lineal debido a su papel principal para la búsqueda de soluciones óptimas en problemas de optimización, las bases del modelado matemático aplicado para el desarrollo de nuestros modelos matemáticos que describiremos y por último una introducción al lenguaje Python, donde se detalla la librería Pyomo para la modelización computacional y sus comandos usuales.

En el capítulo 4 "Modelado y aplicación al caso práctico" se desarrollará el modelado matemático asociado a la problemática descrita en el punto 2.2. Seguidamente, se trasladará el modelo aplicado de nuestro caso práctico al ámbito computacional, explicando los diferentes comandos auxiliares necesarios para el correcto modelado en lenguaje Python.

El capítulo 5 "Resultados" expondrán los diferentes resultados experimentales obtenidos junto con una evaluación de los mismos, para, posteriormente, realizar un análisis experimental de los requisitos laborales, las diferentes soluciones que se pueden obtener en base a variaciones de parámetros y variables y un análisis de la eficiencia laboral.

El capítulo 6 "Extensiones al modelo matemático" contiene una extensión al primer modelo desarrollado con el que se ha trabajado a lo largo del proyecto. En este capítulo se define la problemática y el propósito que se busca para proponer después la formulación matemática del modelo. El capítulo finaliza con una aplicación práctica mediante un ejemplo que demuestre el potencial del mismo.

Por último, el capítulo 7: "Conclusiones" recoge las distintas conclusiones, valoraciones, y cuestiones abiertas que se han ido generando a lo largo del desarrollo del proyecto.

2 Antecedentes

En este capítulo se recogerán, por un lado, las bases de las líneas de ensamblaje Low-Volume Mixtas, así como las características principales que las diferencian de otras para posteriormente introducir la problemática que se plantea en el proyecto en el punto 2.2.1. El propósito y por tanto, el objetivo concreto de nuestra problemática se detalla en el apartado 2.2.2 mientras que el último punto está destinado a la descripción del caso práctico con el que trabajaremos a lo largo del documento.

2.1 Introducción a las Líneas de Ensamblaje Low-Volume Mixtas

2.1.1 Introducción

Las Líneas de Ensamblaje son sistemas de producción de flujo orientado muy cotidianos en la producción industrial para altas cantidades de productos estandarizados además de la producción de productos personalizados de bajo volumen (Low-Volume) [10]. Las líneas de ensamblaje están diseñadas para una organización que implican una serie de operaciones realizadas en el mismo orden en todos los trabajos y organizadas bajo unas estaciones de trabajo, las cuales [16]:

1. Reciben la misma secuencia de trabajo fija.
2. Están acopladas entre sí para no almacenar Work in Process (WIP). Definido según [6], como materias primas, subconjuntos y piezas parcialmente procesadas que no forman parte del inventario de productos terminados.
3. Están equilibradas para que los trabajos se muevan bajo un flujo continuo a ritmo constante.

Los productos pertenecientes a una línea de ensamblaje requieren una serie de operaciones de procesamiento llamadas actividades. La realización de una actividad i requiere un tiempo de proceso t_i y una serie de recursos tales como maquinaria, herramientas y/o trabajadores. Debido a las condiciones tecnológicas y organizativas en una línea de ensamblaje, es común la imposición de restricciones en condiciones de precedencia entre trabajos. Estas restricciones pueden ser resumidas y visualizadas en un grafo de precedencia. Un grafo de precedencias contiene nodos por cada actividad, pesos asignados al nodo para los tiempos de proceso y arcos que definen las restricciones de precedencia [8]. La figura 2.1 muestra un diagrama de precedencia con $n = 7$ actividades. El número interno por cada círculo representa el número de actividad mientras que el número fuera del circulo muestra el tiempo de proceso.

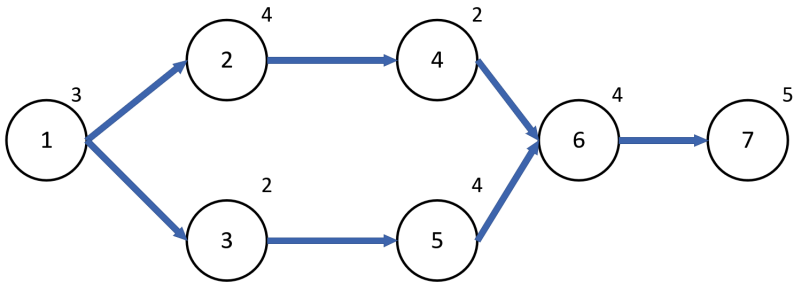

Figura 2.1 Ejemplo de un diagrama de precedenica para 7 actividades [Fuente: Elaboración propia].

Hoy en día, debido a los niveles de globalización y la alta competitividad que existe en el mercado, provocando grandes necesidades de flexibilidad y dinámica en respuesta [37], la personalización en masa de productos se ha convertido en un nuevo objetivo de la fabricación con la misión de satisfacer distintas necesidades de clientes con distintos productos personalizados y bajo una eficiencia de producción y coste [27]. Debido al nuevo mercado exigente, el número de variedades de producto ofertados en el mercado se ha ido incrementando drásticamente. Sin embargo, el incremento de la variedad no supone necesariamente una mejora de los beneficios frente a un incremento en ventas, ya que los costes y complejidad de la fabricación son añadidos como una nueva variable. Para poder alcanzar los máximos beneficios, la oferta de productos y los sistemas de fabricación deberían diseñarse conjuntamente.

Las líneas de ensamblaje de modelos mixtos son ampliamente usadas para la fabricación en masa de productos personalizados. Estos sistemas requieren factores humanos para las operaciones de ensamblado. Con operaciones flexibles manuales, los sistemas de modelo mixtos son capaces de tratar el incremento de la variedad con una fabricación de múltiples productos bajo un mismo sistema. De esta forma, las inversiones y fluctuaciones de demanda son mitigadas.

La figura 2.2 muestra la diferencia entre una línea de ensamblado simple y una mixta.

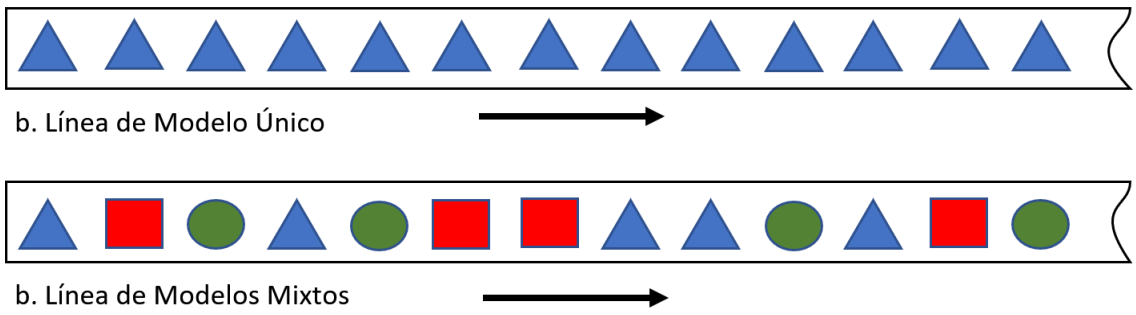

Figura 2.2 Representación de una línea de modelo único junto con otra línea mixta [Fuente: Elaboración propia].

A pesar de las ventajas anteriores, el proceso manual de ensamblado puede volverse complejo en caso de incremento en la variedad de producto, el cual, repercute en un impacto negativo al rendimiento del sistema, tales como calidad y productividad [19].

Por tanto, la variedad de producto en una línea de ensamblaje induce a una complejidad de fabrica-

ción que repercute hoy en día en los sistemas de ensamblaje de modelos mixtos. Por ejemplo, una línea de modelos mixtos provoca la dificultad de división de trabajo entre estaciones y demanda y más tiempo de entrenamiento para los operarios debido a las diferencias en las características de cada modelo. Además, las líneas de modelos mixtos sufren retrasos, y tiempos ociosos a lo largo del flujo de valor, llevando a reducciones en su eficiencia.

A lo largo del documento, trabajaremos con líneas de ensamblaje Low-Volume Mixtas enfocadas al sector aeronáutico para su optimización. La industria aeroespacial está caracterizada por trabajar bajo mano de obra intensiva, y una producción compleja de bajo volumen que cumplan con la normativa de seguridad aérea. Además, las líneas mixtas juegan un papel importante bajo diferentes modelos de producto, por ejemplo, diferentes modelos de avión. La figura 2.3 muestra algunos de los diferentes modelos de aviones C-295 ensamblados en la planta de San Pablo Sur (Sevilla).

Figura 2.3 Fotografía del modelo C-295 AEW (a la izquierda) y el modelo C-295 MSA (a la derecha) [Fuente: [4], [5]].

2.1.2 Características principales de las Líneas de Ensamblaje Low-Volume Mixtas

A continuación, enumeraremos las características principales más comunes que distinguen a las Líneas de Ensamblaje Low-Volume Mixtas frente a otras. Estas características se han tomado en parte al análisis de la literatura como [23] y del caso práctico al haber realizado prácticas en una empresa del sector aeronáutico:

- Algunas de las estaciones están equipadas por recursos auxiliares como grúas o elevadores.
- Son obligatorios los vehículos de transporte de componentes debido a sus grandes dimensiones.
- Mayormente se emplean recursos humanos para la ejecución de la producción o de tareas de ensamblado.
- Debido al gran tamaño de algunos de los componentes, pueden existir buffers de cantidad limitada en cada estación.
- Debido al almacenamiento limitado de buffers, puede ocurrir bloqueos completos del flujo de la línea.
- En cada estación suele haber lugares de almacenamiento de componentes necesarios para el ensamblaje de los productos.
- Cada trabajo requiere un cierto tipo de especialidad de los operarios.

- Las principales características que diferencian las líneas High-Volume y Low-Volume se describen en la tabla 2.1.
- Es usual el empleo de instalaciones de producción fijas, es decir, el producto a procesar en cada estación no realiza movimientos hasta su traslado a la siguiente.

Tabla 2.1 Características que diferencian una línea Low-Volume a una línea High-Volume. [Fuente: [38]].

Criterio	High-Volume	Low-Volume
Sistema de objetos.	Muchos de ellos, normalmente de reducido tamaño.	Normalmente, pequeño número de objetos de grandes dimensiones.
Características del producto.	A menudo productos básicos o productos con un grado relativamente bajo de personalización, listas de materiales y rutas no muy complejas.	Altamente personalizado, con lista de materiales normalmente complejo y múltiples rutas, alto grado de paralelismo.
Tiempos de proceso	Los tiempos de proceso dependen de la maquinaria específica donde se elabora la operación.	Dependen de las habilidades y la cantidad de operarios.
Grado de automatización	Alto.	Normalmente bajo, con mano de obra intensiva.
Origen de la complejidad	Flow shop (producción en masa) o producción tipo taller, muchos objetos de sistema.	Personalización, gran número de proveedores, alto personal cualificado, se producen curvas de aprendizaje laboral, restricciones espaciales.
Recursos	Normalmente estacionarios, por ejemplo, maquinaria o pequeños recursos auxiliares.	Recursos móviles, los trabajadores constituyen un recurso esencial, los recursos auxiliares son importantes.
Buffers	Posibles, normalmente no son una gran restricción	Normalmente mayor restricción debido al tamaño de los objetos del sistema.

2.2 Formalización del problema

2.2.1 Descripción del problema

Para la formalización del problema que se aborda en nuestro proyecto, se considera un problema abstracto de una línea de ensamblaje cuya estructura está formada por un número p de líneas paralelas. Cada una de ellas está compuesta por una o varias estaciones dispuestas en serie. Una vez los productos finalizan en las últimas estaciones de cada línea, aparece una nueva estación de unión de ensamblaje final para posteriormente pasar por las últimas estaciones.

La línea de ensamblaje descrita sirve para modelar una línea de ensamblaje final para aviones, donde, por ejemplo, la primera línea corresponde a estaciones encargadas en el ensamblaje del

fuselaje, la segunda línea para el ensamble del timón, etc. La estación de unión se equipararía a la unión de las partes del avión, mientras que las últimas estaciones son utilizadas para instalación de motores, tests, pintura, etc.

A continuación, se muestra gráficamente la disposición de las estaciones en la línea de ensamblaje de nuestro problema:

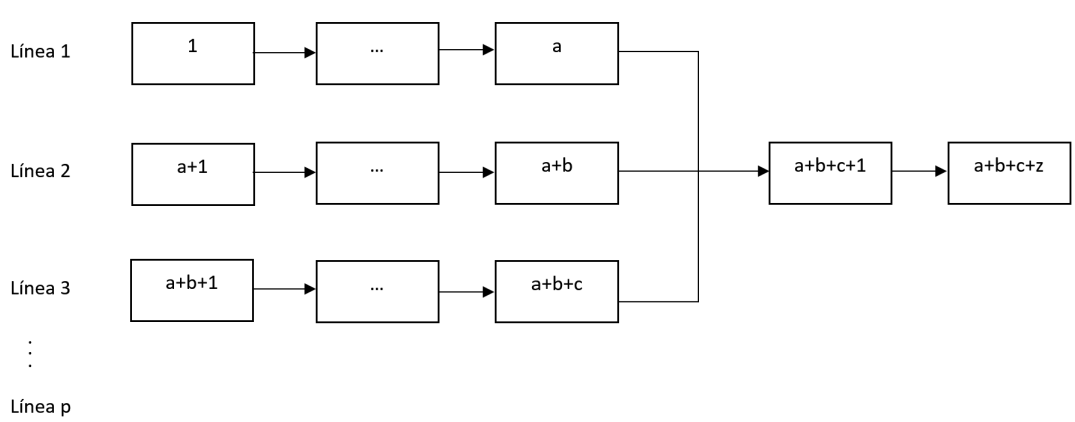

Figura 2.4 Disposición de las estaciones de una línea de ensamblaje [Fuente: Elaboración propia].

Donde:

a : Número de estaciones de la línea 1
b : Número de estaciones de la línea 2
c : Número de estaciones de la línea 3
p : Número de líneas de ensamblaje paralelas
z : Número de estaciones que conforman la última etapa

Cada una de las estaciones está compuesta por una serie de trabajos y un número de operarios asignados, por tanto, uno de los datos de nuestro problema será la matriz de precedencia de cada una de las estaciones junto con el número total de operarios y los requeridos para poder realizar un trabajo en concreto. Una restricción adicional será la imposibilidad de que cualquier operario pueda realizar cualquier tarea dentro de su estación. Por tanto, dividiremos los operarios por tres tipos debido al caso práctico analizado durante el periodo de prácticas:

Operario tipo 1: Operario especializado en los trabajos tipo 1
Operario tipo 2: Operario especializado en los trabajos tipo 2
Operario tipo 3: Operario especializado en los trabajos tipo 3

Para mayor facilidad de modelado, se asumirán los siguientes puntos para nuestro problema:

1. El modelo trabajará bajo una ventana de planificación de tamaño T dividido en periodos de igual longitud correspondiendo un periodo a un día.
2. Un mismo operario no puede tener asignado más de un tipo de especialización.
3. Al tratar con líneas de ensamblaje Low-Volume, los tiempos se trabajarán con variables naturales y no reales.
4. No existe la posibilidad de que un operario pueda trasladarse hacia otra estación.

5. El Lead Time se definirá en periodos, definido como el tiempo que transcurre desde que se inicia el proceso de producción hasta que se completa el último modelo de nuestra línea de ensamblaje.
6. Se asumirá que la precedencia entre actividades y el número de actividades no varía entre modelos.
7. El modelo contempla un número máximo de operarios por estación según especialidad.
8. El tiempo de proceso de una actividad no depende del número de operarios asignados a ella.
9. Se asumirá que los operarios requeridos para una actividad son necesarios durante el tiempo de proceso de la misma.
10. Las actividades no contemplan diferenciación entre su naturaleza (inspecciones, transportes, etc).

Debido a que cada estación puede acoger varios modelos de producto, los datos se diferenciarán para los modelos con los que se vayan a trabajar en la línea de ensamblaje.

2.2.2 Objetivo del problema

El propósito de nuestro problema atiende a dos objetivos:

1. Minimizar el Lead Time de la línea de ensamblaje.
2. Minimizar el número de operarios durante la producción.

Estas dos funciones suponen un gran impacto para asegurar el flujo continuo de varios tipos de modelos de producto en el menor tiempo posible junto con la reducción de costes debido al tiempo que los operarios no están asignados a trabajos en las estaciones. En principio, como los dos objetivos se consideran igual de relevantes, se les asignará el mismo peso para ponderarlos y transformarlo en una función de un único objetivo, aunque más adelante se analizarán otras opciones para ver las respuestas del modelo ante variaciones de parámetros y variables.

A continuación, se muestran los datos para introducir un ejemplo del propósito de nuestro problema, donde el análisis se realizará para un único tipo de modelo de producto y con un total de dos estaciones consecutivas:

Tabla 2.2 Datos que conforman una línea de 2 estaciones [Fuente: Elaboración propia].

Estación	Operarios totales	Trabajo	Precedente	Tiempo de Proceso (horas)	Operarios necesarios
1	4	1	-	2	1
		2	-	1	3
		3	-	4	3
		4	1,2,3	2	2
2	2	5	-	1	1
		6	-	2	1
		7	-	1	1

Posteriormente, hacemos el análisis del Lead Time total de nuestra línea y la proporción del número de operarios ociosos para dos casos diferentes que variarán dependiendo del período de comienzo de cada uno de los trabajos:

Primer caso:

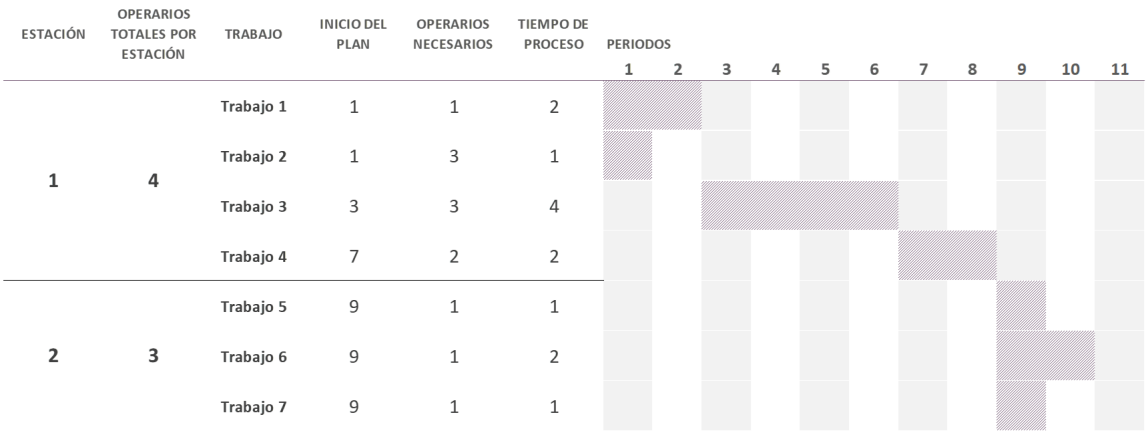

Figura 2.5 Diagrama de Gantt del primer caso [Fuente: Elaboración propia].

Como se puede ver en la línea del tiempo, el Lead Time total es de 10 períodos y la proporción de operarios ociosos es la siguiente:

(3 operarios ociosos durante 1 periodo) + (1 operario ocioso durante 4 periodos) + (2 operarios ociosos durante 2 periodos) + (2 operario ocioso durante 1 periodo)

Obteniendo un resultado de: 13 operarios ociosos $\cdot$ periodo

Segundo caso:

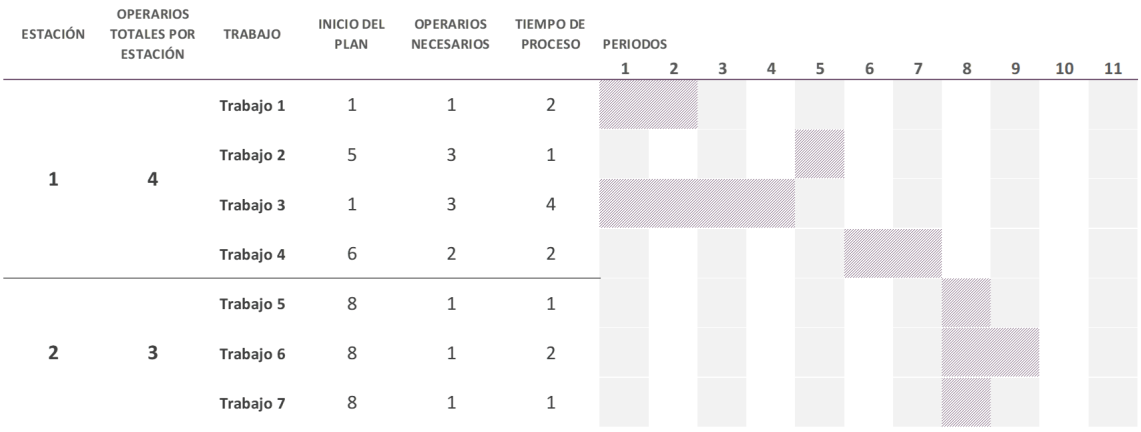

Figura 2.6 Diagrama de Gantt del segundo caso [Fuente: Elaboración propia].

Como se puede ver en la línea del tiempo, el Lead Time total es de 9 períodos y la proporción de operarios ociosos es la siguiente:

(1 operario ocioso durante 2 periodos) + (1 operario ocioso durante 1 periodo) + (2 operarios ociosos durante 2 periodos) + (2 operarios ociosos durante 1 periodo)

Obteniendo un resultado de: 9 operarios ociosos · periodo

Como podemos ver, el segundo caso representa un mejor resultado que el primero debido a que se reduce su Lead Time en 1 periodo y su proporción operarios ociosos · h en 3 unidades.

2.3 Descripción del caso práctico

Una vez formalizado el modelo, procederemos a describir el caso práctico en el que se basa, fruto de la observación durante las prácticas realizadas en una empresa del sector aeronáutico.

A continuación, comenzaremos con la definición de la estructura de la línea de nuestro caso práctico:

La línea de ensamblaje está compuesta por dos líneas paralelas $p = 2$ y una línea final que está compuesta únicamente por la estación de unión de ensamblado final. Por otro lado, las líneas paralelas contienen dos estaciones por cada una dispuestas de manera consecutiva. De este modo, podemos asumir que el número de estaciones de la línea 1 es $a = 2$, al igual que el número de estaciones de la línea 2 es $b = 2$. Por último, el número de estaciones de la línea final z es igual a 1.

En nuestro caso práctico, la línea 1 la conforman las estaciones encargadas en el ensamblado del fuselaje del avión, mientras que la línea 2 se encargará del ensamblado de la cabina. La última estación de la línea une las dos partes del avión. Esta última estación se define normalmente en el sector aeronáutico como la estación Nose-Fuselage Integration, componiendo las actividades de remachado e integración, tests, etc.

A continuación, se puede ver gráficamente en la figura 2.7 la estación 60 de la planta de Airbus San Pablo Sur en Sevilla en la que se ha basado nuestro caso para la integración cabina-fuselaje:

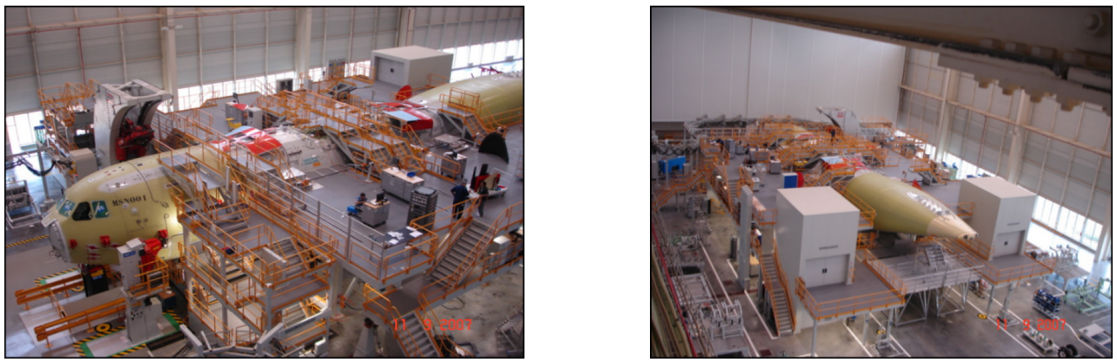

Figura 2.7 Estación 60 de ensamblaje en la planta de Airbus-San Pablo Sur [Fuente: [2]].

Respecto a los trabajos, cada una de las estaciones que componen la línea está compuesta a su vez por una serie de actividades con sus correspondientes órdenes de precedencia. La gráfica a continuación, muestra la estructura de la línea de ensamblaje de nuestro caso práctico, donde se pueden apreciar las actividades que conforman cada una de las estaciones y sus diagramas de precedencia correspondientes:

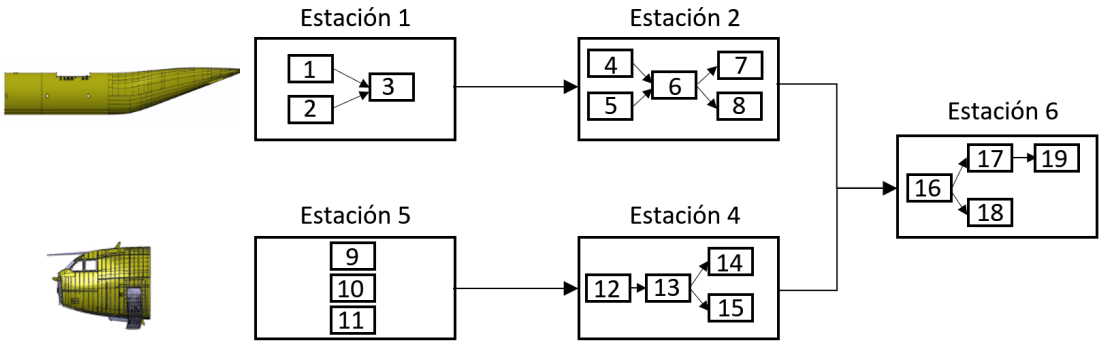

Figura 2.8 Disposición de la estructura de la línea de ensamblaje para el caso práctico [Fuente: Elaboración propia].

La línea de ensamblaje cuenta con un total de 19 trabajos por cada modelo procesado. Se procesarán un total de dos modelos de avión con diferentes tiempos de proceso y operarios requeridos por actividad.

Los datos que conforman cada una de las estaciones para el procesamiento del modelo 1 se muestran en la siguiente tabla, los cuales, han sido inspirados por modelos reales basados en observación (Nota: los datos de precedencia quedan registrados en la figura 2.8 que conforma la estructura de la línea de ensamblaje):

Tabla 2.3 Datos del modelo 1 para cada una de las estaciones [Fuente: Elaboración propia].

Estación	Operarios totales tipo I	Operarios totales tipo II	Operarios totales tipo III	Trabajo	Tiempo de proceso (periodos)	Operarios requeridos tipo I	Operarios requeridos tipo II	Operarios requeridos tipo III
1	2	2	2	1	2	0	2	2
				2	2	2	2	2
				3	1	2	1	2
2	2	2	1	4	1	2	2	1
				5	2	2	2	1
				6	1	1	0	0
				7	1	0	0	1
				8	2	0	1	1
3	1	1	2	9	1	1	1	2
				10	2	1	1	0
				11	1	1	1	1
4	3	2	1	12	1	3	0	0
				13	2	3	0	1
				14	2	3	0	0
				15	1	3	0	1
5	2	3	2	16	2	0	3	2
				17	2	2	0	2
				18	1	1	3	1
				19	1	2	0	0

Mientras que los datos asignados a cada una de las estaciones para el modelo 2 a procesar son los siguientes (Nota: los datos de precedencia quedan registrados en la figura 2.8 que conforma la estructura de la línea de ensamblaje):

Tabla 2.4 Datos del modelo 2 para cada una de las estaciones [Fuente: Elaboración propia].

Estación	Operarios totales tipo I	Operarios totales tipo II	Operarios totales tipo III	Trabajo	Tiempo de proceso (periodos)	Operarios requeridos tipo I	Operarios requeridos tipo II	Operarios requeridos tipo III
1	2	2	2	1	1	1	1	2
				2	2	1	1	0
				3	1	0	1	2
2	2	2	1	4	1	0	2	0
				5	1	0	1	0
				6	1	1	1	1
				7	1	0	0	0
				8	1	0	0	1
3	1	1	2	9	2	0	1	1
				10	2	1	0	1
				11	1	0	0	2
4	3	2	1	12	1	0	1	0
				13	1	1	0	0
				14	1	2	2	1
				15	2	3	2	1
5	2	3	2	16	1	0	1	1
				17	1	1	2	2
				18	1	0	0	2
				19	1	0	2	1

Como podemos comparar en ambas tablas, cabe apreciar que el modelo 2 requiere menos carga de trabajo en las actividades ya que sus tiempos de proceso son menores al igual que los operarios requeridos para su procesamiento. Por tanto, se trata de un modelo menos complejo respecto a fabricación se refiere que el modelo número 1. Para el desarrollo del flujo de valor a lo largo de nuestra línea del tiempo, asumiremos que la línea parte de $t = 0$, es decir, el primer modelo en procesarse será el 1 seguido por el modelo número 2. Además, se supondrá una ventana de planificación de 22 periodos de longitud.

El propósito de nuestro caso práctico tiende a la minimización del Lead Time de la línea de ensamblaje, y por tanto, la minimización de Lt_2, además de la minimización del número total de operarios ociosos en igual proporción, es decir, los pesos que se atribuyan a ambas funciones objetivo W_1 y W_2 serán para ambos del valor de 0.5. De esta manera, podremos hacer un análisis en igual proporción de los resultados obtenidos de nuestro modelo para el caso práctico estudiado.

Puesto a que pueden aparecer problemas computacionales de memoria debido a la cantidad de variables de decisión y restricciones que pueden aparecer en el modelo dependiendo de nuestro número de actividades i, los diferentes modelos j con los que trabaje la línea y la ventana de periodos en los que se desarrolla el flujo de valor T, se ha decidido trabajar con una línea de ensamblaje de dimensión mediana, de esta forma, se podrá evaluar los resultados obtenidos más fácilmente y de forma eficaz.

3 Metodología

Una vez introducido las líneas de ensamblaje Low-Volume Mixtas y ante todo, descrito la problemática y propósitos de optimización para nuestro caso práctico, se procederá a la introducción de la metodología de carácter teórico que se requiere para el desarrollo del caso práctico. Puesto a que se desarrollará un modelo de optimización, el punto 3.1.1 define las bases de la programación lineal. Además, nociones de programación multi-objetivo en el punto 3.1.2 serán necesarias para distintos análisis de la función objetivo. Por último, debido al desarrollo de un modelo matemático que implementaremos y su posterior modelado computacional en lenguaje Python, los puntos 3.2 y 3.3 describen sus principios teóricos.

3.1 Introducción a la programación lineal

3.1.1 Bases de la programación lineal

Según [7] y [29], bajo un modelado matemático, un problema de optimización se suele expresar como se muestra a continuación:

$$\text{Optimizar } f(X) \qquad i = 1...n$$

sujeto a:

$$\begin{aligned} g_j(X) &\geq 0 \qquad j = 1...m \\ h_k(X) &= 0 \qquad k = 1...l \\ q_s(X) &\leq 0 \qquad s = 1...p \end{aligned}$$

$$X \geq 0$$

Donde $X = (x_1,...,x_n)^T$ es un vector n-dimensional en el espacio de números reales R^n. Por tanto, el problema de optimización busca encontrar el vector X que optimice la función $f(x)$ sujeto a las relaciones internas del problema $g_j(X)$,$h_k(X)$ y $q_s(X)$.

Respecto a la formulación matemática representada:

- La función $f(x)$ a optimizar se define como la función objetivo.
- Las relaciones $g_j(X)$,$h_k(X)$ y $q_s(X)$ se denominan restricciones, pudiendo ser de cualquier tipo.
- Las variables x_1,..,x_n se denominan variables de decisión que pueden ser tanto continuas (valores reales $\in R$) como discretas (números enteros $\in N$).

- La terminología optimizar hace referencia a la maximización o minimización de la función objetivo.

En caso de que la función objetivo $f(x)$ y las restricciones $g_j(X)$,$h_k(X)$ y $q_s(X)$ sean lineales en un problema de optimización, podemos llamar al problema como un problema de programación lineal. Por tanto, un problema de programación lineal se puede definir como el problema de maximizar o minimizar una función objetivo lineal sujeta a restricciones lineales, pudiendo ser estas de igualdad o desigualdad.

Terminología de las soluciones en un problema de PL

Para realizar el análisis de las soluciones de un problema de programación lineal, se resolverá un pequeño modelo matemático de dos variables de decisión y por tanto dos dimensiones, para poder trasladarlo así a un procedimiento gráfico.

La figura 3.1 representa la región admisible de nuestro problema, así como la solución óptima, encontrada bajo una traslación de la ecuación función objetiva llevada al vértice óptimo de la región de admisibilidad.

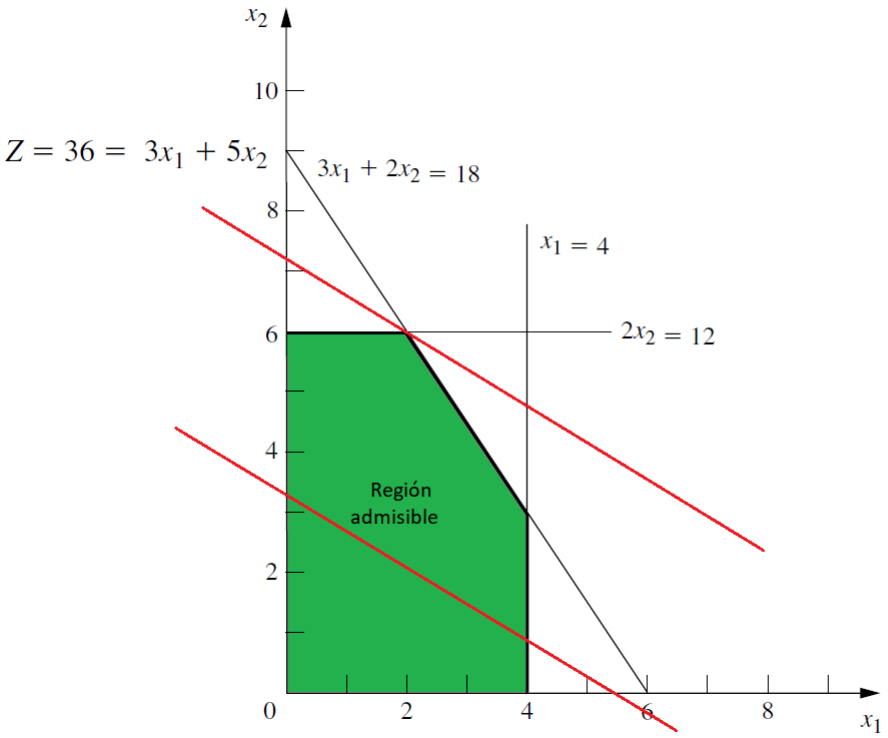

Figura 3.1 Representación gráfica de la región admisible y solución óptima del problema.[Fuente:[24]].

Cualquier valor que pueden optar el vector X que compone las variables de decisión son una solución de nuestro problema de programación lineal, sin importar si es una solución deseable o incluso factible. De acuerdo a los tipos de solución obtenidos, tenemos [11]:

- Solución admisible: Solución la cual todas las restricciones se satisfacen.
- Solución no admisible: Solución la cual al menos una de las restricciones es violada.

Por ejemplo, analizando nuestro problema ejemplo en la figura 3.1, vemos como el punto $x = (6,2)$ no es una solución admisible, mientras que la solución $x = (2,2)$ si lo es por estar dentro de la región de admisibilidad.

Se denomina región de admisibilidad a la región que compone todas las soluciones posibles de nuestro problema [24]. En la figura, el área sombreada se corresponde a la región de admisibilidad de nuestro problema.

Sin embargo, en la programación lineal puede darse el caso de que no exista soluciones admisibles y por tanto no existan soluciones, convirtiéndose así en un problema incompatible. Por ejemplo, si a nuestro problema de optimización añadimos la restricción $3x_1 + 5x_2 \geq 50$, eliminaríamos la región admisible de nuestro problema.

Una vez conformada nuestra región de admisibilidad, un problema de programación lineal busca el resultado óptimo que maximice o minimice su función objetivo. Se denomina función óptima a la solución admisible que tiene el resultado más favorable de la función objetivo [26]. La solución más favorable de un problema es el valor más alto de la función objetivo en caso de maximización y el valor más bajo en caso de minimización.

3.1.2 Programación Multi-Objetivo

Debido a que trabajaremos en nuestro caso práctico dentro del apartado 5.2.2 con conceptos relacionados con la programación multi-objetivo, conviene estudiar sus principales características y métodos.

La optimalidad de Pareto

Un problema de optimización Multi-Objetivo (OMP) es un problema en el cual al menos dos objetivos necesitan ser optimizados de manera simultánea [28]. De esta forma, si las funciones objetivo son $f_1(x), f_2(x) \ldots, f_n(x)$ donde x es un vector de variables de decisión, en términos matemáticos, podemos definir un OMP de la siguiente manera:

$$\min f(x) = [1(x), f_2(x), ..., f_k(x)]^T$$

sujeto a:

$$\begin{aligned} g_j(x) &\leq 0 \qquad j = 1...m \\ h_l(x) &\leq 0 \qquad l = 1...n \end{aligned}$$

$$X \geq 0$$

Donde k denota el número de funciones objetivo del problema, m el número de restricciones de desigualdad y n el número de restricciones de igualdad. $X = (x_1, \ldots, x_n)$ es el vector de variables de decisión.

Generalmente, estas funciones objetivo se encuentra en conflicto, es decir, mejorar un objetivo implica que otro objetivo empeora. Los problemas de optimización Multi-Objetivo con estos problemas de conflicto generan infinidad de soluciones óptimas. La razón por la cual existe más de una optimalidad es porque no se puede considerar que una solución sea mejor que otra respecto a todos los objetivos [30]. Estas soluciones óptimas son conocidas como soluciones Pareto-Óptimas.

Para la optimalidad de Pareto debemos introducir nuevos conceptos de dominancia:

Un vector $u = (u_1,\ldots,u_n)^T \in F$ se dice que domina a otro vector $v = (v_1,\ldots,v_n)^T$ si y solo si $u_i \leq v_i$ para $i \in 1,\ldots,n$ y $\exists i \in 1,..,n : u_i < v_i$. Por tanto, un vector $x^* \in X$ es Pareto-Óptimo si no existe un vector admisible $x \in X$ que mejore alguna función objetivo sin empeorar al menos otra. La figura 3.2 ilustra el ejemplo de la optimalidad de Pareto para un problema con dos funciones objetivo [36].

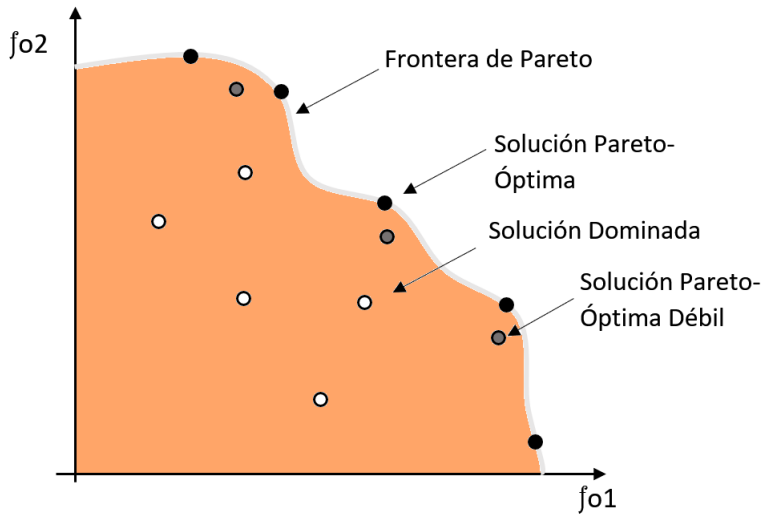

Figura 3.2 Ejemplo ilustrativo de la optimalidad de Pareto [Fuente: Elaboración propia].

El conjunto que forma las diferentes soluciones óptimas de Pareto en el espacio se denomina Frente de Pareto o Frontera de Pareto [32]. En la figura podemos ver como hay soluciones que dominan a otras y otras soluciones que no se sitúan en el frente de Pareto, sino en las proximidades: soluciones Pareto débil.

Por tanto, el paso para la resolución de un problema de optimización Multi-Objetivo consiste en la obtención del Frente de Pareto, el próximo paso, la selección de la solución del Frente de Pareto depende de las preferencias del decisor.

A continuación, procederemos a explicar uno de los métodos generadores más usados debido a su alta funcionalidad para cualquier tipo de problema Multi-Objetivo.

Método de las restricciones

Para el estudio del método de las restricciones, el cual será aplicado en el apartado 5.2.2 para la libre asignación de operarios, haremos uso de un ejemplo para analizar su aplicabilidad a la optimización multi-objetivo mostrado a continuación:

$$fo1\text{: Max } 1000x_1 + 3000x_2$$
$$fo2\text{: Min } x_1 + 2x_2$$

sujeto a:

$$
\begin{aligned}
x_1 &\leq 300 \\
x_2 &\leq 200 \\
x_1 + x_2 &\leq 400 \\
1000x_1 + 3000x_2 &\geq 300000 \\
\\
x_1 \geq 0, x_2 &\geq 0
\end{aligned}
$$

Para poder describir el método de las restricciones, conviene definir antes el concepto de la Matriz de Pagos. Según [32], "la matriz de pagos constituye un útil mecanismo para poder cuantificar el nivel de conflicto existente entre diferentes objetivos". La matriz de pagos es una matriz que devuelve los óptimos por filas separada cada función objetivo por columna. Para el ejemplo anteriormente comentado, tenemos una matriz 3x3 como se muestra:

$$
\text{Matriz de pago} = \begin{bmatrix} 800.000 & 600 \\ 300.000 & 200 \end{bmatrix}
$$

Resultado de la maximización de la función objetivo 1 como único objetivo para la primera fila y, por otra parte, la minimización de la función objetivo 2, devolviendo los resultados que componen la segunda fila de la matriz.

La matriz de pago está compuesta por dos puntos de vital importancia en el análisis multi-objetivo [14]:

- Punto Ideal: Definido por la diagonal de la matriz, para nuestro caso (800.000,200), se trata del punto ideal para el decisor en términos de optimización, ya que devuelve la optimización de cada una de las funciones objetivos de manera no simultánea, y, por tanto, se encuentra en la región de no admisibilidad.
- Punto Anti-Ideal/Nadir: Peor solución que le puedo dar al decisor. Definido por columnas, se compone por la colección de números peores, en nuestro caso: Punto Nadir = (300.000,600).

Para la definición del frente de Pareto, se realiza un proceso iterativo donde las cotas van tomando valores entre el Punto Nadir y el Punto Ideal.

Para nuestro ejemplo, la tabla 3.1 está compuesta por 11 experimentos donde la cota varía entre 300.000-800.000, fijando por tanto la función objetivo 2. Del mismo modo que en la sección anterior, resolveremos cada experimento mediante un modelado en lenguaje Python.

Tabla 3.1 Resultados experimentales del problema mediante el método de las restricciones [Fuente: Elaboración Propia].

Experimento	1	2	3	4	5	6	7	8	9	10	11
fo1	300000	351000	400000	450000	501000	550000	600000	650000	700000	750000	800000
fo2	200	234	267	300	334	367	400	450	500	550	600
x1	0	0	1	0	0	1	0	50	100	150	200
x2	100	117	133	150	167	183	200	200	200	200	200
Cota	300000	350000	400000	450000	500000	550000	600000	650000	700000	750000	800000

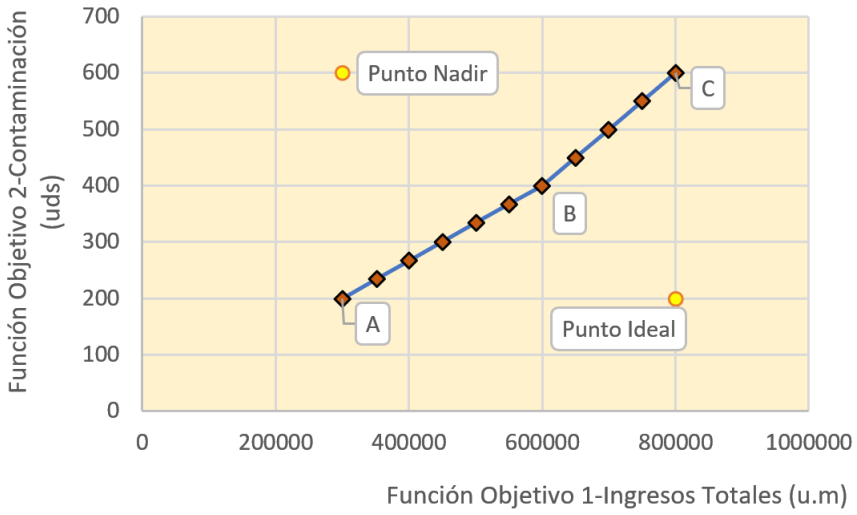

Figura 3.3 Representación gráfica de la frontera de Pareto mediante el método de las restricciones [Fuente: Elaboración propia].

Una vez impuesto la frontera de Pareto en el gráfico 3.3, entra la voz del decisor para la elección de una solución Pareto-Óptima que más se adecue a las necesidades de la empresa. El Anexo A.1 contiene el modelado para la resolución del problema de programación Multi-Objetivo mediante lenguaje Python.

3.2 Bases del modelado matemático

3.2.1 Concepto de modelo

El término "modelo" es comúnmente usado para una estructura que ha sido construida con el propósito de proporcionar funcionalidades y características [33] para comprender aspectos de la vida real.

Dependiendo del nivel de aplicabilidad de un modelo, se puede dividir en concretos o abstractos. Un modelo concreto es únicamente válido para el propósito por el cual se ha creado, no obstante, un modelo abstracto se define mediante simbología algebraica para comprender las relaciones internas entre los objetos de nuestro sistema [35], independientemente de los parámetros del mismo.

Como se muestra en la figura 3.4 los modelos son una medida para la simplificación de procesos complejos de la vida real.

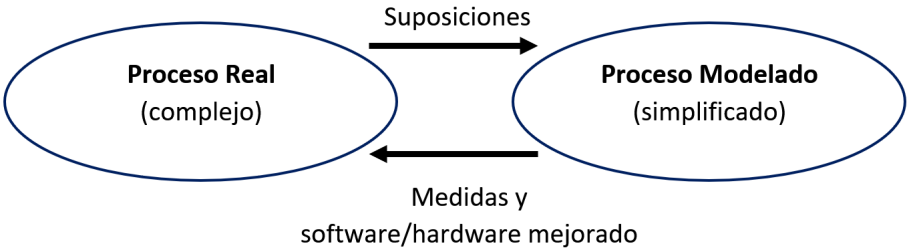

Figura 3.4 Relación entre un proceso real y un proceso modelado [Fuente: [33]].

Estas simplificaciones permiten representaciones estructuradas de conocimientos sobre sistemas reales que facilitan el análisis del modelo resultante. Comúnmente, los modelos son usados para [21]:

- Expicar los fenómenos que comprende el sistema. Realizar predicciones sobre estados futuros del sistema.
- Evaluar factures importantes que influencian el comportamiento del sistema.
- Identificar estados extemos en un sistema que puede representar escenarios de peores casos.
- Analizar las compensaciones para la ayuda de la toma de decisiones.

Los modelos matemáticos representan el conocimiento de un sistema mediante un lenguaje formal matemático el cual está comprendido mediante ecuaciones, inecuaciones y dependencias lógicas.

La característica principal que sustentan los modelos matemáticos es su relación directa con la optimización. Los modelos de optimización son modelos matemáticos que incluyen funciones que representan unos objetivos para el sistema que está siendo modelado. A continuación, se muestra el ejemplo de un modelo matemático de optimización abstracto con parámetros a y b y los parámetros de los vectores a y c.

$$\begin{aligned} & \min \textstyle\sum_{i=1}^{n} c_i x_i \\ \text{sujeto a:} \quad & \\ & \textstyle\sum_{i=1}^{n} a_i x_i \geq b \\ & x_i \geq 0 \qquad \forall i = 1 \ldots n \end{aligned}$$

Los modelos de optimización pueden ser analizados para la exploración de las soluciones admisibles y resultados óptimos de los sistemas. Hoy en día, continúa su desarrollo como una herramienta básica en todas las áreas de aplicación tales como matemáticas, ingeniería, medicina, economía y otras ciencias.

3.2.2 Desarrollo del proceso de modelado

A la hora de desarrollar un modelo matemático, el éxito de su finalización, así como su aplicabilidad a lo que está destinado, está ligado directamente con su proceso para un correcto modelado.
Según [33], el proceso que conforma el modelado de un problema general puede ser dividido en diferentes etapas, las cuales quedan ilustradas en la figura 3.5

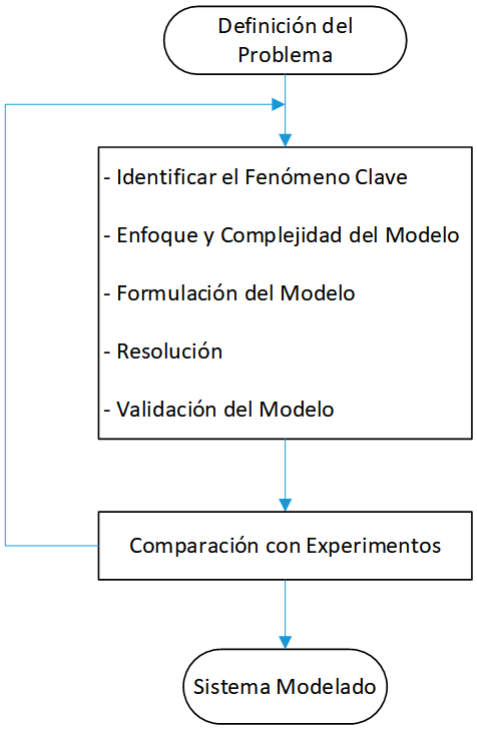

Figura 3.5 Diagrama de flujo del proceso de modelado [Fuente: Elaboración propia].

1. Definición del problema: Definir la razón específica del modelo con claros objetivos y relaciones conllevan al éxito de la formulación.
2. Identificar el fenómeno clave: Entender el fenómeno existente en un sistema es el elemento clave en la formulación de un modelo.
3. Enfoque y complejidad del modelo: En este paso se asigna el tipo de modelo y el nivel de complejidad. El propósito del modelo dicta qué fenómenos del sistema se han incluido, y por consiguiente, que nivel de complejidad abarca.
4. Formulación del modelo: La conversión al lenguaje matemático provoca la aparición de relaciones algebraicas tales como ecuaciones, inecuaciones y dependencias lógicas.
5. Resolución de las ecuaciones: Normalmente mediante Solvers computacionales que almacenan diferentes técnicas de resolución matemática.
6. Validar el modelo: Comparar las predicciones del modelo con los resultados del problema es un paso muy importante para el desarrollo del proceso de modelado.
7. Comparación con experimentos: Por último, las predicciones del modelo se deben comparar con los experimentos reales una vez que se ha demostrado su consistencia numérica en el paso anterior.

3.3 Introducción a la optimización en lenguaje Python

Debido a los múltiples beneficios que proporciona Python tales como facilidad de uso, legibilidad de código, naturalidad de escritura y más importante: el potencial en optimización debido a las múltiples bibliotecas que soporta, optaremos por dicho lenguaje de programación para el modelado de nuestro problema y su resolución mediante Solvers incorporados.

3.3.1 Introducción a Pyomo

Para la herramienta encargada de optimización en Python, haremos uso de Pyomo (Python Optimization Modeling Objetcts). Pyomo soporta la formulación y análisis de modelos matemáticos para aplicaciones de optimización complejas. Esta capacidad está asociada con lenguajes algebraicos de programación (AML), encargados de describir y resolver problemas de alta complejidad para cálculos matemáticos a gran escala [21].

Pyomo incluye clases de Python para definir conjuntos, parámetros y variables, que se pueden usar para formular expresiones algebraicas que definen objetivos y restricciones. Pyomo tiene potencial para representar modelos enteros mixtos lineales, mixtos, no lineales y no lineales para problemas a gran escala del mundo real que implican infinidad de restricciones y variables.

Antes de describir los comandos usuales en Pyomo, describiremos los beneficios principales que hacen de Pyomo uno de los mejores recursos para el modelado en problemas de optimización:

1. Pyomo trabaja con código abierto de licencia flexible, siendo por tanto personalizable y más extensible que algunas alternativas comerciales [20].
2. Capacidad de personalización: Pocos AMLs proporcionan capacidades para personalizar procesos de modelado y optimización. El modelo de código abierto de Pyomo permite un amplio rango para prototipar nuevas capacidades y personalizar el código para aplicaciones específicas [21].
3. Integración de Solvers: Además de los varios Solvers comerciales y de código abierto que se pueden usar mediante diferentes compiladores, Pyomo proporciona Solvers de carácter propio y, por tanto, pueden ser escritos en el propio lenguaje Python.
4. Como detallaremos a continuación, Pyomo ofrece una amplia gama de interfaces para extraer datos de archivos de texto estructurados, hojas de cálculo y bases de datos. Esta capacidad distingue a Pyomo de otras AML de fuente abierta.
5. Scripts de aplicaciones avanzadas: Tal vez la característica más importante de Pyomo es la facilidad con la que se pueden desarrollar las aplicaciones que requieren secuencias de comandos avanzadas [22].

3.3.2 Modelado computacional y comandos usuales

Un modelo en Pyomo consiste en una serie de componentes que definen diferentes aspectos del modelo. Pyomo incluye componentes de modelado que son comúnmente implementados por modernos AMLs: conjuntos indexados, parámetros, variables de decisión, objetivos y restricciones. Estos componentes de modelado se definen en Pyomo a través de las siguientes clases:

Tabla 3.2 Definición las clases más importantes utilizadas en Pyomo [Fuente: Elaboración propia].

Set	Datos de conjuntos usados para definir un modelo
Param	Datos de parámetros usados para definir un modelo
Var	Variables de decisión en un modelo
Objective	Expresiones que son minimizadas o maximizadas en un modelo
Constraint	Expresiones de restricciones en un modelo

A pesar de que Pyomo está diseñado para la definición de modelos abstractos (modelos que permiten la diferenciación entre los datos necesarios y la declaración del modelo), nos centraremos

únicamente en modelos concretos para poder evaluar más fácilmente los diferentes objetos que se empleen para el modelado.

A continuación, se describirá un problema simple de programación lineal para analizar los comandos usuales y más sencillos para su modelado, de esta manera, podremos entender de manera más eficaz el modelo de nuestro problema práctico:

$$\begin{aligned} & \min x_1 + 3x_2 \\ \text{sujeto a:} & \\ & 3x_1 + 4x_2 \geq 1 \\ & 2x_1 + 5x_2 \geq 2 \\ & x_1, x_2 \geq 0 \end{aligned}$$

A la hora de trabajar con modelos concretos de grandes dimensiones como se da en nuestro caso práctico, Pyomo soporta la variable indexado, donde por ejemplo, nuestra variable x puede ser definida con un conjunto indexado N:

```
model.N = Set( initialize = [1,2])
model.x = Var(model.N, within=NonNegativeReals)
```

Como podemos ver, la clase Set soporta la inicialización del conjunto mediante el argumento initialize, quedando así definido el conjunto N=[1,2]. Por otro lado, queda definida la variable de decisión x, indexada bajo los valores del conjunto N y, por tanto, la clase Var recibe dicho conjunto como argumento.

De la misma forma que podemos definir variables indexadas, los parámetros pueden recibir conjuntos como argumentos y poder ser inicializados. Definiremos el parámetro c como los coeficientes de la función objetivo, el parámetro a como los coeficientes que multiplican a las variables de decisión en las restricciones y el parámetro b como las constantes de los términos independientes de las restricciones.

```
model.M = Set( initialize =[1,2])
model.c = Param(model.N, initialize ={1:1, 2:3})
model.a = Param(model.N, model.M, initialize ={(1,1):3, (2,1):4, (1,2):2, (2,2):5})
model.b = Param(model.M, initialize ={1:1, 2:2})
```

De esta forma, por ejemplo, el parámetro a recibe como argumentos tanto el conjunto M como el conjunto N ya que puede tomar valores dependiendo de i y j: a_{ij}, donde $i \in N$ y $j \in M$.

Para la simplificación de modelado de restricciones y funciones objetivo, Pyomo trata con este tema mediante funciones llamadas rules. La idea es que inicializaciones complejas de una colección de restricciones u objetivos puedan ser representadas por una única función que genere cada restricción o función objetivo individualmente [35]. De la misma manera, las rules pueden ser también utilizadas para construir conjuntos o parámetros complejos.

A continuación, se detalla el uso de la función rule para las restricciones de nuestro problema de programación lineal simple:

```
def con_rule(model, j):
    return sum(model.a[i,j]*model.x[i] for i in model.N) >= model.b[j]
model.con = Constraint(model.M, rule=con_rule)
```

El conjunto de índices M define los índices la restricción con, y la función con_rule se usa para la construcción de las expresiones de restricciones individuales. Los argumentos que recibe la función rule además del modelo en sí, son los índices para las restricciones, en este caso $j \in M$. Se ha definido el nombre no restrictivo de la función rule como con, el argumento return devuelve las expresiones de la función con_rule y tanto los parámetros como las variables de decisión son indexadas bajo índices pertenecientes a los diferentes conjuntos.

Cabe destacar el uso de la sintaxis para la iteración suma a lo largo de un conjunto de variables de decisión. La función sum en Python, haciendo uso del bucle for, puede ser usada para el sumatorio a lo largo de los índices pertenecientes al conjunto indexado N.

De la misma forma que ocurre con las restricciones, la función rule hace la misma función para las funciones objetivos. Aplicándolo a nuestro ejemplo, el único argumento que recibe la función es el propio modelo.

```
def obj_rule(model):
    return sum(model.c[i]*model.x[i] for i in model.N)
model.obj = Objective(rule=obj_rule)
```

La función obj_rule queda definida a diferencia de la función con_rule bajo la clase Objective, definiendo así la expresión como la función objetivo del modelo. El anexo A.2 incluye la formulación completa para este modelo concreto.

4 Modelado y aplicación al caso práctico

Comprendidos los conceptos que rodean nuestra problemática, este capítulo aborda la aplicación de la metodología al caso práctico. En primera instancia, se propone y desarrolla un modelo matemático de optimización en el punto 4.1 que atiende a las demandas del propósito de nuestro problema para posteriormente traspasar la formulación matemática a un modelado computacional en lenguaje Python, aplicado a los distintos parámetros datos de nuestro caso práctico.

4.1 Modelo matemático

4.1.1 Formulación matemática

Se supondrá una ventana de planificación de longitud T dividido en periodos de igual dimensión: $t = 1, ..., T$. Se harán uso de los siguientes índices y conjuntos en nuestro problema:

$$\begin{gathered} i = 1,....,N \text{ : trabajos} \\ j = 1,....,J \text{ : modelos} \\ t = 1,....,T \text{ : periodos} \\ k = 1,....,M \text{ : estaciones} \end{gathered}$$

P_i = conjunto de predecesores inmediatos a la tarea i
W_k = conjunto de actividades pertenecientes a la estación k
M_i = conjunto de actividades dentro de la estación perteneciente a i
S_i = conjunto de actividades que pertenecen a la estación predecesora respecto a la estación de la actividad i

Las siguientes variables de decisión serán aplicadas en el modelo:

$$x_{itj} = \begin{cases} 1 & \text{si el trabajo i empieza en el periodo t para el modelo j} \\ 0 & \text{en otro caso} \end{cases}$$

Lt_j = tiempo de Lead Time para el modelo j

El modelo está basado en los siguientes parámetros:

t_{ij} = tiempo de proceso del trabajo i para el modelo j

$hreq1_{ij}$ = nº de operarios de especialidad 1 requeridos para el trabajo i del modelo j
$hreq2_{ij}$ = nº de operarios de especialidad 2 requeridos para el trabajo i del modelo j
$hreq3_{ij}$ = nº de operarios de especialidad 3 requeridos para el trabajo i del modelo j

$htotal1_k$ = nº de operarios totales de especialidad 1 en la estación k
$htotal2_k$ = nº de operarios totales de especialidad 2 en la estación k
$htotal3_k$ = nº de operarios totales de especialidad 3 en la estación k

El resultante modelo matemático de programación lineal entera queda formulado a continuación:

$$\min w_1 T_1 + w_2 T_2$$

donde:

$T_1 = \sum_{t=1}^{T}(\sum_{k=1}^{M}((htotal1_k + htotal2_k + htotal3_k) - \sum_{n\in W_k}\sum_{j=1}^{J}(hreq1_{nj}(\sum_{k=1}^{t} x_{nkj} - \sum_{k=1}^{t-t_{nj}} x_{nkj}) + hreq2_{nj}(\sum_{k=1}^{t} x_{nkj} - \sum_{k=1}^{t-t_{nj}} x_{nkj}) + hreq3_{nj}(\sum_{k=1}^{t} x_{nkj} - \sum_{k=1}^{t-t_{nj}} x_{nkj})))$

$T_2 = Lt_J$

sujeto a:

$$\sum_{t=1}^{T} x_{itj} = 1, \qquad i = 1,...,N, j = 1,...,J \tag{1}$$

$$\sum_{t=1}^{T} tx_{itj} \geq (\sum_{t=1}^{T} tx_{ptj}) + t_{pj}, \qquad i = 1,...,N, p \in P_i \; j = 1,...,J \tag{2}$$

$$\sum_{n\in W_k} hreq1_{nj}(\sum_{k=1}^{t} x_{nkj} - \sum_{k=1}^{t-t_{nj}} x_{nkj}) \leq htotal1_k \qquad k = 1,...,M \; t = 1,...,T, j = 1,...,J \tag{3}$$

$$\sum_{n\in W_K} hreq2_{nj}(\sum_{k=1}^{t} x_{nkj} - \sum_{k=1}^{t-t_{nj}} x_{nkj}) \leq htotal2_k \qquad k = 1,...,M \; t = 1,...,T, j = 1,...,J \tag{4}$$

$$\sum_{n\in W_K} hreq3_{nj}(\sum_{k=1}^{t} x_{nkj} - \sum_{k=1}^{t-t_{nj}} x_{nkj}) \leq htotal3_k \qquad k = 1,...,M \; t = 1,...,T, j = 1,...,J \tag{5}$$

$$Lt_j \geq x_{itj}(t + t_{ij} - 1) \qquad i = 1,...,N, t = 1,...,T, j = 1,...,J \tag{6}$$

$$\sum_{t=1}^{T} tx_{itj} \geq (\sum_{t=1}^{T} tx_{mtj-1}) + t_{mj-1}, \qquad i = 1,...,N, m \in M_i \; j = 2,...,J \tag{7}$$

$$\sum_{t=1}^{T} tx_{itj} \geq (\sum_{t=1}^{T} tx_{stj}) + t_{sj} \qquad i = 1,...,N, s \in S_i \; j = 1,...,J \tag{8}$$

$$x_{itj} \in [0,1], \qquad Lt_j \geq 0 \qquad Lt_j \in \mathbb{Z} \tag{9}$$

4.1.2 Análisis del modelo matemático

Función objetivo

La minimización de la función objetivo $w_1 T_1 + w_2 T_2$ consiste en la suma de los operarios ociosos totales y el Lead Time de la línea de ensamblaje. w_1 y w_2 corresponden a los pesos propios de la función objetivo de forma general. Nuestro modelo trabaja con un problema multi-objetivo donde nuestras funciones son lineales. Sin embargo, T_1 y T_2 son términos correlacionados y, por tanto, satisfacen mutuamente respecto a su minimización, esto es, T_2 busca la minimización del Lead Time

de la línea de ensamblaje, mientras que T_1 minimiza el número de operarios ociosos de la línea. Sin embargo, cuanto más se minimiza T_2, obtenemos una cadena de flujo más compacta donde se realizan más actividades de forma simultánea, disminuyendo así el número de operarios ociosos. Concluimos así que las funciones objetivo no son conflictivas, asegurando así que el término T_2 está relacionado directamente con T_1 en nuestra función objetivo.

A continuación, se detallará de forma más desglosada cada uno de los términos que componen los sumandos T_1 y T_2:

T_1: Minimización del número de operarios ociosos

Para definir el número de operarios ociosos en un periodo conviene trabajar sobre un ejemplo para poder ir identificando las fórmulas matemáticas correspondientes de forma más sencilla y por tanto, detallar nuestro primer término T_1 de la función objetivo.

Supongamos una línea de ensamblaje compuesta por dos estaciones consecutivas en las cuales dentro de la primera estación se está procesando el modelo 2, mientras que en la segunda estación se procesa el modelo 1. Cada estación está comprendida por una serie de trabajos con sus respectivas precedencias. Para mayor facilidad de comprensión, vamos a suponer el mismo tiempo de proceso para un trabajo independientemente del modelo al igual que su número de operarios requeridos para su procesamiento, esto es, el número de operarios requeridos del tipo n para el trabajo 1 será el mismo independientemente del modelo que se esté procesando.

La figura 4.1 muestra la disposición de las estaciones y los trabajos correspondientes. Como podemos ver, la primera estación dispone de trabajos con precedencias mientras que en la segunda estación los trabajos se pueden realizar de forma paralela.

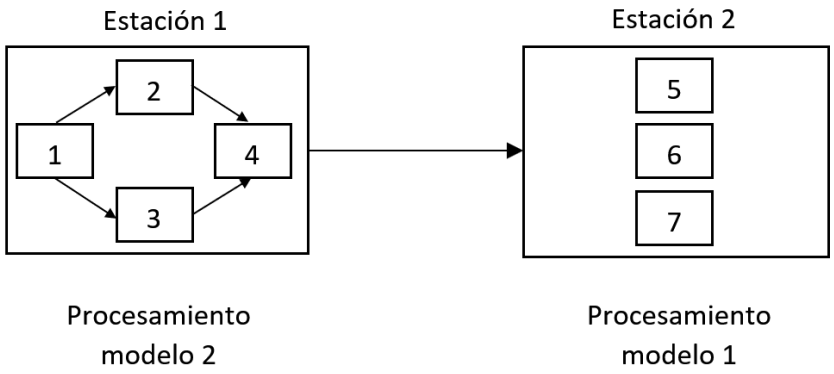

Figura 4.1 Disposición de las etaciones y los órdenes de precedencia entre trabajos del problema biestacional [Fuente: Elaboración propia].

El marco de estudio en este ejemplo se realizará bajo una ventana de 5 periodos de longitud. Por otra parte, se asumirá que ambos modelos iniciarán su procesado en su estación correspondiente en el periodo número 1, por tanto, el procesamiento de dicha línea finalizará con la última tarea procesada del modelo 1 en la estación 2.

La tabla 4.1 representada a continuación muestra los datos del ejemplo propuesto. Cabe destacar que los tiempos de proceso y los operarios requeridos para cada actividad dependen del modelo concreto, aunque en este problema los consideremos del mismo valor:

Tabla 4.1 Datos para la línea de ensamblaje del problema biestacional para el análisis matemático [Fuente: Elaboración propia].

Estación	Operarios totales tipo I	Operarios totales tipo II	Operarios totales tipo III	Trabajo	Precedente	Tiempo de Proceso (horas)		Operarios necesarios tipo I		Operarios necesarios tipo II		Operarios necesarios tipo III	
1	1	2	1	1	-	-	2	-	1	-	1	-	0
				2	1	-	1	-	0	-	2	-	1
				3	1	-	2	-	1	-	0	-	0
				4	2,3	-	1	-	0	-	2	-	0
2	0	1	1	5	-	2	-	0	-	0	-	1	-
				6	-	2	-	0	-	1	-	0	-
				7	-	3	-	0	-	0	-	1	-

Posteriormente, hacemos el análisis del flujo de la línea del tiempo para poder estudiar la disposición de los operarios en cada una de las actividades y por tanto, el número de operarios ociosos en cada periodo con el fin de detallar la fórmula matemática que minimice su valor:

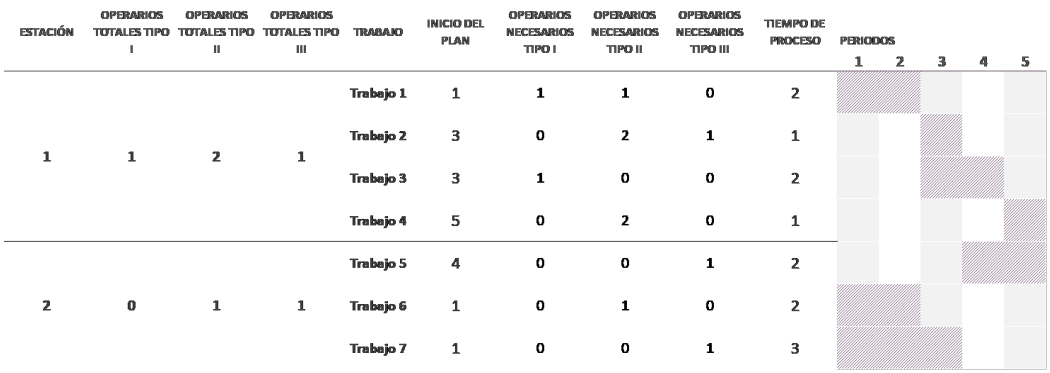

Figura 4.2 Diagrama de Gantt del problema biestacional para el análisis matemático [Fuente: Elaboración propia].

Como podemos ver en la línea del tiempo de la figura 4.2 , el Lead Time de nuestro estudio es de 5 periodos. A continuación, pasaremos a hacer el análisis de los operarios asignados en cada periodo:

Es fácil determinar en nuestra línea del tiempo el número de operarios de cada tipo establecidos en cada uno de los trabajos para las distintas estaciones, sin embargo, puesto que no podemos denotar una nueva variable $h1_{itj}$ debido a que siempre tomará un valor que depende únicamente de x_{itj} demostramos la denotación matemática del número de operarios totales asignados del tipo X en cada periodo:

$$\sum_{n \in W_k} \sum_{j=1}^{J} (hreqX_{ij} * (\sum_{k=1}^{t} x_{ikj} - \sum_{k=1}^{t-t_{ij}} x_{ikj})$$

Por ejemplo, para el periodo 1, analizando la línea del tiempo anterior concluimos que existe un total de 1 operario de tipo I, dos operarios de tipo 2 y 1 operario de tipo 3. Analizando la denotación matemática para el operario tipo 1 en el primer periodo, se asume que para que existan operarios tipo 1 en el periodo, debe darse el caso de que se esté realizando una actividad en ese mismo periodo

y que además requiera de operarios tipo 1 para su procesamiento.

Por tanto, la formulación $\sum_{k=1}^{t} x_{ikj} - \sum_{k=1}^{t-t_{ij}} x_{ikj}$ devuelve 1 en caso de que se esté procesando la actividad i del modelo j en el periodo t y 0 en caso contrario. Haciendo una simple multiplicación por $hreq1_{ij}$ obtenemos el número de operarios asignados en el periodo t de la actividad i para el modelo j. Por último, el total de operarios asignados en el periodo t será la suma para todas las actividades que se estén procesando en ese periodo:

$\sum_{n \in W_k} \sum_{j=1}^{J} (hreq1_{ij} * (\sum_{k=1}^{t} x_{ikj} - \sum_{k=1}^{t-t_{ij}} x_{ikj}) + hreq2_{ij} * (\sum_{k=1}^{t} x_{ikj} - \sum_{k=1}^{t-t_{ij}} x_{ikj}) + hreq3_{ij} * (\sum_{k=1}^{t} x_{ikj} - \sum_{k=1}^{t-t_{ij}} x_{ikj})))$

Una vez analizado los operarios por periodo, para el cálculo del número de operarios ociosos por periodo se realiza simplemente con la resta del total de operarios disponibles en cada estación menos el número de operarios asignados en ese periodo. A sí pues, el término T_1 devuelve el número total de operarios ociosos a lo largo de nuestra línea del tiempo para la línea de ensamblaje estudiada.

Analizando nuestra línea del tiempo del ejemplo anterior es fácil determinar que aparecen en el periodo 1 un total de 2 operarios ociosos (1 operario tipo II + 1 operario tipo III).

T_2: Minimización del Lead Time

Puesto que nuestro propósito del problema, además de la minimización de operarios ociosos en la línea de ensamblado, se basa en la minimización del Lead Time, se asignará el vector Lt_j a la variable para minimizar Lt_J ya que supone el último modelo a procesar de la línea.

Restricciones

La restricción (1) obliga a que cada actividad solo puede empezar en un solo periodo para un modelo concreto.

La restricción (2) asigna el orden de precedencia entre actividades. Para denotar que una actividad no puede comenzar hasta que hayan finalizado sus predecesoras, se hace uso de un conjunto de índices $i \in I$ para definir una familia de conjuntos P_i. El conjunto I se denomina conjunto de índices y cada $i \in I$ es un índice para la familia de conjuntos de P_i. Una familia indexada de conjuntos de conjunto índice I es una función con dominio I (donde I es no vacío) y cuyas imágenes y por tanto, el rango de valores que puede llegar a tener dicho conjunto son a su vez, conjuntos [18].

A modo de ejemplo para la familia de conjuntos, denotamos los vectores $v_1, \ldots, v_2$. La notación $(v_i)_{i \in 1,\ldots,n}$ asigna una familia de vectores. El enésimo vector v_i representa un conjunto de valores que dependen del conjunto de índices i.

Por tanto, volviendo a nuestro ejemplo anteriormente definido, si asumimos el conjunto $i = \{1,2,3,4,5,6,7\}$ como el conjunto de índices $i \in I$, para denotar la restricción de precedencia mediante una familia indexada de conjuntos, se establece lo siguiente:

Cuando $I = \{1,2,3,4,5,6,7\}$:

- Para $i = 1$, $P_1 = \{\emptyset\}$
- Para $i = 2$, $P_2 = \{1\}$
- Para $i = 3$, $P_3 = \{1\}$

- Para $i = 4$, $P_4 = \{2,3\}$
- Para $i = 5$, $P_5 = \{\emptyset\}$
- Para $i = 6$, $P_6 = \{\emptyset\}$
- Para $i = 7$, $P_7 = \{\emptyset\}$

A continuación, se muestra el ejemplo estudiado de forma gráfica mediante la figura 4.3

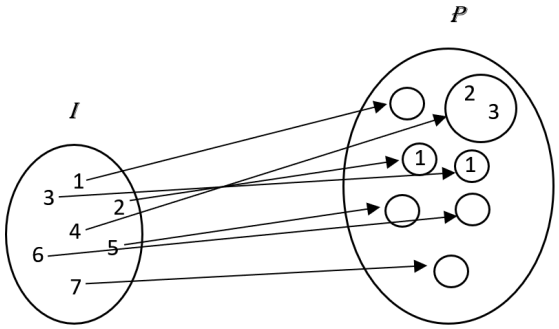

Figura 4.3 Representación gráfica de la familia de conjuntos indexados de P_i [Fuente: Elaboración propia].

De esta manera, atendiendo a la familia indexada de conjuntos, tendríamos las siguientes restricciones para nuestro ejemplo:

- $\sum_{t=1}^{T} tx_{2t2} \geq (\sum_{t=1}^{T} tx_{1t2}) + t_{12}$
- $\sum_{t=1}^{T} tx_{3t2} \geq (\sum_{t=1}^{T} tx_{1t2}) + t_{12}$
- $\sum_{t=1}^{T} tx_{4t2} \geq (\sum_{t=1}^{T} tx_{2t2}) + t_{22}$
- $\sum_{t=1}^{T} tx_{4t2} \geq (\sum_{t=1}^{T} tx_{3t2}) + t_{32}$

La restricción (3) refleja que el número de operarios asignados de tipo I dentro de una estación debe ser inferior al número de operarios disponibles de tipo I en esa misma estación. Como ya hemos comentado anteriormente en la función objetivo, la formulación $\sum_{k=1}^{t} x_{nkj} - \sum_{k=1}^{t-t_{nj}} x_{nkj}$ devuelve 1 en caso de que se esté procesando la actividad i del modelo j en el periodo t y 0 en caso contrario, y por tanto, $hreqX_{ij}(\sum_{k=1}^{t} x_{ikj} - \sum_{k=1}^{t-t_{ij}} x_{ikj})$ devuelve el número de operarios asignados de la especialidad X para la actividad i del modelo j en el periodo t.

De la misma forma que en la restricción anterior, haremos uso de conjuntos de índices, donde W_k representa la familia indexada de conjuntos respecto a la familia de índices k, siendo $k \in K$ el conjunto de estaciones. Por tanto, volviendo a nuestro ejemplo definido, si asumimos el conjunto $k = \{1,2\}$ como el conjunto de índices $k \in K$, para denotar la restricción del número máximo de operarios tipo I por estación mediante una familia indexada de conjuntos, se establece lo siguiente:

Cuando $K = \{1,2\}$:

- Para $k = 1$, $W_1 = \{1,2,3,4\}$
- Para $k = 2$, $W_2 = \{5,6,7\}$

De esta manera, atendiendo a la familia indexada de conjuntos, tendríamos las siguientes restricciones para nuestro ejemplo:

- $\sum_{n\in\{1,2,3,4\}} hreq1_{n2}(\sum_{k=1}^{t} x_{nk2} - \sum_{k=1}^{t-t_{n2}} x_{nk2}) \leq htotal1_1 \qquad t = 1,...,T$
- $\sum_{n\in\{5,6,7\}} hreq1_{n1}(\sum_{k=1}^{t} x_{nk1} - \sum_{k=1}^{t-t_{n1}} x_{nk1}) \leq htotal1_2 \qquad t = 1,...,T$

Las restricciones (4) y (5) representan de la misma manera el número máximo de operarios asignados tipo II y III por estación.

La restricción (6) refleja la definición del Lead Time para cada modelo, esto es, Lt_j deberá ser mayor que el mayor periodo de finalización de las actividades que componen el modelo j. Cabe destacar la resta de 1 periodo entre la suma de t y t_{ij} debido a que una actividad de un modelo dado puede acabar y empezar en el mismo periodo.

De la misma manera que se ha denotado matemáticamente la restricción (2), la precedencia entre modelos se refleja en la restricción (7). Por tanto, una actividad i del modelo j en una estación dada no puede comenzar hasta que no se hayan realizado todas las actividades que conforman esa misma estación para el modelo $j-1$. En esta restricción se hará uso de la familia indexada de conjuntos M_i que contiene los conjuntos de actividades dentro de la estación perteneciente a la actividad i, así pues, $i \in I$ será de nuevo nuestro conjunto de índices.

Por tanto, respecto al ejemplo anterior, si asumimos el conjunto $i = \{1,2,3,4,5,6,7\}$ como el conjunto de índices $i \in I$, la denotación de precedencia entre modelos queda definida mediante una familia indexada de conjuntos:

Cuando $I = \{1,2,3,4,5,6,7\}$:

- Para $i = 1$, $M_1 = \{1,2,3,4\}$
- Para $i = 2$, $M_2 = \{1,2,3,4\}$
- Para $i = 3$, $M_3 = \{1,2,3,4\}$
- Para $i = 4$, $M_4 = \{1,2,3,4\}$
- Para $i = 5$, $M_5 = \{5,6,7\}$
- Para $i = 6$, $M_6 = \{5,6,7\}$
- Para $i = 7$, $M_7 = \{5,6,7\}$

La restricción (8) refleja la precedencia entre estaciones haciendo uso de la familia indexada de conjuntos S_i , de nuevo, siendo $i \in I$ nuestro conjunto de índices. La notación matemática es similar a las representadas en las restricciones (2) y (7) con la diferencia en el parámetro S_i.

- Para $i = 1$, $S_1 = \{\emptyset\}$
- Para $i = 2$, $S_2 = \{\emptyset\}$
- Para $i = 3$, $S_3 = \{\emptyset\}$
- Para $i = 4$, $S_4 = \{\emptyset\}$
- Para $i = 5$, $S_5 = \{1,2,3,4\}$
- Para $i = 6$, $S_6 = \{1,2,3,4\}$
- Para $i = 7$, $S_7 = \{1,2,3,4\}$

Por último, la restricción (10) asegura que las variables de decisión $x_{it\,j}$ sean binarias mientras que Lt_j variables enteras.

4.2 Modelado computacional

Una vez formulado el modelo matemático, procederemos con el modelado de nuestro caso práctico concreto de forma computacional. Para mayor facilidad de seguimiento, no se seguirá el mismo orden descrito en el modelo de la sección 4.1 a la hora de su explicación en esta sección.

Comenzaremos por el modelado de los diferentes conjuntos e índices terminando por las últimas restricciones de nuestro modelo matemático.

4.2.1 Bibliotecas externas implementadas

Una biblioteca es un conjunto de implementaciones funcionales codificadas en un lenguaje de programación que ofrece una interfaz bien definida para la funcionalidad que se invoca [3].

Para nuestra codificación, además de las bibliotecas propias de Pyomo para utilizar en lenguaje Python, haremos uso de la biblioteca Openpyxl. Openpyxl es una biblioteca de Python capaz de leer y escribir archivos xlsx/xlsm/xltx/xltm de Excel 2010 [1]. Debido a la gran cantidad de datos que nuestro modelo maneja, openpyxl resultará ser una herramienta muy útil para el procesamiento de hojas Excel que contienen multitud de datos, los cuales debemos implementar en nuestra codificación.

A lo largo de la sección, iremos detallando el uso de la biblioteca y las estructuras de las hojas excel utilizadas como documentos de volcado de datos. A continuación, quedan distinguidas las bibliotecas necesarias para el modelado de nuestro caso práctico en Python:

```
import pyomo.environ
from pyomo.core import *
from pyomo.opt import SolverFactory
import openpyxl
from openpyxl import load_workbook
```

Donde las tres primeras líneas de código corresponden a las bibliotecas propias de Pyomo en Python, importando además la factoría Solver como optimizador del problema y las dos últimas llaman a la biblioteca Openpyxl, donde únicamente se requiere la lectura de datos, obviando así la escritura.

4.2.2 Modelado de índices y conjuntos

Para el modelado de los distintos conjuntos e índices de nuestro problema de optimización, haremos uso de codificación adicional en Python para formular más fácilmente nuestras restricciones que abordan a nuestro modelo matemático.
Gracias a la facilidad de Python para trabajar algebraicamente, haremos uso de las siguientes matrices compuestas por conjuntos:

M_estructura = Matriz estructura línea de ensamblaje
M_estaciones = Matriz de estaciones
M_precedencia = Matriz de precedencia

A continuación, procederemos a detallar cada una de ellas describiendo su modelado para nuestro caso práctico:

- Matriz estructura línea de ensamblaje (M_estructura).

Matriz la cual está compuesta por tres dimensiones, la primera línea designa la línea de ensamblaje a la que estamos haciendo referencia, siendo la última siempre la línea de la última etapa y las restantes las líneas paralelas. La segunda dimensión hace referencia a la estación de la línea correspondiente, y por último, la tercera dimensión muestra los trabajos pertenecientes a dicha estación.
Para la codificación en Python de la matriz M_estructura, se han recogido los datos de la hoja excel "Hoja2" del archivo "DatosExcel2" mediante diferentes funciones pertenecientes a la biblioteca externa Openpyxl. A continuación, la tabla 4.2 muestra la tabla de datos de la estructura de la línea de ensamblaje de nuestro caso práctico.

Tabla 4.2 Tabla Excel que muestra los datos de la estructura de la línea [Fuente: Elaboración propia].

	A	B	C
1	**Línea**	**Estación**	**Trabajo**
2	1	1	1
3	1	1	2
4	1	1	3
5	1	2	4
6	1	2	5
7	1	2	6
8	1	2	7
9	1	2	8
10	2	1	9
11	2	1	10
12	2	1	11
13	2	2	12
14	2	2	13
15	2	2	14
16	2	2	15
17	3	1	16
18	3	1	17
19	3	1	18
20	3	1	19

Es importante destacar que la columna B llamada "Estación" no muestra el número de la estación de forma independiente, sino que asigna la enésima estación perteneciente a la línea en concreto. Por ejemplo, el trabajo 8 pertenece a la segunda estación de la línea 1, mientras que el trabajo 18 pertenece a la primera estación de la tercera línea, coincidiendo la tercera con la línea final de ensamblado.

La codificación en Python para la definición de la matriz estructura línea de ensamblaje quedaría de la siguiente forma:

```
''' Inicializamos la Matriz estructura y definimos las variables
 numero_filas y linea_final de la hoja "Hoja2"'''

M_estructura =[]
numero_filas = hoja2.max_row
 linea_final = hoja2. cell (row=numero_filas,column=1).value
```

```
''' Inicializamos la primera dimension '''
for s in range(0, linea_final ):
    M_estructura.append([])

''' Definimos el numero de estaciones por cada linea '''
for fila in range(2, numero_filas ):
    if hoja2.cell(row=fila,column=1).value == 1:
        estaciones_linea1 = hoja2.cell(row=fila,column=2).value
    if hoja2.cell(row=fila,column=1).value == 2:
        estaciones_linea2 = hoja2.cell(row=fila,column=2).value
    if hoja2.cell(row=fila,column=1).value == 3:
        estaciones_linea3 = hoja2.cell(row=fila,column=2).value
    if hoja2.cell(row=fila,column=1).value == 4:
        estaciones_linea4 = hoja2.cell(row=fila,column=2).value

''' Inicializamos la segunda dimension '''
for s in range(0, linea_final ):
    if s==0:
        for m in range(0, estaciones_linea1 ):
            M_estructura[s].append([])
    elif s==1:
        for m in range(0, estaciones_linea2 ):
            M_estructura[s].append([])
    elif s==2:
        for m in range(0, estaciones_linea3 ):
            M_estructura[s].append([])
    elif s==3:
        for m in range(0, estaciones_linea4 ):
            M_estructura[s].append([])

''' Asignamos los trabajos correspondientes a las distintas estaciones
de la linea de ensamblaje '''
for fila in range(2, numero_filas+1):
    M_estructura[hoja2.cell(row=fila,column=1).value -1][hoja2.cell \
    (row=fila,column=2).value - 1].append(hoja2.cell(row=fila,\
    column=3).value)
```

Además de los distintos bucles utilizados para recorrer cada una de las filas de nuestra tabla de datos y los condicionales para determinar tanto el número de estaciones por línea como la inicialización de la dimensión encargada en recorrer las estaciones de la línea de ensamblaje, hacemos uso una función nueva llamada hoja2.cell(row=x,column=y).value. Se trata de una función que llama a la biblioteca Openpyxl para devolver el valor de la celda perteneciente a la fila x y columna y dentro de la hoja 'Hoja2'. Los distintos bloques que contiene el código mostrado anteriormente quedan señalados con comentarios para mayor facilidad de entendimiento.

Imprimiendo por pantalla la matriz estructura línea de ensamblaje para nuestro caso práctico mediante la función print(M_estructura), obtenemos:

$$\text{M\_estructura} = \left[\left[[1,2,3] [4,5,6,7,8] \right] \left[[9,10,11] [12,13,14,15] \right] \left[[16,17,18,19] \right] \right]$$

Por tanto, si hacemos referencia a la posición M_estructura[x][y][z], el valor x tomará valores que devolverá la línea a la que estamos haciendo referencia, el valor y la estación perteneciente a la línea $x+1$ (puesto que Python trabaja con posiciones en lista comenzando por el valor 0) y el valor z el trabajo perteneciente dentro de dicha estación.

- Matriz de estaciones (M_estaciones).

  Matriz compuesta por dos dimensiones, siendo la primera la dimensión que designa la estación de la línea de ensamblaje y la segunda agrupa todos y cada uno de los trabajos correspondientes a la estación dada.

  Para la codificación en Python de la matriz M_estaciones, se han obtenido los datos de la hoja "Hoja1" perteneciente al documento "DatosExcel2" mediante diferentes funciones pertenecientes de la biblioteca Openpyxl comentadas anteriormente. A continuación se muestra un estracto de la tabla perteneciente a la hoja "Hoja1", donde nos interesan trabajar sobre la columna A y la columna E para el desarrollo del código:

Tabla 4.3 Tabla Extracto Excel para la codificación de la matriz estaciones [Fuente: Elaboración propia].

	A	B	C	D	E
1	Estación	Operarios totales tipo I	Operarios totales tipo II	Operarios totales tipo III	Trabajo
2	1	2	2	2	1
3	1				2
4	1				3
5	2	2	2	1	4
6	2				5
7	2				6
8	2				7
9	2				8
10	3	1	1	2	9
11	3				10
12	3				11
13	4	3	2	1	12
14	4				13
15	4				14
16	4				15
17	5	2	3	2	16
18	5				17
19	5				18
20	5				19

Posteriormente de inicializar la matriz mediante el comando append, utilizaremos una variable fila que recorrerá mediante un bucle todas las filas de nuestra tabla de datos para ir asignando de esta forma, los trabajos que pertenezcan a la columna 5 a las estaciones correspondientes a la fila número 1. Los distintos bloques que contiene el código se muestran

a continuación con comentarios para mayor facilidad de comprensión.

```
''' Inicializamos la matriz y definimos las variables fila_ultima y
numero_estaciones de la hoja "Hoja1"'''
M_estaciones = []
 fila_ultima  = hoja.max_row
numero_estaciones = hoja. cell (row= fila_ultima ,column=1).value

''' Inicializamos la primera dimension'''
for s in range (0, numero_estaciones ):
    M_estaciones.append ([])

''' Asignamos a cada estacion sus trabajos correspondientes '''
for fila in range (2, fila_ultima +1):
    M_estaciones[hoja. cell (row= fila ,column=1).value - 1].append(hoja\
    . cell (row= fila ,column=5).value)
```

Mediante el código simple mostrado anteriormente, se puede hacer la extracción de los datos del documento Excel para definir la matriz de estaciones de nuestro caso práctico, la cual se define concretamente como:

$$\text{M\_estaciones} = \left[\left[1,2,3 \right] \left[4,5,6,7,8 \right] \left[9,10,11 \right] \left[12,13,14,15 \right] \left[16,17,18,19 \right] \right]$$

- Matriz de precedencia (M_precedencia).

Una matriz de precedencia es una matriz cuadrada, compuesta por ceros y unos en el cual las filas están ordenadas con números de forma consecutiva al igual que la columnas de la matriz [25]. En nuestro caso, las entrada en la matriz son las siguientes:

1. Si el elemento de la columna j precede inmediatamente al elemento de la fila i, se añade un 1 en la fila i y columna j.
2. Cualquier otra entrada corresponde a un 0 en la posición i,j.

A continuación, se muestra un ejemplo de la matriz de precedencia para la estación 1 de nuestro caso práctico en base a su diagrama de precedencias:

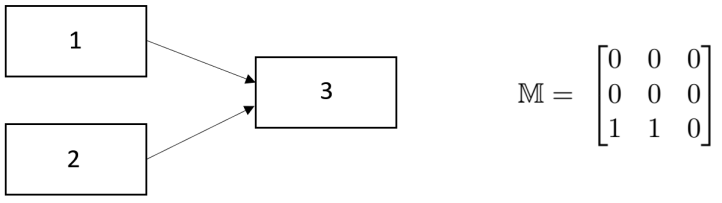

Figura 4.4 Diagrama de precedencia y su matriz de precedencia asociada para un ejemplo de 3 estaciones [Fuente: Elaboración propia].

Para simplificar la codificación en Python de la matriz M_precedencia, y por tanto el modelado de la formulación matemática descrita en la sección 4.1, se creará en primer lugar una

matriz compuesta por listas que recoja los datos externos de precedencia. A esta matriz auxiliar la denotaremos como datos_precedencia. Situado en la hoja "Hoja1" del documento Excel "DatosExcel2" (al igual que para la matriz M_estaciones), la columna 6 y 7 recogen los trabajos precedentes al trabajo correspondiente de la fila *i* y columna 5. Respecto a la base de datos de la tabla, existirán tantas columnas de precedencia como el máximo que un trabajo *i* puede tener en base a trabajos predecesores y rellenándose con 0 tantas veces como un trabajo carezca de predecesor.

La siguiente tabla muestra los datos para nuestro caso práctico:

Tabla 4.4 Tabla Excel que almacena los datos de precedencia de la línea de ensamblaje [Fuente: Elaboración propia].

	E	F	G
1	Trabajo	Precedencia 1	Precedencia 2
2	1	0	0
3	2	0	0
4	3	1	2
5	4	0	0
6	5	0	0
7	6	4	5
8	7	6	0
9	8	6	0
10	9	0	0
11	10	0	0
12	11	0	0
13	12	0	0
14	13	12	0
15	14	13	0
16	15	13	0
17	16	0	0
18	17	16	0
19	18	16	0
20	19	17	0

Para mayor facilidad de seguimiento en la codificación, se han ocultado las primeras 4 columnas correspondientes a las estaciones, los operarios requeridos tipo I, tipo II y tipo III ya que no son necesarios para su extracción de datos.
El código que mostraremos a continuación devuelve la matriz datos_precedencia de dos dimensiones. La primera dimensión recorre los trabajos de la línea de ensamblaje. La segunda dimensión recorre 2 valores para la definición de los trabajos precedentes al trabajo *i*, designando el número de trabajo concreto para el primer valor y el conjunto de trabajos predecesores para el segundo valor. Para nuestro caso práctico obtenemos la siguiente matriz datos_precedencia:

$$\text{datos\_precedencia} = \Big[\ [1,0]\ [2,0]\ [3,[1,2]]\ [4,0]\ [5,0]\ [6,[4,5]]\ [7,6]\ [8,6]\ [9,0]\ [10,0]\ [11,0]\ [12,0]\ [13,12]\ [14,13]\ [15,13]\ [16,0]\ [17,16]\ [18,16]\ [19,17]\ \Big]$$

Por tanto, si hacemos referencia a la posición datos_precedencia[x][0], nos devolverá el valor del trabajo $x+1$, mientras que la posición datos_precedencia[x][1] devolverá el valor o conjunto de trabajos predecesores al trabajo $x+1$. Cabe puntualizar que en caso de que un trabajo no tenga predecesores, su valor en la posición datos_precedencia[x][1] será 0.

El código que recoge la los datos de precedencia para cualquier problema en una línea de ensamblaje Low-Volume Mixta es el siguiente descrito:

```
''' Definimos los limites de la tabla mediante la variable
 numero_trabajos '''
numero_trabajos = hoja.max_row - 1

''' Inicializamos a cero los datos de precedencia '''
datos_precedencia = []
for i in range(numero_trabajos):
    datos_precedencia.append([])
    for j in range(2):
        datos_precedencia[i].append(0)

''' Asignamos a los datos de precedencia los trabajos de la linea '''
for i in range(numero_trabajos):
    datos_precedencia[i][0] = i+1

''' Asignamos a los datos de precedencia los trabajos predecesores '''
for fila in range(2, numero_trabajos+2):
    for columna in range(6, 8):
        if (hoja.cell(row=fila,column=columna).value) != 0:
            if columna == 6:
                datos_precedencia[fila-2][1] = hoja.cell(row=fila\
                ,column=columna).value
            elif columna == 7:
                v=[]
                v.append(hoja.cell(row=fila,column=columna-1).value)
                v.append(hoja.cell(row=fila,column=columna).value)
                datos_precedencia[fila-2][1] = v
            elif columna > 7:
                v.append(hoja.cell(row=fila,column=columna).value)
                datos_precedencia[fila-2][1] = v
```

Para el bloque de asignación a los datos de precedencia de los trabajos predecesores, se ha utilizado de una variable fila para recorrer todos los trabajos de la línea de ensamblaje comenzando por el valor 2 debido a como se puede ver en la tabla anterior, el primer trabajo se establece en dicha posición, dejando la primera para el encabezado. Por otro lado, la variable columna recorrerá las columnas predecesoras de la tabla, en nuestro caso la 6 y 7, pudiendo ser estos valores modificables. Para comenzar con la designación de los trabajos predecesores, debe darse el caso de que el valor de la celda situada en la columna columna sea distinto de 0, debiéndose crear una lista v en caso de que la columna 7 contenga un valor distinto de cero, es decir, que el trabajo que estemos analizando contenga más de un trabajo predecesor.

Una vez tenemos recopilados los datos de precedencia, podemos codificar la matriz de precedencia fácilmente recibiendo como argumento la matriz datos_precedencia:

```
''' Inicializamos a 0 la matriz de precedencia '''

M_precedencia = []
for i in range(numero_trabajos):
    M_precedencia.append([])
    for j in range(numero_trabajos):
        M_precedencia[i].append(0)


''' Asignamos a cada trabajo sus trabajos predecesores
correspondientes '''
for valor in range(numero_trabajos):
    x=datos_precedencia[valor][0]
    y=datos_precedencia[valor][1]
    if type(y) == int:
        if y>0:
            M_precedencia[x-1][y-1] = 1
    else:
        tamano_vconverge = len(y)
        for numero in range(tamano_vconverge):
            M_precedencia[x-1][y[numero]-1] = 1
```

En el código representado en el segundo bloque para la asignación de trabajos predecesores se hace uso de dos variables auxiliares: x e y, las cuales recogen el trabajo y su correspondientes predecesores respectivamente. Puesto que la variable y puede ser de tipo lista en caso de que el trabajo x tenga más de un trabajo predecesor o de tipo int (entero) si sólo tiene un trabajo predecesor o ninguno (el valor de y sería igual a 0) analizamos ambos casos mediante condicionales para posteriormente asignar el valor 1 a la fila y columna de la matriz de precedencia.

La necesidad de codificar matrices y listas adicionales en nuestro problema de optimización viene dado por el uso de familias de conjuntos indexados en el modelo matemático utilizado, ya que sus valores y conjuntos se ven modificados en caso de que se estudie un diferente problema de optimización variando así sus datos y parámetros.

Los conjuntos simples: $i = 1,....,N$, $j = 1,....,J$, $t = 1,....,T$, y $k = 1,....,M$ los podemos modelar fácilmente en Python como hemos visto en anteriores ejemplos:

```
model.TRABAJOS = RangeSet(1,len(M_precedencia))
model.MODELOS = set([1, 2])
model.PERIODOS = RangeSet(1,22)
model.ESTACIONES = RangeSet(1,len(M_estaciones))
```

Donde len(M_precedencia) y len(M_estaciones) devuelven el número de trabajos de la línea de ensamblaje y el número total de estaciones respectivamente. El comando RangeSet define un rango de números de clase Set, recibiendo dos argumentos: el primer y último valor del rango.

Como hemos comentado en la sección 2.3 de nuestro caso práctico, se establecerá una ventana de planificación de longitud 22 periodos.

Para los conjuntos indexados de nuestro modelo matemático, se harán uso de diccionarios de datos en Python. Un diccionario es una estructura de datos y un tipo de datos dentro de Python con características singulares capaz de almacenar cualquier tipo de valor tales como enteros, cadenas, listas e incluso otras funciones. Son similares a las listas, con la excepción de que no están ordenados y pueden ser indexados por cualquier tipo (cadenas, números...). Los índices de los diccionarios son llamados llaves y son únicas para cada diccionario [21].

A continuación se muestra un ejemplo de un diccionario indexado por los valores 1,2 y 3 que contiene conjuntos:

```
Diccionario = {}
Diccionario[1] = [3,5,7]
Diccionario[2] = [1,3,5]
Diccionario[3] = [2,4,6]
```

Un diccionario se define mediante las llaves {} pudiendo ser asignado cualquier tipo (valores, cadenas, listas, etc) bajo sus índices. Si imprimimos por pantalla el diccionario del ejemplo vemos como cada índice contiene su conjunto correspondiente:

```
Diccionario = {1: [3, 5, 7], 2: [1, 3, 5], 3: [2, 4, 6]}
```

Debido a la alta funcionalidad y versatilidad de los diccionarios para las familias de conjuntos indexados, definiremos los siguientes conjuntos de nuestro modelo matemático mediante diccionarios:

P_i = conjunto de predecesores inmediatos a la tarea i
W_k = conjunto de actividades pertenecientes a la estación k
M_i = conjunto de actividades dentro de la estación perteneciente a i
S_i = conjunto de actividades que pertenecen a la estación predecesora respecto a la estación de la actividad i

Para la modelar P_i en Pyomo, crearemos el diccionario precedencia_trabajos recibiendo como principal argumento en la codificación la matriz de precedencia anteriormente descrita. Se mostrarán mediante comentarios las pautas seguidas para la definición del conjunto:

```
''' Definimos precedencia_trabajos como un diccionario '''
precedencia_trabajos = dict()
''' Por cada indice compuesto por los trabajos de la linea, inicializamos
su conjunto asignado '''
for i in range(numero_trabajos):
    precedencia_trabajos[i+1] = []
    ''' Asignamos los trabajos predecesores a i recorriendo la matriz de
    precedencia '''
    for j in range(numero_trabajos):
        if M_precedencia[i][j] == 1:
            precedencia_trabajos[i+1].append(j+1)
```

De la misma manera que el conjunto P_i, W_k podemos definirlo en nuestro modelado en Python mediante el diccionario trabajos_por_estacion:

```
''' Definimos trabajos_por_estacion como un diccionario '''
trabajos_por_estacion = dict ()
''' Por cada indice compuesto por estaciones de la linea , inicializamos
su conjunto asignado '''
for i in range(len(M_estaciones)):
    trabajos_por_estacion [i+1]= []
''' Asignamos los trabajos pertenecientes a la estacion correspondiente '''
for s in range(len(M_estaciones)):
    for m in range(len(M_estaciones[s])):
        trabajos_por_estacion [s+1].append(M_estaciones[s][m])
```

Donde la variable s recorre cada una de las estaciones de la línea de ensamblaje mientras que la variable m recorre las actividades que pertenecen a s, siendo finalmente asignadas a cada índice del diccionario trabajos_por_estacion.

El conjunto M_i, perteneciente a la restricción entre modelos (7) podemos modelarlo bajo el diccionario precedencia_modelos, un diccionario de gran volumen en cuanto a datos se refiere ya que devuelve todas las actividades de la estación perteneciente al trabajo concreto, siendo a su vez las llaves del diccionario.

```
''' Definimos precedencia_modelos como un diccionario '''
precedencia_modelos = dict ()
''' Por cada indice compuesto por los trabajos de la linea , inicializamos
su conjunto asignado '''
for i in range(numero_trabajos):
    precedencia_modelos[i+1] = []
''' Asignamos a cada indice del diccionario , todos los trabajos de su
misma estacion '''
for s in range(len(M_estaciones)):
    for m in range(len(M_estaciones[s])):
        for n in range(len(M_estaciones[s])):
            precedencia_modelos[M_estaciones[s][m]]\
            .append(M_estaciones[s][n])
```

Recibiendo como argumento M_estaciones, la variable s recorre las estaciones de la línea de ensamblaje, mientras que m recorre cada una de las actividades pertenecientes a la estación s. Los índices del diccionario se designan al bucle de ambas variables, mientras que sus conjuntos se establecen gracias a la variable n que traza todos las actividades de la línea.

Por último, procederemos con el modelado del conjunto S_i para su uso en la restricción entre estaciones (8) bajo el diccionario precedencia_estaciones. La codificación en Python resulta algo más laboriosa ya que se requiere del uso de la matriz estructura línea de ensamblaje como argumento. A continuación, comentaremos los puntos más importantes mediante comentarios para mejorar la comprensión del modelado del conjunto indexado:

```
''' Definimos precedencia_estaciones como un diccionario '''
```

```
precedencia_estaciones = dict()
''' Por cada indice compuesto por los trabajos de la linea, inicializamos
su conjunto asignado '''
for i in range(len(M_precedencia)):
    precedencia_estaciones[i+1] = []
''' La variable s recorre cada una de las lineas '''
for s in range(len(M_estructura)):

    ''' Si s corresponde a linea final de ensamblado: '''
    if s == len(M_estructura) - 1:
        for m in range(len(M_estructura[s])):
            if m==0:
                ''' A cada uno de los trabajos de la estacion de
                ensamblado final se asignan los trabajos de las
                estaciones precedentes '''
                for n in range(len(M_estructura[s][m])):
                    for q in range(len(M_estructura)-1):
                        for r in range(len(M_estructura[q][-1])):
                            precedencia_estaciones[M_estructura[s][m][n]]\
                            .append(M_estructura[q][-1][r])
            else:
                ''' Por cada trabajo de la estacion m se asignan todos
                los trabajos de la estacion m-1'''
                for n in range(len(M_estructura[s][m])):
                    for p in range(len(M_estructura[s][m-1])):
                        precedencia_estaciones[M_estructura[s][m][n]]\
                        .append(M_estructura[s][m-1][p])

    ''' Si s corresponde a unas de las lineas paralelas: '''
    else:
        for m in range(len(M_estructura[s])):
            if m == 0:
                ''' Si estamos ante la primera estacion de cada linea, no
                se asignan trabajos, solo se inicializa '''
                for n in range(len(M_estructura[s][m])):
                    precedencia_estaciones[M_estructura[s][m][n]] = []
            else:
                ''' Por cada trabajo de la estacion m se asignan todos
                los trabajos de la estacion m-1'''
                for n in range(len(M_estructura[s][m])):
                    for p in range(len(M_estructura[s][m-1])):
                        precedencia_estaciones[M_estructura[s][m][n]]\
                        .append(M_estructura[s][m-1][p])
```

4.2.3 Modelado de parámetros y variables

Comenzaremos por el modelado de los parámetros de nuestro problema de optimización: t_{ij}, $hreq1_{ij}$, $hreq2_{ij}$, $hreq3_{ij}$, $htotal1_k$, $htotal2_k$ y $htotal3_k$. Al tratarse de parámetros que contiene multitud de datos debido al gran número de trabajos de nuestro caso práctico y al tener que diferenciar los valores por cada uno de los modelos que se procesen en la línea de ensamblaje, no podemos

definirlos con una simple inicialización como vimos en el problema simple de programación lineal del apartado 3.3.2.

Se mostrará el modelado los parámetros t_{ij}, $hreq1_{ij}$ y $htotal1_k$ (ya que los demás se definirían bajo la misma estructura) a partir de la extracción de datos de la hoja "Hoja1" del documento Excel "TablasDatos2", una hoja compuesta por tablas de todos los parámetros y conjuntos de nuestro problema práctico. A continuación, se muestra un extracto del mismo:

Tabla 4.5 Extracto de una hoja Excel que almacena los distintos parámetros dato [Fuente: Elaboración propia].

	K	L	M	N	O	P	Q	Z	AA	AB
1	**TablaTiempo**	**Columna1**	**Columna2**		**TablaHreq1**	**Columna1**	**Columna2**		**TablaHtotal1**	**Columna1**
2	TRABAJOS	1	2		TRABAJOS	1	2		ESTACIONES	htotal1
3	1	1	1		1	0	1		1	2
4	2	2	2		2	2	1		2	2
5	3	1	1		3	2	0		3	1
6	4	1	1		4	2	0		4	3
7	5	2	1		5	2	0		5	2
8	6	1	1		6	1	1			
9	7	1	1		7	0	0			
10	8	2	1		8	0	0			
11	9	1	2		9	1	0			
12	10	2	2		10	1	1			
13	11	1	1		11	1	0			
14	12	1	1		12	3	0			
15	13	2	1		13	3	1			
16	14	2	1		14	3	2			
17	15	1	2		15	3	3			
18	16	2	1		16	0	0			
19	17	2	1		17	2	1			
20	18	1	1		18	1	0			
21	19	1	1		19	2	0			

Donde la tabla "TablaTiempo" designa al parámetro t_{ij}, compuesta por una tabla donde las filas corresponden a los trabajos de la línea de ensamblaje, mientras que las columnas definen los modelos. Del mismo modo, la tabla "TablaHreq1" designa al parámetro $hreq1_{ij}$ donde las columnas y filas de la tabla se distribuyen de la misma forma comentada anteriormente. Por último, la tabla "TablaHtotal1" se define para la extracción de datos del parámetro $htotal1_k$. En la primera columna se enumeran las estaciones de la línea mientras que la segunda se anotan los valores correspondientes de los operarios totales tipo I.

Para su modelado en Pyomo, haremos de la función tabla.cell(row=x,column=y).value mediante la biblioteca Openpyxl para definir un diccionario que contengan todos los datos de cada parámetro. Posteriormente, inicializaremos los parámetros indexados del modelo bajo el comando initialize como hemos visto en el apartado 3.3.2:

```
''' Modelado del parametro t_{ij} '''
tiempo={}
for j in model.MODELOS:
    for i in model.TRABAJOS:
        tiempo[i,j] = tabla.cell(row=i+2,column=11+j).value
model.tiempo=Param(model.TRABAJOS, model.MODELOS, initialize=tiempo)

''' Modelado del parametro hreq1_{ij} '''
hreq1={}
```

```
for j in model.MODELOS:
    for i in model.TRABAJOS:
        hreq1[i,j] = tabla.cell(row=i+2,column=15+j).value
model.hreq1=Param(model.TRABAJOS,model.MODELOS,initialize=hreq1)

''' Modelado del parametro htotal1_{k} '''
htotal1 ={}
for k in model.ESTACIONES:
        htotal1[k] = tabla.cell(row=k+2,column=28).value
model.htotal1 =Param(model.ESTACIONES,initialize=htotal1)
```

Una vez tenemos los parámetros de nuestro caso práctico, modelaremos nuestras variables x_{itj} y Lt_j de forma sencilla mediante la clase Var y definiendo el dominio de cada una de ellas:

```
model.x = Var(model.TRABAJOS, model.PERIODOS, model.MODELOS,\
          within=Boolean)
model.lt = Var(model.MODELOS, within=PositiveIntegers)
```

De esta forma, queda definida automáticamente la restricción (10) de nuestro modelo matemático, ya que Pyomo obliga bajo el argumento within que x_{itj} tome valores binarios (booleanos) mientras que la variable Lt_j solo puede tomar valores naturales positivos.

4.2.4 Modelado de la función objetivo y restricciones

Función objetivo

Al igual que se ha dividio en el modelo matemático la función objetivo bajo los términos T_1 y T_2, en Pyomo, para mayor facilidad de codificación, definiremos ambas expresiones bajo la clase Expression como model.Operarios_ociosos y model.LeadTime respectivamente.

```
''' Termino T_1'''
model.Operarios_ociosos = Expression(expr = sum(sum(model.htotal1[k] +\
model.htotal2[k] + model.htotal3[k] - sum(sum(model.hreq1[n,j] *\
(sum(model.x[n,k,j] for k in sequence(t)) - sum(model.x[n,k,j] for k in\
sequence(t-model.tiempo[n,j]))) + model.hreq2[n,j] * (sum(model.x[n,k,j]\
for k in sequence(t)) - sum(model.x[n,k,j] for k in sequence(t -\
model.tiempo[n,j]))) + model.hreq3[n,j] * (sum(model.x[n,k,j] for k in\
sequence(t)) - sum(model.x[n,k,j] for k in sequence(t -\
model.tiempo[n,j]))) for j in model.MODELOS) for n in\
 trabajos_por_estacion[k]) for k in model.ESTACIONES) for\
t in model.PERIODOS))

''' Termino T_2'''
model.LeadTime = Expression(expr = sum(model.lt[j] for j in\
model.MODELOS))
```

Para mayor facilidad computacional y puesto que no interfiere en los resultados de nuestro caso práctico, se ha tomado el término T_2 como la suma del Lead Time de cada uno de ellos.

Se puede analizar en el código anterior como el conjunto W_k del modelo matemático se ha expresado mediante el diccionario indexado por la estaciones de la línea de ensamblaje trabajos_por_estacion.

Una vez tenemos las expresiones de ambos términos modeladas, simplemente, definiremos la función objetivo (mediante el nombre obj) en Pyomo a través de pesos igualitarios mediante la clase Objective y recibiendo los argumentos de ambas expresiones:

```
model.obj = Objective(expr = 0.5*model.LeadTime +\
            0.5*model.Operarios_ociosos)
```

Restricciones

El modelado de las restricciones lo desarrollaremos en el mismo orden secuencial que aparece en el modelo matemático de la sección 4.1.

- Restricción (1): $\sum_{t=1}^{T} x_{itj} = 1, \qquad i = 1,...,N, j = 1,...,J$

```
def rest1_rule(model, i, j):
    return sum(model.x[i,t,j] for t in model.PERIODOS) == 1
model.rest1 = Constraint(model.TRABAJOS,model.MODELOS,rule=rest1_rule)
```

- Restricción (2): $\sum_{t=1}^{T} tx_{itj} \geq (\sum_{t=1}^{T} tx_{ptj}) + t_{pj}, \qquad i = 1,...,N, p \in P_i \; j = 1,...,J$

```
@model.Constraint(model.TRABAJOS, model.TRABAJOS, model.MODELOS)
def rest2(model, i, p, j):
    if p in precedencia_trabajos[i]:
        return sum(t * model.x[i, t, j] for t in model.PERIODOS) >= \
        (sum(t * model.x[p, t, j] for t in model.PERIODOS)+ \
        model.tiempo[p,j])
    else:
        return Constraint.NoConstraint
```

Del mismo modo que se usan las funciones rule para la simplificación de colecciones de restricciones y funciones objetivos, se puede usar la función @model.Constraint el cual recibe los argumentos de los diferentes conjuntos que conforman las restricciones para posteriormente definir la función que englobará el bloque de restricciones. La particularidad de esta función es permitir el uso de comandos tales como condicionales o bucles. En nuestro caso, nuestra función recibirá el nombre rest2.
De este modo, en caso de que $p \in model.TRABAJOS$ no esté contenido en el conjunto P_i para $i \in model.TRABAJOS$ la función no devolverá ninguna restricción: return Constraint.NoConstraint. En caso contrario, la función devuelve la restricción correspondiente de procedencia para el trabajo i.

- Restricción (3): $\sum_{n \in W_k} hreq1_{nj}(\sum_{k=1}^{t} x_{nkj} - \sum_{k=1}^{t-t_{nj}} x_{nkj}) \leq htotal1_k \qquad k = 1,...,M, \; t = 1,...,T, \; j = 1,...,J$

```
@model.Constraint(model.PERIODOS, model.ESTACIONES, model.MODELOS)
def rest3(model, t, k, j):
        return sum(model.hreq1[n,j] * (sum(model.x[n,k,j] for k in \
        sequence(t)) - sum(model.x[n,s,j] for s in sequence(t - \
        model.tiempo[n,j]))) for n in trabajos_por_estacion[k]) <= \
        model.htotal1[k]
```

- Restricción (4): $\sum_{n\in W_k} hreq2_{nj}(\sum_{k=1}^{t} x_{nkj} - \sum_{k=1}^{t-t_{nj}} x_{nkj}) \leq htotal2_k \qquad k = 1,...,M,$
$t = 1,...,T,\ j = 1,...,J$

```
@model.Constraint(model.PERIODOS, model.ESTACIONES, model.MODELOS)
def rest4(model, t, k, j):
        return sum(model.hreq2[n,j] * (sum(model.x[n,k,j] for k in \
        sequence(t)) - sum(model.x[n,s,j] for s in sequence(t - \
        model.tiempo[n,j]))) for n in trabajos_por_estacion[k]) <= \
        model.htotal2[k]
```

- Restricción (5): $\sum_{n\in W_k} hreq3_{nj}(\sum_{k=1}^{t} x_{nkj} - \sum_{k=1}^{t-t_{nj}} x_{nkj}) \leq htotal3_k \qquad k = 1,...,M,$
$t = 1,...,T,\ j = 1,...,J$

```
@model.Constraint(model.PERIODOS, model.ESTACIONES, model.MODELOS)
def rest5(model, t, k, j):
        return sum(model.hreq3[n,j] * (sum(model.x[n,k,j] for k in \
        sequence(t)) - sum(model.x[n,s,j] for s in sequence(t - \
        model.tiempo[n,j]))) for n in trabajos_por_estacion[k]) <= \
        model.htotal3[k]
```

Como podemos ver en nuestro modelado de las restricciones (3),(4) y (5), se ha usado una nueva función llamada sequence. La función sequence(inicio,final,secuenciación) devuelve una lista aritmética de progresión de números naturales. Con un simple argumento, sequence devuelve una secuencia que empieza por el valor 1. Por tanto, sequence(i) devolverá [1,2,...,i], mientras que con dos argumentos sequence(i,j) devolvería [i, i+1, i+2,...,j][21].

- Restricción (6): $Lt_j \geq x_{itj}(t + t_{ij} - 1) \qquad i = 1,...,N, t = 1,...,T, j = 1,...,J$

```
def rest6_rule(model, i, t, j):
    return model.lt[j] >= model.x[i,t,j] * (t + model.tiempo[i,j] -1)
model.rest6 = Constraint(model.TRABAJOS, model.PERIODOS, \
              model.MODELOS, rule=rest6_rule)
```

- Restricción (7): $\sum_{t=1}^{T} tx_{itj} \geq (\sum_{t=1}^{T} tx_{mtj-1}) + t_{mj-1}, \qquad i = 1,...,N, m \in M_i\ j = 2,...,J$

```
@model.Constraint(model.TRABAJOS, model.TRABAJOS, model.MODELOS)
def rest7(model, i, m, j):
    if j == 1:
        return Constraint.NoConstraint
    else:
        if m in precedencia_modelos[i]:
            return sum(t * model.x[i, t, j] for t in model.PERIODOS)\
            >= (sum(t * model.x[m, t, j-1] for t in model.PERIODOS)+\
            model.tiempo[m,j-1])
        else:
            return Constraint.NoConstraint
```

El primer condicional de la función no devuelve ninguna restricción en caso de que el modelo j sea igual a 1, ya que j se mueve en el conjunto $j = 2,..,J$. Por otro lado, para el modelado de las restricciones entre modelos se ha usado la misma notación que para el modelado de la restricción entre precedencias (2), haciendo uso en este caso del diccionario precedencia_modelos.

- Restricción (8): $\sum_{t=1}^{T} tx_{it\,j} \geq (\sum_{t=1}^{T} tx_{st\,j}) + t_{sj} \qquad i = 1,...,N, s \in S_i \; j = 1,...,J$

```
@model.Constraint(model.TRABAJOS, model.TRABAJOS, model.MODELOS)
def rest8(model, i, s, j):
    if s in precedencia_estaciones[i]:
        return sum(t * model.x[i, t, j] for t in model.PERIODOS) >= \
        (sum(t * model.x[s, t, j] for t in model.PERIODOS)+ \
        model.tiempo[s,j])
    else:
        return Constraint.NoConstraint
```

Para el modelado de la restricción entre estaciones (8), hacemos uso del conjunto S_i bajo el diccionario precedencia_estaciones mediante la misma notación empleada en las restricciones (2) y (8).

La codificación completa en Python del caso práctico se encuentra en el anexo A.3.

5 Resultados

En el presente capítulo, se realizará la llamada al Solver para obtener los resultados computacionales de nuestro caso práctico en el punto 5.1 para finalmente, en el punto 5.2, realizar una serie de análisis experimentales que abordan estos tipos de problemas.

5.1 Resultados computacionales

Para evaluar la efectividad del modelo propuesto, una vez solventado y codificado todos los parámetros y variables, funciones objetivo, restricciones y los distintos conjuntos de nuestro caso práctico, procedemos a la llamada de un Solver para resolver nuestro problema de programación lineal entera.

Para el muestreo de los resultados computacionales, todos los algoritmos implicados, al igual que para el modelado de nuestro problema de optimización, se han codificado en lenguaje Python 3.6 y depurados bajo las siguientes características: 2.7GHZ i5-7200U CPU, Windows 10 PC.

Puesto que nuestro modelo matemático trabaja con gran cantidad de variables, en nuestro caso: $x \in X$, un vector de tres dimensiones: i, j, t compuesto por 836 variables binarias: $x[i,t,j]$ y $lt \in LT$, un vector de una dimensión: j y compuesto por 2 variables enteras $lt[1]$ y $lt[2]$, estamos obligados a la implementación de un código que imprima las variables x_{itj} si su valor devuelve 1, es decir que el trabajo i del modelo j comience en el periodo t.

```
for j in model.MODELOS:
    for t in model.PERIODOS:
        for i in model.TRABAJOS:
            if model.x[i,t,j].value == 1:
                print("x[" + str(i) + "," + str(t) + "," + str(j) + "]=",\
                model.x[i,t,j].value)
```

A continuación, se presentan a modo resumen los resultados obtenidos de nuestro problema práctico para posterior análisis de los mismos junto con el volcado a un diagrama de Gantt para poder analizar de forma visual las precedencias y restricciones que se ha invocado en nuestro modelo de optimización.

Tabla 5.1 Resultados Experimentales del caso práctico [Fuente: Elaboración propia].

x[i,t,1]	Valor
x[1,1,1]	1
x[2,2,1]	1
x[3,4,1]	1
x[4,7,1]	1
x[5,5,1]	1
x[6,8,1]	1
x[7,11,1]	1
x[8,9,1]	1
x[9,4,1]	1
x[10,2,1]	1
x[11,1,1]	1
x[12,5,1]	1
x[13,6,1]	1
x[14,9,1]	1
x[15,8,1]	1
x[16,12,1]	1
x[17,14,1]	1
x[18,16,1]	1
x[19,17,1]	1

x[i,t,2]	Valor
x[1,6,2]	1
x[2,5,2]	1
x[3,8,2]	1
x[4,13,2]	1
x[5,12,2]	1
x[6,15,2]	1
x[7,17,2]	1
x[8,16,2]	1
x[9,6,2]	1
x[10,9,2]	1
x[11,5,2]	1
x[12,11,2]	1
x[13,12,2]	1
x[14,13,2]	1
x[15,14,2]	1
x[16,18,2]	1
x[17,20,2]	1
x[18,19,2]	1
x[19,21,2]	1

Lt[j]	Valor
Lt[1]	17
Lt[2]	21

F.O	493
T1	38
T2	455

hlibre_proceso[t]	Valor
hlibre_proceso[1]	3
hlibre_proceso[2]	2
hlibre_proceso[3]	2
hlibre_proceso[4]	1
hlibre_proceso[5]	8
hlibre_proceso[6]	4
hlibre_proceso[7]	4
hlibre_proceso[8]	9
hlibre_proceso[9]	8
hlibre_proceso[10]	8
hlibre_proceso[11]	9
hlibre_proceso[12]	11
hlibre_proceso[13]	6
hlibre_proceso[14]	3
hlibre_proceso[15]	5
hlibre_proceso[16]	6
hlibre_proceso[17]	8
hlibre_proceso[18]	5
hlibre_proceso[19]	5
hlibre_proceso[20]	2
hlibre_proceso[21]	4
hlibre_proceso[22]	0

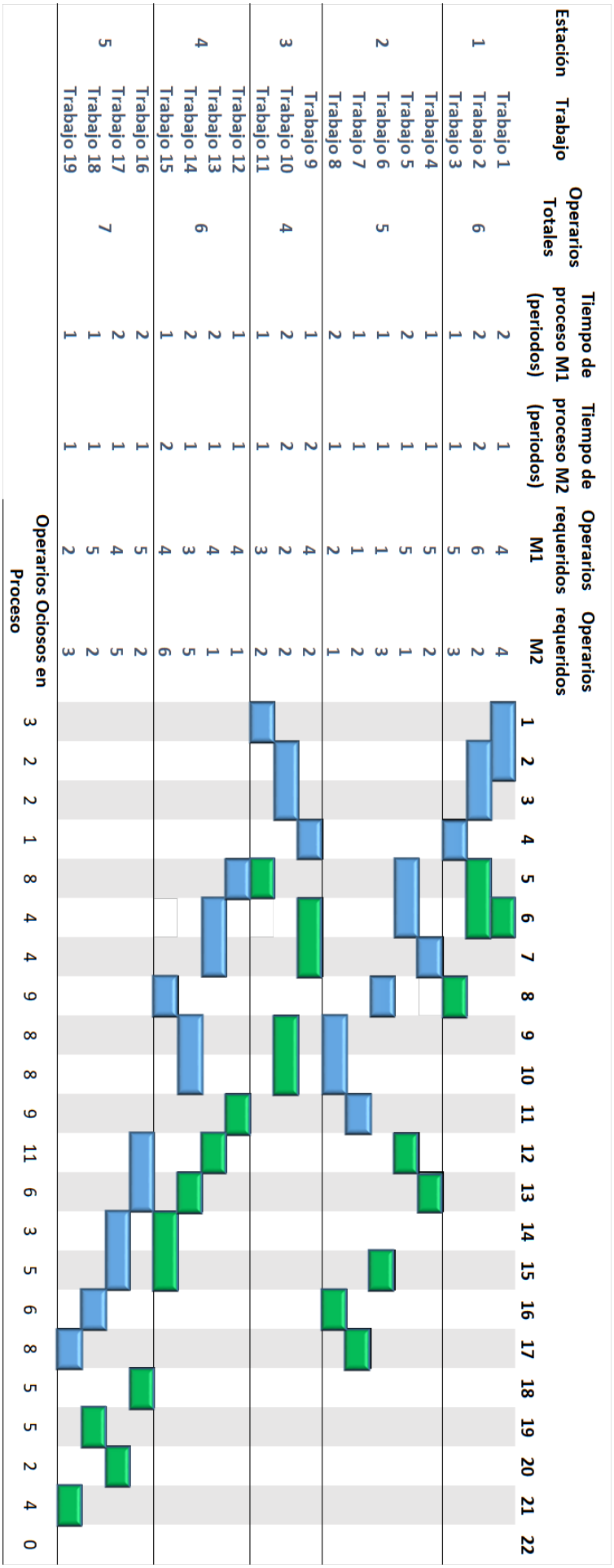

Figura 5.1 Diagrama de Gantt resultante del caso práctico [Fuente: Elaboración propia].

Observando la línea del tiempo, podemos observar cómo se cumplen cada una de las restricciones otorgadas en nuestro modelo matemático en base a los datos proporcionados de precedencia y operarios requeridos de tipo X para el trabajo de un modelo concreto.

Para el estudio de la función objetivo, el término T_1, referido a la suma del Lead Time de cada uno de los modelos, devuelve el valor de 38 periodos, siendo por tanto el Lead Time del modelo 1 de 17 periodos y el Lead Time del modelo 2 de 21 periodos que coincide con el Lead Time total de la línea de ensamblaje del caso práctico.

El término $T_2 = 455$ puede resultar confuso debido al gran valor obtenido, sin embargo, esta cifra no es exacta a efecto de operarios ociosos en nuestra línea de ensamblaje real. Como ya hemos estudiado anteriormente el término T_2 se modeló matemáticamente como:

$$T_2 = \sum_{t=1}^{T}(\sum_{k=1}^{M}((htotal1_k + htotal2_k + htotal3_k) - \sum_{n\in W_k}\sum_{j=1}^{J}(hreq1_{nj} * (\sum_{k=1}^{t} x_{nkj} - \sum_{k=1}^{t-t_{nj}} x_{nkj}) +$$

$$hreq2_{nj} * (\sum_{k=1}^{t} x_{nkj} - \sum_{k=1}^{t-t_{nj}} x_{nkj}) + hreq3_{nj} * (\sum_{k=1}^{t} x_{nkj} - \sum_{k=1}^{t-t_{nj}} x_{nkj})))$$

Donde el término $\sum_{k=1}^{M}(htotal1_k + htotal2_k + htotal3_k)$ representa que en cada periodo se están procesando cada una de las estaciones de nuestra línea de ensamblaje, lo cual no se puede materializar en nuestro caso práctico ya que se ha partido de t=0 con únicamente dos modelos para mejor respuesta computacional. Por tanto, dentro de los 455 operarios ociosos que devuelve nuestro modelo de programación, debemos hacer un estudio de los operarios ociosos a efectos de proceso de la línea de ensamblaje sin contar los operarios ociosos debido a la espera de proceso o por finalización de la línea anterior a la ventana de planificación de 22 periodos, ya que, en la realidad, cada una de las estaciones estarían continuamente ocupadas por distintos modelos.

A efectos de función objetivo, el modelo matemático no diferencia entre tipos de operarios ociosos, por tanto, se obtendrán los mismos resultados independientemente para una expresión que represente que cada una de las estaciones de la línea de ensamblaje está en ocupación a otra que puntualice la minimización de operarios ociosos en proceso.

Para evaluar los operarios ociosos de proceso efectivos en nuestro problema práctico, se ha modelado una matriz binaria de dos dimensiones: t,k. La primera dimensión recorre cada uno de los 22 periodos de nuestra ventana de planificación mientras que la dimensión k recorre cada una de las estaciones de nuestra línea de ensamblaje. Los valores que contiene la matriz dependen de los trabajos que se estén procesando en cada periodo, donde:

$$Matriz_{tk} = \begin{cases} 1 & \text{si algún trabajo perteneciente a la estación k se está procesando en el periodo t} \\ 0 & \text{en otro caso} \end{cases}$$

Para ello, aplicaremos la siguiente expresión matemática:

$$\sum_{n\in W_K}\sum_{j=1}^{J}(\sum_{k=1}^{t} x_{nkj} - \sum_{k=1}^{t-t_{nj}} x_{nkj})$$

Devolviendo 0 en caso de que no se procese ningún trabajo de la estación k en el periodo t y $1,\ldots,p$ donde p representa el número total de trabajos pertenecientes a la estación k. De esta forma, es fácil analizar que, si nuestra expresión matemática devuelve un valor positivo, significará que la estación k está siendo procesada en el periodo t y por tanto podemos hacer su análisis de operarios ociosos en proceso.
A continuación, se muestra la matriz completamente modelada:

```
''' Inicializamos la matriz '''
Matriz = []
for i in range(22):
    Matriz.append([])
    for j in range(len(M_estaciones)):
        Matriz[i].append(0)
''' Asignamos a cada valor de la matriz el numero de trabajos en proceso \
pertenecientes a la estacion k '''
for t in model.PERIODOS:
    for k in model.ESTACIONES:
        Matriz[t-1][k-1]=value(sum(sum((sum(model.x[n,k,j] for k in \
        sequence(t)) - sum(model.x[n,s,j] for s in sequence(t - \
        model.tiempo[n,j])))  for j in model.MODELOS) for n in \
         trabajos_por_estacion[k]))
        if Matriz[t-1][k-1] > 1:
            Matriz[t-1][k-1] = int(1)
```

Por último, una vez creada nuestra matriz binaria, podemos definir la expresión que nos devuelve para cada periodo el número de operarios de proceso efectivos para posteriormente modelarlo en Python:

$$expr = (\sum_{k=1}^{M}((htotal1_k * Matriz_{tk} + htotal2_k * Matriz_{tk} + htotal3_k * Matriz_{tk}) - \sum_{n \in W_k} \sum_{j=1}^{J}(hreq1_{nj} * (\sum_{k=1}^{t} x_{nkj} - \sum_{k=1}^{t-t_{nj}} x_{nkj}) + hreq2_{nj} * (\sum_{k=1}^{t} x_{nkj} - \sum_{k=1}^{t-t_{nj}} x_{nkj}) + hreq3_{nj} * (\sum_{k=1}^{t} x_{nkj} - \sum_{k=1}^{t-t_{nj}} x_{nkj})))$$

```
hlibre_proceso_total = 0
for t in model.PERIODOS:
    hlibre_proceso_t = sum(model.htotal1[k]*valor[t-1][k-1] \
    + model.htotal2[k]*valor[t-1][k-1] + model.htotal3[k]*\
    valor[t-1][k-1] - sum(sum(model.hreq1[p,j] * (sum(model.x[p,k,j] for \
    k in sequence(t)) - sum(model.x[p,k,j] for k in sequence(t - \
    model.tiempo[p,j]))) + model.hreq2[p,j] * (sum(model.x[p,k,j] for k \
    in sequence(t)) - sum(model.x[p,k,j] for k in sequence(t - \
    model.tiempo[p,j]))) + model.hreq3[p,j] * (sum(model.x[p,k,j] for k \
    in sequence(t)) - sum(model.x[p,k,j] for k in sequence(t - \
    model.tiempo[p,j]))) for j in model.MODELOS) for p in \
     trabajos_por_estacion[k]) for k in model.ESTACIONES)

    hlibre_proceso_total = hlibre_proceso_total + value(hlibre_proceso_t)
    print('hlibre_proceso[' + str(t) + ']=', value(hlibre_proceso_t))
```

Donde hlibre_proceso asigna al número de operarios ociosos de proceso y hlibre_proceso_total es una variable que contiene la suma de todos los operarios de proceso ociosos a lo largo de la línea de ensamblaje.

Cabe destacar que en caso de haber representado el término T_1 de nuestra función objetivo bajo la misma expresión mostrada anteriormente, tendríamos un problema de programación no lineal, y por tanto sería mucho más complicado la búsqueda de un óptimo para nuestro problema debido a la multitud de variables y restricciones que soporta.

5.2 Análisis experimental

5.2.1 Análisis de requisitos laborales

La variación de requisitos laborales entre los distintos modelos de avión (Modelo 1 y modelo 2) son la causa en la misma línea de ensamblaje de la generación de ineficiencias laborales asumiendo operarios totales por estación fijados a efectos de un aumento de los operarios ociosos a lo largo de la ventana de planificación T. Las variaciones de los requisitos laborales para cada una de las estaciones en nuestro caso práctico se muestran en la figura 5.2

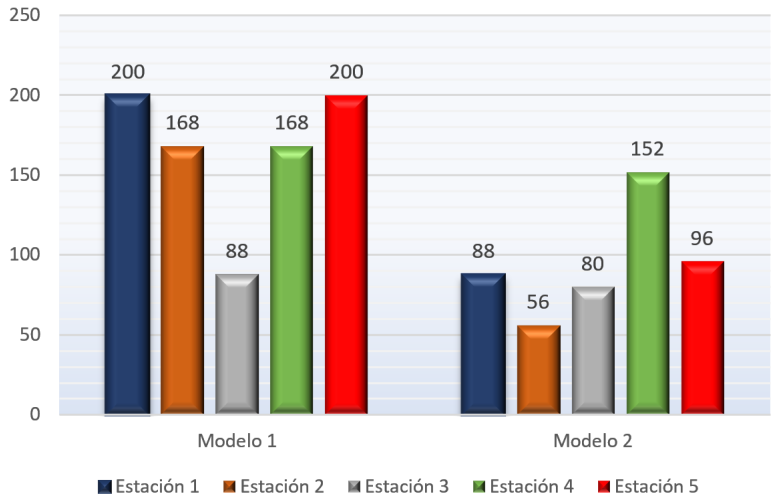

Figura 5.2 Variaciones de los requisitos laborales por cada estación y modelo de la linea de ensamblaje [Fuente: Elaboración propia].

Donde el requisito laboral se ha calculado como: $Yt_{ij}(hreq1_{ij} + hreq2_{ij} + hreq3_{ij})$

Siendo Y =horario regular laboral por periodo (horas): 8 horas/periodo

Se puede analizar en la figura como el modelo 2 tiene una carga laboral mucho menor que el Modelo de avión 1, siendo por tanto necesaria la asignación a cada estación de un número mayor de operarios para satisfacer la carga del Modelo 1 y obteniéndose a su vez un gran número de operarios ociosos para el procesamiento del modelo 2. Este balance de ineficiencia en base a la diferencia de requisitos laborales entre modelos se puede ver reflejado en el siguiente gráfico:

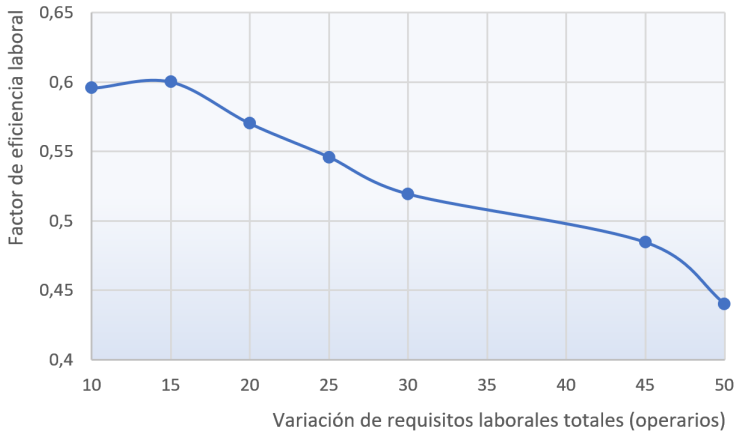

Figura 5.3 Gráfico de la variación de la eficiencia laboral frente a los requisitos laborales totales [Fuente: Elaboración propia].

Donde el estudio se ha reflejado en el caso práctico que nos ataña, atribuyéndole al modelo 2 de avión una disminución de carga de trabajo progresiva manteniendo la carga del modelo 1 puesto que el número de operarios totales por estación se mantiene constante al igual que los tiempos de proceso de cada uno de los trabajos. La tabla que se muestra a continuación refleja los resultados computacionales obtenidos respecto a Lead Time y Operarios Ociosos totales en relación a la variación entre la diferencia de requisitos laborales:

Tabla 5.2 Resultados experimentales frente a una variación de los requisitos laborales totales [Fuente: Elaboración propia].

Variación de requisitos laborales totales (operarios)	Lead Time	Operarios Ociosos de proceso	Nº Medio Op.ociosos por periodo	Nº Medio Op.requeridos por periodo M1	Nº Medio Op.requeridos por periodo M2	Nº Medio Op.requeridos por periodo	Factor de Eficiencia Laboral
10	19	87	4,578947368	3,631578947	3,105263158	6,736842105	0,595348837
15	19	82	4,315789474	3,631578947	2,842105263	6,473684211	0,6
20	19	89	4,684210526	3,631578947	2,578947368	6,210526316	0,570048309
25	19	94	4,947368421	3,631578947	2,315789474	5,947368421	0,54589372
30	19	100	5,263157895	3,631578947	2,052631579	5,684210526	0,519230769
45	19	99	5,210526316	3,631578947	1,263157895	4,894736842	0,484375
50	19	112	5,894736842	3,631578947	1	4,631578947	0,44

Donde el factor de eficiencia laboral se ha obtenido como:

$$\frac{\text{Nº Medio Op.requeridos por periodo}}{\text{Nº Medio Op.requeridos por periodo + Nº Medio Op.ociosos por periodo}}$$

Los experimentos computacionales para el análisis de la eficiencia laboral frente a la variación de requisitos laborales totales se han realizado atendiendo a operarios de una única especialidad debido a que las demandas de especialización de cada uno de los trabajos pueden entorpecer en los resultados de Lead Time y Operarios Ociosos en proceso.

5.2.2 Análisis multi-objetivo

Operarios totales por estación fijados

Puesto que las únicas variables de nuestro problema se atribuyen a $x_{it\,j}$ y lt_j, el análisis multiobjetivo carece de sentido, ya que una vez fijado los operarios requeridos $hreqX_i j$ y los operarios totales por cada estación $htotalX_k$, la minimización del Lead Time siempre traerá consigo una minimización directa de los operarios ociosos dada una cierta ventana de planificación T. Sin embargo, analizando la situación desde el punto de vista contrario, la minimización del número de Operarios Ociosos totales no supone una minimización del Lead Time de la línea de ensamblaje. Para analizar los resultados de ambos experimentos, completaremos la matriz de pago de nuestro caso práctico:

$$\text{Matriz de pago} = \begin{bmatrix} 21 & 455 \\ 22 & 455 \end{bmatrix}$$

Como podemos apreciar, la minimización del término T_1 devuelve el mismo valor obtenido anteriormente bajo la optimización con pesos de igual valor 0.5. Por otro lado, la minimización del término T_2 devuelve no solo el mismo valor para el número total de Operarios ociosos, sino que aumenta el Lead Time de la línea de ensamblaje. Esto se debe a que el modelo plantea una minimización del número de Operarios Ociosos alargando el Lead Time hacia la ventana de planificación T, y por tanto expandiendo la línea del tiempo. Se llega a la conclusión que para unos operarios disponibles totales $htotalX_i j$ el mejor resultado es el obtenido bajo la minimización del término T_1 ya que el número de operarios totales ociosos siempre será el mismo.

Libre asignación de operarios totales por estación

La posibilidad de tener potestad en la asignación de operarios totales por estación resulta de gran interés en nuestro modelo de optimización. Se pueden dar dos casos: tener una total decisión del número de operarios o bien considerar ya fijos un número de operarios con posibilidad de asignar X operarios adicionales como variables de decisión. Para estudiar el comportamiento de nuestro modelo matemático y realizar el análisis multi-objetivo, tomaremos en cuenta que todos los operarios por estación $htotalX_k$ son variables de decisión en nuestro modelo matemático.

El estudio se realizará bajo un caso práctico de una única estación compuesta por 16 trabajos con unos órdenes de precedencia concretos y para un modelo de avión, de esta manera, podremos enfocar los resultados obtenidos de manera mucho más sencilla al reducir la escala de la línea de ensamblaje en una única estación. Se considerará una ventana de planificación de 10 periodos de igual longitud.

Los datos que conforma la estación para el procesamiento del modelo 1 se presentan en la siguiente tabla:

Tabla 5.3 Datos del problema monoestacional con libre asignación de operarios totales [Fuente: Elaboración propia].

Estación	Trabajo	Precedencia 1	Precedencia 2	Tiempo de proceso (periodos) M1	Operarios requeridos tipo I M1	Operarios requeridos tipo II M1	Operarios requeridos tipo III M1
1	1	0	0	2	0	2	2
	2	0	0	2	2	2	2
	3	1	0	1	2	1	2
	4	2	0	1	2	2	1
	5	2	0	2	2	2	1
	6	4	3	1	1	0	0
	7	4	5	1	0	0	1
	8	5	0	2	0	1	1
	9	6	0	1	1	1	2
	10	7	0	2	1	1	0
	11	8	0	1	1	1	1
	12	9	0	1	3	0	0
	13	9	0	2	3	0	1
	14	12	0	2	3	0	0
	15	10	13	1	3	0	1
	16	12	0	2	0	2	2

Para completar la matriz de pago, realizaremos dos experimentos: minimizaremos primero únicamente la función objetivo 1 que define el Lead Time para ver los resultados computacionales y posteriormente minimizaremos la función objetivo 2 referida al número total de operarios ociosos.

A continuación, se muestran los resultados experimentales obtenidos:

Tabla 5.4 Resultados experimentales del problema monoestacional obtenidos para conformar la matriz de pago [Fuente: Elaboración propia].

Experimento	F.O 1	F.O 2	htotal1	htotal2	htotal3	htotal	hlibre en proceso	hlibre por finalización
1	8	74	7	5	4	16	42	32
2	11	37	4	4	4	12	37	0

Cabe destacar que la columna “hlibre por finalización” devuelve el número total de operarios ociosos por haber finalizado el procesamiento en la estación, es decir, como estamos ante una ventana de planificación de 10 periodos y se han asignado un total de 16 operarios en la estación, existen 2 periodos el cual no se procesa el modelo ya que su Lead Time es 8 periodos, esos dos periodos ociosos corresponden los 32 operarios ociosos por finalización del procesamiento.

La matriz de pago resultante se muestra a continuación:

$$\text{Matriz de pago} = \begin{bmatrix} 8 & 74 \\ 11 & 37 \end{bmatrix}$$

Como hemos estudiado, la matriz de pago nos devuelve dos puntos importantes para su análisis:

- Punto ideal: Diagonal de la matriz de pago, en nuestro caso (8; 37) resultado de la minimización de la función objetivo 1 y 2 simultáneamente quedando fuera de la región de admisibilidad de nuestro problema.
- Punto nadir: Definido por columnas de la matriz de pago como la colección de peores resultados por columna =(11;74).

Una vez constituida la matriz de pago, la exploración del frente de Pareto se realizará mediante el Método de las Restricciones debido a su mayor facilidad y funcionalidad para problemas de gran dimensionalidad. De esta forma, tomaremos como función objetivo la minimización del Lead Time, mientras que el número de operarios ociosos en la línea se añadirá como una restricción adicional a nuestro problema de optimización:

$$\begin{aligned} & \min\ fo1 \\ \text{sujeto a:} \quad & \\ & x,lt \in F \\ & fo2 \leq L_1 \end{aligned}$$

Donde L_1 se define como la cota inferior comprendida entre el punto Nadir y el punto Ideal la cual variará por cada experimento sucesivamente para obtener el Frente de Pareto siempre y cuando se coja una cota que esté dentro de la región de admisibilidad F. A continuación, se muestra codificado en Python la nueva restricción:

```
def rest9_rule (model):
    return model.Operarios_ociosos <= L1
model.rest9 = Constraint ( rule=rest9_rule )
```

Al trabajar con variables enteras y binarias, existen muy pocos puntos que se puedan definir como admisibles en nuestro problema. Completaremos un total de siete experimentos los cuales iremos variando la cota L1 desde el punto nadir = 74 hasta el punto ideal =37. Los resultados computacionales que se muestran hacen referencia al nuevo problema mono-estacional para la exploración del frente de Pareto:

Tabla 5.5 Resultados experimentales del problema monoestacional para el conformado del frente de Pareto [Fuente: Elaboración propia].

Punto	Cota	F.O 1	F.O 2	htotal1	htotal2	htotal3	htotal	hlibre en proceso	hlibre por finalización
A	74	8	74	7	4	4	15	42	32
B	73	9	64	6	4	4	14	49	15
C	63	9	54	6	4	4	14	40	14
D	53	11	47	4	5	4	13	47	0
E	52	11	51	6	4	3	13	51	0
F	51	11	47	4	5	4	13	47	0
G	46	11	37	4	4	4	12	37	0

El anexo A.4 muestra la codificación completa en Python para el nuevo problema descrito, donde los datos se han recogido a partir de los documentos Excel: DatosExcel3.xlsx y TablasDatos3.xlsx.

Posteriormente realizamos el volcado de resultados obtenidos a un gráfico compuesto por dos dimensiones: la función objetivo 1 y la función objetivo 2 para analizar el frente de Pareto de nuestro problema y ver cómo influye la toma de decisión de los operarios disponibles por estación:

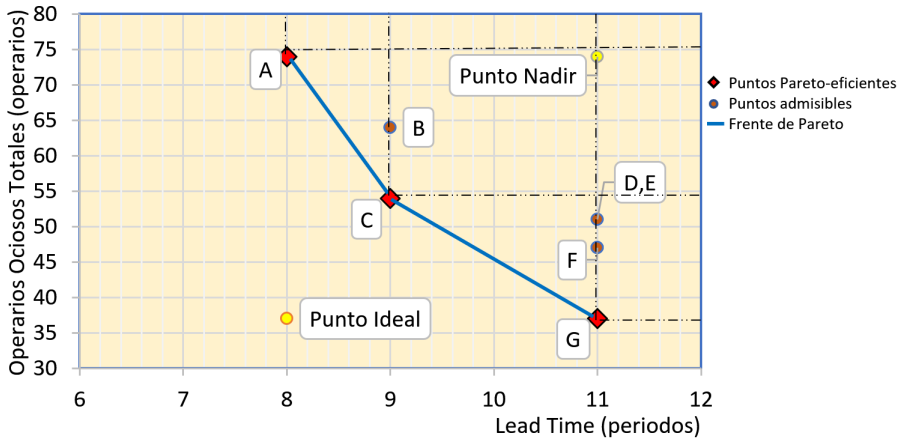

Figura 5.4 Representación gráfica del frente de Pareto [Fuente: Elaboración propia].

Respecto a la dominancia de Pareto, podemos observar como puntos de la región son dominados por otros. Por ejemplo el punto B=(9;64) es dominado por C=(9;54) ya que a mismo Lead Time obtenido de 8 periodos, el punto C obtiene una minimización mayor del número de operarios ociosos totales, esto es: el vector C = (9,54) domina al vector B = (9, 64) ya que para todo $i \in 1,2$ (compuesto por los índices de cada uno de los vectores), se tiene que $C_i \leq B_i$. Del mismo modo, los puntos D=(11, 47), E=(11,47) y F=(11,51) son dominados por el punto G=(11,37) ya que ha mismo Lead Time, el punto G proporciona un menor número de operarios ociosos de la línea.

Analizando los puntos Pareto-eficientes o Pareto-óptimos, obtenemos que las soluciones obtenidas A, C y G son soluciones Pareto-óptimas ya que no existe otras soluciones que dominen a cada una de ellas analizándolas por separado, esto es, no existe ningún vector $v = (v_1, v_2, \ldots, v_n)$ que domine a la solución $u* = (u_1, u_2, \ldots, u_n)$ para cada uno de los puntos Pareto-óptimos. Por tanto, podemos asumir que A, C y G son resultados óptimos ya que no existe otra solución que haga mejorar un objetivo sin que empeore simultáneamente el otro.

Cabe destacar que el Punto Ideal, el cual sería la solución que nuestro cliente querría en caso ideal, resulta una solución que está fuera de la región de admisibilidad.

Por tanto, en caso de decidir entre una solución óptima para nuestro problema de optimización, elegiríamos entre la solución A, C o G, siendo la frontera que conforman estos puntos como la frontera de Pareto. A partir de este punto se procederá a la decisión de que solución interesa teniendo en cuenta la tasa de intercambio entre criterios, es decir, en que proporción aumenta por ejemplo el Lead Time si queremos disminuir el número de operarios ociosos totales:

$$T_{AC} = \frac{9 \text{ periodos - } 8 \text{ periodos}}{74 \text{ operarios - } 54 \text{ operarios}} = 0{,}05 \text{periodos/operario}$$

$$T_{CG} = \frac{11 \text{ periodos - } 9 \text{ periodos}}{54 \text{ operarios - } 37 \text{ operarios}} = 0{,}118 \text{periodos/operario}$$

De esta forma, cuando se disminuye 1 operario ocioso se incrementará el Lead Time de media en 0,05 periodos para el tramo A-C, y 0.118 periodos para el tramo C-G, resultando más interesante el tramo A-C para la toma de decisiones.

5.2.3 Análisis de la eficiencia laboral

De la misma forma que se analizó la eficiencia laboral frente a la variación de requisitos laborales en una línea de ensamblaje para nuestro modelo matemático, resulta interesante hacer el análisis

frente a la variación de operarios totales por estación.

El tiempo de proceso en una estación y por consiguiente el Lead Time de un modelo de avión es afectado por el número de operarios asignados. De la misma forma, una variación de operarios totales provoca una variación del número total de operarios ociosos.

Los gráficos mostrados a continuación, reflejan el estudio entre el Lead Time de una línea de ensamblaje y el número de Operarios Ociosos en proceso frente a la variación de operarios asignados por estación para nuestro modelo matemático:

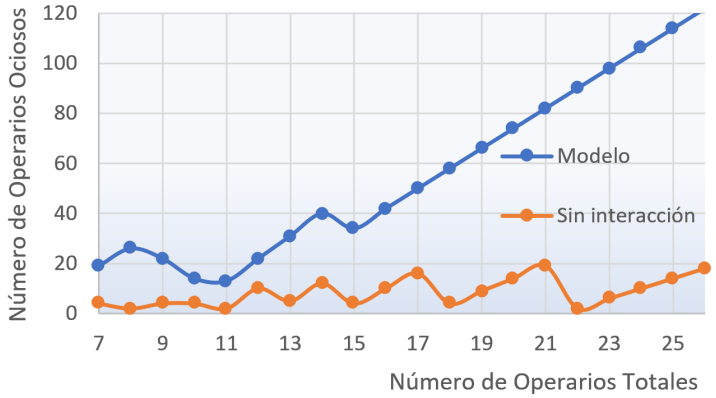

Figura 5.5 Gráfico variación del número operarios ociosos frente al número de operarios totales por estación [Fuente: Elaboración propia].

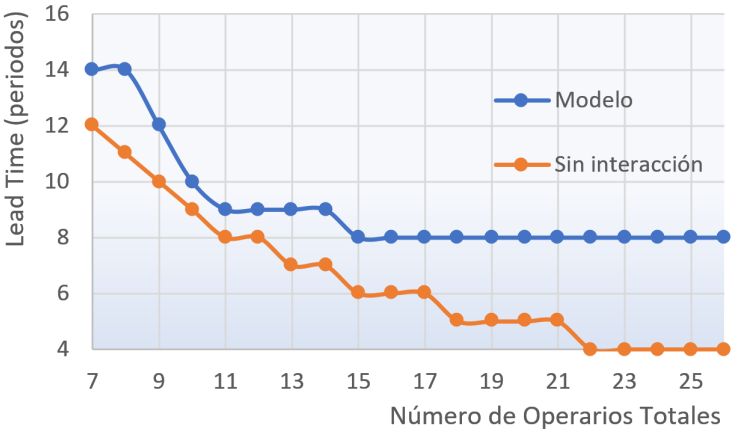

Figura 5.6 Gráfico variación del Lead Time frente al número de operarios totales por estación [Fuente: Elaboración propia].

Los experimentos se han realizados para el mismo problema mono-estacional anteriormente comentado con la singularidad de haber considerado un único tipo de especialidad laboral ya que no interfiere en nuestro estudio y trabajando bajo una ventana de planificación de 15 periodos.

Como podemos ver en las gráficas, se han diferenciado la curva de nuestro modelo, y la curva del modelo sin las interacciones entre operarios provocadas por las precedencias entre trabajos. Para el modelo, existen variaciones entre 7 y 15 operarios totales, esto se debe a las interacciones existen-

tes entre los operarios, las cuales, resultan interesantes para el análisis de la eficiencia en dicho rango.

En caso de no existir interacciones por precedencias laborales, podemos ver unas fluctuaciones constantes debido a los requerimientos laborales de la estación, es decir, un incremento de un operario asignado puede provocar la reducción del Lead Time o en el incremento de operarios ociosos dependiendo de los operarios requeridos de cada trabajo.

Además, podemos analizar en nuestro modelo, cómo a partir de 15 operarios asignados, su incremento devuelve una variación constante ya que cada uno de los trabajos que conforman la estación ya tienen la completitud de sus operarios asignados necesarios, por tanto, un incremento aún mayor no modifica el Lead Time de la línea de ensamblaje, sino que por consiguiente, incrementa de forma proporcional el número de operarios ociosos.

A continuación, se procederá con el estudio de la eficiencia laboral frente al número de operarios asignados por estación de nuestro problema mono-dimensional para el análisis de la respuesta experimental de nuestro modelo.

La tabla 5.6 refleja los resultados obtenidos, así como el cálculo de la eficiencia laboral:

Tabla 5.6 Resultados experimentales obtenidos para el estudio de la eficiencia laboral [Fuente: Elaboración propia].

Experimento	Operarios totales	Lead Time	Operarios Ociosos	Nº Medio Op.ociosos por periodo	Nº Medio Op.requeridos por periodo	Eficiencia Laboral
1	7	14	19	1,357142857	4	0,746666667
2	8	14	26	1,857142857	4	0,682926829
3	9	12	22	1,833333333	4,666666667	0,717948718
4	10	10	14	1,4	5,6	0,8
5	11	9	13	1,444444444	6,222222222	0,811594203
6	12	9	22	2,444444444	6,222222222	0,717948718
7	13	9	31	3,444444444	6,222222222	0,643678161
8	14	9	40	4,444444444	6,222222222	0,583333333
9	15	8	34	4,25	7	0,622222222
10	16	8	42	5,25	7	0,571428571
11	17	8	50	6,25	7	0,528301887
12	18	8	58	7,25	7	0,49122807
13	19	8	66	8,25	7	0,459016393
14	20	8	74	9,25	7	0,430769231
15	21	8	82	10,25	7	0,405797101
16	22	8	90	11,25	7	0,383561644
17	23	8	98	12,25	7	0,363636364
18	24	8	106	13,25	7	0,345679012
19	25	8	114	14,25	7	0,329411765
20	26	8	122	15,25	7	0,314606742

Donde el factor de eficiencia laboral se ha calculado como:

$$\frac{\text{Nº Medio Op.requeridos por periodo}}{\text{Nº Medio Op.requeridos por periodo} + \text{Nº Medio Op.ociosos por periodo}}$$

Obteniendo así la siguiente gráfica para la eficiencia laboral frente al número de operarios totales asignados:

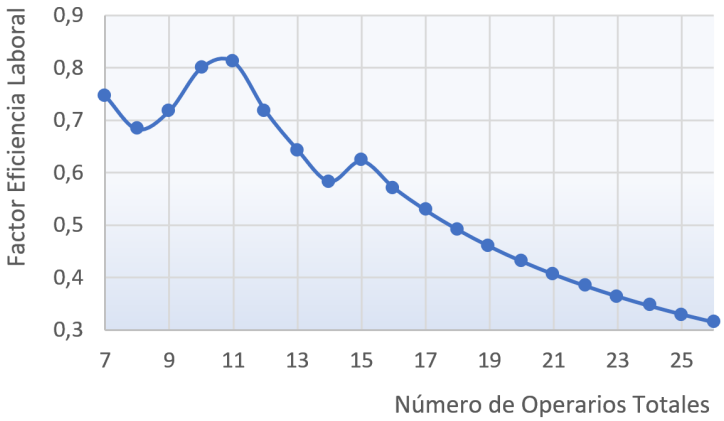

Figura 5.7 Gráfico de la variación de la eficiencia laboral frente al número total de operarios asignados [Fuente: Elaboración propia].

Como era de esperar, a partir de 15 operarios totales asignados, el factor eficiencia laboral cae drásticamente ya que no es posible mayor disminución de Lead Time debido a la interacción de operarios por precedencia y requisitos laborales. La solución con mayor factor de eficiencia laboral se sitúa en 11 y 10 operarios asignados ya que la diferencia entre el número medio de operarios requeridos por periodo y el número medio de operarios ociosos por periodo se aproxima más a los requerimientos laborales del problema.

6 Extensiones al modelo matemático

En este capítulo se introducirá una variante del modelo desarrollado en el capítulo 4 y el modelo que propone [12] para las líneas de ensamblaje Low-Volume Mixtas. Se trata de un modelo que no profundiza a nivel interno en cada una de las estaciones y busca la minimización de costes laborales y de mantenimiento de inventario. El apartado 6.1 describe la situación abstracta de la problemática que aborda el modelo, posteriormente, un ejemplo para comprender su objetivo de detalla en el punto 6.2. La formulación matemática del modelo queda reflejada en el punto 6.3 para finalmente estudiar las aplicaciones prácticas en el último punto del capítulo.

6.1 Introducción y descripción de la problemática

El problema que definiremos a continuación se ajusta ligeramente al anteriormente modelado. En este caso, consideramos una línea de flujo de una línea de ensamblaje con dos líneas dispuestas paralelamente, cada una con igual número de estaciones para terminar culminando en la estación de ensamblado final.

De esta manera, cada línea de ensamblaje se encarga del ensamblado de una estructura concreta del producto a fabricar. A continuación, se muestra la disposición de la línea:

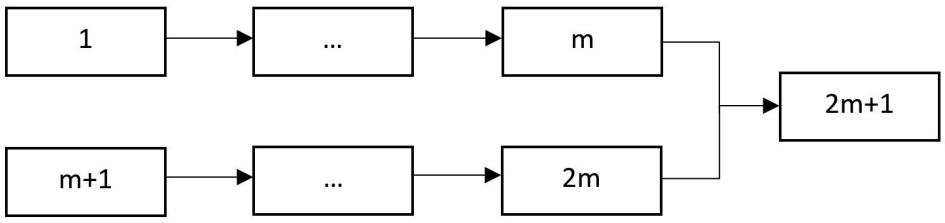

Figura 6.1 Línea de ensamblaje compuesta por 2m+1 estaciones [Fuente: [12]].

Donde las estaciones $1,\ldots,m$ pertenecen a la primera línea, la estaciones $m+1,\ldots,2m$ a la segunda y la estación $2m+1$ se asigna a la estación de ensamblaje global final. Por tanto, m tomará el valor de un parámetro en nuestro modelo que define el número de estaciones en cada una de las líneas paralelas.

A diferencia del problema anterior estudiado, el modelo no detallará un número de trabajos por cada estación y, por consiguiente, sus precedencia. Por tanto, no especificará la estructura interna de cada una de las estaciones, sino que el estudio y análisis se realizará entre estaciones.

Partimos del objeto de estudio del proyecto considerando modelos mixtos en nuestra línea de ensamblaje de manera secuencial, esto es, ningún producto de un modelo concreto puede saltar entre niveles dentro de la línea y por consiguiente el modelo 1 será el primero en ser procesado y n el modelo final.

Respecto a los trabajos, a lo largo de la línea se procesarán un set de n trabajos, perteneciendo cada uno de ellos al trabajo asociado del modelo de producto concreto. Cada una de las estaciones tendrá asignado un único paquete de trabajo, el cual, cada trabajo deberá completar. Por ejemplo, la estación j de la línea de ensamblaje tiene que procesar un paquete de trabajo asignado al trabajo i. Por tanto, un parámetro que se deberá considerar en nuestro modelo será la carga de trabajo causada por el paquete de trabajo j que debe realizar el trabajo i.

La asignación de operarios supone un punto muy importante en este problema al igual que en el anterior modelo. A diferencia del comentado, trataremos con una política de colocación de operarios flexible a lo largo de las estaciones. De esta manera, dado un número máximo de operarios en la línea en cada periodo, se podrán asignar a cada estación teniendo total libertad de movimiento entre una estación y otra.

El tiempo total que un trabajo permanece en una estación lo definiremos como el tiempo de flujo del trabajo i en la estación j. A su vez, se compone por el tiempo de carga y el tiempo de proceso. El tiempo de carga lo conforman todas las actividades relacionadas con set-up, inspección de calidad y mantenimiento. Por otro lado, el tiempo de proceso, el cual, se procesa de manera consecutiva al tiempo de carga, es el tiempo que el producto está siendo realmente procesado a través de la cadena de valor.

La figura siguiente muestra la situación descrita:

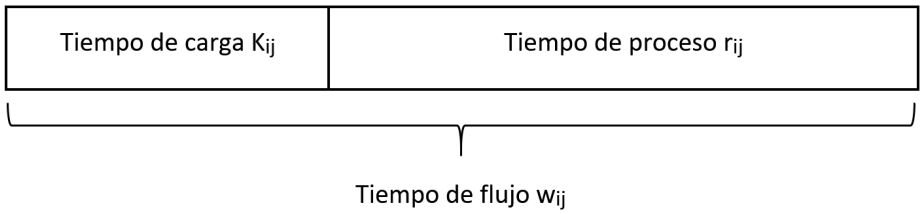

Figura 6.2 Definición del tiempo de flujo de un trabajo i para un modelo j [Fuente: [12]].

Se define el tiempo de ciclo como el tiempo transcurrido entre dos trabajos consecutivos completados en una estación. En este modelo, a diferencia del anterior, el número de operarios asignados depende directamente del tiempo de ciclo en una estación y, por tanto, el tiempo transcurrido desde que se inicia un trabajo i en la estación j hasta su finalización.

Para el procesamiento de un trabajo, el tiempo de carga requiere de operarios para poder ser realizado, no se contempla un número menor o mayor al requerido y, por tanto, su tiempo de

procesamiento no dependerá del número de operarios, sino que se tomará como un parámetro constante en nuestro modelo.

Respecto al tiempo de proceso, dependiendo del trabajo que se esté procesando y la estación donde se esté realizando, cada trabajo deberá completar un paquete de trabajo. Por tanto, el tiempo de proceso no se asigna como un parámetro constante, sino que depende directamente del número de operarios que estén asignados para ese trabajo siempre y cuando no se supere el número máximo de operarios asignados por estación y el número total disponible de operarios en cada periodo. De esta manera, la completitud del tiempo de proceso se conseguirá siempre y cuando el número de operarios asignados multiplicado por las horas regulares disponibles por cada operario y periodo cumplan las horas de carga de trabajo que demanda el paquete de trabajo de la estación j para el trabajo i.

Para el modelado de nuestro problema, se asumirán los siguientes puntos:

1. El modelo trabajará bajo una ventana de planificación de tamaño T dividido en periodos de igual longitud correspondiendo un periodo a un día.
2. No existe la posibilidad de que un operario tenga libertad de movimiento entre las estaciones dentro de cada periodo.
3. Para mayor facilidad de modelado, la carga de trabajo deberá ser múltiplo de las horas regulares laborales por cada periodo y operario, permitiendo así que un trabajo pueda terminar en la estación justo en la finalización del periodo.
4. La carga de trabajo se definirá en horas mientras que el tiempo de carga en periodos.
5. Cada paquete de trabajo solo puede ser asignado únicamente a una estación.
6. Una estación no puede procesar a la vez más de un trabajo.
7. Se contemplará un número máximo total de operarios disponibles por periodo para posterior asignación a lo largo de las estaciones.
8. El modelo contempla un número máximo posible de operarios por cada estación.
9. Se asumirá que todos los trabajadores tienen las mismas cualificaciones para poder realizar cualquier tipo de trabajo a lo largo de las estaciones.
10. Se contemplarán horas-extras de trabajo cuando el tiempo de proceso sea demasiado pequeño para poder completar el correspondiente paquete de trabajo dado una cantidad de operarios asignados.
11. No existirán buffers en ninguna de las estaciones debido a la naturaleza del proyecto.

6.2 Propósito y objetivos del problema

Como ya hemos ido comentando, los costes laborales y de inventario suponen un gran impacto dentro de las líneas de ensamblaje Low-Volume Mixtas debido a las dimensiones de los productos a ensamblar además de la cantidad de operarios necesarios para movilizar y procesar estos mismos. De este modo, nuestro modelo trata de minimizar la suma de estos dos costes para mantener un correcto funcionamiento de la línea bajo un coste optimizado.

A continuación, desglosaremos tanto los costes laborales como los de inventario para poder modelizarlos en su completitud:

1. Costes laborales
    - Costes de operarios ociosos: Coste asociado a la cantidad de operarios que no están procesando un producto.
    - Costes de operarios en horario regular: Coste asociado al número total de operarios en una línea multiplicado por su salario y horario regular.
    - Costes de operarios en horario extra: Coste asociado al número de operarios que trabajan en horario extra multiplicado por el salario extra.
    - Costes de operarios de carga en horario regular: Coste asociado al número total de operarios de carga multiplicado por su salario y horario regular.
2. Costes de mantenimiento de inventario
    - Costes de mantenimiento de inventario entre estaciones adyacentes: Coste asociado al número de periodos sin procesar entre dos estaciones adyacentes multiplicado por el coste de mantenimiento por periodo.
    - Costes de mantenimiento de inventario por estación: Coste asociado al número de periodos que una estación está siendo procesada multiplicado por el coste de mantenimiento por periodo.

Por tanto, a la hora de modelar matemáticamente nuestro problema, estos costes se tratarán de minimizar con nuestra función objetivo. Debido a la diferente naturaleza de ambos costes, estamos de nuevo ante un problema de optimización multi-objetivo el cual se les aplicará los métodos descritos anteriormente para su modelado.

A continuación, se mostrarán los datos para el análisis de un ejemplo que describiremos, evaluando así de forma práctica nuestro problema. Supondremos un total de dos trabajos con dos estaciones consecutivas, en la primera estación se está procesando el trabajo tipo 2, mientras que en la estación 1 se está procesando el trabajo tipo 2 (Nota: supondremos que ambos trabajos se están realizando paralelamente):

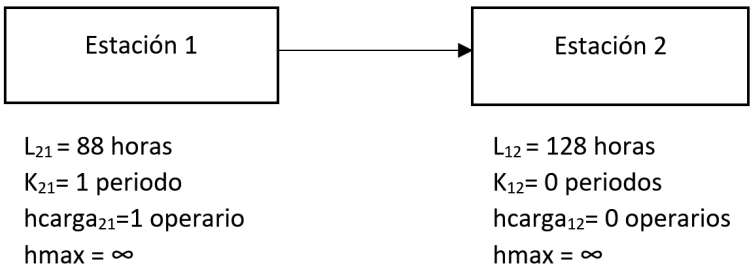

Figura 6.3 Datos requeridos por cada estación [Fuente: Elaboración propia].

Se va a considerar una línea de flujo con 10 operarios disponibles en cada periodo. La carga de trabajo del trabajo 2 en la estación 1 es 88 horas, mientras que para el trabajo 1 en la estación 2 es 128 horas con un horario de trabajo regular de 8 horas. Como se puede ver en los datos, el trabajo 2 requiere un tiempo de carga K_{21} de 1 periodo con 1 operario de carga $hcarga_{21}$.

Posteriormente, hacemos el análisis de nuestros costes para dos casos concretos. Así se evaluará y

se estudiará más fácilmente el propósito de nuestro problema.

Primer caso:

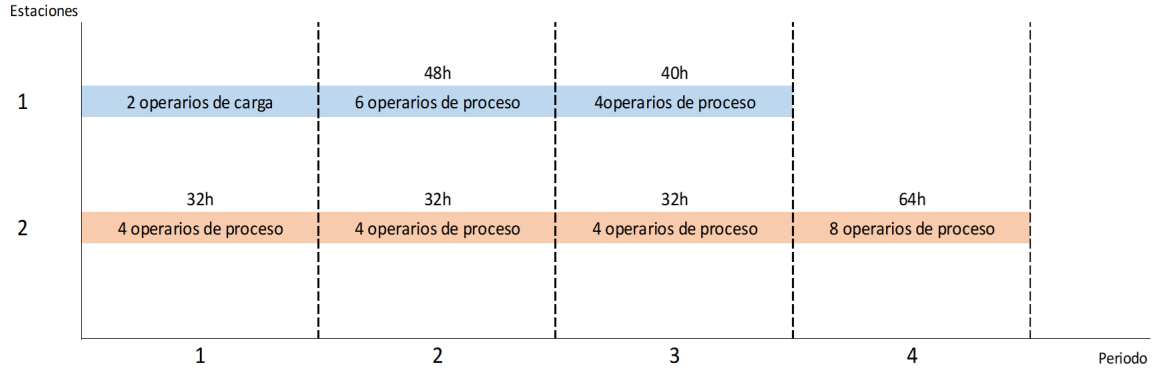

Figura 6.4 Representación del flujo de tiempo para el primer caso [Fuente: Elaboración propia].

Como se puede ver en la línea del tiempo, el tiempo total de flujo en la primera estación es de 3 periodos, mientras que en la segunda estación son 4 periodos. El número total de operarios ociosos es de 8 operarios, y, por tanto, tendrá su coste asociado.

El coste de mantenimiento de inventario por estación se refleja en 3 unidades de periodo para la estación 1 y 4 unidades de periodo para la estación 2.

Segundo caso:

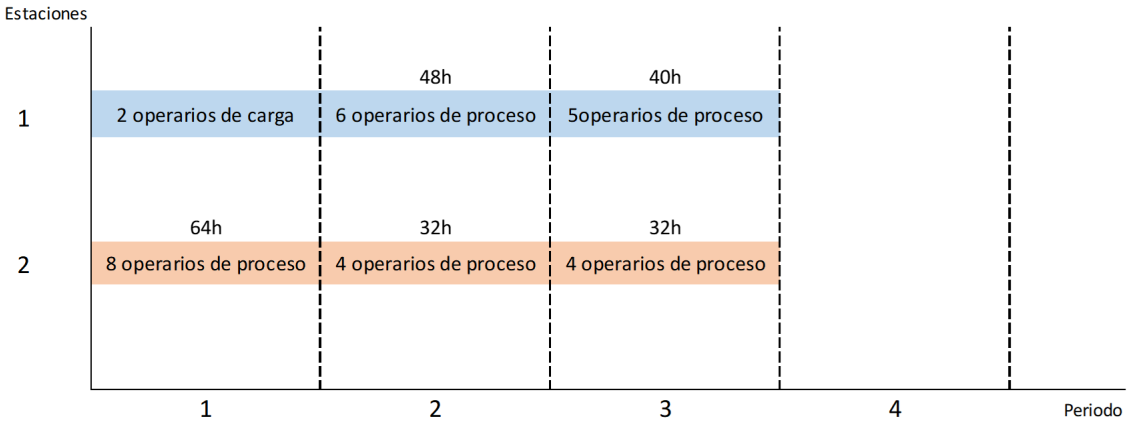

Figura 6.5 Representación del flujo de tiempo para el segundo caso [Fuente: Elaboración propia].

Como se puede ver en la línea del tiempo, el tiempo total de flujo tanto en la primera como en la segunda estación es de 3 periodos completando ambos en el mismo momento la carga de trabajo demandada para cada trabajo. Se contempla un solo operario ocioso a lo largo de la línea de flujo, donde aparece en el periodo 3 y por tanto tendrá su coste asociado.

El coste de mantenimiento de inventario por estación se refleja en 3 unidades de periodo para la estación 1 al igual que para la estación 2.

Es fácil analizar y concluir que el segundo caso proporciona un mejor resultado a nivel de costes, tanto costes laborales como de mantenimiento de inventario ya que se reduce la cantidad de operarios ociosos en 7 operarios y el tiempo de flujo de la estación dos en 1 periodo.

6.3 Formulación matemática

Se supondrá una ventana de planificación de longitud T dividido en periodos de igual dimensión: $t = 1,...,T$. Para mayor simplicidad, abreviaremos como x^+ la función max(x,0) definida como:

$$max(x,0) = \begin{cases} x & \text{si } x > 0 \\ 0 & \text{en otro caso} \end{cases}$$

Se harán uso de los siguientes índices y conjuntos en nuestro problema:

$$\begin{array}{ll} i = 1,....,n & : \text{trabajos} \\ j = 1,....,2m+1 & : \text{estaciones} \\ t = 1,....,T & : \text{periodos} \end{array}$$

Las siguientes variables de decisión serán aplicadas en el modelo:

$$x_{ijt} = \begin{cases} 1 & \text{si el trabajo i empieza en la estación j para el periodo t} \\ 0 & \text{en otro caso} \end{cases}$$

$$y_{ijt} = \begin{cases} 1 & \text{si el trabajo i finaliza en la estación j para el periodo t} \\ 0 & \text{en otro caso} \end{cases}$$

$h_{ijt} =$ nº de operarios asignados al trabajo i en la estación j para el periodo t

El modelo está basado en los siguientes parámetros:

$A =$ Salario regular por operario y hora
$B =$ Salario extra por operario y hora
$C =$ Coste de operarios ociosos por hora
$Y =$ Horario laboral regular por periodo y operario
$ot =$ Máximo tiempo extra en horas por recurso y periodo
$D_j =$ Coste de mantenimiento de inventario en la estación j por periodo
$K_{ij} =$ Tiempo de carga (en periodos)
$L_{ij} =$ Carga de trabajo para el trabajo i en la estación j

$hmax_t =$ Número máximo de operarios disponibles por periodo
$hmaxest_j =$ Máximo número posible de operarios asignados a la estación j en un periodo
$hcarga_{ij} =$ Número de operarios requeridos para la carga del trabajo i en la estación j

El resultante modelo matemático queda formulado a continuación:

$$\min C_1 + C_2$$

donde:

$$C_1 = \sum_{i=1}^{n}\sum_{j=1}^{2m+1}(AY\sum_{t=1}^{T}h_{ijt} + B(L_{ij} - Y\sum_{t=1}^{T}h_{ijt})^{+} + AY\sum_{t=1}^{T}hcarga_{ij}(\sum_{k=t-K_{ij}+1}^{t}x_{ijk})) +$$

$$CY(\sum_{t=1}^{T}(hmax_t - \sum_{i=1}^{n}\sum_{j=1}^{2m+1}(h_{ijt} + hcarga_{ij}(\sum_{k=t-K_{ij}+1}^{t}x_{ijk}))))$$

$$C_2 = \sum_{i=1}^{n}(\sum_{j\notin m,2m+1}D_j(\sum_{t=1}^{T}(tx_{i,j+1,t} - ty_{ijt}) - 1) + D_m(\sum_{t=1}^{T}(tx_{i,2m+1,t} - ty_{imt}) - 1) +$$

$$\sum_{i=1}^{n}\sum_{j=1}^{2m+1}D_j(\sum_{t=1}^{T}(ty_{ijt} - tx_{ijt}) + 1)$$

sujeto a:

$$\sum_{t=1}^{T}x_{ijt} = 1, \qquad i = 1,...,n,\ j = 1,...,2m+1 \tag{1}$$

$$\sum_{t=1}^{T}y_{ijt} = 1, \qquad i = 1,...,n,\ j = 1,...,2m+1 \tag{2}$$

$$\sum_{t=1}^{T}tx_{ijt} + K_{ij} \le \sum_{t=1}^{T}ty_{ijt}, \qquad i = 1,...,n,\ j = 1,...,2m+1 \tag{3}$$

$$\sum_{t=1}^{T}ty_{ijt} + 1 \le \sum_{t=1}^{T}tx_{i,j+1,t}, \qquad i = 1,...,n,\ j = 1,...,2m,\ j \neq m \tag{4}$$

$$\sum_{t=1}^{T}ty_{imt} + 1 \le \sum_{t=1}^{T}tx_{i,2m+1,t}, \qquad i = 1,...,n \tag{5}$$

$$\sum_{t=1}^{T}ty_{i-1,j,t} + 1 \le \sum_{t=1}^{T}tx_{ijt}, \qquad i = 2,...,n,\ j = 1,...,2m+1 \tag{6}$$

$$\sum_{t=1}^{T}tx_{i+1,j,t} \ge \sum_{t=1}^{T}tx_{i,j+1,t}, \qquad i = 1,...,n-1,\ j = 1,...,2m,\ j \neq m \tag{7}$$

$$\sum_{t=1}^{T}tx_{i+1,m,t} \ge \sum_{t=1}^{T}tx_{i,2m+1,t}, \qquad i = 1,...,n-1 \tag{8}$$

$$h_{ijt} \le hmaxest_j(\sum_{k=1}^{t-K_{ij}}x_{ijk} - \sum_{k=1}^{t-1}y_{ijk}) \qquad i = 1,...,n,\ j = 1,...,2m+1,\ t = 1,...,T \tag{9}$$

$$\sum_{j=1}^{2m+1}\sum_{i=1}^{n}(h_{ijt} + (hcarga_{ij}(\sum_{k=t-K_{ij}+1}^{t}x_{ijk}))) \le hmax_t \qquad t = 1,...,T \tag{10}$$

$$\sum_{t=1}^{T}(Y + ot)h_{ijt} \ge L_{ij} \qquad i = 1,...,n,\ j = 1,...,2m+1 \tag{11}$$

$$x_{ijt}, y_{ijt} \in [0,1] \qquad h_{ijt} \in \mathbb{Z} \tag{12}$$

Función objetivo

La minimización de la función objetivo $C_1 + C_2$ es dada como la suma de los costes laborales y los costes de mantenimiento de inventario totales. Atribuyéndonos al primer término del sumando C_1, éste contiene los costes generados por el trabajo laboral regular mientras que el segundo refleja los costes laborales de horas extra. El tercer término representa los costes de los operarios de carga en horario regular. Por último, el cuarto y último término minimiza los costes generados por el sumatorio total de los operarios ociosos en cada periodo.
A continuación, se detallará de forma más desglosada cada uno de los términos que componen el sumando C_1:

Primer término: $\sum_{i=1}^{n}\sum_{j=1}^{2m+1}AY\sum_{t=1}^{T}h_{ijt}$

Para cada uno de los n trabajos existentes en la línea de ensamblado y dentro de cada estación j, se busca minimizar el coste ocasionado por el número total de operarios h_{ijt} asignados durante los tiempos de procesado r_{ij}.

Segundo término: $\sum_{i=1}^{n}\sum_{j=1}^{2m+1} B(L_{ij} - Y\sum_{t=1}^{T} h_{ijt})^{+}$

La función $max(x,0)$ posibilita dos salidas ante el segundo término. En el caso de que existan suficientes operarios asignados que satisfagan la carga de trabajo L_{ij} que demanda una estación j para un trabajo i, la ecuación $L_{ij} - Y\sum_{t=1}^{T} h_{ijt}$ sería menor o igual a cero, devolviendo por tanto ningún coste asociado a trabajos laborales de tiempo extra. Sin embargo, si los operarios asignados no satisfacen dicha carga de trabajo, la resta resultaría positiva y por tanto existiría un coste laboral de horas extra asociado.

Tercer término: $\sum_{i=1}^{n}\sum_{j=1}^{2m+1} AY\sum_{t=1}^{T} hcarga_{ij}(\sum_{k=t-K_{ij}+1}^{t} x_{ijk})$

De la misma manera en la que hemos representado el primer término, para cada uno de los n trabajos existentes dentro de la línea y en cada estación j, se busca minimizar el coste ocasionado por el número total de operarios de carga, el cual queda definido más adelante en la restricción (10) de nuestro modelo.

Cuarto término: $CY(\sum_{t=1}^{T}(hmax_t - \sum_{i=1}^{n}\sum_{j=1}^{2m+1}(h_{ijt} + hcarga_{ij}(\sum_{k=t-K_{ij}+1}^{t} x_{ijk}))))$

Debido al parámetro $hmax_t$ definido anteriormente como el número máximo de operarios disponibles por periodo, es fácil modelar que dicho término menos el sumatorio de todos los operarios asignados de cada estación de la línea de ensamblaje (tanto operarios de procesado como operarios de carga) para ese mismo periodo, nos devuelve los operarios ociosos que no están atribuidos a ningún trabajo dentro de la línea. El coste asociado por dichos operarios es generado a lo largo de la línea del tiempo estudiada T.

Atribuyéndonos al primer y segundo término del sumando C_2, se han modelado los costes de mantenimiento de inventario entre estaciones adyacentes, tanto para las estaciones pertenecientes a las líneas paralelas, como para la estación unión de ensamblaje final $2m+1$. El tercer término refleja los costes de inventario dentro de una estación j en la que se procesa un trabajo i.
A continuación, se detallará de forma más desglosada cada uno de los términos que componen el sumando C_2:

Primer término: $\sum_{i=1}^{n}(\sum_{j\notin m,2m+1} D_j(\sum_{t=1}^{T}(tx_{i,j+1,t} - ty_{ijt}) - 1)$

Para todos los trabajos que conforman la línea de ensamblaje y para las estaciones que conforman las líneas paralelas excepto la estación m, el primer término busca la minimización de los costes de inventario resultantes por la espera (en periodos) del trabajo de una estación para su posterior paso a la siguiente estación consecutiva.

Segundo término: $\sum_{i=1}^{n} D_m(\sum_{t=1}^{T}(tx_{i,2m,t} - ty_{imt}) - 1)$

Puesto que nuestro problema involucra una estación de ensamblado final la cual es receptora de dos líneas paralelas e independientes, de igual forma se ha tratado con el primer término, en el segundo se refleja la minimización de costes de inventario generados por la espera (en periodos) del trabajo de la estación m para su posterior procesamiento en la estación de ensamblado final $2m+1$.

Tercer término: $\sum_{i=1}^{n}\sum_{j=1}^{2m+1} D_j(\sum_{t=1}^{T}(ty_{ijt} - txijt)+1)$

Por último, el tercer término del sumando C_2 minimiza los costes de de mantenimiento de inventario dentro de cada estación para todos los n trabajos y estaciones j que conforman la línea de ensamblaje. De esta manera, el término $\sum_{t=1}^{T}(ty_{ijt} - txijt$ devuelve el número de periodos el cual un trabajo es procesado dentro de una estación, atribuyéndose por tanto, sus costes asociados.

Restricciones

La restricción (1) obliga a que cada trabajo solo puede empezar en un solo periodo para una estación concreta. De la misma forma, la restricción (2) obliga a que cada trabajo solo puede finalizar en un solo periodo para una estación concreta.
La restricción (3) determina que el tiempo de carga de un trabajo en una estación precede al tiempo de proceso, a la vez que asegura que el tiempo de flujo es siempre mayor que el tiempo de carga. Nótese que un trabajo puede empezar y acabar en un mismo periodo dentro de una estación.
La restricción (4) refleja que un trabajo dentro de una estación no puede empezar hasta que ese mismo trabajo no haya sido completado en la estación anterior, haciendo un flujo de valor aguas arriba en la línea de ensamblaje. Debido a la existencia de una estación de ensamblado final, la restricción (5) asegura que el trabajo dentro de la última estación no puede empezar hasta que además se haya finalizado el trabajo en la estación m.
La restricción (6) refleja la precedencia entre trabajos, esto es, un trabajo no puede iniciar su procesado en una estación hasta que no lo haya realizado en la estación predecesora.
La restricción (7) representa la simultaneidad de iniciación entre estaciones, asegurando que un trabajo i+1 no puede empezar en la estación j si el trabajo i no se ha iniciado en la estación j+1. De la misma forma, queda representado para la estación de ensamblado final por la restricción (8).

Debido al mayor volumen de formulación matemática que representan las restricciones (9) y (10), explicaremos de forma más detallada sus funciones:

Restricción (9): $h_i jt \leq hmaxest_j(\sum_{k=1}^{t-K_{ij}} x_{ijk} - \sum_{k=1}^{t-1} y_{ijk}) \quad i = 1,...,n,\ j = 1,...,2m+1,\ t = 1,...,T$

Restricción que sostiene la limitación del número de operarios asignados por estación y periodo. Para que dicho número de operarios no supere el límite $hmaxest_j$, debe cumplirse que estemos ante un periodo que conforme el tiempo de proceso r_{ij}, para ello, se hace uso de la siguiente formulación que depende tanto de la variable x_{ijt} como de la variable y_{ijt}:

$$\sum_{k=1}^{t-K_{ij}} x_{ijk} - \sum_{k=1}^{t-1} y_{ijk}$$

De esta manera, atendiendo al trabajo que se esté procesando, en la estación donde se esté procesando y el periodo en el que transcurra, dicha formulación devuelve 1 en caso de pertenecer a un periodo contenido en el tiempo de proceso del trabajo i en la estación j y 0 en cualquier otro caso. Queda asegurado por tanto que h_{ijt} vale 0 si no nos encontramos en ningún periodo perteneciente al tiempo de proceso r_{ij}.

Restricción (10): $\sum_{j=1}^{2m+1}\sum_{i=1}^{n}(h_{ijt} + (hcarga_{ij}(\sum_{k=t-K_{ij}+1}^{t} x_{ijk}))) \leq hmax_t \qquad t = 1,...,T$

Como hemos comentando anteriormente, el parámetro $hmax_t$ refleja el número máximo de operarios disponibles en cada periodo para posterior asignación a las diferentes estaciones que conforman la línea de ensamblaje. Por tanto, la suma de operarios de procesado h_{ijt} y de operarios de carga debe ser inferior a los operarios disponibles en el periodo t.

Para que se asignen operarios de carga debe darse obligatoriamente el caso de que el periodo que se esté analizando lo conforme el tiempo de carga de un trabajo i en una estación dada j. Para modelar dicho afirmación, se hace uso de la siguiente formulación matemática que depende de x_{ijt}:

$$\sum_{k=t-K_{ij}+1}^{t} x_{ijk}$$

La cual devuelve 1 para un periodo si el trabajo i se está procesando dentro del tiempo de carga K_{ij} para una estación j dada y 0 en caso contrario.
De esta manera, podemos definir la siguiente fórmula como el número de operarios de carga del trabajo i dentro de la estación j para el periodo t.

$$hcarga_{ij}(\sum_{k=t-K_{ij}+1}^{t} x_{ijk})$$

La completitud de un trabajo dentro de una estación queda reflejada en la restricción (11), el cual, bajo un número de operarios asignados, la suma de sus horas regulares de trabajo y sus horas extras deben satisfacer la carga de trabajo que demande una estación para un trabajo i.

Por último, la restricción (12) asegura que las variables de decisión x_{ijt}, e y_{ijt} sean binarias y la variable h_{ijt} una variable entera.

6.4 Aplicaciones prácticas

Para comprobar y poner en marcha la eficacia del modelo adicional descrito, realizaremos un pequeño ejemplo práctico compuesto por dos estaciones consecutivas dispuestas en serie. La línea de ensamblaje trabajará con dos modelos diferentes, y, por tanto, cada estación deberá completar la carga de trabajo correspondiente a su trabajo. Se trabajará bajo una ventana de planificación de 7 periodos de igual longitud. La tabla que se muestra a continuación muestra los datos requeridos por cada estación:

Tabla 6.1 Datos ejemplo práctico por cada estación [Fuente: Elaboración propia].

Estación	Operarios max por estación	Trabajo	Carga de trabajo (horas)	Tiempo de carga (periodos)	Operarios de carga requeridos
1	6	1	88	1	2
		2	40	1	2
2	8	1	128	0	0
		2	32	0	0

Analizando los datos que componen la tabla, la estación número uno es la única que requiere de tiempo de carga, y, por consiguiente, operarios especializados en ello. Los operarios totales disponibles por periodo quedan definidos en la tabla siguiente:

Tabla 6.2 Operarios disponibles por periodo [Fuente: Elaboración propia].

Periodo	1	2	3	4	5	6	7
Operarios disponibles	2	6	7	14	9	6	5

Respecto a las condiciones laborales de la línea, cada operario trabaja 8 horas regulares por periodo ($Y = 8 horas/periodo * operario$) con un salario de 12 u.m la hora ($A = 8 u.m/hora * operario$). Para simplificar la aplicación práctica, asumiremos que no hay horas extras de trabajo, o lo que es lo mismo, un operario no puede recibir salario extra por cada hora extra que trabaja.

En caso de considerar los costes asociados a las horas extras de trabajo, el término $min \sum_{i=1}^{n} \sum_{j=1}^{2m+1} B(L_{ij} - Y \sum_{t=1}^{T} h_{ijt})^+$ se traduce en un término no lineal mediante la incorporación de 4 nuevas variables:

$$\begin{aligned} & \min B(z1_{ij} + z2_{ij}) \\ \text{sujeto a:} & \\ & z1_{ij} \geq (L_{ij} - Y \textstyle\sum_{t=1}^{T} h_{ijt}) c1_{ij} \\ & z2_{ij} \geq (L_{ij} - Y \textstyle\sum_{t=1}^{T} h_{ijt}) c2_{ij} \\ & c1_{ij} + c2_{ij} = 1 \\ & \\ & z1_{ij}, z2_{ij} \geq 0 \\ & z1_{ij}, z2_{ij} \in \mathbb{Z} \\ & c1_{ij}, c2_{ij} \in [0,1] \end{aligned}$$

Donde las variables binarias $c1_{ij}$ y $c2_{ij}$ inducen a que el valor $L_{ij} - Y \sum_{t=1}^{T} h_{ijt}$ valga 0 en caso de ser negativo o que valga $L_{ij} - Y \sum_{t=1}^{T} h_{ijt}$ si devuelve un número positivo.

Por último, la estación 1 tiene asignado un coste de mantenimiento de inventario de 300 u.m mientras que la estación 2 de 400 u.m. ($D_1 = 30 u.m$, $D_2 = 40 u.m$).

Debido a que nuestro ejemplo práctico no dispone de líneas paralelas y una estación de ensamblaje final, es fácil concluir que las restricciones (5) y (8) impuestas en el modelo matemático no serán usadas para la resolución del caso.

Una vez tenemos los datos necesarios, procederemos con el modelado computacional en Python. Al no trabajar el modelo adicional con familias de conjuntos indexadas, el modelado resulta más sencillo a nivel de codificación que el modelo estudiado a lo largo del proyecto. El anexo A.5 contiene el modelado completo.

De la misma forma, procederemos con la llamada al Solver para resolver el modelo adicional, un modelo de programación lineal entera. A continuación, se muestran los resultados computacionales obtenidos:

Tabla 6.3 Resultados experimentales ejemplo práctico [Fuente: Elaboración propia].

x[i,j,t]	Valor
x[1,1,1]	1
x[1,2,4]	1
x[2,1,4]	1
x[2,2,7]	1

y[i,j,t]	Valor
y[1,1,3]	1
y[1,2,5]	1
y[2,1,6]	1
y[2,2,7]	1

h[i,j,t]	Valor
h[1,1,2]	6
h[1,1,3]	6
h[1,2,4]	8
h[1,2,5]	8
h[2,1,5]	1
h[2,1,6]	6
h[2,2,7]	5

F.O	12824
T1	5024
T2	7800

Volcando los datos a un diagrama de Gantt:

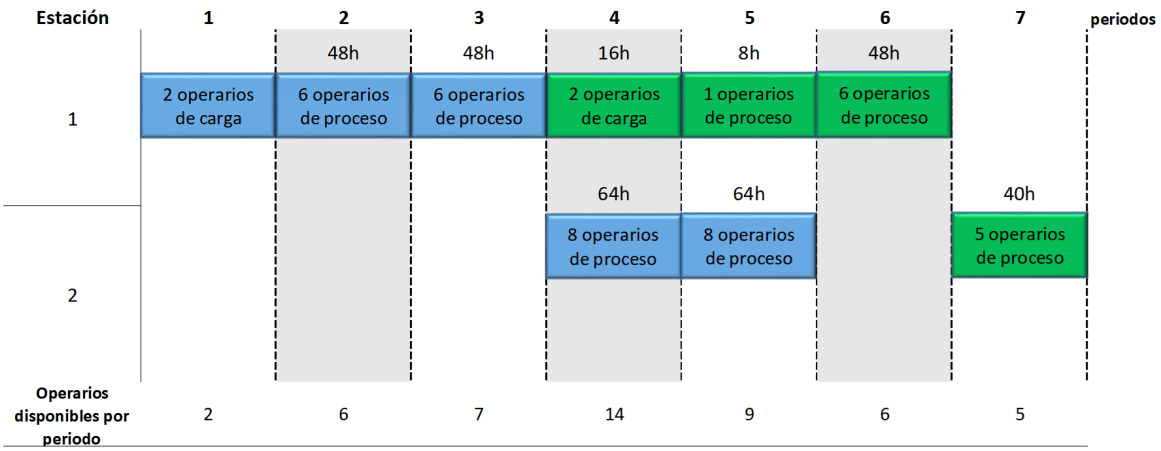

Figura 6.6 Diagrama de Gantt ejemplo práctico [Fuente: Elaboración propia].

Analizando la línea del tiempo, donde el color azul se asigna al modelo 1 y el color verde al modelo 2, podemos ver como se cumplen los órdenes de precedencia entre trabajos al igual que se cumplen entre estaciones. De la misma forma, no se superan los operarios disponibles por periodo además de los operarios máximos que pueden estar asignados en una estación.
En cada periodo, se puede apreciar las cargas de trabajos realizadas, las cuales dependen del número de operarios asignados. Haciendo un sumatorio para cada modelo, se puede concluir que se las cargas de trabajo que demandan cada trabajo para cada estación se satisfacen vistos los resultados obtenidos.

Respecto a los términos de la función objetivo, el término T_1 definido como el coste laboral resulta ser de 5024 u.m mientras que el término T_2 de 7800 u.m. Desglosando el término T_1 se puede concluir que además del coste que genera la mano de obra regular, en nuestro ejemplo podemos ver como a lo largo de la ventana de planificación existen 5 operarios ociosos, traduciéndose en un 17 % del coste laboral total.

En resumen, realizando este pequeño ejemplo, podemos ver como el modelo adicional presentado en el capítulo tiene un gran potencial en aras de minimizar los costes laborales y los costes de mantenimiento que surgen a lo largo de una línea de ensamblaje Low-Volume Mixta.

7 Conclusiones

A modo de conclusión, vamos a describir los puntos más importantes que componen el proyecto para terminar con el análisis y valoraciones finales de los resultados obtenidos.

- Se ha realizado el estudio de la problemática de planificación de producción de las líneas de ensamblaje Low-Volume Mixtas, identificando y evaluando previamente el comportamiento, así como los distintos parámetros y variables que las involucran. Además de ello, se han identificado y analizado las respuestas que presentan mediante análisis de requisitos laborales, análisis multi-objetivo, así como análisis en la eficiencia laboral de la línea a través de variaciones de distintos parámetros y variables.

- Se ha planteado el problema de planificación de producción en dichas líneas para así desarrollar un modelo de programación lineal que minimiza los costes asociados a mantenimiento de inventario debido a la reducción del Lead Time, y los costes laborales provocados por el número de operarios ociosos generados a lo largo de la línea. Posteriormente, se ha aplicado dicho modelo a un problema real como caso práctico del sector aeronáutico, caracterizado por la producción en este tipo de líneas.

- El problema real se ha modelado computacionalmente en un lenguaje Python para aprovechar el potencial de este lenguaje de programación tomando distintos parámetros y variables existentes en nuestro caso práctico. Se ha conseguido traducir las restricciones de precedencia y requisitos laborales en matrices capaces de comportarse como familias de conjuntos indexadas.

- Se ha podido comprobar el correcto funcionamiento del modelo de programación lineal que hemos desarrollado obteniendo el mínimo Lead Time y número de operarios ociosos que se puede obtener para nuestro caso práctico dadas unas restricciones de precedencia entre trabajos, modelos y estaciones, además de los requisitos laborales diferenciados por especialización que demanda cada uno de los trabajos y teniendo en cuenta un número de operarios disponibles por estación fijado.

- Debido a las múltiples aplicaciones que tiene, se ha desarrollado una variante del modelo estudiado a lo largo del proyecto junto con otro modelo presentado por AIRBUS Group Innovations y se ha realizado un pequeño ejemplo práctico para comprobar su aplicabilidad y potencial frente a la minimización de costes totales laborales y de mantenimiento de inventario generados a lo largo de una línea de ensamblaje Low-Volume Mixta.

Respecto a las valoraciones finales de los resultados obtenidos fruto de la aplicación de nuestro modelo de programación lineal al caso práctico real del sector aeronáutico, podemos concluir el fuerte impacto que tiene un desequilibrio de las cargas de trabajo entre distintos modelos que confluyen a lo largo de la cadena de valor. Este desequilibrio provoca un aumento del número de

operarios ociosos debido a la necesidad de asignar un mayor número de operarios a los modelos que requieren de más carga de trabajo. Además de ello, permitiendo una libre asignación de operarios disponibles por estación, se ha podido comprobar cómo influye el factor de eficiencia laboral frente a la variación de los mismos, donde, dependiendo de cada problema, se obtienen puntos más eficientes que otros.

Apéndice A
Codifiación en Python

A.1 Ejemplo multi-objetivo: Método de las restricciones

```
import pyomo.environ
from pyomo.core import *
from pyomo.opt import SolverFactory

model=ConcreteModel()

model.cota = Param( initialize =800000)

model.x_1 = Var(within=NonNegativeIntegers)
model.x_2 = Var(within=NonNegativeIntegers)

model.f1 = Expression(expr=1000*model.x_1+3000*model.x_2)
model.f2 = Expression(expr=model.x_1+2*model.x_2)

model.obj=Objective(expr=model.f2, sense=minimize)

model.a = Constraint (expr=model.x_1 <= 300)
model.b = Constraint (expr=model.x_2 <= 200)
model.c = Constraint (expr=model.x_1 + model.x_2 <= 400)
model.d = Constraint (expr=1000*model.x_1 + 3000*model.x_2 >= 300000)
model.e = Constraint (expr=model.f1 >= model.cota)

opt = SolverFactory("glpk")

results = opt.solve(model, tee=True)
model.solutions.load_from( results )

def pyomo_postprocess(options=None, instance=None, results=None):
    model.x_1.display(), model.obj.display(), model.x_2.display()

print(model.x_1.value,model.x_2.value, value(model.f1), value(model.f2))
```

A.2 Ejemplo introducción a Pyomo

```
import pyomo.environ
from pyomo.core import *

model = ConcreteModel()

model.N = Set( initialize = [1,2])
model.M = Set( initialize = [1,2])
model.c = Param(model.N, initialize ={1:1, 2:3})
model.a = Param(model.N, model.M, initialize ={(1,1):3, (2,1):4, (1,2):2, (2,2):5})
model.b = Param(model.M, initialize ={1:1, 2:2})

model.x = Var(model.N, within=NonNegativeReals)

def obj_rule(model):
    return sum(model.c[i]*model.x[i] for i in model.N)
model.obj = Objective(rule=obj_rule)

def con_rule(model, m):
    return sum(model.a[i,m]*model.x[i] for i in model.N) >= model.b[m]
model.con = Constraint(model.M, rule=con_rule)
```

A.3 Caso práctico

```
import pyomo.environ
from pyomo.core import *
from pyomo.opt import SolverFactory
import openpyxl
from openpyxl import load_workbook


wb = openpyxl.load_workbook("C:\\Users\\UX430U\\CursoPython\\DatosExcel2.xlsx")
hoja = wb['Hoja1']

numero_trabajos = hoja.max_row - 1

datos_precedencia = []
for i in range(numero_trabajos):
    datos_precedencia.append([])
    for j in range(2):
        datos_precedencia[i].append(0)

for i in range(numero_trabajos):
    datos_precedencia[i][0] = i+1

for fila in range(2, numero_trabajos+2):
    for columna in range(6, 8):
```

```
            if (hoja.cell(row=fila,column=columna).value) != 0:
                if columna == 6:
                    datos_precedencia[fila-2][1] = hoja.cell(row=fila,column=columna).value
                elif columna == 7:
                    v=[]
                    v.append(hoja.cell(row=fila,column=columna-1).value)
                    v.append(hoja.cell(row=fila,column=columna).value)
                    datos_precedencia[fila-2][1] = v
                elif columna > 7:
                    v.append(hoja.cell(row=fila,column=columna).value)
                    datos_precedencia[fila-2][1] = v


M_precedencia = []
for i in range(numero_trabajos):
    M_precedencia.append([])
    for j in range(numero_trabajos):
        M_precedencia[i].append(0)

for valor in range(numero_trabajos):
    x=datos_precedencia[valor][0]
    y=datos_precedencia[valor][1]
    if type(y) == int:
        if y>0:
            M_precedencia[x-1][y-1] = 1
    else:
        tamaño_vconverge = len(y)
        for numero in range(tamaño_vconverge):
            M_precedencia[x-1][y[numero]-1] = 1

hoja2 = wb['Hoja2']

M_estructura =[]
numero_filas = hoja2.max_row
linea_final = hoja2.cell(row=numero_filas,column=1).value

for s in range(0, linea_final):
    M_estructura.append([])

for fila in range(2, numero_filas):
    if hoja2.cell(row=fila,column=1).value == 1:
        estaciones_linea1 = hoja2.cell(row=fila,column=2).value
    if hoja2.cell(row=fila,column=1).value == 2:
        estaciones_linea2 = hoja2.cell(row=fila,column=2).value
    if hoja2.cell(row=fila,column=1).value == 3:
        estaciones_linea3 = hoja2.cell(row=fila,column=2).value
    if hoja2.cell(row=fila,column=1).value == 4:
        estaciones_linea4 = hoja2.cell(row=fila,column=2).value

for s in range(0, linea_final):
```

```
    if s==0:
        for m in range(0, estaciones_linea1 ):
            M_estructura[s].append([])
    elif s==1:
        for m in range(0, estaciones_linea2 ):
            M_estructura[s].append([])
    elif s==2:
        for m in range(0, estaciones_linea3 ):
            M_estructura[s].append([])
    elif s==3:
        for m in range(0, estaciones_linea4 ):
            M_estructura[s].append([])

for fila in range(2, numero_filas+1):
    M_estructura[hoja2.cell(row=fila,column=1).value -1][hoja2.cell(row=fila,\
    column=2).value - 1].append(hoja2.cell(row=fila,column=3).value)

M_estaciones = []
fila_ultima = hoja.max_row
numero_estaciones = hoja.cell(row=fila_ultima,column=1).value
for s in range(0,numero_estaciones):
    M_estaciones.append([])
for fila in range(2, fila_ultima+1):
    M_estaciones[hoja.cell(row=fila,column=1).value - 1].append(hoja.cell(row=fila,\
    column=5).value)

wb2 = openpyxl.load_workbook("C:\\Users\\UX430U\\CursoPython\\TablasDatos2.xlsx")
tabla = wb2['Hoja1']

precedencia_trabajos = dict()
for i in range(numero_trabajos):
    precedencia_trabajos[i+1] = []
    for j in range(numero_trabajos):
        if M_precedencia[i][j] == 1:
            precedencia_trabajos[i+1].append(j+1)

precedencia_estaciones = dict()
for i in range(len(M_precedencia)):
    precedencia_estaciones[i+1] = []

for s in range(len(M_estructura)):
    if s == len(M_estructura) - 1:
        for m in range(len(M_estructura[s])):
            if m==0:
                for n in range(len(M_estructura[s][m])):
                    for q in range(len(M_estructura)-1):
                        for r in range(len(M_estructura[q][-1])):
                            precedencia_estaciones[M_estructura[s][m][n]]\
                            .append(M_estructura[q][-1][r])
            else:
```

```
                for n in range(len(M_estructura[s][m])):
                    for p in range(len(M_estructura[s][m-1])):
                        precedencia_estaciones[M_estructura[s][m][n]].\
                        append(M_estructura[s][m-1][p])
    else:
        for m in range(len(M_estructura[s])):
            if m == 0:
                for n in range(len(M_estructura[s][m])):
                    precedencia_estaciones[M_estructura[s][m][n]] = []
            else:
                for n in range(len(M_estructura[s][m])):
                    for p in range(len(M_estructura[s][m-1])):
                        precedencia_estaciones[M_estructura[s][m][n]].\
                        append(M_estructura[s][m-1][p])

precedencia_modelos = dict()
for i in range(numero_trabajos):
    precedencia_modelos[i+1] = []
for s in range(len(M_estaciones)):
    for m in range(len(M_estaciones[s])):
        for n in range(len(M_estaciones[s])):
            precedencia_modelos[M_estaciones[s][m]].append(M_estaciones[s][n])

trabajos_por_estacion = dict()
for i in range(len(M_estaciones)):
    trabajos_por_estacion[i+1]= []
for s in range(len(M_estaciones)):
    for m in range(len(M_estaciones[s])):
            trabajos_por_estacion[s+1].append(M_estaciones[s][m])

model = ConcreteModel()

model.TRABAJOS = RangeSet(1,len(M_precedencia))
model.PERIODOS = RangeSet(1,22)
model.MODELOS = set([1, 2])
model.ESTACIONES = RangeSet(1,len(M_estaciones))

tiempo={}
for j in model.MODELOS:
    for i in model.TRABAJOS:
        tiempo[i,j] = tabla.cell(row=i+2,column=11+j).value
model.tiempo=Param(model.TRABAJOS, model.MODELOS, initialize=tiempo)

hreq1={}
for j in model.MODELOS:
    for i in model.TRABAJOS:
        hreq1[i,j] = tabla.cell(row=i+2,column=15+j).value
model.hreq1=Param(model.TRABAJOS,model.MODELOS,initialize=hreq1)

hreq2={}
```

```
for j in model.MODELOS:
    for i in model.TRABAJOS:
        hreq2[i,j] = tabla.cell(row=i+2,column=19+j).value
model.hreq2=Param(model.TRABAJOS,model.MODELOS,initialize=hreq2)

hreq3={}
for j in model.MODELOS:
    for i in model.TRABAJOS:
        hreq3[i,j] = tabla.cell(row=i+2,column=23+j).value
model.hreq3=Param(model.TRABAJOS,model.MODELOS,initialize=hreq3)

htotal1={}
for k in model.ESTACIONES:
        htotal1[k] = tabla.cell(row=k+2,column=28).value
model.htotal1=Param(model.ESTACIONES,initialize=htotal1)

htotal2={}
for k in model.ESTACIONES:
        htotal2[k] = tabla.cell(row=k+2,column=31).value
model.htotal2=Param(model.ESTACIONES,initialize=htotal2)

htotal3={}
for k in model.ESTACIONES:
        htotal3[k] = tabla.cell(row=k+2,column=34).value
model.htotal3=Param(model.ESTACIONES,initialize=htotal3)

model.x = Var(model.TRABAJOS, model.PERIODOS, model.MODELOS, within=Boolean)
model.lt = Var(model.MODELOS, within=PositiveIntegers)

model.LeadTime = Expression(expr = sum(model.lt[j] for j in model.MODELOS))
model.Operarios_ociosos = Expression(expr = sum(sum(model.htotal1[k] +\
model.htotal2[k] + model.htotal3[k] - sum(sum(model.hreq1[n,j] *\
(sum(model.x[n,k,j] for k in sequence(t)) - sum(model.x[n,k,j] for k in\
sequence(t-model.tiempo[n,j]))) + model.hreq2[n,j] * (sum(model.x[n,k,j]\
for k in sequence(t)) - sum(model.x[n,k,j] for k in sequence(t -\
model.tiempo[n,j]))) + model.hreq3[n,j] * (sum(model.x[n,k,j] for k in\
sequence(t)) - sum(model.x[n,k,j] for k in sequence(t -\
model.tiempo[n,j]))) for j in model.MODELOS) for n in\
 trabajos_por_estacion[k]) for k in model.ESTACIONES) for\
t in model.PERIODOS))

model.obj = Objective(expr = 0.5*model.LeadTime + 0.5*model.Operarios_ociosos)

def rest1_rule(model, i, j):
    return sum(model.x[i,t,j] for t in model.PERIODOS) == 1
model.rest1 = Constraint(model.TRABAJOS, model.MODELOS, rule=rest1_rule)

@model.Constraint(model.TRABAJOS, model.TRABAJOS, model.MODELOS)
def rest2(model, i, p, j):
    if p in precedencia_trabajos[i]:
```

```
        return sum(t * model.x[i, t, j] for t in model.PERIODOS) >= (sum(t *\
         model.x[p, t, j] for t in model.PERIODOS)+ model.tiempo[p,j])
    else:
        return Constraint.NoConstraint

@model.Constraint(model.PERIODOS, model.ESTACIONES, model.MODELOS)
def rest3(model, t, k, j):
        return sum(model.hreq1[n,j] * (sum(model.x[n,k,j] for k in sequence(t)) - \
        sum(model.x[n,s,j] for s in sequence(t - model.tiempo[n,j]))) for \
        n in trabajos_por_estacion[k]) <= model.htotal1[k]

@model.Constraint(model.PERIODOS, model.ESTACIONES, model.MODELOS)
def rest4(model, t, k, j):
        return sum(model.hreq2[n,j] * (sum(model.x[n,k,j] for k in sequence(t)) - \
        sum(model.x[n,s,j] for s in sequence(t - model.tiempo[n,j]))) for\
         n in trabajos_por_estacion[k]) <= model.htotal2[k]

@model.Constraint(model.PERIODOS, model.ESTACIONES, model.MODELOS)
def rest5(model, t, k, j):
        return sum(model.hreq3[n,j] * (sum(model.x[n,k,j] for k in sequence(t)) - \
        sum(model.x[n,s,j] for s in sequence(t - model.tiempo[n,j]))) \
        for n in trabajos_por_estacion[k]) <= model.htotal3[k]

def rest6_rule(model, i, t, j):
    return model.lt[j] >= model.x[i,t,j] * (t + model.tiempo[i,j] -1)
model.rest6 = Constraint(model.TRABAJOS, model.PERIODOS, model.MODELOS, \
rule=rest6_rule)

@model.Constraint(model.TRABAJOS, model.TRABAJOS, model.MODELOS)
def rest7(model, i, m, j):
    if j == 1:
        return Constraint.NoConstraint
    else:
        if m in precedencia_modelos[i]:
            return sum(t * model.x[i, t, j] for t in model.PERIODOS) >= \
            (sum(t * model.x[m, t, j-1] for t in model.PERIODOS)\
            + model.tiempo[m,j-1])
        else:
            return Constraint.NoConstraint

@model.Constraint(model.TRABAJOS, model.TRABAJOS, model.MODELOS)
def rest8(model, i, s, j):
    if s in precedencia_estaciones[i]:
        return sum(t * model.x[i, t, j] for t in model.PERIODOS) >=\
         (sum(t * model.x[s, t, j] for t in model.PERIODOS)+ model.tiempo[s,j])
    else:
        return Constraint.NoConstraint


opt = SolverFactory("glpk")
```

```
print ("\nSolución␣óptima␣encontrada\n" + '-'*80)
results = opt.solve(model, tee=True)
model.solutions.load_from( results )

print ("\nResumen␣de␣variables\n"+ '-'*80)
for j in model.MODELOS:
    for t in model.PERIODOS:
        for i in model.TRABAJOS:
            if model.x[i,t,j].value == 1:
                print ('x[' + str(i) + ',' + str(t) + ',' + str(j) + ']␣=␣',\
                model.x[i,t,j].value)

for j in model.MODELOS:
    print ('lt[' + str(j) + ']␣=␣',model.lt[j].value)

print ("\nPeriodo␣de␣inicio␣de␣cada␣uno␣de␣los␣trabajos\n"+ '-'*80)
for j in model.MODELOS:
    for t in model.PERIODOS:
        for i in model.TRABAJOS:
            if model.x[i,t,j].value == 1:
                print ("-␣Inicio␣trabajo ",i,",␣modelo",j,":",t)

print ("\n")

valor = []
for i in range(22):
    valor.append([])
    for j in range(5):
        valor[i].append(0)
for t in model.PERIODOS:
    for k in model.ESTACIONES:
        valor[t-1][k-1]=value(sum(sum((sum(model.x[n,k,j] for k in sequence(t))\
        - sum(model.x[n,s,j] for s in sequence(t - model.tiempo[n,j])))\
          for j in model.MODELOS) for n in trabajos_por_estacion[k]))
        if valor[t-1][k-1] > 1:
            valor[t-1][k-1] = int(1)

print ("\nOperarios␣ociosos␣por␣periodo\n"+ '-'*80)
hlibre_proceso_total = 0
for t in model.PERIODOS:
    hlibre_proceso_t = sum(model.htotal1[k]*valor[t-1][k-1] +\
    model.htotal2[k]*valor[t-1][k-1] + model.htotal3[k]*valor[t-1][k-1] -\
    sum(sum(model.hreq1[p,j] * (sum(model.x[p,k,j] for k in sequence(t)) -\
    sum(model.x[p,k,j] for k in sequence(t - model.tiempo[p,j]))) +\
    model.hreq2[p,j] * (sum(model.x[p,k,j] for k in sequence(t)) -\
    sum(model.x[p,k,j] for k in sequence(t - model.tiempo[p,j]))) +\
    model.hreq3[p,j] * (sum(model.x[p,k,j] for k in sequence(t)) -\
    sum(model.x[p,k,j] for k in sequence(t - model.tiempo[p,j])))\
```

```
    for j in model.MODELOS) for p in trabajos_por_estacion[k])\
    for k in model.ESTACIONES)
     hlibre_proceso_total = hlibre_proceso_total + value(hlibre_proceso_t)
    print('hlibre_proceso[' + str(t) + ']=', value(hlibre_proceso_t))

print("\n")
print('-El número total de operarios ociosos en proceso de la línea de ensamblaje\
es:', hlibre_proceso_total)
print('-El número total de operarios ociosos tras finalización de la línea de\
ensamblaje es:', value(model.Operarios_ociosos)- hlibre_proceso_total)
print('-El número total de operarios ociosos de la línea de ensamblaje es:',\
 value(model.Operarios_ociosos))

print('-El Lead Time total de la línea de ensamblaje es:',model.lt[2].value,\
'periodos')
```

A.4 Problema monoestacional

```
import pyomo.environ
from pyomo.core import *
from pyomo.opt import SolverFactory
import openpyxl
from openpyxl import load_workbook


wb = openpyxl.load_workbook("C:\\Users\\UX430U\\CursoPython\\DatosExcel3.xlsx")
hoja = wb['Hoja1']

numero_trabajos = hoja.max_row - 1

datos_precedencia = []
for i in range(numero_trabajos):
    datos_precedencia.append([])
    for j in range(2):
        datos_precedencia[i].append(0)

for i in range(numero_trabajos):
    datos_precedencia[i][0] = i+1

for fila in range(2, numero_trabajos+2):
    for columna in range(6, 8):
        if (hoja.cell(row=fila,column=columna).value) != 0:
            if columna == 6:
                datos_precedencia[fila-2][1] = hoja.cell(row=fila,column\
                =columna).value
            elif columna == 7:
                v=[]
                v.append(hoja.cell(row=fila,column=columna-1).value)
                v.append(hoja.cell(row=fila,column=columna).value)
```

```
                    datos_precedencia[fila-2][1] = v
                elif columna > 7:
                    v.append(hoja.cell(row=fila,column=columna).value)
                    datos_precedencia[fila-2][1] = v

M_precedencia = []
for i in range(numero_trabajos):
    M_precedencia.append([])
    for j in range(numero_trabajos):
        M_precedencia[i].append(0)

for valor in range(numero_trabajos):
    x=datos_precedencia[valor][0]
    y=datos_precedencia[valor][1]
    if type(y) == int:
        if y>0:
            M_precedencia[x-1][y-1] = 1
    else:
        tamaño_vconverge = len(y)
        for numero in range(tamaño_vconverge):
            M_precedencia[x-1][y[numero]-1] = 1

hoja2 = wb['Hoja2']

M_estructura =[]
numero_filas = hoja2.max_row
linea_final = hoja2.cell(row=numero_filas,column=1).value

M_estaciones = []
fila_ultima = hoja.max_row
numero_estaciones = hoja.cell(row=fila_ultima,column=1).value

for s in range(0,numero_estaciones):
    M_estaciones.append([])

for fila in range(2, fila_ultima+1):
    M_estaciones[hoja.cell(row=fila,column=1).value - 1].\
    append(hoja.cell(row=fila,column=5).value)

wb2 = openpyxl.load_workbook("C:\\Users\\UX430U\\CursoPython\\TablasDatos3.xlsx")
tabla = wb2['Hoja1']

precedencia_trabajos = dict()
for i in range(numero_trabajos):
    precedencia_trabajos[i+1] = []
    for j in range(numero_trabajos):
        if M_precedencia[i][j] == 1:
            precedencia_trabajos[i+1].append(j+1)

trabajos_por_estacion = dict()
```

```
for i in range(len(M_estaciones)):
    trabajos_por_estacion[i+1]= []
for s in range(len(M_estaciones)):
    for m in range(len(M_estaciones[s])):
        trabajos_por_estacion[s+1].append(M_estaciones[s][m])

model = ConcreteModel()

model.TRABAJOS = RangeSet(1,len(M_precedencia))
model.PERIODOS = RangeSet(1,10)
model.MODELOS = [1]
model.ESTACIONES = RangeSet(1,len(M_estaciones))

tiempo={}
for j in model.MODELOS:
    for i in model.TRABAJOS:
        tiempo[i,j] = tabla.cell(row=i+2,column=11+j).value
model.tiempo=Param(model.TRABAJOS, model.MODELOS, initialize=tiempo)

hreq1={}
for j in model.MODELOS:
    for i in model.TRABAJOS:
        hreq1[i,j] = tabla.cell(row=i+2,column=15+j).value
model.hreq1=Param(model.TRABAJOS,model.MODELOS,initialize=hreq1)

hreq2={}
for j in model.MODELOS:
    for i in model.TRABAJOS:
        hreq2[i,j] = tabla.cell(row=i+2,column=19+j).value
model.hreq2=Param(model.TRABAJOS,model.MODELOS,initialize=hreq2)

hreq3={}
for j in model.MODELOS:
    for i in model.TRABAJOS:
        hreq3[i,j] = tabla.cell(row=i+2,column=23+j).value
model.hreq3=Param(model.TRABAJOS,model.MODELOS,initialize=hreq3)

model.x = Var(model.TRABAJOS, model.PERIODOS, model.MODELOS, within=Boolean)
model.lt = Var(model.MODELOS, within=PositiveIntegers)
model.htotal1 = Var(model.ESTACIONES, within=PositiveIntegers)
model.htotal2 = Var(model.ESTACIONES, within=PositiveIntegers)
model.htotal3 = Var(model.ESTACIONES, within=PositiveIntegers)

model.LeadTime = Expression(expr = sum(model.lt[j] for j in model.MODELOS))
model.Operarios_ociosos = Expression(expr = sum(sum(model.htotal1[k] +\
model.htotal2[k] + model.htotal3[k] - sum(sum(model.hreq1[n,j] *\
(sum(model.x[n,k,j] for k in sequence(t)) - sum(model.x[n,k,j] for k in\
sequence(t-model.tiempo[n,j]))) + model.hreq2[n,j] * (sum(model.x[n,k,j]\
for k in sequence(t)) - sum(model.x[n,k,j] for k in sequence(t -\
model.tiempo[n,j]))) + model.hreq3[n,j] * (sum(model.x[n,k,j] for k in\
```

```
sequence(t)) - sum(model.x[n,k,j] for k in sequence(t -\
model.tiempo[n,j]))) for j in model.MODELOS) for n in\
 trabajos_por_estacion[k]) for k in model.ESTACIONES) for\
t in model.PERIODOS))

model.obj = Objective(expr = model.LeadTime)

def rest1_rule(model, i, j):
    return sum(model.x[i,t,j] for t in model.PERIODOS) == 1
model.rest1 = Constraint(model.TRABAJOS, model.MODELOS, rule=rest1_rule)

@model.Constraint(model.TRABAJOS, model.TRABAJOS, model.MODELOS)
def rest2(model, i, p, j):
    if p in precedencia_trabajos[i]:
        return sum(t * model.x[i, t, j] for t in model.PERIODOS) >= (sum(t *\
         model.x[p, t, j] for t in model.PERIODOS)+ model.tiempo[p,j])
    else:
        return Constraint.NoConstraint

@model.Constraint(model.PERIODOS, model.ESTACIONES, model.MODELOS)
def rest3(model, t, k, j):
        return sum(model.hreq1[n,j] * (sum(model.x[n,k,j] for k in sequence(t)) - \
        sum(model.x[n,s,j] for s in sequence(t - model.tiempo[n,j]))) for \
        n in trabajos_por_estacion[k]) <= model.htotal1[k]

@model.Constraint(model.PERIODOS, model.ESTACIONES, model.MODELOS)
def rest4(model, t, k, j):
        return sum(model.hreq2[n,j] * (sum(model.x[n,k,j] for k in sequence(t)) - \
        sum(model.x[n,s,j] for s in sequence(t - model.tiempo[n,j]))) for\
         n in trabajos_por_estacion[k]) <= model.htotal2[k]

@model.Constraint(model.PERIODOS, model.ESTACIONES, model.MODELOS)
def rest5(model, t, k, j):
        return sum(model.hreq3[n,j] * (sum(model.x[n,k,j] for k in sequence(t)) - \
        sum(model.x[n,s,j] for s in sequence(t - model.tiempo[n,j]))) \
        for n in trabajos_por_estacion[k]) <= model.htotal3[k]

def rest6_rule(model, i, t, j):
    return model.lt[j] >= model.x[i,t,j] * (t + model.tiempo[i,j] -1)
model.rest6 = Constraint(model.TRABAJOS, model.PERIODOS, model.MODELOS, \
rule=rest6_rule)

def rest9_rule(model):
    return model.Operarios_ociosos <= 73
model.rest9 = Constraint(rule=rest9_rule)

opt = SolverFactory("glpk")

print("\nSolución óptima encontrada\n" + '-'*80)
results = opt.solve(model, tee=True)
```

```
model.solutions.load_from(results)

def pyomo_postprocess(options=None, instance=None, results=None):
    model.x.display(), model.obj.display(), model.Operarios_ociosos.display()
pyomo_postprocess(None, None, results)

print("\nResumen de la solución encontrada\n"+'-'*80)
for j in model.MODELOS:
    for t in model.PERIODOS:
        for i in model.TRABAJOS:
            if model.x[i,t,j].value == 1:
                print("El trabajo",i,"del modelo",j,"comienza en el periodo:",t)

print("\n")

valor = []
for i in range(22):
    valor.append([])
    for j in range(5):
        valor[i].append(0)
for t in model.PERIODOS:
    for k in model.ESTACIONES:
        valor[t-1][k-1]=value(sum(sum((sum(model.x[n,k,j] for k in sequence(t))\
        - sum(model.x[n,s,j] for s in sequence(t - model.tiempo[n,j])))\
          for j in model.MODELOS) for n in trabajos_por_estacion[k]))
        if valor[t-1][k-1] > 1:
            valor[t-1][k-1] = int(1)

print("\nOperarios ociosos por periodo\n"+'-'*80)
hlibre_total = 0
for t in model.PERIODOS:
    hlibre_t = sum(model.htotal1[k]*valor[t-1][k-1] + model.htotal2[k]*\
    valor[t-1][k-1] + model.htotal3[k]*valor[t-1][k-1] - sum(sum(model.hreq1[p,j]\
    * (sum(model.x[p,k,j] for k in sequence(t)) - sum(model.x[p,k,j] for k\
    in sequence(t - model.tiempo[p,j]))) + model.hreq2[p,j] * (sum(model.x[p,k,j] for \
    k in sequence(t)) - sum(model.x[p,k,j] for k in sequence(t - model.tiempo[p,j])))\
    + model.hreq3[p,j] * (sum(model.x[p,k,j] for k in sequence(t)) -\
    sum(model.x[p,k,j] for k in sequence(t - model.tiempo[p,j]))) for\
    j in model.MODELOS) for p in trabajos_por_estacion[k]) for k in\
    model.ESTACIONES)
    hlibre_total = hlibre_total + value(hlibre_t)
    print('hlibre['+str(t)+'] =', value(hlibre_t))

print("\n")
print('htotal1[1]=', value(model.htotal1[1]))
print('htotal2[1]=', value(model.htotal2[1]))
print('htotal3[1]=', value(model.htotal3[1]))

print("\n")
print('- El número total de operarios ociosos de la línea de ensamblaje es:',\
```

```
 hlibre_total )
print ('- El Lead Time total de la línea de ensamblaje es:',model.lt [1]. value ,\
'periodos')
```

A.5 Modelo adicional. Ejemplo práctico

```
import pyomo.environ
from pyomo.core import *
from pyomo.opt import SolverFactory

model = ConcreteModel()

model.TRABAJOS = RangeSet(1,2)
model.ESTACIONES = RangeSet(1,2)
model.PERIODOS = RangeSet(1,7)

model.K = Param(model.TRABAJOS, model.ESTACIONES, initialize={(1,1):1, (1,2):0, \
 (2,1):1,  (2,2):0})
model.hmaxt = Param(model.PERIODOS, initialize={1:2, 2:6, 3:7, 4:14, 5:9, 6:6, 7:5})
model.hmaxest = Param(model.ESTACIONES, initialize={1:6, 2:8})
model.hcarga = Param(model.TRABAJOS, model.ESTACIONES, initialize={(1,1):2, (1,2):0, \
 (2,1):2,  (2,2):0})
model.Y = Param( initialize =8)
model.L = Param(model.TRABAJOS, model.ESTACIONES, initialize={(1,1):88, (1,2):128, \
 (2,1):40,  (2,2):32})
model.A = Param( initialize =12)
model.C = Param( initialize =20)
model.D = Param(model.ESTACIONES, initialize={1:30, 2:40})

model.x = Var(model.TRABAJOS, model.ESTACIONES, model.PERIODOS, within=Boolean)
model.y = Var(model.TRABAJOS, model.ESTACIONES, model.PERIODOS, within=Boolean)
model.h = Var(model.TRABAJOS, model.ESTACIONES, model.PERIODOS, \
within=NonNegativeIntegers)
model.z = RangeSet(1,1)

model.CosteLaboral = Expression(expr = sum(sum(model.A*model.Y*sum(model.h[i,j,t] for \
t in model.PERIODOS)+model.A*model.Y*sum((model.hcarga[i,j]*sum(model.x[i,j,k] for \
k in sequence(t-model.K[i,j]+1,t))) for t in model.PERIODOS) for j in \
model.ESTACIONES) for i in model.TRABAJOS) + model.C*model.Y*\
(sum(model.hmaxt[t] - sum(sum(model.h[i,j,t] + (model.hcarga[i,j]*\
sum(model.x[i,j,k] for k in sequence(t-model.K[i,j]+1,t))) for \
j in model.ESTACIONES) for i in model.TRABAJOS) for t in model.PERIODOS)))

model.MatenimientoInventario = Expression(expr = sum(sum(model.D[j]*(sum((t*\
model.x[i,j+1,t]-t*model.y[i,j,t])-1 for t in model.PERIODOS)) for \
j in model.z) for i in model.TRABAJOS) + sum(sum(model.D[j]*(sum((t*model.y[i,j,t]\
- t*model.x[i,j,t])+1 for t in model.PERIODOS)) for j in model.ESTACIONES) for \
i in model.TRABAJOS))
model.obj = Objective(expr = model.CosteLaboral + model.MatenimientoInventario)
```

```
def rest1_rule(model, i, j):
    return sum(model.x[i,j,t] for t in model.PERIODOS) == 1
model.rest1 = Constraint(model.TRABAJOS, model.ESTACIONES, rule=rest1_rule)

def rest2_rule(model, i, j):
    return sum(model.y[i,j,t] for t in model.PERIODOS) == 1
model.rest2 = Constraint(model.TRABAJOS, model.ESTACIONES, rule=rest2_rule)

def rest3_rule(model, i, j):
    return sum(t*model.x[i,j,t] for t in model.PERIODOS) + model.K[i,j] <= \
    sum(t*model.y[i,j,t] for t in model.PERIODOS)
model.rest3 = Constraint(model.TRABAJOS, model.ESTACIONES, rule=rest3_rule)

@model.Constraint(model.TRABAJOS, model.ESTACIONES)
def rest4_rule(model, i, j):
    if j == 2:
        return Constraint.Skip
    else:
        return sum(t*model.y[i,j,t] for t in model.PERIODOS) + 1 <= \
        sum(t*model.x[i,j+1,t] for t in model.PERIODOS)

@model.Constraint(model.TRABAJOS, model.ESTACIONES)
def rest6_rule(model, i, j):
    if i == 1:
        return Constraint.Skip
    else:
        return sum(t*model.y[i-1,j,t] for t in model.PERIODOS) + 1 <= \
        sum(t*model.x[i,j,t] for t in model.PERIODOS)

@model.Constraint(model.TRABAJOS, model.ESTACIONES)
def rest7_rule(model, i, j):
    if j == 2 or i == 2:
        return Constraint.Skip
    else:
        return sum(t*model.x[i+1,j,t] for t in model.PERIODOS) >= \
        sum(t*model.x[i,j+1,t] for t in model.PERIODOS)

def rest9_rule(model, i, j, t):
        return model.h[i,j,t] <= model.hmaxest[j]*(sum(model.x[i,j,s] for s in \
        sequence(t-model.K[i,j])) - sum(model.y[i,j,m] for m in sequence(t - 1)))
model.rest9 = Constraint(model.TRABAJOS, model.ESTACIONES, model.PERIODOS, \
rule=rest9_rule)

def rest10_rule(model, t):
    return sum(sum(model.h[i,j,t] + (model.hcarga[i,j]*sum(model.x[i,j,s] for s in \
    sequence(t-model.K[i,j]+1, t))) for i in model.TRABAJOS) for j in \
    model.ESTACIONES) <= model.hmaxt[t]
model.rest10 = Constraint(model.PERIODOS, rule=rest10_rule)
```

```
def rest11_rule (model, i, j):
    return sum(model.Y*model.h[i,j,t] for t in model.PERIODOS) >= model.L[i,j]
model.rest11 = Constraint (model.TRABAJOS, model.ESTACIONES, rule=rest11_rule)

opt = SolverFactory ("glpk")

print ("\nSolución óptima encontrada\n" + '-'*80)
results = opt.solve(model, tee=True)
model.solutions.load_from( results )
def pyomo_postprocess(options=None, instance=None, results=None):
    model.x.display (), model.y.display (), model.obj.display (), model.h.display ()
pyomo_postprocess(None, None, results )
```

Bibliografía

[1] *openpyxl - A Python library to read/write Excel 2010 xlsx/xlsm files — openpyxl 2.5.6 documentation*, *https://openpyxl.readthedocs.io/en/stable/* [Último acceso: Agosto 2018].

[2] *Airbus*, 2018, Documento interno Airbus (A400M FAL Build Process).

[3] *Biblioteca (informática)*, July 2018, *https://es.wikipedia.org/w/index.php?title=Biblioteca_(inform%C3%A1tica)&oldid=109137870* [Último acceso: Agosto 2018].

[4] *CASA C-295 AEW*, May 2018, *https://es.wikipedia.org/w/index.php?title=CASA_C-295_AEW&oldid=108278589* [Último acceso: Agosto 2018].

[5] Aerocontact, *Airbus C295 - Airbus Defence and Space - Transport and Tanker Aircrafts*, August 2018, *https://www.aerocontact.com/en/virtual-aviation-exhibition/product/361-airbus-c295* [Último acceso: Agosto 2018].

[6] Faris Al Barrak, Youssef Al Meriouh, and Meriem Zniber El Mouhabbis, *Work In Process stock integrity in the automotive industry*, Production & Manufacturing Research **5** (2017), no. 1, 2–14.

[7] Bismark Ameyaw and Kwaku Darkwah, *Linear Programming: Theory and Computations*, LAP LAMBERT Academic Publishing, July 2012.

[8] Kavit R. Antani, Bryan Pearce, Laine Mears, Rahul Renu, Mary E. Kurz, and Joerg Schulte, *Application of System Learning to Precedence Graph Generation for Assembly Line Balancing*, (2014), V001T04A001.

[9] Ghassan Ezzulddin Arif and Yarub Al-Douri, *Mathematical Modeling of Physical Properties for Hexagonal Binaries*, Scholars'Press, February 2016.

[10] Christian Becker and Armin Scholl, *A survey on problems and methods in generalized assembly line balancing*, European Journal of Operational Research **168** (2006), no. 3, 694–715.

[11] Dimitris Bertsimas, John N. Tsitsiklis, and John Tsitsiklis, *Introduction to Linear Optimization*, edición: unknown ed., Athena Scientific, Belmont, Mass, February 1997.

[12] Alexander Biele and Lars Mönch, *Hybrid approaches to optimize mixed-model assembly lines in low-volume manufacturing*, Journal of Heuristics **24** (2018), no. 1, 49–81.

[13] Stephen P. Bradley, Arnoldo C. Hax, and Thomas L. Magnanti, *Applied Mathematical Programming*, Addison-Wesley Publishing Company, January 1977, Google-Books-ID: MSWdWv3Gn5cC.

[14] Jean-François Bérubé, Michel Gendreau, and Jean-Yves Potvin, *An exact -constraint method for bi-objective combinatorial optimization problems: Application to the Traveling Salesman Problem with Profits*, European Journal of Operational Research **194** (2009), no. 1, 39–50.

[15] Joseph Bukchin, Ezey M. Dar-El, and Jacob Rubinovitz, *Mixed model assembly line design in a make-to-order environment*, Computers & Industrial Engineering **41** (2002), no. 4, 405–421.

[16] LAWRENCE D. BURNS and CARLOS F. DAGANZO, *Assembly line job sequencing principles*, International Journal of Production Research **25** (1987), no. 1, 71–99.

[17] Der-San Chen, Robert G. Batson, and Yu Dang, *Applied Integer Programming: Modeling and Solution*, John Wiley & Sons, September 2011, Google-Books-ID: tjZBwmHs49sC.

[18] George R. Exner, *An Accompaniment to Higher Mathematics*, Undergraduate Texts in Mathematics, Springer-Verlag, New York, 1996.

[19] Marshall L Fisher and Christopher D Ittner, *THE IMPACT OF PRODUCT VARIETY ON AUTOMOBILE ASSEMBLY OPERATIONS: ANALYSIS AND EVIDENCE*, 31.

[20] William E. Hart, *Python Optimization Modeling Objects (Pyomo)*, Operations Research and Cyber-Infrastructure (John W. Chinneck, Bjarni Kristjansson, and Matthew J. Saltzman, eds.), Operations Research/Computer Science Interfaces, Springer US, 2009, pp. 3–19.

[21] William E. Hart, Carl Laird, Jean-Paul Watson, and David L. Woodruff, *Pyomo – Optimization Modeling in Python*, Springer Optimization and Its Applications, Springer-Verlag, New York, 2012.

[22] William E. Hart, Jean-Paul Watson, and David L. Woodruff, *Pyomo: modeling and solving mathematical programs in Python*, Mathematical Programming Computation **3** (2011), no. 3, 219.

[23] G. Heike, M. Ramulu, E. Sorenson, P. Shanahan, and K. Moinzadeh, *Mixed model assembly alternatives for low-volume manufacturing: The case of the aerospace industry*, International Journal of Production Economics **72** (2001), no. 2, 103–120.

[24] Frederick Hillier, *Introduction to Operations Research*, February 2014, Google-Books-ID: 1mhzCgAAQBAJ.

[25] Thomas R. Hoffmann, *Assembly Line Balancing with a Precedence Matrix*, Management Science **9** (1963), no. 4, 551–562.

[26] Bernard Kolman and Robert E. Beck, *Elementary Linear Programming with Applications*, Elsevier, May 2014, Google-Books-ID: iLrSBQAAQBAJ.

[27] Suresh Kotha and B. Joseph Pine, *Mass Customization: The New Frontier in Business Competition, B. Joseph Pine, II*, The Academy of Management Review **19** (1994), no. 3, 588–592.

[28] Jonathan Oesterle and Lionel Amodeo, *Efficient Multi-objective Optimization Method for the Mixed-model-line Assembly Line Design Problem*, Procedia CIRP **17** (2014), 82–87.

[29] Jose David Canca Ortiz and Pedro Luis Gonzalez Rodríguez, *Técnicas de optimización*, edición: 1 ed., Iris-Copy, Mairena de Aljarafe, Sevilla, April 2015.

[30] Subbaraj Potti, Chitra Chinnasamy, Subbaraj Potti, and Chitra Chinnasamy, *Strength Pareto Evolutionary Algorithm based Multi-Objective Optimization for Shortest Path Routing Problem in Computer Networks*, Journal of Computer Science **7** (2010), no. 1, 17–26.

[31] B. Rekiek, P. De Lit, and A. Delchambre, *Designing mixed-product assembly lines*, IEEE Transactions on Robotics and Automation **16** (2000), no. 3, 268–280.

[32] Carlos Romero, *Teoría de la decisión multicriterio: conceptos, técnicas y aplicaciones*, Alianza Editorial, 1993, Google-Books-ID: 6XtvPgAACAAJ.

[33] Brian G. Thomas, *Modeling of the continuous casting of steel—past, present, and future*, Metallurgical and Materials Transactions B **33** (2002), no. 6, 795–812.

[34] H. P. Williams, *The problem with integer programming*, IMA Journal of Management Mathematics **22** (2011), no. 3, 213–230.

[35] H. Paul Williams, *Model Building in Mathematical Programming*, John Wiley & Sons, January 2013, Google-Books-ID: YJRh0tOes7UC.

[36] Xin-She Yang, *Nature-Inspired Optimization Algorithms*, Elsevier, February 2014, Google-Books-ID: BbHVAQAAQBAJ.

[37] Ayman M. A. Youssef and Hoda A. ElMaraghy, *Assessment of manufacturing systems reconfiguration smoothness*, The International Journal of Advanced Manufacturing Technology **30** (2006), no. 1, 174–193.

[38] Timm Ziarnetzky, Lars Mönch, and Alexander Biele, *Simulation of Low-volume Mixed Model Assembly Lines: Modeling Aspects and Case Study*, Proceedings of the 2014 Winter Simulation Conference (Piscataway, NJ, USA), WSC '14, IEEE Press, 2014, pp. 2101–2112.