VNIVERSITAT DE VALÈNCIA

Planificación de la producción en un entorno multifábrica

Trabajo final de grado. Curso 2017 - 2018

Autor: **Juan Luis García Dus**

Tutor: Ramón Álvarez-Valdés Olaguíbel

Facultat de Ciències Matemàtiques

Grado en Matemáticas

En Burjassot, a 12 de julio de 2018

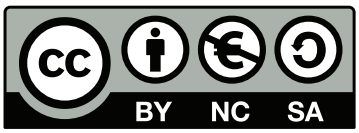
Resumen

Las cadenas de suministro en las que se insertan los procesos de producción son cada vez más complejas. Las empresas que cuentan con **fábricas** en diferentes localizaciones **reparten** entre ellas la **fabricación** de piezas y componentes. E incluso llegan a **externalizar** el **montaje** de los **productos finales**, enviando sus componentes a otras plantas para su montaje final.

Surgen, por tanto, **nuevos problemas de planificación y programación de la producción** inherentes a esta forma de producción descentralizada. Se ha de asignar la producción de cada planta, programar su producción y, llegados el caso, planificar los envíos a la planta en la que se realizará el montaje.

Si bien es cierto que cada uno de estos problemas se pueden estudiar de forma separada, la coyuntura actual no lo permite. En un entorno altamente competitivo y voraz, en el que se busca minimizar costes y diferenciarse de la competencia, lo ideal es **abordarlos de forma conjunta e integrada para obtener, así, las mejores soluciones globales**, que mejoren el rendimiento global de la producción.

En este trabajo se estudiará el problema de planificación de la producción en este **entorno multiplanta**, realizando la pertinente búsqueda bibliográfica y la revisión de los trabajos publicados en el tema. Se **analizarán diferentes modelos de Programación Lineal Entera y algunos Algoritmos Heurísticos**, que puedan proporcionar buenas soluciones en tiempos de computación adecuados.



Índice general

Capítulo 1

Introducción

En la actualidad, los sistemas de producción se circunscriben dentro de un entorno global, complejo y en constante variación, donde

- Los **consumidores reclaman** cada vez más **personalizaciones** en los bienes que consumen y **exigen plazos de entrega muy ajustados**.
- Existe una **competencia voraz** enmarcada en un **mercado global**, sin fronteras e interconectado, donde la supervivencia viene marcada por la **minimización de costes** de producción, los **valores añadidos** (como la personalización, las técnicas de marketing agresivas o con prolongaciones en el ciclo de vida de los productos) y los **servicios postventa**.

En este escenario, resulta difícil tener éxito afrontando los problemas de planificación de forma clásica. Y se requieren **nuevos modelos de producción** para hacer frente a estos retos.

Una forma de producción que puede contribuir a tener éxito en el escenario actual consiste en establecer una **red de fabricación parcialmente distribuida**, y descentralizada, conformada por varios centros de producción independientes en los que se realiza la fabricación de los productos finales en paralelo.

Pero, quizás, la forma más interesante y efectiva de lidiar con la situación actual sea contar con una **red de fabricación totalmente distribuida**, en la que los componentes de los productos se fabriquen en diferentes localizaciones y se ensamblen al final, pudiendo llegar a permitir al cliente la personalización del producto.

En cualquier caso, **resulta imprescindible contar con herramientas, modelos y técnicas que permita controlar y mejorar la eficiencia y el rendimiento de estas redes de fabricación distribuidas**.

Ya no solo para planificar la ejecución de la producción *per se* (cómo venía siendo habitual durante el siglo pasado), sino para **controlar** los **costes** asociada a ésta y **determinar**, antes de realizar grandes inversiones en manufactura y equipos, si resultan **asumibles** los **objetivos** y los **riesgos** asociados a la producción.

En un escenario global y exigente, **es impensable iniciar la producción sin tener claro y planificado absolutamente todos los detalles** de ésta.

Es, por ello, que **este trabajo pretende analizar las diferentes herramientas, modelos y técnicas disponibles** para planificar la producción, con todos sus detalles, en los **entornos de fabricación distribuida más habituales** hoy en día. Y ofrecer una visión global para este tipo de problemas del área de la investigación operativa.

El **objetivo** de este capítulo es recoger una pequeña **introducción** que **capacite** al lector para seguir y **comprender** el **desarrollo** de este **trabajo** fin de grado. Es decir, un *background* que permita desarrollar con éxito toda la teoría y técnica posterior.

Este capítulo contendrá, por tanto, una **revisión** de los principales **conceptos**, métodos y sistemas involucrados en la **programación de la producción** (*Machine Scheduling*) clásica. Y desarrollará, brevemente, **nuevas formas de producción y ensamblaje** con las que se pretende tener éxito en el escenario actual, entre las que se incluyen las líneas de montaje final o redes de fabricación distribuida.

Además, por completitud, se comentará en qué consiste la optimización y qué métodos hay para obtener soluciones a los problemas de optimización.

1.1. Scheduling

Antes de iniciar cualquier producción, **se necesita conocer**

- En qué **orden** se va a realizar la manufactura.
- En qué **momento** se iniciará y se completará la producción.
- Los **costes** asociados a ésta.
- La **secuencia** que debe seguir la fabricación.
- Si resultan **asequibles** los **objetivos** marcados con cada producto (fechas de entregas comprometidas), con respecto a las capacidades disponibles y otros compromisos.
- E incluso **si** se **puede realizar toda** la **producción** prevista.

Se necesita, por tanto, programar la producción.

La programación o ***scheduling*** consiste en la **asignación de tiempo y recursos** (limitados) a **tareas** que debe **realizar**

- una industria, manufactura o empresa,
- en un determinado periodo de tiempo,
- bajo una serie de restricciones y condiciones específicas y
- de forma que se optimice un objetivo. [38]

El concepto de tareas y recursos pueden ser muy amplio [18]. Así,

- El concepto de **tarea** abarca desde **establecer** el **orden** de aterrizaje de los aviones en un aeropuerto hasta **crear** el **plan maestro de producción** (la secuenciación de producción en cualquier industria o manufactura), pasando por **elaborar** los **turnos** de trabajadores y operarios.
- Los **recursos** de los que se disponen incluyen las **máquinas** o las **materias primas** de la factoría en cuestión, las pistas de un determinado aeropuerto o el **personal disponible** para acometer la tarea, las **cargas y estimaciones de producción**...

Se deduce, por tanto, que los problemas de programación aparecen en ambientes muy diversos. Así, por ejemplo, se habla de:

- Problemas de secuenciación de tareas en máquinas (***Machine scheduling***).
- Problemas de secuenciación de proyectos (***Project scheduling***).
- Problemas de horarios (***Timetabling***).
- Problemas de turnos de trabajadores (***Labour scheduling***).
- Problemas de transporte y distribución (***Transportation and distribution scheduling***).

Los **objetivos que se buscan optimizar** en la programación son muy diversos, y dependen, en gran medida, del ambiente donde se realiza la programación.

Centrando el tema del que versa este documento, en un ambiente de ***Machine Scheduling*** el objetivo más común es la **minimización** del tiempo máximo de completación de las tareas (***makespan***) [18]. Pero otros pueden ser la **minimización** del número de tareas completadas después de sus respectivas fechas de vencimiento (***tardiness of jobs***) [38] o **minimizar** el **retraso máximo** que experimentan las tareas.

1.2. Clasificación de los problemas de *Machine Scheduling*

Se usará para clasificar los problemas de secuenciación de tareas en máquinas la notación introducida en [15], siguiendo la estructura propuesta en [6].

Los problemas de *machine scheduling* se clasifican atendiendo a tres parámetros,

$$\alpha \mid \beta \mid \gamma,$$

donde α hace referencia a las características de las máquinas, β a las características de las tareas y γ a la de los objetivos.

1.2.1. Características de las máquinas

Respecto a la estructura o configuración de las máquinas, básicamente hay tres clases de problemas: problemas con una sola máquina (***single-machine problems***), problemas con varias máquinas que realizan tareas con una sola operación[I] (***single-stage multi-machine problems***) y problemas con varias máquinas que realizan tareas con varias operaciones sucesivas (***multi-stage multi-machine problems***)

Single-machine problems

Este tipo de problemas son los más sencillos y menos complejos conceptualmente, que parten de una situación ideal, en la que **se desea planificar la producción de una sola máquina**: en ella, exclusivamente, se deben procesar las tareas (*jobs*) que se precisan.

Su utilidad no es meramente teórica, sino que su estudio histórico y su comprensión han permitido realizar un **mejor análisis de modelos más complejos**:

- bien porque muchas de las **soluciones** de este modelo (que cuentan con buenas propiedades) pueden dar ideas que son susceptibles de ser **aplicadas a modelos complejos**,
- bien porque algunos sistemas complejos se pueden descomponer como subproblemas de una máquina (**cuellos de botella**).

[I] Es decir, diversas máquinas colocadas en paralelo, que deben realizar tareas con una sola operación.

Single-stage multi-machine problems

Debido a las **múltiples aplicaciones en las líneas de producción reales**, este problema ha sido ampliamente estudiado durante las últimas décadas, creando así una extensa literatura sobre él. [18]

El problema persigue **programar exactamente n trabajos, con una sola operación, en m máquinas, dispuestas en paralelo**, estableciendo, así, una secuenciación de los trabajos a realizar en cada máquina.

Atendiendo a la **velocidad de procesado** que ofrecen las máquinas de las que se disponen, se identifican varios **subtipos** de problemas:

- Aquellos con m **máquinas idénticas** en paralelo, en los que el **tiempo de procesado** de cada trabajo es el **mismo en todas las máquinas**. Estos problemas se denotan por $\mathbf{P_m}$.
- Aquellos con m **máquinas** en paralelo, en los que el **tiempo de procesado** de cada trabajo **depende**, inversamente, de la **velocidad de cada máquina**[II]. Estos problemas se denotan por $\mathbf{Q_m}$.
- Aquellos con m **máquinas** en paralelo, en los que el **tiempo de procesado** de cada trabajo en cada máquina es diferente[III]. Estos problemas se denotan por $\mathbf{R_m}$.

Multi-stage multi-machine problems

El problema persigue **programar** n **trabajos**, **compuestos** de varias operaciones o **subtareas sucesivas**, **en** m **máquinas** diferentes (cada una ejecuta una operación), **estableciendo** una **secuenciación** de las tareas a realizar en cada máquina.

En función de las características, los problemas se dividen en tres tipos:

- **Flowshop problems** (F_m), donde cada una de los n trabajos a realizar pasa por las m máquinas con la misma ruta, completándose en cada una de ellas las diferentes subtareas que componen el trabajo. Indicar que el tiempo de procesado de cada operación es diferente en cada máquina[IV].
- **Jobshop problems** (J_m), donde cada una de los n trabajos a realizar pasa por las m máquinas con su propia ruta[V].
- **Openshop problems** (O_m), donde cada una de los n trabajos a realizar pasa por las m máquinas. Los trabajos no tienen una ruta preestablecida, un orden de ejecución de las subtareas, simplemente se deben de ejecutar para completar el trabajo.

[II] La velocidad de cada máquina, por tanto, es uniforme para todas las tareas

[III] La velocidad con la que realiza las operaciones cada máquina depende de la tarea que esté ejecutando

[IV] En general, cada máquina ejecuta una subtarea diferente. Y, por ello, para cada trabajo, el tiempo de procesado en cada máquina es diferente.

[V] Cada trabajo a realizar tiene un orden específico para las operaciones que se deben realizar sobre él. Es decir, cada trabajo dispone de una secuencia propia, de un recorrido por las máquinas concreto.

Tradicionalmente, para simplificar y modelizar las situaciones en las que se enmarca la producción, cuando no existen los respectivos modificadores en el parámetro β, se han supuesto varias condiciones ideales:

- Las **máquinas** involucradas están **siempre disponible** durante todo el tiempo.
- Las máquinas solo pueden procesar **una tarea a la vez**.
- Mientras una máquina está realizando una **tarea**, **no** se puede **interrumpir** ésta y se debe esperar a que termine para iniciar otra.
- El **tiempo de procesado** en cada máquina resulta **conocido** y es un valor **entero**, **positivo** e **independiente**.
- Cada trabajo está disponible en tiempo cero, es decir, desde el principio está disponible para ser procesado en las máquinas.

1.2.2. Características de las tareas

Con respecto a la clasificación según β, existen varios **parámetros intrínsecos a las tareas** y otros relacionados con las **particularidades de las máquinas** utilizadas en la producción.

A saber, por un lado, es necesario indicar, para cada tarea, j,

- La **duración** (*processing time*) de esta tarea en cada máquina, i, p_{ij}.
- La **fecha de disponibilidad** (*release time*) de ésta, r_j, si existe la posibilidad de que no sea inmediata.
- La **fecha de entrega** (*due date*) de ésta, d_j, si existe.
- La **fecha límite** (*deadline*) de ésta, $\overline{d}_j$, si existe.
- El **tiempo de preparación** (*setup time*) en la máquina i entre las tareas j y k, s_{ijk}, si existe.
- La importancia, o **peso** ponderado, de la **tarea** j, w_j, si se establece.

Y por otro lado, los siguientes modificadores restablecen características y condiciones especiales del proceso de producción, ignoradas tradicionalmente.

- **pmtn**: La posibilidad de interrumpir cualquier tarea y reanudarla más adelante (*preemption*).
- **prec**: Existen relaciones de precedencia entre las tareas a realizar (*precedence relation*).
- **brkdwn**: Hay máquinas que no están disponibles en algunos momentos (*breakdown*).
- **prmu**: Todas las tareas siguen la misma ruta[VI]. No se cambia el orden de los trabajos al pasar de una máquina a otra, por lo que las soluciones son permutaciones de los trabajos (*permutation*).
- **nwt**: Las tareas se deben ejecutar de forma continuada, sin posibilidad de parar entre las máquinas (*no wait*).

[VI] En la gran mayoría de literatura sobre *machine scheduling*, se entiende que $F_m|prmu|\cdot$ es $F_m|\cdot|\cdot$.

- **block**: Indica que, en un sistema de producción en el que existe un *buffer* intermedio[VII], en caso de llenarse este *buffer*, se bloquea la máquina.
- **recrc**: Una tarea pasa más de una vez por una máquina (*recirculation*).

1.2.3. Características de los objetivos

Conociendo, para **cada trabajo** a realizar, j,

- su fecha de entrega, d_j,
- la fecha de disponibilidad, r_j,
- el peso asignado correspondiente, w_j, si existe, y
- la programación en la que se inserta,

se puede calcular:

- El **tiempo** en el que se ha **finalizado** el **trabajo** (*completion time*), C_j.
- El **tiempo** que se ha **requerido** para **finalizar** éste (*flow time*), $F_j := C_j - r_j$.
- El **adelanto cometido** en el **trabajo** a realizar con respecto a su fecha de entrega (*earliness*), $E_j := \text{máx}\{d_j - C_j, 0\}$.
- El **retraso cometido** sobre el trabajo a realizar (*lateness*), $L_j := C_j - d_j$.
- La **tardanza cometida** sobre éste (*tardiness*), $T_j := \text{máx}\{L_j, 0\}$.
- La **función unitaria penalizadora** (*unit penalty*),

$$U_j := \begin{cases} 1 & \text{si } C_j > d_j \\ 0 & \text{en caso contrario} \end{cases}$$

Los **criterios** de optimización usados con mayor frecuencia consisten en la **minimización** de una **función objetivo**, **construida a partir de los anteriores conceptos**. Comúnmente, el problema consiste en **minimizar** uno de los siguientes aspectos:

- El **tiempo máximo de completación** de todas las tareas (*makespan*), $C_{\text{máx}} = \text{máx}_j\, C_j$.
- El **tiempo total de completación** de todas las tareas (ponderadamente a través de los pesos w_j), $\sum_j (w_j) C_j$
- El **máximo retraso cometido**, $L_{\text{máx}} = \text{máx}_j\, L_j$.
- La **tardanza total** (ponderadamente a través de los pesos w_j), $\sum_j (w_j) T_j$.
- El **máximo adelanto cometido**, $E_{\text{máx}} = \text{máx}_j\, E_j$.
- El **tiempo total de adelanto** (ponderadamente a través de los pesos w_j), $\sum_j (w_j) E_j$.
- El **tiempo total requerido** para **finalizar** los trabajos (ponderadamente a través de los pesos w_j), $\sum_j (w_j) F_j$.
- La **suma** (ponderada con los pesos w_j) **de adelantos y tardanzas**, $\sum_j (w_j)\big(E_j + T_j\big)$.

[VII] Espacio de espera en la que los trabajos se almacenan antes de pasar a la siguiente máquina, tras haber finalizado la operación que la máquina anterior debía realizar sobre ellos

1.3. Los nuevos sistemas de control de producción

A raíz de los nuevos desafíos que subyacen en un escenario global, complejo y volátil, ha resultado indispensable **innovar** en los modelos de producción clásicos, **dejando atrás estructuras de producción rígidas, basadas en una planificación y control central** a través de planes de producción maestros (*Dedicated manufacturing lines*, DML).

Un primer intento se produjo durante el siglo XX, creando ***fábricas ágiles*** (*Agile Manufacturing*, AM) [14]. Un término acuñado para describir a una **industria capaz de reaccionar rápidamente**

- ante **cambios** en las necesidades de los consumidores y del mercado
- manteniendo, en cada cambio, unos estándares de **calidad** y un **control** sobre los **costes** involucrados en la producción
- desarrollando y usando **procesos**, **herramientas**, **máquinas** y **pautas** flexibles, que permitan ser **adaptadas** y **reformuladas** en caso de necesidad, para crear un nuevo producto o modificar uno existente.

Y es que, para afrontar a los retos actuales, **los nuevos sistemas de producción deben** ser capaces de [18]:

- **Responder**, dinámicamente, a los **cambios** del entorno.
- **Generar** diversos modelos y **variantes** de los **productos**, a necesidad de los consumidores y del mercado.
- **Proporcionar** mayores niveles de **flexibilidad**, **robustez** e **inteligencia** en los sistemas de fabricación.

Varios modelos y paradigmas han tratado de incorporar, desde entonces, a los sistemas de producción estas características. Algunos de ellos se enumeran a continuación.

Red de fabricación flexible. *Flexible Manufacturing System* (FMS) [21]

Este sistema de fabricación se caracteriza por ser capaz de reaccionar ante cambios, previstos o imprevistos, y permitir reanudar la producción de forma prácticamente inmediata. Se sustenta en dos principios:

- Flexibilidad en las máquinas (***Machine Flexibility***), disponiendo de diferentes máquinas y herramientas capaces de realizar la misma operación sobre una pieza. Por tanto, cada tarea se puede realizar en, al menos, dos máquinas distintas. Y, así, se garantiza que el sistema pueda absorber cambios a gran escala (aumento del volumen de fabricación, averías, producción de modelos con variaciones...).
- Flexibilidad en las rutas (***Routing Flexibility***), que permite al sistema modificar el orden en el que se realizan parte de las operaciones y cubrir la capacidad de fabricación de nuevos modelos y tipos de productos.

El sistema lo componen tres elementos principales:

- Las **máquinas de trabajo**, habitualmente automatizadas.
- Un sistema de control y manejo de materiales (***Material Handling System***), conectado con las máquinas, para optimizar el flujo de los componentes.
- Un **ordenador central**, que controla los movimientos del material y el flujo de trabajo (*Machine Flow*).

Aunque tienen un **alto coste inicial** (*initial set-up cost*) y requieren de una complicada **planificación previa** (*pre-planning*), a largo plazo **reducen el coste de fabricación**, aumentan la **eficiencia**, productividad y la **calidad** de la producción y proporcionan **flexibilidad** ante cambios.

Red de fabricación reconfigurable. *Reconfigurable Manufacturing System* (RMS) [23]

Este sistema de producción se diseñó para **permitir cambios rápidos y seguros** en

- su estructura,
- sus máquinas de producción (a nivel software y hardware) y
- su capacidad de producción.

Se sustenta en dos pilares básicos:

- En el **diseño del sistema de producción**, que permita su futura **escalabilidad**, **actualización** e **integración**, tanto a **nivel estructural** (añadiendo máquinas) como a **nivel máquina** (sistemas de control modulares, software adaptable). Así, se puede absorber futuras demandas del mercado y la fabricación de nuevos productos.
- En el **diseño de las máquinas usadas** en la producción, para fabricar en ellas **familias de productos agrupables**[VIII]. Así, para cada modelo o producto dentro de la misma familia, se podrán utilizar indiferentemente una cantidad significativa de máquinas comunes (previa adaptación de su programación o componentes).

[VIII] Productos que contengan piezas, patrones, procesos o componentes comunes, diferenciados en la personalización.

Red de fabricación Holónica. *Holonic Manufacturing System* (HMS) [3], [47], [4], [12]

Un sistema de fabricación holónico constituye un nuevo (*next-generation*) método (teórico y en desarrollo) de producción altamente distribuido. Se basa en el concepto de **agentes autónomos cooperativos**, llamados **holones**. Estos son los encargados de realizar las tareas sobre las piezas, de almacenarlas, transportarlas y validarlas; almacenando y procesando, en todo caso la información necesaria para ello.

Estos holones se caracterizan por

- Disponer de **autonomía**, para crear y controlar la ejecución de su propia programación y estrategias. No requieren de un control central.
- Ser **cooperativos**, en la ejecución y desarrollo de tareas, con el resto de holones.
- Ser **inteligentes**, autoorganizándose para alcanzar los objetivos de producción.

Los holones proporcionan un sistema de producción **dinámico** y **descentralizado**, principalmente **automatizado**, aunque **integrado** con los operarios, donde es posible efectuar cambios productivos.

Fábricas fractales. *Fractal Factories* (FF) [50]

El concepto de fábrica fractal está basado en el de fractales. Es decir, son fábricas

- donde se desarrollan **procesos** o componentes con un alto grado de **similitud** (*self-similarity*) y **recursividad** (*pattern-inside-of-pattern*),
- en las que se producen aparentemente productos irregulares, pero cuya **estructura** y proceso de producción **se repite** en los diferentes componentes (*fractal entities*) que lo conforman.
- Y que están **divididas** en diferentes secciones (***fractal sections***), que se encargan de producir las *fractal entities*.

Cada fractal dispone de:

- **Autoorganización**, que implica que cada componente del sistema de fabricación tenga libertad para ejecutar y organizar las tareas a ejecutar.
- **Autooptimización**, permitiendo mejoras del proceso de producción tras cambios e influencias del entorno en el sistema de producción.
- **Coordinación** y participación entre los diferentes actores implicados en la producción, para desarrollar la producción de forma sistematizada, coherente y ordenada.

Red de fabricación distribuida. ***Distributed Manufacturing System*** **(DMS)** [18]

Una red de fabricación distribuida es una forma de producción descentralizada,

- conformada por varios **centros de producción independientes**,
- normalmente ubicados en **diferentes localizaciones** y pertenecientes a **diferentes socios comerciales** o productores,
- en los que se realiza la **fabricación** del producto final (o parte de él) **en paralelo**.

Cada factoría dispone de las herramientas y máquinas apropiadas para realizar la labor asignada, de forma independiente al resto de fábricas, que trabajan en esa u otra labor, relacionadas con la producción.

Este tipo de fabricación se circunscribe a mercados

- altamente competitivos, en el que **asumir individualmente el riesgo y los costes** de maquinaria asociados a la producción puede resultar **inviable** económicamente
- que reclaman variedad, actualizaciones y personalización de los productos, que **raramente puede ser asumido** apropiadamente **por un único centro** de producción
- en los que se desea comercializar productos complejos que requieren de **mano de obra y maquinaria altamente especializada** para producirlos (el coste y riesgo de formar y adquirir es casi inasumible y se prefiere subcontratar).

Estudios, citados en la referencia a la que se alude en este epígrafe, han demostrado que la fabricación distribuida **permite** a las empresas, en comparación con la producción en una única fábrica,

- **mejorar** la **calidad** de sus productos,
- **reducir costes** de producción, y
- **diluir** los **riesgos** de gestión asociados.

Ahora bien, su programación resulta más compleja porque se deben tomar varias decisiones:

- El **reparto** de trabajos entre las diferentes **factorías**, y
- La **programación** de los **trabajos** asignados en cada fábrica.

E incluso, llegado el caso, se debe determinar la secuencia de ensamblado en las diferentes líneas de montaje de las que se disponga.

1.4. Problemas de *Assembly Scheduling*

Una **línea de ensamblaje** o montaje final (*assembly system*) es un proceso de manufactura en donde los diferentes **componentes** producidos en la línea de **producción**, de forma **independiente**, son **ensamblados secuencialmente** para formar el bien o producto final. El ensamblado se produce mientras el ensamble va recorriendo la línea, de una **estación de trabajo** a la siguiente, con una ruta definida, donde se van agregando las diferentes partes que conforman el bien final.

Cada estación de trabajo está identificada y tiene asociada unos recursos y trabajadores, que van realizando las operaciones establecidas en esa estación. Y, en cada una de ellas, **resulta conocido de antemano el tiempo de procesamiento de las tareas asignadas**.

Indicar, por un lado, que, sobre las líneas de ensamblaje, se deben realizar, a grandes rasgos, dos tareas en cuanto a **organización de la producción**: **equilibrar la carga de trabajo**[IX] y **secuenciar las tareas a realizar**[X]. Pueden existir **relaciones de precedencia entre las diferentes estaciones de trabajo**, que, en todo caso, generarían un problema de asignación, del que no nos ocuparemos en esta introducción.

Y por otro lado, hacer explícito que, en los últimos años, el número de factorías, con sistema de producción en cadena (*mass production*), que emplean **líneas de ensamblaje final, ha aumentado significativamente** [44]. La principal razón es que permite **abaratar y reducir los costes** asociados a la producción porque:

- El movimiento de los trabajadores, herramientas y máquinas se minimiza lo máximo posible.
- Todas las partes o ensambles se transportan mecánicamente, evitando su transporte manual y el levantamiento de cargas pesadas.
- Cada trabajador realiza una operación relativamente simple, ignorando el resto de operaciones a realizar, con lo que se reduce el coste formativo y los posibles errores en el proceso de montaje.
- Resulta conocida de antemano la necesidad de nuevas piezas y componentes a ensamblar en cada estación de trabajo al ser la producción uniforme y controlable.
- La tarea de reponer componentes y partes en cada estación es más sencilla y eficaz.
- Permiten una alta mecanización y robotización en las tareas más sencillas y costosas.

Y además, permite **opciones de personalización del bien producido**, satisfaciendo una gama más amplia en las demandas del mercado [18].

[IX] Se persigue que las diferentes estaciones que componen la línea (y, por ende, los trabajadores que conforman cada estación) tengan repartida la carga de trabajo de forma equitativa.

[X] Establecer el orden en el que se pasa a ensamblar cada modelo, teniendo en cuenta el stock existente, las demandas, los trabajadores disponibles, los costes de reorganizar y preparar la línea de ensamblaje...

Las líneas de ensamblaje se pueden clasificar, atendiendo a [44], en:

- **Single Model Assembly**, donde cada modelo[XI] a manufacturar se asigna a una única línea de ensamblaje.
- **Batch Assembly**, donde se asignan a una determinada línea de montaje unas cantidades preasignadas de productos de diferentes modelos a ensamblar. Cada modelo se ensambla hasta alcanzar un nivel de stock predefinido[XII], momento en el que se cambia el modelo a montar en esa línea[XIII].
- **Mixed Model Assembly for Make-to-Stock**, donde se asignan a la misma línea de ensamblaje el montaje de varios modelos al mismo tiempo. Cada modelo se ensambla hasta alcanzar un nivel de stock predefinido mínimo.
- **Mixed Model Assembly for Make-to-Order**, que se da principalmente cuando se ofrece a los clientes la opción de personalizar el producto. Cada ensamblaje es único y, normalmente, no hay dos productos iguales, que permitan organizar un ensamblaje en cadena[XIV]. Entonces, la ruta que siguen los productos a ser ensamblados es dinámica y varía en función de las opciones elegidas por el cliente. Cuando se disponen de las piezas y componentes necesarios para ensamblar el bien final y es próxima la fecha de entrega, se programa el ensamblaje y se establece una ruta por las diferentes estaciones de trabajo que conforman la línea.
- **Postponement Assembly**, muy utilizada cuando se permite al cliente una amplia personalización de los productos en cuestiones de acabados[XV]. Como los bienes a ensamblar cuentan con una gran cantidad de partes y componentes comunes, el montaje se produce en dos etapas:
  - En una primera etapa, se ensamblan la mayor cantidad de partes comunes y se almacena el ensamblado parcial.
  - Cuando el cliente encarga el producto con su personalización y acabado único, se retoma el ensamblado, añadiendo las partes personalizadas y el acabado elegido.
- **One Station Assembly**, que ocurre cuando la línea de ensamblado está compuesta por únicamente una única estación de trabajo con un único operario, que se encarga de ensamblar un único modelo, idéntico en todos los casos. Principalmente se da en la producción artesanal.

[XI] Cada unidad del modelo manufacturado es idéntica, sin variaciones.

[XII] Este stock mínimo viene marcado por estimaciones realizadas a la demanda de los consumidores de cada modelo.

[XIII] Y, con ello, deben cambiarse los operarios, herramientas y ensambles disponibles en cada estación de trabajo de la línea para poder producir el nuevo modelo

[XIV] Hacer explícito que, por tanto, se asignan a la misma línea de ensamblaje diferentes modelos a montar.

[XV] Por ejemplo, el color de un coche y los extras, a nivel de software y conectividad, que incluye de serie.

1.5. La optimización

Resolver un **problema de optimización** consiste en **encontrar una solución** (o soluciones) óptima (o aproximadamente óptima), entre todas las posibles soluciones del problema, **que cumplen una serie de restricciones, condiciones y objetivos**.

Los **problemas de programación** (*Scheduling problems*) **son un tipo de problemas de optimización dentro de las ciencias informáticas** (*computer science*) **e investigación operativa** (*operation research*) definidos a partir de una serie de variables de decisión, restricciones y funciones objetivos. Por lo general, son **computacionalmente difíciles de resolver** exactamente por el alto tiempo requerido para encontrar una solución óptima.

El tiempo de computación requerido para resolverlos exactamente se debe, principalmente, al gran número de operaciones que deben realizar los algoritmos usados. Y es que, en la actualidad, **no** existen algoritmos, para la gran mayoría de problemas de programación, que los resuelvan exactamente en un **tiempo polinómico**.

Se detalla, a continuación, varios métodos y técnicas para resolver los problemas de optimización.

1.5.1. Métodos para obtener soluciones exactas

Existen varios métodos y herramientas que garantizan encontrar una solución óptima al problema de optimización planteado. Pero, por lo general, **resultan inviables cuando el tamaño del problema crece** porque requerirían una gran cantidad de tiempo y una potencia computacional desorbitada para hallar la solución.

Entre los principales métodos está

- La enumeración, acotada, de las posibles soluciones del problema (***Bounded Enumeration*, BE**)
- La ramificación y acotación (***Branch and Bound*, B&B**), construyendo un árbol de soluciones donde cada rama conduce a una solución posterior a la actual. El algoritmo es capaz de detectar en que momento la ramificación actual no está dando soluciones mejores, para detenerla y seguir la búsqueda en otra rama.
- La programación matemática (***Mathematical Programming*, MP**).

En este documento se usará como herramienta fundamental la programación matemática. **Los problemas de programación de la producción serán modelizados** a través de

- una **función objetivo** y
- una serie de **restricciones matemáticas**

para estudiar las relaciones internas y poder **obtener**, a partir de ellos, una **solución** del problema.

Y es que los modelos matemáticos **permiten**

- **Comprender** y entender la **situación** real que modelizan.
- **Descubrir relaciones** no aparentes inicialmente.
- Analizar el modelo matemáticamente y **sugerir estrategias** que no eran obvias al principio.
- **Experimentar cambios e innovaciones** en la manufactura sin comprometer la producción.

Son útiles por la información que proporcionan, pero se debe tener en cuenta que, en general, la **solución del modelo no es la solución al problema original y real**.

1.5.2. Métodos heurísticos

Mientras que los métodos anteriores garantizaban encontrar una solución, los métodos heurísticos, por lo general, no **encuentran** soluciones óptimas, sino **soluciones buenas, que se esperan que estén cerca de la óptima** en cuanto al coste de la función objetivo. Ahora bien, lo hacen, por lo general, en un **tiempo muy razonable con un coste computacional aceptable**.

En problemas de programación, estos métodos **consisten en reglas** de producción y ejecución, a menudo **complejas y diseñadas específicamente para un determinado tipo de problema**, con unas ciertas restricciones y objetivos concretos.

Los métodos heurísticos pueden ser:

- **Deterministas** (*Deterministic*), si cuando se ejecutan sobre el mismo problemas y los mismos datos obtienen la misma solución.
- **Estocásticos** (*Stochastic*), en caso contrario, produciendo soluciones distintas en cada ejecución.

En este documento **se incluirán**, para cada problema de producción analizado, **métodos heurísticos** que **permitan** encontrar soluciones en tiempos razonables para **organizar la producción ante cambios inesperados y no planificados del mercado**.

1.5.3. Métodos metaheurísticos

Al igual que los métodos heurísticos, los metaheurísticos proporcionan **soluciones buenas en tiempos y costes computacionales aceptables**. Pero, en cambio, **resultan aplicables a una gran variedad de problemas y situaciones**. Esto se debe a que la gran mayoría de ellos se inspiran en la naturaleza o en procesos físicos.

Se pueden clasificar en dos grupos:

- ***Single-Solution Based Metaheuristics***, que empiezan con una solución inicial y van recorriendo el espacio de soluciones, describiendo una trayectoria definida, para localizar soluciones con mejor coste.
- ***Population-Based Metaheuristics***, en los que se trabaja con un conjunto de soluciones que va evolucionando. Hay dos enfoques diferentes, en función de los métodos empleados para la evolución:
  - Los **algoritmos genéticos**, donde las soluciones de las que se disponen van cambiando y cribándose mediante *operadores mutadores*, *de cruce* y *eliminatorios*, pudiéndose obtener nuevas con mejor coste, y
  - La **optimización por enjambre de partículas**, que trabaja con una población (llamada enjambre) de soluciones candidatas (partículas). Estas partículas se desplazan a lo largo del espacio mediante una serie de reglas matemáticas (inspiradas en el comportamiento social o biológico) parametrizadas a partir de la mejor posición local y global conocida, en ese momento, en el espacio de búsqueda.

1.6. Estructura del documento

Este trabajo pretende analizar las diferentes herramientas, modelos y técnicas disponibles para planificar la producción, con todos sus detalles y de forma conjunta, en los **entornos de fabricación parcialmente y totalmente distribuidos**.

En ambos entornos, el objetivo que se perseguirá será el de minimizar el tiempo máximo dedicado a completar todas las operaciones y tareas a realizar ($C_{\text{máx}}$). Y es que el **tiempo máximo de completación es un buen indicador del rendimiento general del sistema**.

Completar lo antes posible la producción implica:

- Disponer antes del producto final para poder comercializarlo.
- Poder obtener cuanto antes beneficios.
- Acortar el tiempo que se está con el capital inmovilizado.
- Ahorrar en costes operacionales, al evitar esperas y tiempos muertos en las máquinas.

Conviene tener en cuenta que, salvo casos excepcionales, por lo general, el coste de las materias primas en la industria no suele variar periodo a periodo. Por ello, el coste, por producto final, de las materias primas que lo componen no se ven especialmente alterado a lo largo del tiempo.

Y, por su parte, el coste operativo industrial unitario (máquinas, operarios, electricidad...) también, habitualmente, permanece constante a lo largo de tiempo.

Por tanto, no parece especialmente relevante plantearse minimizar los costes de producción unitarios. Y conviene más centrarse en **aumentar la productividad, reduciendo el tiempo de producción total necesario como herramienta para reducir los costes totales**.

Por ello, **este documento está formado por dos capítulos principales**, dedicados a **analizar**, por separado, cada uno de **estos dos entornos de producción distribuidos cuando se pretende minimizar $\mathbf{C}_{\text{máx}}$**.

Así, el **capítulo 2** se ocupará de abordar la planificación de la producción cuando la fabricación está parcialmente distribuida, con varios centros de producción idénticos, emplazados en diferentes localizaciones. Es decir, abordará el estudio del **problema DPFSP**.

Y el **capítulo 3** estudiará la **integración**

- en una **red de fabricación distribuida**
- de una línea de montaje de tipo **Mixed Model Assembly for Make-to-Order**, que dispone de una única estación de trabajo con una sola máquina.

Es decir, analizará la planificación de la producción cuando la fabricación está totalmente distribuida. El **problema DAPFSP**.

La **estructura** y el esquema de contenido que se presenta en **los dos capítulos es semejante**.

Ambos capítulos **comienzan situando, histórica y conceptualmente, el problema de planificación de la producción allí tratado**. Y es que mostrar el contexto en el que se circunscriben estos problemas no es un asunto baladí.

El contexto no sólo define las posibles aplicaciones de las ideas, sino que marca la forma en las que surgen. Y, cómo tal, merece ser tenido en cuenta en este documento. Ya no sólo para ilustrar al lector, sino para facilitarle, muy posiblemente, la tarea de comprender todo lo analizado en este texto.

Las **secciones 2.1 y 3.1 enunciarán, formalmente, los problemas DPFSP y DAPFSP**, respectivamente. Mostrarán, taxativamente, las decisiones que se deben tomar en la programación de la ejecución de las tareas en los respectivos problemas. Y establecerán las características y las particularidades a las que está sujeta la programación.

Adicionalmente se ha calculado el **espacio de soluciones posibles del problema DPFSP en la sección 2.2**. La intención detrás de este cálculo es la de mostrar la complejidad computacional del problema y justificar, en cierta forma, la necesidad de desarrollar métodos heurísticos eficaces en este documento.

No se ha calculado explícitamente el espacio de soluciones del problema DAPFSP porque ya resulta evidente, a priori, que este problema requiere excesivos recursos computacionales para resolverse óptimamente.

Por su parte, las **secciones 2.3 y 3.2 detallarán varios modelos matemáticos para describir adecuadamente los problemas DPFSP y DAPFSP**. Y, **las secciones 2.4 y 3.3 se dedicarán a evaluar experimentalmente el rendimiento de estos modelos** a la hora de resolver óptimamente pequeñas instancias de los problemas de producción analizados.

Por último, estos dos capítulos finalizarán con el **estudio de métodos heurísticos y metaheurísticos para resolver, en tiempos razonables y con soluciones aceptables, los problemas DPFSP y DAPFSP**. Y es que las **secciones 2.5 (y 2.6) y 3.4 (y 3.5)** analizarán pormenorizadamente los principales métodos heurísticos y los valorarán en cuestión de rendimiento y eficacia mostrada.

El documento se cerrará con el capítulo 4, en el que se comentará brevemente:

- La importancia de este trabajo y las aplicaciones que descienden de él para con la industria.
- Las limitaciones que muestran las herramientas presentadas en este documento para modelizar la realidad.
- Y las posibles expansiones de contenido que derivan del que se presenta en este TFG.

Capítulo 2

The Distributed Permutation Flowshop Scheduling Problem

Durante el último tercio del siglo XX y la primera década del siglo XXI, un número relevante de investigadores del área de la investigación operativa centraron sus **esfuerzos en estudiar y desarrollar métodos para resolver** la planificación de la producción en problemas FlowShop con permutación (***Permutation Flowshop Scheduling Problems*, PFSP**). Un esfuerzo llevado a cabo, en parte, para satisfacer las **necesidades** de la **industria** manufacturera y química, que buscaba mejorar y optimizar sus procesos de producción [20].

La producción, en aquel entonces, se realizaba en una **misma fábrica**, en la que se asumía que cada **trabajo** era **procesado** por el **mismo conjunto de máquinas** y en el **mismo orden**. Y, aunque, a pequeña escala, la realidad hoy en día no difiere de la presentada, se da con menor frecuencia [34].

Las **grandes empresas**, que manejan grandes volúmenes de fabricación, han ido cambiando su paradigma de producción durante esta primera década. Ya no se desea que la fabricación se dé en una única fábrica, sino que **se aboga por entornos distribuidos**.

Las causas de este cambio se deben buscar en la necesidad de desarrollar un producto de mejor calidad y con menores costes de fabricación, en las que el riesgo de gestión resulte distribuido entre múltiples actores [10].

Ahora bien, **organizar la producción en este entorno resulta computacionalmente más complejo**. Y, por consiguiente, requiere de nueva literatura y nuevos estudios para organizarla óptimamente y en tiempos razonables.

En este capítulo **se formulará**, formalmente, el problema de programación de la producción en entornos distribuidos para factorías con producción FlowShop con permutación (***Distributed Permutation Flowshop Scheduling Problem*, DPFSP**). Y **se estudiarán modelos y métodos heurísticos para resolverlo** cuando se desee minimizar el *makespan* máximo de todas las fábricas involucradas.

De igual forma, se realizará una **implementación** y se reproducirá sendos **estudios comparativos**, en cuestión de **rendimiento**, de los modelos y métodos propuestos.

2.1. Formulación del problema

Formalmente, el problema de programación de la producción en entornos distribuidos para factorías con producción FlowShop con permutación (**DPFSP**) **se enuncia así**:

Se desea **programar** la **ejecución** de un conjunto de trabajos, $\mathcal{J}$[I], **en algunas** (o en todas) las **fábricas** de las que se dispone, enumeradas en el conjunto $\mathcal{F}$[II]. Esta programación está sujeta a ciertas características y particularidades:

- **Cada fábrica**, que es **idéntica al resto**, es **capaz de procesar**, en las m máquinas que dispone (listadas en el conjunto $\mathcal{M}$), **todos** los **trabajos** que se le **asigne**.
- **No** existe la posibilidad de **transferencia de trabajos entre las factorías**, es decir, una vez asignado un trabajo a una fábrica, éste no puede ser ejecutado (total o parcialmente) en otra fábrica. Y debe ser completado en la factoría de origen.
- Se asume que, al ser las plantas idénticas, el **tiempo de procesado** de cada trabajo, en cada máquina, es idéntico en todas las factorías. Es decir, este tiempo **no depende de la fábrica en la que se esté realizado el trabajo**. Y, por tanto

  $$P_{jif} = P_{ji}$$

  donde P_{jif}, que denota al tiempo de procesado del trabajo $j \in \mathcal{J}$ en la máquina $i \in \mathcal{M}$ y en la factoría $f \in \mathcal{F}$, es igual a una cierta aplicación, $P_{ji} : \mathcal{J} \times \mathcal{M} \to \mathbb{R}$, que no es función del conjunto $\mathcal{F}$.

Con respecto a la **programación de las tareas** en sí, al tratarse de una red de fabricación distribuida, consta de dos decisiones:

- Establecer el **reparto**, entre las diferentes **factorías**, de los n trabajos programados.
- Programar la **ejecución** de los **trabajos** asignados en **cada fábrica**.

El **objetivo** que perseguirá la programación de la producción del problema DPFSP será la de **minimizar el tiempo máximo de completación de todas las tareas, entre todas las fábricas** (*maximum makespan among all factories*), $C_{\text{máx}}$.

Por tanto, el problema estudiado, denotándolo adecuadamente, será

$$DF|prmu|C_{\text{máx}}$$

[I] Se supondrá que se desea programar la ejecución de n trabajos, es decir, $|\mathcal{J}| = n$.
[II] Se supondrá que, en total, se cuenta con F fábricas. Esto es $|\mathcal{F}| = F$.

2.2. Espacio de soluciones

La finalidad de esta sección es **calcular el número de posibles soluciones que tiene el problema DPFSP**. Esto es, el espacio de soluciones del problema. La sección está organizada en cuatro subsecciones principales:

- Una **primera sección**, en la que se **calcula directamente el número de posibles soluciones** del problema $DF|prmu|\cdot$, sin tener en cuenta el criterio para optimizar.
- Una **segunda sección**, en la que **se comprueba que**, efectivamente, el espacio de soluciones calculado **es demasiado extenso** para la potencia computacional actual.
- Una **tercera sección**, donde se trata de **acotar el espacio de soluciones** para el problema $DF|prmu|C_{\text{máx}}$.
- Y una **cuarta sección**, que **analiza las mejoras** en cuanto al **nuevo espacio** de soluciones del problema $DF|prmu|C_{\text{máx}}$.

2.2.1. Una primera aproximación al espacio de soluciones del problema DPFSP.

A continuación, se procederá a calcular el espacio de soluciones posibles del problema DPFSP en tres etapas o pasos, para facilitar la comprensión al lector.

Etapa 1: Establecer el número de trabajos asignados a cada factoría.

Si denotamos por f_h al número de trabajos asignados a la factoría $h \in G$, lo que se desea es **calcular es el número de soluciones enteras que tiene la ecuación**

$$f_1 + \ldots + f_F = n, \qquad \text{donde } 0 \leq f_h \leq n, \quad \forall h \in G.$$

Para ello, se construye la función generatriz

$$\varphi(t) = (1 + t + \ldots + t^n)^F$$

sobre la que se busca el coeficiente de grado n.

Empleando la fórmula de la suma de una serie geométrica y el desarrollo del binomio de Newton generalizado, se llega a que

$$\varphi(t) = (1 - t^{n+1})^F \cdot (1 - t)^{-F} = \left(\sum_{k=0}^{F} (-1)^k \binom{F}{k} t^{(n+1)k} \right) \cdot \left(\sum_{k=0}^{\infty} \binom{F+k-1}{k} t^k \right),$$

de donde se deduce[III] que el coeficiente de grado n de φ es

$$\underbrace{\binom{F}{0}}_{1} \cdot \underbrace{\binom{F+n-1}{n}}_{\binom{F+n-1}{F+n-1-n}} = \binom{F+n-1}{F-1}$$

Por tanto, **existen $\binom{\mathbf{F+n-1}}{\mathbf{F-1}}$ formas diferentes de establecer el número de trabajos asignados a cada factoría.**

[III] En el producto de los dos grandes factores que conforman a φ, la única forma de conseguir grado n es en la multiplicación del coeficiente de grado cero del primer factor con el de grado n del segundo factor

Etapa 2: Conocida la distribución de los trabajos por las diferentes factorías, determinar las formas de asignarlo a la producción.

Manteniendo la notación anterior, se supone conocido los valores de $\{f_1, \ldots, f_n\}$, es decir, la carga de trabajo asignada a cada fábrica. Así, procediendo por inducción,

- Para la primera fábrica.
  - La producción se compone de n trabajos, de los cuales f_1 están asignados a la primera fábrica.
  - Hay $\binom{n}{f_1}$ formas de elegir f_1 trabajos sobre n disponibles.
  - Hay $f_1!$ secuencias posibles en la planificación de la producción en esa fábrica, para cada elección de trabajos sobre los disponibles.
  - En total, hay $\binom{n}{f_1}\ f_1!$ formas de organizar la producción en la primera fábrica.
- Para la segunda fábrica, organizada la producción en la primera.
  - La producción se compone de n trabajos. La primera fábrica tiene asignados y organizados f_1 de esos trabajos. Y la segunda fábrica tiene asignados f_2.
  - Hay $\binom{n-f_1}{f_2}$ formas de elegir f_2 trabajos sobre $n - f_1$ disponibles.
  - Hay $f_2!$ secuencias posibles en la planificación de la producción en esa fábrica, para cada elección de trabajos sobre los disponibles.
  - En total, hay $\binom{n-f_1}{f_2}\ f_2!$ formas de organizar la producción en la segunda fábrica.
- Para la fábrica F, organizada la producción en todas las anteriores.
  - La producción se compone de n trabajos, particionados según $\{f_1, \ldots f_F\}$.
  - Hay $\binom{n-\sum_{h=1}^{F-1} f_h}{f_F}$ formas de elegir f_F trabajos sobre el resto de trabajos no asignados, $n - \sum_{h=1}^{F-1} f_h$.
  - Hay $f_F!$ secuencias posibles en la planificación de la producción en esa fábrica, para cada elección de trabajos sobre los disponibles.
  - En total, hay $\binom{n-\sum_{h=1}^{F-1} f_h}{f_F}\ f_F!$ formas de organizar la producción en la última fábrica.

Por tanto, **existen**

$$\binom{n}{f_1}\ f_1!\ \times\ \binom{n-f_1}{f_2}\ f_2!\ \times \ldots \times\ \binom{n-\sum_{h=1}^{F-1} f_h}{f_F}\ f_F! = \mathbf{n!}$$

Operando adecuadamente[IV].

formas de organizar la producción en las F fábricas, **conociendo** la **carga de trabajo asignada** a cada una de ellas.

[IV] Se ha usado la definición de número combinatorio y que $\sum_{h=1}^{F} f_h = n$, porque la producción se compone de n trabajos. Simplificando algebraicamente se obtiene el resultado.

Etapa 3: Conclusión.

En total, el **número de soluciones posibles para el problema DPFSP**, con n trabajos repartidos en F fábricas, es de

$$\binom{\mathbf{F+n-1}}{\mathbf{F-1}}\,\mathbf{n!}$$

2.2.2. Análisis del espacio de soluciones del problema DPFSP.

El siguiente paso natural es **comparar el espacio de soluciones del problema DPFSP con el problema** histórico de referencia, **PFSP**. Se puede demostrar que el número de soluciones del problema PFSP es $n!$, donde n es el número de trabajos a organizar en la fábrica.

Y, por tanto

$$\frac{\text{N.Sol. DPFSP}}{\text{N. Sol PFSP}} = \binom{F+n-1}{F-1} \tag{2.1}$$

El gráfico 2.1 muestra la relación entre el número de soluciones del problema DPFSP y del PFSP. Por ejemplo, para el caso en que se desee organizar la producción de 100 trabajos, existen $4 \cdot 10^{12}$ veces más soluciones para el problema DPFSP con 10 fábricas que con para problema PFSP.

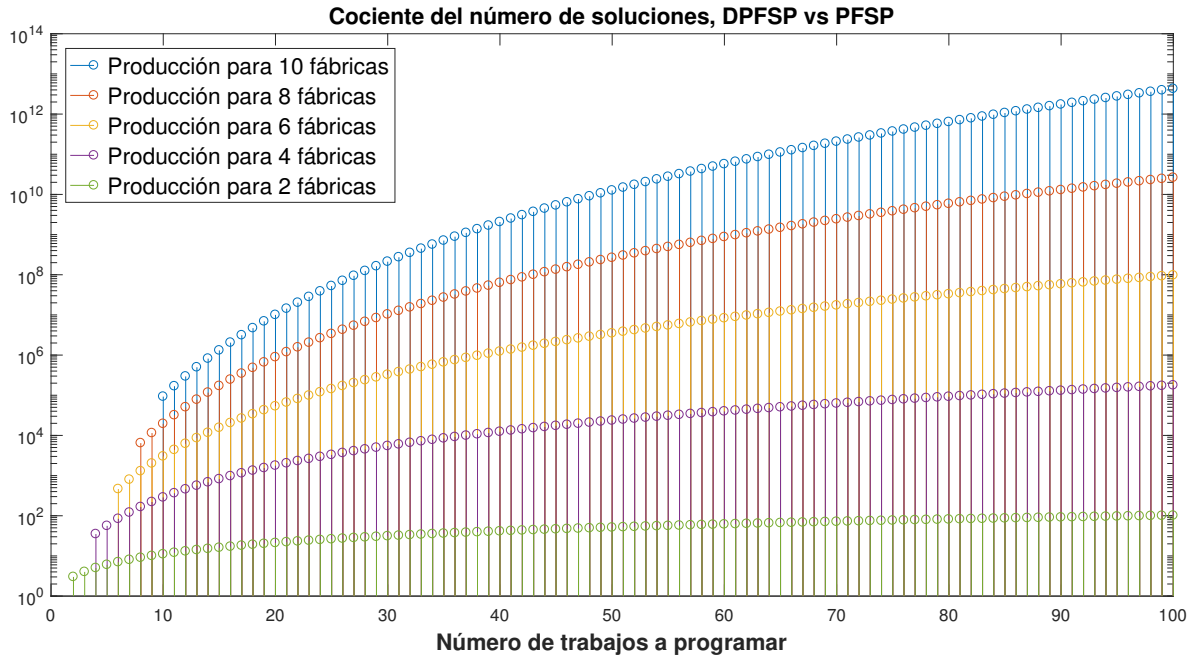


Figura 2.1: **Comparativa del espacio de soluciones de los problemas DPFSP y PFSP**. Se muestra la relación entre el número de soluciones de ambos problemas, expresada en la ecuación 2.1. Adviértase que la escala para el eje de ordenadas es logarítmica. Y respecto al eje de abscisas, ésta recoge el número de trabajos a programar. La leyenda del gráfico indica, en cada caso, el número de factorías de las que se dispone para resolver el problema DPFSP.

Además, se incluye la gráfica 2.2, que muestra el tamaño del espacio de soluciones del problema DPFSP. Por ejemplo, para el caso en que se desee organizar la producción de 100 trabajos sobre 2 fábricas, existen más de $9,4 \cdot 10^{159}$ soluciones[v] para el problema.

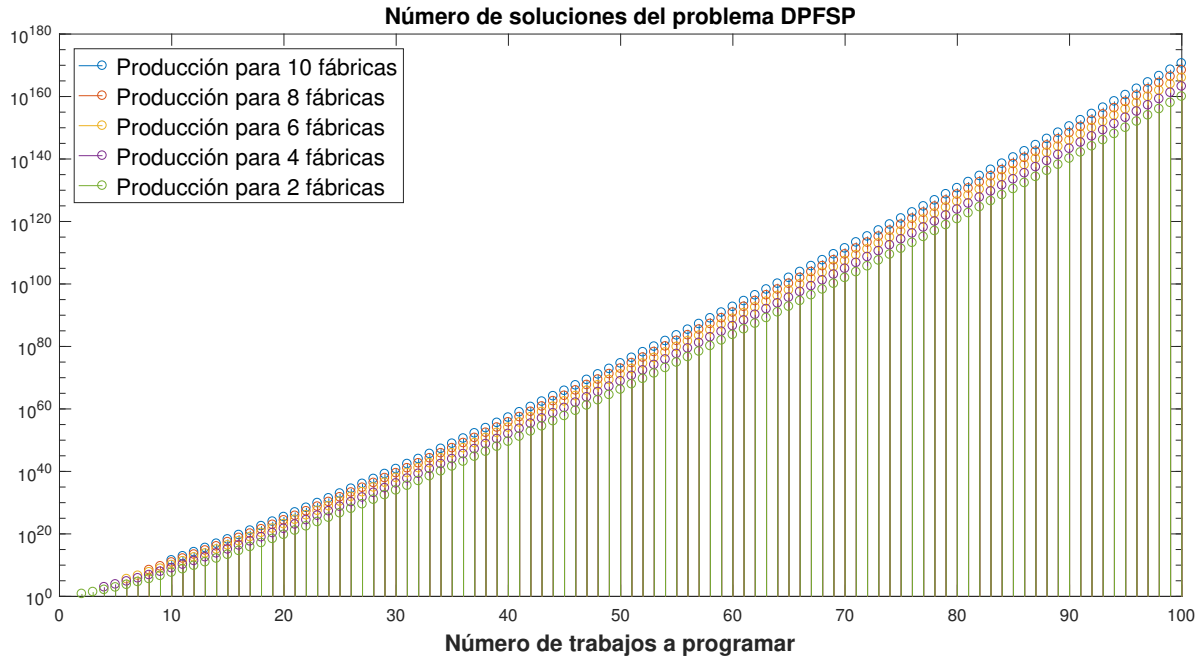


Figura 2.2: **Tamaño del espacio de soluciones del problema DPFSP**. Adviértase que la escala para el eje de ordenadas es logarítmica. Y respecto al eje de abscisas, ésta recoge el número de trabajos a programar. La leyenda del gráfico indica, en cada caso, el número de factorías de las que se dispone para resolver el problema DPFSP.

El **crecimiento exponencial** que experimenta el espacio de soluciones del problema DPFSP,

- tanto en relativo, al compararse con el problema PFSP,
- como en absoluto, al observar las variables n (número de trabajos a programar) y F (número de fábricas disponibles),

resulta **desalentador** para **resolver**, de **forma óptima** y exacta, **problemas** relativamente **grandes**.

Surge la **necesidad** de **acotar**, todavía más, el **espacio de posibles soluciones y caracterizar**, de cierta forma, las **soluciones óptimas para el criterio $\mathbf{C}_{\text{máx}}$**. El siguiente resultado lo hará posible.

[v] Aproximadamente, 9,4259e+159 soluciones

2.2.3. El espacio de soluciones del problema $DF|prmu|C_{\text{máx}}$.

Teorema 1. ***Acotación del espacio de soluciones del problema*** $DF|prmu|C_{\text{máx}}$**.**
Siempre que

- *el criterio de optimización para el problema DPFSP sea la minimización de* $C_{\text{máx}}$*, es decir, la minimización del tiempo máximo de completación de todas las tareas entre todas las fábricas, y*
- *siempre que existan más trabajos que programar en la producción que fábricas en las que realizarlos, es decir,* $n > F$

se tendrá que, en al menos una de las soluciones óptimas exactas y posibles para el problema, ***toda fábrica tiene asignado, por lo menos, un trabajo****.*

Demostración: Supongamos encontrada una solución óptima, s, en la que

$$C_{\text{máx}} := \text{máx}\{C_1, \ldots, C_x, \ldots C_y, \ldots, C_F\} = C_y,$$

denotándose por C_h el tiempo de completación máximo de la factoría h. Es decir, que en la solución óptima obtenida, la fábrica y es la que más tarda en completar las tareas que le han sido asignadas, entre las restantes fábricas.

Supongamos, además, que, en esta solución, la fábrica x no tiene asignado ningún trabajo y, por tanto, $C_x = 0$. Entonces, como $n > F$, alguna de las fábricas tiene asignado 2 o más trabajos.

Distinguimos dos casos:

- **Caso 1. La fábrica** y **tiene asignado un único trabajo.**
  - Sea h $(x \neq h \neq y)$ una factoría que tiene asignado 2 o más trabajos.
  - Y asignemos un trabajo de esa factoría h a la fábrica x. Esta nueva forma de asignar los trabajos en las diferentes fábricas es una nueva solución del problema.
  - Denotando con el superíndice * a los nuevos tiempos de completación, se tiene que

$$\begin{cases} 0 = C_x < C_x^* \leq C_h \\ \\ C_h^* \leq C_h \leq C_y \\ \\ C_\alpha = C_\alpha^* \quad \forall \alpha : h \neq \alpha \neq x \end{cases}$$

  Y, por tanto, $C_{\text{máx}}^* = C_y = C_y^*$. Esto es, la factoría y sigue siendo la que más tarda en completar las tareas que le han sido asignadas, entre las restantes fábricas.

  De ello resulta que la nueva posible solución tiene el mismo coste en la función objetivo. Y, en consecuencia, es una **solución óptima para el problema**, **en la que todas las factorías tienen asignado, al menos, un trabajo**.

- **Caso 2. La fábrica y tiene asignados dos o más trabajos.**

  - Supongamos que el trabajo j es uno de los que tiene asignado la fábrica y.
  - Así, se puede descomponer el tiempo de completación máximo en la fábrica y en dos sumandos

$$C_y = C_y^* + \delta_j^y$$

Tiempo de completación máximo en y sin el trabajo j

Efecto del trabajo j sobre el tiempo de completación máximo de la factoría y

  - Veamos que entonces s no es la solución óptima, encontrando una en la que se mejore el coste de la función objetivo.
  - Construimos la solución $\overline{s}$ a partir de s, reasignando el trabajo j de la factoría y a la factoría x. Por tanto, siguiendo la misma notación que en el caso anterior,

$$\begin{cases} 0 < C_y^* \leq C_y \\ 0 \leq \delta_j^y \leq C_y \\ 0 < C_x^* \leq C_y \\ C_\alpha = C_\alpha^* \quad \forall \alpha : y \neq \alpha \neq x \end{cases}$$

  Y, por tanto, $C^*_{\text{máx}} = \text{máx}\{C_1, \ldots, C_y^*, \ldots, C_x^*, \ldots, C_F\} < C_{\text{máx}}$.

Esto es una **contradicción** con la suposición inicial porque la solución s no es la óptima del problema.

■

A raíz de este resultado, se puede reducir el espacio de soluciones en las que buscar la solución óptima. Se procede, análogamente, como al principio de la sección, dividiendo el proceso en tres etapas:

Etapa 1: Establecer el número de trabajos asignados a cada factoría.

El teorema nos asegura que una de las soluciones óptimas al problema asigna, al menos, a cada factoría un trabajo. Por tanto, siguiendo la misma notación que anteriormente,

$$f_h \geq 1 \quad \forall h \in G.$$

Escribiendo $f'_h = f_h - 1$ y calculando el número de soluciones enteras (para las variables $0 \leq f'_h \leq n - F$) que tiene la respectiva ecuación, se llega a que **existen $\binom{\mathbf{n-1}}{\mathbf{F-1}}$ formas diferentes de establecer el número de trabajos asignados a cada factoría.**

Etapa 2: Conocida la distribución de los trabajos por las diferentes factorías, determinar las formas de asignarlo a la producción.

Como el teorema no aporta ninguna mejora en este aspecto, se mantiene que existen $n!$ **formas de organizar la producción** en las F fábricas, **conociendo** la **carga de trabajo asignada** a cada una de ellas.

Etapa 3: Conclusión.

En total, el **número de soluciones posibles para el problema DPFSP**, con n trabajos repartidos en F fábricas y **cuyo criterio óptimo es minimizar $\mathbf{C}_{\text{máx}}$**, es de

$$\binom{\mathbf{n-1}}{\mathbf{F-1}} \mathbf{n!}$$

2.2.4. Análisis del espacio de soluciones del problema $DF|prmu|C_{\text{máx}}$.

Comparamos, en esta ocasión, **el espacio de soluciones del problema DF|prmu|$C_{\text{máx}}$ con el problema DF|prmu|$\cdot$**. Y, por tanto, calculamos

$$\frac{\text{N.Sol. } DF|prmu|\cdot}{\text{N. Sol } DF|prmu|C_{\text{máx}}} = \frac{\binom{F+n-1}{F-1}}{\binom{n-1}{F-1}} \tag{2.2}$$

El gráfico 2.3 muestra la relación entre el número de soluciones del problema $DF|prmu|C_{\text{máx}}$ y del $DF|prmu|\cdot$. Por ejemplo, para el caso en que se desee organizar la producción de 100 trabajos en 10 fábricas, se ha conseguido reducir el espacio de soluciones dos veces y media[VI] al considerar el criterio de optimización $C_{\text{máx}}$.

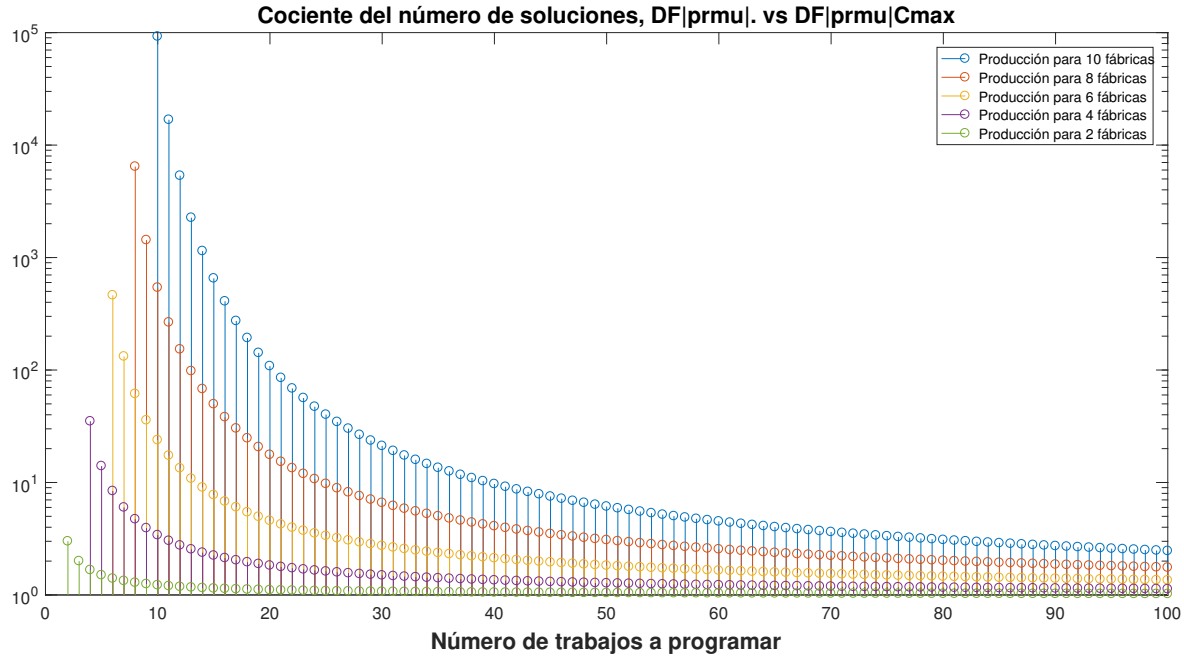


Figura 2.3: **Comparativa del espacio de soluciones de los problemas DF|prmu|$C_{\text{máx}}$ y DF|prmu|$\cdot$**. Se muestra la relación entre el número de soluciones de ambos problemas, expresada en la ecuación 2.2. Adviértase que la escala para el eje de ordenadas es logarítmica. Y respecto al eje de abscisas, ésta recoge el número de trabajos a programar. La leyenda del gráfico indica, en cada caso, el número de factorías de las que se dispone para resolver el problema.

[VI] Aproximadamente, 2,4629 veces.

Además, se incluye la gráfica 2.4, que muestra el tamaño del espacio de soluciones del problema $DF|prmu|C_{\text{máx}}$. Por ejemplo, para el caso en que se desee organizar la producción de 100 trabajos sobre 2 fábricas, existen más de $9,2 \cdot 10^{159}$ soluciones[VII] para el problema. Se ha producido una reducción del orden de 10^{158} soluciones[VIII].

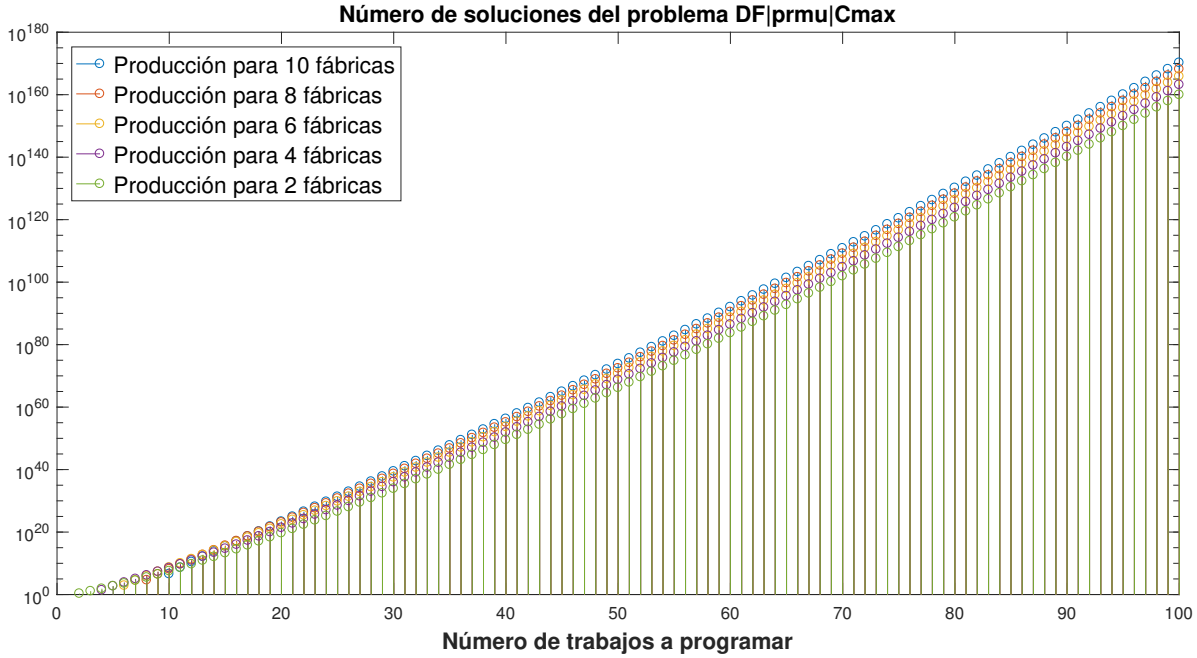


Figura 2.4: **Tamaño del espacio de soluciones del problema DPFSP**. Adviértase que la escala para el eje de ordenadas es logarítmica. Y respecto al eje de abscisas, ésta recoge el número de trabajos a programar. La leyenda del gráfico indica, en cada caso, el número de factorías de las que se dispone para resolver el problema DPFSP.

[VII] Aproximadamente, 9,2393e+159 soluciones

[VIII] Aproximadamente, 1,8665e+158 soluciones posibles menos.

2.3. Modelos matemáticos para el problema $DF|prmu|C_{\text{máx}}$

Analizado el número de posibles soluciones que tiene el problema, es evidente que los posibles **modelos** matemáticos planteados resultarán únicamente **útiles para resolver problemas** de planificación de la producción **pequeños**. Aun así, resultan **interesantes** desde un punto de vista **conceptual y** son un buen punto de partida para **conseguir**, después, métodos **heurísticos eficaces**.

Se presenta, primeramente, los parámetros, índices y notación que se empleará durante toda la sección:

Notación	**Descripción**
$\mathcal{J}$	Conjunto de trabajos que se deben producir en las factorías.
$\mathcal{M}$	Conjunto de máquinas de las que dispone cada factoría.
$\mathcal{F}$	Conjunto de fábricas disponibles.
O_{ji}	Tarea del trabajo j realizada en la máquina i
M	Una constante suficientemente grande (*Big-M*)

Índices	**Descripción**
$j, k \in \mathcal{J}$	Índices para denotar a los trabajos o la posición que ocupa cada uno de ellos en la secuencia ordenada de tareas a realizar.
$i, l \in \mathcal{M}$	Índices que denotan a las máquinas de las factorías.
$f \in \mathcal{F}$	Índices referido a las fábricas del problema.

Parámetros	**Descripción**
n	Número de tareas que deben programarse en las factorías. $\lvert\mathcal{J}\rvert = n$.
m	Número de máquinas disponibles en cada una de las factorías. $\lvert\mathcal{M}\rvert = m$.
F	Número de fábricas disponibles. $\lvert\mathcal{F}\rvert = F$.
P_{ji}	Tiempo de procesado del trabajo j en la máquina i en las factorías.

Y, a continuación, se analizarán diferentes modelos para modelizar la producción descrita por el problema $DF|prmu|C_{\text{máx}}$.

Aclarar que, en todo caso, **los modelos analizados** se enmarcarán dentro de la programación lineal entera mixta (*Mixed-Integer Lineal Problems*, **MILP**) porque:

- las **restricciones** que conformarán el problema y la **función objetivo** de éste serán **lineales**, y
- **algunas variables** de decisión del problema estarán **restringidas** a valores reales **enteros** (o a **binarios**).

2.3.1. Modelo 1: Basado en la secuenciación [34]

El primer modelo que se presenta está **basado en la secuenciación de las tareas**. En otras palabras,

- se emplean **variables binarias para asignar**, a cada **factoría**, los **trabajos** que deben realizar, y
- se usan, de nuevo, **variables binarias para marcar las relaciones de precedencia** entre los trabajos asignados, y construir, así, una secuencia de tareas a llevar a cabo en cada factoría.

Por tanto, las **variables de decisión** empleadas serán:

- $\mathbf{X_{kjf}} = \begin{cases} 1 & \text{si el trabajo } j \text{ es procesado en la factoría } f \text{ inmediatamente después del trabajo } k \\ 0 & \text{otro caso} \end{cases}$

  donde $X_{kif} \in \{0, 1\}$ y aparece definida para los subíndices $f \in \mathcal{F}$ y $k, j \in \mathcal{J} : j \neq k$.

- $\mathbf{Y_{jf}} = \begin{cases} 1 & \text{si el trabajo } j \text{ se procesa en la factoría } f \\ 0 & \text{otro caso} \end{cases}$

  donde $Y_{jf} \in \{0, 1\}$ y aparece definida para los subíndices $f \in \mathcal{F}$ y $j \in \mathcal{J}$.

- $\mathbf{C_{ji}} \in \mathbb{R}^+$, que es el tiempo de completación del trabajo $j \in \mathcal{J}$ en la máquina $i \in \mathcal{M}$. Es decir, el tiempo que ha estado procesándose el trabajo j en la máquina i, junto con el tiempo previo acumulado por el trabajo j en otras máquinas y en esperas.

- $\mathbf{C}_{\text{máx}} \in \mathbb{R}^+$, que es el tiempo de completación de todas las tareas, entre todas las fábricas.

Luego, para planificar la producción de n trabajos en F fábricas con m máquinas cada una, este modelo requerirá

- $\mathbf{nm + 1}$ variables reales (*continuous variable*, **CVs**)
- $n(n-1)F + nF = \mathbf{n^2F}$ variables binarias (*binary variables*, **BVs**)

Además, por razones lógicas (predecesor al primer trabajo), se hace necesario establecer un ***dummy job***, un trabajo ficticio que **preceda al primer trabajo real procesado en cada factoría**, que se denotará por el subíndice 0. Es por ello que, si se emplea el subíndice k para representar las posiciones que ocuparán los trabajos en cada fábrica, k deberá comenzar en cero para albergarlo.

Asimismo, el **problema** de producción está **sujeto a una serie de restricciones**, limitaciones y condiciones lógicas que deben verificarse. A saber:

a) El **dominio de las variables** involucradas.

$$\begin{cases} C_{j,i} \geq 0 & \forall j, i \\ Y_{j,f} \in \{0,1\} & \forall j, f \\ X_{k,j,f} \in \{0,1\} & \forall k, j, i : j \neq k \\ C_{\text{máx}} \geq 0 \end{cases}$$

b) **Cada trabajo debe estar asignado exactamente a una fábrica**. Esto es,

$$\sum_{f=1}^{F} Y_{j,f} = 1 \qquad \forall j$$

c) **Cada trabajo debe aparecer en una única posición de la secuencia de producción**, asociada a la fábrica donde se realiza. Por tanto,

$$\sum_{\substack{k=0 \\ j \neq k}}^{n} \sum_{f=1}^{F} X_{k,j,f} = 1 \qquad \forall j$$

d) **Si** un **trabajo** (*real*) **no** está asociado a una **fábrica**, éste **no puede estar** asignado a **su secuencia de producción**, es decir, no puede ser predecesor ni sucesor de ningún trabajo en esa fábrica.

Traducido en variables, esto significa que, dado un trabajo j y una factoría f,

$$Y_{jf} = 0 \ \rightarrow \ \begin{cases} X_{k,j,f} = 0 & \forall k : k \neq j \\ X_{j,k,f} = 0 & \forall k : k \neq j \end{cases}$$

Y expresado en forma de restricción, teniendo en cuenta el dominio de $X_{\cdot,\cdot,\cdot}$,

$$\sum_{\substack{k=1 \\ j \neq k}}^{n} (X_{k,j,f} + X_{j,k,f}) \leq 2 \cdot Y_{j,f} \qquad \forall j, f$$

e) **Definición de la variable $\mathbf{X}_{\cdot,\cdot,\cdot}$** con respecto a la **relación de precedencia inmediata**.

$$\sum_{\substack{j=1 \\ j \neq k}}^{n} \sum_{f=1}^{F} X_{k,j,f} \leq 1 \qquad \forall k \in \{1, \ldots, n\}$$

f) Los ***dummy jobs* tienen un sucesor en la secuencia de cada fábrica**.

$$\sum_{j=1}^{n} X_{0,j,f} = 1 \qquad \forall f$$

g) **Dados dos trabajos** asignados a una fábrica, o **no hay una relación** de precedencia inmediata entre ellos **o uno de los dos precede inmediatamente al otro**. No se puede preceder a un trabajo y seguirlo a la vez.

$$\sum_{f=1}^{F} (X_{k,j,f} + X_{j,k,f}) \leq 1 \qquad \forall k \in \{1, \ldots, n-1\},\ \forall j > k$$

h) **Definición del tiempo máximo de completación** de todas las tareas entre todas las fábricas.

El tiempo máximo de completación del problema, que denotamos por $C_{\text{máx}}$, es el máximo entre los tiempo de completación de los trabajos al acabar todo su procesado (es decir, tras pasar por la última máquina, m), $C_{\cdot,m}$.

Así, si se establece la siguiente desigualdad,

$$C_{\text{máx}} \geq C_{j,m} \qquad \forall j,$$

como el problema de optimización tiene como objetivo minimizar el valor de $C_{\text{máx}}$, la anterior expresión se cumplirá sin holgura para algún trabajo. Y, por consiguiente, $C_{\text{máx}}$ será, efectivamente, el tiempo máximo de completación global.

i) **Definición de completación** para cada O_{ji}.

Se establecen dos desigualdades, que, separadamente, acotan el tiempo de completación inferiormente. Pero combinadas y teniendo en cuenta la relación previa, h), y que se está minimizando $C_{\text{máx}}$, definen correctamente el tiempo de completación para cada tarea y para máquina de la ruta.

Nótese, antes de comenzar, que se define $C_{j,0} := 0\ \forall j$.

Primera Desigualdad

$$C_{j,i} \geq C_{j,i-1} + P_{j,i} \qquad \forall j, i$$

El tiempo de completación de la tarea j en la máquina i

Debe ser superior o igual a...

El tiempo de procesamiento de j en i

Más

El tiempo de completación acumulado en la ruta

Segunda Desigualdad

$$C_{j,i} \geq C_{k,i} + P_{j,i} + M\left(\left(\sum_{f=1}^{F} X_{k,j,f}\right) - 1\right) \qquad \forall k \in \{1,\ldots,n\},\ \forall j,i : j \neq k$$

Analicémosla por casos

- **Si en cierta factoría f**, **el trabajo k precede al trabajo j**, entonces

  $$X_{k,j,f} = 1$$

  Y, junto con las restricciones a) y c), se establece que $\sum_{f=1}^{F} X_{k,j,f} = 1$.

  Por tanto, la desigualdad se traduce en que

  $C_{j,i} \geq C_{k,i} + P_{j,i}$

  - $C_{j,i}$: El tiempo de completación de la tarea j en la máquina i
  - $\geq$: Debe ser superior o igual a...
  - $C_{k,i}$: El tiempo de completación en la máquina i del trabajo previo a j, k
  - $+$: Más
  - $P_{j,i}$: El tiempo de procesamiento de j en i

- **Si en ninguna factoría el trabajo k precede al trabajo j**, la restricción propuesta equivale a

  $$C_{j,i} \geq C_{k,i} + P_{j,i} - M$$

  Como, en este caso, la restricción no debe modelizar ninguna condición lógica, es necesario anularla. Tomando M suficientemente grande, se establece que $C_{j,i} \geq 0$.

Volviendo de nuevo a analizar el impacto a nivel computacional de este modelo, el **número de restricciones requeridas** para modelizar la planificación de n trabajos, en F fábricas con m máquinas cada una, es

$$\mathbf{4n + nF + F + n^2m + \frac{1}{2}n(n-1)}$$

Este número se obtiene a partir de la siguiente tabla, en la que se calcula, para cada expresión, la cantidad de restricciones que genera.

Restricción	**Cantidad**	**Restricción**	**Cantidad**
a)	0	f)	F
b)	n	g)	$n(n-1)/2$
c)	n	h)	n
d)	nF	i) - 1)	nm
e)	n	i) - 2)	$nm(n-1)$

2.3.2. Modelo 2: Modificación de un modelo de la familia de Wagner para PFSP, basado en las posiciones que ocupan los trabajos y adaptado a un entorno multifábrica [34]

Los **modelos** de la familia de **Wagner** son, posiblemente, unos de los mejores modelos para la resolución del problema **PFSP**, en cuanto a que presentan un **alto rendimiento** en la búsqueda de soluciones ([41], [45]). En parte, esto se debe a que la modelización **no** necesita **incluir** una ***Big-M*** para establecer ciertas condiciones lógicas, **evitando**, así, que se produzcan **relajaciones lineales extremadamente pobres** en la resolución del problema.

Por consiguiente, resulta razonable pensar que funcionarán relativamente bien en la resolución de problemas DPFSP, modificándolos adecuadamente para incluir la producción distribuida.

El **modelo** que se presenta a continuación es

- una **adaptación** a un entorno **multifábrica** de las ideas subyacentes en los modelos de **Wagner**,
- en el que se usan las **posiciones** que ocuparían **los trabajos** en la secuencia de producción de cada fábrica.

Por un lado, las **variables de decisión** empleadas serán:

- $\mathbf{X_{j,k,f}} = \begin{cases} 1 & \text{si el trabajo } j \text{ ocupa la posición } k \text{ en la factoría } f \\ 0 & \text{otro caso} \end{cases}$
- $\mathbf{I_{i,k,f}} \in \mathbb{R}^+$, que es el tiempo que la máquina i permanece desocupada tras finalizar el trabajo que ocupa la posición k en la factoría f, hasta que empieza el siguiente, el que ocupa la posición $k+1$ (*Idle time*). Nótese que $k < n$.
- $\mathbf{W_{i,k,f}} \in \mathbb{R}^+$, que es el tiempo que transcurre desde que el trabajo que ocupa la posición k en la factoría f sale de la máquina i y entra en la máquina $i+1$ (*Waiting time*). Nótese que $i < m$.
- $\mathbf{C_f} \in \mathbb{R}^+$, que es el tiempo de completación de todas las tareas (*makespan*) en la fábrica f.
- $\mathbf{C_{máx}} \in \mathbb{R}^+$, que es el tiempo de completación de todas las tareas, entre todas las fábricas.

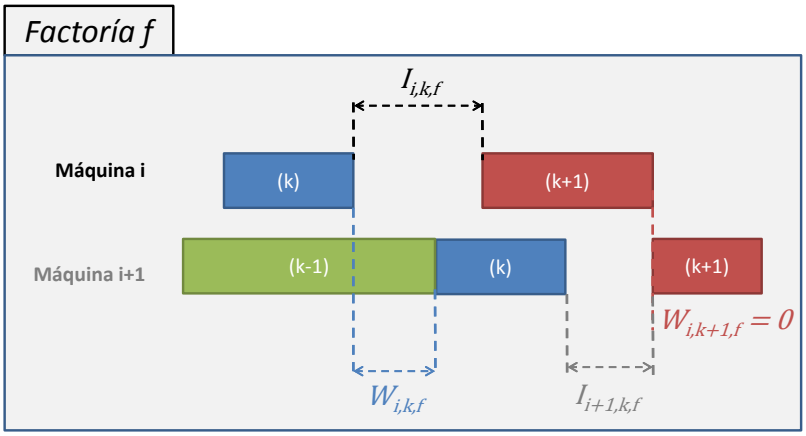


Figura 2.5: Ejemplo de representación de la modelización llevada a cabo en la planificación de la producción en la fábrica f usando un diagrama de Gantt. Se han marcado el valor de las variables $W_{\cdot,\cdot,f}$ y $I_{\cdot,\cdot,f}$ para facilitar la comprensión de ambas variables.

Y, entonces, para planificar la producción de n trabajos en F fábricas con m máquinas cada una, este modelo requerirá

- $\mathbf{F(1+2nm-n-m)+1}$[IX] variables reales (**CVs**)
- $\mathbf{n^2F}$ variables binarias (**BVs**)

Por otro lado, el **problema** de producción, que está **sujeto a una serie de restricciones**, limitaciones y condiciones lógicas que deben verificarse, se modeliza así:

a) El **dominio de las variables** involucradas.

$$\begin{cases} I_{i,k,f} \geq 0 & \forall i,k,f \ : \ k < n \\ W_{i,k,f} \geq 0 & \forall i,k,f \ : \ i < m \\ X_{j,k,f} \in \{0,1\} & \forall k,j,f \\ C_{\text{máx}} \geq 0 & \end{cases}$$

b) **Todo trabajo tiene asignada una factoría** en la que realizarse **y una posición** dentro del sistema de producción global. Esto es,

$$\sum_{k=1}^{n}\sum_{f=1}^{F} X_{j,k,f} = 1 \qquad \forall j$$

c) **Toda posición** del sistema de producción global está ocupada por **un trabajo** y asignada a **una factoría**.

$$\sum_{j=1}^{n}\sum_{f=1}^{F} X_{j,k,f} = 1 \qquad \forall k$$

d) **Relación entre las variables $\mathbf{X}_{\cdot,\cdot,\cdot}$, $\mathbf{I}_{\cdot,\cdot,\cdot}$ y $\mathbf{W}_{\cdot,\cdot,\cdot}$**

Para estudiar esta relación, se utilizará el operador $\Delta_{i,k,f}$

$\Delta_{i,k,f} :=$ Diferencia entre el tiempo en el que el trabajo $(k+1)$-ésimo empieza en la máquina $i+1$ y el tiempo en el que el trabajo (k)-ésimo acaba en la máquina i

Gráficamente se razonará las siguientes igualdades[X]:

$$\begin{cases} \Delta_{i,k,f} = W_{i,k+1,f} + P_{(k+1),i} + I_{i,k,f} \\ \Delta_{i,k,f} = W_{i,k,f} + P_{(k),i+1} + I_{i+1,k,f} \end{cases}$$

comprobando que, efectivamente, coinciden ambos términos de la igualdad. Y, por tanto, se tendrá que

$$I_{i,k,f} - W_{i,k,f} + \sum_{j=1}^{n} X_{j,k+1,f}P_{j,i} \ = \ I_{i+1,k,f} - W_{i,k+1,f} + \sum_{j=1}^{n} X_{j,k,f}P_{j,i+1} \qquad \forall k,i,f \ : \ k < n, i < m$$

[IX] $F(1+2nm-n-m)+1 = m(n-1)F + (m-1)nF + F + 1$ variables reales

[X] Se utiliza la notación (k) para referirse al trabajo que ocupa la posición k-ésima en la secuencia.

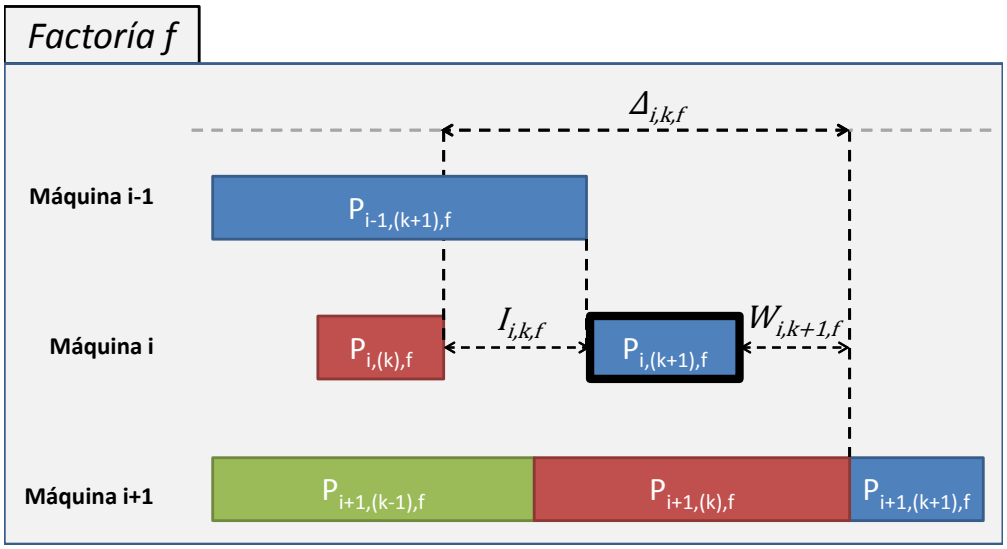


Figura 2.6: $\Delta_{i,k,f}$ se puede descomponer en tres sumandos: en $I_{i,k,f}$, en $P_{i,(k+1),f}$ y en $W_{i,k+1,f}$. Dicho de otra forma,

- En el tiempo que la máquina i, de la fábrica f, permanece desocupada tras finalizar el trabajo que ocupa el lugar k, hasta que empiece a procesar el siguiente, $(k+1)$.
- En el tiempo de procesamiento del trabajo $(k+1)$ en la máquina i, de la fábrica f.
- Y en el tiempo que transcurre para el trabajo $(k+1)$ desde que acaba en la máquina i y entra a la máquina $i+1$.

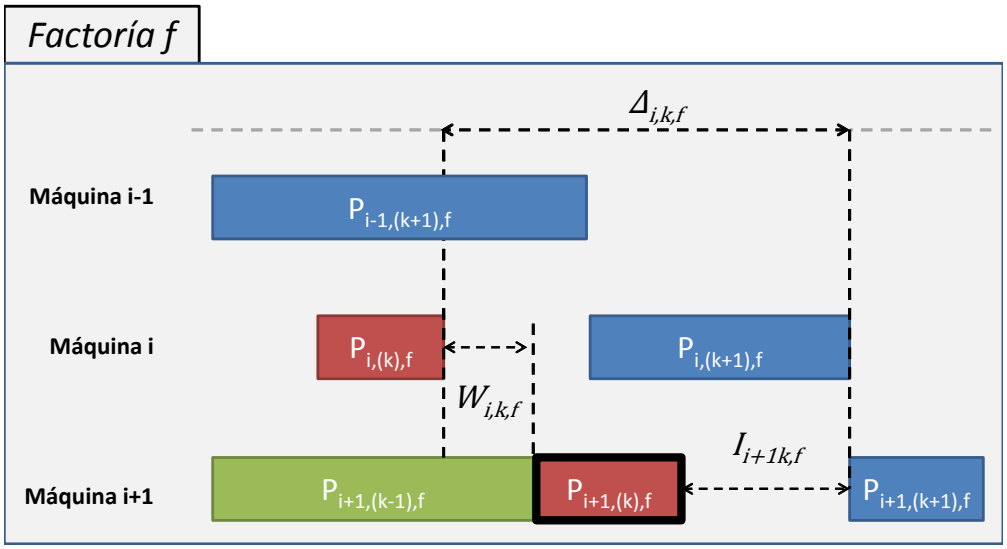


Figura 2.7: $\Delta_{i,k,f}$ se puede descomponer, de nuevo, en tres sumandos: en $W_{i,k,f}$, en $P_{i+1,(k),f}$ y en $I_{i+1,k,f}$. Dicho de otra forma,

- En el tiempo que transcurre para el trabajo (k) desde que acaba en la máquina i y entra a la máquina $i+1$.
- En el tiempo de procesamiento del trabajo (k) en la máquina $i+1$, de la fábrica f.
- Y en el tiempo que la máquina $i+1$, de la fábrica f, permanece desocupada tras finalizar el trabajo (k), hasta que empiece a procesar el siguiente, $(k+1)$.

Destáquese, en todo caso, que

$$\begin{cases} I_{1,k,f} = 0 & \forall k, f : k < n \\ W_{i,1,f} = 0 & \forall i, f : i < m \end{cases}$$

e) **Definición del tiempo máximo de completación**.

El tiempo máximo de completación del problema, que denotamos por $C_{\text{máx}}$, es el máximo entre los tiempos de completación de cada factoría. Y, de ahí que

$$C_{\text{máx}} \geq C_f \qquad \forall f$$

Como el problema de optimización tiene como objetivo minimizar el valor de $C_{\text{máx}}$, la anterior expresión se cumplirá sin holgura para alguna fábrica. Y, por consiguiente, $C_{\text{máx}}$ será, efectivamente, el tiempo máximo de completación global.

f) **Definición de completación** para cada fábrica.

El **tiempo de completación de una fábrica**, f, se denota por C_f. Y es el **tiempo** en el que el **último trabajo** que se debe procesar, en esta factoría, **termina** su **recorrido** por las diferentes máquinas **y sale de la línea de producción**.

Pero, como todos los trabajos siguen la misma ruta[XI] y acaban en la máquina m, el **tiempo de completación de cada fábrica se puede obtener observando los tiempos en** esta **última máquina**.

$$C_f = \underbrace{\sum_{i=1}^{m-1} \underbrace{\sum_{j=1}^{n} X_{j,1,f} P_{j,i}}_{= P_{i,(1),f}\ \text{[XII]}}}_{\text{Suma de los tiempos de procesamiento en todas las máquinas de la ruta, salvo la última, para el 1er trabajo ejecutado en la fábrica } f\ \text{[XIV]}.} + \underbrace{\sum_{k=1}^{n-1} I_{m,k,f}}_{\text{Tiempo en el que la máquina } m\text{, de la factoría } f\text{, permanece desocupada, tras procesar su 1er trabajo.}} + \underbrace{\sum_{k=1}^{n} \underbrace{\sum_{j=1}^{n} X_{j,k,f} P_{j,m}}_{= P_{i,(k),f}\ \text{[XIII]}}}_{\text{Suma de los tiempos de procesamiento en la máquina } m \text{ de todos los trabajos ejecutados en la fábrica } f.} \qquad \forall f$$

[XI] Se está estudiando la planificación de la producción en entornos distribuidos para factorías con producción FlowShop con permutación.

[XII] Tiempo de procesamiento en la máquina i del 1er trabajo ejecutado en la fábrica f.

[XIII] Tiempo de procesamiento del trabajo (k), ejecutado en la máquina m de la factoría f.

[XIV] Este tiempo es el momento en el que la máquina m empieza a procesar el primer trabajo de la fábrica. Durante todo el periodo anterior, la máquina ha estado inactiva y no ha procesado ningún trabajo.

Volviendo de nuevo a analizar el impacto a nivel computacional de este modelo, el **número de restricciones requeridas** para modelizar la planificación de n trabajos, en F fábricas con m máquinas cada una, es

$$\mathbf{2n + (n - 1)(m - 1)F + 2F}$$

Este número se obtiene a partir de la siguiente tabla, en la que se calcula, para cada expresión, la cantidad de restricciones que genera.

Restricción	**Cantidad**	**Restricción**	**Cantidad**
a)	0	d)	$(n-1)(m-1)F$
b)	n	e)	F
c)	n	f)	F

2.3.3. Modelo 3: Basado en las posiciones que ocupan los trabajos [34]

El **modelo** analizado **anteriormente requería un número excesivo de variables reales** para modelizar la situación de producción. Y, aunque éste no incluyese una *Big-M* que pueda causar relajaciones lineales pobres, la resolución del modelo puede verse truncada por **requerir demasiado tiempo y** consumir excesivos **recursos** computacionales.

El **modelo** que **se presenta a continuación**

- **usa**, de nuevo, las **posiciones que ocuparían los trabajos** en la secuencia de producción de cada fábrica,
- **reduce el número de variables continuas** a emplear, en comparación con el modelo 2.3.2 y
- **evita**, de nuevo, el uso de una ***Big-M***.

Se trata de un modelo híbrido, que **combina los modelos 2.3.1 y 2.3.2**, construido tomando las ideas más ventajosas de cada uno de ellos para **mejorar la resolubilidad del modelo final**.

Las **variables de decisión** que se emplearán serán:

- $\mathbf{X_{j,k,f}} = \begin{cases} 1 & \text{si el trabajo } j \text{ ocupa la posición } k \text{ en la factoría } f \\ 0 & \text{otro caso} \end{cases}$

  que se toma del modelo 2.3.2.

- $\mathbf{C_{k,i,f}} \in \mathbb{R}^+$, que es el tiempo de completación en la máquina i de la tarea que ocupa la posición k en la secuencia de producción de la fábrica f. Es una adaptación al tiempo de completación definido en el modelo 2.3.1.

- $\mathbf{C}_{\text{máx}} \in \mathbb{R}^+$, que es el tiempo de completación de todas las tareas, entre todas las fábricas.

Así, para planificar la producción de n trabajos en F fábricas con m máquinas cada una, este modelo requerirá

- $\mathbf{n^2F}$ variables binarias (**BVs**), cifra que permanece inalterada con respecto a 2.3.2, y
- $\mathbf{nmF + 1}$ variables reales (**CVs**), contra las $F(1 + 2nm - n - m) + 1$ que necesitaba 2.3.2.

Y las **restricciones**, limitaciones y condiciones lógicas que deben verificarse son:

a) El **dominio de las variables** involucradas.

$$\begin{cases} X_{j,k,f} \in \{0,1\} & \forall j,k,f \\ C_{k,i,f} \geq 0 & \forall k,i,f \\ C_{\text{máx}} \geq 0 \end{cases}$$

b) **Todo trabajo tiene asignada una factoría** en la que realizarse **y una posición** dentro del sistema de producción global. Tomando del modelo 2.3.2, esto es,

$$\sum_{k=1}^{n}\sum_{f=1}^{F} X_{j,k,f} = 1 \qquad \forall j$$

c) **Toda posición** del sistema de producción global está ocupada por **un trabajo** y asignada a **una factoría**. Tomando del modelo 2.3.2,

$$\sum_{j=1}^{n}\sum_{f=1}^{F} X_{j,k,f} = 1 \qquad \forall k$$

d) **Definición del tiempo máximo de completación** de todas las tareas entre todas las fábricas.

El tiempo máximo de completación del problema, que denotamos por $C_{\text{máx}}$, es el máximo entre los tiempo de completación de los trabajos al acabar todo su procesado (es decir, tras pasar por la última máquina, m), en la fábrica en la que han sido asignados, $C_{\cdot,m,\cdot}$.

Así, si se establece la siguiente desigualdad,

$$C_{\text{máx}} \geq C_{k,m,f} \qquad \forall k,f$$

como el problema de optimización tiene como objetivo minimizar el valor de $C_{\text{máx}}$, la anterior expresión se cumplirá sin holgura para algún trabajo en alguna fábrica. Y, por consiguiente, $C_{\text{máx}}$ será, efectivamente, el tiempo máximo de completación global.

e) **Definición de completación** para cada O_{ji}, en cada fábrica.

Análogamente a como se hacía en el modelo 2.3.1, se establecerán dos desigualdades. Separadamente, acotarán el tiempo de completación inferiormente. Pero combinadas y teniendo en cuenta se está minimizando $C_{\text{máx}}$, definirán correctamente el tiempo de completación para cada tarea y para máquina de la ruta en cada fábrica.

Estas dos desigualdades son adaptaciones a las respectivas restricciones del modelo 2.3.1 para hacerlas compatibles con un modelo basado en las posiciones.

Nótese, antes de comenzar, que se define $C_{k,0,f} := 0 \ \forall k,f$.

Primera Desigualdad

El procesado en la máquina i del trabajo que ocupa la posición k en la factoría f **solo puede comenzar cuando** este trabajo **ha acabado** su **procesamiento en** las **máquinas anteriores**.

$$C_{k,i,f} \geq C_{k,i-1,f} + \underbrace{\sum_{j=1}^{n} X_{j,k,f} P_{j,i}}_{} \qquad \forall k, i, f$$

El tiempo de completación de la tarea (k) en la máquina i de la fábrica f

Debe ser superior o igual a...

El tiempo de completación acumulado en la ruta

Más

El tiempo de procesamiento de (k) en i en f

Segunda Desigualdad

El procesado en la máquina i del trabajo que ocupa la posición k en la factoría f **solo puede comenzar cuando la máquina está libre**. Esto es, si la máquina i ya ha procesado el trabajo que ocupa la posición $k-1$ en la secuencia de producción de la fábrica f.

$$C_{k,i,f} \geq C_{k-1,i,f} + \underbrace{\sum_{j=1}^{n} X_{j,k,f} P_{j,i}}_{} \qquad \forall k, i, f \ : \ k > 1$$

El tiempo de completación de la tarea (k) en la máquina i de la fábrica f

Debe ser superior o igual a...

El tiempo de completación del trabajo previo, $(k-1)$, en la máquina i de la factoría f

Más

El tiempo de procesamiento de (k) en i en f

Volviendo de nuevo a analizar el impacto a nivel computacional de este modelo, el **número de restricciones requeridas** para modelizar la planificación de n trabajos, en F fábricas con m máquinas cada una, es

$$\mathbf{2n + F\big(2nm - m + n\big)}$$

Este número se obtiene a partir de la siguiente tabla, en la que se calcula, para cada expresión, la cantidad de restricciones que genera.

Restricción	**Cantidad**	**Restricción**	**Cantidad**
a)	0	d)	nF
b)	n	e) - 1)	nmF
c)	n	e) - 2)	$(n-1)mF$

2.3.4. Modelo 4: Basado en las posiciones que ocupan los trabajos, disgregando la secuenciación de producción de la asignación de factoría [34]

Aunque el **modelo previo**, 2.3.3, reducía el número de variables continuas con respecto a 2.3.2, éste todavía **podría requerir excesivo tiempo y recursos** para resolverse.

Un **segundo intento** para reducir el número de variables implicadas en el modelo pasa por **separar las dos decisiones que se deben tomar**, con respecto a la programación de las tareas. A saber:

- Establecer el reparto, entre las diferentes factorías, de los n trabajos programados, y
- programar la ejecución de los trabajos asignados en cada fábrica.

Por consiguiente, este cuarto **modelo**

- seguirá basándose en las **posiciones que ocuparían los trabajos** en la secuencia de producción de cada fábrica,
- pero definirá **dos conjuntos de variables binarias diferenciadas**[XV], que, por separado, establecerán el reparto de trabajos entre fábricas y la secuencia de producción en cada una de ellas.

De esta forma, el número de posibles posiciones que pueden ocupar los trabajos pasará de nF (en 2.3.2 y en 2.3.3) a n. Pero, en cambio, el número de restricciones requeridas será mucho mayor e implicará hacer uso, de nuevo, de una *Big-M*.

Las **variables de decisión** que se emplearán serán:

- $\mathbf{X_{j,k}} = \begin{cases} 1 & \text{si el trabajo } j \text{ ocupa la posición } k \text{ en la secuencia de producción} \\ 0 & \text{otro caso} \end{cases}$
- $\mathbf{Y_{k,f}} = \begin{cases} 1 & \text{si el trabajo que está en la posición } k \text{ se procesa en la factoría } f \\ 0 & \text{otro caso} \end{cases}$

  que se toma y se adapta del modelo 2.3.1. Allí se determinaba la factoría en la que se producía cada trabajo. Aquí se determinará la fábrica a la que está asociada cada posición de la secuencia de producción.
- $\mathbf{C_{k,i}} \in \mathbb{R}^+$, que es el tiempo de completación en la máquina i de la tarea que ocupa la posición k en la secuencia de producción.
- $\mathbf{C_{máx}} \in \mathbb{R}^+$, que es el tiempo de completación de todas las tareas, entre todas las fábricas.

Así, para planificar la producción de n trabajos en F fábricas con m máquinas cada una, este modelo requerirá

- $\mathbf{n^2 + nF}$ variables binarias (**BVs**) y
- $\mathbf{nm + 1}$ variables reales (**CVs**).

[XV] En los modelos 2.3.2 y 2.3.3, las dos decisiones estaban unidas en una única variable binaria con tres índices, $X_{j,k,f}$.

Y las **restricciones**, limitaciones y condiciones lógicas que deben verificarse son:

a) El **dominio de las variables** involucradas.

$$\begin{cases} X_{j,k} \in \{0,1\} & \forall j,k \\ Y_{k,f} \in \{0,1\} & \forall k,f \\ C_{k,i} \geq 0 & \forall k,i \\ C_{\text{máx}} \geq 0 & \end{cases}$$

b) **Todo trabajo ocupa exactamente una posición** en la secuencia de producción. Esto es,

$$\sum_{k=1}^{n} X_{j,k} = 1 \qquad \forall j$$

c) **Cada posición** en la secuencia de producción se asigna una sola vez. En concreto,

$$\sum_{j=1}^{n} X_{j,k} = 1 \qquad \forall k$$

d) **Cada trabajo se asigna a una sola fábrica**, es decir,

$$\sum_{f=1}^{F} Y_{k,f} = 1 \qquad \forall k$$

e) **Definición del tiempo máximo de completación** de todas las tareas entre todas las fábricas.

El tiempo máximo de completación del problema, que denotamos por $C_{\text{máx}}$, es el máximo entre los tiempo de completación de los trabajos (que ocuparán una determinada posición en la secuenciación), al acabar todo su procesado (es decir, tras pasar por la última máquina, m), $C_{\cdot,m}$.

Así, si se establece la siguiente desigualdad,

$$C_{\text{máx}} \geq C_{k,m} \qquad \forall k,$$

como el problema de optimización tiene como objetivo minimizar el valor de $C_{\text{máx}}$, la anterior expresión se cumplirá sin holgura para alguna posición de la secuencia de producción. Y, por consiguiente, $C_{\text{máx}}$ será, efectivamente, el tiempo máximo de completación global.

f) **Definición de completación** para cada O_{ji}.

Análogamente a como se hacía en el modelo 2.3.1, se establecerán dos desigualdades. Separadamente, acotarán el tiempo de completación inferiormente. Pero combinadas y teniendo en cuenta se está minimizando $C_{\text{máx}}$, definirán correctamente el tiempo de completación para cada tarea y para máquina de la ruta en cada fábrica.

Estas dos desigualdades son adaptaciones a las respectivas restricciones del modelo 2.3.3 para hacerlas compatibles con la disgregación de las decisiones.

Nótese, antes de comenzar, que se define $C_{k,0} := 0 \ \forall k$.

Primera Desigualdad

El procesado en la máquina i del trabajo que ocupa la posición k en la factoría f **sólo puede comenzar cuando** este trabajo **ha acabado** su **procesamiento en** las **máquinas anteriores**.

$$C_{k,i} \geq C_{k,i-1} + \underbrace{\sum_{j=1}^{n} X_{j,k} P_{j,i}} \qquad \forall k, i$$

El tiempo de completación de la tarea (k) en la máquina i

Debe ser superior o igual a...

El tiempo de completación acumulado en la ruta

Más

El tiempo de procesamiento de (k) en i

Segunda Desigualdad

El procesado en la máquina i del trabajo que ocupa la posición k **sólo puede comenzar cuando la máquina** que lo va a procesar **está libre**.

$$C_{k,i} \geq C_{l,i} + \sum_{j=1}^{n} X_{j,k} P_{j,i} - M(1 - Y_{k,f}) - M(1 - Y_{l,f}) \qquad \forall k, l, i, f \ : \ k > 1, \ l < k$$

Analicémosla por casos

- **Si en cierta factoría f**, **se procesan** sendos **trabajos que ocupan**, respectivamente, **las posicione**s en la secuencia **l** y **k** $(l < k)$, entonces

  $$Y_{k,f} = 1, \quad Y_{l,f} = 1$$

  Por tanto, la desigualdad se traduce en que

$$C_{k,i} \geq C_{l,i} + \underbrace{\sum_{j=1}^{n} X_{j,k} P_{j,i}}$$

El tiempo de completación de la tarea (k) en la máquina i

Debe ser superior o igual a...

El tiempo de completación en la máquina i del trabajo previo, (l)

Más

El tiempo de procesamiento de (k) en i

- **Si en cierta factoría f**, **se procesa** cierto trabajo (**k**) **pero** el trabajo (**l**) **no** se realiza allí ($l < k$), entonces

  $$Y_{k,f} = 1, \quad Y_{l,f} = 0$$

  Y la restricción propuesta equivale a

  $$C_{k,i} \geq C_{l,i} + \sum_{j=1}^{n} X_{j,k} P_{j,i} - M$$

  Como, en este caso, la restricción no debe modelizar ninguna condición lógica, es necesario anularla. Tomando M suficientemente grande, se establece que $C_{k,i} \geq 0$.

- **Si en cierta factoría f**, **se procesa** cierto trabajo (**l**) **pero** el trabajo (**k**) **no** se realiza allí ($l < k$), entonces

  $$Y_{k,f} = 0, \quad Y_{l,f} = 1$$

  Y la restricción propuesta equivale a

  $$C_{k,i} \geq C_{l,i} + \sum_{j=1}^{n} X_{j,k} P_{j,i} - M$$

  Como, en este caso, la restricción no debe modelizar ninguna condición lógica, es necesario anularla. Tomando M suficientemente grande, se establece que $C_{k,i} \geq 0$.

- **Si en cierta factoría f**, **no se procesan** sendos trabajos, (**k**) y (**l**) ($l < k$), entonces

  $$Y_{k,f} = 0, \quad Y_{l,f} = 0$$

  Y la restricción propuesta equivale a

  $$C_{k,i} \geq C_{l,i} + \sum_{j=1}^{n} X_{j,k} P_{j,i} - 2M$$

  Como, en este caso, la restricción no debe modelizar ninguna condición lógica, es necesario anularla. Tomando M suficientemente grande, se establece que $C_{k,i} \geq 0$.

Volviendo de nuevo a analizar el impacto a nivel computacional de este modelo, el **número de restricciones requeridas** para modelizar la planificación de n trabajos, en F fábricas con m máquinas cada una, es

$$\mathbf{4n + nm + \frac{1}{2}n(n-1)mF}$$

Este número se obtiene a partir de la siguiente tabla, en la que se calcula, para cada expresión, la cantidad de restricciones que genera.

Restricción	**Cantidad**	**Restricción**	**Cantidad**
a)	0	e)	n
b)	n	f) - 1)	nm
c)	n	f) - 2)	$n(n-1)mF/2$
d)	n		

2.3.5. Modelo 5: Basado en la secuenciación de tareas a partir de la presencia de *dummy jobs*, que actuarán como separadores entre fábricas en la secuenciación [34]

El modelo que se propone a continuación vuelve a explorar, como se hizo en 2.3.1, la idea de **organizar la producción mediante la secuenciación de las tareas**. Por ello, y al igual que en 2.3.1, la modelización hará uso de trabajos ficticios,***dummy jobs***.

En concreto, el modelo construirá una **única secuencia global** para la producción

- en la que se incluirá la **asignación y secuenciación** de las **tareas en** todas las **fábricas** implicadas, y
- que **se generará en bloques de tareas** usando ***dummy jobs* como separadores**[XVI].

Cada bloque de la secuencia global estará conformado por la secuencia de producción de una única fábrica[XVII]. Es decir, en cada bloque aparecerá el orden de ejecución de las trabajos en la fábrica correspondiente.

Con respecto a las **variables de decisión** empleadas, éstas serán:

- $\mathbf{X_{k,j}} = \begin{cases} 1 & \text{si el trabajo } j \text{ se procesa inmediatamente después del trabajo } k \\ 0 & \text{otro caso} \end{cases}$

  donde $X_{k,j} \in \{0,1\}$ y aparece definida para los subíndices $k, j \in \mathcal{J} \cup \{0\} : j \neq k$.

- $\mathbf{C_{j,i}} \in \mathbb{R}^+$, que es el tiempo de completación del trabajo $j \in \mathcal{J}$ en la máquina $i \in \mathcal{M}$. Es decir, el tiempo que ha estado procesándose el trabajo j en la máquina i, junto con el tiempo previo acumulado por el trabajo j en otras máquinas o en esperas.

- $\mathbf{C_{máx}} \in \mathbb{R}^+$, que es el tiempo de completación de todas las tareas, entre todas las fábricas.

Y, por tanto, para planificar la producción de n trabajos en F fábricas con m máquinas cada una, este modelo requerirá

- $\mathbf{nm + 1}$ variables reales (*continuous variable*, **CVs**)
- $\mathbf{(n + 1)n}$ variables binarias (*binary variables*, **BVs**)

Por razones de simplicidad, se entenderá que $k, j \in \mathcal{J} \cup \{0\}$. Añadir al dominio de los subíndices el cero resulta necesario para albergar al trabajo ficticio, que precede al primer trabajo real procesado en cada fábrica, y poder emplearlo como separador en la secuencia de producción final.

[XVI] Así, como se emplearán F trabajos ficticios como separadores, la secuencia tendrá $n + F$ posiciones.

[XVII] Realmente, no importará que fábrica tenga asociado cada bloque porque las fábricas son idénticas entre sí. Por simplicidad, se asociará a cada bloque una fábrica por orden cardinal.

Las **restricciones**, limitaciones y condiciones lógicas que se deben verificar en el **problema** de producción son:

a) El **dominio de las variables** involucradas.

$$\begin{cases} C_{j,i} \geq 0 & \forall j, i : j \neq 0 \\ X_{k,j} \in \{0,1\} & \forall k, j \ : \ j \neq k \\ C_{\text{máx}} \geq 0 & \end{cases}$$

b) **Cada trabajo debe aparecer en una única posición de la secuencia de producción**. Por tanto,

$$\sum_{\substack{k=0 \\ j \neq k}}^{n} X_{k,j} = 1 \qquad \forall j : j \neq 0$$

c) **Definición de la variable $\mathbf{X}_{\cdot,\cdot}$** con respecto a la **relación de precedencia inmediata**.

$$\sum_{\substack{j=0 \\ j \neq k}}^{n} X_{k,j} \leq 1 \qquad \forall k : k \neq 0$$

d) Como hay F fábricas, **se emplean exactamente F separadores** en la secuencia global, **que suceden a F − 1 trabajos** de esta secuencia.

$$\sum_{j=1}^{n} X_{0,j} = F \qquad\qquad \sum_{k=1}^{n} X_{k,0} = F - 1$$

e) **Dados dos trabajos** (reales) a procesar

- o aparecen juntos en la secuencia de producción (uno precede al otro)
- o están separados (y no hay relación de precedencia entre ellos en la secuencia de producción).

$$X_{k,j} + X_{j,k} \leq 1 \qquad \forall k \in \{1, \ldots, n-1\}, \ \forall j > k$$

f) **Definición del tiempo máximo de completación** de todas las tareas entre todas las fábricas.

El tiempo máximo de completación del problema, que denotamos por $C_{\text{máx}}$, es el máximo entre los tiempo de completación de los trabajos al acabar todo su procesado (es decir, tras pasar por la última máquina, m), $C_{\cdot,m}$.

Así, si se establece la siguiente desigualdad,

$$C_{\text{máx}} \geq C_{j,m} \qquad \forall j \ : \ j \neq 0,$$

como el problema de optimización tiene como objetivo minimizar el valor de $C_{\text{máx}}$, la anterior expresión se cumplirá sin holgura para algún trabajo. Y, por consiguiente, $C_{\text{máx}}$ será, efectivamente, el tiempo máximo de completación global.

g) **Definición de completación** para cada O_{ji}.

Se establecen dos desigualdades, prácticamente idénticas a las establecidas en 2.3.1, que, separadamente, acotan el tiempo de completación inferiormente. Pero combinadas y teniendo en cuenta que se está minimizando $C_{\text{máx}}$, definen correctamente el tiempo de completación para cada tarea y para máquina de la ruta.

Nótese, antes de comenzar, que se define $C_{j,0} := 0 := C_{0,l}\ \forall j, l$.

Primera Desigualdad

$$C_{j,i} \geq C_{j,i-1} + P_{j,i} \qquad \forall j, i : j \neq 0$$

- $C_{j,i}$: El tiempo de completación de la tarea j en la máquina i
- $\geq$: Debe ser superior o igual a...
- $C_{j,i-1}$: El tiempo de completación acumulado en la ruta
- $+$: Más
- $P_{j,i}$: El tiempo de procesamiento de j en i

Segunda Desigualdad

$$C_{j,i} \geq C_{k,i} + P_{j,i} + M\,(X_{k,j} - 1) \qquad \forall j, k, i \ :\ k \neq j,\ j \neq 0$$

Analicémosla por casos

- **Si el trabajo k precede al trabajo j**, entonces

  $$X_{k,j} = 1$$

  Y, por tanto, la desigualdad se traduce en que

  $$C_{j,i} \geq C_{k,i} + P_{j,i}$$

  - $C_{j,i}$: El tiempo de completación de la tarea j en la máquina i
  - $\geq$: Debe ser superior o igual a...
  - $C_{k,i}$: El tiempo de completación en la máquina i del trabajo previo a j, k
  - $+$: Más
  - $P_{j,i}$: El tiempo de procesamiento de j en i

- **Si en ninguna factoría el trabajo k precede al trabajo j**, la restricción propuesta equivale a

  $$C_{j,i} \geq C_{k,i} + P_{j,i} - M$$

  Como, en este caso, la restricción no debe modelizar ninguna condición lógica, es necesario anularla. Tomando M suficientemente grande, se establece que $C_{j,i} \geq 0$.

Volviendo de nuevo a analizar el impacto a nivel computacional de este modelo, el **número de restricciones requeridas** para modelizar la planificación de n trabajos, en F fábricas con m máquinas cada una, es

$$\mathbf{3n + 2 + n(n + 1)m + \frac{1}{2}n(n - 1)}$$

Este número se obtiene a partir de la siguiente tabla, en la que se calcula, para cada expresión, la cantidad de restricciones que genera.

Restricción	**Cantidad**	**Restricción**	**Cantidad**
a)	0	e)	$n(n-1)/2$
b)	n	f)	n
c)	n	g) - 1)	nm
d)	2	g) - 2)	$n(n-1)m + nm$

2.3.6. Modelo 6: Modificación del modelo de Manne para problemas JobShop, basado en la secuenciación de las tareas y adaptado a un entorno multifábrica de producción FlowShop con permutación [34]

El modelo para la resolución de problemas de tipo JobShop de *Alan S. Manne*, publicado al principio de la década de los sesenta en [31], ha resultado ser una considerable **fuente de innovación** en el paradigma de la formulación de modelos, que ha influido en más de 600 artículos y libros[XVIII].

Aunque el modelo en cuestión se basaba, de nuevo, en la **secuenciación de los trabajos**, éste permitía construir la secuencia de producción **sin necesidad de conocer que trabajo se procesaba inmediatamente después de que otro**. Le bastaba únicamente con conocer la precedencia entre trabajos, no necesariamente inmediata.

De este modo, esta forma de modelizar, además de producir una evidente **reducción de la cantidad de variables binarias usadas** (en comparativa con modelos más tradicionales), también hacía **innecesaria la inclusión de *dummy jobs***. Simplificaba, así, los modelos tradicionales y facilitaba la resolubilidad de los problemas de producción.

El **modelo** que se presenta a continuación es

- una **adaptación** de las ideas subyacentes en el modelo de *Manne*,
- y que está basado en la **secuenciación de las tareas** para en un entorno **multifábrica** con factorías de producción **FlowShop** con **permutación**.

[XVIII] Google Académico informa, a fecha de 5 de abril de 2018, que se ha citado 627 veces este artículo desde su publicación.

Con respecto a las **variables de decisión** empleadas, éstas serán:

- $\mathbf{X_{k,j}} = \begin{cases} 1 & \text{si el trabajo } j \text{ se procesa después del trabajo } k \\ 0 & \text{otro caso} \end{cases}$

  siempre que $k < j$ y $k \neq n$.

- $\mathbf{Y_{j,f}} = \begin{cases} 1 & \text{si el trabajo } j \text{ se procesa en la factoría } f \\ 0 & \text{otro caso} \end{cases}$

- $\mathbf{C_{j,i}} \in \mathbb{R}^+$, que es el tiempo de completación del trabajo j en la máquina i. Es decir, el tiempo que ha estado procesándose el trabajo j en la máquina i, junto con el tiempo previo acumulado por el trabajo j en otras máquinas o en esperas.

- $\mathbf{C}_{\text{máx}} \in \mathbb{R}^+$, que es el tiempo de completación de todas las tareas, entre todas las fábricas.

Y, por tanto, para planificar la producción de n trabajos en F fábricas con m máquinas cada una, este modelo requerirá

- $\mathbf{nm + 1}$ variables reales (*continuous variable*, **CVs**)
- $\mathbf{nF + n(n-1)/2}$ variables binarias (*binary variables*, **BVs**)

Las **restricciones**, limitaciones y condiciones lógicas que se deben verificar en el **problema** de producción son:

a) El **dominio de las variables** involucradas.

$$\begin{cases} C_{j,i} \geq 0 & \forall j, i \\ Y_{j,f} \in \{0,1\} & \forall j, f \\ X_{k,j} \in \{0,1\} & \forall k, j \ : \ j > k, k \neq n \\ C_{\text{máx}} \geq 0 \end{cases}$$

b) **Cada trabajo debe estar asignado exactamente a una fábrica**. Esto es,

$$\sum_{f=1}^{F} Y_{j,f} = 1 \qquad \forall j$$

c) **Definición del tiempo máximo de completación** de todas las tareas entre todas las fábricas.

El tiempo máximo de completación del problema, que denotamos por $C_{\text{máx}}$, es el máximo entre los tiempo de completación de los trabajos al acabar todo su procesado (es decir, tras pasar por la última máquina, m), $C_{\cdot,m}$.

Así, si se establece la siguiente desigualdad,

$$C_{\text{máx}} \geq C_{j,m} \qquad \forall j,$$

como el problema de optimización tiene como objetivo minimizar el valor de $C_{\text{máx}}$, la anterior expresión se cumplirá sin holgura para algún trabajo. Y, por consiguiente, $C_{\text{máx}}$ será, efectivamente, el tiempo máximo de completación global.

d) **Definición de completación** para cada O_{ji}.

Se establecen tres desigualdades, que, separadamente, acotan el tiempo de completación inferiormente. Pero combinadas y teniendo en cuenta que se está minimizando $C_{\text{máx}}$, definen correctamente el tiempo de completación para cada tarea y para máquina de la ruta.

Nótese, antes de comenzar, que se define $C_{j,0} := 0 \ \forall j$.

Primera Desigualdad

$$C_{j,i} \ \geq \ C_{j,i-1} \ + \ P_{j,i} \qquad \forall j, i$$

El tiempo de completación de la tarea j en la máquina i

Debe ser superior o igual a...

El tiempo de procesamiento de j en i

Más

El tiempo de completación acumulado en la ruta

Segunda Desigualdad

$$C_{k,i} \ \geq \ C_{j,i} + P_{k,i} - MX_{k,j} - M\left(1 - Y_{k,f}\right) - M\left(1 - Y_{j,f}\right) \qquad \forall i, k, j, f \ : \ k \neq n, j > k$$

Analicémosla por casos

A. **Si los trabajos k y j en la secuencia se ejecutan en la fábrica f**, entonces

$$Y_{k,f} = Y_{j,f} = 1.$$

Y, por tanto, la desigualdad se transforma en

$$C_{k,i} \ \geq \ C_{j,i} \ + \ P_{k,i} \ - \ MX_{k,j}.$$

Se presentan dos subcasos:

- **Si el trabajo j se realiza antes que el trabajo k** en la secuencia global de producción, entonces

$$X_{k,j} = 0.$$

Y, por tanto, la desigualdad toma la expresión

$$C_{k,i} \ \geq \ C_{j,i} \ + \ P_{k,i}.$$

El tiempo de completación de la tarea k en la máquina i de la fábrica f

Debe ser superior o igual a...

El tiempo de procesamiento de k en i

Más

El tiempo de completación en la máquina i de j, uno de los trabajos previos a k en la factoría f

- **En caso contrario**, la desigualdad equivale a

$$C_{k,i} \ \geq \ C_{j,i} \ + \ P_{k,i} \ - \ M$$

Como, en este caso, la restricción no debe modelizar ninguna condición lógica, es necesario anularla. Tomando M suficientemente grande, se establece que $C_{k,i} \geq 0$.

B. Si los trabajos k o j no se ejecutan en la fábrica f, entonces

$$Y_{k,f} = 1,\ Y_{j,f} = 0 \quad \text{ó} \quad Y_{k,f} = 0,\ Y_{j,f} = 1.$$

Y, por tanto, la desigualdad se transforma en

$$C_{k,i} \ \geq \ C_{j,i} \ + \ P_{k,i} \ - \ MX_{k,j} \ - \ M.$$

En este caso, la restricción no debe modelizar ninguna condición lógica y, por tanto, se debe anular. Analizando los dos subcasos que proporciona la variable $X_{k,j}$ de forma análoga a como se hizo en A., se llega a la conclusión que esto ocurre tomando M suficientemente grande.

C. Si los trabajos k y j no se ejecutan en la fábrica f, entonces

$$Y_{k,f} = Y_{j,f} = 0.$$

Y, por tanto, la desigualdad se transforma en

$$C_{k,i} \ \geq \ C_{j,i} \ + \ P_{k,i} \ - \ MX_{k,j} \ - \ 2M.$$

En este caso, de nuevo, la restricción no debe modelizar ninguna condición lógica y, por tanto, se debe anular. Analizando los dos subcasos que proporciona la variable $X_{k,j}$ de forma análoga a como se hizo en A., se llega a la conclusión que esto ocurre tomando M suficientemente grande.

Tercera Desigualdad

$$C_{j,i} \ \geq \ C_{k,i} + P_{j,i} - M\,(1 - X_{k,j}) - M\,(1 - Y_{k,f}) - M\,(1 - Y_{j,f}) \qquad \forall i,k,j,f \ : \ k \neq n, j > k$$

Su comportamiento se justifica, de forma análoga, a como se ha hecho con la segunda desigualdad.

Volviendo de nuevo a analizar el impacto a nivel computacional de este modelo, el **número de restricciones requeridas** para modelizar la planificación de n trabajos, en F fábricas con m máquinas cada una, es

$$\mathbf{2n + nm + n(n-1)mF}$$

Este número se obtiene a partir de la siguiente tabla, en la que se calcula, para cada expresión, la cantidad de restricciones que genera.

Restricción	**Cantidad**	**Restricción**	**Cantidad**
a)	0	d) - 1)	nm
b)	n	d) - 2)	$mFn(n-1)/2$
c)	n	d) - 3)	$mFn(n-1)/2$

2.3.7. Resumen de los principales costes computacionales de cada modelo

Se presenta, a modo de sumario, la siguiente tabla. Resume las principales características, a nivel computacional, de los seis modelos propuestos y analizados para resolver el problema DPFSP.

Modelo	**Número de BV**	**Número de CV**	**Número de Restricciones**	***Big-M***	***Dummy Jobs***
2.3.1. Modelo 1	n^2F	$nm+1$	$4n+F+nF+n^2m+{}^{n(n-1)}/_{2}$	Sí	Sí
2.3.2. Modelo 2	n^2F	$(1+2nm-n-m)F+1$	$2n+(n-1)(m-1)F+2F$	No	No
2.3.3. Modelo 3	n^2F	$nmF+1$	$2n+(2nm-m+n)F$	No	No
2.3.4. Modelo 4	n^2+nF	$nm+1$	$4n+nm+{}^{n(n-1)mF}/_{2}$	Sí	No
2.3.5. Modelo 5	$n(n+1)$	$nm+1$	$3n+2+nm(1+n)+{}^{n(n-1)}/_{2}$	Sí	Sí
2.3.6. Modelo 6	${}^{n(n+1)}/_{2}+nF$	$nm+1$	$2n+nm+n(n-1)mF$	Sí	No

2.4. Rendimiento de los modelos analizados

El objetivo que persigue en esta sección es **presentar una evaluación experimental de los modelos** analizados anteriormente que permita **mostrar evidencias sobre el rendimiento** de éstos.

La valoración tendrá lugar mediante la generación aleatoria de pequeñas instancias del problema de producción DPFSP. Cada una de ellas se tratará de resolver en el *solver* comercial CPLEX 12.7.1 haciendo uso de los diferentes modelos estudiados.

2.4.1. Configuración del equipo

La computación se ejecutará sobre la siguiente configuración:

CPU	Intel Core i5-750 @2.67GHz
Placa base	Gigabyte P55A-UD4
Memoria Ram	Kingston 8+4GiB @1333MHZ
Swap	20GiB sobre SSD SanDisk Ultra II en SATA3
Sistema Operativo	Ubuntu 16.04 + Kernel 4.13.0-38-generic
Notas	La computación tiene lugar en tty1 con las X detenidas (*service lightdm stop*). El swappiness del núcleo de Linux se fija en 10 (*sysctl -w vm.swappiness=10*).

2.4.2. Generación de las instancias e identificación de factores

Las **instancias** se generarán a partir de la **combinación de tres parámetros que definen el problema** DPFSP. A saber:

- El número de trabajos que se deben ejecutar, tomándose $\mathbf{n} \in \{\mathbf{4}, \mathbf{7}, \mathbf{10}, \mathbf{13}\}$.
- El número de fábricas disponibles para ejecutarlos, evaluándose $\mathbf{F} \in \{\mathbf{2}, \mathbf{3}, \mathbf{4}\}$.
- El número de máquinas (idéntico en todas las fábricas) que conforman el procesado de los trabajos, restringiéndose a $\mathbf{m} \in \{\mathbf{2}, \mathbf{4}, \mathbf{6}\}$.

La combinación de los anteriores valores generan **36 posibles problemas** diferentes.

Para cada uno de ellos se generarán **5 instancias diferentes** en las que el **tiempo de procesado** de cada trabajo en las diferentes máquinas será entero y se tomará **uniformemente sobre el rango** $(\mathbf{1}, \mathbf{99})$. Luego, se dispondrá, **en total**, de **180 instancias** sobre las que ejecutar los diferentes modelos propuestos.

Cada ejecución dispondrá de un tiempo máximo de ejecución **y un número máximo de núcleos** (*threads*) disponibles. En concreto, cada instancia será ejecutada 4 veces: una para cada posible configuración obtenida a través de la combinación de estos dos factores:

- El tiempo máximo disponible, tomándose en $\{300s, 1800s\}$
- El número máximo de núcleos, tomándose en $\{3 \text{ núcleos}, 4 \text{ núcleos}\}$

El factor tiempo limita la ejecución de los modelos a tiempos razonables de computación (5 y 30 minutos, respectivamente). Y es que, en problemas pequeño como los que se plantean en este estudio, **es impensable** tener que **esperar más tiempo para obtener una solución óptima**.

El valor 5 minutos se elige como el valor ideal máximo que un usuario debería estar esperando para obtener la solución óptima del problema. Pero se llega a permitir un máximo de 30 minutos para que el solver resuelva óptimamente el problema.

En lo que respecta al **factor limitante de núcleos** disponibles, éste responde a los **límites domésticos** que marca la **computación actual**: procesadores con cuatro núcleos.

Está claro que, a nivel empresarial, los servidores cuentan con muchos más núcleos de procesamiento o se dispone de una red distribuida de ordenadores para ejecutar tareas arduas de computación. Pero **destinar tanta potencia de cálculo a resolver óptimamente unas instancias pequeñas es absurdo**.

También es absurdo destinar muy pocos *threads* para resolver las instancias adecuadamente. Es un desperdicio computacional emular entornos mononúcleos porque, además de que hoy en día son entornos residuales[XIX], el solver comercial CPLEX está programado para aprovechar la computación paralela.

A raíz de esto, se estudia el efecto *número de threads* con los niveles 3 y 4 núcleos para computar las instancias.

[XIX] La única situación donde se da esta configuración, con un solo core para computar, es cuando, habitualmente, se ejecutan operaciones no paralelizables sobre entornos virtualizados. En estos entornos es posible hacer redistribuciones y crear más máquinas virtuales para aprovechar la potencia de computación no utilizada.

En resumidas cuentas, **la siguiente tabla muestra los diferentes factores contemplados para la computación y evaluación del rendimiento de los modelos analizados**:

Factor del problema	Símbolo	**Número de niveles**	**Valores**
Número de trabajos a programar	n	4	4, 7, 10, 13
Número de fábricas disponibles	F	3	2, 3, 4
Número de máquinas	m	3	2, 4, 6

Factor computacional	Símbolo	**Número de niveles**	**Valores**
Solver	*Solver*	1	CPLEX 12.7.1
Núcleos disponibles	*ThreadLimit*	2	3, 4
Tiempo disponible	*TimeLimit*	2	300s, 1800s

Conviene observar que el sistema operativo, el lanzador y CPLEX van a compartir núcleo cuando se permita, en la ejecución, el uso de 4 *threads*.. Y que este comportamiento **podría sesgar el estudio. Pero esto no ocurre** porque

- El **lanzador**, mientras está en espera, no consume tiempo de CPU. Y, en caso de necesidad, la memoria Ram que pueda estar usando se podría trasladar a swap.
- El consumo que hace el **sistema operativo en modo tty** es ridículo.

Y es que, en modo tty, sólo se está ejecutando el kernel de Linux y, residualmente, algunos servicios, como

- el del sonido (*alsa*),
- el de montaje de medios (*mount*), o
- los demonios de *systemd*.

Los posibles ciclos de computación que pueda llegar a emplear el núcleo de Linux resultan totalmente despreciables porque éste se encarga de:

- gestionar la memoria Ram
- administrar el tiempo de procesador, y
- controlar los buses y E/S a través del chipset norte y sur de la placa.

Tareas fundamentales y vitales, pero con un insignificante ciclo computacional y transparente de cara a la ejecución de otros procesos.

2.4.3. Programación de la ejecución

Como lanzador para organizar la ejecución de las diferentes instancias con sus características específicas y factores, se empleará un script en `R` (3.4.4, x86_64-pc-linux-gnu), que hará uso de la librería[xx] de tipo *wrapper*[xxi] de CPLEX para `C` a través del paquete `Rcplex` (1.3.3).

Los datos recogidos, para cada instancia y ejecución, son:

- El **tamaño del problema** (factores problema): n, m y F.
- Las **limitaciones computacionales** impuestas (factores computación): *ThreadLimit* y *TimeLimit*.
- El **tiempo de computación dedicado** por instancia.
- El estado del solver CPLEX al finalizar su ejecución (**si ha encontrado la solución óptima o no**).
- Y **el GAP alcanzado** por CPLEX en su ejecución.

2.4.4. Análisis descriptivo del rendimiento

La tabla de la **figura 2.8 resume**, **para cada modelo y factor computacional, las características estudiadas**. Las tablas de la **figura 2.9** hace lo propio para cada modelo al dividir los datos en función del **número de trabajos a programar en las instancias y del tiempo máximo permitido para resolver óptimamente**.

Las tres figuras muestran:

- La **tasa de optimalidad alcanzada**, con las instancias, para cada modelo
- El **GAP medio alcanzado**, con las instancias, para cada modelo, y
- El **tiempo empleado por CPLEX** en cada instancia, para cada modelo.

A estas tablas **se les ha aplicado un formato condicional para comparar los valores entre los modelos estudiados para resolver el problema** $DF|prmu|C_{\text{máx}}$. En concreto:

- Se muestran barras de datos para comparar (relativamente, entre el mejor de los datos de esa categoría), la tasa de optimalidad alcanzada en cada modelo (en positivo) y el GAP medio alcanzado entre modelos (en negativo).
- Y se escoge una representación icónica a través de la esfera de un reloj para comparar el tiempo empleado.

En ellas se puede observar que **el modelo 6 es el que mejor rendimiento ofrece** sobre las instancias estudiadas: mejor tiempo medio, mejor tasa de optimalidad y menor GAP medio.

[xx] Una librería está formada por un conjunto de implementaciones funcionales o subprogramas. Está diseñada para ser ejecutada por otros programas y ofrecer interacciones con, o implementaciones de, funciones o servicios.

[xxi] Las bibliotecas de tipo *wrapper* están formadas por código y funciones ligeras (*shim*), que traducen la interfaz existente de otra biblioteca, a una interfaz compatible con el invocador de esta biblioteca. En el caso de `R`, la librería `Rcplex` hace accesible la API en `C` de CPLEX para `R`.

Caracteristicas de la instancia	Límite tiempo	300s		1800s		Valores medios
	Núcleos	3	4	3	4	
Modelo 1	% opt	65,00	64,44	72,22	72,78	68,61
	% GAP	8,22	8,09	6,48	6,13	7,23
	Tiempo medio (s)	115,47	113,34	580,71	565,82	343,84
Modelo 2	% opt	63,33	64,44	68,89	70,56	66,81
	% GAP	4,47	4,30	3,06	2,92	3,68
	Tiempo medio (s)	121,62	116,70	625,87	601,39	366,39
Modelo 3	% opt	75,00	75,56	81,11	81,67	78,33
	% GAP	2,10	1,97	1,24	1,24	1,64
	Tiempo medio (s)	87,44	86,46	401,61	387,72	240,81
Modelo 4	% opt	67,78	68,33	74,44	75,56	71,53
	% GAP	4,82	4,56	3,11	2,87	3,84
	Tiempo medio (s)	111,73	108,99	526,90	514,16	315,45
Modelo 5	% opt	82,22	82,22	83,89	83,89	83,06
	% GAP	4,67	4,49	3,69	3,48	4,08
	Tiempo medio (s)	68,40	64,98	320,81	310,59	191,20
Modelo 6	% opt	94,44	93,89	97,22	97,78	95,83
	% GAP	0,77	0,77	0,29	0,18	0,50
	Tiempo medio (s)	26,16	27,26	80,81	80,35	53,64
Valores medios	% opt	*74,63*	*74,81*	*79,63*	*80,37*	*77,36*
	% GAP	*4,17*	*4,03*	*2,98*	*2,80*	*3,50*
	Tiempo medio (s)	*88,47*	*86,29*	*422,78*	*410,00*	*251,89*

Figura 2.8: **Tabla resumen y comparativa del rendimiento mostrado por los modelos estudiados**. Por ejemplo, el modelo 6, cuando se le permite el uso de 3 núcleos y de 300 segundos para la computación, encuentra una solución óptima, para las instancias estudiadas, en el 94.44 % de los casos. El GAP medio alcanzado, en este escenario, es del 0.77 %. Emplea, de media, 26.16 segundos de computación para procesar cada una de las 180 instancias propuestas.

Caracteristicas de la instancia	Límite tiempo	300s				Valores medios
	Núm. Trabajos	4	7	10	13	
Modelo 1	% opt	100,00	100,00	56,67	2,22	64,72
	% GAP	0,00	0,00	7,15	25,46	8,15
	Tiempo medio (s)	0,02	0,26	163,87	293,48	114,41
Modelo 2	% opt	100,00	100,00	44,44	11,11	63,89
	% GAP	0,00	0,00	4,49	13,04	4,38
	Tiempo medio (s)	0,05	6,77	198,78	271,03	119,16
Modelo 3	% opt	100,00	100,00	86,67	14,44	75,28
	% GAP	0,00	0,00	0,82	7,33	2,04
	Tiempo medio (s)	0,03	0,57	80,37	266,83	86,95
Modelo 4	% opt	100,00	100,00	61,11	11,11	68,06
	% GAP	0,00	0,00	3,02	15,72	4,69
	Tiempo medio (s)	0,03	0,99	172,54	267,88	110,36
Modelo 5	% opt	100,00	100,00	100,00	28,89	82,22
	% GAP	0,00	0,32	0,00	18,00	4,58
	Tiempo medio (s)	0,02	0,07	25,65	241,04	66,69
Modelo 6	% opt	100,00	100,00	100,00	76,67	94,17
	% GAP	0,00	0,00	0,00	3,08	0,77
	Tiempo medio (s)	0,03	0,12	3,62	103,07	26,71
Valores medios	% opt	*100,00*	*100,00*	*74,81*	*24,07*	*74,72*
	% GAP	*0,00*	*0,05*	*2,58*	*13,77*	*4,10*
	Tiempo medio (s)	*0,03*	*1,46*	*107,47*	*240,56*	*87,38*

Caracteristicas de la instancia	Límite tiempo	1800s				Valores medios
	Núm. Trabajos	4	7	10	13	
Modelo 1	% opt	100,00	100,00	87,78	2,22	72,50
	% GAP	0,00	0,00	1,54	23,68	6,30
	Tiempo medio (s)	0,02	0,26	529,77	1763,00	573,26
Modelo 2	% opt	100,00	100,00	62,22	16,67	69,72
	% GAP	0,00	0,00	2,38	9,56	2,99
	Tiempo medio (s)	0,05	6,78	882,95	1564,73	613,63
Modelo 3	% opt	100,00	100,00	97,78	27,78	81,39
	% GAP	0,00	0,00	0,13	4,83	1,24
	Tiempo medio (s)	0,03	0,57	145,20	1432,87	394,67
Modelo 4	% opt	100,00	100,00	88,89	11,11	75,00
	% GAP	0,00	0,00	0,35	11,60	2,99
	Tiempo medio (s)	0,03	0,99	477,05	1604,04	520,53
Modelo 5	% opt	100,00	100,00	100,00	35,56	83,89
	% GAP	0,00	0,32	0,00	14,02	3,58
	Tiempo medio (s)	0,02	0,07	25,66	1237,06	315,70
Modelo 6	% opt	100,00	100,00	100,00	90,00	97,50
	% GAP	0,00	0,00	0,00	0,94	0,24
	Tiempo medio (s)	0,03	0,12	3,62	318,54	80,58
Valores medios	% opt	*100,00*	*100,00*	*89,44*	*30,56*	*80,00*
	% GAP	*0,00*	*0,05*	*0,73*	*10,77*	*2,89*
	Tiempo medio (s)	*0,03*	*1,47*	*344,04*	*1320,04*	*416,39*

Figura 2.9: **Tablas resumen y comparativas del rendimiento mostrado por los modelos estudiados** al considerarse como factor discriminante el ***número de trabajos a ejecutar***. La primera tabla muestra los valores de los estadísticos de interés estudiados cuando se limita la ejecución del *solver* a 300 segundos. Y la segunda, cuando es de 1800 segundos.

Por ejemplo, el modelo 6, cuando debe programar la ejecución de 13 trabajos y se le permiten 300 segundos para la computación, encuentra una solución óptima, para las instancias estudiadas, en el 76.67 % de los casos. El GAP medio alcanzado, en este escenario, es del 3.08 %. Empleará, de media, 103.07 segundos de computación para resolver óptimamente.

Un análisis detallado de los datos ofrecidos en 2.8 permite observar que, en algunas condiciones, por ejemplo, **la tasa de optimalidad se reduce al permitir más núcleos en la ejecución** (manteniendo inalterados el resto de factores que podrían intervenir). Un efecto curioso y relevante para valorar el estado de los datos recogidos.

Puesto que la CPU usada dispone, físicamente, de los cuatro *cores* empleados y no emula *threads*, sólo hay una posible justificación para este comportamiento. **Se trata de la gestión que hace CPLEX de la computación paralela**.

CPLEX debe organizar el estudio del árbol de posibilidades (generado por un algoritmo propietario y no público de Branch&Cut). Como resulta lógico, **primero** recorre y **estudia aquellas** posibilidades que **predice como las más rápidas de evaluar** para, así, disponer de cotas con las que realizar cortes (y dejar sin estudiar ramas que van a proporcionar peores soluciones de las que ya se disponen).

Y es aquí donde se causa este curioso efecto. **Al variar el número de núcleos disponibles, las ramas del árbol se recorren de forma diferente**. Y, por tanto, las cotas que se obtienen deberían ser diferentes. Se debe suponer que IBM ha implementado su algoritmo para maximizar el beneficio obtenido (obtener muy pocas veces peores cotas al aumentar la potencia computacional).

Pero claro, es imposible que estos algoritmos ofrezcan siempre el comportamiento deseado. Y, a veces, ocurre lo que muestran los datos de la tabla 2.8, que la forma de recorrer el árbol empeora la tasa de optimalidad.

Justificación del análisis descriptivo.

El hecho de incluir estas tablas, y no otras, en este documento está razonado:

- Incluir la **figura 2.8** permite mostrar, descriptivamente, el efecto de los **factores computacionales** considerados en los modelos estudiados y **valorar** cualitativamente el **impacto** que tienen en la tasa de optimalidad, en el GAP y en el tiempo medio. La figura 2.12 mostrará que los factores computacionales no son especialmente importantes.

- Las tablas de la **figura 2.9** permiten **valorar** el **efecto** de la variable n, **número de trabajos a programar, sobre la resolubilidad de las instancias en el modelo correspondiente**. Y es que, como mostrará la figura 2.12 será el factor más relevante para explicar el porcentaje de optimalidad obtenido en cada modelo.

  Se debe comparar a través del factor *tiempo disponible para resolver óptimamente* porque, entre otras variables, se mide el tiempo medio de computación en cada instancia. Y comparar instancias en las que se les ha impuesto dos máximos distintos va en contra de la independencia y aleatorización del estudio.

- **No** se ha incluido el respectivo **análisis descriptivo usando los factores *número de fábricas* y *número de máquinas* por no considerar suficientemente relevante la información que aportan**. Conviene notar que la figura 2.12 ya los valorará estadísticamente.

2.4.5. Análisis formal de los datos

Aunque la literatura científica normalmente realiza un análisis estadístico empleando técnicas CHAID (*Chi-Squared Automatic Iteration Detection*), el análisis que se llevará a cabo usará **técnicas del aprendizaje-máquina** (*machine learning, ensemble learning*) para clasificar los resultados obtenidos.

Resulta inviable realizar una detallada disquisición acerca del *machine learning* y las diferentes técnicas que se han empleado para analizar los datos. Simplemente, se comentará el modelo que ofrece mejores resultados. Y las conclusiones que se pueden obtener a partir de él.

Tras haber probado una batería de modelos y técnicas diferentes con los que tratar los datos (como GLM, Boosted LM, Boosted Tree o Logistic Model Trees), la que proporcionó **mejores resultados** fue ***Random Forest***[XXII], con la implementación del paquete `randomForest` en `R`.

Random Forest.

La **variable respuesta** del modelo será categórica: el **tipo de solución obtenida en CPLEX** (S si la solución es óptima, N si no lo es). Y la **variabilidad** observada se tratará de **explicar mediante** las siguientes variables:

- El **número de trabajos** contemplados, `n`
- El **número de fábricas** disponibles, `f`
- El **número de máquinas** estudiadas, `m`
- El **tiempo** de computación **disponible**, `t_disponible`
- El número de **núcleos disponibles**, `cpu_disponible`
- El **modelo empleado** en la computación, `modelo`.

Las **instancias** serán particionadas en **dos subconjuntos**: de entrenamiento y de validación. La partición se hace **de forma equilibrada** entre las diferentes clases y factores: La primera instancia de cada configuración se toma para la validación, y las cuatro restantes para el entrenamiento.

[XXII] *Random Forest* (o bosques aleatorios en castellano) es una técnica de aprendizaje profundo que nace con el objetivo de mejorar los árboles de decisión. Un árbol de decisión es un mapa, un esquema, construido para representar, visualmente y de forma explícita, los resultados emanados de las diferentes decisiones posibles que se pueden tomar.

Random Forest combina una cantidad grande de árboles de decisión independientes y probados sobre diferentes conjuntos derivados de los datos originales. Estos conjuntos son selecciones aleatorias (con o sin reemplazamiento) de los datos observados, que se usan para incluir la aleatoriedad en la generación de los diferentes árboles. Incluso se puede llegar a elegir variables al azar en cada nodo de los árboles generados, dejándolas crecer en profundidad (sin podar).

Al final, las predicciones de este modelo para nuevos datos se obtienen usando el voto mayoritario de los árboles (positivo si la mayoría de los árboles lo predicen como positivo).

Árboles generados.

RandomForest combina una gran cantidad de árboles de decisión independientes para tratar de modelizar, a través de ellos, los datos presentados. Y es que **el objetivo de un árbol de decisión es modelizar una serie de datos para poder predecir el valor de una variable (respuesta) en función del valor que toman diversas variables de entradas**.

En estas representaciones, las hojas representan etiquetas de clase y las ramas representan el conjunto de características (valores de las variables de entrada) que conducen a esas etiquetas de clase. Por ello, en este estudio, las dos posibles etiquetas de clase son S y N, marcadas por el tipo de solución obtenida en CPLEX. Y el recorrido de las ramas se marca mediante Y o N, según si se cumple la determinada característica o no.

La primera decisión que se debe tomar para construir *RandomForest* es **determinar el número de árboles a usar** en su generación. Por ello se decide hacer uso de una **validación** para determinar este parámetro. Entre un conjunto de posibles valores prefijados y arbitrarios para configurar el número de árboles a usar en el modelo, se elegirá finalmente aquel que le proporcione a *RandomForest* mayor tasa de acierto (precisión) al ejecutarse sobre el conjunto de validación.

El resultado de esta validación se observa en la figura 2.10, donde se comprueba que **usando 3000 árboles se obtiene la mayor precisión**. El resto de posibles parámetros probados no andan muy lejos de la precisión máxima observada y podrían usarse también como parámetros (a costa una pérdida insignificante de precisión).

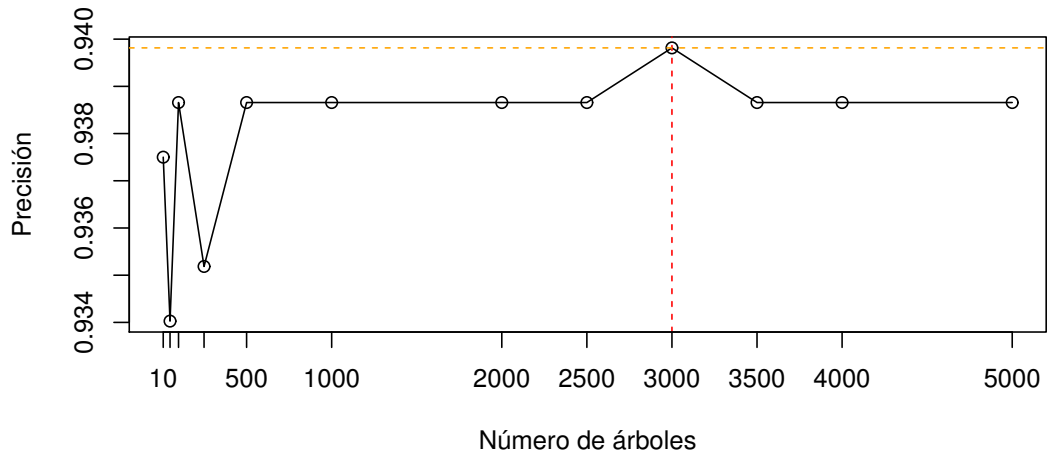


Figura 2.10: **Análisis del parámetro *número de árboles* en la generación de *RandomForest***. Se estudia la influencia que tiene el número de árboles empleados para generar el modelo sobre la precisión final de éste. La precisión se calcula al ejecutar el modelo obtenido sobre el conjunto de validación. **La gráfica muestra que, cuando se usan 3000 árboles, la precisión obtenida es máxima**.

Exploración de un árbol generado.

De los 3000 árboles construidos en este estudio para explicar la resolubilidad óptima de instancias a través de CPLEX, en la **figura 2.11 se muestran los primeros 5 niveles, de los 10 que tiene el primer árbol generado por random forest**. Y es que resulta de vital importancia explorar alguno de estos árboles para ganar intuición sobre las las relaciones que se han formado entre las variables.

Una rápida observación a la estructura de este árbol señala que la **primera división** que se produce en éste es el **modelo analizado**. Separa el modelo sexto (el que presentaba mayor rendimiento) del resto de modelos.

Así, por ejemplo, para este modelo sexto (N), el árbol analiza, a continuación, el número de fábricas disponibles. En caso de contar con dos fábricas (Y), analiza, posteriormente, el número de máquinas disponibles. Si se disponen de más de dos fábricas (N), se pregunta sobre el número de trabajos a programar.

Que este árbol muestre esta estructura no significa que el resto de árboles generados también establezcan los mismos nodos y hojas[XXIII]. Cada árbol se genera a través de conjuntos de datos parcialmente diferente, derivado de los datos originales y conformado seleccionando un subconjunto del total de variables y registros disponibles.

Y, por ello, las predicciones que se extraigan finalmente de este modelo *RandomForest* o la relevancia de cada una de las variables para explicar la variabilidad observada puede diferir de lo que se muestra en la figura 2.11.

[XXIII] Por su naturaleza, los árboles de decisión que se generan a través de *RandomForest* no tienen una estructura predeterminada. Son modelos no paramétricos, generados a través de los propios datos usados como entrada.

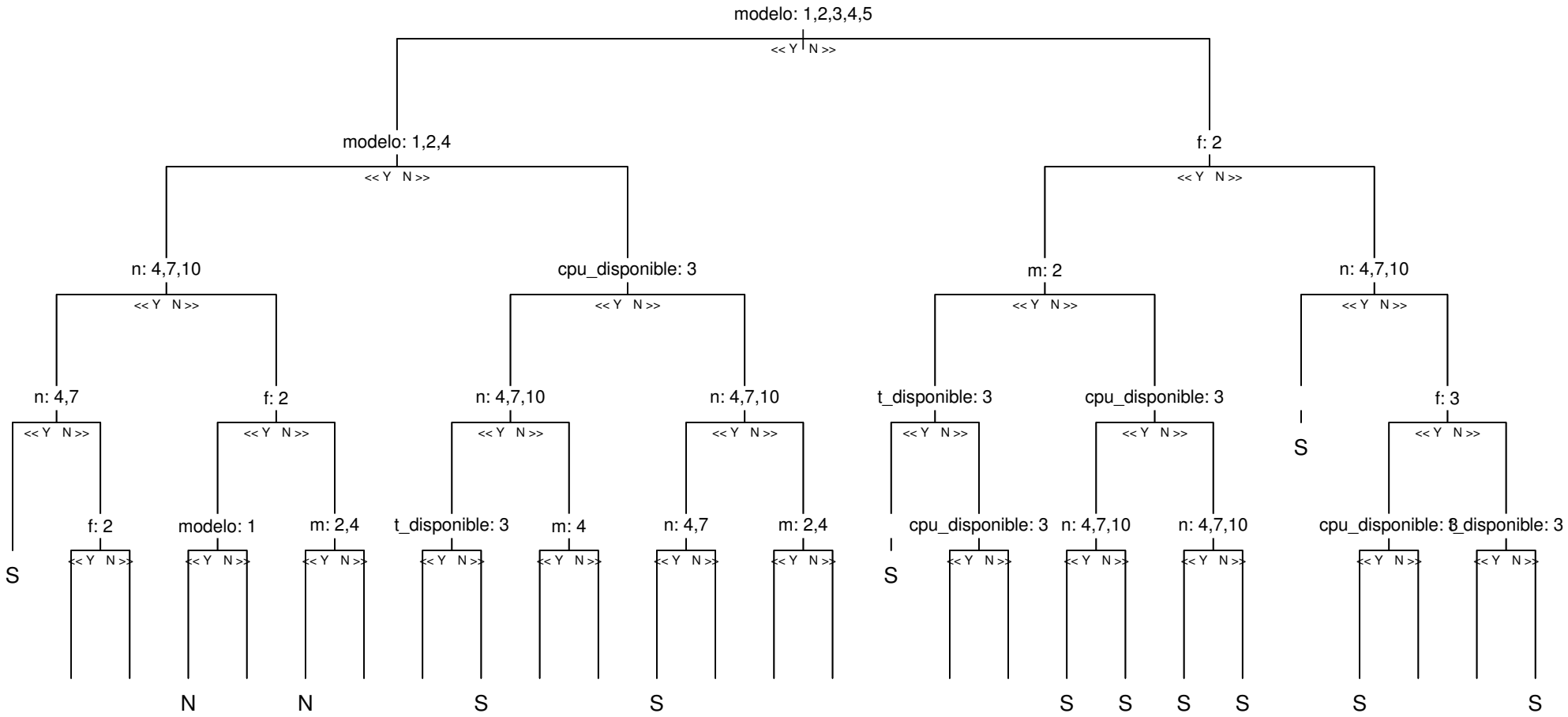


Figura 2.11: **Árbol de decisión generado en la primera iteración de Random Forest**. Se muestran únicamente **los cinco primeros niveles**. El árbol ofrece hasta 10 niveles y, por eso, algunos niveles en esta representación no acaban clasificando los datos observados (S ó N).

Importancia de las variables estudiadas.

La **figura 2.12** recoge dos gráficas que **valoran, estadísticamente,** la **importancia de los factores considerados** en este estudio para explicar la variabilidad que presenta la variable respuesta sobre los datos experimentales recogidos.

Y es que la medida ***Mean Decrease Accuracy*** [30] permite **conocer el impacto relativo que tendría sobre el modelo clasificador elaborado** con *RandomForest* **al eliminarle una determinada variable de entrada**. Y, con ello, conocer que variables son las que ejercen un mayor poder predictivo.

Con respecto a la métrica *Mean Decrease Gini*, se calcula a partir de la llamada *función de impurezas* o *índice de Gini*, muy relacionado con la construcción de cada árbol de decisión. Y es que este índice interviene directamente en la forma de seleccionar que variables y valores conforman los nodos en cada ramificación de cada uno de los árboles construidos.

Para cada árbol, el algoritmo busca siempre minimizar el valor del índice en cada nodo de éste porque el valor del índice representa la frecuencia con la que una observación, elegida aleatoriamente, se etiquetaría erróneamente cuando se siguiese la estructura ya creada por el árbol y se añadiese este nuevo nodo.

Dicho de otra forma, valora el grado de *pureza* que se presentaría en cada partición de datos creada por este hipotético nodo en ese árbol. La *pureza* de una partición se refiere al número de datos, en relativo, cuya variable respuesta observacional no se corresponde con la de la mayoría de datos que conforman esa partición.

Por ello, al ir añadiendo más nodos al árbol es evidente que el índice de Gini va disminuyendo. Y en eso se basa **la métrica *Mean Decrease Gini*, que se mide, para cada variable, como la media, sobre el total de árboles, de la suma de los decrementos atribuidos a esta variable** [30].

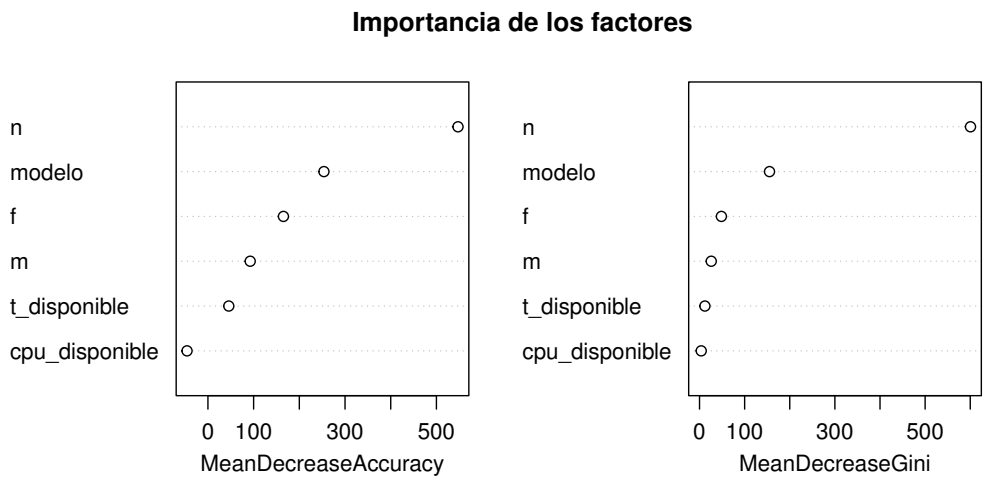


Figura 2.12: **Valoración de la importancia de los diferentes factores contemplados en el análisis con Random Forest**. Las dos medidas consideradas (la precisión y la de Gini) indican que el número de trabajos a procesar y el modelo elegido para computar son los dos factores más relevantes. En cambio, los factores número de fábricas, número de máquinas y tiempo disponible para resolver son los menos importantes. El factor número de núcleos disponibles empeora los resultados obtenidos y conviene eliminarlo.

Volviendo a la **figura 2.12**, ambas métricas coinciden al ordenar a las variables según su importancia (*en presencia de las restantes*[XXIV]). La **variable más importante**, con diferencia, **para explicar la variabilidad observada** sobre los datos, según RandomForest, **es el número de trabajos a programar**. Y, **en segunda posición, el modelo empleado para resolver la instancia**.

El **número de fábricas** de las que se dispone y el **número de máquinas** quedan **relegadas** a un tercer y cuarto puesto, mientras que el **tiempo disponible** para resolver una instancia aparece en el **penúltimo lugar**.

El **factor número de núcleos disponibles** para computar con los dos niveles estudiados resulta **totalmente despreciable para explicar la resolubilidad de las instancia**s. Y, según se observa la figura 2.12, eliminarlo aumentaría la precisión del modelo. Esto justifica la necesidad de elaborar un nuevo modelo que prescinda de esta variable.

Conviene hacer notar que, aunque el árbol presentado en la figura 2.11 considere que el modelo empleado para resolver una determinada instancia es el factor que más explica la variabilidad presentada en los resultados obtenidos (Gini), según se observa en 2.12 este factor aparece relegado a una segunda posición. Esto no contradice los resultados presentados.

Calidad del modelo.

Con respecto a la **calidad predictora del model**o, señalar que

- El valor estimado del error OOB (*Out Of Bag Error*) es del 4.43 %.
- El error en la clasificación de los valores observados en el conjunto de validación es del 6.02 %.

La matriz de confusión OOB presentada por el modelo es:

	Predicción		
Observ.	S	N	Error clasif.
S	2630	49	0.01829041
N	104	673	0.13384813

Y la obtenida al ejecutar el modelo sobre el conjunto de validación es:

	Predicción		
Observ.	S	N	Error clasif.
S	648	15	0.02262443
N	37	164	0.18407960

[XXIV] Aunque no se haga explícito en cada oración, la importancia está condicionada a la presencia de las demás variables que conforman el estudio. Una variable es o no es importante, en presencia de las restantes.

Un nuevo RandomForest.

Se procede a eliminar el factor que recoge el número de núcleos dedicados a la computación como sugería la figura 2.12 y a elaborar un nuevo modelo *RandomForest*.

La **figura 2.13** justifica el hecho de **emplear 150 árboles** para generar el modelo. Muchísimos menos que los 3000 utilizados anteriomente.

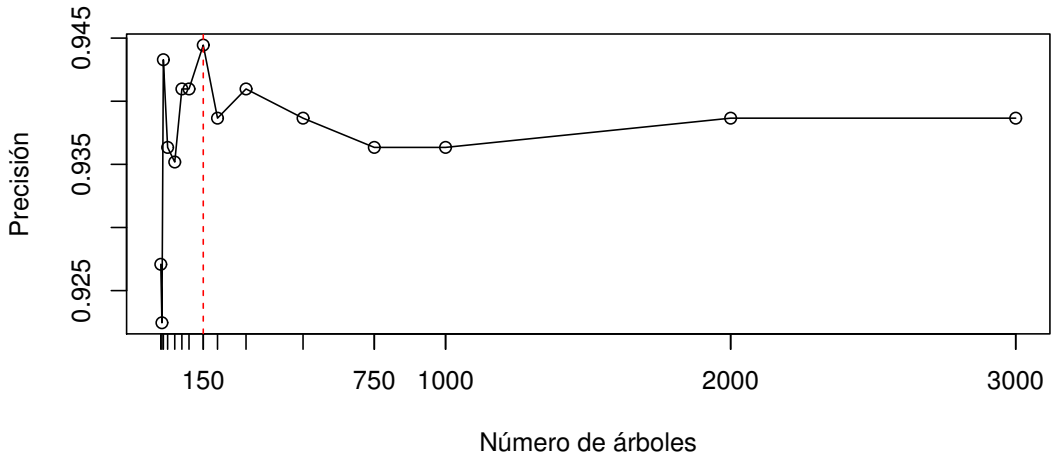


Figura 2.13: **Análisis del parámetro *número de árboles* en la generación del nuevo modelo de *RandomForest***. Se estudia la influencia que tiene el número de árboles empleados para generar el modelo sobre la precisión final de éste. La precisión se calcula al ejecutar el modelo obtenido sobre el conjunto de validación. **La gráfica muestra que, cuando se usan 150 árboles, la precisión obtenida es máxima**.

Por su lado, la figura 2.14 muestra los primeros cinco niveles del primer árbol de decisión generado por este nuevo modelo con 150 árboles de decisión. Se recomienda al lector observarlo para ganar intuición.

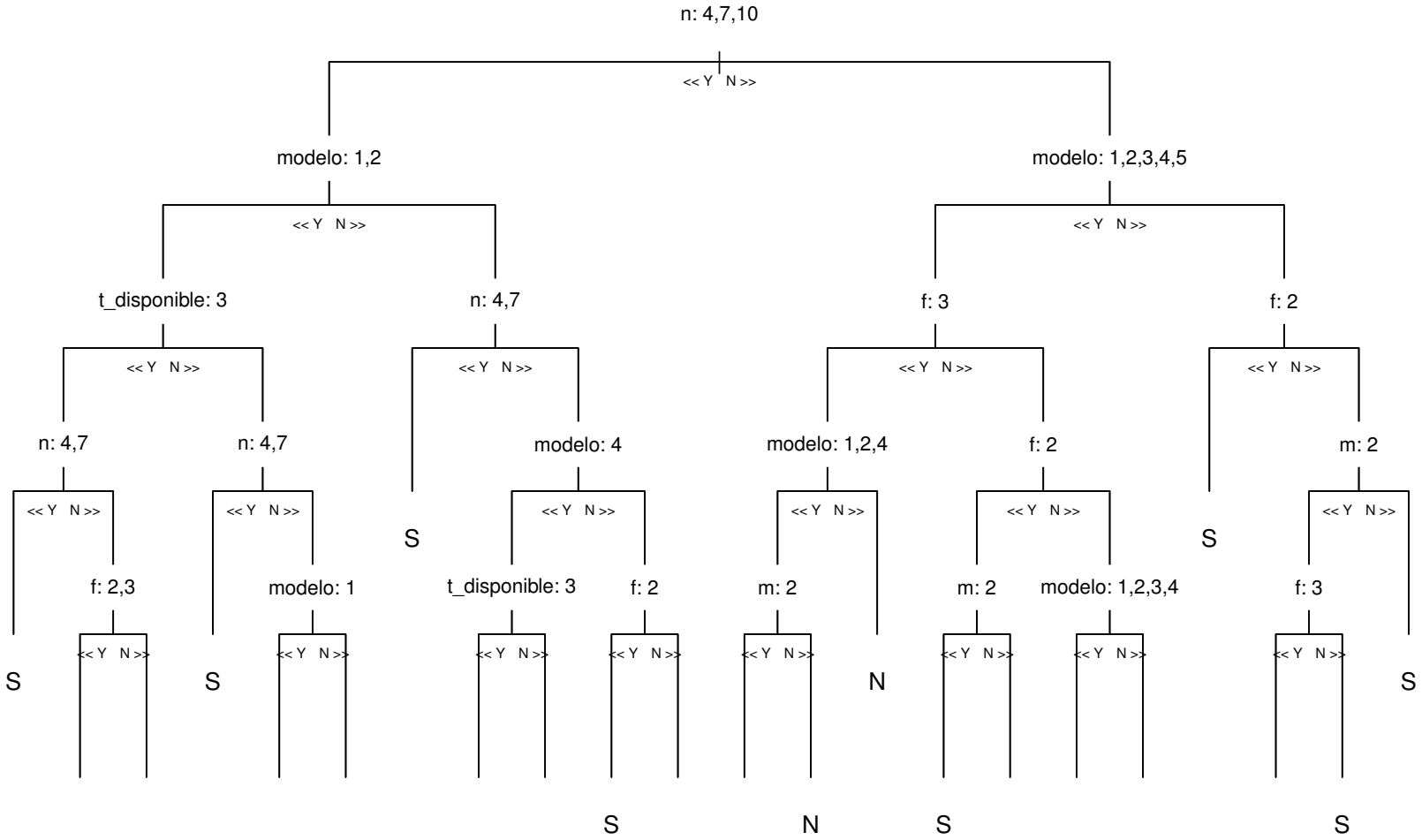


Figura 2.14: **Árbol de decisión generado en la primera iteración del nuevo modelo de Random Forest**. Se muestran únicamente **los cinco primeros niveles**. El árbol ofrece hasta 9 niveles y, por eso, algunos niveles en esta representación no acaban clasificando los datos observados (S ó N).

La **figura 2.15** analiza la **importancia de los factores considerados**. En ella **se observa la misma tendencia** que ofrecía la figura 2.12 con respecto a la importancia de los factores. Y se señala que eliminar algún otro factor reduciría la variabilidad explicada por el nuevo modelo.

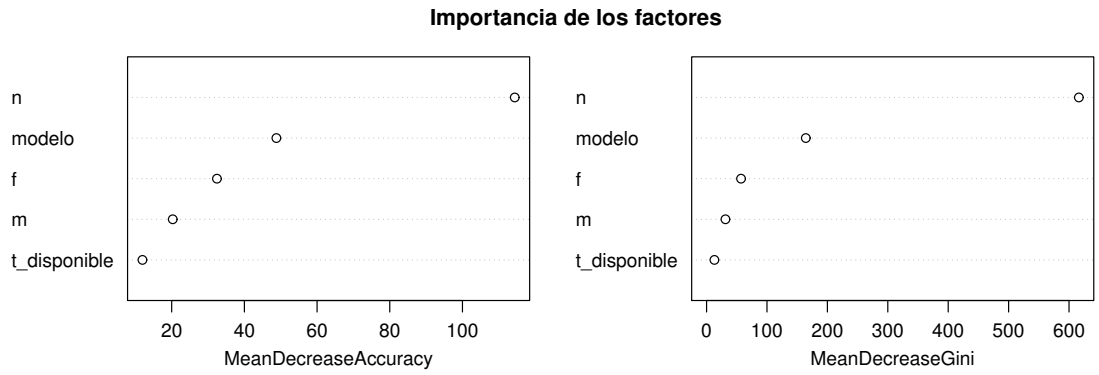


Figura 2.15: **Valoración de la importancia de los diferentes factores contemplados en el análisis con Random Forest**. Las dos medidas consideradas (la precisión y la de Gini) indican que el número de trabajos a procesar y el modelo elegido para computar son los dos factores más relevantes. En cambio, los factores número de fábricas y número de máquinas son los menos importantes. El factor tiempo disponible para resolver es casi irrelevante, aunque prescindir de él empeoraría los resultados obtenidos.

Por último, con respecto a la **calidad predictora del modelo**, señalar que

- El valor estimado del error OOB (*Out Of Bag Error*) es del 4.05 %.
- El error en la clasificación de los valores observados en el conjunto de validación es del 5.44 %.

La matriz de confusión OOB presentada por el modelo es:

	Predicción		
Observ.	S	N	Error clasif.
S	2628	51	0.01903695
N	89	688	0.11454311

Y la obtenida al ejecutar el modelo sobre el conjunto de validación es:

	Predicción		
Observ.	S	N	Error clasif.
S	647	16	0.02409639
N	31	170	0.1542289

Este **modelo** se considera, objetivamente, **muy adecuado para explicar la variabilidad observada en los datos recogidos** con el fin de **valorar el rendimiento de los modelos presentados**. Y para usarse como clasificador y **predecir la resolubilidad de nuevas instancias** cuyas características estén incluidas dentro de los valores paramétricos estudiados.

2.4.6. Otro estudio del rendimiento de los modelos

En [34] se puede encontrar otro estudio del rendimiento ofrecido para los modelos analizados en este documento. **Las condiciones de realización de éste son muy diferentes** a las que se han considerado aquí:

- Una CPU con apenas dos núcleos (vs 4 núcleos) y la mitad de memoria caché disponible (4MB de caché L2 vs 8MB de SmartCache).
- Seis veces menos de memoria Ram disponible (2GB DDR2 vs 12GB DDR3).
- Diferente sistema operativo (Windows XP vs Ubuntu 16.04).
- Distinto lanzador (C++ vs R).
- Diferentes niveles para los factores del problema ($n \in \{4, 6, 8, 10, 12, 14, 16\}$, $m \in \{2, 3, 4, 5\}$, $F \in \{2, 3, 4\}$) y para los computacionales (1 y 2 núcleos vs 3 y 4 núcleos; CPLEX 11 vs CPLEX 12.7.1).
- Y diferentes instancias generadas (disponibles en *soa.iti.es*).

Se incluye la imagen 2.16, que resume el análisis descriptivo del rendimiento realizado por [34]. Sus datos señalan que:

- **El modelo quinto es el que ofrece mejor tasa de óptimos y mejor tiempo de computación, aunque con valores muy cercanos a los ofrecidos por el modelo sexto, bajo las anteriores condiciones**.
- **El modelo tercero proporciona un menor GAP medio bajo las anteriores condiciones, pese a que los valores de GAP medio del modelo sexto están bastante próximos en esas condiciones**.

Se lleva a cabo un análisis estadístico con técnicas CHAID. Los factores estudiados fueron n, m, F, el modelo usado, el tiempo disponible, y el número de núcleos permitidos. La variable respuesta fue el tipo de solución ofrecida por CPLEX (si era óptima o no).

Este análisis señala, de nuevo, que **el factor más importante es n, seguido del modelo computado, de F y del tiempo disponible de computación**. El número de núcleos disponibles para computar, con esos dos niveles y bajo esas condiciones de computación, no es significativo estadísticamente.

Model	Time limit (s)				Average
	300		1800		
	Serial	Parallel	Serial	Parallel	
1					
% opt	44.52	46.43	48.33	49.52	47.20
GAP%	15.75	15.69	14.60	14.52	15.14
Time	175.45	169.28	973.67	948.07	566.62
2					
% opt	44.05	46.67	50.24	54.05	48.75
GAP%	9.04	8.05	6.33	5.41	7.21
Time	179.82	169.76	957.84	899.96	551.84
3					
% opt	51.19	55.00	57.86	61.67	56.43
GAP%	7.12	6.20	4.82	4.57	5.68
Time	157.06	148.98	824.20	759.78	472.51
4					
% opt	47.86	50.00	53.81	56.67	52.08
GAP%	14.73	13.06	11.48	9.66	12.23
Time	165.10	158.44	890.00	847.88	515.35
5					
% opt	55.48	58.10	61.43	63.33	59.58
GAP%	12.69	12.20	11.06	10.67	11.66
Time	140.21	135.46	753.89	709.07	434.66
6					
% opt	53.10	57.14	58.10	62.62	57.74
GAP%	9.51	7.86	7.40	6.79	7.89
Time	147.31	138.27	796.67	733.39	453.91
Average					
% opt	49.37	52.22	54.96	57.98	53.63
GAP%	11.47	10.51	9.28	8.60	9.97
Time	160.82	153.36	866.04	816.36	499.15

Figura 2.16: Imagen tomada de [34]. Rendimiento ofrecido por los diferentes modelos sobre Intel Core 2 Duo E6000 con 2GB de RAM sobre un Windows XP. El solver fue CPLEX 11, lanzado a través de un programa escrito en C++.

2.5. Métodos heurísticos y metaheurísticos para el problema $DF|prmu|C_{\text{máx}}$

Como se avanzaba al principio de la sección 2.3, los **modelos** matemáticos planteados resultarían útiles para **resolver, únicamente, problemas de planificación de la producción pequeños**. Y, por tanto, sería **imprescindible** usar **métodos heurísticos** o metaheurísticos para obtener, al menos, soluciones aceptables, en tiempos razonables, a problemas de planificación grandes.

La sección 2.4 respalda las anteriores afirmaciones en el caso del problema $DF|prmu|C_{\text{máx}}$ y pone en relieve, expresamente, la necesidad de desarrollar métodos heurísticos y metaheurísticos para resolverlo.

Esta sección está organizada en 4 subsecciones principales:

- Una **primera subsección**, que conforma una **primera aproximación al desarrollo de heurísticos** para el problema DPFSP cuando se pretende minimizar el tiempo máximo de completación de todas las tareas, entre todas las fábricas.
- Una **segunda subsección**, que desarrolla dos **métodos VND** (*Variable Neightbourhood Descent*) para resolver el problema $DF|prmu|C_{\text{máx}}$, **en cuanto a que los anteriores métodos resultarán ser limitados y medianamente efectivos**.
- Una **tercera subsección**, que presenta un **algoritmo de búsqueda dispersa para resolver el problema**, **como respuesta** al principal inconveniente de los métodos **VND**: qué están **fuertemente condicionados por la solución inicial** proporcionada.
- Y una **cuarta subsección**, que reproduce un estudio estadístico para **valorar el rendimiento que ofrecen estos métodos heurísticos y metaheurísticos** para resolver el problema $DF|prmu|C_{\text{máx}}$.

Hacer notar que, en la siguiente sección, la 2.6, se incluirá una **breve recopilación de métodos** (heurísticos y metaheurísticos) para resolver la planificación en problemas $DF|prmu|C_{\text{máx}}$. Y que, por el carácter de este documento, ha resultado **imposible incorporarlos como parte del análisis detallado realizado**.

2.5.1. Una primera aproximación en el desarrollo de heurísticos efectivos

Las investigaciones llevadas a cabo durante la segunda mitad del siglo XX produjeron grandes avances en la resolución, mediante heurísticos, de los problemas de planificación de la producción **F|prmu|C**$_{\text{máx}}$. Se desarrollaron, principalmente, **6 heurísticos eficaces**[xxv]:

a) La **regla SPT** (*Shortest Processing Time*), [1].

b) La **regla LP**T (*Largest Processing Time*), [1].

c) La **regla de Johnson**, cuando se dispone de, únicamente, dos máquinas, [22].

d) La **regla de Palmer**, [37].

e) La **regla CDS** (*Campbell-Dudek-Smith*), [5].

f) El **algoritmo NEH** (*Nawaz-Enscore-Ham*), [36].

Estos heurísticos permitieron, en tiempos razonables, programar la ejecución de los trabajos para el problema $F|prmu|C_{\text{máx}}$. Ahora bien, la **pregunta** natural que surge, por tanto, es si éstos **pueden ser modificados y empleados para resolver el problema DF|prmu|C**$_{\text{máx}}$.

Un problema, recuérdese, mucho más complejo porque debe establecer el reparto de los trabajos entre las diferentes factorías y, además, programar la ejecución de ellos en éstas.

Una **primera forma de obtener heurísticos** para el problema en cuestión consistirá en:

- **Construir una secuencia de procesamiento** de los trabajos **a partir de los heurísticos del problema F|prmu|C**$_{\text{máx}}$ (salvo con el algoritmo NEH[xxvi]).
- **Y usar una regla para, posteriormente, decidir en qué fábrica se realiza cada trabajo.**

En [34] se proponen dos reglas para establecer este reparto entre fábricas después de tener establecida una secuencia de procesamiento de los trabajos.

Regla NR 1. **Asignar el trabajo j a la factoría que, tras las asignaciones previas y sin considerar este trabajo, tenga un C**$_{\text{máx}}$ **menor**.

Regla NR 2. **Asignar el trabajo j a la factoría que, tras asignarle este trabajo, tenga un C**$_{\text{máx}}$ **menor**.

Estas dos reglas resultan, conceptualmente y computacionalmente, sencillas. Tanto es así que:

- **La primera regla no añade ningún coste computacional** sobre el correspondiente a establecer una secuencia base, de procesamiento de los trabajos.
- Y **la segunda** requiere secuenciar, adicionalmente sobre la secuencia base, las m tareas que conforman cada uno de los trabajos a programar en todas las fábricas. Es decir, se **añade una complejidad del orden mF** ($\mathcal{O}(mF)$).

[xxv] En el apéndice A se describirán estos heurísticos.

[xxvi] Como se explica en el anexo A, este heurístico va construyendo, iterativamente, la secuenciación final a partir de secuenciaciones parciales. En contra de suponer que existe una única fábrica y, a partir de ahí, generar la secuencia final, se modificará el algoritmo para adaptarlo a un entorno multifábrica.

A continuación y por completitud, se incluye un **ejemplo de aplicación de estas dos reglas** para aclarar, si existiera, una posible ambigüedad en ellas.

Supongamos que se desea programar la ejecución de $n = 5$ trabajos sobre $m = 2$ máquinas en $F = 2$ fábricas idénticas. Los tiempos de procesamiento usados para el ejemplo se detallan en la tabla 2.1.

	Trabajo 1 (*J1*)	*J2*	*J3*	*J4*	*J5*
Máquina 1 (*M1*)	10	6	8	9	3
Máquina 2 (*M2*)	5	7	4	6	11

Tabla 2.1: Sobre los tiempos de procesamiento usados como ejemplo, para programar la ejecución de $n = 5$ trabajos sobre $m = 2$ máquinas en $F = 2$ fábricas idénticas

Y supongamos que un heurístico de los anteriores ha proporcionado la siguiente secuencia de procesamiento

$$\theta = \{J3, J5, J1, J4, J2\}.$$

Aplicación de la Regla 1.

Denotaremos por $C_{\text{máx}}^{(f)}$ al tiempo que tarda la fábrica f en completar todas sus tareas asignadas hasta ese momento. Por tanto, la aplicación de la regla resultará de la siguiente forma:

- Se comienza seleccionado $J3$, el primero de la secuencia. Como ambas fábricas no tienen asociado ningún trabajo, no existe diferencia entre ellas. Por tanto, se asigna $J3$ a la primera fábrica.
- Se selecciona el siguiente en la secuencia, $J5$. Como
$$\begin{cases} C_{\text{máx}}^{(1)} = 12 \\ C_{\text{máx}}^{(2)} = 0 \end{cases}$$
se asigna $J5$ a la segunda fábrica (al tener ésta un menor tiempo de completación).
- Se selecciona el siguiente en la secuencia, $J1$. Como
$$\begin{cases} C_{\text{máx}}^{(1)} = 12 \\ C_{\text{máx}}^{(2)} = 14 \end{cases}$$
se asigna $J1$ a la primera fábrica (al tener ésta un menor tiempo de completación).
- Y se procede de forma análoga con el resto de los trabajos que conforman la secuencia.

Se obtiene, tras acabar el proceso, la asignación de trabajos detallada en la figura 2.17.

Aplicación de la Regla 2.

Denotaremos por $C_{\text{máx}}^{(f)}$ al tiempo que tardaría la fábrica f en completar todas las tareas asignadas hasta ese momento y la que se pretende asignar en la correspondiente iteración del proceso.

El proceso iterativo de aplicación de la regla es el siguiente:

- Se comienza seleccionado $J3$, el primero de la secuencia. Como ambas fábricas no tienen asociado ningún trabajo, no existe diferencia entre ellas. Por tanto, se asigna $J3$ a la primera fábrica.
- Se selecciona el siguiente en la secuencia, $J5$. Como
$$\begin{cases} C^1_{\text{máx}} = 12 + 11 = 23 \\ C^2_{\text{máx}} = 0 + 14 = 14 \end{cases}$$
se asigna $J5$ a la segunda fábrica (al presentar ésta un menor tiempo de completación tras asignársele).
- Se selecciona el siguiente en la secuencia, $J1$. Como
$$\begin{cases} C^1_{\text{máx}} = 12 + 11 = 23 \\ C^2_{\text{máx}} = 14 + 6 = 19 \end{cases}$$
se asigna $J1$ a la segunda fábrica (al presentar ésta un menor tiempo de completación tras asignársele).
- Y se procede de forma análoga con el resto de los trabajos que conforman la secuencia.

Se obtiene, tras acabar el proceso, la asignación de trabajos detallada en la figura 2.18.

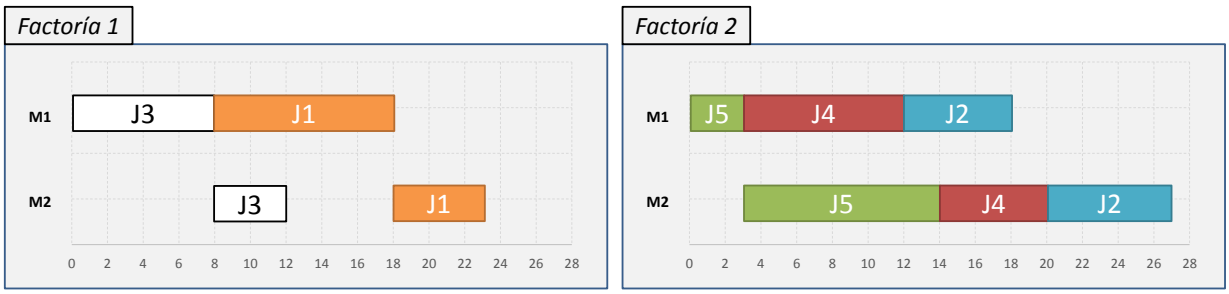


Figura 2.17: Se representa en sendos diagramas de Gannt la **programación de la producción propuesta como ejemplo usando la primera regla**. Destáquese que, en el tiempo 23, la primera fábrica finaliza su producción. Y que la segunda fábrica lo hace en el instante 27. Por tanto, $C_{\text{máx}} = 27$.

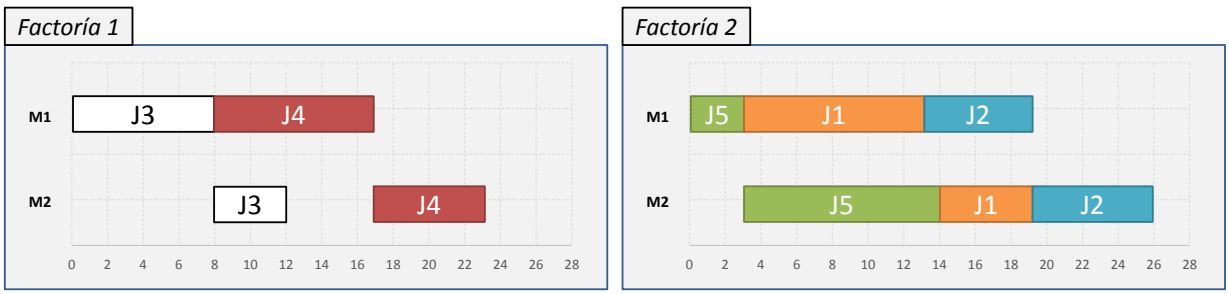


Figura 2.18: Se representa en sendos diagramas de Gannt la **programación de la producción propuesta como ejemplo usando la segunda regla**. Destáquese que, en el tiempo 17, la primera fábrica finaliza su producción. Y que la segunda fábrica lo hace en el instante 26. Por tanto, $C_{\text{máx}} = 26$.

Otra posible **vía** para generar un nuevo heurístico consiste en **modificar**, ligeramente, el **algoritmo NEH original para adaptarlo a un entorno multifábrica**.

A partir de las dos reglas citadas anteriormente y del algoritmo NEH original, [34] presenta el siguiente **algoritmo adaptado** al problema **DF|prmu|C**$_{\text{máx}}$:

1. Se calcula el tiempo total necesario para procesar cada trabajo en todas las máquinas
$$P_j = \sum_{i=1}^{m} P_{ij}$$
2. Los trabajos son ordenados en función de P_j, en orden descendente, y se obtiene, así, una secuencia de producción
3. Se procede a la asignación por fábricas y la inclusión en la producción de cada una de ellas usando una de estas reglas:

   Regla NEH 1. **Asignar el trabajo** j **a la factoría que, tras las asignaciones previas y sin considerar este trabajo, tenga un C**$_{\text{máx}}$ **menor. Una vez asignado, el trabajo se insertará en todas las posibles posiciones de la secuencia** de producción de la fábrica **y se seleccionará aquella que produzca un menor C**$_{\text{máx}}$

   Regla NEH 2. **Asignar el trabajo** j **a la factoría que, tras asignarle este trabajo, tenga un C**$_{\text{máx}}$ **menor. Para calcular este C**$_{\text{máx}}$ **en cada fábrica, el trabajo se insertará en todas las posibles posiciones** de la secuencia de producción de la fábrica **y se seleccionará aquella que produzca un menor C**$_{\text{máx}}$.

De nuevo, para ejemplificar su aplicación, se hará uso del problema DPFSP propuesto a través de la tabla 2.1.

Aplicación de la Regla NEH 1.

Denotaremos por $C_{\text{máx}}^{(f)}$ al tiempo que tarda la fábrica f en completar todas sus tareas asignadas hasta ese momento. Por tanto, tras la lista ordenada de trabajos,
$$L = \{J1, J4, J5, J2, J3\},$$
la aplicación de la regla resultará de la siguiente forma:

- Las dos primeras iteraciones de la regla asignarán, trivialmente, el trabajo $J1$ a la primera fábrica y $J2$ a la segunda fábrica.
- Tras esto, la regla tratará de determinar sobre que fábrica y en que posición de la secuencia se debe insertar el trabajo $J5$.

  La regla asignará, rompiendo el empate, el trabajo $J5$ a la fábrica 1 tras calcular
$$\begin{cases} C_{\text{máx}}^{(1)} = 15 \\ C_{\text{máx}}^{(2)} = 15 \end{cases}$$

Sobre ella probará las dos posibles secuencias de producción:

a. Fábrica 1: $\{J1, J5\}$ y Fábrica 2: $\{J4\}$, con $C_{\text{máx}} = 26$
b. Fábrica 1: $\{J5, J1\}$ y Fábrica 2: $\{J4\}$, con $C_{\text{máx}} = 19$

Y seleccionará la secuencia b.

- A continuación, la regla selecciona del listado el trabajo $J2$. Como

$$\begin{cases} C^{(1)}_{\text{máx}} = 19 \\ C^{(2)}_{\text{máx}} = 15 \end{cases}$$

asignará este trabajo a la fábrica segunda.

Sobre ésta probará las dos posibles secuencias de producción:

a. Fábrica 1: $\{J5, J1\}$ y Fábrica 2: $\{J4, J2\}$, con $C_{\text{máx}} = 22$
b. Fábrica 1: $\{J5, J1\}$ y Fábrica 2: $\{J2, J4\}$, con $C_{\text{máx}} = 21$

Por tanto, selecciona la secuencia b.

- Por último, la regla procesa el trabajo $J3$ del listado. Como

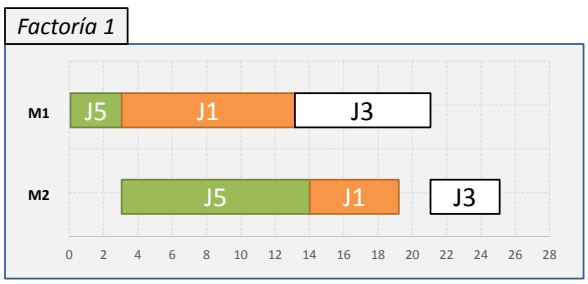
$$\begin{cases} C^{(1)}_{\text{máx}} = 19 \\ C^{(2)}_{\text{máx}} = 21 \end{cases}$$

asignará este trabajo a la fábrica primera.

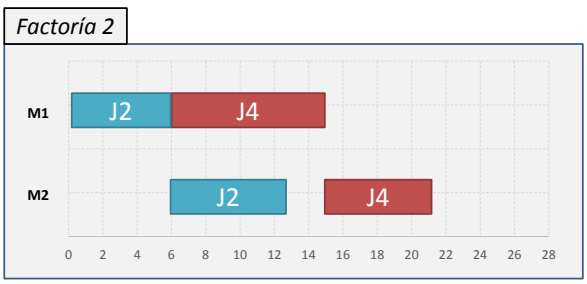
La regla asigna la secuencia a. de las tres posibles secuencias de producción:

a. Fábrica 1: $\{J5, J1, J3\}$ y Fábrica 2: $\{J2, J4\}$, con $C_{\text{máx}} = 25$
b. Fábrica 1: $\{J5, J3, J1\}$ y Fábrica 2: $\{J2, J4\}$, con $C_{\text{máx}} = 26$
c. Fábrica 1: $\{J3, J5, J1\}$ y Fábrica 2: $\{J2, J4\}$, con $C_{\text{máx}} = 28$

Se obtiene, tras acabar el proceso, la asignación de trabajos detallada en la figura 2.19.



(a) Secuenciación propuesta para la fábrica 1



(b) Secuenciación propuesta para la fábrica 2

Figura 2.19: Este diagrama de Gannt representa la **programación de la producción propuesta** por las dos reglas de [34], NEH 1 y NEH 2, para el problema de ejemplo.

Aplicación de la Regla NEH 2.

El proceso iterativo de aplicación de la regla es el siguiente:

- Las dos primeras iteraciones de la regla asignarán, trivialmente, el trabajo $J1$ a la primera fábrica y $J2$ a la segunda fábrica.
- Tras esto, la regla tratará de determinar sobre que fábrica y en que posición de la secuencia se debe insertar el trabajo $J5$.

  Para ello, la regla secuenciará la producción de estas cuatro secuencias

  a. Fábrica 1: $\{J1, J5\}$ y Fábrica 2: $\{J4\}$, con $C_{\text{máx}} = 26$
  b. Fábrica 1: $\{J5, J1\}$ y Fábrica 2: $\{J4\}$, con $C_{\text{máx}} = 19$
  c. Fábrica 1: $\{J1\}$ y Fábrica 2: $\{J4, J5\}$, con $C_{\text{máx}} = 26$
  d. Fábrica 1: $\{J1\}$ y Fábrica 2: $\{J5, J4\}$, con $C_{\text{máx}} = 20$

  Y seleccionará la secuencia b.

- A continuación, la regla selecciona del listado el trabajo $J2$. Se calculará, para este trabajo, la mejor posición para insertarlo sobre la secuencia obtenida en la anterior iteración:

  a. Fábrica 1: $\{J5, J1\}$ y Fábrica 2: $\{J4, J2\}$, con $C_{\text{máx}} = 22$
  b. Fábrica 1: $\{J5, J1\}$ y Fábrica 2: $\{J2, J4\}$, con $C_{\text{máx}} = 21$
  c. Fábrica 1: $\{J5, J1, J2\}$ y Fábrica 2: $\{J4\}$, con $C_{\text{máx}} = 26$
  d. Fábrica 1: $\{J5, J2, J1\}$ y Fábrica 2: $\{J4\}$, con $C_{\text{máx}} = 26$
  e. Fábrica 1: $\{J2, J5, J1\}$ y Fábrica 2: $\{J4\}$, con $C_{\text{máx}} = 29$

  Por tanto, selecciona la secuencia b.

- Por último, la regla procesa el trabajo $J3$ del listado. La regla asigna la secuencia a. de las seis posibles secuencias de producción:

  a. Fábrica 1: $\{J5, J1, J3\}$ y Fábrica 2: $\{J2, J4\}$, con $C_{\text{máx}} = 25$
  b. Fábrica 1: $\{J5, J3, J1\}$ y Fábrica 2: $\{J2, J4\}$, con $C_{\text{máx}} = 26$
  c. Fábrica 1: $\{J3, J5, J1\}$ y Fábrica 2: $\{J2, J4\}$, con $C_{\text{máx}} = 28$
  d. Fábrica 1: $\{J5, J1\}$ y Fábrica 2: $\{J2, J4, J3\}$, con $C_{\text{máx}} = 27$
  e. Fábrica 1: $\{J5, J1\}$ y Fábrica 2: $\{J2, J3, J4\}$, con $C_{\text{máx}} = 29$
  f. Fábrica 1: $\{J5, J1\}$ y Fábrica 2: $\{J3, J2, J4\}$, con $C_{\text{máx}} = 29$

Tras acabar el proceso, la asignación de trabajos que se obtiene es la misma que la proporcionada por la regla NEH 1. Ésta se detalla en 2.19.

Otra posibilidad para generar un nuevo heurístico para el problema **DF|prmu|C**$_{\text{máx}}$ a partir del **algoritmo NEH original** se detalla en [10]. Se trata de un *refinamiento* de los heurísticos basados en NEH presentados en [34].

Está estructurado en tres etapas, que se pasan a detallar:

- **Primera etapa**. Elaboración de una secuencia de selección de trabajos.

  Se calcula, para todos los trabajos a programar, el tiempo medio y la desviación típica del tiempo de completación requerido para llevarlos a cabo. Se hace mediante las siguientes expresiones

$$\begin{cases} \text{AVG}(j) = \dfrac{1}{m}\displaystyle\sum_{i=1}^{m} P_{ij} \\ \\ \text{STD}(j) = \left(\dfrac{1}{m-1}\displaystyle\sum_{i=1}^{m}\left(P_{ij} - \text{AVG}(j)\right)^2\right)^{1/2} \end{cases}$$

  Tras ello, los trabajos se orden en orden decreciente en una lista, L, a partir del valor de la expresión

$$\text{AVG}(j) + \text{STD}(j).$$

- **Segunda etapa**: Asignación de trabajos.

  Esta asignación se produce mediante dos procedimientos distintos:

  - **Procedimiento 1**. Se toma el primer trabajo no asignado de la lista L y, mediante la ***Regla NEH 2***, se encuentra una fábrica en la que, tras asignarlo, no aumente el $C_{\text{máx}}$ parcial y total del problema.

  - **Procedimiento 2**. Se usa la siguiente regla para procesar y asignar, de golpe, F trabajos a la secuencia de producción:

    Regla NEH F: Se toman los primeros F trabajos no asignados de L y se determina, mediante un procedimiento *Branch & Bound* no especificado, la asignación final entre las diferentes fábricas y los F trabajos seleccionados. [10] y la literatura posterior no aclaran diseños ni métodos para formular en *Branch & Bound* que usa esta regla.

  El procedimiento 1 se aplicará por defecto, para asignar el primer trabajo de L que no esté fijado a ninguna fábrica. **Si no es posible** encontrar la fábrica de la que habla este procedimiento, **se usará el procedimiento 2**, que asignará F trabajos de golpe.

  Tras la asignación y programación de la secuenciación**, se repite el proceso mientras que en la lista hayan sin asignar más de F − 1 trabajos**.

- **Tercera etapa**: Finalización.

  El **resto de los trabajos** que quedan sin asignar a una fábrica (un número menor que F), se hacen mediante la ***Regla NEH 2***.

El estudio que realizan los autores de este heurístico, [10], pone de manifiesto que:

- Este heurístico **no** produce una **mejora** significativa con respecto al heurístico formado a través de la regla *Regla NEH 2* de [34].
- Y que, al emplearse, se **añade** una cierta **complejidad computacional**. Según los datos que presentan, supone un aumento ligeramente superior al 15 % con respecto a la regla *Regla NEH 2* de [34].

Estas afirmaciones, que también realiza [35], van en contra de las conclusiones que los autores presentan en su estudio. Unas conclusiones, por cierto, basadas en la siguiente gráfica (extraída del propio artículo). **Nótese que los intervalos de confianza al 95 % para los métodos NEH2 (*Regla NEH 2*) y NEH-df (*Regla NEH F*) son compatibles.**

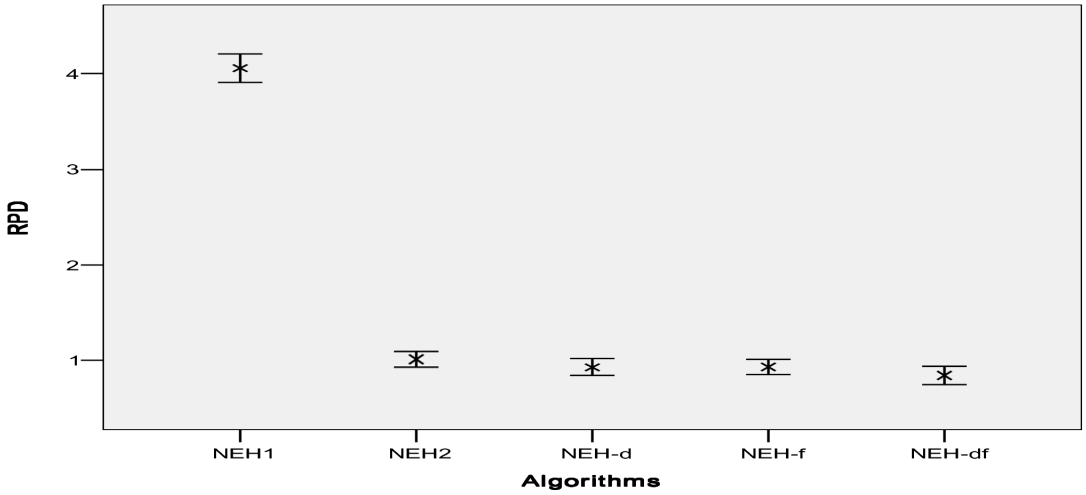


Figura 2.20: Imagen tomada de [10]. Se representan los intervalos de confianza al 95 % de la media de la variable RPD, para cada método, sobre el total de instancias usadas. La variable RPD representa la desviación (en porcentaje relativo) de la solución obtenida (por el método indicado) entre la mejor solución conocida para la instancia en cuestión.

2.5.2. Algoritmos de búsqueda VNS. Estudio de los métodos metaheurísticos VND propuestos para resolver el problema DF|prmu|$C_{máx}$

Agotada la anterior vía para generar heurísticos que resuelvan, aceptablemente y en tiempos razonables, el problema de optimización $DF|prmu|C_{máx}$, **diversos autores** ([34], [7], [49]...) **han optado por generar metaheurísticos a través de algoritmos de búsqueda de vecindad variable**.

Un algoritmo de **Búsqueda de Vecindad Variable** (*Variable Neighbourhood Search*, ***VNS***) es tipo de algoritmo **metaheurístico** (*Single-Solution Based Metaheuristics*), originalmente formulado por [33] a finales del siglo XX, para la resolución de problemas de optimización.

Se trata de un **método de búsqueda local** ampliamente usado para resolver, rápidamente y sin garantías de optimalidad, problemas de secuenciación de tareas, de horarios, de transporte, de localización o de diseño de rutas, entre otros ([16], [40] y [42]). Se puede entender como un **proceso iterativo**,

- que **comienza con una solución** del problema, y
- que, mediante **cambios sistemáticos**[XXVII] en la estructura de *vecindad* durante su ejecución, **va mejorándola**[XXVIII], con la idea de **alcanzar diferentes óptimos locales**.

Este proceso de búsqueda está **sustentado en tres hechos simples**:

- Un mínimo local, con respecto a una estructura de vecindad, no lo es necesariamente con respecto a otra.
- Un mínimo global es un mínimo local con respecto a todas las posibles estructuras de vecindades.
- Empíricamente, en muchos problemas, el mínimo local con respecto a una o varias estructuras de vecindad está relativamente cerca.

Debido a su versatilidad, se han creado **diferentes variantes** del método original, entre las que se destacan:

- *Variable Neighbourhood Descent*, **VND**.
- *Basic Variable Neighbourhood Search*, **BVNS**.
- *Reduced Variable Neighbourhood Search*, **RVNS**.
- *General Variable Neighbourhood Search*, **GVNS**.

Y, aunque todas ellas constituyen una gran metodología de trabajo para abordar problemas de optimización, por el carácter que el autor pretende dar a este texto, **en este documento** únicamente se analizarán los pasos dados con el **método VND para resolver**, aceptablemente y en tiempos razonables, el problema de optimización **DF|prmu|$C_{máx}$**.

Por ello se estudiará y analizará con detalle los dos métodos VND propuestos en [34].

[XXVII] Estos cambios caracterizan cada proceso de búsquedas y, junto con la solución inicial, constituyen un factor clave para la eficacia del método.

[XXVIII] Al encontrar, durante la búsqueda, una solución que mejora la que dispone en ese momento, la remplaza y continúa el proceso, hasta que no pueda mejorarla más.

Conviene aclarar, antes de comenzar a desarrollar el contenido de esta sección, **dos aspectos generales en cuanto a** los métodos **VND**:

- Puesto que los algoritmos VND son métodos metaheurísticos basados en VNS, la idea subyacente en ellos es la **búsqueda sistemática**, y **en diferentes estructuras de vecindad**, de soluciones del problema, **hasta obtener un mínimo local**. Este mínimo local es mínimo con respecto a todos los esquemas de vecindad estudiados.
- Las vecindades **son exploradas siguiendo un orden** establecido previamente, **en función del tiempo estimado que llevará su evaluación** (desde las que suponen un bajo coste computacional hasta aquellas que acarrean un alto grado de complejidad).

Y, de igual forma, se hace necesario **señalar sobre los dos métodos VND propuestos** por [34], lo siguiente:

- Las **soluciones** que van a ir explorando sendos métodos **se representarán de forma indirecta**, **mediante** F **listas** (una para cada fábrica del problema). Cada lista contendrá el orden con el que se deben procesar los trabajos en la respectiva fábrica. Y, por tanto, será necesario un proceso de cálculo para obtener el $C_{\text{máx}}$ del problema.
- Puesto que los **métodos VND deben partir de una solución posible del problema** y ésta condiciona fuertemente los resultados obtenidos, [34] propone **usar**, para sus métodos VND, la solución **obtenida a través de la regla NEH 2**.

Estudio de métodos VND para resolver DF|prmu|$C_{\text{máx}}$

Los métodos propuestos por [34] hacen uso de dos búsquedas locales:

- **Una** que **trata de mejorar la secuenciación de trabajos dentro de cada fábrica**, y que se denotará como **LS 1** (*Local Search 1*).
- Y **otra**, que **intenta mejorar la asignación de trabajos entre las fábricas**, y que se denotará como **LS 2** (*Local Search 2*).

Cada uno de estas dos búsquedas locales se transcriben en los algoritmos 2.1 y 2.2. E irán seguidas de sendas explicaciones de su funcionamiento.

Lo que diferenciará los dos métodos VND estudiados será el criterio de aceptación (del resultado de la búsqueda local) **usado durante la ejecución del algoritmo 2.2**. Criterios que, por cierto, se detallarán mientras se desarrolla esta búsqueda local.

Por tanto, la **estructura general de ejecución de los dos métodos VND** resultará idéntica. Ésta se detallará **en el algoritmo 2.3**.

Algoritmo 2.1 Búsqueda local LS 1 sobre la fábrica f, con f_f trabajos asignados

Entrada: Una fábrica, f, y la secuencia de producción asignada a ella
Salida: Una secuencia de producción para f que reduce o mantiene el $C_{\text{máx}}$ de la fábrica

1: `Mejora` ← **cierto**
2: **mientras** `Mejora` **hacer**
3: `Mejora` ← **falso**
4: **para** `i:=1` **hasta** f_f **hacer**
5: Se retira de la secuencia de producción de la factoría f el trabajo k, que ocupa la posición i-ésima en ésta.
6: Se inserta el trabajo k en todas las posibles posiciones nuevas de la secuencia de producción. Y se calcula en cada caso el $C_{\text{máx}}$ de la fábrica con esa secuenciación.
7: **si** alguna nueva secuenciación obtiene un mejor $C_{\text{máx}}$ para la fábrica f **entonces**
8: Se actualiza la secuenciación de la producción de la factoría f, insertando el trabajo k en la posición en la que el $C_{\text{máx}}$ de la fábrica es menor.
9: `Mejora` ← **cierto**
10: **fin si**
11: **fin para**
12: **fin mientras**

La búsqueda local *LS 1* sobre una determinada fábrica se produce **iterativamente**:

- Supogamos que *LS 1* se ejecuta sobre la fábrica f. El algoritmo, en cada paso, irá seleccionando un trabajo asignado a esa fábrica, hasta que haya probado con todos ellos (en total, f_f).
- Si suponemos que el trabajo seleccionado es k y éste ocupa la posición i-ésima en la secuencia de producción de la fábrica, el algoritmo lo desasignará de esta posición y probará a insertarlo en todas las posibles posiciones restantes de la secuencia de producción.
- Finalmente, el algoritmo asignará el trabajo k en la posición en la que la fábrica f tenga un menor $C_{\text{máx}}$. En caso de empate entre su posición original y otra, regresará a su posición original, i.
- Tras finalizar la ejecución del algoritmo, si en alguna de las iteraciones, el correspondiente trabajo no ha regresado a su posición original en la secuencia de producción, *LS 1* volverá a ejecutarse.

Algoritmo 2.2 Búsqueda local LS 2 sobre la fábrica, y, cuyo $C_{\text{máx}}$ es el del problema

Entrada: Una fábrica, y, con f_y trabajos asignados y cuyo $C_{\text{máx}}$ sea el mayor entre los de las restantes fábricas. El listado de trabajos asignados a y. Y la secuenciación de la producción en el resto de fábricas.

Salida: Una asignación de trabajos entre fábricas que reduce o mantiene el $C_{\text{máx}}$ del problema

1: **para** i:=1 **hasta** f_y **hacer**

2: Se retira de la secuencia de producción de la factoría y el trabajo k, que ocupa la posición i-ésima en ésta.

3: **para** f=1 **hasta** F **hacer**

4: **si** $f \neq y$ **entonces**

5: Se inserta el trabajo k en todas las posibles posiciones de la secuencia de producción de la fábrica f. Y se calcula, en cada caso, las variables implicadas en el criterio de aceptación del movimiento.

6: **fin si**

7: **fin para**

8: Se devuelve el trabajo k a su fábrica y posición inicial

9: **fin para**

10: **si** se cumple, para algún movimiento, el criterio de aceptación **entonces**

11: Se realiza el movimiento mejor valorado. El trabajo se desasigna de y, y se asigna adecuadamente en la secuencia de producción de la fábrica correspondiente.

12: **fin si**

Por cuestión de uniformidad de **notación**:

- Se utilizará el **subíndice** x en la expresión C_x para denotar al tiempo de completación máximo de la factoría x.
- El **subíndice** máx en $C_{\text{máx}}$ se empleará para referirse al tiempo de completación máximo del problema.
- En general, **la fábrica y será la que requiera mayor tiempo para completar sus tareas** (y por tanto $C_{\text{máx}} = C_y$). **El algoritmo, a efectos de notación, propondrá mover el trabajo k** de ésta fábrica, **hacia la fábrica x**. Allí, el movimiento propondrá insertarlo en la posición (de la correspondiente secuencia de producción) en la que la fábrica x experimente un menor tiempo de completación de todas sus tareas.
- Se usará el **superíndice** $*$, para denotar a los nuevos tiempos de completación del problema ($C^*_{\text{máx}}$) y de cada fábrica (C^*_x) causados por el movimiento estudiado.

La búsqueda local *LS 2* está **fundamentada en** las tres siguientes observaciones:

- Como resulta conocido, el $\mathbf{C}_{\text{máx}}$ **del problema** DPFSP viene marcado por la **factoría que requiera mayor tiempo para completar sus tareas**. Esto es, el $C_{\text{máx}}$ del problema será el mayor C_x, con $x \in \{1, \ldots, F\}$.
- Por tanto, una **estrategia para mejorar** el $C_{\text{máx}}$ actual del problema consiste en **reasignar algún de trabajo de la fábrica más ocupada a otra** de las fábricas del problema, **más desocupada**.
- Para **elegir que trabajo se desasigna** de y y como se asigna a otra fábrica y posición, **se seguirá un proceso** sistemático, **razonado en algún criterio** que permita seleccionar el mejor movimiento de tareas entre todos los contemplados.

La búsqueda local *LS 2* procederá de forma iterativa **para determinar el trabajo a reasignar, la fábrica receptora implicada y la nueva secuenciación**:

- En cada paso irá seleccionando un trabajo asignado a la fábrica y, hasta que haya probado con todos ellos.
- Si suponemos que el trabajo seleccionado es el k, éste se desasignará temporalmente de la fábrica y. Y se probará su asignación en cada una de las restantes fábricas del problema.
- Una vez asignado, temporalmente, a una fábrica, el algoritmo probará a insertarlo en todas las posibles posiciones de la secuencia de producción de esta fábrica. En cada paso, con cada prueba de inserción, el algoritmo calculará el **criterio de aceptación del movimiento**.
- Al acabar, con el trabajo k, todas las pruebas en todas las fábricas, éste se devolverá a su posición inicial y se seguirá con el siguiente trabajo asignado a y.
- **Al acabar el proceso, se seleccionará aquel movimiento (fábrica y posición en la secuencia de producción) mejor valorado por el criterio**[xxix], para realizarse definitivamente.

Cada criterio de aceptación generará un método VND diferente. [34] propone estos **dos criterio**s:

VND a. **Aceptar aquel movimiento que mejore** el $\mathbf{C}_{\text{máx}}$ **del problema**. Esto es, que cumpla que:

$$C^*_{\text{máx}} < C_{\text{máx}} \quad \text{y} \quad C^*_x < C_{\text{máx}}$$

VND b. **Aceptar aquel movimiento que mejore el reparto de trabajos entre las dos factorías involucradas**. Esto es, que se cumpla:

$$\left(C_{\text{máx}} - C^*_{\text{máx}}\right) + \left(C_x - C^*_x\right) > 0$$

Hacer explícito ambas búsquedas locales **VND finalizan afectando, únicamente, a dos fábricas**. A la y y a la x. **El resto de fábricas y de secuenciaciones permanecen inalterada tras la ejecución de *LS 2*.**

[xxix] Es decir, aquel movimiento que produzca más mejora. Con respecto a las desigualdades propuestas, se seleccionará aquel movimiento que experimente mayor holgura en éstas

Algoritmo 2.3 Método VND a. y VND b.

Entrada: Datos del problema $DF|prmu|C_{\text{máx}}$, con F fábricas, n trabajos a programar y m máquinas

Salida: Una secuencia de producción aceptable para el problema

1: Se genera una secuencia de producción inicial con la *regla heurística NEH 2*

2: `Mejora` ← **cierto**

3: `Cambios`[F] ← **cierto**

4: **mientras** `Mejora` **hacer**

5: **para** f:=1 **hasta** F **hacer**

6: **si** `Cambios`[f] **entonces**

7: Ejecutar la búsqueda local LS 1 sobre f

8: `Cambios`[f] ← **falso**

9: **fin si**

10: **fin para**

11: Encontrar la factoría y (la que requiere mayor tiempo para completar sus tareas en la actual asignación y secuenciación)

12: Ejecutar la búsqueda local LS 2 sobre la factoría y, usando como criterio de aceptación a VND a. o VND b, respectivamente

13: **si** se cumple el criterio de aceptación y se traslada un trabajo de la fábrica y a la x **entonces**

14: `Cambios`[x] ← **cierto**

15: `Cambios`[y] ← **cierto**

16: **si no**

17: `Mejora` ← **falso**

18: **fin si**

19: **fin mientras**

2.5.3. Un método de Búsqueda Dispersa para $DF|prmu|C_{\text{máx}}$

Una de los inconvenientes de los **métodos VND** presentados anteriormente es que, al ser métodos simples y fundamentados en búsquedas locales, están **fuertemente condicionados por la solución inicial proporcionada**. Este comportamiento se pone de manifiesto cuando la solución inicial conduce a un óptimo local muy alejado del óptimo del problema: El metaheurístico VND no es capaz de mejorar la solución de la que dispone y la acepta como óptimo global.

Este comportamiento, que resulta sumamente indeseable para obtener buenas soluciones, ha propiciado el **desarrollo de nuevos métodos más eficaces**. Uno de los mejores valorados, para resolver heurísticamente el problema que nos ocupa, es un **algoritmo de búsqueda dispersa** (*Scatter Search*, **SS**) diseñado por [35].

Scatter Search es un **método metaheurístico evolutivo** (*Population-Based Metaheuristics*), originalmente introducido en los años setenta y ampliamente extendido durante las dos últimas décadas para resolver problemas difíciles. Su éxito para resolverlos se debe, principalmente, a las técnicas que emplea para generar, recombinar y diversificar las soluciones de las que va disponiendo en cada momento. Estas soluciones temporales aparecen en el llamado *conjunto de referencia* del método (*RefSet*).

Aunque existen diferentes formas de estructurar los algoritmos de búsqueda dispersa, la elección que se hace para desarrollar esta sección es la que se presenta en [24] y en [32], y que también emplea [35] para presentar su investigación.

Por ello, **el método** en cuestión **constará de cinco** elementos o **subalgoritmos principales que lo vertebrarán**[xxx]. A saber:

- Un **generador de soluciones variadas** y diversas (*Diversification Generation Method*).
- Un **método de mejora de las soluciones**, tanto de las del conjunto de referencia como de las obtenidas mediante el proceso de combinación (*Improvement Method*).
- Un **método para crear y actualizar el conjunto de referencia** (*Reference Set Update Method*).
- Un **método de combinación de soluciones** (*Solution Combination Method*).
- Un **método que seleccione grupos de soluciones para ser combinadas** por el anterior método (*Subset Generation Method*).

Estos elementos se analizarán a continuación para, finalmente, presentar el método de búsqueda dispersa propuesto por [35] en el algoritmo 2.5.

Conviene aclarar, en todo caso, la forma de **representar las soluciones** de las que el método va ir haciendo uso. De idéntica forma a cómo se abordó esta cuestión en los métodos VND, la representación tendrá lugar de forma indirecta, mediante **F listas** (una para cada fábrica del problema). **Cada lista contendrá el orden con el que se deben procesar los trabajos en la respectiva fábrica**.

[xxx] En el anexo B se puede leer una descripción detallada del método Scatter Search y de los subalgoritmos que lo componen.

Diversification Generation Method

El **conjunto *RefSet*** se forma como la unión de dos conjuntos:

$$RefSet = \mathcal{H} \cup \mathcal{S}.$$

Por un lado, el conjunto $\mathcal{S}$, que se construye, en cada iteración, como un **vector de asignación de fábrica para trabajos**, **con l entradas generadas aleatoriamente**[xxxi]. El hecho de incluir la aleatoriedad en la generación de $\mathcal{S}$ se debe a la necesidad de obtener nuevas soluciones variadas y diversas en el conjunto de referencia del método.

Y por otro lado, el conjunto $\mathcal{H}$, que está **formado por las b mejores soluciones encontradas hasta ese momento** (en el formato de listas comentado anteriormente).

Inicialmente **se construye a través de 25 permutaciones**[xxxii] de los trabajos a programar:

- Veinticuatro de ellas se generan aleatoriamente.
- Y la última se obtiene ordenando los trabajos, de mayor a menor tiempo de procesamiento total necesario[xxxiii].

Usando estas permutaciones como listas ordenadas **sobre las que aplicar la regla NEH 2** se obtendrán **25 soluciones (mejoradas) del problema**. De éstas, **se seleccionarán las b mejores para incluirse**, inicialmente, en el conjunto $\mathcal{H}$.

Como quedará patente más adelante, el conjunto $\mathcal{H}$ **se irá actualizándose en cada iteración del método** y contendrá las b mejores soluciones conocidas en ese momento (*the best b visited solutions*).

[xxxi] Quizás pueda resultar algo oscuro el conjunto S porque no contiene soluciones completas. Únicamente asignaciones de trabajos a fábricas, sin considerar como se insertan éstos en ellas. Se tratará de aclarar con un ejemplo.

Si el problema de programación de la producción cuenta con 3 fábricas, un posible conjunto $\mathcal{S}$ sería $\{3, 1, 2, 1, 2, 3, 2, 3, 2\}$. Éste involucraría la asignación de 9 trabajos a las tres fábricas del problema. Entonces, el primer trabajo considerado se asignaría a la fábrica tercera. El segundo, a la primera fábrica. Y así sucesivamente, hasta considerar el noveno trabajo, que se asociaría a la segunda fábrica.

[xxxii] Entiéndase que una permutación de un conjunto X no deja de ser más que una función biyectiva de X en sí mismo. Al fin y al cabo, conforma una secuencia que ordena los elementos que incluye.

[xxxiii] Nótese que esta última permutación se obtiene de la misma forma a como el algoritmo NEH genera la lista, inicial y ordenada, de trabajos. Véase el anexo A y revísese la sección 2.5.1.

Subset Generation Method y Solution Combination Method

En los algoritmos de búsqueda dispersa resulta sumamente crucial el proceso de combinado de soluciones para obtener buenos resultados. El algoritmo que se analiza propone un **proceso iterativo y sistemático** para ello, en el que se combinan dos soluciones (una del conjunto $\mathcal{H}$ y otra del conjunto $\mathcal{S}$) para generar diversidad y variedad sobre el conjunto de referencia.

El método establece que **las soluciones a combinar son los pares del producto cartesiano de $\mathcal{H}$ y $\mathcal{S}$**, es decir,

$$\mathcal{H} \times \mathcal{S} = \{(p_1, p_2) : p_1 \in \mathcal{H}, p_2 \in \mathcal{S}\}$$

Para ello, el algoritmo irá **seleccionando**, secuencialmente, **un par del conjunto hasta** haber combinado **todos** los pares.

Así, **para combinar el par** $(p_1, p_2) \in \mathcal{H} \times \mathcal{S}$ y formar la solución compuesta p, el método comienza copiando la solución p_1 sobre p. Tras ello, **irá seleccionando**, cada vez, **un trabajo** (de forma aleatoria y sin repetir) del problema, **hasta** agotar los **n** que hay.

Para cada trabajo seleccionado, el método **genera un número r**, **uniformemente distribuido entre cero y uno** ($rand$). Y establece la siguiente casuística:

- **Si** el número generado **es inferior a** un cierto parámetro fijo del método, **t**, **entonces** se ejecuta una comprobación: Se **verifica si** ambas soluciones, **p** y **p_2 asignan, a la misma factoría, este trabajo**.

  En caso negativo, el algoritmo deberá **modificar** la solución **p** y **asignarle** al **trabajo** la **fábrica que marque p_2**. La posición (en la secuencia de producción de tal fábrica) en la que se debe insertar el proceso, es aquella en la que la fábrica complete antes sus tareas.

- **Si** el número generado **es superior (o igual) a t**, la solución p no se ve alterada.

Tras esta dicotomía, el **algoritmo prosigue** hasta **agotar todos los trabajos** del problema. **Y finaliza** su ejecución **cuando** se hayan **combinado todos los pares**.

El **parámetro t** tiene como función la de controlar la intensidad de la diversificación y, en cierta forma, **poner coto a la variabilidad con la que se generan las soluciones**. Un valor cercano a cero, para t, causará que la soluciones generadas por el método tiendan a parecerse a las del conjunto $\mathcal{H}$ original. En cambio, si es cercano a 1, las soluciones generadas distarán en exceso a las originales.

Más adelante, junto con el resto de parámetros, se discutirá el valor que debe tomar t para que el método resulte efectivo.

Conviene **ejemplificar el método de combinación de soluciones** con el siguiente caso. Supongamos que se desean combinar estas dos soluciones, fijando el parámetro $t = 0.1$:

$$p_1 = \begin{cases} 3, 1, 2, 5 \\ 4, 9, 6 \\ 7, 8 \end{cases} \qquad p_2 = (3, 1, 2, 1, 2, 3, 2, 3, 2)$$

Por tanto:

1. El proceso comienza asignando la solución combinada como p_1. Es decir, $p = p_1$.
2. Primera iteración:
   - El método selecciona, aleatoriamente y sin repetir, un trabajo (entre los 9 disponibles). Imaginemos que elige el 4.
   - A continuación genera un número aleatorio entre 0 y 1. Supongamos que $r = 0.08$.
   - Como $r < t$, se deberá realizar la comprobación explicada:
     - En la solución p, la fábrica *segunda* debe procesar el trabajo 4.
     - En la solución p_2, la fábrica *primera* procesa el trabajo 4.
   - Al encontrar discrepancias entre ambas soluciones, el trabajo 4 se asigna a la fábrica que marca p_2. Es decir, a la primera fábrica.
   - Allí, se inserta en todas las posibles posiciones de la secuencia de producción. Estas son las posibles cinco secuencias resultantes:

$$\begin{cases} 4,3,1,2,5 \\ 9,6 \\ 7,8 \end{cases} \quad \begin{cases} 3,4,1,2,5 \\ 9,6 \\ 7,8 \end{cases} \quad \begin{cases} 3,1,4,2,5 \\ 9,6 \\ 7,8 \end{cases} \quad \begin{cases} 3,1,2,45 \\ 9,6 \\ 7,8 \end{cases} \quad \begin{cases} 3,1,2,5,4 \\ 9,6 \\ 7,8 \end{cases}$$

   - Finalmente se asigna el trabajo 4 en la posición de la secuencia en la que la primera fábrica experimente un menor $C_{\text{máx}}$. Pongamos por caso que se toma

$$p = \begin{cases} 4,3,1,2,5 \\ 9,6 \\ 7,8 \end{cases}$$

3. Segunda iteración:
   - Se toma, aleatoriamente y sin repetir, uno de los trabajos del problema (entre los 8 disponibles). Supongamos que es el 2.
   - Al generar el número aleatorio entre cero y uno se obtiene que $r = 0.41$. Y, por tanto, la solución p no se ve alterada.
4. Tercera iteración:
   - Se toma, aleatoriamente y sin repetir, uno de los trabajos del problema (entre los 7 disponibles). Supongamos que es el 9.
   - Al generar el número aleatorio entre cero y uno se obtiene que $r = 0.003$.
   - Como $r < t$, se debe ejecutar la pertinente comprobación:
     - En la solución p, la fábrica *segunda* debe procesar el trabajo 9.
     - En la solución p_2, la fábrica *segunda* procesa el trabajo 9.
   - Como coinciden, la solución p no se modifica.

Y se prosigue, así, hasta haber iterado con todos los trabajos del problema.

A continuación se muestra, en forma de pseudocódigo, el método de combinación de soluciones:

Algoritmo 2.4 Solution Combination Method

Entrada: El valor del parámetro t y el par $(p_1, p_2) \in \mathcal{H} \times \mathcal{S}$
Salida: Una solución, p, que es combinación de (p_1, p_2)

1: $p \leftarrow p_1$
2: **para** iteración:=1 **hasta** n **hacer**
3: Se selecciona un trabajo, j, de forma aleatoria y sin repetir, entre los n del problema
4: $r \leftarrow rand(0, 1)$
5: **si** $r < t$ **entonces**
6: **si** (factoría en la que se procesa j en la solución $p \neq$ factoría de j en p_2) **entonces**
7: En la solución p se asigna el trabajo j a la factoría que marque la solución p_2
8: El trabajo j se inserta en todas las posibles posiciones de la secuencia de producción de la factoría (a la que acaba de ser asignado)
9: El trabajo j se inserta, finalmente, en la posición en la que fábrica experimente un menor $C_{\text{máx}}$.
10: **fin si**
11: **fin si**
12: **fin para**

Improvement Method

Para mejorar las soluciones de las que se van disponiendo tras el **proceso de combinación** de soluciones (*Solution Combination Method*), se empleará el **método VNDa**.

Este algoritmo de búsqueda se explicó en este documento, en la subsección 2.5.2 y se emplaza al lector para que lo revise de nuevo.

Reference Set Update Method

El conjunto de referencia necesita ser actualizado en cada iteración del método. Tanto porque $\mathcal{S}$ debe generarse aleatoriamente tras cada ciclo del método. Como porque, por definición del conjunto $\mathcal{H}$, éste debe contener las b mejores soluciones encontradas hasta ese momento.

Surge una cuestión realmente importante con la definición de $\mathcal{H}$. El **cómo establecer la forma de comparar dos soluciones, para saber que b soluciones deben formar parte del éste conjunto**.

Los autores de este método afirman haber elaborado y probado estrategias[xxxiv]más elaboradas para responder esta pregunta. Pero obtuvieron idénticos resultados al **aplicar la siguiente regla simple**:

Una solución p reemplazará a otra solución, q, del conjunto $\mathcal{H}$ si, y solo sí:

1. La solución p no está incluida ya en el conjunto $\mathcal{H}$.
2. La solución q es la que genera el mayor $C_{\text{máx}}$ del problema, entre los tiempos de completación de la producción que ofrecen las restantes soluciones del conjunto $\mathcal{H}$.
3. Y el $C_{\text{máx}}$ del problema que ofrece la solución p es menor al que se genera con q.

Ahora bien, que $\mathcal{H}$ disponga siempre de las b mejores soluciones no parece ser muy apropiado si la búsqueda dispersa se caracteriza por su capacidad de generar, recombinar y diversificar las soluciones de las que va disponiendo en el conjunto de referencia.

Para **asegurar la variabilidad** y la diversidad **entre la soluciones** que lleguen a formar parte **del conjunto $\mathcal{H}$**, los autores del método consideraron apropiado que, **después de *a* iteraciones del algoritmo sin mejorar el conjunto $\mathcal{H}$, éste debía de ser regenerado**.

Por ello dispusieron que, tras la a-ésima iteración del método sin modificar $\mathcal{H}$, el 50 % del peor contenido de este conjunto debía ser reemplazado. **Las b/2 peores soluciones** del conjunto **serían sustituidas por las b/2 mejores soluciones obtenidas** tras ejecutar, de nuevo, el **generador de soluciones variadas** (*Diversification Generation Method*).

Como medida garantista establecieron que **este descarte debía ocurrir** tras cada iteración, **hasta** que se cumpliesen las condiciones necesarias para que ***Reference Set Update Method*** **actualizase $\mathcal{H}$**.

[xxxiv] A los autores les preocupaba poder asegurar variabilidad en el conjunto de referencia. Creyeron que podía resultar insuficiente la que otorgaba el conjunto $\mathcal{S}$. Y decidieron probar con añadir condiciones extra de selección para garantizar que, en caso que una solución sustituyera a otra en $\mathcal{H}$, la nueva mantuviese o aportase mayor diversidad. No aclaran que condiciones probaron, simplemente explican que resultaron demasiado costosas computacionalmente. Y comentan, eso sí, que éstas condiciones no mejoraron experimentalmente los resultados que se obtenían con el método de búsqueda dispersa, en comparación con la regla simple que terminan proponiendo.

Método de búsqueda dispersa propuesto

Algoritmo 2.5 Método de búsqueda dispersa para resolver $DF|prmu|C_{\text{máx}}$

Entrada: Los parámetros b, l, a y t. Los datos relativos al problema a resolver. Un criterio de parada de búsqueda.
Salida: Una solución para resolver el problema en cuestión.

1: $contador \leftarrow 0$
2: Generación inicial del conjunto $\mathcal{H}$ (*Diversification Generation Method*)
3: **mientras** no se den las condiciones de finalización **hacer**
4: Regenerar el conjunto $\mathcal{S}$ (*Diversification Generation Method*)
5: **para** i:=1 **hasta** b **hacer**
6: **para** j:=1 **hasta** l **hacer**
7: $p_1 \leftarrow$ solución i-ésima del conjunto $\mathcal{H}$
8: $p_2 \leftarrow$ solución j-ésima del conjunto $\mathcal{S}$
9: $p \leftarrow$ salida del algoritmo 2.4 (*Solution Combination Method*) ejecutado sobre (p_1, p_2, t)
10: $p \leftarrow$ salida del algoritmo VND (*Improvement Method*) ejecutado sobre p
11: Aplicación del método de actualización, con la solución p, para $\mathcal{H}$ (*Reference Set Update Method*)
12: **fin para**
13: **fin para**
14: **si** el conjunto $\mathcal{H}$ ha sido actualizado **entonces**
15: $contador \leftarrow 0$
16: **si no**
17: $contador++$
18: **fin si**
19: **si** $contador > a$ **entonces**
20: Aplicar el procedimiento de reinicio del conjunto $\mathcal{H}$ (*Reference Set Update Method*)
21: **fin si**
22: **fin mientras**

Calibración de los parámetros del método

Para hacer totalmente operativo el método, se deben determinar **los valores que toman los parámetros del método**. A saber:

- El tamaño del conjunto $\mathcal{H}$, **b**
- El número de entradas del conjunto $\mathcal{S}$, **l**
- El número de iteraciones que se permiten sin alterar el conjunto $\mathcal{H}$, **a**
- La intensidad con la que se diversifican las soluciones en *Solution Combination Method*, **t**.

El valor t resultó **prefijado de antemano a 0.1** pues, según refieren los autores, resultaba suficiente para garantizar la variabilidad en la etapa *Solution Combination Method*. No proveen un pertinente estudio estadístico para reflejar la idoneidad del parámetro y se basan en pruebas experimentales que realizaron durante el desarrollo del método.

En cambio, para el **resto de parámetros**, los autores del método sí que decidieron realizar un **estudio** estadístico (**de factores cruzados**) para responder a la cuestión.

Los **valores de los parámetros** estudiados fueron:

- $b \in \{2, 5, 10, 15\}$
- $l \in \{2, 5, 10\}$
- $a \in \{10, 20, 30, 40\}$

Es decir, se analizaron 48 métodos de búsqueda dispersa diferentes.

La **variable respuesta** del modelo estadístico planteado para responder la cuestión fue el **RPD**:

$$RPD = \frac{\text{Solución obtenida por el método} - \text{Mejor solución conocida para la instancia}}{\text{Mejor solución conocida para la instancia}} \cdot 100$$

Es decir, la desviación, en porcentaje relativo, de la solución obtenida (por el método con los respectivos parámetros) entre la mejor solución conocida para la instancia en la que se ejecuta. Nótese que, a **menor RPD**, menor desviación entre la solución obtenida y la mejor conocida. Y, por tanto, **el método funciona mejor**.

Para cada método de búsqueda dispersa considerado (**cruzamiento**) se seleccionaron **50 instancias aleatorias** obtenidas de un conjunto de 720 instancias para el problema DPFSP, disponibles en *soa.iti.es*. Por tanto, la **muestra analizada constaba de**

$$50 \text{ instancias} \cdot 48 \text{ métodos diferentes} \cdot 5 \text{ repeticiones } = \mathbf{12000 \text{ valores RPD}}.$$

Los **métodos de búsqueda dispersa** fueron **implementados en C++**. La computación tuvo lugar un servidor blade de 30 tarjetas, con una potencia de cómputo total de 240 cores (a través de 60 Intel Xeon E540) y 480GB de memoria RAM. Se virtualizó la computación a través de máquinas virtuales corriendo un Windows XP con un núcleo y 2 GB de RAM.

El **criterio de parada** para la **ejecución del método** de búsqueda dispersa fue el tiempo de computación. Pero, para evitar sesgos entre las instancias más pequeñas y las más grandes, el **tiempo (en milisegundos) permitido de ejecución**, **por instancia del problema** (con n trabajos, m máquinas y F fábricas), fue determinado a través de la fórmula

$$\mathbf{10 \cdot n \cdot m \cdot F}$$

Tras una análisis de las varianzas (**ANOVA**), en el que se verificaron rigurosamente las **hipótesis de aplicabilidad** (independencia, normalidad y homocedasticidad), se llegaron a las siguientes **conclusiones**:

- Los **parámetros b, l** y **a** son **significativos estadísticamente**, con un nivel de significancia de $\alpha = 0.05$, en presencia del resto, **para explicar la variabilidad observada**.
- Hay suficiente evidencia estadística para afirmar que **existe diferencia**, para la variable respuesta, **entre los diferentes niveles de cada parámetro estudiado** (b, l y a).
- El parámetro **b** es el que tiene **mayor influencia** sobre la variable respuesta. El **segundo** más influyente es **l**. Y el **menos influyente**, **a**.

Tras aceptar que al menos uno de los niveles, de cada parámetro estudiado, tiene la media diferente, ANOVA no aclara qué nivel es, ni tan solo si es sólo uno. Una posibilidad para mejorar las conclusiones es utilizar comparaciones dos a dos de las medias de los grupos teniendo presente la protección del error global.

Por ello, los autores emplean el **test de Tukey, como método de comparación a posteriori, para estos parámetros**. El test resulta aplicable porque se cumplen las **condiciones de aplicabilidad** de éste, que son las mismas que para ANOVA. A través de él se concluye que:

- Para el **parámetro b**
  - **Los niveles 10 y 15 son equivalentes estadísticamente**, con un nivel de significancia de $\alpha = 0.05$. Y conforman un grupo.
  - **El nivel 2 es estadísticamente diferente al resto de niveles** de este parámetro, con un nivel de significancia de $\alpha = 0.05$. Y conforma un grupo compuesto por él solo.
  - **El nivel 5 es estadísticamente diferente al resto de niveles** de ese parámetro, con un nivel de significancia de $\alpha = 0.05$. Y conforma un grupo compuesto por él solo.
- Para el **parámetro l**
  - **Los niveles 5 y 10 son equivalentes estadísticamente**, con un nivel de significancia de $\alpha = 0.05$. Y conforman un grupo.
  - **El nivel 2 es estadísticamente diferente al resto de niveles** de este parámetro, con un nivel de significancia de $\alpha = 0.05$. Y conforma un grupo compuesto por él solo.
- Para el **parámetro a**
  - **Los niveles 20, 30 y 40 son equivalentes estadísticamente**, con un nivel de significancia de $\alpha = 0.05$. Y conforman un grupo.
  - **El nivel 10 es estadísticamente diferente al resto de niveles** de este parámetro, con un nivel de significancia de $\alpha = 0.05$. Y conforma un grupo compuesto por él solo.

Entre los diferentes grupos de medias para cada parámetro, se seleccionó aquel grupo (y parámetro) **que presentaba una media para el parámetro RPD más baja**.

Por ello, los autores seleccionaron estos valores para el método de búsqueda dispersa propuesto:

$$\mathbf{b = 10}, \qquad \mathbf{l = 10}, \qquad \mathbf{a = 40}$$

2.5.4. Rendimiento de los métodos heurísticos analizados

Los métodos heurísticos y metaheurísticos son capaces de proporcionar soluciones aceptables a problemas de optimización complejos en tiempos muy razonables. Ahora bien, estas soluciones, aunque posibles, no suelen ser óptimas habitualmente.

La calidad de la solución ofrecida depende del diseño del heurístico que se esté empleando. Por tanto, resulta imprescindible valorar, en cuestión de calidad, los métodos heurísticos y metaheurísticos analizados en este documento para resolver el problema $DF|prmu|C_{\text{máx}}$.

Normalmente, la **medida** típica para **valorar la calidad de las soluciones proporcionadas por los diferentes heurísticos**, para una instancia del problema concreta, es el **RPD**.

Como se comentaba en la subsección anterior, 2.5.3, mientras se calibraba el método de búsqueda dispersa propuesto, el **RPD es una medida de desviación**, en porcentaje relativo, de la solución obtenida por el método heurístico correspondiente, para una determinada instancia, en comparación con la mejor solución conocida para esa instancia.

Realizar un estudio propio, comparativo y formal sobre esta cuestión **está totalmente fuera lugar**, al constituir este documento un trabajo final del grado en matemáticas. Por ello, **este trabajo se hará eco** de los resultados obtenidos en **dos de los estudios publicados sobre los métodos analizados**. En concreto,

- De [34], que valora la eficacia de la primera aproximación estudiada en 2.5.1 con la de los métodos VND analizados en 2.5.2.
- Y de [35], que estudia la eficacia del método de búsqueda dispersa estudiado en 2.5.3 con la de los métodos VND analizados en 2.5.2.

Como **ambos estudios usan como nexo común los métodos VND estudiados, será posible establecer una comparativa entre todos los métodos heurísticos y metaheurísticos analizados**.

Por un lado, **el estudio de [34] señala que**:

- Entre los **métodos** que constituyen la **primera aproximación** al desarrollo de heurísticos, **aquellos que usan la regla NR 2** y su variación (la regla NEH 2), **ofrecen mejores resultados que aquellos que usan la regla NR 1** y su variación (la regla NEH 1). La diferencia es estadísticamente significativa.
- No existe una diferencia estadísticamente significativa en el coste computacional de ambas reglas (NR 1 y NR 2). Y tampoco entre y con sus variaciones (regla NEH 1 y NEH 2).
- **Los métodos VND** (tanto VNDa. como VNDb.) **son realmente rápidos** (por debajo de los 4 segundos en el peor de los casos), **aunque bastante más lentos que los otros métodos**, que están por debajo de los 0.15 segundos.
- **Entre los métodos VNDa. y VNDb. no existe diferencia estadística significativa en cuestión de eficacia y rendimiento**. Ambos ofrecen tiempos de computación y valores de RPD estadísticamente equivalentes.

- **Los métodos VNDa, VNDb y NEH2 son los métodos más efectivos**, entre los analizados en este estudio. **La diferencia** con el resto de métodos **es estadísticamente significativa**. Entre estos tres métodos, no existe diferencia significativa en cuestión de efectividad.

Aunque la diferencia entre el valor RPD de los métodos VNDa, VNDb y NEH2 no es significativa, [34] concluye que el **método VNDa ofrece un mejor rendimiento** puesto que, empíricamente, el valor medio de RPD resulta inferior al que ofrecen los otros dos métodos.

Por otro lado, **el estudio de [35]**:

- **Reafirma lo dicho sobre los métodos VN**D, que **son rápidos** y que, entre ellos, no existe diferencia estadística significativa en cuestión de eficacia y rendimiento.
- **Afirma que el método de búsqueda dispersa estudiado es estadísticamente más efectivo en comparación con los métodos VND analizados**, cuando se ejecutan bajo idénticas condiciones. La búsqueda dispersa ofrece un menor RPD para las instancias estudiadas y esa diferencia es significativa.

Y, por tanto, **de estos análisis se puede concluir que el método heurístico que mejor funciona, entre los estudiados, es el algoritmo de búsqueda dispersa.**

2.6. Otros métodos heurísticos y metaheurísticos para $DF|prmu|C_{\text{máx}}$

La **última década** ha resultado ser **muy fructífera**, en cuanto a la cantidad de avances e investigaciones llevadas a cabo con el fin de desarrollar **métodos** (heurísticos y metaheurísticos) para resolver la planificación de la producción en entornos distribuidos, para factorías con producción FlowShop con permutación (**DPFSP**).

Por el carácter de este documento, resultaría imposible incluir un análisis detallado de cada uno de estos avances. Ahora bien, el autor considera imprescindible elaborar una **breve síntesis** de éstos e incluirla, por completitud, en este trabajo final de grado. Un ejercicio, en todo caso, dirigido a construir una sobria y concisa revisión bibliográfica sobre estas investigaciones y **poner, en contexto, el problema DAPFSP**, estudiado en el siguiente capítulo.

Por ello, se reseña de la siguiente forma:

Un metaheurístico basado en el electromagnetismo

En [29] se puede encontrar un metaheurístico que toma como base las relaciones físicas existentes entre el magnetismo y la electricidad. El **electromagnetismo**, que **unifica los fenómenos eléctricos y magnéticos** en una sola teoría **a través de** las conocidas **ecuaciones de Maxwell**, sirve como **inspiración para generar este método basado en la búsqueda de vecindad variable (VNS)**.

Para ello, y a través de las ecuaciones de Maxwell, se usan diferentes búsquedas locales para

- insertar y extraer trabajos de la fábrica crítica (la que tiene mayor $C_{\text{máx}}$ en el problema),
- reasignar trabajos entre las diferentes fábricas,
- intercambiar la condición de fábrica crítica entre las diferentes factorías, para mejorar el $C_{\text{máx}}$ del problema, y
- reconfigurar los trabajos en el interior de cada fábrica.

Hacer constar que **los autores proceden inadecuadamente al no realizar un riguroso estudio comparativo entre** los diferentes **métodos heuríticos y metaheurísticos, disponibles en su momento** para resolver el problema $DF|prmu|C_{\text{máx}}$. En su artículo simplemente aluden a que, sobre las instancias disponibles en *soa.iti.es*, **su metaheurístico mejora 151 de las mejores soluciones conocidas hasta ese momento**. **No tienen en cuenta factores** tan importantes como el tiempo de CPU empleado, **si esas mejoras resultan significativas estadísticamente** o **qué sucede** con el **resto de las instancias** estudiadas.

Indicar que **sí que existe un pertinente estudio estadístico**, realizado por otros autores en [35], donde se manifiesta que, aunque sí que existe esa mejora en 151 de las instancias usadas como base comparativa, el **rendimiento medio del algoritmo resulta pobre y**, en todo caso, **a expensas de un alto coste computacional**.

Un algoritmo genético híbrido

En [9] se presenta un **algoritmo genético que incluye una fase de búsqueda local para mejorar su eficacia**. Su funcionamiento resulta sencillo conceptualmente, al estar inspirado en [39], un algoritmo genético bien conocido para resolver el problema PFSP regular.

Consta de las tradicionales fases de

- **Cruzamiento** y **mutación** de los genes disponibles,
- **Evaluación** de los resultados obtenidos por los genes,
- **Reproducción** genética (ponderada) de los genes, y
- **Selección de genes** supervivientes

E **intercala una nueva fase de búsqueda de vecindad variable**. Esta búsqueda es **prácticamente idéntica a la que se realiza en el algoritmo 2.3** para construir el metaheurístico VNDa, salvo porque **incluye una tercera búsqueda local para reasignar trabajos de la fábrica crítica hacia otras fábricas del problema**.

Los autores proponen que el algoritmo genético emplee, como **inicialización**, los resultados obtenidos por los **heurísticos NEH2** (presentado en la subsección 2.5.1) **y VNDa** (presentado en la subsección 2.5.2).

El estudio estadístico que desarrollan los autores del algoritmo pone de manifiesto que resulta más efectivo que los métodos heurísticos y metaheurísticos desarrollados hasta ese momento (en el año 2011). Pero a costa de un **alto coste computacional**. Usando el criterio de parada propuesto por los autores (*detener tras 30 iteraciones del algoritmo*) que, en teoría, debería ofrecer los mejores resultados, se requieren 246 veces más de tiempo para obtener los resultados finales, en comparación con el tiempo empleado por el método VNDa.

El **estudio estadístico** que realiza [35], en el año 2014, a fin de comparar los diferentes heurísticos y metaheurísticos disponibles para resolver el problema $DF|prmu|C_{\text{máx}}$, **pone de manifiesto que, bajo el mismo tiempo de computación**, el último método desarrollado en este documento (en la subsección 2.5.3, **método de búsqueda dispersa, SS**) **es**, estadísticamente, **más eficaz y obtiene mejores resultados que el algoritmo genético del que se está hablando**.

Un método de búsqueda tabú

La **búsqueda tabú** es un algoritmo metaheurístico que utiliza un procedimiento de **búsqueda local** para moverse por el espacio de soluciones. A diferencia de otros métodos basados en búsquedas locales, la búsqueda tabú **usa estructuras de memorias para poder explorar una enorme cantidad de regiones del espacio de búsqueda** que, de otra forma, permanecerían inexploradas (por el efecto de óptimos locales). Para ello, a medida que avanza la búsqueda, ésta **va modificando la estructura de vecinos para cada solución hallada**. Las estructuras de memoria del método (como las listas tabú) determinarán la nueva estructura de vecinos.

En [11] se desarrolla un método de búsqueda tabú para resolver el problema que nos ocupa, $DF|prmu|C_{\text{máx}}$. El proceso de **búsqueda local** que desarrolla resulta ser **casi idéntico** al que se presenta en el **algoritmo genético híbrido comentado anteriormente** (disponible en [9]). Aunque, eso sí, la **búsqueda resulta más profunda** porque, además del efecto memoria del método, el tiempo en que se permanece dentro de ella es mayor.

Como **inicialización** para el método de búsqueda se vuelve a emplear la solución obtenida usando el **algoritmo NEH2**. Y como estructura de memoria se emplea una **lista tabú, formada por pares de trabajos ordenados**. **Cada par representa una secuencia, codificada, conformada por el método.**

Aunque quizás no sea la forma más ortodoxa de explicar la codificación de movimientos que emplea el método, sí que es la más práctica y útil para entenderla:

- Supongamos que se dispone, en alguna iteración del método, de la siguiente planificación de la producción:
$$\begin{cases} 2,3,4,5 \\ 1,6,7 \end{cases}$$
- Un cálculo del tiempo de producción en cada fábrica para esta solución determina que la fábrica con mayor $C_{\text{máx}}$ (fábrica crítica) es la primera.
- Algorítmicamente, el método tabú ha determinado que el mejor movimiento válido [xxxv] es intercambiar la secuencia $\{2,3\}$ de la primera fábrica, por la secuencia $\{1,6\}$ de la segunda. Y, por tanto, la solución que propone en esa iteración es
$$\begin{cases} 1,6,4,5 \\ 2,3,7 \end{cases}$$
- Una vez aceptado y realizado el movimiento, se añadirá a la lista tabú, si no estaba en ella, el par $(3,4)$. Es decir, el trabajo 3 (último trabajo de la secuencia intercambiada, que se ejecutaba antes en la fábrica crítica) y el trabajo 4 (el trabajo que seguía, en la anterior solución y en la susodicha fábrica crítica, a 3).
- A partir de ese momento, si alguna búsqueda local propusiera que el trabajo 3 precediera a la ejecución del trabajo 4 en alguna fábrica, éstos no se podrían separar en posteriores iteraciones del método.

[xxxv] Para que un movimiento sea válido, no debe contradecir a las secuencias que aparecen en la lista tabú. El algoritmo lo considera el mejor movimiento porque el resto de movimientos válidos producirán soluciones con mayor $C_{\text{máx}}$ total.

El estudio de rendimiento del método, que realizan sus propios autores, señala un excesivo coste computacional aunque, eso sí, una considerable eficacia, estadísticamente significativa, con respecto a los métodos existentes en aquel momento.

Al igual que sucedió con el algoritmo genético híbrido, el **estudio estadístico** que realiza [35], en el año 2014, a fin de comparar los diferentes heurísticos y metaheurísticos disponibles para resolver el problema $DF|prmu|C_{\text{máx}}$, **vuelve a señalar que, bajo el mismo tiempo de computación**, **este método tabú es**, estadísticamente, **menos eficaz y obtiene peores resultados que** el último método desarrollado en este documento (en la subsección 2.5.3, **método de búsqueda dispersa, SS**).

Un algoritmo heurístico voraz

Un algoritmo voraz (*greedy method*) es un método heurístico que se enmarca dentro de los algoritmos de búsqueda local. El método va siguiendo un **proceso secuencial** donde, en cada paso, va **tomando**, de forma **sistemática**, **una decisión sobre la forma de resolver el problema**. Las decisiones que va tomando siempre buscan elegir la opción óptima en cada paso de la búsqueda local, con la esperanza de llegar a una solución general óptima.

La característica principal de estos algoritmos es que **no vuelven nunca a reconsiderar alguna de las decisiones tomadas**. Una vez han decidido asignar, a una variable, un valor localmente óptimo que hace que la secuencia de decisiones sea factible, no vuelve a reconsiderar asignarle un nuevo valor a ésta.

En [27] se pueden encontrar **cuatro variantes de un algoritmo voraz para resolver el problema** $DF|prmu|C_{\text{máx}}$, aunque **el que resulta presentar mejor rendimiento es** el que llaman $\mathbf{IG_{VST}}$. Las diferentes variantes **se diferencian en el valor que se asigna a los parámetros que afectan a la toma de decisiones del algoritmo**.

Todas estas variantes siguen la estructura típica de un algoritmo voraz y, por tanto, contienen:

- Un conjunto finito de candidatos, de posibles soluciones (totales o parciales) del problema
- Una función de selección, para valorar a los mejores candidatos
- Una función de factibilidad, para determinar si los candidatos constituyen soluciones al problema
- Una función solución, que evalúa al conjunto de candidatos seleccionados para resolver el problema

La evaluación del algoritmo que realizan los autores, durante el primer cuatrimestre del año 2013, lo señala como el más efectivo al medirse con los algoritmos disponibles en aquel año. Pero el **estudio estadístico** que realiza [35], en el año 2014, a fin de comparar los diferentes heurísticos y metaheurísticos disponibles para resolver el problema $DF|prmu|C_{\text{máx}}$, **pone de manifiesto que, bajo el mismo tiempo de computación**, el último método desarrollado en este documento (en la subsección 2.5.3, **método de búsqueda dispersa, SS**) **es**, estadísticamente, **más eficaz y obtiene mejores resultados que el algoritmo voraz del que se está hablando**.

Un algoritmo de estimación de distribución (*EDA*)

Los algoritmos de Estimación de Distribución (*EDA*) son métodos metaheurísticos y estocásticos, derivados de la familia de los algoritmos evolutivos.

Mientras que los **algoritmos evolutivos convencionales** tratan de encontrar una solución al problema de optimización generando soluciones candidatas, mediante el **uso de operadores de cruzamiento y mutación** (distribución implícita definida por estos operadores), los **algoritmos EDA usan**, para ello, una **distribución de probabilidad explícita**, **codificada** por una **red bayesiana**, una **distribución normal multivariante** u otros modelos.

Por convenio, los algoritmos EDA conservan el argot usado en los algoritmos evolutivos convencionales. Se entiende

- que un **individuo** es una **solución candidata**, y
- que una **población** es un **conjunto de individuos**.

Pero, debido a que, en éstos, la población no se genera a partir de individuos, sino a partir de distribuciones de probabilidad, **no tiene sentido hablar de operadores de cruzamiento ni de mutación**.

En [49] se desarrolla un método EDA para resolver el problema $DF|prmu|C_{\text{máx}}$. Al igual que sucedía con los anteriores métodos, un **estudio estadístico independiente** ([35]) **lo señala como altamente ineficaz, con un rendimiento pobre y con alto coste computacional difícilmente justificable**.

Una comparación indirecta y en condiciones de computación distintas (a favor del método EDA) en [35] señala que el último método desarrollado en este documento (en la subsección 2.5.3, **método de búsqueda dispersa, SS**) ofrece mejores resultados y mejores soluciones que el método EDA comentado.

Capítulo 3

The Distributed Assembly Permutation Flowshop Scheduling Problem

Como se comentaba en el capítulo primero de este documento, hoy en día **los sistemas de producción se circunscriben dentro de un entorno global, complejo y en constante variación**, donde resulta realmente complicado tener éxito cuando los problemas de planificación de la producción se abordan de forma clásica.

Se planteaban, entonces, **dos nuevos sistemas de control de la producción** para hacer frente a los retos del nuevo entorno.

Por un lado, existía la posibilidad de emplear una **red de fabricación distribuida** como respuesta al desafío. Y que **fue estudiada en el capítulo segundo**, cuando

- El objetivo que se persigue es minimizar el tiempo de completación de las tareas, en total.
- Se disponen de factorías idénticas, con capacidad de producción ilimitada, que fabrican bajo una configuración FlowShop con permutación.
- Y no existe posibilidad de transferencia de trabajos entre las diferentes fábricas.

Es decir, cuando el esquema de producción está descrito por $DF|prmu|C_{\text{máx}}$.

Por otro lado, se podía contemplar la opción de establecer una **red de fabricación totalmente distribuida**, en la que los componentes de los productos se fabriquen en diferentes localizaciones y se ensamblen al final, pudiendo llegar a permitir la personalización del producto.

Este planteamiento se estudiará en el presente capítulo, a través del problema de programación de la producción en entornos totalmente distribuidos, para factorías con producción FlowShop con permutación y para una línea de montaje global, que dispone de una única estación de trabajo con una sola máquina (***Distributed Assembly Permutation FlowShop Scheduling Problem*, DAPFSP**).

Para ello, se **formulará formalmente el problema DAPFSP**. Y se **estudiarán modelos y métodos heurísticos para resolverlo** cuando se desee minimizar el *makespan* máximo del sistema de producción global. Además, se presentará una **implementación** y sendos **estudios** para evaluar el **rendimiento** del modelo y de los métodos heurísticos propuestos.

3.1. Formulación del problema

Formalmente, el problema de programación de la producción en entornos totalmente distribuidos,

- para factorías con producción FlowShop con permutación, y
- para una línea de montaje global, común para todas las fábricas, que dispone de una única estación de trabajo con una sola máquina,

conocido como ***Distributed Assembly Permutation FlowShop Scheduling Problem*** (**DAPFSP**), consiste en la **combinación de dos problemas** asociados al *scheduling*:

- Un problema de **planificación de la producción** de tipo **DPFSP**, y
- Un problema de **ensamblaje final de tipo AFSP** (*Assembly FlowShop Scheduling Problem*) sobre una línea de montaje **Mixed Model Assembly for Make-to-Order**.

Es, por ello, que el problema se estructura sobre **dos etapas** bien diferenciadas:

- Una primera etapa de **producción** (de los componentes que conforman el producto final), y
- Una segunda etapa de **ensamblaje** (para montarlos y constituir el producto final).

Primera etapa

La **etapa de producción** consistirá en la **programación de la ejecución de n componentes**[1] (recogidos en el conjunto $\mathcal{J}$), **en algunas** (o en todas) las **fábricas** de las que se dispone. En total se cuenta con F fábricas, enumeradas en el conjunto $\mathcal{F}$.

Al ejecutarse la producción sobre un entorno DPFSP, ésta estará sujeta a ciertas características y particularidades, que se enumeraron y detallaron en la sección 2.1. Se recomienda al lector, por tanto, que consulte dicha sección en el segundo capítulo.

Se recuerda que, al tratarse de una red de fabricación distribuida, esta etapa constará de dos decisiones fundamentales:

- Establecer el **reparto**, entre las diferentes **factorías**, de los n trabajos programados, y
- Programar la **ejecución** de los **trabajos** asignados en **cada fábrica**.

[1] En el argot, estos componentes se tratan de los trabajos a programar en el problema DPFSP.

Segunda etapa

Con respecto a la **etapa de ensamblaje**, **una única máquina**, M_A, se encargará de **construir t productos finales**[11] (recogidos en el conjunto $\mathcal{T}$) **a partir de los componentes fabricados** en la etapa de producción.

El ensamblaje aparece **caracterizado** y restringidos a las siguientes particularidades y **propiedades**:

- **Cada producto** tendrá definido **una rutina de ensamble, particular y específica para él** (sobre la que, llegado el caso, el cliente podrá incluir sus personalizaciones).
- Cada rutina está **formada** por un determinado número de fases, **subrutinas** u operaciones a realizar para construir el producto final.
- **En cada subrutina se incluirá un componente de los fabricados** en la etapa de producción. Y, por tanto, resulta evidente que el **ensamblaje solo** podrá tener lugar **cuando todos los componentes necesarios** hayan sido fabricados y **estén disponibles** en la línea de montaje.
- La rutina de ensamble para un cierto producto, h, se denotará por el conjunto ordenado N_h. Y estará formado por las $|N_h|$ subrutinas necesarias para construirlo.
- El conjunto $\{J_j\}_{j\in N_h}$ estará formado, en cambio, por los componentes necesarios (producidos en la primera etapa) para construir el producto final h.

Por tanto y puesto que cada componente que se incluye y se utiliza en cada subrutina no se puede emplear en otras posteriores, se cumple que $\sum_{h=1}^{t} |N_h| = n$.

El **objetivo** que se perseguirá con la programación de la producción y ensamblaje del problema DAPFSP, en este capítulo, será el de **minimizar el tiempo de completación de todas las operaciones de ensamblaje** sobre la línea de montaje (*makespan at the assembly factory*). Sin riesgo de ambigüedad, se denotará a este tiempo como $C_{\text{máx}}$.

Por tanto, el problema estudiado, denotándolo adecuadamente, será

$$DAF|prmu|C_{\text{máx}}$$

[11] En el argot, la construcción de estos productos finales constituyen los trabajos a programar en el problema AFSP.

3.2. Un modelo matemático para el problema $DAF|prmu|C_{\text{máx}}$

Los modelos matemáticos son una buena herramienta para describir apropiadamente los problemas de *Scheduling*. Y es que, a partir de ellos, se puede realizar un análisis profundo de la situación real en la que se enmarcan estos problemas y descubrir estrategias resolutivas que no eran obvias al principio. De ahí que resulte incuestionable que, desde un punto de vista intrínseco y conceptual, los modelos matemáticos sean sumamente interesantes y útiles para comprender el problema que modelizan.

En esta sección se incluye un modelo para resolver el problema $DAF|prmu|C_{\text{máx}}$, presentado en [17] y enmarcado dentro de la programación lineal entera mixta (*Mixed-Integer Lineal Problems*, **MILP**).

Pero, primeramente, se comentarán los parámetros, índices y notación que se emplearán en éste:

Notación	**Descripción**
$\mathcal{J}$	Conjunto de componentes que se deben producir en las factorías. Incluye a un componente ficticio, *dummy job*, denotado por 0.
$\mathcal{M}$	Conjunto de máquinas de las que dispone cada factoría de la etapa de producción.
$\mathcal{F}$	Conjunto de fábricas disponibles en la etapa de producción.
$\mathcal{T}$	Conjunto de productos finales que se deben ensamblar en la línea de montaje. Incluye un producto final ficticio, *dummy job*, denotado por 0.
M	Una constante suficientemente grande (*Big-M*)

Índices	**Descripción**
$j, k \in \mathcal{J}$	Índices para denotar a los componentes que se deben producen en las factorías.
$i \in \mathcal{M}$	Índice que denota a las máquinas de las factorías en la etapa de producción.
$l, s \in \mathcal{T}, s \neq 0$	Índices referido a los productos finales, ensamblados en la línea de montaje.

Parámetros	**Descripción**
n	Número de componentes que se deben producirse en las factorías. $\lvert\mathcal{J}\rvert = n$.
m	Número de máquinas disponibles en cada una de las factorías. $\lvert\mathcal{M}\rvert = m$.
F	Número de fábricas disponibles en la etapa de producción. $\lvert\mathcal{F}\rvert = F$.
t	Número de productos finales a ensamblar en la línea de montaje. $\lvert\mathcal{T}\rvert = t$.
P_{ji}	Tiempo de procesado del componente j en la máquina i en las factorías.
PA_s	Tiempo necesario en la línea de montaje para formar el producto final $s \in \mathcal{T}$.
G_{js}	Parámetro binario. Toma el valor 1 si el componente $j \in \mathcal{J}$ se ensambla dentro del producto final $s \in \mathcal{T}$. Y 0 en otro caso.

3.2.1. Modelo basado en la secuenciación de tareas a partir de la presencia de *dummy jobs*, que actuarán como separadores entre fábricas en la secuenciación [17]

El modelo que se presenta a continuación es el primer intento (y el único realizado hasta el momento) para modelizar el problema DAPFSP. Una modelización cuya estructura interna respeta y diferencia las dos etapas del problema, de producción y ensamblaje.

La **etapa de producción** se modeliza de idéntica forma a como hacía el modelo analizado en 2.3.5 para resolver el problema DPFSP. Este modelo exploraba la idea de **organizar la producción mediante la secuenciación de las tareas**, haciendo uso de trabajos ficticios, ***dummy jobs***, para construir una **única secuencia global** para la producción

- en la que se incluiría la **asignación y secuenciación** de las **tareas en** todas las **fábricas** implicadas, y
- que **se generaría en bloques de tareas** usando ***dummy jobs* como separadores**[III].

Y, por ende, en cada uno de estos bloques aparecerá el orden de fabricación de los componentes en la factoría correspondiente[IV].

Con respecto a la **etapa de ensamblaje**, la modelización está basada también en la **secuenciación de las tareas que debe ejecutar la línea de montaje**. Ahora bien, como solo hay una línea de montaje, no necesita hacer uso de trabajos ficticios como separadores. Solo empleará **un *dummy jobs*** por razones lógicas, **como predecesor al primer producto final a ensamblar**.

Esta división resulta sumamente clara pero, a pesar de todo, se ha decidido **marcar a través de colores los elementos que afectan a cada una de las partes y los que sirven de nexo entre ambas**. Así:

- Las variables y restricciones que modelicen la etapa de producción se marcarán en azul.
- Las que modelicen la etapa de ensamblaje, de verde.
- Y las que sirvan de nexo entre ambas etapas, de naranja.

[III] Así, como se emplearán F trabajos ficticios como separadores, la secuencia tendrá $n + F$ posiciones.

[IV] Realmente, no importará que fábrica tenga asociado cada bloque porque las fábricas son idénticas entre sí. Por simplicidad, se asociará a cada bloque una fábrica por orden cardinal.

Con respecto a las **variables de decisión** empleadas, éstas serán:

- $\mathbf{X_{k,j}} = \begin{cases} 1 & \text{si el componente } j \text{ se produce inmediatamente después del componente } k \\ 0 & \text{otro caso} \end{cases}$

  donde $X_{k,j} \in \{0,1\}$ y aparece definida para los subíndices $k, j \in \mathcal{J} : j \neq k$.

- $\mathbf{Y_{l,s}} = \begin{cases} 1 & \text{si el producto final } s \text{ se produce inmediatamente después del producto final } l \\ 0 & \text{otro caso} \end{cases}$

  donde $Y_{l,s} \in \{0,1\}$ y aparece definida para los subíndices $l, s \in \mathcal{T} : l \neq s, s \neq 0$.

- $\mathbf{C_{j,i}} \in \mathbb{R}^+$, que es el tiempo de completación del componente $j \in \mathcal{J} \setminus \{0\}$ en la máquina $i \in \mathcal{M}$. Es decir, el tiempo que ha estado procesándose el componente j en la máquina i, junto con el tiempo previo acumulado por el componente j en otras máquinas y en esperas.

- $\mathbf{CA_s} \in \mathbb{R}^+$, que es el tiempo de completación del producto final $s \in \mathcal{T} \setminus \{0\}$ en la linea de montaje. Es decir, el tiempo que ha estado ensamblándose el producto final s en la línea de montaje, junto con el tiempo acumulado en espera hasta disponer de todos sus componentes y poder entrar en la línea de montaje.

- $\mathbf{C_{máx}} \in \mathbb{R}^+$, que es el tiempo de completación de todas las operaciones de ensamblaje sobre la línea de montaje.

Y, por tanto, para planificar la producción de n componentes en F fábricas con m máquinas cada una y efectuar su ensamblaje para conformar t productos finales, este modelo requerirá

- $\mathbf{nm + t + 1}$ variables reales (*continuous variable*, **CVs**)
- $\mathbf{(n+1)n + t^2}$ variables binarias (*binary variables*, **BVs**)

Las **restricciones**, limitaciones y condiciones lógicas que se deben verificar en el **problema** de producción y ensamblaje son:

a) El **dominio de las variables** involucradas.

$$\begin{cases} C_{j,i} \geq 0 & \forall j,i \ : \ j \neq 0 \\ X_{k,j} \in \{0,1\} & \forall k,j \ : \ j \neq k \end{cases} \qquad \begin{cases} CA_s \geq 0 & \forall s \ : \ s \neq 0 \\ Y_{l,s} \in \{0,1\} & \forall l,s \ : \ l \neq s \\ C_{\text{máx}} \geq 0 \end{cases}$$

b) **Cada componente debe aparecer en una única posición de la secuencia de producción**. Por tanto,

$$\sum_{\substack{k=0 \\ j \neq k}}^{n} X_{k,j} = 1 \qquad \forall j : j \neq 0$$

c) **Definición de la variable $\mathbf{X}_{.,.}$** con respecto a la **relación de precedencia inmediata**.

$$\sum_{\substack{j=0 \\ j\neq k}}^{n} X_{k,j} \leq 1 \qquad \forall k : k \neq 0$$

d) Como hay F fábricas, **se emplean exactamente F separadores** en la secuencia global de producción, **que suceden a F − 1 componentes** secuenciados.

$$\sum_{j=1}^{n} X_{0,j} = F \qquad\qquad \sum_{k=1}^{n} X_{k,0} = F - 1$$

e) **Dados dos componentes** (reales) a procesar

- o aparecen juntos en la secuencia de producción (uno precede al otro)
- o están separados (y no hay relación de precedencia entre ellos en la secuencia de producción).

$$X_{k,j} + X_{j,k} \leq 1 \qquad \forall k \in \{1, \ldots, n-1\},\ \forall j > k$$

f) **Definición de completación** para cada componente y para cada máquina de la ruta de fabricación.

Estas dos desigualdades, separadamente, acotan el tiempo de completación inferiormente. Pero combinadas y teniendo en cuenta que se está minimizando $C_{\text{máx}}$ y la relación entre las líneas de fabricación y ensamblaje, definen correctamente el tiempo de completación para cada componente y para máquina de la ruta.

Nótese, antes de comenzar, que se define $C_{j,0} := 0 := C_{0,l}\ \forall j, l$.

Primera Desigualdad

$$C_{j,i} \geq C_{j,i-1} + P_{j,i} \qquad \forall j, i : j \neq 0$$

El tiempo de completación del componente j en la máquina i

Debe ser superior o igual a...

El tiempo de completación acumulado en la ruta

Más

El tiempo de procesamiento de j en i

Segunda Desigualdad

$$C_{j,i} \geq C_{k,i} + P_{j,i} + M\,(X_{k,j} - 1) \qquad \forall j,k,i \;:\; k \neq j,\; j \neq 0$$

Analicémosla por casos

- **Si el componente k precede al componente j**, entonces

$$X_{k,j} = 1$$

Y, por tanto, la desigualdad se traduce en que

$$C_{j,i} \;\geq\; C_{k,i} \;+\; P_{j,i}$$

El tiempo de completación del componente j en la máquina i

Debe ser superior o igual a...

El tiempo de completación en la máquina i del trabajo previo a j, k

Más

El tiempo de procesamiento de j en i

- **Si en ninguna factoría el componente k precede al componente j**, la restricción propuesta equivale a

$$C_{j,i} \geq C_{k,i} + P_{j,i} - M$$

Como, en este caso, la restricción no debe modelizar ninguna condición lógica, es necesario anularla. Tomando M suficientemente grande, se establece que $C_{j,i} \geq 0$.

g) **Cada producto final debe aparecer en una única posición de la secuencia de ensamblaje**. Por tanto,

$$\sum_{\substack{l=0 \\ l \neq s}}^{t} Y_{l,s} = 1 \qquad \forall s : s \neq 0$$

h) **Definición de la variable $\mathbf{Y}_{\cdot,\cdot}$** con respecto a la **relación de precedencia inmediata**.

$$\sum_{\substack{s=1 \\ l \neq s}}^{t} Y_{l,s} \leq 1 \qquad \forall l : l \neq 0$$

i) **Dados dos productos finales** (reales) a ensamblar

- o aparecen juntos en la secuencia de montaje (uno precede al otro)
- o están separados (y no hay relación de precedencia entre ellos en la secuencia de montaje).

$$Y_{l,s} + Y_{s,l} \leq 1 \qquad \forall l \in \{1,\ldots,n-1\},\; \forall s > l$$

j) **Definición de completación** para cada producto final en la línea de montaje.

El razonamiento que se hizo con las desigualdades de f), permite afirmar que las siguientes dos desigualdades definen correctamente el tiempo de completación para cada producto final.

Nótese, antes de comenzar, que se define $CA_0 := 0$.

Primera Desigualdad

Esta restricción se encarga de impedir que el montaje de un determinado producto pueda comenzar antes de que se hayan fabricado los componentes que lo conforman. Nótese que $G_{j,s}$ es un parámetro cuyo valor está definido y es binario.

$$CA_s \geq \underbrace{G_{j,s}C_{j,m}} + PA_s \qquad \forall j, s : j \neq 0, s \neq 0$$

El tiempo de completación del producto final s en la línea de ensamblaje

Debe ser superior o igual a...

El tiempo en el que están disponibles los componentes que lo conforman

Más

El tiempo de montaje de s

Segunda Desigualdad

$$CA_s \geq CA_l + PA_s + M\,(Y_{l,s} - 1) \qquad \forall l, s \ : \ l \neq s, \ s \neq 0$$

Analicémosla por casos

- **Si el producto l precede al producto s** en la línea de ensamblaje, entonces

  $$Y_{l,s} = 1$$

  Y, por tanto, la desigualdad se traduce en que

  $$CA_s \geq CA_l + PA_s$$

  El tiempo de completación del producto final s en la línea de ensamblaje

  Debe ser superior o igual a...

  El tiempo de completación en la línea de montaje del producto previo a s, l

  Más

  El tiempo de montaje de s

- **Si en la línea de montaje el producto l no precede al componente s**, la restricción propuesta equivale a

  $$CA_s \geq CA_l + PA_s - M$$

  Como, en este caso, la restricción no debe modelizar ninguna condición lógica, es necesario anularla. Tomando M suficientemente grande, se establece que $CA_s \geq 0$.

k) **Definición del tiempo máximo de completación** de todos los productos finales en la línea de montaje

Se establece que

$$C_{\text{máx}} \geq CA_s \qquad \forall s \ : \ s \neq 0$$

Y como el problema de optimización tiene como objetivo minimizar el valor de $C_{\text{máx}}$, la anterior expresión se cumplirá sin holgura para algún producto final. Y, por consiguiente, $C_{\text{máx}}$ será, efectivamente, el tiempo máximo de completación en la línea de montaje.

Analizando el impacto a nivel computacional de este modelo, el **número de restricciones requeridas** para modelizar la planificación de n componentes, en F fábricas con m máquinas cada una y que serán ensamblados finalmente en una línea de montaje para conformar t productos finales, es

$$\mathbf{2n + 2 + n(n+1)m + \frac{1}{2}n(n-1) + 3t + t(t+n) + \frac{1}{2}t(t-1)}$$

Este número se obtiene a partir de la siguiente tabla, en la que se calcula, para cada expresión, la cantidad de restricciones que genera.

Restricción	**Cantidad**	**Restricción**	**Cantidad**
a)	0	g)	t
b)	n	h)	t
c)	n	i)	$t(t-1)/2$
d)	2	j) - 1)	nt
e)	$n(n-1)/2$	j) - 2)	$t(t-1)+t$
f) - 1)	nm	k)	t
f) - 2)	$n(n-1)m + nm$		

3.3. Rendimiento del modelo analizado

El objetivo que persigue en esta sección es **presentar una evaluación experimental del modelo** analizado anteriormente, que permita **mostrar evidencias sobre su rendimiento**.

La valoración tendrá lugar mediante la generación aleatoria de pequeñas instancias del problema de producción DAPFSP. Cada una de ellas se tratará de resolver en el *solver* comercial CPLEX 12.7.1 haciendo uso del modelo estudiado.

3.3.1. Configuración del equipo

La computación se ejecutará sobre la siguiente configuración:

CPU	Intel Core i5-750 @2.67GHz
Placa base	Gigabyte P55A-UD4
Memoria Ram	Kingston 8+4GiB @1333MHZ
Swap	40GiB sobre SSD SanDisk Ultra II en SATA3
Sistema Operativo	Ubuntu 16.04 + Kernel 4.13.0-38-generic
Notas	La computación tiene lugar en tty1 con las X detenidas (*service lightdm stop*). El swappiness del núcleo de Linux se fija en 10 (*sysctl -w vm.swappiness=10*).

La configuración es idéntica a la utilizada en la sección 2.4, salvo por la cantidad de memoria Swap disponible, que duplica a la permitida para resolver las instancias de DPFSP. Una decisión totalmente lógica al tener en cuenta que DAPFSP combina dos problemas asociados al *scheduling* (DPFSP y AFSP).

3.3.2. Generación de las instancias e identificación de factores

Las **instancias** se generarán a partir de la **combinación de cuatro parámetros que definen el problema** DAPFSP. A saber:

- El número de componentes que se deben producir, tomándose $\mathbf{n} \in \{\mathbf{4}, \mathbf{8}, \mathbf{12}, \mathbf{16}\}$.
- El número de fábricas disponibles para producirlos, evaluándose $\mathbf{F} \in \{\mathbf{2}, \mathbf{3}, \mathbf{4}\}$.
- El número de máquinas (idéntico en todas las fábricas) que conforman el procesado de los componentes, restringiéndose a $\mathbf{m} \in \{\mathbf{2}, \mathbf{4}, \mathbf{6}\}$.
- El número de productos finales a fabricar, donde $\mathbf{f} \in \{\mathbf{2}, \mathbf{3}, \mathbf{4}\}$

La combinación de los anteriores valores generan **108 posibles problemas** diferentes. Y, para cada uno de ellos, se generarán **5 instancias diferentes**. Por tanto, se dispondrá, **en total**, de **540 instancias** sobre las que ejecutar el modelo propuesto.

En cada una de estas instancias, el **tiempo de procesado** de cada componente en las diferentes máquinas y el **tiempo de ensamblaje** necesario para construir cada producto final será entero y se tomará aleatoria y **uniformemente sobre el rango** $(\mathbf{1}, \mathbf{99})$. **Los componentes que conforman cada producto final se elegirán, en cada instancia, de forma aleatoria**.

De idéntica forma a como se hizo en 2.4, **cada ejecución dispondrá de un tiempo máximo** de ejecución **y un número máximo de núcleos** (*threads*) disponibles. En concreto, cada instancia será ejecutada 4 veces: una para cada posible configuración obtenida a través de la combinación de estos dos factores:

- El tiempo máximo disponible, tomándose en $\{600s, 3600s\}$
- El número máximo de núcleos, tomándose en $\{3 \text{ núcleos}, 4 \text{ núcleos}\}$.

El factor tiempo limita la ejecución de los modelos a tiempos razonables de computación (10 y 60 minutos, respectivamente). Y es que, en problemas pequeños como los que se plantean en este estudio, **es impensable** tener que **esperar más tiempo para obtener una solución óptima**.

Los tiempos elegidos son justo el doble a los considerados para el estudio del rendimiento del problema DPFSP, considerándose 10 minutos como el valor ideal máximo.

En lo que respecta al **factor limitante de núcleos** disponibles, éste responde a los **límites domésticos** que marca la **computación actual**: procesadores con cuatro núcleos. A raíz de esto, se estudia el efecto *número de threads* con los niveles 3 y 4 núcleos para computar las instancias, al igual que se hizo con el problema DPFSP.

Conviene observar que el sistema operativo, el lanzador y CPLEX van a compartir núcleo cuando se permita, en la ejecución, el uso de 4 *threads*. Y que este comportamiento **podría sesgar el estudio**. Para aclarar esta situación, se recomienda al lector que se dirija a la subsección 2.4.2, homónima a ésta.

En resumidas cuentas, **la siguiente tabla muestra los diferentes factores contemplados para la computación y evaluación del rendimiento de los modelos analizados**:

Factor del problema	Símbolo	**Número de niveles**	**Valores**
Número de componentes a producir	n	4	4, 8, 12, 16
Número de fábricas disponibles	F	3	2, 3, 4
Número de máquinas	m	3	2, 4, 6
Número de trabajos finales	t	3	2, 3, 4

Factor computacional	Símbolo	**Número de niveles**	**Valores**
Solver	*Solver*	1	CPLEX 12.7.1
Núcleos disponibles	*ThreadLimit*	2	3, 4
Tiempo disponible	*TimeLimit*	2	600s, 3600s

3.3.3. Programación de la ejecución

Usando como referencia la subsección 2.4.3, como lanzador para organizar la ejecución de las diferentes instancias con sus características específicas y factores, se empleará un script en `R` (3.4.4, x86_64-pc-linux-gnu). Éste hará uso de la librería de tipo *wrapper* de CPLEX para `C` a través del paquete `Rcplex` (1.3.3).

Los datos recogidos, para cada instancia y ejecución, serán:

- El **tamaño del problema** (factores problema): n, m, F y t.
- Las **limitaciones computacionales** impuestas (factores computación): *ThreadLimit* y *TimeLimit*.
- El **tiempo de computación dedicado** por instancia.
- El estado del solver CPLEX al finalizar su ejecución (**si ha encontrado la solución óptima o no**).
- Y **el GAP alcanzado** por CPLEX en su ejecución.

3.3.4. Análisis descriptivo del rendimiento

Inspirado en la subsección 2.4.4, se elaborarán diferentes tablas para analizar el efecto de los factores computacionales y de algunos de los factores problema. Las tablas mostrarán:

- La **tasa de optimalidad alcanzada**,
- El **GAP medio conseguido** y
- El **tiempo empleado por CPLEX**,

en cada instancia, para cada factor representado en ellas.

A cada tabla **se le ha aplicado un formato condicional para facilitar la comparación de los valores mostrados**, clasificados según factores. Por ello,

- Se muestran barras de datos para comparar (relativamente, entre el mejor de los datos de esa categoría), la tasa de optimalidad alcanzada (en positivo) y el GAP medio alcanzado.
- Y se escoge una representación icónica a través de la esfera de un reloj para comparar el tiempo empleado.

Por su lado, la **figura 3.1 analiza el efecto** que tienen los **factores computacionales y el número de componentes a fabricar** sobre los estadísticos de interés estudiados. Se observa que:

- El **efecto del factor *número de núcleos usados en la computación* parece ser muy débil** puesto que los valores de los estadísticos analizados son prácticamente idénticos cuando únicamente se tiene en cuenta este factor.
- Existe un **claro efecto del factor *número de componentes a fabricar*** puesto que los valores de los estadísticos analizados son muy dispares cuando únicamente se tiene en cuenta este factor. Por lo que se observa, al aumentar el número de componentes a fabricar, la instancia resulta más difícil de resolver.
- El **efecto**, sobre los estadísticos estudiados, del ***límite** impuesto hacia el **tiempo de computación*** **depende del número de componentes que se fabriquen**. Para valores pequeños ($n = 4, n = 8$) parece no existir efecto, pero cuando aumenta la cifra de componentes ($n = 12, n = 16$), el *solver* necesita mucho más tiempo para resolver óptimanente la instancia.

Las tablas de la **figura 3.2** permiten **observar** el **efecto** que tienen el ***número de componentes a producir*** y el ***número de fábricas*** sobre los estadísticos de interés estudiados, al considerar como **factor clasificatorio el *tiempo máximo permitido para computar***.

A partir de ellas dos, se puede afirmar **cualitativamente**, que, **cuando se dispone de un mayor número de fábricas la complejidad computacional se reduce**.

Esto se observa, por ejemplo, cuando se programa la producción de 12 componentes, donde el porcentaje de instancias resueltas aumenta de forma súbita al pasar de dos a tres fábricas. E incluso cuando $n = 16$, que, de no conseguir resolver ninguna instancia de forma óptima cuando se dispone de 2 y 3 fábricas y 3600 segundos de computación, a conseguir la optimalidad en el 8.89 % de las instancias estudiadas con $f = 4$.

Y las tablas de la figura 3.3 examinan el **impacto que tiene el *número de componentes a producir* y el *número de productos finales a ensamblar*** sobre los estadísticos de interés estudiados, al considerar como **factor clasificatorio el *tiempo máximo permitido para computar***.

De ellas se deduce que el **efecto del factor *número de productos finales a ensamblar* parece ser muy débil** puesto que los valores de los estadísticos analizados son prácticamente idénticos cuando únicamente se tiene en cuenta este factor.

Caracteristicas de la instancia	Límite tiempo	600s		3600s		Valores medios
	Núcleos	3	4	3	4	
n = 4	% opt	100,00	100,00	100,00	100,00	100,00
	% GAP	0,00	0,00	0,00	0,00	0,00
	Tiempo medio (s)	0,02	0,02	0,02	0,02	0,02
n = 8	% opt	100,00	100,00	100,00	100,00	100,00
	% GAP	0,03	0,03	0,03	0,03	0,03
	Tiempo medio (s)	0,21	0,17	0,21	0,17	0,19
n = 12	% opt	59,26	63,70	67,41	69,63	65,00
	% GAP	9,40	9,06	6,47	6,09	7,75
	Tiempo medio (s)	311,88	291,68	1328,12	1272,52	801,05
n = 16	% opt	0,00	0,00	2,22	3,70	1,48
	% GAP	27,06	26,96	25,51	25,25	26,19
	Tiempo medio (s)	600,17	600,27	3549,25	3540,72	2072,60
Valores medios	% opt	*64,81*	*65,93*	*67,41*	*68,33*	*66,62*
	% GAP	*9,12*	*9,01*	*8,00*	*7,84*	*8,49*
	Tiempo medio (s)	*228,07*	*223,03*	*1219,40*	*1203,36*	*718,47*

Figura 3.1: **Tabla resumen y comparativa del rendimiento mostrado por el modelo estudiado**. Por ejemplo, cuando se trata de resolver una de las instancias propuestas en la que se deben producir 16 componentes, el modelo no encuentra nunca una solución óptima al permitirle usar 3 núcleos y un tiempo máximo de 600s. El GAP medio alcanzado, en este escenario, es del 27.06 %. Empleará, de media, 600.17 segundos de computación para procesarla.

Caracteristicas de la instancia	Límite tiempo	600s			Valores medios
	Núm. Fábricas	2	3	4	
n = 4	% opt	100,00	100,00	100,00	100,00
	% GAP	0,00	0,00	0,00	0,00
	Tiempo medio (s)	0,02	0,02	0,01	0,02
n = 8	% opt	100,00	100,00	100,00	100,00
	% GAP	0,00	0,00	0,08	0,03
	Tiempo medio (s)	0,43	0,10	0,05	0,19
n = 12	% opt	0,00	84,44	100,00	61,48
	% GAP	26,95	0,74	0,00	9,23
	Tiempo medio (s)	600,03	292,81	12,51	301,78
n = 16	% opt	0,00	0,00	0,00	0,00
	% GAP	42,96	25,42	12,64	27,01
	Tiempo medio (s)	600,01	600,14	600,51	600,22
Valores medios	% opt	*50,00*	*71,11*	*75,00*	*65,37*
	% GAP	*17,48*	*6,54*	*3,18*	*9,07*
	Tiempo medio (s)	*300,12*	*223,27*	*153,27*	*225,55*

Caracteristicas de la instancia	Límite tiempo	3600s			Valores medios
	Núm. Fábricas	2	3	4	
n = 4	% opt	100,00	100,00	100,00	100,00
	% GAP	0,00	0,00	0,00	0,00
	Tiempo medio (s)	0,02	0,02	0,01	0,02
n = 8	% opt	100,00	100,00	100,00	100,00
	% GAP	0,00	0,00	0,08	0,03
	Tiempo medio (s)	0,43	0,10	0,05	0,19
n = 12	% opt	5,56	100,00	100,00	68,52
	% GAP	18,83	0,00	0,00	6,28
	Tiempo medio (s)	3544,96	343,49	12,51	1300,32
n = 16	% opt	0,00	0,00	8,89	2,96
	% GAP	42,01	24,30	9,83	25,38
	Tiempo medio (s)	3600,78	3601,17	3433,00	3544,98
Valores medios	% opt	*51,39*	*75,00*	*77,22*	*67,87*
	% GAP	*15,21*	*6,07*	*2,48*	*7,92*
	Tiempo medio (s)	*1786,55*	*986,19*	*861,39*	*1211,38*

Figura 3.2: **Tablas resumen y comparativas del rendimiento mostrado por el modelo estudiado** al considerarse como factores discrimantes el ***número de componentes a producir*** y el ***número de fábricas***. La primera tabla muestra los valores de los estadísticos de interés estudiados cuando se limita la ejecución del *solver* a 600 segundos. Y la segunda, cuando es de 3600 segundos.

Por ejemplo, cuando se trata de resolver una de las instancias propuestas en la que se deben producir 16 componentes en 2 fábricas, el modelo no encuentra nunca una solución óptima al permitirle tiempo máximo de 3600s. El GAP medio alcanzado, en este escenario, es del 42.01 %. Empleará, de media, 3600.78 segundos de computación para procesarla.

Caracteristicas de la instancia	Límite tiempo	600s			Valores medios
	Num. Productos	2	3	4	
n = 4	% opt	100,00	100,00	100,00	100,00
	% GAP	0,00	0,00	0,00	0,00
	Tiempo medio (s)	0,02	0,02	0,02	0,02
n = 8	% opt	100,00	100,00	100,00	100,00
	% GAP	0,00	0,00	0,08	0,03
	Tiempo medio (s)	0,20	0,17	0,20	0,19
n = 12	% opt	61,11	61,11	62,22	61,48
	% GAP	9,33	9,65	8,71	9,23
	Tiempo medio (s)	306,06	289,08	310,21	301,78
n = 16	% opt	0,00	0,00	0,00	0,00
	% GAP	26,56	27,56	26,90	27,01
	Tiempo medio (s)	600,20	600,24	600,22	600,22
Valores medios	% opt	*65,28*	*65,28*	*65,56*	*65,37*
	% GAP	*8,97*	*9,30*	*8,92*	*9,07*
	Tiempo medio (s)	*226,62*	*222,37*	*227,66*	*225,55*

Caracteristicas de la instancia	Límite tiempo	3600s			Valores medios
	Num. Productos	2	3	4	
n = 4	% opt	100,00	100,00	100,00	100,00
	% GAP	0,00	0,00	0,00	0,00
	Tiempo medio (s)	0,02	0,02	0,02	0,02
n = 8	% opt	100,00	100,00	100,00	100,00
	% GAP	0,00	0,00	0,08	0,03
	Tiempo medio (s)	0,20	0,17	0,20	0,19
n = 12	% opt	67,78	67,78	70,00	68,52
	% GAP	6,46	6,74	5,64	6,28
	Tiempo medio (s)	1316,26	1314,87	1269,81	1300,32
n = 16	% opt	4,44	2,22	2,22	2,96
	% GAP	25,04	25,94	25,16	25,38
	Tiempo medio (s)	3500,90	3554,69	3579,37	3544,98
Valores medios	% opt	*68,06*	*67,50*	*68,06*	*67,87*
	% GAP	*7,87*	*8,17*	*7,72*	*7,92*
	Tiempo medio (s)	*1204,35*	*1217,44*	*1212,35*	*1211,38*

Figura 3.3: **Tablas resumen y comparativas del rendimiento mostrado por el modelo estudiado** al considerarse como factores discrimantes el ***número de componentes a producir*** y el ***número de productos finales***. La primera tabla muestra los valores de los estadísticos de interés estudiados cuando se limita la ejecución del *solver* a 600 segundos. Y la segunda, cuando es de 3600 segundos.

Por ejemplo, cuando se trata de resolver una de las instancias propuestas en la que se deben producir 16 componentes para 2 productos finales, el modelo encuentra una solución óptima en el 4.44 % de los casos, al permitirle tiempo máximo de 3600s. El GAP medio alcanzado, en este escenario, es del 25.04 %. Empleará, de media, 3500.90 segundos de computación para procesarla.

3.3.5. Análisis formal de los datos

Se usa como referencia la subsección 2.4.5.

Aunque la literatura científica normalmente realiza un análisis estadístico empleando técnicas CHAID (*Chi-Squared Automatic Iteration Detection*), el análisis que se llevará a cabo usará **técnicas del aprendizaje-máquina** (*machine learning, ensemble learning*) para clasificar los resultados obtenidos.

Al igual que sucedió en la la subsección 2.4.5, tras haber probado una batería de modelos y técnicas diferentes con los que tratar los datos (como GLM, Boosted LM, Boosted Tree o Logistic Model Trees), la que proporcionó **mejores resultados** fue ***Random Forest***, con la implementación del paquete `randomForest` en `R`.

Random Forest.

La **variable respuesta** del modelo será categórica: el **tipo de solución obtenida en CPLEX** (S si la solución es óptima, N si no lo es). Y la **variabilidad** observada se tratará de **explicar mediante** las siguientes variables:

- El **número de componentes** fabricados, `n`
- El **número de fábricas** disponibles, `f`
- El **número de máquinas** estudiadas, `m`
- El **número de productos finales** construidos, `t`
- El **tiempo** de computación **disponible**, `t_disponible`
- El número de **núcleos disponibles**, `cpu_disponible`.

Las **instancias** serán particionadas en **dos subconjuntos**: de entrenamiento y de validación. La partición se hace **de forma equilibrada** entre las diferentes clases y factores: La primera instancia de cada configuración se toma para la validación, y las cuatro restantes para el entrenamiento.

Árboles generados.

RandomForest combina una gran cantidad de árboles de decisión independientes para tratar de modelizar, a través de ellos, los datos presentados.

La primera decisión que se debe tomar para construir *RandomForest* es **determinar el número de árboles a usar** en su generación. Por ello se decide hacer uso de una **validación** para determinar este parámetro. Entre un conjunto de posibles valores prefijados y arbitrarios para configurar el número de árboles a usar en el modelo, se elegirá finalmente aquel que le proporcione a *RandomForest* mayor tasa de acierto (precisión) al ejecutarse sobre el conjunto de validación.

El resultado de esta validación se observa en la figura 3.4, donde se comprueba que **usando 5 árboles se obtiene la mayor precisión**. A partir de 16 árboles, la precisión no aumenta.

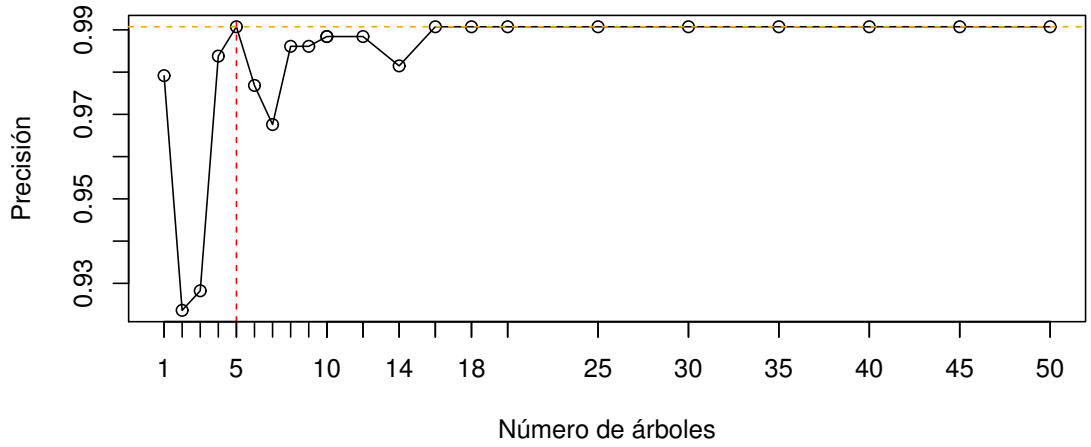


Figura 3.4: **Análisis del parámetro *número de árboles* en la generación de *RandomForest***. Se estudia la influencia que tiene el número de árboles empleados para generar el modelo sobre la precisión final de éste. La precisión se calcula al ejecutar el modelo obtenido sobre el conjunto de validación. **La gráfica muestra que, cuando se usan 5 árboles, la precisión obtenida es máxima**.

Exploración de un árbol generado.

De los 5 árboles construidos en este estudio para explicar la resolubilidad óptima de instancias a través de CPLEX, en la **figura 3.5 se muestra el primer árbol generado por random forest**. Y es que resulta de vital importancia explorar alguno de estos árboles para ganar intuición sobre las las relaciones que se han formado entre las variables.

El árbol resulta llamativo tras la primera división. Cuando $n = 16$ (N), el modelo no es capaz de resolver el problema de forma óptima, salvo casos muy contados. El árbol trata de ajustar este comportamiento según el valor que toman el resto de los factores, usando más nodos y dividiéndose en más ramas. Finalmente, no consigue explicar el porqué de este ínfimo número de instancias sí resueltas. El árbol las predecirá como no resueltas, cometiendo un error.

Este comportamiento también aparece en otras partes del árbol. Por ejemplo, tras el tercer nivel, cuando $n = 12$(N), $f = 2$ (Y) y $m = 2$ o 4 (Y). O en el cuarto nivel, cuando $n = 12$ (N), $f = 3, 4$ (N) y $m = 2$ (Y). O también en este cuarto nivel, cuando $n = 12$ (N), $f = 3, 4$ (N) y $m = 4, 6$ (Y).

La situación es totalmente opuesta cuando $n = 4$ o $n = 8$, donde el modelo estudiado siempre puede resolver las instancias propuestas. Y esto queda reflejado en el árbol de la figura 3.5.

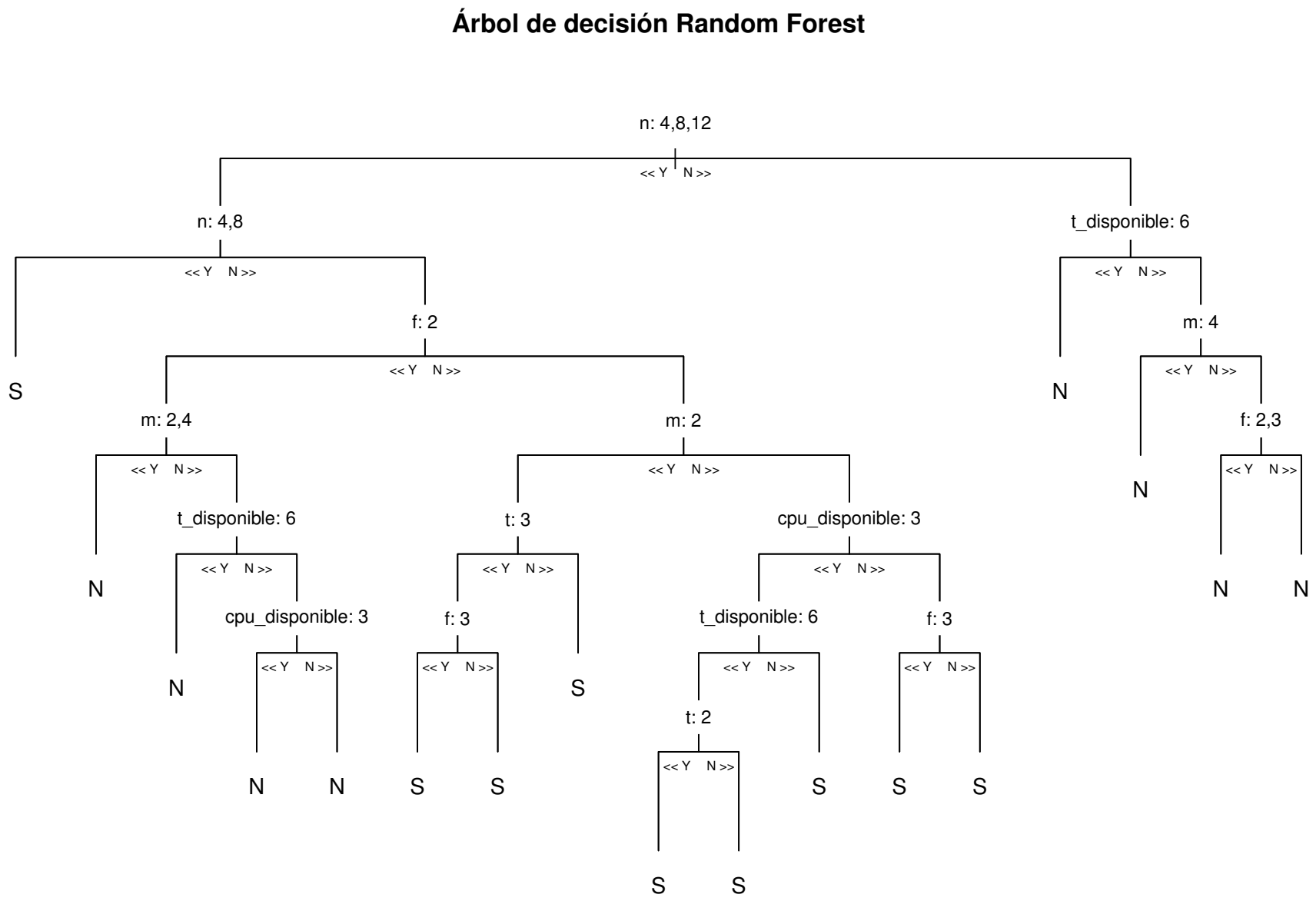


Figura 3.5: **Árbol de decisión generado en la primera iteración de Random Forest**. El árbol ofrece hasta 7 niveles para algunos nodos.

Importancia de las variables estudiadas.

La **figura 3.6** recoge dos gráficas que **valoran, estadísticamente,** la **importancia de los factores considerados** en este estudio para explicar la variabilidad que presenta la variable respuesta sobre los datos experimentales recogidos.

Las dos gráficas coinciden al señalar las principales variables según su importancia (*en presencia de las restantes*[v]). La **variable más importante**, con diferencia, **para explicar la variabilidad observada** sobre los datos, según RandomForest, **es el número de componentes a fabricar**. Y, **en segunda posición, el número de fábricas disponibles** .

Con respecto al **resto de factores estudiados**, ***RandomForest*** los considera **prácticamente irrelevantes** para explicar la variabilidad observada, en presencia del resto de los factores.

La explicación técnica sobre las dos medidas de importancia usadas en la figura 3.6 se puede encontrar en la subsección 2.4.5, usada como referencia.

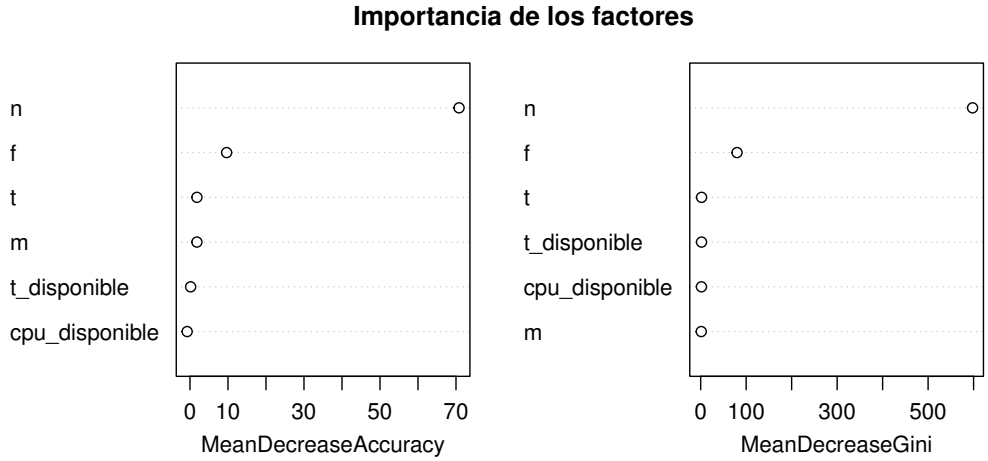


Figura 3.6: **Valoración de la importancia de los diferentes factores contemplados en el análisis con Random Forest**. Las dos medidas consideradas (la precisión y la de Gini) indican que el número de componentes a producir y el número de fábricas disponibles son los dos factores más relevantes. El resto de los factores los considera irrelevantes.

[v] Aunque no se haga explícito en cada oración, la importancia está condicionada a la presencia de las demás variables que conforman el estudio. Una variable es o no es importante, en presencia de las restantes.

Calidad del modelo.

Con respecto a la **calidad predictora del model**o, señalar que

- El valor estimado del error OOB (*Out Of Bag Error*) es del 4.74 %.
- El error en la clasificación de los valores observados en el conjunto de validación es del 0.93 %.

La matriz de confusión OOB presentada por el modelo es:

	Predicción		
Observ.	S	N	Error clasif.
S	1012	21	0.02032914
N	53	476	0.10018904

Y la obtenida al ejecutar el modelo sobre el conjunto de validación es:

	Predicción		
Observ.	S	N	Error clasif.
S	287	3	0.01034483
N	1	141	0.00704226

Este **modelo** se considera, objetivamente, **muy adecuado para explicar la variabilidad observada en los datos recogidos** con el fin de **valorar el rendimiento de los modelos presentados**. Y para usarse como clasificador y **predecir la resolubilidad de nuevas instancias** cuyas características estén incluidas dentro de los valores paramétricos estudiados.

3.3.6. Otro estudio del rendimiento del modelo

En [18] se puede encontrar otro estudio del rendimiento ofrecido por el modelo analizado en este documento. **Las condiciones de realización de éste son muy diferentes** a las que se han considerado aquí:

- Una CPU con apenas dos núcleos (vs 4 núcleos) y mucha más memoria caché disponible (12MB de caché L2 vs 8MB de SmartCache).
- Seis veces menos de memoria Ram disponible (2GB DDR3 vs 12GB DDR3).
- Diferente sistema operativo (Windows XP vs Ubuntu 16.04).
- Distinto lanzador (C++ vs R).
- Diferentes niveles para los factores del problema ($n \in \{8, 12, 16, 20, 24\}$, $m \in \{2, 3, 4, 5\}$, $F \in \{2, 3, 4\}$, $t \in \{2, 3, 4\}$) y para los computacionales (1 y 2 núcleos vs 3 y 4 núcleos; CPLEX 12.3 + GUROBI 4.6.1 vs CPLEX 12.7.1).
- Y diferentes instancias generadas (disponibles en *soa.iti.es*).

Se incluye la imagen 3.7, que resume el breve y escaso análisis descriptivo del rendimiento realizado por [18]. Aunque, lo más relevante de su estudio es el análisis estadístico con técnicas CHAID que realiza.

Los factores estudiados fueron n, m, F, t, el tiempo disponible de computación, el número de núcleos permitidos y el *solver* usado. La variable respuesta fue el tipo de solución ofrecida por CPLEX (si era óptima o no).

Este análisis señala que **el factor más importante es el número de componentes fabricados, n, seguido por el número de fábricas disponibles f**. El **número de productos finales producidos**, **t**, es el **tercer y último factor importante** (salvo para el nodo $n = 12, f = 3$, donde es el número de máquinas usadas en cada fábrica, **m**).

A partir de ahí, ya no existen divisiones estadísticamente significativas.

Solver	Time Limit	900s		3600s	
	Thread	1	2	1	2
CPLEX	% opt	59.44	61.22	63.11	61.89
	GAP%	29.62	30.77	32.23	36.46
	Av Time (s)	390.41	380.69	1426.53	1441.80
GUROBI	% opt	66.89	68.33	70.78	73.00
	GAP%	2.19	2.04	1.81	1.70
	Av Time (s)	328.15	315.57	1152.36	1089.00

Figura 3.7: Imagen tomada de [18]. Rendimiento ofrecido por el modelo analizado sobre un servidor blade con cuatro tarjetas Intel Core XEON E5420 y 16GB de RAM que virtualizaba ocho Windows XP. Los *solvers* usados fueron CPLEX 12.3 y GUROBI 4.6.1, lanzados a través de un programa escrito en C++.

3.4. Métodos heurísticos y metaheurísticos para el problema $DAF|prmu|C_{\text{máx}}$

Tal y como se afirmaba en el capítulo de introducción del documento, los modelos matemáticos son útiles para **resolver, únicamente, problemas de planificación de la producción pequeños** ya que para resolver problemas grandes requerirían un excesivo tiempo y deberían hacer un uso excesivo de recursos computacionales.

La sección 3.3 respalda empíricamente estas afirmaciones, mostrando la incapacidad del modelo planteado en 3.2.1 para resolver instancias del problema $DAF|prmu|C_{\text{máx}}$ cuando se dispone de muchas fábricas, trabajos o máquinas.

Es **imprescindible**, por tanto, usar **métodos heurísticos** o metaheurísticos para obtener, al menos, soluciones aceptables, en tiempos razonables, a problemas de planificación grandes.

Esta sección está dividida en 3 subsecciones principales:

- Una **primera subsección**, que conforma una **primera aproximación al desarrollo de heurísticos** para el problema DAPFSP cuando se pretende minimizar el tiempo máximo de completación de todas las tareas, entre todas las fábricas.
- Una **segunda subsección**, que desarrolla dos **métodos VND** (*Variable Neightbourhood Descent*) para resolver el problema $DAF|prmu|C_{\text{máx}}$, **en cuanto a que los anteriores métodos resultarán ser limitados y medianamente efectivos**.
- Y una **tercera subsección**, que reproduce un estudio estadístico para **valorar el rendimiento que ofrecen estos métodos** para resolver el problema $DF|prmu|C_{\text{máx}}$.

Además, de idéntica forma a como se hizo en el capítulo segundo de este documento, la siguiente sección, la 3.5, incluirá una **breve recopilación de métodos** para resolver la planificación en problemas $DAF|prmu|C_{\text{máx}}$. Y que, por el carácter de este documento, **no se han incluido como parte de este análisis detallado**.

3.4.1. Una primera aproximación en el desarrollo de heurísticos efectivos

En esta subsección se recoge el **primer intento** del se tiene constancia para la **elaboración de heurísticos** que resuelvan, en tiempos razonables, el **problema DAF|prmu|$\mathbf{C}_{\text{máx}}$** y que se puede encontrar en [17].

Esta **primera aproximación escinde las tres decisiones fundamentales a tomar** para programar la producción sobre este problema:

- Primeramente, **el reparto, entre las diferentes factorías disponibles**, de la producción de los componentes necesarios para conformar la totalidad de productos finales.
- A continuación, **la secuencia de producción, en cada fábrica,** de los componentes asignados a ella.
- Y, por último, sobre la línea de montaje, **la secuenciación del ensamblaje** de los componentes para conformar los diferentes productos finales.

Y **genera diferentes heurísticos variando** el algoritmo, la **forma y el orden de tomar estas decisiones**.

Pero, para poder analizar correctamente esta aproximación (e incluso futuros métodos heurísticos), se hace necesario introducir la siguiente **notación**, tomada de [17]:

- π, que denota la **secuencia** seguida en la **línea de montaje** para ensamblar los productos finales
- $\pi_{\mathbf{s}}$, que representa una **secuencia parcial de fabricación** para los **componentes** del producto final $\mathbf{s} \in \mathcal{T}$. Contiene el orden en el que se deben producir los componentes de s, pero no indica que fábrica debe procesar que componente.
- $\pi_{\mathbf{T}}$, que constituye la **concatenación** (**ordenada** según π) de las **secuencias parciales** de los diferentes productos finales. Esto es

$$\pi_T = \cup_{s \in \pi} \pi_s.$$

Recuérdese, en todo caso, la notación introducida en las secciones 3.1 y 3.2, que también resultará imprescindible.

Puesto que, para **evitar posibles interpretaciones erróneas en la nueva notación**, es necesario introducir una **situación de ejemplo**, ésta se elige para que sirva también de ejemplo para los diferentes heurísticos que se van a desarrollar.

Se supone, por tanto, un entorno de **2 fábricas**, cada una con **2 máquinas**, que producen **9 componentes** para conformar **3 productos finales** en la línea de montaje.

Además, se establece que los componentes que necesitan los productos finales sean

$$N_1 = \{2, 9, 5, 1\}, \quad N_2 = \{4, 6, 8\}, \quad N_3 = \{7, 3\}$$

Con respecto a los tiempos de fabricación de los componentes y del ensamblaje estos se transcriben en la tabla 3.1.

Así, por tanto, un **posible ejemplo**, **con la notación introducida**, de secuenciación es esta:

$$\begin{cases} \pi_1 = \{2, 1, 5, 9\} \\ \pi_2 = \{8, 6, 4\} \\ \pi_3 = \{3, 7\} \end{cases} \qquad \pi = \{2, 3, 1\} \qquad \pi_T = \{8, 6, 4, 3, 7, 2, 1, 5, 9\}$$

Componente	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Máquina 1 (*M1*)	2	1	8	7	4	9	9	3	5
Máquina 2 (*M2*)	5	3	1	5	3	7	3	4	8
Producto final		**1**			**2**			**3**	
Máquina M_A		19			6			12	

Tabla 3.1: **Tabla con los tiempos de fabricación de los componentes y del ensamblaje propuestos para ilustrar una situación de ejemplo**, con la que poder ejemplificar los heurísticos analizados y la notación empleada.

Se detallan, a continuación, los diferentes heurísticos construidos:

Heurístico 1, $H_{1,1}$ y $H_{1,2}$ [17]

Este heurístico se desarrolla **poniendo el centro de atención sobre la operación de ensamblado**. Y es que la idea detrás de esto es la de **priorizar aquellos productos finales que se ensamblan antes**, por encima de aquellos que requieren más tiempo en la línea de montaje.

Por ello se hace uso de la **regla SPT**[VI], sobre el tiempo necesario para ensamblar los productos finales, para decidir la secuenciación en la línea de montaje. Es decir, para construir π.

Una vez secuenciada la línea de montaje, se **prioriza la producción de los componentes que forman parte de los primeros productos finales** a ensamblar. La intención de esto es contar, cuanto antes, de estos componentes para poder iniciar el ensamblaje (y evitar esperas innecesarias sobre la línea de montaje).

Ahora bien, las dos decisiones sobre la producción (el orden de producción y la asignación entre fábricas) se toman por separado:

Por un lado, **para determinar las secuencias parciales de fabricación de los componentes de los productos finales**, $\{\pi_s\}_{s\in\mathcal{T}}$, el heurístico hace uso de un **algoritmo derivado del heurístico** de Framinan y Leisten (**FL**) para problemas en los que se quiere minimizar el *flow time* total, y que se puede leer en [8].

En concreto, el algoritmo usado **procederá de esta forma**:

1. Inicialmente, **construirá el conjunto $\mathbf{R_h}$**, que contendrá una lista de los componentes necesarios para ensamblar el producto final h. Esta lista estará ordenada de forma descendente según el tiempo mínimo necesario para producir cada uno de estos componentes.

2. A continuación, seleccionará (y eliminará) los dos primeros componentes de R_h y formará con ellos el **conjunto ordenado $\mathbf{S_h}$**. Entre las dos posibles ordenaciones de estos dos componentes, elegirá aquella que cause un menor *makespan*. Este *makespan* será calculado considerando que ambos componentes son producidos en la misma factoría.

3. Sucesivamente, seleccionará (y eliminará) **secuencialmente un componente de $\mathbf{R_h}$ para insertarlo en $\mathbf{S_h}$**. El algoritmo probará la inserción en cada una de las posibles posiciones dentro de S_h y seleccionará, finalmente, aquella que cause un menor *makespan*. Este *makespan* será calculado considerando que los componentes son producidos en la misma factoría.

4. **Tras cada iteración del paso anterior**, se tratará de **mejorar** a **$\mathbf{S_h}$ probando todas las posibles permutas de posición entre dos de los componentes**, para todas las posibles parejas de componentes (*all posible sequences by carrying out pairwise exchanges between jobs*). Y finalmente se elegirá aquella que más mejore la asignación de S_h (si hay alguna).

[VI] Se puede encontrar una descripción detallada de la regla SPT en el anexo A.

Por otro lado, para **resolver en que factoría se produce cada uno de los componentes** se emplea una de estas dos **reglas**: **NR 1** o **NR 2**[VII]. **En función de cual se use**, se hablará del **heurístico $\mathbf{H_{1,1}}$** o **$\mathbf{H_{1,2}}$**, respectivamente.

Y, con todo ello, quedará secuenciada totalmente la producción y ensamblaje.

Por completitud, a continuación, se reproduce el **algoritmo de este primer heurístico**:

Algoritmo 3.1 Primer heurístico, $H_{1,1}$ y $H_{1,2}$

Entrada: Los datos relativos al problema a resolver.
Salida: Una solución para resolver el problema en cuestión.

1: $\pi \leftarrow$ Secuenciación, para la línea de montaje, de los productos finales usando la regla SPT (a partir del tiempo necesario para ensamblar cada producto final).

2: **para** $s \in \mathcal{T}$ **hacer**

3: $\pi_s \leftarrow$ Secuencia parcial de fabricación de los componentes necesarios para ensamblar el producto final s, obtenida usando el algoritmo basado en FL.

4: **fin para**

5: $\pi_T \leftarrow \cup_{s \in \pi} \pi_s$

6: Asignar la producción de los componentes a las diferentes fábricas del problema usando la regla NR 1 (para $H_{1,1}$) o la regla NR 2 (para $H_{1,2}$).

[VII] Estas dos reglas, de [34], para establecer el **reparto**, entre las factorías de **producción**, de los componentes a fabricar fueron analizadas en este documento, en 2.5.1. Aunque, para comodidad del lector, se vuelven a reproducir (previa adaptación) a continuación:

Regla NR 1. Asignar la producción del componente j a la factoría que, tras las asignaciones previas y sin considerar la carga extra que aporta j, tenga un $\mathbf{C}_{\text{máx}}$ menor.

Regla NR 2. Asignar la producción del componente j a la factoría que, tras asignarle la producción este componente, tenga un $\mathbf{C}_{\text{máx}}$ menor.

Para clarificarlo, se resuelve a través de $H_{1,1}$ y $H_{1,2}$ el problema de ejemplo propuesto anteriormente (definido a partir de la tabla 3.1):

1. Usando la **regla SPT** sobre el tiempo necesario para ensamblar los productos finales, se determina que la secuencia a seguir en la línea de montaje es $\pi = \{2, 3, 1\}$.

2. Aplicando el **algoritmo** basado en **FL**, se obtiene que $\pi_{\mathbf{2}} = \{\mathbf{8}, \mathbf{6}, \mathbf{4}\}$. Este algoritmo procede de la siguiente forma:

   - Calcula los **tiempos mínimos necesarios para producir los componentes del segundo producto final**:

     $$C_{2,4} = 12, \qquad C_{2,6} = 16, \qquad C_{2,8} = 7$$

   - Construye el conjunto $R_2 = \{8, 4, 6\}$
   - **Forma** $\mathbf{S_2} = \{\mathbf{8}, \mathbf{4}\}$ porque, **entre las dos posibilidades de ordenación** ($\{8, 4\}$ y $\{4, 8\}$) es la que tiene un menor *makespan* al considerar que ambos componentes se producen en la misma factoría (15 y 16 unidades temporales, respectivamente).
   - **Prueba las tres posibilidades de inserción del trabajo** 6 sobre S_1: $\{6, 8, 4\}$, $\{8, 6, 4\}$ y $\{8, 4, 6\}$ y calcula los respectivos *makespan* al producirse todo en una misma fábrica: 25, 24 y 26 unidades temporales, respectivamente. Y, finalmente, actualiza a S_2 con la que tenga menor *makespan*. Por tanto,

     $$S_2 = \{8, 6, 4\}$$

   - **Prueba todos las posibles permutas de dos componentes** de S_1: $\{6, 8, 4\}$, $\{4, 6, 8\}$ y $\{8, 4, 6\}$. Los respectivos *makespans* al producirse todo en una misma fábrica son 25, 27 y 26 unidades temporales. Como ninguna mejora a S_2, ésta no se actualiza.
   - El proceso termina aquí porque no hay más componentes en R_2.

3. **Análogamente**, empleando el algoritmo basado en FL, se llega a que $\pi_1 = \{2, 1, 5, 9\}$ y $\pi_3 = \{7, 3\}$. El *makespan* asociado a estas dos secuenciaciones parciales es 20 y 18, respectivamente.

4. **Se construye**, usando π y $\{\pi_s\}_{s \in \mathcal{T}}$, a $\pi_{\mathbf{T}}$:

   $$\pi_T = \{8, 6, 4, 7, 3, 2, 1, 5, 9\}$$

5. **Aplicando la regla NR1 se lleva a la solución dibujada en la figura 3.8-(a)**, con un $C_{\text{máx}} = 55$. Y **aplicando la regla NR2, a la solución de 3.8-(b)**, con un $C_{\text{máx}} = 53$.

Heurístico 2, $H_{2,1}$ y $H_{2,2}$ [17]

La idea detrás de este heurístico es la de hacer uso del pseudoaxioma **ESAP** (*Earliest Start Time to Assembly Product*). Es decir, **priorizar el ensamblado de aquellos productos cuyos componentes requieran menor tiempo de producción en las factorías**.

Por esto se pone el foco en la **etapa de producción**, para poder iniciar la operación de ensamblaje lo antes posible en aquellos productos finales más rápidos de producir. De ahí que **se calcule, para cada producto final s**, el tiempo mínimo necesario para producir los componentes de éste y comenzar, así, su operación de ensamblaje. Este tiempo se denotará por $\mathbf{E_s}$.

Algorítmicamente, este heurístico **comienza determinando** el orden de producción de los componentes necesarios para un mismo producto final. Es decir, decidiendo **las secuencias parciales de fabricación de los componentes de los productos finales**, $\{\pi_s\}_{s\in\mathcal{T}}$. Para ello, hace uso del mismo **algoritmo derivado del heurístico** de Framinan y Leisten (**FL**) que se explicó cuando se analizó $H_{1,1}$ y $H_{1,2}$.

A continuación **calcula $\mathbf{E_s}$ para todo producto final** s. Es decir, el tiempo mínimo necesario para producir los componentes de cada uno de los productos finales.

Este cálculo se realiza **usando las reglas NR 1 o NR 2** (cuando se trate de $H_{2,1}$ o $H_{2,2}$, respectivamente) **para simular la asignación de la producción de los componentes del producto s a las diferentes factorías de las que se dispone**. Se ignoran, así, en esta simulación, el resto de componentes a fabricar y las posibles cargas previas o futuras, no relacionadas con el producto final s, que puedan tener las factorías.

El objetivo de este cálculo es decidir la secuenciación en la línea de montaje, priorizando el ensamblado de aquellos productos cuyos componentes requieran menor tiempo de producción en las factorías. Por ello, **se construye π ordenando los productos finales por su $\mathbf{E_s}$, de forma ascendente.**

Finalmente, se obtiene π_T a partir de π y de $\{\pi_s\}_{s\in\mathcal{T}}$ y se secuencia totalmente la producción y ensamblaje haciendo uso de las reglas **NR 1** o **NR 2**. **En función de qué** regla **se use**, se hablará del **heurístico $\mathbf{H_{2,1}}$** o **$\mathbf{H_{2,2}}$**, respectivamente.

Por completitud, a continuación, se reproduce el **algoritmo de este segundo heurístico**:

Algoritmo 3.2 Segundo heurístico, $H_{2,1}$ y $H_{2,2}$

Entrada: Los datos relativos al problema a resolver.
Salida: Una solución para resolver el problema en cuestión.

1: **para** $s \in \mathcal{T}$ **hacer**
2: $\pi_s \leftarrow$ Secuencia parcial de fabricación de los componentes necesarios para ensamblar el producto final s, obtenida usando el algoritmo basado en FL.
3: Calcular E_s usando la regla NR1 (para $H_{2,1}$) o NR2 (para $H_{2,2}$).
4: **fin para**
5: $\pi \leftarrow$ Secuenciación, para la línea de montaje, de los productos finales, obtenida ordenando, de forma ascendente, $\{E_s\}_{s\in\mathcal{T}}$.
6: $\pi_T \leftarrow \cup_{s\in\pi}\pi_s$
7: Asignar la producción de los componentes a las diferentes fábricas del problema usando la regla NR 1 (para $H_{2,1}$) o la regla NR 2 (para $H_{2,2}$).

Para clarificarlo, se resuelve a través de $H_{2,1}$ y $H_{2,2}$ el problema de ejemplo propuesto anteriormente (definido a partir de la tabla 3.1):

1. Usando el **algoritmo** basado en **FL** se obtiene que

$$\pi_1 = \{2, 1, 5, 9\}, \quad \pi_2 = \{8, 6, 4\}, \quad \pi_3 = \{7, 3\}.$$

2. Aplicando la regla NR1 o la regla NR2, los **tiempos mínimos necesarios para producir los componentes de cada uno de los productos finales** son:

$$E_1 = 15, \quad E_2 = 16, \quad E_3 = 12.$$

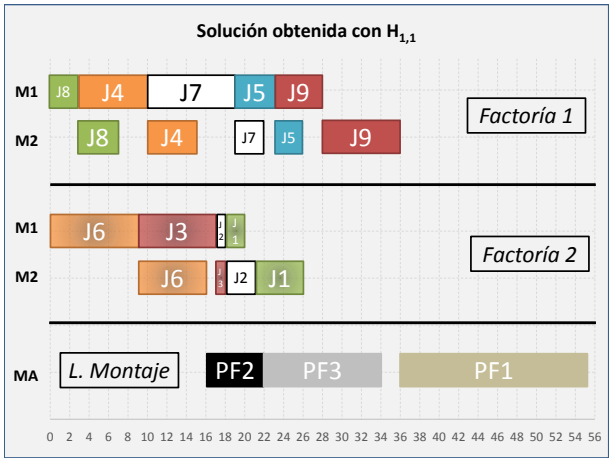
3. **Ordenando los productos finales** de forma ascendente **según el tiempo mínimo necesario para comenzar a ensamblarse**, se determina la secuencia a seguir en la línea de montaje:

$$\pi = \{3, 2, 1\}.$$

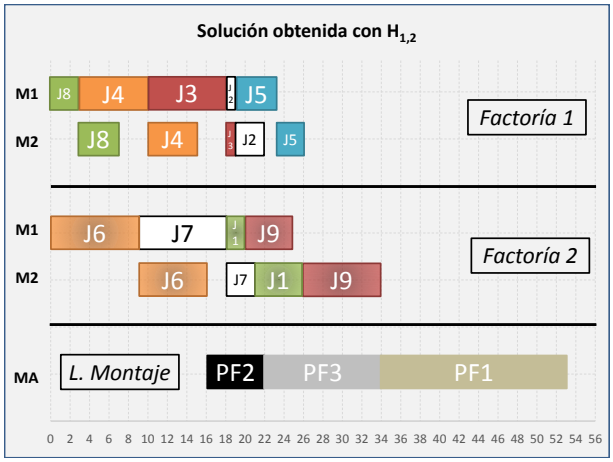
4. **Se construye**, usando π y $\{\pi_s\}_{s\in\mathcal{T}}$, a $\pi_{\mathbf{T}}$:

$$\pi_T = \{7, 3, 2, 1, 5, 9, 8, 6, 4\}.$$

5. **Aplicando la regla NR1 se lleva a la solución dibujada en la figura 3.9-(a)**, con un $C_{\text{máx}} = 51$. Y **aplicando la regla NR2, a la solución de 3.9-(b)**, con un $C_{\text{máx}} = 50$.

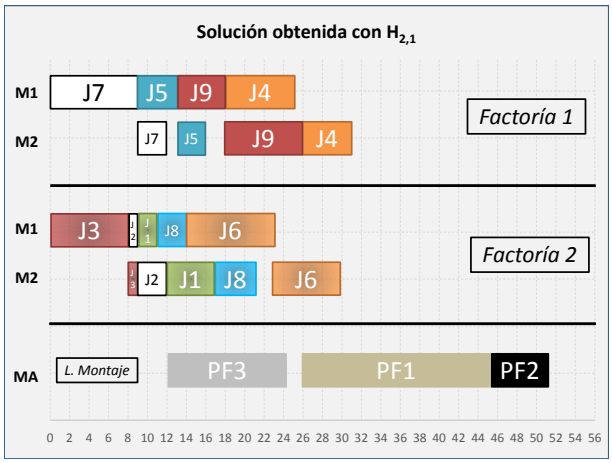


(a) El heurístico $H_{1,1}$ ofrece una solución cuyo $C_{\text{máx}}$ es de 55 unidades temporales.

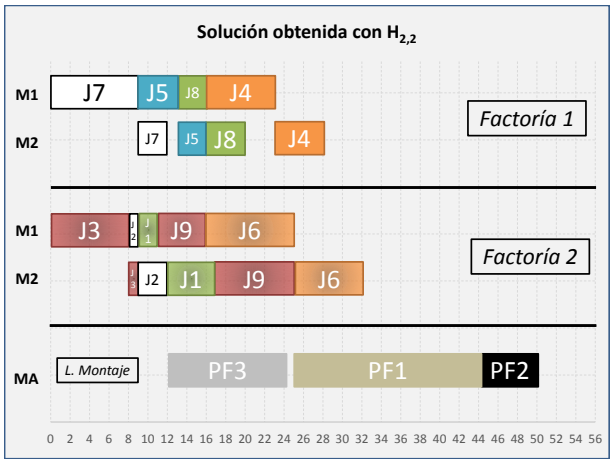


(b) El heurístico $H_{1,2}$ ofrece una solución cuyo $C_{\text{máx}}$ es de 53 unidades temporales.

Figura 3.8: Aplicación de los heurísticos $H_{1,1}$ y $H_{1,2}$ para resolver el problema de ejemplo propuesto. Se representa la solución obtenida en cada caso haciendo uso de los diagramas de Gantt.



(a) El heurístico $H_{2,1}$ ofrece una solución cuyo $C_{\text{máx}}$ es de 51 unidades temporales.



(b) El heurístico $H_{2,2}$ ofrece una solución cuyo $C_{\text{máx}}$ es de 50 unidades temporales.

Figura 3.9: Aplicación de los heurísticos $H_{2,1}$ y $H_{2,2}$ para resolver el problema de ejemplo propuesto. Se representa la solución obtenida en cada caso haciendo uso de los diagramas de Gantt.

Heurístico 3, $H_{3,1}$ y $H_{3,2}$ [17]

Este tercer tipo de heurístico es una variación de los heurísticos $H_{2,1}$ y $H_{2,2}$, que nace con la idea de simplificar y reducir la posible potencia computacional que podrían llegar a requerir éstos. Aunque, eso sí, con la intención de mantener o mejorar la eficacia que muestran los heurísticos originales[VIII].

Así, $H_{3,1}$ y $H_{3,2}$ resultan idénticos a $H_{2,1}$ y $H_{2,2}$, salvo porque varían la forma de construir las secuencias parciales de fabricación de los componentes finales, $\{\pi_s\}_{s\in\mathcal{T}}$. Mientras que $H_{2,1}$ y $H_{2,2}$ emplean el algoritmo derivado del heurístico de Framinan y Leisten (FL), $H_{3,1}$ y $H_{3,2}$ usarán la regla SPT sobre el tiempo necesario para producir cada componente.

Por tanto, **algoritmicamente** este tercer tipo de heurístico construye la programación de la producción y del ensamblaje de la siguiente forma:

Algoritmo 3.3 Segundo heurístico, $H_{3,1}$ y $H_{3,2}$

Entrada: Los datos relativos al problema a resolver.
Salida: Una solución para resolver el problema en cuestión.

1: **para** $s \in \mathcal{T}$ **hacer**
2: $\pi_s \leftarrow$ Secuencia parcial de fabricación de los componentes necesarios para ensamblar el producto final s, obtenida usando la regla SPT sobre los tiempos de completación de cada componente.
3: Calcular E_s usando la regla NR1 (para $H_{3,1}$) o NR2 (para $H_{3,2}$).
4: **fin para**
5: $\pi \leftarrow$ Secuenciación, para la línea de montaje, de los productos finales, obtenida ordenando, de forma ascendente, $\{E_s\}_{s\in\mathcal{T}}$.
6: $\pi_T \leftarrow \cup_{s\in\pi}\pi_s$
7: Asignar la producción de los componentes a las diferentes fábricas del problema usando la regla NR 1 (para $H_{3,1}$) o la regla NR 2 (para $H_{3,2}$).

[VIII] A priori, y con el ejemplo que se analiza en ese documento, parece que la eficacia se mantiene. Pero, ahora bien, el estudio estadístico que realizan los autores en [17] muestra que no, que el RPD medio de este tipo de heurístico es muy superior a los de $H_{2,1}$ y $H_{2,2}$

Para clarificarlo, se resuelve a través de $H_{3,1}$ y $H_{3,2}$ el problema de ejemplo propuesto anteriormente (definido a partir de la tabla 3.1):

1. **Se calculan los tiempos de completación de los componentes**:

Componente	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
$C_{2,\cdot}$	7	4	9	12	7	16	12	7	13

2. Usando la regla SPT sobre los anteriores tiempos de completación, se obtiene que

$$\pi_1 = \{2, 1, 5, 9\}, \quad \pi_2 = \{8, 4, 6\}, \quad \pi_3 = \{3, 7\}.$$

3. Aplicando la regla NR1 o la regla NR2, los **tiempos mínimos necesarios para producir los componentes de cada uno de los productos finales** son:

$$E_1 = 15, \quad E_2 = 19, \quad E_3 = 12.$$

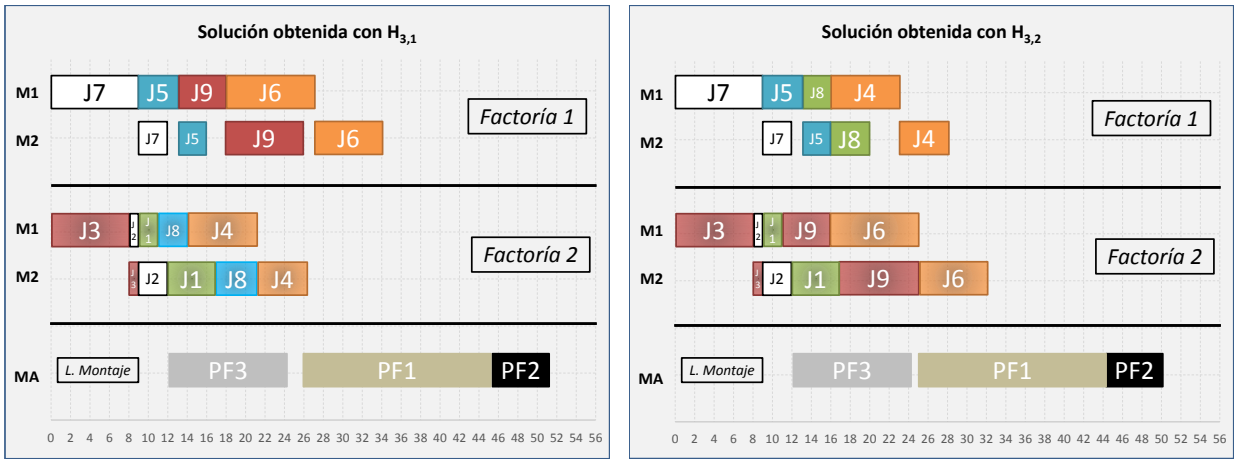
4. **Ordenando los productos finales** de forma ascendente **según el tiempo mínimo necesario para comenzar a ensamblarse**, se determina la secuencia a seguir en la línea de montaje:

$$\pi = \{3, 1, 2\}.$$

5. **Se construye**, usando π y $\{\pi_s\}_{s\in\mathcal{T}}$, a $\pi_{\mathbf{T}}$:

$$\pi_T = \{3, 7, 2, 1, 5, 9, 8, 4, 6\}.$$

6. **Aplicando la regla NR1 se lleva a la solución dibujada en la figura 3.10-(a)**, con un $C_{\text{máx}} = 51$. Y **aplicando la regla NR2, a la solución de 3.10-(b)**, con un $C_{\text{máx}} = 50$.



(a) El heurístico $H_{3,1}$ ofrece una solución cuyo $C_{\text{máx}}$ es de 51 unidades temporales.

(b) El heurístico $H_{3,2}$ ofrece una solución cuyo $C_{\text{máx}}$ es de 50 unidades temporales.

Figura 3.10: Aplicación de los heurísticos $H_{3,1}$ y $H_{3,2}$ para resolver el problema de ejemplo propuesto. Se representa la solución obtenida en cada caso haciendo uso de los diagramas de Gantt.

3.4.2. Estudio de métodos metaheurísticos VND para resolver el problema DAF|prmu|$C_{máx}$

Otra posible vía para generar heurísticos que resuelvan, aceptablemente y en tiempos razonables, el problema de optimización $DAF|prmu|C_{máx}$ es **emplear algoritmos metaheurísticos de búsqueda de vencidad variable** (*Variable Neighbourhood Search,* ***VNS***)[IX].

Entre las diferentes posibilidades, la escasa literatura científica existente únicamente ha desarrollado un algoritmo VNS efectivo y simple: el **algoritmo de búsqueda variable variable descendente, VND**, propuesto por [17]. Por consiguiente, esta subsección analizará y estudiará con detalle este método VND.

Conviene señalar sobre este método, lo siguiente:

- Puesto que los **métodos VND deben partir de una solución posible del problema** y ésta condiciona fuertemente los resultados obtenidos, [17] propone **usar**, para su método VND, **la solución obtenida a través de los heurísticos estudiados anteriormente, en la primera aproximación**. Así, existirán **seis versiones del algoritmo VND propuesto**[X].
- Las **soluciones** que van a ir explorando las diferentes versiones del método VND **se representarán de forma indirecta**, **a partir de $\pi_{\mathbf{T}}$**. Por tanto, hará falta un **proceso de decodificación y cálculo** para formar la secuencia completa de producción y obtener el $C_{máx}$ del problema.

Este proceso de decoficación se basa en las siguientes **limitaciones y reglas**, que se imponen:

- Se limitarán las posibles soluciones estudiadas por las variantes del método VND, estableciendo que **todos los componentes que formen un determinado producto final nunca deben producirse por separado. Es decir, deben aparecer consecutivamente sobre el $\pi_{\mathbf{T}}$ correspondiente**.
- La **línea de montaje** ensamblará los productos finales **según el orden que marque $\pi_{\mathbf{T}}$ al ordenar la producción de los respectivos componentes**.

  → De ahí que se pueda construir $\{\pi_s\}_{s\in\mathcal{T}}$ y π a partir de π_T, por la forma de secuenciar el montaje y de limitar la variabilidad de soluciones.

- **La fabricación de los componentes se asignará a las diferentes factorías usando las reglas NR 1 o NR 2**, según la variación del método VND empleado[XI].

  → De ahí que se pueda **establecer** la **secuenciación completa** de la producción y ensamblaje.

Por tanto, se puede utilizar a π_T como representación de las soluciones estudiadas, ya que resulta factible la decodificación y el **cálculo del $C_{máx}$ del problema**.

[IX] Recuérdese que en la sección 2.5.2 se explicó el concepto de Búsqueda de Vecindad Variable y se analizó las diferentes variantes de ésta, entre las que se incluye la *Variable Neighbourhodd Descent,* ***VND***.

[X] Se denotarán indicando como subíndice el heurístico empleado para generar la solución inicial. Por ejemplo $\text{VND}_{H_{1,1}}$ indica que la solución inicial tomada por el método VND es la que proporciona el heurístico $H_{1,1}$.

[XI] Las variantes $\text{VND}_{H_{\cdot,1}}$ emplearán la regla NR 1. Y las variantes $\text{VND}_{H_{\cdot,2}}$, la regla NR 2

Estudio del método VND para resolver DAF|prmu|$C_{máx}$

El método propuesto por [17] hace uso de dos búsquedas locales:

- **Una** que **trata de mejorar la secuenciación en la línea de montaje** para conseguir reducir el $C_{máx}$ del problema, y que se denotará como $\mathbf{LS}_P$ (*Product Local Search*).
- Y **otra**, que **intenta perfeccionar la secuenciación parcial de fabricación de los componentes que conforman cada producto final** para conseguir una mejor organización productiva que reduzca las esperas sobre la línea de montaje, y que se denotará como $\mathbf{LS}_J$ (*Job Local Search*).

En otras palabras, **estas dos búsquedas locales estudian cambios sistemáticos** y variaciones **sobre**:

- π, que es la secuenciación en la línea de montaje, y
- $\pi_{\mathbf{s}}$ $\forall \mathbf{s} \in \mathcal{T}$, que son las secuenciaciones parciales de fabricación de los componentes que se ensamblan para formar productos finales.

Y, por tanto, **indirectamente**, **van alterando** $\pi_{\mathbf{T}} = \cup_{s \in \pi} \pi_s$.

Se produce, así, una **búsqueda sistemática y en diferentes estructuras de vecindad, de soluciones del problema, hasta obtener un mínimo local**. O sea, aquella solución que, entre las estudiadas, ofrece un mejor $C_{máx}$.

Cada uno de estas dos búsquedas locales se transcriben en los algoritmos 3.4 y 3.5. E irán seguidas de sendas explicaciones de su funcionamiento.

La **estructura general de ejecución del método VND** se detallará **en el algoritmo 3.6**.

Algoritmo 3.4 Búsqueda local LS_P (*Product Local Search*) sobre la secuenciación de los productos finales en la línea de montaje.

Entrada: La secuenciación actual en la línea de montaje, π. Las secuenciaciones parciales de los componentes a producir, $\{\pi_s\}_{s\in\mathcal{T}}$. La regla NR (1 o 2) que usa la versión del algoritmo VND en la que se inserta esta búsqueda local.

Salida: Una secuenciación de línea de montaje, π, que mantiene o mejora el $C_{\text{máx}}$ del problema.

```
1: Mejora ← cierto
2: mientras Mejora hacer
3:   Mejora ← falso
4:   l ← 1
5:   mientras l ≤ t hacer
6:     Se retira de π el producto final a, que ocupa la posición l-ésima en ésta.
7:     Se inserta el producto a en las t − 1 posiciones posibles restantes de π. En cada caso,
       se secuencia la producción y ensamblaje resultantes, y se calcula el nuevo C_máx del
       problema.
8:     si alguna nueva secuenciación obtiene un mejor C_máx para el problema entonces
9:       Se actualiza π con aquella que ofrece un mejor C_máx para el problema
10:      Mejora ← cierto
11:      l ← t+1
12:    si no
13:      l ← l+1
14:    fin si
15:  fin mientras
16: fin mientras
```

La búsqueda local LS_P sobre la línea de montaje se produce **iterativamente**, seleccionando, en cada paso una posición de la secuencia ensamblaje (entre las t disponibles), hasta que haya probado con todas.

En cada iteración, el algoritmo realiza las siguientes acciones:

- Si suponemos que el producto final seleccionado es a y que éste ocupa la posición l-ésima sobre π, la búsqueda local comenzará desasignándolo de esta posición y probará a insertarlo en todas las posibles posiciones restantes de la secuencia de montaje $(t-1)$.
- A continuación, para cada nueva asignación, calculará la secuencia de producción[XII] y ensamblaje asociada. Y, además, se determinará el $C_{\text{máx}}$ de la nueva secuenciación.
- Finalmente, si hay alguna secuenciación de π que ofrezca una mejora sobre el $C_{\text{máx}}$ del problema, la original se actualizará. Y se obligará al algoritmo a reiniciar su ejecución desde el principio, con la nueva π.
- En caso contrario, el algoritmo seguirá con una nueva iteración y seleccionará el producto que ocupa la posición $l+1$-ésima en la secuencia de ensamblaje π.

[XII] Puesto que se conoce $\{\pi_s\}_{s\in\mathcal{T}}$ y el nuevo π, se puede construir π_T y aplicar la regla NR correspondiente para determinar la asignación entre y dentro de las fábricas.

Algoritmo 3.5 Búsqueda local LS_J (*Job Local Search*) sobre la secuenciación en las fábricas de los componentes que conforman cada producto final

Entrada: La secuenciación actual en la línea de montaje, π. Las secuenciaciones parciales de los componentes a producir, $\{\pi_s\}_{s\in\mathcal{T}}$. La regla NR (1 o 2) que usa la versión del algoritmo VND en la que se inserta esta búsqueda local.

Salida: Para cada producto final, una secuenciación parcial de fabricación de los componentes que lo conforman, $\{\pi_s\}_{s\in\mathcal{T}}$, que mantiene o mejora el $C_{\text{máx}}$ del problema.

1: **para** h:=1 **hasta** t **hacer**
2: `Mejora` $\leftarrow$ **cierto**
3: **mientras** `Mejora` **hacer**
4: `Mejora` $\leftarrow$ **falso**
5: $j \leftarrow 1$
6: **mientras** $j \leq |N_h|$ **hacer**
7: Se retira de π_h el componente b, que ocupa la posición j-ésima en ésta.
8: Se inserta el componente b en las $|N_h|-1$ posiciones posibles restantes de π_h. En cada caso, se secuencia la producción y ensamblaje resultantes, y se calcula el nuevo $C_{\text{máx}}$ del problema.
9: **si** alguna nueva secuenciación obtiene un mejor $C_{\text{máx}}$ para el problema **entonces**
10: Se actualiza π_h con aquella que ofrece un mejor $C_{\text{máx}}$ para el problema
11: `Mejora` $\leftarrow$ **cierto**
12: $j \leftarrow N_h + 1$
13: **si no**
14: $j \leftarrow j + 1$
15: **fin si**
16: **fin mientras**
17: **fin mientras**
18: **fin para**

La búsqueda local LS_J sobre la secuenciación en las fábricas de los componentes que conforman cada producto final se produce **iterativamente**. En cada paso, la búsqueda local actúa sobre la secuenciación parcial para cada producto final ($\pi_\cdot$), hasta que haya probado con todos ellos.

En cada paso, el algoritmo **estudia variaciones sistemáticas sobre la secuenciación parcial de los componentes del correspondiente producto**. En concreto, si suponemos que la secuencia estudiada corresponde al producto final h, la búsqueda local estudia individualmente, para cada componente a producir de h, la mejor inserción en la secuencia π_h, con respecto a los restantes componentes.

Para ello, partiendo de un π_h, esta sub-búsqueda comienza seleccionando un componente de la secuencia π_h (hasta haberlo hecho con todos). Si suponemos que el **componente** seleccionado es b y que éste ocupaba la posición j en π_h, **el algoritmo probará a insertarlo en las restantes** $|N_h| - 1$ **posiciones** de π_h.

Para cada secuencia nueva generada de π_h, se calculará la secuencia de producción[XIII] y ensamblaje asociada para **calcular el nuevo** $\mathbf{C_{máx}}$ **del problema**.

Finalmente, **si hay alguna** secuenciación de π_h que ofrezca una **mejora** sobre el $C_{máx}$ del problema, la original **se actualizará**. Y se **obligará** a la sub-búsqueda a reiniciarse y **comenzar de nuevo** con el nuevo π_h.

En caso contrario, la sub-búsqueda seguirá hasta haber probado con los restantes componentes y no haber conseguido mejorar la secuenciación de π_h.

La búsqueda local finaliza cuando se hayan estudiado todas las variaciones sistemáticas sobre π. para todos los productos finales.

Algoritmo 3.6 Método VND. Estructura del método

Entrada: Datos del problema $DAF|prmu|C_{máx}$, que dispone de una única línea de montaje global, en la que ensamblar t productos finales a partir de los n componentes fabricados en las F fábricas con m máquinas disponibles. Versión del algoritmo a ejecutar.

Salida: Una secuencia de producción y ensamblaje aceptable para el problema

1: Se genera una secuencia de producción y ensamblaje inicial con $H_{\cdot,\cdot}$ (según la versión del método VND).

2: `Mejora` ← **cierto**

3: **mientras** `Mejora` **hacer**

4: Ejecutar la búsqueda local LP_P.

5: `Mejora` ← **falso**

6: Ejecutar la búsqueda local LS_J

7: **si** ha habido alguna mejora sobre el $C_{máx}$ del problema **entonces**

8: `Mejora` ← **cierto**

9: **fin si**

10: **fin mientras**

[XIII] Puesto que se conoce π, $\{\pi_s\}_{s\in\mathcal{T}}$ y el nuevo π_h temporal, se puede construir π_T y aplicar la regla NR correspondiente para determinar la asignación entre y dentro de las fábricas.

3.4.3. Rendimiento de los métodos heurísticos analizados

Al inicio de esta sección se comentaba que los métodos heurísticos y metaheurísticos eran capaces de proporcionar soluciones aceptables a problemas de optimización complejos en tiempos muy razonables. Ahora bien, estas soluciones, aunque posibles, no suelen ser óptimas habitualmente.

La calidad de la solución ofrecida depende del diseño del heurístico que se esté empleando. Y, por tanto, resulta imprescindible valorar, en cuestión de calidad, los métodos heurísticos y metaheurísticos analizados en este documento para resolver el problema $DAF|prmu|C_{\text{máx}}$.

Como se afirmó en la subsección 2.5.4, **realizar un estudio propio**, comparativo y formal sobre esta cuestión **está totalmente fuera lugar**, al constituir este documento un trabajo final del grado en matemáticas.

Y, por tanto, se decide **recurrir al estudio estadístico que realiza [18] para valorar los 12 heurísticos analizados**. A saber: $H_{1,1}$, $H_{1,2}$, $H_{2,1}$, $H_{2,2}$, $H_{3,1}$, $H_{3,2}$, $\text{VND}_{H_{1,1}}$, $\text{VND}_{H_{1,2}}$, $\text{VND}_{H_{2,1}}$, $\text{VND}_{H_{2,2}}$, $\text{VND}_{H_{3,1}}$ y $\text{VND}_{H_{3,2}}$.

Como es habitual, el estudio emplea para **valorar la calidad de las soluciones proporcionadas por los diferentes heurísticos**, para una instancia del problema concreta, el **RPD**.

Y estudia el comportamiento de los métodos sobre:

- 900 instancias pequeñas, y
- 810 grandes,

ambas disponibles en `http://soa.iti.es`.

El hecho de hacer esta separación entre ambos tipos de instancias se debe al cálculo del RDP, que tiene en cuenta el mejor valor conocido de la función objetivo para cada instancia. Como las instancias pequeñas resultan computables por el modelo matemático analizado en 3.2.1 (a diferencia de las grandes), es lógico incluir, sobre estas instancias pequeñas y como base de cálculo del RPD, las soluciones obtenidas con el modelo.

Con respecto a la generación de estas instancias, mientras que los tiempos de procesamiento de los componentes y del ensamblaje se han generado aleatoriamente, tomándose uniformemente sobre $[1, 99]$, el resto de datos han sido considerados como factores de estudio. De ahí que la tabla 3.2 recoja, según el tamaño de la instancia de la que se trate, los factores y los respectivos niveles considerados en el estudio estadístico.

Aclarar que, para cada cruce de factores, se han considerado 5 (resp. 10) réplicas para el comportamiento de los heurísticos sobre instancias pequeñas (resp. grandes).

De todo ello, [18] señala que:

- **No existe diferencia estadística significativa** entre el valor RPD obtenido por los **algoritmos VND que usan la misma regla** (NR 1 o NR 2) **para asignar los fabricación de los componentes a las diferentes fábricas**. Es decir, los métodos de la familia $\text{VND}_{H_{\cdot,1}}$ son equivalentes estadísticamente entre ellos. Y los heurísticos $\text{VND}_{H_{\cdot,2}}$ también.
- Existe evidencia estadística suficiente para afirmar que los **métodos $\mathbf{VND_{H_{\cdot,2}}}$ son más efectivos** (proporcionan mejor valor RPD) **que los métodos $\mathbf{VND_{H_{\cdot,1}}}$**.
- Entre los heurísticos simples (de la primera aproximación), existe evidencia estadística suficiente para afirmar que **el rendimiento** que ofrecen los **heurísticos $\mathbf{H_{2,1}}$** y **$\mathbf{H_{2,2}}$** es **superior** al que ofrecen el **resto de heurísticos simples** ($H_{1,1}$, $H_{1,2}$, $H_{3,1}$ y $H_{3,2}$).
- Al **comparar** los heurísticos **$\mathbf{H_{2,1}}$** y **$\mathbf{H_{2,2}}$ y los métodos VND**, existe evidencia estadística para afirmar que los **métodos VND ofrecer un mejor valor RPD**, en comparación con los métodos simples.
- Los métodos **VND son realmente rápidos**: En el peor de los casos, empíricamente, el tiempo máximo empleado para resolver una instancia ha sido menor a un minuto. Y el tiempo medio usado sobre las instancias grandes, de apenas 8.03 segundos. **Aunque, eso sí, bastante más lentos que los métodos simples**, cuyo valor medio es inferior a 0.02 segundos.

Y, por último, concluye que:

- **Los métodos $\mathbf{VND_{H_{\cdot,2}}}$ ofrecen mejores soluciones, entre todos los analizados. Destaca**, entre los métodos de esta familia, el algoritmo **$\mathbf{VND_{H_{\cdot,2}}}$** porque ofrece un coste computacional equilibrado y, empíricamente, obtiene mejores resultados (aunque estadísticamente sean equivalentes).
- Los **métodos heurísticos simples** (primera aproximación) son una **buena opción si** se desea **priorizar** el **consumo de CPU sobre** la **calidad de la solución** ofrecida.

Instancias pequeñas.

Factor del problema	Símbolo	**Número de niveles**	**Valores**
Número de componentes a programar	n	3	8,12,16,20,24
Número de fábricas disponibles	F	3	2,3,4
Número de máquinas	m	4	2,3,4,5
Número de productos finales	t	3	2,3,4

Instancias grandes.

Factor del problema	Símbolo	**Número de niveles**	**Valores**
Número de componentes a programar	n	3	100,200,500
Número de fábricas disponibles	F	3	4,6,8
Número de máquinas	m	3	5,10,20
Número de productos finales	t	3	30,40,50

Tabla 3.2: Factores analizados sobre las diferentes instancias consideradas en el estudio estadístico de [18], con el objetivo de valorar la eficacia de los heurísticos estudiados para resolver el problema $DAF|prmu|C_{\text{máx}}$.

3.5. Otros métodos heurísticos y metaheurísticos para $DAF|prmu|C_{\text{máx}}$

Desde el primer estudio formal que se hizo del problema $DAF|prmu|C_{\text{máx}}$ en el año 2013 por [17], **diversos investigadores han propuesto heurísticos y metaheurísticos más eficaces** que los disponibles en [17] y que los tratados en este documento.

Nuevos métodos que, en todo caso, terminan ofreciendo mejores soluciones al problema, en tiempos razonables y con un uso moderado de los recursos computacionales disponibles.

En concreto, la pertinente búsqueda bibliográfica arroja **tres nuevos métodos** para resolver este problema de optimización. A saber:

- **Un algoritmo híbrido metaheurístico, basado en la biogeografía** (*Hybrid biogeography-based optimization algorithm*), desarrollado en [26].
- **Un algoritmo memético de estimación de la distribución** (*An Estimation of Distribution Algorithm-Based Memetic Algorithm*), explicado en [48].
- **Un algoritmo de vuelta atrás** (*Backtracking search hyper-heuristic*), expuesto en [25].

Ahora bien, **estos métodos resultan extremadamente complejos**. Más de lo que el carácter de este documento permite explayarse para explicarlos detalladamente. Y **el autor se ha conformado con citarlos** en el párrafo anterior, animado por la necesidad de construir una sobria y formal revisión bibliográfica sobre el problema $DAF|prmu|C_{\text{máx}}$.

El lector interesado podrá, así, dirigirse a tales estudios para valorarlos y analizarlos adecuadamente.

Capítulo 4

Conclusiones

4.1. Resumen del trabajo realizado

El presente trabajo ha tenido como **objetivo estudiar el problema de planificación de la producción en un entorno multiplanta, con cargas total o parcialmente distribuidas**, en función del lugar donde se realizara la operación de ensamblaje de los productos finales. Y sobre el que se deseaba minimizar el tiempo máximo de completación de las tareas a realizar ($C_{\text{máx}}$).

Un **problema sumamente interesante**, según la opinión del autor, pero **muy complejo** al tratarse de un problema de optimización combinatoria del que no se conoce ningún algoritmo que garantice su solución óptima en un tiempo polinómico.

Los **modelos** de programación lineal entera estudiados en este trabajo han resultado ser una **poderosa herramienta para resolver instancias pequeñas** de este problema de forma exacta.

Sobre todo

- los modelos 2.3.5 y 2.3.6 , para resolver el problema DPFSP, y
- el modelo 3.2.1, para resolver el problema DAPFSP,

que, según la **evaluación del rendimiento realizada** en las secciones 2.4 y 3.3, resultan altamente eficaces en su cometido sobre instancias relativamente pequeñas.

Ahora bien, como era esperable, estos modelos no serán capaces de encontrar soluciones óptimas en tiempos razonables sobre instancias grandes.

De ahí la necesidad de incluir en este trabajo métodos heurísticos y metaheurísticos para resolver los problemas DPFSP y DAPFSP, que proporcionarán **buenos resultados en tiempos muy adecuados** sin garantizar la optimalidad en la solución obtenida.

La sección 2.5 se ocupará de analizar métodos heurísticos y metaheurísticos eficaces para el problema DPFSP y la sección 3.4, por su parte, estudiará los métodos heurísticos más instructivos, aplicados al problema DAPFSP.

Las secciones 2.6 y 3.5 se usarán para mostrar otros métodos heurísticos que, por su complejidad o bajo rendimiento, no han sido incluidos como parte del cuerpo de este documento.

4.2. Trabajo futuro

Aunque **este trabajo ha abarcado una gran variedad de herramientas y modelos para planificar la producción en entornos multifábrica**, lo cierto es que todavía éstas ofrecen **ciertas limitaciones y no son fieles totalmente a la realidad**.

Y es que **los problemas DPFSP y DAPFSP constituyen una primera aproximación** y una modelización bastante simplificada del ambiente empresarial y de los sistemas de producción actuales. Y pueden resultar no ser lo suficientemente prácticos al necesitar incluir, todavía, mayores restricciones y nuevos elementos a la modelización.

Sin ir más lejos, **el problema DAPFSP no tiene siquiera en cuenta los tiempos de transporte entre las fábricas productoras y la línea de ensamblaje**. O las **posibles limitaciones en el transporte** que puedan existir, relacionadas con la capacidad de transporte o a los cupos mínimos.

También ignora la posible variabilidad en la línea de montaje en la etapa de ensamblaje de los productos. Y es que analiza únicamente la integración de la producción con **una línea de tipo *Mixed Model Assembly form Make-to-Order*, ignorando el resto de posibles configuraciones** que ofrecen otros tipos de línea de montaje.

Tampoco es **realista** considerar que el **buffer intermedio**

- entre máquinas (en las fábricas),
- entre etapas (en la red de transporte, desde la fábrica hasta la línea de montaje), o
- entre estaciones de montaje (en la línea de ensamble)

tiene **infinita capacidad** para almacenar los componentes o productos que se van produciendo.

O que las **máquinas siempre deban estar disponibles para su utilización**, a pesar de las posibles averías que puedan sufrir, del tiempo dedicado al mantenimiento, o del tiempo necesario para restablecerlas o limpiarlas después de realizar alguna tarea de las programadas (set up dependiente de la secuencia obtenida).

Y, aunque tradicionalmente se ha modelizado la producción distribuida pensando que las **fábricas usadas** eran virtualmente idénticas, la realidad **suele ser diferente**:

- Es habitual que las **máquinas** de las que dispone **cada fábrica sean distintas**, aunque realicen la misma tarea, porque las fábricas no se suelen construir a la vez y el equipo industrial se va renovando para incorporar las últimas novedades.
- Y es bastante común que los tiempos de ejecución de un cierto trabajo difieran entre sí, sobre todo si en la ejecución de las tareas intervienen operarios, creándose así una cierta **incertidumbre en los tiempos de producción**.

Ahora bien, esto no desmerece este trabajo. **Aunque los modelos o los heurísticos no sean totalmente fieles a la realidad, siempre proporcionan una muy buena orientación sobre la planificación final de la producción**. Y se deben entender así, como meras herramientas de planificación.

4.3. Aplicaciones de este trabajo

Los **sistemas de producción** actuales aparecen inmersos en un **entorno global y complejo**, sobre el que existe una **competencia voraz** y en el que los clientes cada vez más exigen menores plazos de entrega y un nivel cada vez mayor de personalizaciones en los productos que adquieren.

Una de las posibles aplicaciones que tendría este trabajo es la de **facilitar** a estos actores la **planificación de su producción**, al proporcionarles **herramientas matemáticas para optimizar las tareas de fabricación y ensamblaje que realizan**.

Aunque no la única porque el **autor** también **espera haber aportado su saber sobre estas metodologías** y haber facilitado una mejor **transmisión de conocimientos entre el mundo académico y el empresarial**. O incluso **poder llegar a servir de inspiración** en futuros trabajos de investigación o desarrollo.

Anexos

Anexo A

Métodos heurísticos para resolver el problema $\mathbf{F|prmu|C_{máx}}$

Las investigaciones llevadas a cabo durante la segunda mitad del siglo XX produjeron grandes avances en la resolución, mediante heurísticos, de los problemas de planificación de la producción $\mathbf{F|prmu|C_{máx}}$. Estos heurísticos permitieron, en tiempos razonables y computacionalmente aceptables, programar la ejecución de los trabajos.

Se desarrollaron, principalmente, **6 heurísticos eficaces**:

a) La **regla SPT** (*Shortest Processing Time*), [1].

b) La **regla LPT** (*Largest Processing Time*), [1].

c) La **regla de Johnson**, cuando se dispone de, únicamente, dos máquinas, [22].

d) La **regla de Palmer**, [37].

e) La **regla CDS** (*Campbell-Dudek-Smith*), [5].

f) El **algoritmo NEH** (*Nawaz-Enscore-Ham*), [36].

Este anexo describe cada uno de ellos mientras que, paralelamente, va desarrollando la producción para el siguiente ejemplo.

Supongamos que se desea programar la ejecución de $n = 5$ trabajos sobre $m = 2$ máquinas. Los tiempos de procesamiento usados para el ejemplo se detallan en la siguiente tabla:

	Trabajo 1 (*J1*)	Trabajo 2 (*J2*)	Trabajo 3 (*J3*)	Trabajo 4 (*J4*)	Trabajo 5 (*J5*)
Máquina 1 (*M1*)	10	6	8	9	3
Máquina 2 (*M2*)	5	7	4	6	11

La regla SPT

La **regla SPT**(*Shortest Processing Time*), junto con la regla LPT, es una de las reglas más sencillas para programar, heurísticamente, la ejecución de los trabajos. La regla establece que los trabajos se secuencian por orden creciente de sus tiempos de procesamiento. Esto es, **primero se ejecutan aquellos trabajos con un tiempo de procesamiento total menor**. En caso que exista trabajos con el mismo tiempo de procesamiento total, da igual cual se procese antes, a efectos de la regla.

Calculando los tiempos de procesamiento totales requeridos para ejecutar la producción del ejemplo, se obtiene

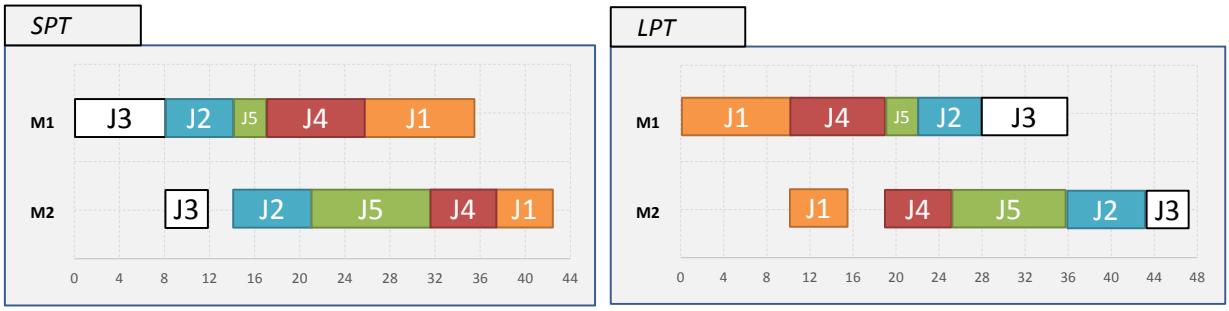
	J1	*J2*	*J3*	*J4*	*J5*
Tiempo total de procesamiento	15	13	12	15	14

La secuencia de producción se representa en la figura A.1a.

La regla LPT

La **regla LPT**(*Largest Processing Time*) establece que los trabajos se secuencian por orden decreciente de sus tiempos de procesamiento. Esto es, **primero se ejecutan aquellos trabajos con un tiempo de procesamiento total mayor**. En caso que exista trabajos con el mismo tiempo de procesamiento total, da igual cual se procese antes, a efectos de la regla.

Empleando los tiempos de procesamiento totales calculados para la regla SPT, la secuencia de producción resultante se representa en la figura A.1b.



(a) Aplicación de la regla SPT (b) Aplicación de la regla LPT

Figura A.1: Se representa en los respectivos diagramas de Gannt la **programación de la producción** propuesta como **ejemplo** de la aplicación de los métodos heurísticos para resolver el problema $\mathbf{F|prmu|C_{máx}}$ de ejemplo.

La regla de Johnson

La **regla de Johnson** que se presenta resulta aplicable cuando se disponen de dos máquinas o sobre algunos casos particulares con tres máquinas (que se omiten por su complejidad técnica). Cuando se disponen de únicamente dos máquinas, es decir, para $\mathbf{F2|prmu|C_{máx}}$, el resultado que proporciona la regla es el óptimo.

La regla comienza dividiendo los trabajos en dos conjuntos:

- En el conjunto I aparecen los trabajos con $P_{1,j} \leq P_{2,j}$
- Y en el conjunto II, aquellos con $P_{1,j} > P_{2,j}$

Entonces, la regla establece que

- **Primero se secuencian los trabajos del conjunto I usando la regla SPT sobre** los tiempos de procesamiento en la primera máquina, $\mathbf{P_{1,j}}$
- Y **después se secuencian los del conjunto II usando la regla LPT sobre** los tiempos de procesamiento en la segunda máquina, $\mathbf{P_{2,j}}$

Para el ejemplo que se está desarrollando, los conjuntos resultantes son:

$$\text{Conjunto I: } \{J2, J5\} \qquad \text{Conjunto II: } \{J1, J3, J4\}$$

La secuencia de producción resultante se representa en la figura A.2a.

La regla de Palmer

La **regla de Palmer** propone organizar la producción mediante un índice de máximo orden, basado en los tiempos de proceso. La idea es dar prioridad a los trabajos cuyos tiempos de proceso, en cada máquina, se van incrementando. Los trabajos cuyo tiempo decrece máquina a máquina reciben una baja prioridad.

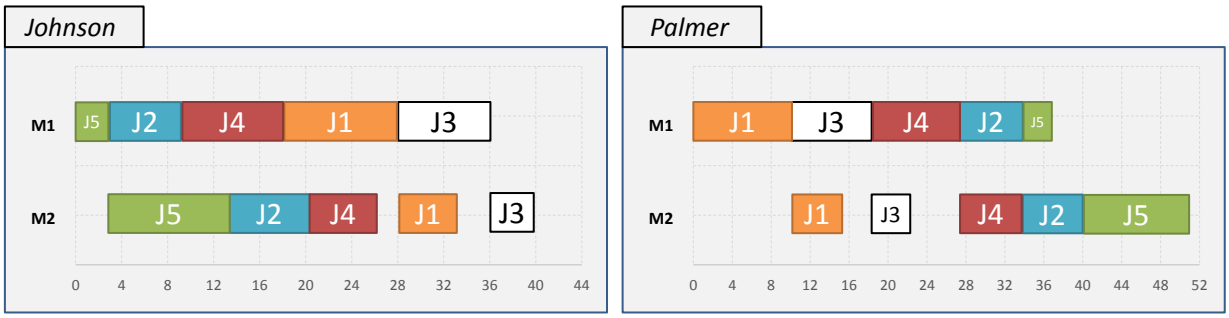
La regla programa, de este modo, los trabajos en orden decreciente según el valor del índice. Este índice se calcula para cada trabajo j, de la siguiente forma

$$S_j = \sum_{i=1}^{m} \Big(m - 2i + 1\Big) P_{i,j}$$

Para el ejemplo que se está desarrollando, los índices resultantes se expresan en la siguiente tabla

	J1	*J2*	*J3*	*J4*	*J5*
Índice de procesamiento	5	-1	4	3	-8

La secuencia de producción resultante se representa en la figura A.2b.



(a) Aplicación de la regla de Johnson (b) Aplicación de la regla de Palmer

Figura A.2: Se representa en los respectivos diagramas de Gannt la **programación de la producción** propuesta como **ejemplo** de la aplicación de los métodos heurísticos para resolver el problema **F|prmu|C**$_{\text{máx}}$ de ejemplo.

La regla CDS

La **regla CDS** (*Campbell-Dudek-Smith*) **se basa en** la **aplicación de la regla de Jonhson**, iterativamente, $m-1$ veces. Para ello, **en cada iteración**, controlada por el contador $k \in \{1,\ldots,m-1\}$, **el problema que se resuelve es el formado por dos máquinas ficticias**. Los tiempos de procesamiento de estas dos máquinas, $T_{\cdot,i,j}$, se calculan mediante estas expresiones:

$$\begin{cases} T_{1,j,k} = \sum_{i=1}^{k} P_{i,j} \\ T_{2,j,k} = \sum_{i=m-k+1}^{m} P_{i,j} \end{cases}$$

Una vez resueltos los m − 1 problemas, **la regla CDS selecciona aquel cuyo C**$_{\text{máx}}$ **es el menor** como solución al problema original con m máquinas.

Como el **ejemplo** propuesto anteriormente no resulta interesante para esta regla, para ejemplificar la regla se propone el siguiente problema, con 3 máquinas en lugar de dos.

Supongamos que se desea programar la ejecución de $n = 5$ trabajos sobre $m = 3$ máquinas. Los tiempos de procesamiento usados para el ejemplo se detallan en la siguiente tabla:

	Trabajo 1 (*J1*)	Trabajo 2 (*J2*)	Trabajo 3 (*J3*)	Trabajo 4 (*J4*)	Trabajo 5 (*J5*)
Máquina 1 (*M1*)	4	3	6	0	3
Máquina 2 (*M2*)	7	0	3	1	1
Máquina 3 (*M3*)	2	5	1	4	2

En la **primera iteración** de la regla, las dos máquinas ficticias que componen el problema que se debe resolver, usando la regla de Johnson, tienen estos tiempos de procesamiento

	J1	*J2*	*J3*	*J4*	*J5*
T1	4	3	6	0	3
T2	2	5	1	4	2

La regla de Johnson proporciona la siguiente secuencia de procesamiento con $C_{\text{máx}} = 20$

$$\{J4, J2, J1, J5, J3\}.$$

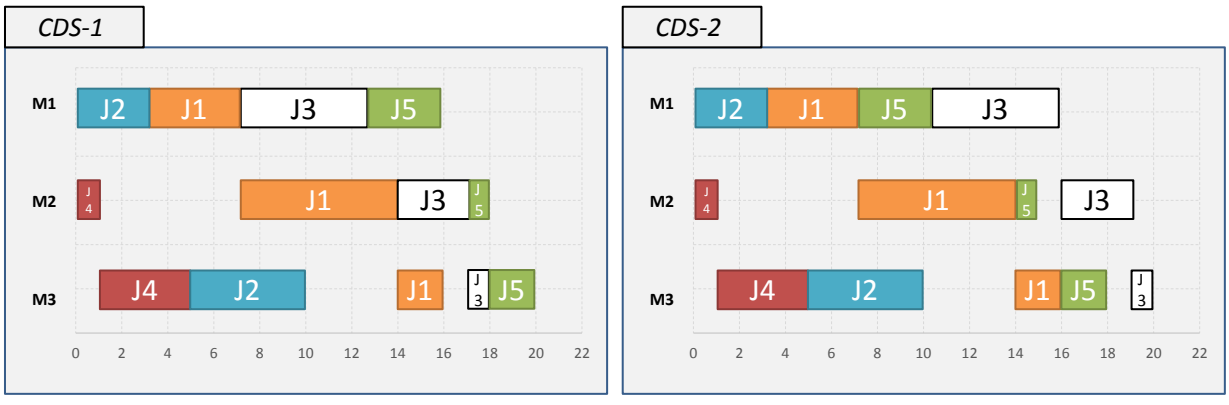
En la **segunda iteración** de la regla, las dos máquinas ficticias que componen el problema que se debe resolver, usando la regla de Johnson, tienen estos tiempos de procesamiento

	J1	*J2*	*J3*	*J4*	*J5*
T1	11	3	9	1	9
T2	9	5	4	5	3

La regla de Johnson proporciona la siguiente secuencia de procesamiento con $C_{\text{máx}} = 20$

$$\{J4, J2, J1, J3, J5\}.$$

Como ambas iteraciones han proporcionado sendas secuencias con el mismo $\mathbf{C}_{\text{máx}}$, se selecciona arbitrariamente una de las dos como solución al ejemplo propuesto. Las secuencias generadas en cada iteración se representa en las figuras A.3a y A.3b.



(a) Aplicación de la regla CDS. Primera iteración. (b) Aplicación de la regla CDS. Segunda iteración.

Figura A.3: Se representa en los respectivos diagramas de Gannt la **programación de la producción** propuesta como **ejemplo** de la aplicación de la regla CDS $\mathbf{F}|\mathbf{prmu}|\mathbf{C}_{\text{máx}}$ de ejemplo.

El algoritmo NEH

El **algoritmo NEH** (*Nawaz-Enscore-Ham*) es un método heurístico diseñado en 1983 por *Nawaz*, *Enscore* y *Ham*, que toma ciertas ideas de *Branch & Bound*.

El algoritmo comienza calculando el tiempo de procesamiento total, para cada trabajo. Este cálculo se usará para ordenador los trabajos de mayor a menor tiempo, creándose así una **lista de trabajos ordenada**.

- En primer lugar, se seleccionan los **dos primeros trabajos de la lista** (j_1, j_2) y se forman las **dos posibles secuencias que los contienen**

  $$\{j_1, j_2\} \qquad \text{ó} \qquad \{j_2, j_1\}$$

  Entonces, **se elige aquella que tenga un menor** $\mathbf{C}_{\text{máx}}$.

Esta secuencia, provisional, se irá mejorando en cada iteración del algoritmo.

- A continuación, y hasta finalizar con todos los elementos del listado, **se selecciona el siguiente trabajo** (que ocupa la posición $k \in \{3, \ldots, n\}$). Este trabajo **se inserta en todas las** k **posibles posiciones de la secuencia provisional del paso anterior**. Entre todas ellas, **se elige aquella secuencia que tenga un menor** $\mathbf{C}_{\text{máx}}$. **Y se repite el proceso** con el siguiente elemento de la lista.

Para calcular la secuencia de producción de ejemplo propuesta por este algoritmo, se emplean los tiempos de procesamiento totales calculados para la regla SPT. La lista resultante es

$$L := \{J4, J1, J5, J2, J3\}$$

- La **primera iteración** del algoritmo evalúa las secuencias $\{J4, J1\}$, con un $C_{\text{máx}} = 24$, y $\{J1, J4\}$, con un $C_{\text{máx}} = 23$. Selecciona la secuencia $\{J4, J1\}$.

- La **segunda iteración** evalúa:
  - $\{J4, J1, J5\}$, con $C_{\text{máx}} = 35$,
  - $\{J4, J5, J1\}$, con $C_{\text{máx}} = 31$, y
  - $\{J5, J4, J1\}$, con $C_{\text{máx}} = 27$.

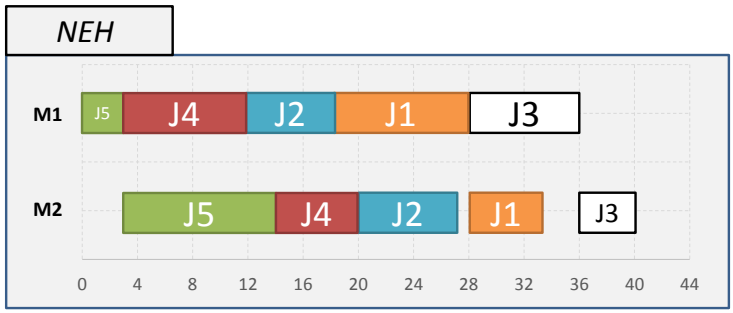
  Selecciona la secuencia $\{J5, J4, J1\}$.

- La **tercera iteración** evalúa:
  - $\{J5, J4, J1, J2\}$, con $C_{\text{máx}} = 35$,
  - $\{J5, J4, J2, J1\}$, con $C_{\text{máx}} = 33$,
  - $\{J5, J2, J4, J1\}$, con $C_{\text{máx}} = 33$, y
  - $\{J2, J5, J4, J1\}$, con $C_{\text{máx}} = 35$.

  Selecciona la secuencia $\{J5, J4, J2, J1\}$.

- La **última iteración** evalúa las secuencias:
  - $\{J5, J4, J2, J1, J3\}$, con $C_{\text{máx}} = 40$,
  - $\{J5, J4, J2, J3, J1\}$, con $C_{\text{máx}} = 41$,
  - $\{J5, J4, J3, J2, J1\}$, con $C_{\text{máx}} = 41$,
  - $\{J5, J3, J4, J2, J1\}$, con $C_{\text{máx}} = 41$, y
  - $\{J3, J5, J4, J2, J1\}$, con $C_{\text{máx}} = 41$.

  Selecciona la secuencia $\{J5, J4, J2, J1, J3\}$, que se representa en A.4a



(a) Resultado de la aplicación del algoritmo NEH.

Figura A.4: Se representa la **programación de la producción** propuesta como **ejemplo** de la aplicación del respectivo método heurístico para resolver el problema $\mathbf{F}|\mathbf{prmu}|\mathbf{C}_{\text{máx}}$ de ejemplo.

Anexo B

Fundamentos y bases de los algoritmos de búsqueda dispersa

Un algoritmo de **búsqueda dispersa** (*Scatter Search*, **SS**) es un **método metaheurístico evolutivo** (*Population-Based Metaheuristics*), originalmente introducido en los años setenta por [13] y ampliamente extendido durante las dos últimas décadas para resolver problemas difíciles. Su éxito para resolverlos se debe, principalmente, a las técnicas que emplea para generar, recombinar y diversificar las soluciones de las que va disponiendo en cada momento. Estas soluciones temporales aparecen en el llamado *conjunto de referencia* del método (*RefSet*).

Tal y como se expone en [24] y en [32], los **algoritmos de búsqueda dispersa constan de cinco** elementos o **subalgoritmos principales que los vertebran**. A saber:

1. Un **generador de soluciones variadas** y diversas (*Diversification Generation Method*).

   La base del método es ir generando, en cada iteración, un conjunto de soluciones diversas y variadas del problema, de las que extraerá, finalmente b de ellas, para incorporarlas al conjunto de referencia. El método empleado para generar soluciones caracterizará el respectivo algoritmo de búsqueda dispersa.

2. Un **método de mejora de las soluciones**, tanto de las del conjunto de referencia como de las obtenidas mediante el proceso de combinación (*Improvement Method*).

   Aunque existen otros métodos, normalmente, los diferentes algoritmos de búsqueda dispersa hacen **uso de búsquedas locales para mejorar las soluciones**. En caso de que no lo consiga, la solución inicial se toma como mejorada y se sigue la ejecución del algoritmo.

   Aclarar que, en caso que el método parta o vaya obteniendo (mediante el método de diversificación comentado antes) soluciones no factibles, se debe incluir, en el método de mejora, un paso previo para conseguir que estas soluciones sean factibles para el problema que se estudia.

3. Un **método para crear y actualizar *RefSet*** (*Reference Set Update Method*).

   Habitualmente, el algoritmo propone procedimientos diferentes para crear o actualizar el conjunto de referencia. Pero, como regla general, el tamaño del conjunto de referencia permanecerá inalterado durante la ejecución del método. Y, por tanto, siempre mantendrá el tamaño inicial, b.

   Aunque existen algoritmos de búsqueda más avanzados y que emplean técnicas más sofisticadas, un procedimiento estándar para crear el conjunto de referencia, con b soluciones, consiste en generar un número grande, alrededor de 100, soluciones diversas (mediante el procedimiento *Diversification Generation Method*). De ellas, se terminan seleccionando para conformar el conjunto de referencia:

   - aquellas $b/2$ soluciones con mejor valor para la función objetivo, y
   - aquellas $b/2$ soluciones que *disten*[1] más de las anteriormente seleccionadas.

   Debido a que los procedimientos de combinación y mejora son capaces de proporcionar mejores soluciones de las que ya se dispone, es necesario establecer un criterio para ir actualizando el conjunto de referencia. En las implementaciones más rudimentarias se obvia el criterio de diversidad y únicamente se valora la calidad de las incorporaciones. Es por ello que una solución sustituye a la peor solución del conjunto de referencia si mejora el valor de la función objetivo del problema.

4. Un **método de combinación de soluciones** (*Solution Combination Method*).

   Puesto que el método emplea, como principal estrategia, la combinación de soluciones, se debe establecer, en primer lugar, que soluciones se combinan. En implantaciones sencillas, la combinación ocurrirá entre todas las posibles parejas de soluciones del conjunto de referencia que se puedan formar.

5. Un **método que seleccione grupos de soluciones para ser combinadas** por el anterior método (*Subset Generation Method*).

   Una vez establecido el criterio para seleccionar soluciones del conjunto de referencia a combinar, deben combinarse con un método apropiado. Este método definirá, principalmente, al algoritmo de búsqueda dispersa y marcará su efectividad.

   Las soluciones combinadas, finalmente, pueden valorarse directamente para ser incluidas en el conjunto de referencia. O mantenerse en espera hasta haber concluido el proceso de combinación y, entonces, valorar la inclusión de las x mejores.

En el algoritmo B.1 se muestra un esquema básico del método de búsqueda dispersa. Además, se incluye la imagen B.1 que ilustra al algoritmo.

[1] Se hará necesario definir previamente una función distancia en el problema.

Algoritmo B.1 Método de búsqueda dispersa

1: Generación inicial de soluciones para el problema (*Diversification Generation Method*)
2: Aplicar el método de mejora sobre las soluciones generadas (*Improvement Method*)
3: Seleccionar b soluciones de ellas, para formar el conjunto de referencia (*Reference Set Update Method*)
4: `Mejora` ← **cierto**
5: **mientras** `Mejora` **hacer**
6: `Mejora` ← **falso**
7: Selección de las soluciones a combinar (sólo aquellas combinaciones no explotadas aún)
8: **mientras** queden soluciones a combinar **hacer**
9: Aplicar el método de combinación sobre la selección previa (*Solution Combination Method*)
10: Aplicar el método de mejora sobre la combinación obtenida (*Improvement Method*)
11: Valorar si la solución obtenida entra a formar parte del conjunto de referencia del método (sustituyendo a otra incluida en el conjunto) (*Reference Set Update Method*)
12: **si** la solución entra en *RefSet* **entonces**
13: `Mejora` ← **cierto**
14: **fin si**
15: **fin mientras**
16: **fin mientras**

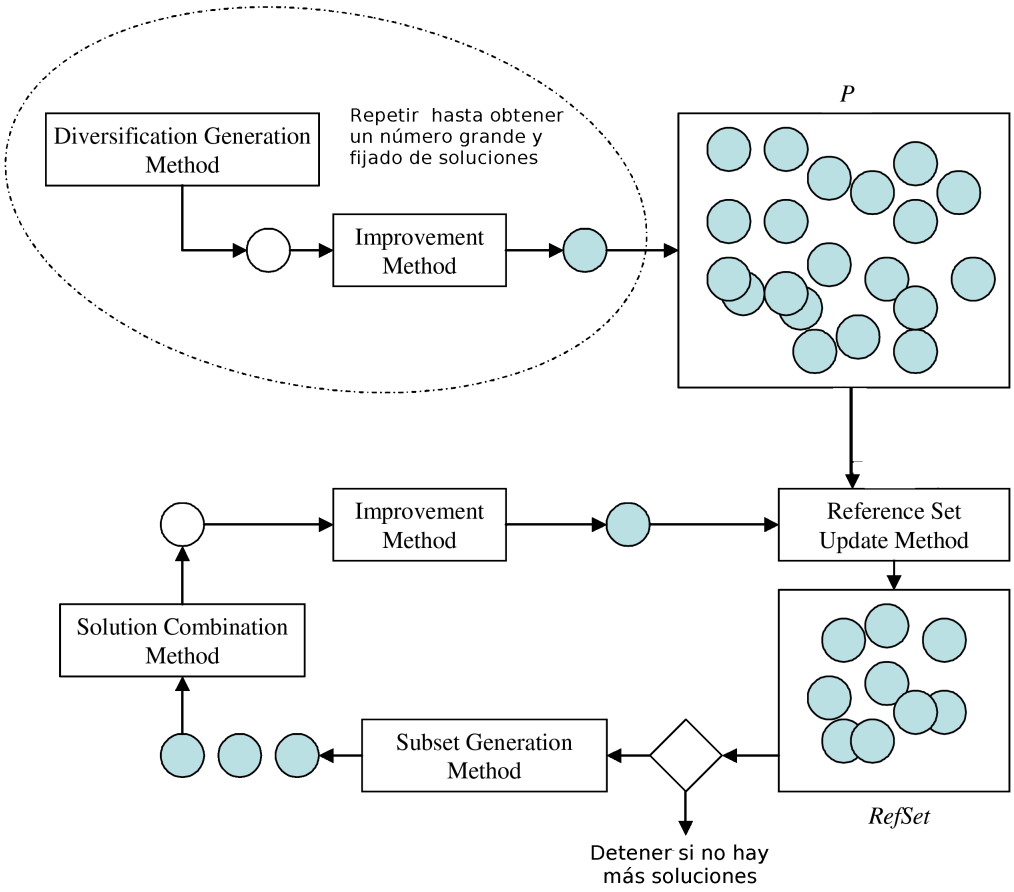


Figura B.1: Imagen adaptada de [24]. Esquema del método de búsqueda dispersa.

Anexo C

Implementación en GNU MathProg de los modelos matemáticos analizados para resolver el problema $\mathbf{DF|prmu|C_{máx}}$ y $\mathbf{DAF|prmu|C_{máx}}$

Por completitud, se ha decidido incluir en este documento PDF la **implementación de los modelos analizados**

- en la sección 2.3 para **resolver** el problema $\mathbf{DF|prmu|C_{máx}}$, y
- en la sección 3.2 para **resolver** el problema $\mathbf{DAF|prmu|C_{máx}}$.

Esta implementación ha tenido lugar sobre **GNU MathProg**, un lenguaje de alto nivel para crear modelos de programación matemática y específico de GLPK (*GNU Linear Programming Kit*) [51].

GLPK, por su lado, es un kit diseñado para resolver problemas de programación lineal a gran escala (*Linear Programming,* ***LP***) programación entera mixta (*Mixed Integer Programming,* ***MIP***), y otros problemas relacionados [52].

Se ha decidido incluir en el código de este documento datos de ejemplo para cada modelo.

```
# Capítulo 2 – The Distributed Permutation Flowshop Scheduling Problem
# Estudio de modelos para el problema DF|prmu|Cmax
# Modelo 1: Basado en la secuenciación

# Definición de parámetros y conjuntos

  param num_trabajos;
  param num_maquinas;
  param num_fabricas;

  set trabajos:=1..num_trabajos;    #:=J
  set maquinas:=1..num_maquinas;    #:=M
  set fabricas:=1..num_fabricas;    #:=F

  param P{trabajos, maquinas};    # Tiempo de procesamiento P_{ji}

  param M;          #Constante suficientemente grande

# Definición de variables
  var X{k in trabajos, j in trabajos, f in fabricas : (k>j or k<j)} binary;
      # 1 si el trabajo j se procesa en la factoría f inmediatamente después del ...
        trabajo k

  var Xdummy{j in trabajos, f in fabricas} binary;
      # 1 si el trabajo j se procesa en primer lugar en la factoría f
      # Es el trabajo que seguiría tras el dummy job.

  var Y{j in trabajos, f in fabricas} binary;
      # 1 si el trabajo j se procesa en la factoría f

  var C{j in trabajos, i in maquinas} ≥0;
      # Tiempo de completación del trabajo j en la máquina i

  var Cmax ≥0;
      # Tiempo máximo de completación de todas las tareas, entre todas las fábricas
      # Maxium makespan among all factories

# Función objetivo y restricciones
  minimize tiempo_maximo_completacion:
    Cmax;

  # Cada trabajo debe estar asignado exactamente a una fábrica
  subject to UnTrabajoUnaFabrica{j in trabajos}:
    sum{f in fabricas} Y[j,f] = 1;

  # Cada trabajo debe aparecer en una única posición de la secuencia de ...
    producción, asociada a la fábrica donde se realiza
  subject to UnTrabajoUnaPosicion{j in trabajos}:
    sum{f in fabricas} ( sum{k in trabajos : (j>k or j<k)}  X[k,j,f] + ...
      Xdummy[j,f] ) = 1;

  # Si un trabajo –real– no está asociado a una fábrica, no puede estar en su ...
    secuencia de producción
  subject to RelacionXeY{j in trabajos, f in fabricas}:
    sum{k in trabajos : j>k or j<k} ( X[k,j,f] + X[j,k,f] ) ≤ 2 * Y[j,f];

  # Definición de X, con respecto a la relación de precedencia inmediata.
  subject to defX{k in trabajos}:
    sum{j in trabajos: j>k or j<k} sum{f in fabricas} X[k,j,f] ≤ 1;

```

```
57
58
59    # Cada dummy-job tiene un suscesor en la secuencia de cada fábrica.
60    subject to defXdummy{f in fabricas}:
61      sum{j in trabajos} Xdummy[j,f] = 1;
62
63    # No se puede preceder a un trabajo, y luego seguirlo, a la vez.
64    subject to defprecedencia{k in trabajos, j in trabajos : (k<num_trabajos and ...
          j>k)}:
65      sum{f in fabricas} ( X[k,j,f] + X[j,k,f] ) ≤ 1;
66
67
68    # Definición de completacion
69    subject to defcompletacion0{j in trabajos}:
70      C[j,1] ≥ 0 + P[j,1];
71
72    subject to defcompletacion1{j in trabajos, i in maquinas : i > 1}:
73      C[j,i] ≥ C[j, i-1] + P[j,i];
74
75    subject to defcompletacion2{k in trabajos, j in trabajos, i in maquinas : (j>k ...
          or j<k)}:
76      C[j,i] ≥ C[k,i] + P[j,i] + M * ( (sum{f in fabricas} X[k,j,f] ) - 1);
77
78
79    # Definición de Cmax
80    subject to defCmax{j in trabajos}:
81      Cmax ≥ C[j,num_maquinas];
82
83
84
85
86  data;
87    param num_trabajos:=10;
88    param num_maquinas:=2;
89    param num_fabricas:=10;
90    param M:=98;
91    param P:
92          1   2:=
93    1     5   3
94    2     7   4
95    3     3   5
96    4     2   7
97    5     2   9
98    6     1   10
99    7     7   11
100   8     1   6
101   9     3   1
102   10    3   8;
```

```
# Capítulo 2 — The Distributed Permutation Flowshop Scheduling Problem
# Estudio de modelos para el problema DF|prmu|Cmax
# Modelo 2: Basado en las posiciones de Wagner

# Definición de parámetros y conjuntos

  param num_trabajos;
  param num_maquinas;
  param num_fabricas;

  set trabajos:=1..num_trabajos;   #:=J
  set maquinas:=1..num_maquinas;   #:=M
  set fabricas:=1..num_fabricas;   #:=F

  param P{trabajos, maquinas};   # Tiempo de procesamiento P_{ji}

# Definición de variables
  var X{j in trabajos, k in trabajos, f in fabricas} binary;
      # 1 si el trabajo j ocupa la posición k en la factoría f

  var I{i in maquinas, k in trabajos, f in fabricas : k < num_trabajos} ≥0;
      # Tiempo que la máquina i permanece desocupada tras finalizar el trabajo ...
          que ocupa la posición k en la factoría, hasta que empieza el ...
          siguiente, que ocupa la posición k+1
      # Idle time

  var W{i in maquinas, k in trabajos, f in fabricas : i < num_maquinas} ≥0;
      # Tiempo que transcurre desde que el trabajo que ocupa la posición k en la ...
          factoría f sale de la máquina i y entra en la máquina i+1
      # Waiting time

  var C{f in fabricas} ≥0;
      # Tiempo de completación de todas las tareas en la fábrica f

  var Cmax ≥0;
      # Tiempo máximo de completación de todas las tareas, entre todas las fábricas
      # Maxium makespan among all factories

# Función objetivo y restricciones
  minimize tiempo_maximo_completacion:
    Cmax;

  # Cada trabajo tiene asignada una factoría en la que realizarse y una posición ...
      dentro del sistema de producción global
  subject to UnTrabajoUnaFabrica{j in trabajos}:
    sum{k in trabajos} sum{f in fabricas} X[j,k,f] = 1;

  # Toda posición del sistema de producción global está ocupada por un trabajo y ...
      asignada a una factoría
  subject to UnaPosicionUnTrabajo{k in trabajos}:
    sum{j in trabajos} sum{f in fabricas}  X[j,k,f] = 1;

  # Relación entre variables X, I, W
  subject to RelacionXIW_i1k1f{k in trabajos, i in maquinas, f in fabricas : ( ...
      k<num_trabajos and i<num_maquinas and k=1 and i=1 )}:
    0 — 0 + sum{j in trabajos} X[j,k+1,f]*P[j,i] = I[i+1,k,f] — W[i,k+1,f] + ...
        sum{j in trabajos} X[j,k,f]*P[j,i+1];
```

```
54
55
56    subject to RelacionXIW_i1{k in trabajos, i in maquinas, f in fabricas : ( ...
          k<num_trabajos and i<num_maquinas and k>1 and i=1 )}:
57      0 - W[i,k,f] + sum{j in trabajos} X[j,k+1,f]*P[j,i] = I[i+1,k,f] - ...
            W[i,k+1,f] + sum{j in trabajos} X[j,k,f]*P[j,i+1];
58
59    subject to RelacionXIW_k1{k in trabajos, i in maquinas, f in fabricas : ( ...
          k<num_trabajos and i<num_maquinas and k=1 and i>1 )}:
60      I[i,k,f] - 0 + sum{j in trabajos} X[j,k+1,f]*P[j,i] = I[i+1,k,f] - ...
            W[i,k+1,f] + sum{j in trabajos} X[j,k,f]*P[j,i+1];
61
62    subject to RelacionXIW{k in trabajos, i in maquinas, f in fabricas : ( ...
          k<num_trabajos and i<num_maquinas and k>1 and i>1 )}:
63      I[i,k,f] - W[i,k,f] + sum{j in trabajos} X[j,k+1,f]*P[j,i] = I[i+1,k,f] - ...
            W[i,k+1,f] + sum{j in trabajos} X[j,k,f]*P[j,i+1];
64
65
66    # Definición de Cmax
67    subject to defCmax{f in fabricas}:
68      Cmax ≥ C[f];
69
70    # Definición de completacion
71    subject to defcompletacion{f in fabricas}:
72      C[f] = sum{i in maquinas : i<num_maquinas } sum{j in trabajos} ...
            X[j,1,f]*P[j,i] + sum{k in trabajos : k<num_trabajos} ...
            I[num_maquinas,k,f] + sum{k in trabajos} sum{j in trabajos} ...
            X[j,k,f]*P[j,num_maquinas];
73
74
75
76
77
78  data;
79    param num_trabajos:=10;
80    param num_maquinas:=2;
81    param num_fabricas:=10;
82    param P:
83          1   2:=
84    1     5   3
85    2     7   4
86    3     3   5
87    4     2   7
88    5     2   9
89    6     1   10
90    7     7   11
91    8     1   6
92    9     3   1
93    10    3   8;
```

```
# Capítulo 2 — The Distributed Permutation Flowshop Scheduling Problem
# Estudio de modelos para el problema DF|prmu|Cmax
# Modelo 3: Basado en las posiciones de los trabajos

# Definición de parámetros y conjuntos

  param num_trabajos;
  param num_maquinas;
  param num_fabricas;

  set trabajos:=1..num_trabajos;    #:=J
  set maquinas:=1..num_maquinas;    #:=M
  set fabricas:=1..num_fabricas;    #:=F

  param P{trabajos, maquinas};    # Tiempo de procesamiento P_{ji}



# Definición de variables
  var X{j in trabajos, k in trabajos, f in fabricas} binary;
      # 1 si el trabajo j ocupa la posición k en la factoría f

  var C{k in trabajos, i in maquinas, f in fabricas} ≥0;
      # tiempo de completación en la máquina i de la tarea que ocupa la posición ...
          k en la secuencia de producción de la fábrica f

  var Cmax ≥0;
      # Tiempo máximo de completación de todas las tareas, entre todas las fábricas
      # Maxium makespan among all factories



# Función objetivo y restricciones
  minimize tiempo_maximo_completacion:
    Cmax;

  # Cada trabajo tiene asignada una factoría en la que realizarse y una posición ...
      dentro del sistema de producción global
  subject to UnTrabajoUnaFabrica{j in trabajos}:
    sum{k in trabajos} sum{f in fabricas} X[j,k,f] = 1;

  # Toda posición del sistema de producción global está ocupada por un trabajo y ...
      asignada a una factoría
  subject to UnaPosicionUnTrabajo{k in trabajos}:
    sum{j in trabajos} sum{f in fabricas}  X[j,k,f] = 1;

  # Definición de Cmax
  subject to defCmax{k in trabajos, f in fabricas}:
    Cmax ≥ C[k, num_maquinas, f];

  # Definición de completacion
  subject to defcompletacion0{k in trabajos, i in maquinas, f in fabricas}:
     C[k,1,f] ≥ 0 + sum{j in trabajos} X[j,k,f]*P[j,1];

  subject to defcompletacion1{k in trabajos, i in maquinas, f in fabricas : i>1}:
     C[k,i,f] ≥ C[k,i-1,f] + sum{j in trabajos} X[j,k,f]*P[j,i];

  subject to defcompletacion2{k in trabajos, i in maquinas, f in fabricas : k>1}:
     C[k,i,f] ≥ C[k-1,i,f] + sum{j in trabajos} X[j,k,f]*P[j,i];

```

```
58
59
60
61 data;
62   param num_trabajos:=10;
63   param num_maquinas:=2;
64   param num_fabricas:=10;
65   param P:
66         1  2:=
67   1     5  3
68   2     7  4
69   3     3  5
70   4     2  7
71   5     2  9
72   6     1  10
73   7     7  11
74   8     1  6
75   9     3  1
76   10    3  8;
```

```
# Capítulo 2 — The Distributed Permutation Flowshop Scheduling Problem
# Estudio de modelos para el problema DF|prmu|Cmax
# Modelo 4: Modelo disgregado, basado en las posiciones de los trabajos

# Definición de parámetros y conjuntos

  param num_trabajos;
  param num_maquinas;
  param num_fabricas;

  set trabajos:=1..num_trabajos;   #:=J
  set maquinas:=1..num_maquinas;   #:=M
  set fabricas:=1..num_fabricas;   #:=F

  param P{trabajos, maquinas};   # Tiempo de procesamiento P_{ji}

  param M;          #Constante suficientemente grande

# Definición de variables
  var X{j in trabajos, k in trabajos} binary;
      # 1 si el trabajo j ocupa la posición k en la secuencia de producción

  var Y{k in trabajos, f in fabricas} binary;
      # 1 si el trabajo que está en la posición k se procesa en la factoría f

  var C{k in trabajos, i in maquinas} ≥0;
      # tiempo de completación en la máquina i de la tarea que ocupa la posición ...
          k en la secuencia de producción de la fábrica f

  var Cmax ≥0;
      # Tiempo máximo de completación de todas las tareas, entre todas las fábricas
      # Maxium makespan among all factories

# Función objetivo y restricciones
  minimize tiempo_maximo_completacion:
    Cmax;

  # Todo trabajo ocupa exactamente una posición en la secuencia de producción
  subject to UnTrabajoUnaPosicion{j in trabajos}:
    sum{k in trabajos} X[j,k] = 1;

  # Cada posición en la secuencia de producción se asigna una sola vez
  subject to UnaPosicionUnTrabajo{k in trabajos}:
    sum{j in trabajos} X[j,k] = 1;

  # Cada trabajo se asigna a una sola fábrica
  subject to UnTrabajoUnaFabrica{k in trabajos}:
    sum{f in fabricas} Y[k,f] = 1;

  # Definición de Cmax
  subject to defCmax{k in trabajos}:
    Cmax ≥ C[k, num_maquinas];
```

```
60
61
62    # Definición de completacion
63    subject to defcompletacion0{k in trabajos}:
64       C[k,1] ≥ 0 + sum{j in trabajos} X[j,k]*P[j,1];
65
66    subject to defcompletacion1{k in trabajos, i in maquinas : i>1}:
67       C[k,i] ≥ C[k,i-1] + sum{j in trabajos} X[j,k]*P[j,i];
68
69    subject to defcompletacion2{k in trabajos, l in trabajos, i in maquinas, f in ...
          fabricas : k>1 and l<k}:
70       C[k,i] ≥ C[l,i] + sum{j in trabajos} X[j,k]*P[j,i] - M*(1 - Y[k,f]) - M*(1 ...
             - Y[l,f]);
71
72
73
74
75
76
77
78  data;
79    param num_trabajos:=10;
80    param num_maquinas:=2;
81    param num_fabricas:=10;
82    param M:=98;
83    param P:
84          1   2:=
85    1     5   3
86    2     7   4
87    3     3   5
88    4     2   7
89    5     2   9
90    6     1   10
91    7     7   11
92    8     1   6
93    9     3   1
94    10    3   8;
```

```
# Capítulo 2 — The Distributed Permutation Flowshop Scheduling Problem
# Estudio de modelos para el problema DF|prmu|Cmax
# Modelo 5: Modelo basado en una secuenciación minimal a partir de dummy jobs

# Definición de parámetros y conjuntos

  param num_trabajos;
  param num_maquinas;
  param num_fabricas;

  set trabajos:=1..num_trabajos;    #:=J
  set maquinas:=1..num_maquinas;    #:=M
  set fabricas:=1..num_fabricas;    #:=F

  param P{trabajos, maquinas};    # Tiempo de procesamiento P_{ji}

  param M;          #Constante suficientemente grande

# Definición de variables
  var X{k in trabajos, j in trabajos : (k>j or k<j)} binary;
      # 1 si el trabajo j se procesa inmediatamente después del trabajo k

  var X_jdummy{k in trabajos} binary;
      # 1 si k es el trabajo que se procesa en último lugar en la factoría asociada
      # X_{k,0}

  var X_kdummy{j in trabajos} binary;
      # 1 si j es el trabajo que se procesa en primer lugar en la factoría asociada
      # X_{0,j}

  var C{j in trabajos, i in maquinas} ≥0;
      # Tiempo de completación del trabajo j en la máquina i

  var Cmax ≥0;
      # Tiempo máximo de completación de todas las tareas, entre todas las fábricas
      # Maxium makespan among all factories

# Función objetivo y restricciones
  minimize tiempo_maximo_completacion:
    Cmax;

  # Cada trabajo debe aparecer en una única posición de la secuencia de producción.
  subject to UnTrabajoUnaPosicion{j in trabajos}:
    X_kdummy[j] + sum{k in trabajos : (j>k or j<k) } X[k,j] = 1;

  # Definición X con respecto a la inmediatez.
  subject to DefX{k in trabajos}:
   X_jdummy[k] + sum{j in trabajos : (j>k or j<k)} X[k,j] ≤ 1;

  # Como hay F fábricas, se emplean exactamente F separadores en la secuencia ...
      global, que suceden a F−1 trabajos de esta secuencia
  subject to Separadores1:
    sum{j in trabajos} X_kdummy[j] = num_fabricas;

  subject to Separadores2:
    sum{k in trabajos} X_jdummy[k] = num_fabricas − 1;
```

```
60
61
62    # No se puede preceder a un trabajo, y luego seguirlo, a la vez.
63    subject to defprecedencia{k in trabajos, j in trabajos : (k<num_trabajos and ...
          j>k)}:
64       X[k,j] + X[j,k]  ≤ 1;
65
66    # Definición de Cmax
67    subject to defCmax{j in trabajos}:
68      Cmax ≥ C[j,num_maquinas];
69
70    # Definición de completacion
71    subject to defcompletacion0{j in trabajos}:
72      C[j,1] ≥ 0 + P[j,1];
73
74    subject to defcompletacion1{j in trabajos, i in maquinas : i > 1}:
75      C[j,i] ≥ C[j, i-1] + P[j,i];
76
77    subject to defcompletacion2{j in trabajos, i in maquinas }:
78      C[j,i] ≥ 0 + P[j,i] + M * ( X_kdummy[j] - 1);
79
80    subject to defcompletacion3{k in trabajos, j in trabajos, i in maquinas : (j>k ...
          or j<k)}:
81      C[j,i] ≥ C[k,i] + P[j,i] + M * ( X[k,j]  - 1);
82
83
84
85
86
87  data;
88    param num_trabajos:=10;
89    param num_maquinas:=2;
90    param num_fabricas:=10;
91    param M:=98;
92    param P:
93          1   2:=
94    1     5   3
95    2     7   4
96    3     3   5
97    4     2   7
98    5     2   9
99    6     1   10
100   7     7   11
101   8     1   6
102   9     3   1
103   10    3   8;
```

```
# Capítulo 2 — The Distributed Permutation Flowshop Scheduling Problem
# Estudio de modelos para el problema DF|prmu|Cmax
# Modelo 6: Basado en la secuenciación

# Definición de parámetros y conjuntos

  param num_trabajos;
  param num_maquinas;
  param num_fabricas;

  set trabajos:=1..num_trabajos;    #:=J
  set maquinas:=1..num_maquinas;    #:=M
  set fabricas:=1..num_fabricas;    #:=F

  param P{trabajos, maquinas};    # Tiempo de procesamiento P_{ji}

  param M;          #Constante suficientemente grande

# Definición de variables
  var X{k in trabajos, j in trabajos : (k<j and k<num_trabajos)} binary;
      # 1 si el trabajo j se procesa después del trabajo k

  var Y{j in trabajos, f in fabricas} binary;
      # 1 si el trabajo j se procesa en la factoría f

  var C{j in trabajos, i in maquinas} ≥0;
      # Tiempo de completación del trabajo j en la máquina i

  var Cmax ≥0;
      # Tiempo máximo de completación de todas las tareas, entre todas las fábricas
      # Maxium makespan among all factories


# Función objetivo y restricciones
  minimize tiempo_maximo_completacion:
    Cmax;

  # Cada trabajo debe estar asignado exactamente a una fábrica
  subject to UnTrabajoUnaFabrica{j in trabajos}:
    sum{f in fabricas} Y[j,f] = 1;

  # Definición de Cmax
  subject to defCmax{j in trabajos}:
    Cmax ≥ C[j,num_maquinas];

  # Definición de completacion
  subject to defcompletacion0{j in trabajos}:
    C[j,1] ≥ 0 + P[j,1];

  subject to defcompletacion1{j in trabajos, i in maquinas : i > 1}:
    C[j,i] ≥ C[j, i−1] + P[j,i];

  subject to defcompletacion2{k in trabajos, j in trabajos, i in maquinas, f in ...
      fabricas : (j > k and k < num_trabajos)}:
    C[k,i] ≥ C[j,i] + P[k,i] − M * X[k,j] − M * (1 − Y[k,f]) − M * (1 − Y[j,f]);

  subject to defcompletacion3{k in trabajos, j in trabajos, i in maquinas, f in ...
      fabricas : (j > k and k < num_trabajos)}:
    C[j,i] ≥ C[k,i] + P[j,i] − M * (1−X[k,j]) − M * (1 − Y[k,f]) − M * (1 − Y[j,f]);

```

```
59
60
61 data;
62   param num_trabajos:=10;
63   param num_maquinas:=2;
64   param num_fabricas:=10;
65   param M:=98;
66   param P:
67         1   2:=
68   1     5   3
69   2     7   4
70   3     3   5
71   4     2   7
72   5     2   9
73   6     1   10
74   7     7   11
75   8     1   6
76   9     3   1
77   10    3   8;
```

```
# Capítulo 3 — The Distributed Assembly Permutation Flowshop Scheduling Problem
# Estudio de modelos para el problema DAF|prmu|Cmax
# Modelo 1: Modelo basado en una secuenciación minimal a partir de dummy jobs

# Definición de parámetros y conjuntos

  param num_componentes;
  param num_maquinas;
  param num_fabricas;
  param num_productos;

  set componentes:=1..num_componentes;   #:=J
  set maquinas:=1..num_maquinas;   #:=M
  set fabricas:=1..num_fabricas;   #:=F
  set productos:=1..num_productos; #:=T

  param P{componentes, maquinas};   # Tiempo de procesamiento P_{ji}
  param PA{productos};              # Tiempo de ensamblaje PA_s

  param G{componentes, productos};  # Tabla binaria. 1 si el componente j se ...
      ensambla dentro del producto final s

  param M;          #Constante suficientemente grande

# Definición de variables
  var X{k in componentes, j in componentes : (k>j or k<j)} binary;
      # 1 si el componente j se procesa inmediatamente después del componente k

  var X_jdummy{k in componentes} binary;
      # 1 si k es el componente que se procesa en último lugar en la factoría ...
          asociada
      # X_{k,0}

  var X_kdummy{j in componentes} binary;
      # 1 si j es el componente que se procesa en primer lugar en la factoría ...
          asociada
      # X_{0,j}

  var Y{l in productos, s in productos : (l>s or l<s)} binary;
      # 1 si el producto final s se produce inmediatamente después del producto ...
          final l

  var Y_ldummy{s in productos} binary;
      # 1 si s es el producto final que se procesa en primer lugar en la línea ...
          de montaje
      # Y_{0,s}

  var C{j in componentes, i in maquinas} ≥0;
      # Tiempo de completación del componente j en la máquina i

  var CA{s in productos} ≥0;
      # Tiempo de completación del producto final s en la línea de montaje

  var Cmax ≥0;
      # Tiempo máximo de completación de todas las operaciones de ensamblaje ...
          sobre la línea de montaje
```

```
55
56
57  # Función objetivo y restricciones
58    minimize tiempo_maximo_completacion:
59      Cmax;
60
61    # Cada componente debe aparecer en una única posición de la secuencia de ...
          producción.
62    subject to UnComponenteUnaPosicion{j in componentes}:
63      X_kdummy[j] + sum{k in componentes : (j>k or j<k) } X[k,j] = 1;
64
65    # Definición X con respecto a la inmediatez.
66    subject to DefX{k in componentes}:
67     X_jdummy[k] + sum{j in componentes : (j>k or j<k)} X[k,j] ≤ 1;
68
69    # Como hay F fábricas, se emplean exactamente F separadores en la secuencia ...
          global, que suceden a F-1 componentes de esta secuencia
70    subject to Separadores1:
71      sum{j in componentes} X_kdummy[j] = num_fabricas;
72
73    subject to Separadores2:
74      sum{k in componentes} X_jdummy[k] = num_fabricas - 1;
75
76
77    # No se puede preceder a un trabajo, y luego seguirlo, a la vez.
78    subject to defprecedencia{k in componentes, j in componentes : ...
          (k<num_componentes and j>k)}:
79       X[k,j] + X[j,k]  ≤ 1;
80
81    # Definición de completacion
82    subject to defcompletacion0{j in componentes}:
83      C[j,1] ≥ 0 + P[j,1];
84
85    subject to defcompletacion1{j in componentes, i in maquinas : i > 1}:
86      C[j,i] ≥ C[j, i-1] + P[j,i];
87
88    subject to defcompletacion2{j in componentes, i in maquinas }:
89      C[j,i] ≥ 0 + P[j,i] + M * ( X_kdummy[j] - 1);
90
91    subject to defcompletacion3{k in componentes, j in componentes, i in maquinas ...
          : (j>k or j<k)}:
92      C[j,i] ≥ C[k,i] + P[j,i] + M * ( X[k,j]  - 1);
93
94    # Cada producto final debe aparecer en una única posición de la secuencia de ...
          ensamblaje.
95    subject to UnProductoUnaPosicion{s in productos}:
96      Y_ldummy[s] + sum{l in productos : (l>s or l<s) } Y[l,s] = 1;
97
98    # Definición Y con respecto a la inmediatez.
99    subject to DefY{l in productos}:
100     sum{s in productos : (l>s or l<s)} Y[l,s] ≤ 1;
101
102   # No se puede preceder a un producto final, y luego seguirlo, a la vez.
103   subject to defprecedencia_pr{l in productos, s in productos : (l<num_productos ...
          and s>l)}:
104      Y[l,s] + Y[s,l]  ≤ 1;
105
106
107
108
109
```

```
110
111
112   # Definición de completacion
113   subject to defcompletacion4{s in productos, j in componentes}:
114     CA[s] ≥ G[j,s] * C[j,num_maquinas] + PA[s];
115
116   subject to defcompletacion5{s in productos}:
117     CA[s] ≥ 0 + PA[s] + M * ( Y_ldummy[s] − 1);
118
119   subject to defcompletacion6{s in productos, l in productos : (l>s or l<s)}:
120     CA[s] ≥ CA[l] + PA[s] + M * ( Y[l,s] − 1);
121
122   # Definición de Cmax
123   subject to defCmax{s in productos}:
124     Cmax ≥ CA[s];
125
126
127
128
129
130 data;
131   param num_componentes:=10;
132   param num_maquinas:=2;
133   param num_fabricas:=2;
134   param num_productos:=2;
135
136   param P:
137         1  2:=
138   1     5  3
139   2     7  4
140   3     3  7
141   4     2  8
142   5     2  9
143   6     1  5
144   7     7  10
145   8     1  3
146   9     3  3
147   10    3  4;
148
149   param PA:=
150   1      2
151   2      5;
152
153     param G:
154         1  2:=
155   1     1  0
156   2     1  0
157   3     1  0
158   4     1  0
159   5     1  0
160   6     0  1
161   7     0  1
162   8     0  1
163   9     0  1
164   10    0  1;
165
166   param M:=97;
```

Bibliografía

[1] K. R. Baker, *Introduction to sequencing and scheduling.* New York: John Wiley, 1974.

[2] V. Botti and A. Giret, *ANEMONA: A Multi-agent Methodology for Holonic Manufacturing Systems*, 1st ed. Springer International Publishing, 2008, ch. 2: Holonic Manufacturing Systems, ISBN: 978-1-84800-309-5. [Online]. Disponible en: https://www.springer.com/gp/book/9781848003095

[3] H. V. Brussel, L. Bongaerts, J. Wyns, P. Valckenaers, and T. V. Ginderachter, "A conceptual framework for holonic manufacturing: Identification of manufacturing holons," *Journal of Manufacturing*, vol. 18, no. 1, pp. 35–52, 1999. [Online]. Disponible en: https://www.sciencedirect.com/science/article/pii/S0278612599800119

[4] H. V. Brussel, J. Wyns, P. Valckenaers, L. Bongaerts, and P. Peeters, "Reference architecture for holonic manufacturing systems: Prosa," *Computers in Industry*, vol. 37, no. 3, pp. 255 – 274, 1998. [Online]. Disponible en: http://www.sciencedirect.com/science/article/pii/S016636159800102X

[5] H. G. Campbell, R. A. Dudek, and M. L. Smith, "A heuristic algorithm for the n job, m machine sequencing problem," *Management Science*, vol. 16, no. 10, pp. B–630–B–637, 1970. [Online]. Disponible en: https://doi.org/10.1287/mnsc.16.10.B630

[6] B. Chen, C. N. Potts, and G. J. Woeginger, *A Review of Machine Scheduling: Complexity, Algorithms and Approximability.* Boston, MA: Springer US, 1999, pp. 1493–1641. [Online]. Disponible en: https://doi.org/10.1007/978-1-4613-0303-9_25

[7] W. Duan, Z. Li, M. Ji, Y. Yang, S. Wang, and B. Liu, "A hybrid estimation of distribution algorithm for distributed permutation flowshop scheduling with flowline eligibility," in *2016 IEEE Congress on Evolutionary Computation (CEC)*, July 2016, pp. 2581–2587. [Online]. Disponible en: https://doi.org/10.1109/CEC.2016.7744111

[8] J. Framinan and R. Leisten, "An efficient constructive heuristic for flowtime minimisation in permutation flow shops," *Omega*, vol. 31, no. 4, pp. 311 – 317, 2003. [Online]. Disponible en: http://www.sciencedirect.com/science/article/pii/S0305048303000471

[9] J. Gao and R. Chen, "A hybrid genetic algorithm for the distributed permutation flowshop scheduling problem," *International Journal of Computational Intelligence Systems*, vol. 4, no. 4, pp. 497–508, 2011. [Online]. Disponible en: https://www.tandfonline.com/doi/abs/10.1080/18756891.2011.9727808

[10] ——, "An neh-based heuristic algorithm for distributed permutation flowshop scheduling problems," *Scientific Research and Essays*, vol. 6, no. 14, pp. 3094–3100, 2011. [Online]. Disponible en: http://www.academicjournals.org/journal/SRE/article-abstract/8AF595125948

[11] J. Gao, R. Chen, and W. Deng, "An efficient tabu search algorithm for the distributed permutation flowshop scheduling problem," *International Journal of Production Research*, vol. 51, no. 3, pp. 641–651, 2013. [Online]. Disponible en: https://doi.org/10.1080/00207543.2011.644819

[12] A. Giret and V. Botti, "Engineering holonic manufacturing systems," *Computers in Industry*, vol. 60, no. 6, pp. 428 – 440, 2009, collaborative Engineering: from Concurrent Engineering to Enterprise Collaboration. [Online]. Disponible en: http://www.sciencedirect.com/science/article/pii/S0166361509000542

[13] F. Glover, "Heuristics for integer programming using surrogate constraints," *Decision Sciences*, vol. 8, no. 1, pp. 156–166, January 1997. [Online]. Disponible en: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1540-5915.1977.tb01074.x

[14] S. Goldman, R. Nagel, and K. Preiss, *Agile Competitors and Virtual Organizations: Strategies for Enriching the Customer*, ser. Industrial Engineering Series. Van Nostrand Reinhold, 1995. [Online]. Disponible en: https://books.google.es/books?id=33G1AAAAIAAJ

[15] R. Graham, E. Lawler, J. Lenstra, and A. Kan, "Optimization and approximation in deterministic sequencing and scheduling: a survey," in *Discrete Optimization II*, ser. Annals of Discrete Mathematics, P. Hammer, E. Johnson, and B. Korte, Eds. Elsevier, 1979, vol. 5, pp. 287 – 326. [Online]. Disponible en: http://www.sciencedirect.com/science/article/pii/S016750600870356X

[16] P. Hansen and N. Mladenović, "Variable neighborhood search: Principles and applications," *European Journal of Operational Research*, vol. 130, no. 3, pp. 449 – 467, 2001. [Online]. Disponible en: http://www.sciencedirect.com/science/article/pii/S0377221700001004

[17] S. Hatami, R. Ruiz, and C. Andrés-Romano, "The distributed assembly permutation flowshop scheduling problem," *International Journal of Production Research*, vol. 51, no. 17, pp. 5292–5308, 2013. [Online]. Disponible en: https://doi.org/10.1080/00207543.2013.807955

[18] S. Hatami, "The distributed and assembly scheduling problem," Ph.D. dissertation, Universitat Politénica de Valencia, 2016. [Online]. Disponible en: https://riunet.upv.es/handle/10251/64072

[19] h.c.E. Westkämpfer, "Manufacturing on demand in production networks," *CIRP Annals*, vol. 46, no. 1, pp. 329 – 334, 1997. [Online]. Disponible en: http://www.sciencedirect.com/science/article/pii/S0007850607608361

[20] S. R. Hejazi and S. Saghafian, "Flowshop-scheduling problems with makespan criterion: a review," *International Journal of Production Research*, vol. 43, no. 14, pp. 2895–2929, 2005. [Online]. Disponible en: https://doi.org/10.1080/0020754050056417

[21] N. K. Jha, *Handbook of flexible manufacturing systems*, 1st ed. Academic Press, 1991, ISBN: 978-0-12385-310-3. [Online]. Disponible en: https://doi.org/10.1016/C2009-0-21580-7

[22] S. M. Johnson, "Optimal two- and three-stage production schedules with setup times included," *Naval Research Logistics (NRL)*, vol. 1, 1954. [Online]. Disponible en: https://onlinelibrary.wiley.com/doi/abs/10.1002/nav.3800010110

[23] Y. Koren, U. Heisel, F. Jovane, T. Moriwaki, G. Pritschow, G. Ulsoy, and H. V. Brussel, "Reconfigurable manufacturing systems," *CIRP Annals*, vol. 48, no. 2, pp. 527 – 540, 1999. [Online]. Disponible en: http://www.sciencedirect.com/science/article/pii/S0007850607632326

[24] M. Laguna and R. Martí, *Scatter search: Methodology and implementations in C*. Kluwer Academic Publishers, 2003. [Online]. Disponible en: https://doi.org/10.1007/978-1-4615-0337-8

[25] J. Lin, Z.-J. Wang, and X. Li, "A backtracking search hyper-heuristic for the distributed assembly flow-shop scheduling problem," *Swarm and Evolutionary Computation*, vol. 36, pp. 124 – 135, 2017. [Online]. Disponible en: http://www.sciencedirect.com/science/article/pii/S2210650216305028

[26] J. Lin and S. Zhang, "An effective hybrid biogeography-based optimization algorithm for the distributed assembly permutation flow-shop scheduling problem," *Computers & Industrial Engineering*, vol. 97, pp. 128 – 136, 2016. [Online]. Disponible en: http://www.sciencedirect.com/science/article/pii/S0360835216301450

[27] S.-W. Lin, K.-C. Ying, and C.-Y. Huang, "Minimising makespan in distributed permutation flowshops using a modified iterated greedy algorithm," *International Journal of Production Research*, vol. 51, no. 16, pp. 5029–5038, 2013. [Online]. Disponible en: https://www.tandfonline.com/doi/abs/10.1080/00207543.2013.790571

[28] B. Liu, K. Wang, and R. Zhang, "Variable neighborhood based memetic algorithm for distributed assembly permutation flowshop," in *2016 IEEE Congress on Evolutionary Computation (CEC)*, July 2016, pp. 1682–1686. [Online]. Disponible en: https://doi.org/10.1109/CEC.2016.7743990

[29] H. Liu and L. Gao, "A discrete electromagnetism-like mechanism algorithm for solving distributed permutation flowshop scheduling problem," in *2010 International Conference on Manufacturing Automation*, Dec 2010, pp. 156–163. [Online]. Disponible en: https://doi.org/10.1109/ICMA.2010.17

[30] G. Louppe, L. Wehenkel, A. Sutera, and P. Geurts, "Understanding variable importances in forests of randomized trees," in *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 1*, ser. NIPS'13. USA: Curran Associates Inc., 2013, pp. 431–439. [Online]. Disponible en: http://dl.acm.org/citation.cfm?id=2999611.2999660

[31] A. S. Manne, "On the job-shop scheduling problem," *Operations Research*, vol. 8, no. 2, pp. 219–223, 1960. [Online]. Disponible en: https://doi.org/10.1287/opre.8.2.219

[32] R. Martí, M. Laguna, and F. Glover, "Principles of scatter search," *European Journal of Operational Research*, vol. 169, no. 2, pp. 359 – 372, 2006, feature Cluster on Scatter Search Methods for Optimization. [Online]. Disponible en: http://www.sciencedirect.com/science/article/pii/S0377221704005430

[33] N. Mladenović and P. Hansen, "Variable neighborhood search," *Computers & Operations Research*, vol. 24, no. 11, pp. 1097 – 1100, 1997. [Online]. Disponible en: http://www.sciencedirect.com/science/article/pii/S0305054897000312

[34] B. Naderi and R. Ruiz, "The distributed permutation flowshop scheduling problem," *Computers & Operations Research*, vol. 37, no. 4, pp. 754 – 768, 2010. [Online]. Disponible en: http://www.sciencedirect.com/science/article/pii/S0305054809001786

[35] ——, "A scatter search algorithm for the distributed permutation flowshop scheduling problem," *European Journal of Operational Research*, vol. 239, no. 2, pp. 323 – 334, 2014. [Online]. Disponible en: http://www.sciencedirect.com/science/article/pii/S0377221714004470

[36] M. Nawaz, E. E. Enscore, and I. Ham, "A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem," *Omega*, vol. 11, no. 1, pp. 91 – 95, 1983. [Online]. Disponible en: http://www.sciencedirect.com/science/article/pii/0305048383900889

[37] D. S. Palmer, "Sequencing jobs through a multi-stage process in the minimum total time – a quick method of obtaining a near optimum," *OR*, vol. 16, 03 1965. [Online]. Disponible en: https://link.springer.com/article/10.1057/jors.1965.8

[38] M. L. Pinedo, *Scheduling: Theory, Algorithms and Systems*, 5th ed. Springer International Publishing, 2016, ISBN: 978-3-319-26580-3. [Online]. Disponible en: https://www.springer.com/gp/book/9781489990433

[39] R. Ruiz, C. Maroto, and J. Alcaraz, "Two new robust genetic algorithms for the flowshop scheduling problem," *Omega*, vol. 34, no. 5, pp. 461 – 476, 2006. [Online]. Disponible en: http://www.sciencedirect.com/science/article/pii/S0305048305000174

[40] R. Ruiz and T. Stützle, "A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem," *European Journal of Operational Research*, vol. 177, no. 3, pp. 2033 – 2049, 2007. [Online]. Disponible en: http://www.sciencedirect.com/science/article/pii/S0377221705008507

[41] E. F. Stafford Jr, F. T. Tseng, and J. N. D. Gupta, "Comparative evaluation of milp flowshop models," *The Journal of the Operational Research Society*, vol. 56, no. 1, pp. 88–101, 2005. [Online]. Disponible en: http://www.jstor.org/stable/4102253

[42] T. Stützle, "Applying iterated local search to the permutation flow shop problem," *FG Intellektik, TU Darmstadt, Darmstadt, Germany*, 1998. [Online]. Disponible en: http://iridia.ulb.ac.be/~stuetzle/publications/AIDA-98-04.pdf

[43] A. Tharumarajah, A. J. Wells, and L. Nemes, "Comparison of emerging manufacturing concepts," in *Systems, Man, and Cybernetics, 1998. 1998 IEEE International Conference on*, vol. 1, Oct 1998, pp. 325–331 vol.1. [Online]. Disponible en: https://doi.org/10.1109/ICSMC.1998.725430

[44] N. T. Thomopoulos, *Assembly Line Planning and Control*. Springer International Publishing, 2014, ISBN: 978-3-319-01398-5. [Online]. Disponible en: https://www.springer.com/gp/book/9783319013985

[45] F. T. Tseng and E. F. Stafford, "New milp models for the permutation flowshop problem," *The Journal of the Operational Research Society*, vol. 59, no. 10, pp. 1373–1386, 2008. [Online]. Disponible en: http://www.jstor.org/stable/20202212

[46] K. Ueda, "A concept for bionic manufacturing systems based on dna-type information," in *Human Aspects in Computer Integrated Manufacturing*, G. OLLING and F. KIMURA, Eds. Amsterdam: Elsevier, 1992, pp. 853 – 863. [Online]. Disponible en: https://www.sciencedirect.com/science/article/pii/B9780444894656500788

[47] H. Van Brussel, *Holonic Manufacturing Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 654–659. [Online]. Disponible en: https://doi.org/10.1007/978-3-642-20617-7_6556

[48] S. Y. Wang and L. Wang, "An estimation of distribution algorithm-based memetic algorithm for the distributed assembly permutation flow-shop scheduling problem," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 46, no. 1, pp. 139–149, Jan 2016. [Online]. Disponible en: https://doi.org/10.1109/TSMC.2015.2416127

[49] S.-y. Wang, L. Wang, M. Liu, and Y. Xu, "An effective estimation of distribution algorithm for solving the distributed permutation flow-shop scheduling problem," *International Journal of Production Economics*, vol. 145, no. 1, pp. 387 – 396, 2013. [Online]. Disponible en: http://www.sciencedirect.com/science/article/pii/S0925527313002296

[50] H.-J. Warnecke, *The Fractal Factory — an Integrating Approach.* Berlin, Heidelberg: Springer Berlin Heidelberg, 1993, pp. 137–217. [Online]. Disponible en: https://doi.org/10.1007/978-3-642-78124-7_4

[51] Wikibooks, "Glpk/gmpl (mathprog) — wikibooks, the free textbook project," 2017, [Revisado 17-Mayo-2018]. [Online]. Disponible en: https://en.wikibooks.org/w/index.php?title=GLPK/GMPL_(MathProg)&oldid=3208474

[52] ——, "Glpk — wikibooks, the free textbook project," 2018, [Revisado 17-Mayo-2018]. [Online]. Disponible en: https://en.wikibooks.org/w/index.php?title=GLPK&oldid=3378358

[53] Wikipedia, "Notation for theoretic scheduling problems — wikipedia, the free encyclopedia," 2017, revisado 12-Marzo-2018. [Online]. Disponible en: https://en.wikipedia.org/w/index.php?title=Notation_for_theoretic_scheduling_problems