Trabajo Fin de Grado
Grado en Ingeniería de Organización Industrial

Taller de flujo regular con tiempos de cambio dependientes de la secuencia: Heurísticas constructivas basadas en la memoria.

Autora: Belén Navarro García
Tutor: Víctor Fernández-Viagas Escudero

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

Sevilla, 2020

Trabajo Fin de Grado
Grado en Ingeniería de Organización Industrial

Taller de flujo regular con tiempos de cambio dependientes de la secuencia: Heurísticas constructivas basadas en la memoria.

Autora:
Belén Navarro García

Tutor:
Víctor Fernández-Viagas Escudero
Profesor Contratado Doctor

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

Sevilla, 2020

Trabajo Fin de Grado: Taller de flujo regular con tiempos de cambio dependientes de la secuencia: Heurísticas constructivas basadas en la memoria.

Autora: Belén Navarro García

Tutor: Víctor Fernández-Viagas Escudero

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

Presidente:

Vocal/es:

Secretario:

Acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Sevilla, 2020

Agradecimientos

Quiero agradecer el apoyo y la confianza de quienes me han animado en la realización de este proyecto que pone fin a una etapa de mi vida como estudiante. En especial a mi familia y amigos, cuya motivación y orgullo me han impulsado a perseguir mis metas; y a Álvaro, por estar a mi lado estos cuatro años, animándome siempre a continuar con su constante apoyo y comprensión.

Gracias también a Víctor, la persona que me ha guiado durante este proceso haciendo posible la realización de este trabajo. Gracias por su tiempo, su confianza en mí y motivación constante, y por poner su mejor esfuerzo en iniciarme en la investigación, siendo un gran referente a seguir.

A todo aquel que invierta su tiempo en leer este trabajo, gracias.

Belén Navarro García

Sevilla, 2020

Resumen

Este Trabajo Fin de Grado aborda un problema de programación de la producción para un taller con una configuración de tipo *Flow-Shop*. Se presenta como restricción la permutación y la existencia de tiempos de *set-up* no anticipatorios dependientes de la máquina y la secuencia.

Se pretende alcanzar una solución de calidad para el problema en un tiempo de cálculo razonable, buscando como objetivo la minimización del makespan. Los algoritmos utilizarán una pequeña modificación de la función objetivo genérica, priorizando en caso de empate las secuencias con menor tiempo ocioso entre el procesado de los trabajos.

Los métodos aproximados propuestos para la resolución de este problema de secuenciación serán heurísticas constructivas basadas en la memoria. Incluirán una búsqueda local iterativa en cada paso de construcción de la secuencia, repitiendo las combinaciones de trabajos más prometedores en futuras iteraciones.

Se han implementado cuatro heurísticas similares, realizando además una evaluación computacional con 1000 instancias para distintas versiones de ellas. Un total de 17 heurísticas han sido comparadas, que demuestran la eficiencia de las propuestas.

Abstract

This thesis addresses a production scheduling problem using a *flow shop* layout and makespan minimisation. The permutation and the existence of non-anticipatory sequence-dependent setup times are presented as constraints.

The goal of this study is to achieve a good solution for the problem in a reasonable computational time. To deal with it, several approximate algorithms are proposed, considering and idle-time based indicator to break ties with the same makespan. These proposed approximate methods are based on a memory mechanism, which works in a contrary way that the tabu search.

In addition, they will include an iterative local search at each step of the sequence construction, repeating the most promising job combinations in future iterations.

As a result, four heuristics have been implemented, and their performance has been tested on a benchmark composed of 1000 instances. A total of 17 heuristics have been compared.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

1 INTRODUCCIÓN

Como comienzo de este Trabajo Fin de Grado, se realiza una breve explicación de las razones que han llevado a la realización del presente documento, cuál es su propósito, ámbito, alcance, así como una pequeña introducción del proceso de resolución del problema que describe la metodología utilizada y las conclusiones que se esperan alcanzar.

1.1 Motivación

La programación de la producción es una rama de la organización industrial de gran importancia. Una adecuada secuenciación de las tareas a procesar en un sistema productivo proporciona grandes ventajas, ya que se produce una gran optimización en cuanto al aprovechamiento de los recursos. La naturaleza de estos recursos y tareas es bastante amplia, dado que por ejemplo, un recurso productivo puede incluir desde una máquina hasta un conjunto de operarios (recursos humanos).

Gracias a una programación de la producción ajustada se puede conseguir minimizar los costes y el tiempo, consiguiendo un mayor alcance y cumpliendo los objetivos marcados en la producción de manera eficiente. Como indica Pinedo (2012), esta secuenciación conlleva un alto volumen de decisiones centradas en la asignación de recursos a tareas en un determinado periodo temporal, con el fin de optimizar dicha colocación en base a uno o múltiples objetivos fijados.

La secuenciación no solo aplica al ámbito industrial, sino que además juega un rol importante en cuanto a logística y distribución, o en cualquier servicio con una duración determinada. Su uso es fundamental para empresas con un volumen de tareas y recursos medio o elevado, permitiendo una competitividad en el mercado mucho mayor.

Por ello, el interés de este Trabajo Fin de Grado reside en aprovechar la capacidad de los recursos de un taller de flujo regular con características específicas, para conseguir una producción óptima y reducir los tiempos de procesado de las tareas.

1.2 Objetivo y alcance del proyecto

Por los motivos expuestos en el apartado anterior, el objetivo de este proyecto será el estudio de un problema de programación de la producción: un taller de flujo regular con restricción de permutación y con la existencia de tiempos de *set-up* ($F_m|prmu, S_{ijk}|C_{max}$). El taller de flujo regular es uno de los problemas más estudiados en la investigación operativa (Fernández-Viagas et al., 2018). Fundamentalmente se debe a dos aspectos: por un lado, es un problema muy presente en el sector industrial; y por otro lado, los algoritmos desarrollados para este problema se han demostrado muy eficientes en problemas similares a lo largo de los últimos años. Se buscan así métodos que simplifiquen la búsqueda y proporcionen soluciones cercanas al óptimo en estos problemas.

Como resultado, la solución que se pretende obtener es una secuencia que contenga el orden de secuenciación de los trabajos a procesar en el taller. Ésta será de mayor calidad cuanto menor tiempo de proceso total conlleve (minimización de *makespan*).

Al tratarse de un problema complejo (*NP-hard*), no se tratará de evaluar todas las combinaciones de soluciones posibles, sino que se emplearán métodos aproximados que proporcionen soluciones próximas al óptimo en un tiempo de cálculo razonable. Se pretenden combinar heurísticas constructivas con mecanismos propios de metaheurísticas de búsqueda local, con el objetivo de proporcionar mejores soluciones que la conocida

heurística NEH (Nawaz, Enscore Jr & Ham, 1983).

El funcionamiento de la búsqueda local seleccionada será similar al de la conocida búsqueda tabú, con la diferencia de que en lugar de conservar en una lista los movimientos prohibidos, se mantendrán las combinaciones más prometedoras que se hayan realizado durante el procedimiento de construcción de la secuencia. Estas combinaciones prometedoras se repetirán para conseguir soluciones de mayor calidad, y evitar que se descarten soluciones, que aunque a priori no sean las mejores, podrían llegar a serlo.

Como se verá en el último capítulo del documento, se presentarán cuatro procedimientos de resolución del problema que proporcionan soluciones de buena calidad en un determinado tiempo. Generalmente cuando el tiempo que conlleve el cálculo sea mayor, se generarán mejores soluciones.

1.3 Estructura del documento

El contenido del trabajo se distribuye en los siguientes capítulos:

- Capítulo 1: Introducción.

  En este primer documento se describe el propósito del presente Trabajo Fin de Grado. Se comenta brevemente su interés de estudio, su aplicabilidad, sus objetivos, ámbito, alcance, así como la metodología aplicada y un pequeño avance de las conclusiones obtenidas tras la investigación.

- Capítulo 2: Marco teórico.

  A modo introductorio, se trata de enmarcar el proyecto dentro del ámbito de la organización industrial, comentando algunos de los conceptos teóricos sobre la programación de la producción necesarios para la correcta comprensión del documento. Además, se realiza una breve introducción al problema objeto del proyecto.

- Capítulo 3: Análisis del problema.

  En este tercer capítulo, se describe en profundidad el problema a resolver, con mayor nivel de detalle. Se exponen todas las características del entorno, restricciones, objetivos, y cualquier aspecto que ayude a comprender el problema en cuestión.

- Capítulo 4: Análisis de la metodología

  En este capítulo se hace referencia a las metodologías existentes que suelen emplearse en la actualidad para la resolución de problemas similares al que se trata en este proyecto, comentando con mayor detalle aquellas que se usarán como base para la resolución del problema.

- Capítulo 5: Aplicación de la metodología.

  Este apartado describe la metodología aplicada en la resolución del problema descrito en el capítulo 3. En concreto, se exponen los algoritmos aproximados que se han desarrollado, explicando detalladamente el funcionamiento de cada uno de ellos.

- Capítulo 6: Evaluación computacional.

  En este capítulo se presentan los resultados obtenidos al resolver el problema haciendo uso de cada uno de los algoritmos descritos en el capítulo anterior (4 variantes), desarrollados en lenguaje de programación C#. Para ello, se especifica el conjunto de pruebas realizadas, y se muestra la comparativa de los resultados obtenidos en cuanto al nivel de calidad de la solución y al tiempo computacional del algoritmo.

- Capítulo 7: Conclusiones.

  Para finalizar, este último capítulo recoge las conclusiones obtenidas con el desarrollo del presente Trabajo Fin de Grado, así como las posibles futuras líneas de investigación que se podrían seguir.

Al final del documento se encuentran las referencias bibliográficas de la información consultada para su elaboración.

2 MARCO TEÓRICO

Con el objetivo de enmarcar el desarrollo del proyecto en el ámbito de la organización de la producción, en este capítulo se introducen los conceptos considerados necesarios para una adecuada comprensión del documento.

2.1 Introducción

"La organización de la producción (*Production Managment*) es un proceso industrial consistente en la toma de un número elevado de decisiones a lo largo del tiempo para asegurar la entrega al cliente de los productos con la máxima calidad, mínimo coste, y mínimo tiempo de entrega." [Framiñán, Leisten y Ruiz, 2014]

En todo proceso productivo se realizan transformaciones sobre unas determinadas entradas con el fin de obtener las salidas deseadas, agregándole valor al producto mediante el uso de recursos. Todas las acciones de planificación y gestión que incluye cualquier proceso productivo son abarcadas por la organización de la producción. Esta engloba un gran volumen de decisiones diversas que aplican a distintos ámbitos y plazos de tiempo.

Visualmente, en el siguiente esquema de la Figura 2.1 se recogen los bloques más comunes siguiendo la conocida estructura denominada "*Advanced Planning System*, APS":

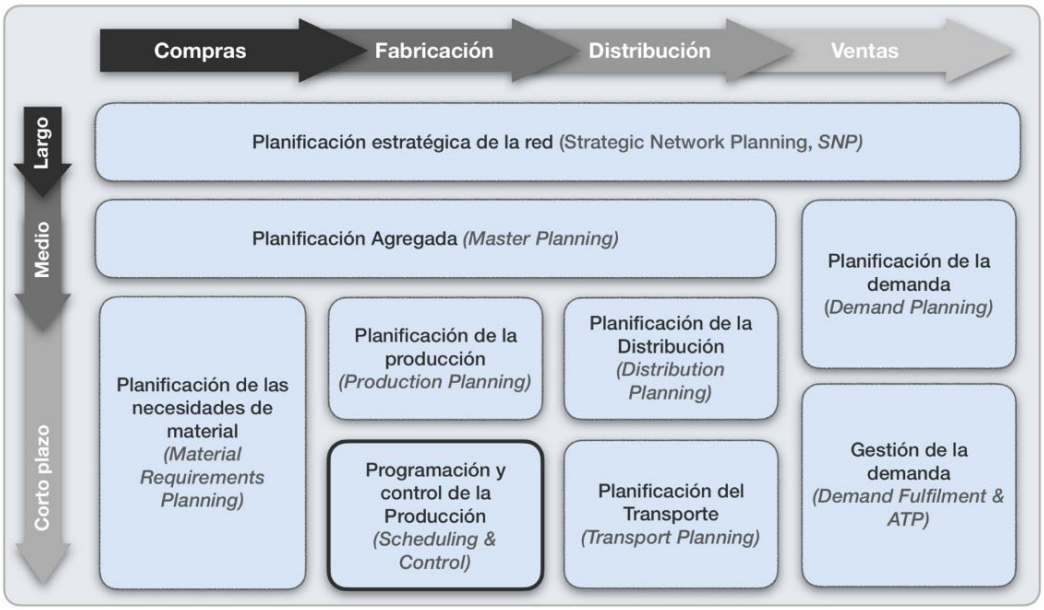

Figura 2.1. Diagrama APS, "*Advanced Planning System*". [Fuente: Elaboración propia]

Como se puede observar, entre todos los aspectos de planificación que engloba la organización de la producción, en concreto la programación de la producción (*Manufacturing Scheduling*) se sitúa en la toma de decisiones a corto plazo relativas al proceso de fabricación.

Gracias a la programación de la producción, es posible establecer el periodo de tiempo asignado para la realización de cada una de las operaciones de transformación o transporte de material necesarias para la fabricación de un conjunto de productos, así como qué recursos de la empresa se emplean para ello (materia prima, mano de obra, maquinaria, logística…).

Estos recursos con capacidad productiva para el transporte o transformación del producto objeto de la operación (denominado trabajo) se denominan máquinas. El concepto de máquina no solo representa a un objeto mecánico, sino también abarca a recursos humanos, como son los operarios o un grupo de objetos mecánicos que realicen una operación en su conjunto. De igual modo, el concepto de trabajo puede representar tanto a un único objeto, como a un grupo de objetos físicos.

Toda operación se realiza durante una determinada duración temporal, que se conoce como tiempo de proceso (*processing time*). Dicho tiempo dependerá tanto del trabajo (*job*) a procesar como de la máquina (*machine*) en la que se realice la operación.

Como resultado de la programación de la producción se obtiene el denominado programa (*schedule*), que recoge la asignación en la escala temporal concreta de las máquinas para el procesado de los trabajos. Es decir, el programa asigna las fechas de comienzo de cada trabajo en cada máquina, pero el orden en que cada trabajo comienza a procesarse en cada máquina viene dado por una secuencia (*sequence*). Para que un programa sea admisible debe cumplir con todas las restricciones y características del entorno productivo. Para conseguirlo, se emplean los algoritmos, que nos permiten obtener un programa factible que cumpla con los objetivos marcados.

Cabe destacar la importancia de las decisiones relativas a la programación de la producción, que, a pesar de ser tomadas en cortos intervalos de tiempo, pueden influir en gran medida en el resultado final. Para la asignación de los recursos llevada a cabo, se requiere una serie de decisiones exhaustivas y precisas tomadas en el instante adecuado. Por este motivo, el tiempo es un factor muy importante para tener en cuenta a la hora de desarrollar un algoritmo, que a veces debe proporcionar decisiones instantáneas, incluso en tiempo real.

Dichas decisiones vienen condicionadas por diversos factores que dependen del tipo de entorno en el que nos encontremos, las restricciones impuestas, o los objetivos fijados. Graham et al. (1979) propone una de las notaciones que más se usan en la actualidad con el fin de definir las características de un problema de programación de la producción, basada en el uso de tres parámetros de la siguiente forma: $\alpha \mid \beta \mid \gamma$. Cada uno de ellos describe las características de las máquinas (entorno, α), de los trabajos (restricciones, β), y de la función objetivo (objetivos, γ), respectivamente (véase Figura 2.2). A continuación, se van a comentar con mayor detalle.

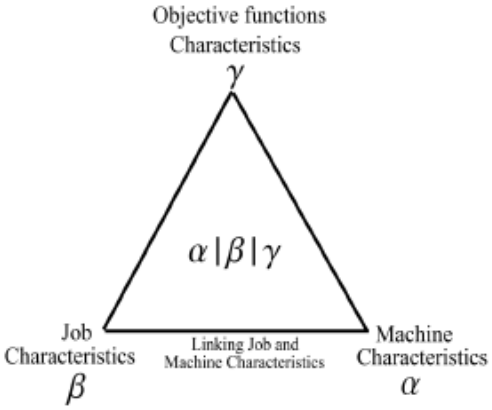

Figura 2.2. Representación de los entornos, restricciones y objetivos.
[Fuente: Temario Programación y Control de la Producción, Escuela Técnica Superior de Ingeniería]

2.2 Entornos (α)

Existen distintos tipos de entornos de fabricación (*layout*) con distintas características en función del número, distribución y características de las máquinas. A continuación, se comentarán brevemente algunos de los entornos más frecuentes, englobados en la clasificación de la Figura 2.3.

- Una máquina (*Single Machine, $\alpha = 1$*). Es el modelo más sencillo, ya que solo dispone de una etapa con una máquina que procesa todos los trabajos uno a uno, una única vez, sin ruta posible.
- Máquinas en paralelo (*Parallel Machines*). Es el mismo caso que el anterior, pero con mayor número de máquinas disponibles para procesar los trabajos, de nuevo sin ruta posible, ya que se trata de una única etapa. Según la velocidad de las máquinas, se diferencia entre:
    - Máquinas idénticas ($\alpha = P_m$): todas las máquinas tienen la misma velocidad de procesado.
    - Máquinas uniformes ($\alpha = Q_m$): la velocidad de procesado de cada máquina es distinta.
    - Máquinas no relacionadas, ($\alpha = R_m$): cada máquina procesa cada trabajo a una velocidad distinta, por lo que el tiempo de proceso dependerá de la máquina a la que se asigne.
- Tipo taller: se dispone de varias máquinas en serie, donde cada una realiza una operación distinta sobre el trabajo, el cuál debe pasar por las máquinas del taller según su ruta de fabricación. Según el tipo de ruta que siguen los trabajos a la hora de ser procesados, se diferencian 3 tipos de talleres:
    - Taller de flujo regular (*Flow Shop, $\alpha = F_m$*): todos los trabajos siguen la misma ruta de fabricación predeterminada.
    - Taller de trabajos (*Job Shop, $\alpha = J_m$*): cada trabajo sigue una ruta predeterminada distinta.
    - Taller abierto (*Open Shop, $\alpha = O_m$*): la ruta de los trabajos es libre, no está predeterminada.
- Híbridos/Flexibles ($\alpha = HF_m, HJ_m, HO_m$). Se trata de un taller (de flujo regular, de trabajos, o abierto) formado por varias etapas, donde cada una corresponde a una sola máquina o varias máquinas paralelas.

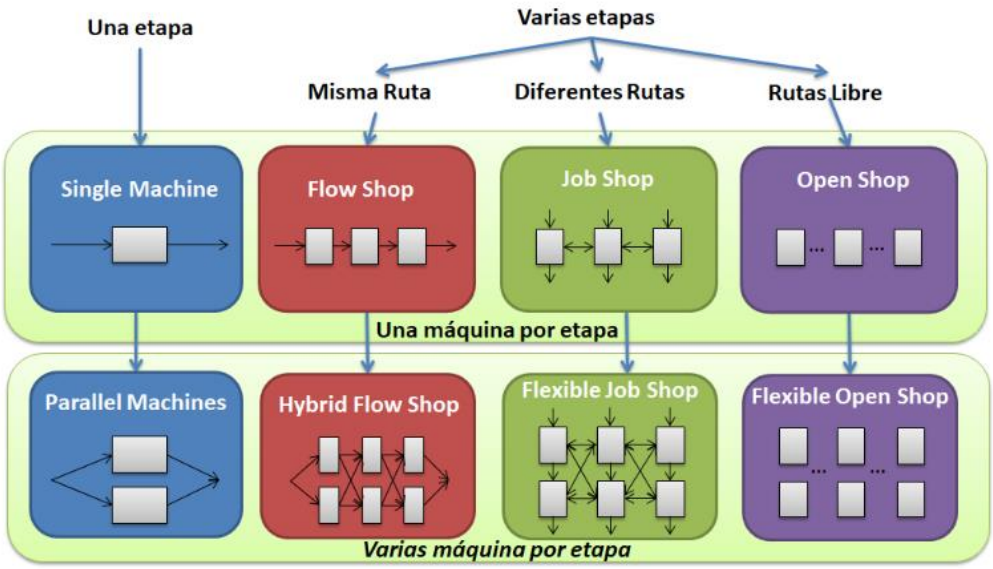

Figura 2.3. Entornos clásicos de fabricación.
[Fuente: Fernández-Viagas, V. (s. f.)]

2.3 Restricciones (β)

Según las características de los trabajos que van a ser procesados en las máquinas, el sistema tendrá ciertas restricciones. Existen multitud de ellas, por lo que se van a destacar algunas de las más comunes.

En primer lugar, las suposiciones generales por defecto presentes en los problemas de programación de la producción (cuando el campo del parámetro β se encuentra vacío, $\beta = \emptyset$) son las siguientes:

- Disponibilidad total de máquinas y trabajos en el instante inicial.
- No existe la posibilidad de interrumpir un trabajo que está siendo procesado. Una vez que inicia, se debe continuar hasta que transcurra por completo su tiempo de proceso correspondiente.
- Una máquina no puede procesar varios trabajos simultáneamente.
- Un trabajo no puede ser procesado en varias máquinas simultáneamente.
- El buffer entre máquinas es infinito, no hay un límite de unidades de trabajos en cola de espera para ser procesados en una máquina.
- Los tiempos de transporte entre máquinas es despreciable.

En cambio, cuando el campo del parámetro β no se encuentra vacío, en su lugar se pueden encontrar alguna de las restricciones contenidas en la Tabla 2.1, entre otras, o la combinación de varias de ellas.

Tabla 2.1. Restricciones habituales en el campo β.

Restricción	β	Descripción
Interrupción (*preemption*)	*pmtn-non-resumable, pmtn-semi-resumable, pmtn-resumable*	Posibilidad de interrumpir una operación de procesado de un trabajo una vez comenzada. Se diferencian 3 tipos en función de si es posible continuar el trabajo en el mismo punto en el que se encontraba en el momento de la interrupción (*resumable*), si no es posible (*non-resumable),* o si se pierde una proporción de la tarea realizada (*semi-resumable*).
Tiempo de cambio (*set-up*)	s_{ij}, s_{ijk}	La máquina requiere de un tiempo de ajuste o preparación antes de comenzar a procesar el trabajo. Estos tiempos pueden ser dependientes tanto de las máquinas (subíndice *i*), como de los trabajos (*j*) y de la secuencia (*k*), o una combinación de ellos.
Tiempos de proceso especiales	p_{ij}	Los tiempos de proceso de los trabajos son constantes, varían en función de la velocidad de las máquinas, o son cambiantes por efecto del aprendizaje o deterioro.
Fecha de llegada (*release date*)	r_j	La disponibilidad de los trabajos no se corresponde con el instante inicial del horizonte de la programación.
Fecha de entrega (*due date*)	$\overline{d_j}$, $d_j = d$	Los trabajos poseen una fecha de entrega obligatoria. Esta puede ser común para todos ellos.
Precedencia (*precedence*)	*prec*	Existen relaciones de precedencia entre los trabajos, es decir, el comienzo de una operación está condicionado a la previa finalización de otras operaciones.

Lotes (*batching*)	*p-batch(b), s-batch(b)*	Las máquinas tienen capacidad para procesar los trabajos por lotes, por lo que pueden procesar simultáneamente hasta *b* trabajos. Los trabajos pueden procesarse tanto en paralelo como en serie.
Permutación	*prmu*	En entornos compuestos por varias etapas, el orden de procesado de los trabajos en cada etapa es el mismo.
Tiempos ociosos no permitidos	*No-idle*	No se permiten tiempos ociosos en las máquinas. Es decir, no puede transcurrir tiempo entre el procesado de dos trabajos distintos en una misma máquina.
Almacenes de máquinas (*buffer*)	*Buffer* = b_i	La capacidad de espera de los trabajos para ser procesados en una máquina o etapa *i* está limitada a b_i trabajos.
Esperas no permitidas	*No-wait*	No se permite la espera de trabajos entre etapas (inexistencia de buffer entre máquinas).

2.4 Objetivos (γ)

El último de los aspectos mencionados para tener en cuenta a la hora de describir un problema de programación de la producción se corresponde con los objetivos. A la hora de buscar una solución se fija un objetivo a conseguir, que viene marcado por una función objetivo, cuyo valor se pretende que sea mínimo, máximo, fijo y predeterminado, u acotado en un intervalo.

Las funciones objetivo se pueden clasificar en 4 categorías en función de las 3 variables que componen el conocido "triángulo de hierro": tiempo, coste, y calidad (véase Figura 2.4). Estas tres variables están relacionadas entre sí, ya que el aumento de una de ellas suele implicar una variación del resto. Por ejemplo, si se requiere un producto con alta calidad, su coste y tiempo de fabricación serán más elevados. Al equilibrio entre las tres se le denomina flexibilidad.

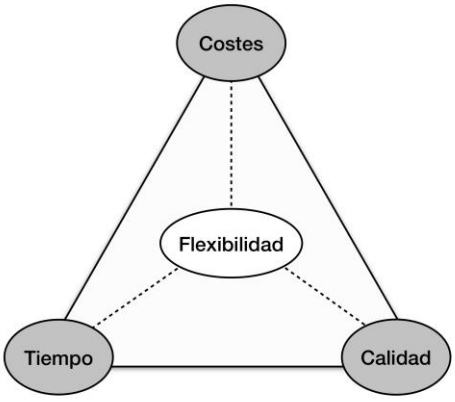

Figura 2.4. Representación del triángulo de hierro. [Fuente: Elaboración propia]

A continuación, se exponen algunas de las medidas más comunes respecto a la variable tiempo, ya que el objetivo del problema a abordar en este proyecto se centrará en esta categoría de funciones:

- Tiempo de terminación (*Completion time,* C_j): mide el periodo de tiempo en el que el trabajo finaliza su procesado.

- Tiempo de flujo (*Flowtime, F_j*): mide el intervalo de tiempo que el trabajo está en el entorno, desde su fecha de llegada (r_j), hasta que termina de ser procesado (C_j).

$$F_j = C_j - r_j \tag{2.1}$$

- Retraso (*Lateness, L_j*): cuantifica las unidades de tiempo que transcurren entre su fecha de entrega (d_j) hasta el instante de finalización (C_j), pudiendo tomar valores negativos cuando se complete el trabajo antes de su fecha de entrega.

$$L_j = C_j - d_j \tag{2.2}$$

- Tardanza (*Tardiness, T_j*): mide las unidades de tiempo transcurridas cuando existe retraso en la entrega de un trabajo. Si el trabajo termina de ser procesado antes de su fecha de entrega, la tardanza tomará un valor de 0.

$$T_j = máx\{0, L_j\} = máx\{0, C_j - d_j\} \tag{2.3}$$

- Adelanto (*Earliness, E_j*): al contrario que en la medida anterior, se cuantifican las unidades temporales transcurridas entre el instante de fin de un trabajo y su fecha de entrega, solo cuando el trabajo se completa antes de dicha fecha de entrega.

$$E_j = máx\{0, -L_j\} = máx\{0, d_j - C_j\} \tag{2.4}$$

- Trabajo tarde (*Tardy job, U_j*): marca el número de trabajos que se completan en una fecha posterior a su fecha de entrega, es decir, cuando la tardanza de un trabajo es positiva.

$$\begin{cases} U_j=1 \;\; si\; T_j>0 \quad (C_j<d_j) \\ U_j=0 \;\; en\; caso\; contrario \end{cases} \tag{2.5}$$

Estas medidas pueden formar parte de la función objetivo tanto en formato sumatorio (*p.ej.* $\sum C_j$) como en formato de máximo *(p.ej.* C_{max}). También es común aplicar una ponderación a cualquiera de estos dos formatos para marcar la prioridad/importancia de cada trabajo, haciendo uso de la variable peso (*weight, w_j*).

2.5 Características del entorno Flow-Shop (F_m)

Una vez introducidos algunos de los conceptos relativos a la programación de la producción, a continuación, se procede a explicar de forma más detallada el funcionamiento y las principales particularidades del entorno conocido como *Flow-Shop*, puesto que además de ser uno de los problemas más estudiados en la actualidad, el problema a resolver en este documento se centra en este tipo de entorno.

El taller de flujo regular, conocido comúnmente por el término anglosajón *Flow Shop*, se compone de varias etapas de producción de una sola máquina, como se ha explicado en el apartado 2.2. El número de máquinas, y por tanto el número de etapas, viene definido mediante el subíndice *m*. Los trabajos procesados en este entorno se caracterizan por seguir una misma ruta de fabricación, es decir, todos pasan por las mismas máquinas en el mismo orden, como se muestra en el esquema de la Figura 2.5.

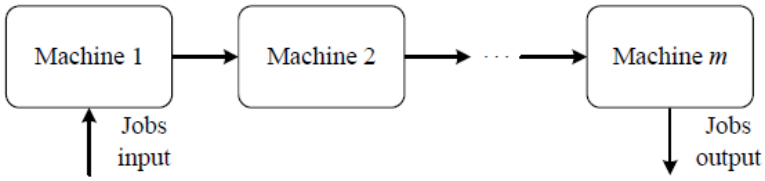

Figura 2.5. Esquema de flujo en un entorno *Flow-Shop*.
[Fuente: Temario Programación y Control de la Producción, Escuela Técnica Superior de Ingeniería]

A continuación, en la Figura 2.6 se presenta un ejemplo del diagrama de Gantt propio de un entorno *Flow-Shop*. En concreto pertenece a un problema de secuenciación de un taller de tipo regular con permutación, conocido como *Permutation Flow Shop Scheduling Problem (PFSP).*

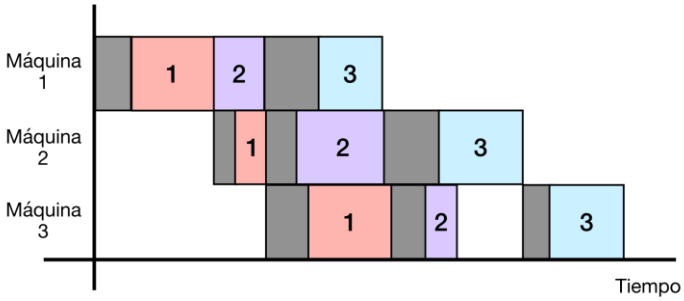

Figura 2.6. Diagrama de Gantt para el PFSP con tiempos de *set-up*. [Fuente: Elaboración propia]

Como su propio nombre indica, el PFSP presenta la restricción de permutación detallada en el apartado 2.3. Cuando un problema presenta esta restricción, como se puede ver en el diagrama de Gantt, la secuencia de procesado de los trabajos es la misma para todas las máquinas. Por ello, la secuencia correspondiente a un programa factible para este tipo de problemas (si no existen restricciones adicionales) se representa como un único vector que contiene el orden de procesado del conjunto de trabajos $N = \{1,\ldots, n\}$, válido para todo el conjunto de máquinas $M = \{1,\ldots, m\}$.

Si no existen limitaciones de capacidad en cuanto a buffers, cuando un trabajo termina de ser procesado en una máquina pasa a formar parte de la cola de espera de la siguiente máquina. Si la máquina esta libre, continúa con el procesado de trabajos en el orden de la cola. En el caso de que no haya ningún trabajo disponible en cola para ser procesado, esa máquina no estará procesando ningún trabajo, produciéndose lo que se denomina tiempo ocioso, conocido en inglés como *Idle time*.

Otro de los aspectos a tener en cuenta, también visible en la Figura 2.6, es la existencia de tiempos de ajuste o preparación de las máquinas, también conocidos como tiempos de *set-up*. Como su propio nombre indica, cuando en una máquina se procesan distintos trabajos, a veces se requiere de un tiempo de ajuste de la maquina como preparación para realizar la próxima tarea, como, por ejemplo, el calentamiento o el cambio de una herramienta. Este tiempo existente entre el procesado de trabajos, puede depender de la máquina, del trabajo en cuestión y de la secuencia de trabajos. Cuando depende simultáneamente de los tres factores mencionados, su notación se corresponde con: S_{ijk}, siendo: $i \in M$ (máquinas); $j, k \in N$ (trabajos).

Además, los tiempos de *set-up* se pueden clasificar en dos tipos: anticipatorios o no anticipatorios. Se habla de tiempos de *set-up* anticipatorios cuando el ajuste de la máquina puede llevarse a cabo mientras el trabajo se está procesando en otra máquina, adelantando ese tiempo (y, típicamente, reduciendo así el tiempo de terminación total). Por el contrario, para los no anticipatorios, el ajuste de la máquina se comienza a realizar una vez el trabajo ha terminado de ser procesado en la máquina anterior y está listo para ser procesado en la maquina en la que se va a hacer el ajuste. La diferencia entre ambos tipos puede verse representada en el ejemplo de la siguiente Figura 2.7.

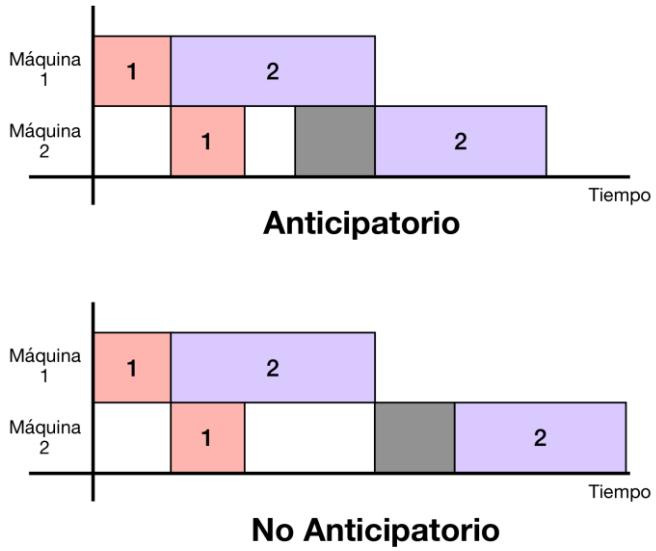

Figura 2.7. Representación diagrama de Gantt: tiempos Anticipatorios y No Anticipatorios.
[Fuente: Elaboración propia]

Los modelos de programación de la producción pueden clasificarse según su complejidad como: polinomiales (*P*) y no polinomiales (*NP-hard*). Los de clase *P* son de menor complejidad, y por tanto su resolución es más simple; pero en cambio los de clase *NP-hard* son más complejos, y es poco probable encontrar una solución óptima para una instancia realista en un tiempo polinomial. Dentro de esta última clase, Pinedo (2008) distingue entre los *NP-hard* en sentido débil (*ordinary sense*) y los *NP-hard* en sentido fuerte (*strongly sense*), siendo los de sentido fuerte más difíciles de resolver de forma exacta que los de sentido débil, para los que existen algoritmos capaces de resolverlos en un tiempo pseudo-polinomial.

El problema de *Flow-shop* pertenece generalmente a los de tipo *NP-hard* debido a su nivel de complejidad, ya que resolverlo de forma óptima implica evaluar $(n!)^m$ posibles programas para una instancia de *n* trabajos y *m* máquinas. Por tanto, la adición de la restricción S_{ijk} (existencia de tiempos de *set-up* dependientes de la máquina y la secuencia) en un entorno *Flow-shop* aumenta en gran medida la complejidad del problema, no siendo (de manera general) posible encontrar la solución óptima en un tiempo razonable. Para su resolución se suele optar por emplear métodos aproximados, con el objetivo de alcanzar una solución que se aproxime a la solución óptima en la mayor medida posible, en un tiempo de cómputo admisible y razonable.

El problema sobre el que se va a desarrollar el presente documento trata de un caso de programación de operaciones con las siguientes características: $F_m|prmu, S_{ijk}|C_{max}$. En el siguiente capítulo se comentarán sus detalles en profundidad.

3 ANÁLISIS DEL PROBLEMA

Como se ha mencionado en el punto anterior, el problema a resolver en este documento se corresponde con un entorno *Flow-shop* con restricción de permutación, donde las máquinas requieren tiempos de *set-up* (dependientes de la secuencia), y se busca como objetivo un programa que proporcione el mínimo *makespan*. En este apartado se describirán en profundidad todos los aspectos relativos al problema.

3.1 Descripción del problema

La notación del problema a resolver es la siguiente: $F_m|prmu, S_{ijk}|C_{max}$.

En primer lugar, en el campo del parámetro α, correspondiente al entorno de fabricación, se tiene un taller de flujo regular, conocido como *Flow-shop*. Como se ha comentado con anterioridad en el capítulo 2.5, este tipo de entorno se compone de *m* etapas, donde cada etapa se corresponde con una sola máquina, y todos los trabajos objeto de las operaciones realizadas en las máquinas siguen la misma ruta de fabricación predeterminada. Así, para un conjunto de trabajos $N = \{1,..., n\}$ a procesar en un entorno F_m compuesto por el conjunto de máquinas $M = \{1,..., m\}$, cada uno de los *n* trabajos se procesarán una vez en cada una de las *m* máquinas.

En segundo lugar, en cuanto a las restricciones del problema correspondientes al parámetro β, se presentan la restricción de no permutación (*prmu*) y la restricción de tiempos de *set-up* (S_{ijk}), además de todas las suposiciones generales para $\beta = 0$. No se considerarán interrupciones de los procesos u otras restricciones que no se especifiquen. (Véase sección 2.3)

En último lugar, en cuanto al parámetro γ, el problema tiene como objetivo la minimización del *makespan*, por tanto se buscará que el conjunto de tareas se procese completamente en el mínimo tiempo posible. No será necesario tener en cuenta datos de entrada relativos a fechas de llegada al sistema de los trabajos (*release dates,* r_j) o fechas de entrega de estos (*due dates,* d_j).

El *makespan* o C_{max} mide el tiempo de terminación máximo del conjunto de trabajos. Cada trabajo se inicia y se termina de procesar en una máquina en un determinado instante de tiempo, siendo ese intervalo transcurrido el correspondiente a su tiempo de proceso (p_{ij}). Al instante de finalización del trabajo *i* en la máquina *j* se le denomina *completion time* (C_{ij}). En este tipo de problema, el C_{max} se corresponde con el *completion time* de la última tarea de la secuencia procesada en la última máquina de la ruta de fabricación. Tendrán influencia en la magnitud del C_{max} tanto los tiempos de proceso (p_{ij}), como los tiempos de *set-up* (S_{ijk}) y el tiempo ocioso (*idle time*). Este último se corresponde con el intervalo de tiempo que se da en una máquina cuando termina de procesar un trabajo y tiene que esperar a que le llegue el próximo trabajo.

En el siguiente diagrama de la Figura 3.1 se muestra un sencillo ejemplo del cálculo correspondiente a la función objetivo del problema (C_{max}) para un conjunto de 3 trabajos procesados en un *Flow-shop* de 2 máquinas ($F_2|prmu, S_{ijk}|C_{max}$). Además, se señalan los intervalos de tiempo correspondientes a los tiempos de *set-up*, *idle-time*, y tiempos de proceso (p_{ij}).

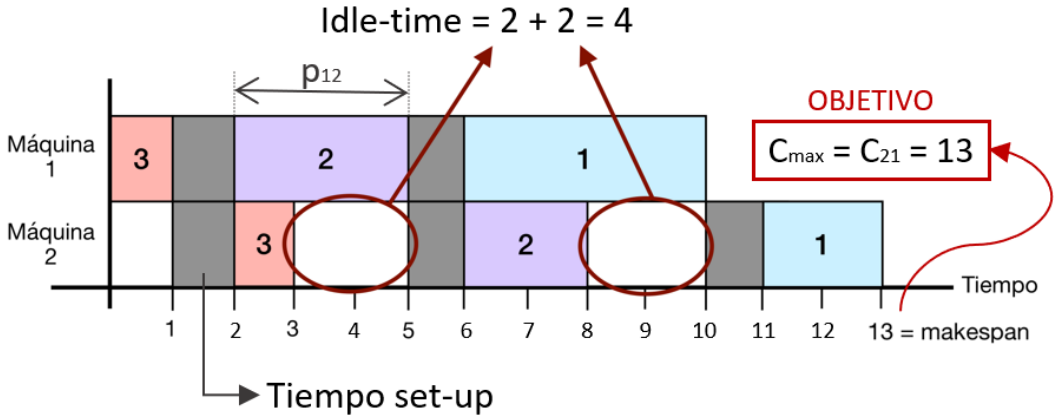

Figura 3.1. Ejemplo de diagrama de Gantt para $F_2|prmu, S_{ijk}|C_{max}$. [Fuente: Elaboración propia]

3.2 Complejidad del problema

Como se comentó en el capítulo anterior, este tipo de problema pertenece a los conocidos como *NP-hard*. Por tanto, no es posible determinar el programa óptimo para este problema en un tiempo admisible, por lo que no se puede tener la certeza de que el programa dado como solución será mejor que cualquier otro programa admisible.

Para la resolución de este problema se desarrollará un algoritmo que busque el vector secuencia que contenga la combinación de trabajos a procesar ordenados de forma que se obtenga el mínimo valor posible del *makespan* en un tiempo acorde. No se conocerá cuál sería el programa óptimo, ni cuánto se aproxima el valor de la función objetivo de la secuencia obtenida como solución en el algoritmo, al valor de la función objetivo correspondiente al programa óptimo.

3.3 Notación

A modo recopilatorio, se presenta la siguiente Tabla 3.1, que contiene las notaciones mencionadas con anterioridad que se usarán a lo largo del documento.

Tabla 3.1. Notación utilizada.

Notación	Descripción
n	Número de trabajos a procesar
m	Número de máquinas consideradas
p_{ij}	Tiempo de proceso del trabajo j en la máquina i
S_{ijk}	Tiempo de *set-up* para el trabajo k, con el trabajo j como predecesor, en la máquina i
Idle time	Tiempo ocioso de las máquinas durante el procesado de trabajos
C_{ij}	Tiempo de terminación del trabajo j en la máquina i
C_{max}	Instante de terminación del último de los trabajos
F_m	*Flow-shop* de m máquinas
prmu	Restricción de permutación
FO	Función objetivo
Π	Secuencia de trabajos

4 ANÁLISIS DE LA METODOLOGÍA

El problema a abordar, al ser un problema de programación de la producción, es de tipo combinatorio, por lo que teóricamente evaluando todas las combinaciones posibles que componen el conjunto finito de soluciones sería suficiente para determinar cuál de ellas es la óptima para el objetivo fijado. Cuando se trata de problemas complejos o de gran tamaño cuyo conjunto finito de soluciones tiene un tamaño elevado, no es viable emplear este método para su resolución, ya que obtener la solución óptima implicaría un gran tiempo de cálculo.

Por este motivo, para su resolución se emplean métodos de optimización que describen el procedimiento a seguir para acelerar la búsqueda de soluciones. Es muy común hacer uso de algoritmos en los que se codifica dicho procedimiento con un número finito de pasos y se ejecuta mediante un ordenador. Se diferencia entre algoritmos exactos, que alcanzan el óptimo del problema, y algoritmos aproximados, que proporcionan una solución aproximada. En este capítulo se comentarán algunos de los métodos generales empleados para la resolución de problemas de programación de la producción.

4.1 Procedimientos generales de resolución

4.1.1 Métodos exactos

Los métodos exactos proporcionan la solución óptima del problema, ya que evalúan todas las combinaciones posibles del modelo, explícita o implícitamente.

En cuanto a algoritmos exactos, se puede distinguir entre:

- Algoritmos constructivos exactos. Resuelven problemas polinomiales, de clase *P*, evaluando todas las soluciones del modelo para obtener la óptima de ellas.

  Pertenecen a este grupo, entre otros, los conocidos algoritmos de Johnson, Lawler, Moore, y algunas reglas de despacho que especifican el orden de procesado de los trabajos, para problemas de programación muy sencillos, como por ejemplo:

  - *First In First Out, FIFO*: el primer trabajo que llega es el primero en ser procesado. Se obtiene así el óptimo para el problema $1|r_j|C_{max}$.
  - *Shortest Processing Time first, SPT*: el orden de procesado de los trabajos es de menor a mayor tiempo de proceso. Esta regla de despacho resuelve de forma óptima los problemas: $1||\sum C_j$ y $P_m||\sum C_j$.
  - *Earlier Due Date first, EDD*: se procesan los trabajos en orden de menor a mayor fecha de entrega o due date. Obtiene el óptimo para los problemas $1||máx\, T_j$ y $1||máx\, L_j$.
  - *Weighted Shortest Processing Time first, WSPT*: el orden en el que se procesan los trabajos viene dado por el valor del cociente entre el peso y tiempo de proceso del trabajo que se va a procesar, ordenados de mayor a menor. Proporciona el óptimo para el problema $1||\sum w_i C_j$.
  - *Shortest Remaining Processing Time on the Fastest Machine first, SRPT-FM*: el trabajo de menor duración se procesa primero en la máquina más rápida, y el resto se comienzan a procesar en el resto de las máquinas disponibles. Cuando la máquina más rápida se queda libre, se analiza de nuevo cuál es el trabajo con menor tiempo de proceso restante, asignándolo de nuevo a la máquina más rápida, y así sucesivamente hasta completar todos los trabajos. Con esta regla se obtiene el óptimo para el problema $Q_m|pmtn|\sum C_j$.

- Algoritmos enumerativos. Aunque no evalúan todas las soluciones explícitamente, garantizan la obtención de la solución óptima. Las soluciones que no son evaluadas explícitamente se evalúan implícitamente, ya que existen evidencias que indican que la evaluación de esta implicaría un valor de la función objetivo igual o superior al de otra solución ya evaluada. Al ser de tipo No polinomial, sólo dan soluciones óptimas para problemas con instancias pequeñas.

  Pertenecen a este grupo la programación matemática (en concreto los modelos de programación lineal entera mixta), el método de ramificación y acotación conocido como *Branch and Bound*, y la programación dinámica.

4.1.2 Métodos aproximados

Debido a la inviabilidad de hallar una solución óptima mediante un método exacto en un tiempo razonable, existen los métodos aproximados. Con ellos no existe certeza de que la solución alcanzada sea la óptima, ya que no se evalúa la totalidad del conjunto de soluciones posibles, sino parte de él, consiguiendo una buena solución en un tiempo aceptable.

4.1.2.1 Heurísticas

"Las heurísticas son procedimientos simples, a menudo guiados por el sentido común, que están destinados a proporcionar soluciones buenas pero no necesariamente óptimas a problemas difíciles, de manera fácil y con rapidez." [Zanakis y Evans, 1981]

Son de gran utilidad para problemas que no requieran una solución óptima o en los que no se puedan aplicar métodos exactos por otros motivos. También suelen emplearse como paso previo a la aplicación de otro algoritmo o cuando los datos del problema tienen una alta variabilidad o no son fiables.

Una de las heurísticas más usadas en la actualidad por su efectividad en la resolución del problema de *Flow-shop* es la NEH propuesta por (Nawaz, Enscore Jr & Ham, 1983). Se trata de una heurística constructiva que consiste en construir la solución en un número finito de pasos, insertando uno a uno los trabajos aún sin secuenciar en la posición de la secuencia parcial que menor valor de la función objetivo proporcione.

4.1.2.2 Metaheurísticas

Cuando se abordan problemas complejos para los que las heurísticas resultan ineficientes e inefectivas, se hace uso de las metaheurísticas. Se tratan de procedimientos heurísticos adecuados para realizar una búsqueda de soluciones eficaz y efectiva, encontrando el equilibrio adecuado entre intensificación y diversificación.

Estas dos características de la búsqueda son muy importantes. Por un lado, la intensificación está relacionada con la explotación del espacio en la búsqueda local, centrándose en las que aparentan ser más prometedoras, consiguiendo así soluciones de mayor calidad. Por otro lado, la diversificación, consigue una apropiada exploración en la búsqueda global, abarcando un espacio mayor de soluciones variadas mediante la búsqueda en regiones que contrastan fuertemente con las regiones exploradas hasta el momento, no aleatoriamente, sino teniendo en consideración el proceso de búsqueda anterior.

Es fundamental buscar el equilibrio correcto entre ambas características en el proceso de diseño de una metaheurística efectiva, ya que una falta de intensificación provoca que la exploración sea demasiado larga, y una falta de diversificación produce una rápida convergencia a mínimos locales.

Los métodos metaheurísticos pueden presentar los siguientes enfoques:

- Enfoque constructivo, basado en generar la solución paso a paso, tomando una decisión cada vez que se secuencia un trabajo, disminuyendo así el tamaño del problema en cada decisión al descartar posibles soluciones.
- Enfoque iterativo, que realiza una búsqueda local en cada iteración, estudiando las soluciones del entorno de la solución que realiza. Este entorno se denomina vecindario, y se corresponde con el conjunto de soluciones formadas a partir de una modificación de una solución. Uno de los métodos más conocidos de este tipo de enfoque es la conocida búsqueda Tabú (*Tabú Search*).

- Enfoque evolutivo, inspirado en mecanismos biológicos como son la selección y evolución natural, genera nuevas soluciones seleccionando, cruzando, o mutando las soluciones existentes en un conjunto denominado población inicial, obteniendo así soluciones de mayor calidad "evolucionadas" a partir de las existentes.

4.2 Metodología seleccionada

Para el problema a resolver, $F_m|prmu, S_{ijk}|C_{max}$, se ha decidido emplear métodos aproximados, ya que como se mencionó, su solución óptima no se puede alcanzar en un tiempo razonable debido a su complejidad y número elevado de programas factibles. En concreto, se pretenden construir 4 algoritmos aproximados basados en dos de los métodos mencionados en el apartado 4.1.2: la Búsqueda tabú y la heurística NEH.

Además, se hará uso también de una de las reglas de despacho: *Longest Processing Time first* (LPT), que consiste en procesar los trabajos según su tiempo de proceso, de mayor a menor, al contrario que en la SPT.

4.2.1 NEH

En cuanto a la heurística constructiva NEH, a pesar de que rara vez se consigue obtener el óptimo con este procedimiento, proporciona muy buenos resultados para el problema $F_m|prmu|C_{max}$, y aporta como ventajas su eficacia, sencillez y rapidez en la búsqueda de soluciones.

Según su procedimiento de aplicación en un taller de flujo regular, en primer lugar se suman para cada uno de los trabajos, todos sus tiempos de proceso en todas las máquinas donde se procesan, y posteriormente se ordenan de mayor a menor. En función de estos tiempos totales de proceso de cada trabajo, se aplica la regla LPT, obteniendo así la secuencia de trabajos ordenada según este criterio. A continuación, comienza la fase de inserción, donde progresivamente se va construyendo la secuencia, incorporando uno a uno los trabajos en el orden previamente marcado por la regla LPT. Para ello se crea inicialmente una secuencia inicial compuesta por la mejor combinación entre los primeros dos trabajos, y se van insertando sucesivamente trabajos en la posición de la secuencia parcial generada que proporcione un mejor valor de la función objetivo, hasta secuenciar todos los trabajos. A continuación, en la Figura 4.1 se muestra el pseudocódigo correspondiente a este procedimiento.

Entrada: datos de la instancia
Salida: Π, C_{max}
Inicio
  $P_{\gamma_j} = \sum_{i=1}^{m} p_{i\gamma_j}$;
  $\Pi = \emptyset, J = \{\gamma_1, \ldots, \gamma_n\}$ verificándose que $P_{\gamma_1} \geq P_{\gamma_2} \geq \ldots P_{\gamma_n}$;
  $\Pi^{(1)} = \{\gamma_1\}$;
  Para $k = 2$ **hasta** n **hacer**
    Insertar el trabajo γ_k en la posición de $\Pi^{(k)}$ que conlleve el menor C_{max};
  Fin
  Devolver Π, C_{max}
Fin

Figura 4.1. Pseudocódigo de NEH para $F_m|prmu|C_{max}$. [Fuente: Elaboración propia]

Si bien es cierto (como muestran Companys y Rivas, 2012), este procedimiento no siempre mejora la calidad de la secuencia, por lo que es recomendable evaluar siempre la secuencia inicial antes para compararla con la solución resultante del método y quedarse con la mejor, lo cual se tendrá en cuenta en el desarrollo de los algoritmos.

A continuación se presenta un breve y sencillo ejemplo numérico de la aplicación este procedimiento para el problema a resolver descrito en el apartado 3.1 ($F_m|prmu, S_{ijk}|C_{max}$).

Ejemplo 4.2.1. *Hallar una solución aproximada mediante la heurística NEH para la siguiente instancia en el problema* $F_2|prmu, S_{ijk}|C_{max}$.

Número de trabajos: n = 3

Tiempos de proceso (p_{ij})*:*

m \ n	1	2	3
1	2	3	1
2	2	2	1

Tiempos de set-up (s_{ijk})*:*

Máquina 1 (i = 1):

j \ k	1	2	3
0	1	2	0
1	-	2	2
2	3	-	1
3	2	1	-

Máquina 2 (i = 2):

j \ k	1	2	3
0	0	1	1
1	-	3	1
2	1	-	1
3	1	3	-

Fase 1: Secuencia inicial LPT.

$$P_j = \sum_{i=1}^{m} p_{ij}$$

$P_1 = 2+2 = 4 \rightarrow 2^o$

$P_2 = 3+2 = 5 \rightarrow 1^o$

$P_3 = 1+1 = 2 \rightarrow 3^o$

Secuencia LPT = [2, 1, 3]

Fase 2: inserción del trabajo seleccionado en la posición con menor C_{max}.

[1, 2] → C_{max} = 13

[2, 1] → C_{max} = 13

Como se puede observar, ambas combinaciones proporcionan el mismo resultado en cuanto al objetivo del problema (C_{max} = 13). Por ello, en caso de empate, en las propuestas de resolución que se presentarán en el siguiente capítulo se analizará un método de desempate que evalúa el tiempo ocioso entre el procesado de los trabajos en las máquinas, usándolo como criterio para decidir qué combinación se elige en caso de empate. En este ejemplo, el *idle-time* de la primera combinación [1, 2] es de 3 unidades temporales, mientras que para la segunda combinación [2, 1] es de 2 unidades. La construcción de la secuencia se continuará con esta última secuencia parcial.

[3, 2, 1] → C_{max} = 13 ← Menor valor, solución del problema.

[2, 3, 1] → C_{max} = 14

[2, 1, 3] → C_{max} = 15

Gráficamente se puede ver la solución obtenida representada en el diagrama de Gantt de la *Figura 4.2*.

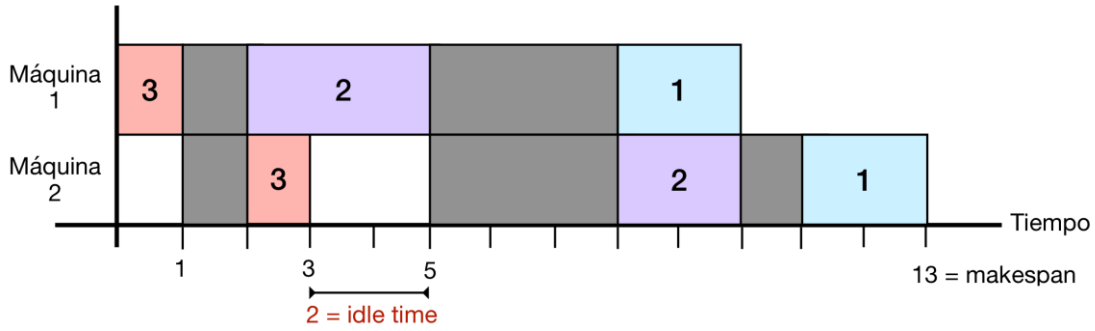

Figura 4.2. Diagrama Gantt resultado de la heurística NEH en el ejemplo 4.2.1. [Fuente: Elaboración propia]

4.2.2 Búsqueda Tabú

La conocida metaheurística denominada búsqueda tabú, es una técnica de búsqueda local basada en la memoria que proporciona muy buenos resultados en problemas de optimización debido a su versatilidad. Se considera muy eficiente en la resolución de problemas combinatorios complejos, como es el conocido problema del viajero, problemas de rutas y envíos de productos, y por supuesto, problemas de secuenciación.

Como se ha comentado en el apartado 4.1.2.2, el enfoque de este método es iterativo, ya que prohíbe determinados movimientos durante un número de iteraciones, consiguiendo así evitar que el procedimiento caiga en óptimos locales, lo cual presenta una favorable ventaja a la hora de seleccionar esta metodología.

Inicialmente, la búsqueda tabú parte de una solución, generando otra nueva a partir de esta. Para ello transforma la solución inicial mediante la realización de movimientos, que pueden consistir en retirar de la secuencia un trabajo ya secuenciado y colocarlo en otra posición distinta, o simplemente intercambiar dos de los trabajos secuenciados. De esta forma puede crearse una solución que incluso será de menor calidad que la actual, pero gracias a ello se consigue escapar de óptimos locales y poder alcanzar el óptimo global del problema. Véase Figura 4.3.

Para evitar probar ciclos de soluciones poco diversas, se define un conjunto de movimientos tabú (prohibidos), registrados en la denominada lista tabú. El tamaño de este conjunto viene definido por un parámetro denominado tamaño de la lista. En el caso en que el número de movimientos contenidos en la lista alcance el tamaño total de la lista tabú, antes de añadir un nuevo movimiento a la lista hay que eliminar uno de los que ya contiene. Para decidir qué movimiento se elimina, se establecen criterios de aspiración por defecto, donde se elimina el más antiguo (el movimiento que lleve más tiempo en la lista), o criterios de aspiración por objetivo, donde solo se admite el movimiento si produce una solución mejor a la obtenida hasta el momento. Este último suele ser el criterio mayoritariamente utilizado.

La Figura 4.4 muestra el diagrama de flujo de la búsqueda tabú, donde *MS* representa la mejor solución hasta el momento, y *AS* la actual solución durante el proceso de búsqueda.

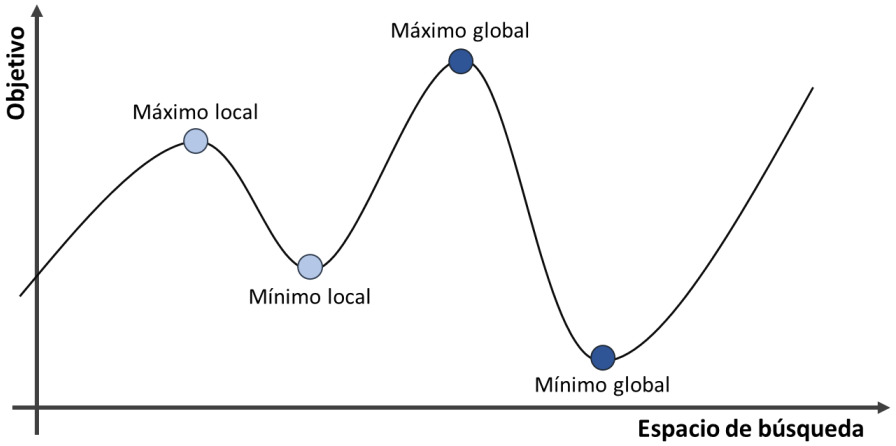

Figura 4.3. Representación mínimos y máximos (locales y globales). [Fuente: Elaboración propia]

Es importante encontrar el equilibrio correcto entre intensificación y diversificación, como se comentó en el apartado 4.1.2.2. En la búsqueda tabú generalmente se presenta un equilibrio estático, ya que cuando mayor sea el tamaño de la lista, se producirá una mayor diversificación, pero una menor intensificación. Es recomendable que el equilibrio sea oscilatorio entre ambas, por lo que una posible forma de conseguirlo sería cambiar dinámicamente el tamaño de la lista tabú.

Basándose en esta metaheurística, la búsqueda tabú, para la resolución del problema en cuestión se ha decidido emplear el procedimiento inverso en cuanto a los movimientos introducidos en la lista. Como se va a explicar en el siguiente capítulo, los trabajos que se introducen en la lista tabú dejan de ser movimientos prohibidos, y pasan a ser movimientos prometedores, pasando a denominarse la lista prometedora (*promise*). Así, se mantendrán en la lista los movimientos más prometedores para volver a repetirlos en futuras iteraciones de la búsqueda, como proponen Fernández-Viagas, Molina-Pariente y Framiñán (2018). Con esto se consigue mantener las combinaciones que en un principio fueron descartadas por existir otra combinación que diese mejores resultados, pero en otro momento más avanzado de la búsqueda puedan proporcionar mejoras.

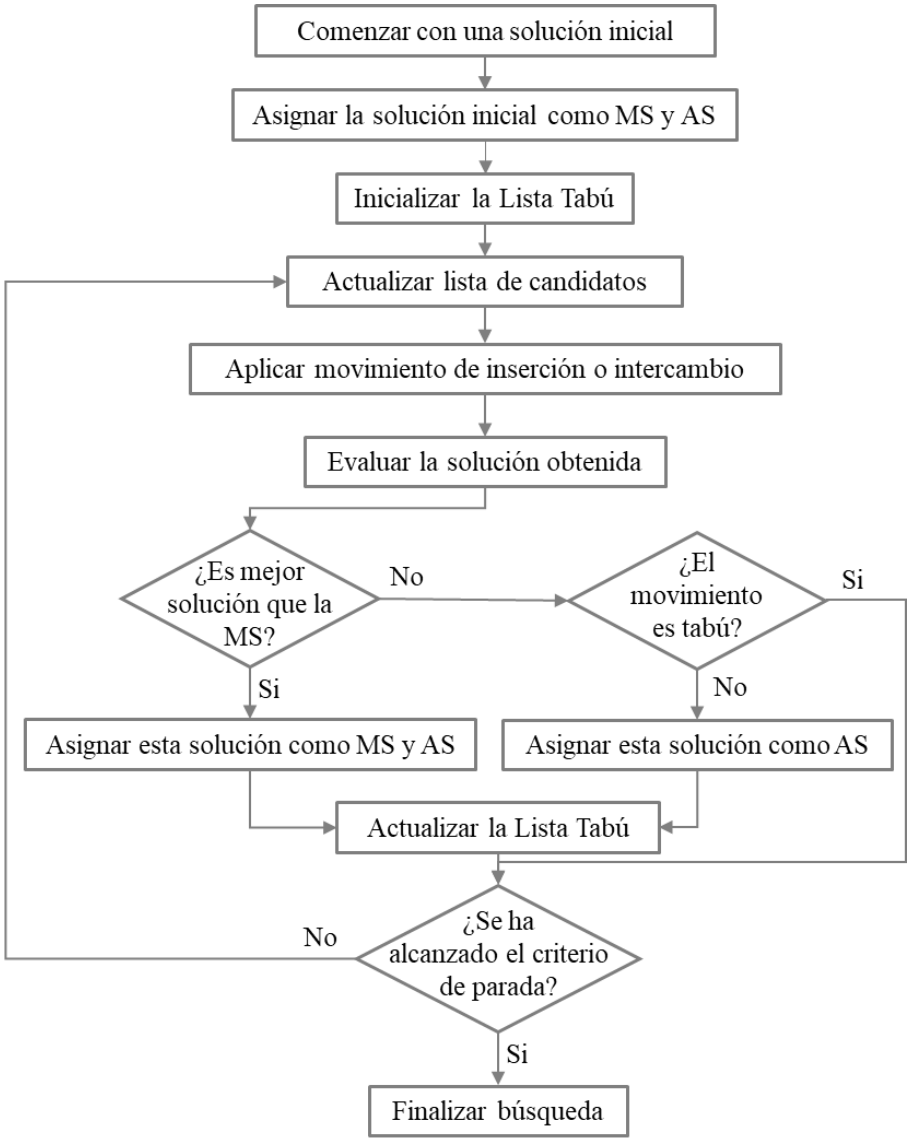

Figura 4.4. Diagrama de flujo para la búsqueda tabú. [Fuente: Adaptación al español de Abiri et al. (2009)]

5 APLICACIÓN DE LA METODOLOGÍA

Una vez explicada la metodología que se ha tenido como base para la resolución del problema, se procede a comentar su aplicación. A continuación se explicarán detalladamente cada uno de los 4 procedimientos de resolución propuestos, para los cuales se ha implementado el algoritmo aproximado correspondiente en lenguaje de programación C# en el entorno de desarrollo Microsoft Visual Studio Community en su versión 2019.

5.1 Procedimiento propuesto

Los métodos aproximados propuestos para resolver el problema se tratan de heurísticas constructivas basadas en la memoria, ya que están fundamentados en la combinación de la heurística NEH y una versión inversa de la metaheurística de búsqueda tabú (véase sección 4.2).

Como se ha comentado en capítulos anteriores, la heurística constructiva NEH se guía únicamente por el criterio de minimización de la función objetivo para determinar cuál es la mejor secuencia parcial para continuar el proceso. Este criterio puede no ser fiable en los primeros pasos de la construcción donde la secuencia parcial está compuesta por un número bajo de trabajos, pudiendo dar lugar a elecciones erróneas. Por tanto, la adición de la búsqueda local desde un enfoque iterativo a la heurística constructiva se considera muy conveniente. Gracias a ella se vuelven a evaluar esas soluciones que han podido ser descartadas inicialmente por no ser las que mejor valor de la función objetivo han proporcionado, pero podrían serlo en una fase más avanzada del proceso de construcción de la secuencia. Además, la búsqueda local ayuda a escapar de óptimos locales ampliando la búsqueda, lo cual también es ventajoso.

En base a lo anterior, las 4 versiones propuestas a continuación siguen un procedimiento común que consiste en construir la secuencia del mismo modo que la heurística NEH, incluyendo además en cada paso de dicha construcción (cada vez que se inserta un nuevo trabajo en la secuencia parcial) una búsqueda local a partir la vecindad de dicha solución parcial. Esta búsqueda se realiza generando vecinos mediante la reinserción de un trabajo en una posición distinta a la actual. Dicha posición viene determinada por una lista de movimientos prometedores, que al contrario que la lista tabú, contiene los movimientos (combinación de dos trabajos) que en iteraciones anteriores proporcionaban buenos resultados en cuanto a la función objetivo (5.1). Con esta función se tiene en cuenta principalmente el objetivo del problema (C_{max}), pero además, en caso de empate en cuanto al valor del *makespan* de dos soluciones distintas, se tomará como criterio la minimización del tiempo ocioso (*idle-time*). El valor de N en la ecuación se corresponde con un número suficientemente alto.

$$FO = C_{\max} + \frac{idle - time}{N} \qquad (5.1)$$

En la siguiente Figura 5.1 se muestra un diagrama de flujo que representa el procedimiento general común en las 4 versiones presentadas, que varían en cuanto a la configuración de la lista de movimientos prometedores que se va a explicar a continuación. Además, cada una de las versiones tiene como entrada un parámetro que determinará la cantidad y calidad de los movimientos que se mantendrán y reinsertarán en el proceso de búsqueda. Este procedimiento inicialmente requiere de una solución inicial, la cual será dada por la regla LPT, que como se ha comentado con anterioridad, consiste en ordenar los trabajos decrecientemente según su tiempo de proceso total (suma de su tiempo de proceso en todas las máquinas que componen su ruta de fabricación). Esta secuencia inicial LPT determina el orden en el que se secuencian los trabajos, construyéndose una secuencia parcial con los trabajos de mayor duración en primer lugar, e insertando sucesivamente entre ellos los trabajos que implican un menor tiempo de proceso en cada paso de la construcción. Este método se conoce que proporciona buenos resultados para el problema de *Flow-shop*.

Por tanto, el trabajo en primera posición de la secuencia inicial LPT formará la secuencia parcial a la que se irán añadiendo uno a uno los trabajos de las siguientes posiciones de dicha secuencia inicial. El siguiente trabajo se insertará en cada una de las posiciones de la secuencia parcial, evaluando el valor resultante de la función objetivo para cada posición. La secuencia elegida para continuar con el procedimiento será la que aporte un menor valor de la función objetivo, mientras que el resto de las combinaciones se guardan en una lista (cuáles y cuántas dependerá de cada versión propuesta). Tras cada nueva inserción se volverán a repetir dichas combinaciones de la lista, actualizando la secuencia parcial cada vez que se muestre una mejora respecto al objetivo marcado: la minimización de la función objetivo. Finalmente, la búsqueda termina una vez que todos los trabajos que conforman la secuencia inicial LPT hayan sido secuenciados, obteniendo una secuencia completa como solución del problema.

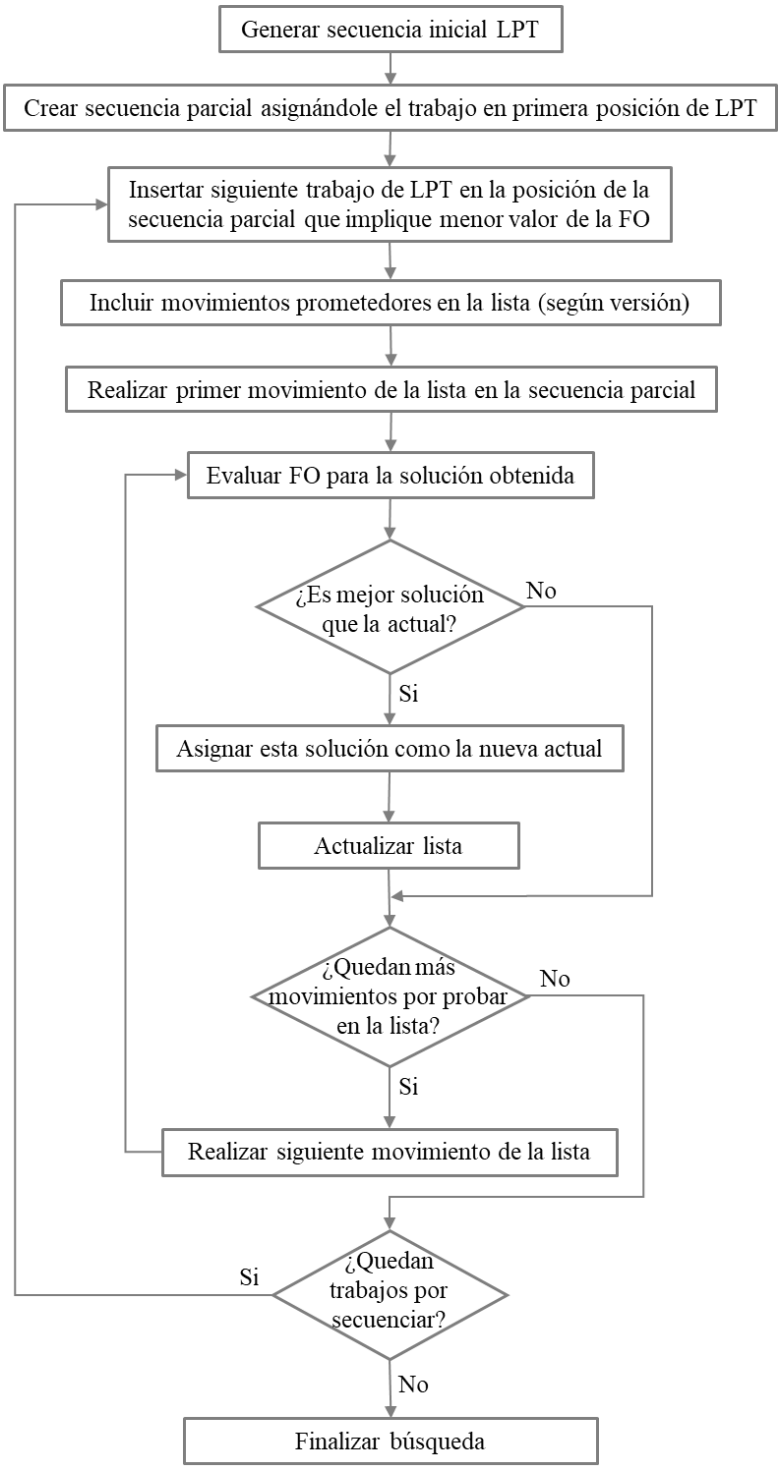

Figura 5.1. Diagrama de flujo del procedimiento general propuesto. [Fuente: Elaboración propia]

5.2 Versión 1 (V.1)

La primera de las versiones propuestas es similar a una adaptación de la heurística MCH (*Memory-based Constructive Heuristic*) propuesta para talleres de flujo regular híbridos (varias etapas con varias máquinas) por Fernández-Viagas, Molina-Pariente y Framiñán (2018). En su procedimiento, tras cada inserción de un nuevo trabajo en la secuencia parcial, se vuelven a evaluar un porcentaje x de los movimientos más prometedores resultantes en el paso de construcción anterior, siendo $x \in [0, 1]$ medido en tanto por 1. Esta constante de proporcionalidad x será un parámetro de entrada del algoritmo.

Para ello, tras insertar el trabajo j ($j \in \{2,..., n\}$) de la secuencia inicial LPT en la secuencia parcial, se deshace la inserción realizada anterior a esta, retirando de la secuencia parcial el trabajo ($j - 1$) para seguidamente insertarlo en la posición designada por el movimiento prometedor s de la fila de la lista correspondiente a la iteración anterior ($r_s^{(k-1)}$). El valor de $s \in \{1,..., S(x)\}$ tiene como máximo $S(x)$, que determina de forma dinámica el número de movimientos prometedores a evaluar en cada paso de la construcción de la secuencia en función del tamaño de la secuencia parcial en dicho instante ($S(x) = k*x$). Su tamaño incrementa linealmente, ya que el tamaño de la secuencia parcial (k) es mayor en cada iteración.

En la Figura 5.2 se muestra a modo de ejemplo una matriz con movimientos prometedores, y las posibles combinaciones que se repetirían según los valores del parámetro x. Cada fila de la matriz se corresponde con cada paso de construcción de la secuencia (inserción del siguiente trabajo sin secuenciar) y contiene los posibles movimientos a repetir, que son aquellos que han sido descartados en esa iteración. En el segundo paso por ejemplo, se repetirían el 100% de los movimientos del paso anterior correspondiente al color rojo ($x = 1$). En el tercer paso se repetirían solo la mitad ($x = 0.5$) del paso anterior (color verde), y tras la inserción del cuarto trabajo en la secuencia parcial (cuarta fila de la matriz) se probarían el 100% de los movimientos más prometedores del paso anterior (color azul).

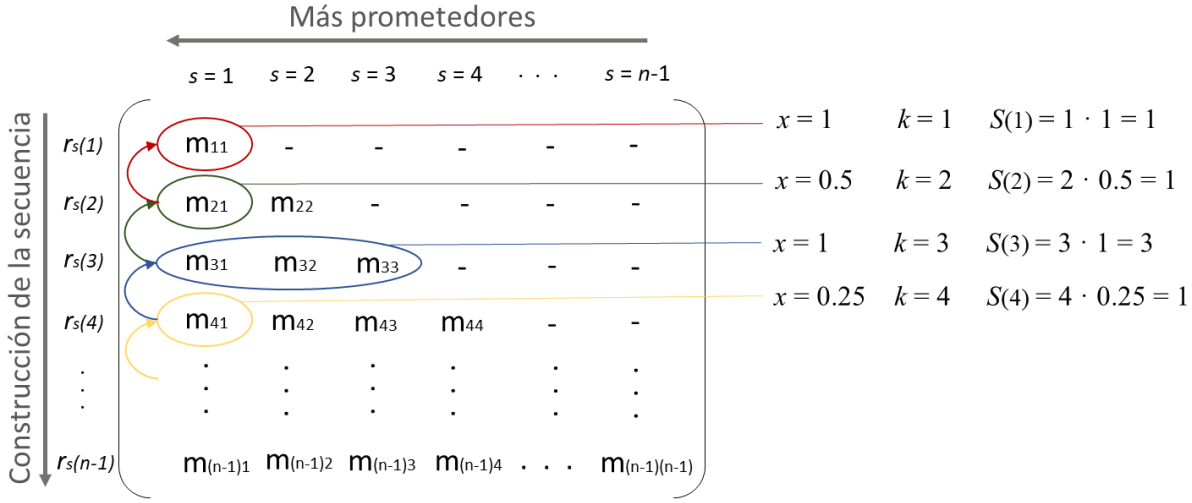

Figura 5.2. Matriz de movimientos prometedores para V.1. [Fuente: Elaboración propia]

De esta forma, cada vez que uno de los trabajos reinsertados proporcione una mejora en cuanto al objetivo fijado (minimización de la ecuación 5.1), la secuencia parcial se actualizará. Esta tomará el valor de la recién construida, siendo la nueva mejor secuencia parcial hasta el momento, a partir de la cual se continuará el proceso de búsqueda. Este procedimiento sigue el funcionamiento de la búsqueda local de tipo "*First Best*", ya que nos quedamos con la mejor solución hasta ese instante y seguimos buscando a partir de ella, formando un vecindario variable.

En la Figura 5.3 se muestra el pseudocódigo correspondiente a esta primera versión (V.1) del procedimiento.

Entrada: datos de la instancia, parámetro x
Salida: Π_b, Obj_b
Inicio
  $P_{\gamma_j} = \sum_{i=1}^{m} p_{i\gamma_j}$;
  $J = \{\gamma_1, \ldots, \gamma_n\}$ verificándose que $P_{\gamma_1} \geq P_{\gamma_2} \geq \ldots P_{\gamma_n}$;
  Asignar a *Obj* el valor de la función objetivo para *J;*
  $\Pi^{(1)} = \{\gamma_1\}$;
  Para $k = 2$ **hasta** n **hacer**
    $S^{(k)} = k \cdot x$;
    Insertar el trabajo γ_k en todas las posibles posiciones j de $\Pi^{(k-1)}$, con $j \in \{1, \ldots, k\}$;
    $\Pi^{(k)}$ = secuencia obtenida insertando γ_k en la posición de $\Pi^{(k-1)}$ con menor valor de la función objetivo (Obj_b);
    $r_s^{(k)}$ (con $s \in \{1, \ldots, S^{(k)}\}$) = trabajo anterior a γ_k , tras evaluar γ_k en la s-ésima posición con menor Obj_b;
    Si $k > 2$ **entonces**
      Para $s = 1$ **hasta** $S^{(k-1)}$ **hacer**
        $\Pi'^{(k)}$ = permutación obtenida retirando el trabajo $\gamma_{(k-1)}$ de $\Pi^{(k)}$ y reinsertándolo detrás del trabajo $r_s^{(k-1)}$. Asignar a *Obj*' su valor de la función objetivo;
        Si *Obj*' < Obj_b **entonces**
          $\Pi_b^{(k)} = \Pi'^{(k)}$;
          $Obj_b = Obj'$;
        Fin
      Fin
    Fin
  Fin
  Si *Obj* < Obj_b **entonces**
    $Obj_b = Obj$;
    $\Pi_b = J$;
  Fin
  Devolver Π_b, Obj_b
Fin

Figura 5.3. Pseudocódigo procedimiento V.1. [Fuente: Adaptación de Fernández-Viagas et al. (2018)]

5.3 Versión 2 (V.2)

Para la segunda variante de resolución propuesta se parte de la versión anterior, modificando el hecho de que se reinserten los movimientos prometedores correspondientes únicamente a la iteración anterior. En concreto, tras cada inserción de un nuevo trabajo en la secuencia parcial, se volverán a evaluar el porcentaje x de los movimientos más prometedores de los z pasos de construcción anteriores, siendo $x \in [0, 1]$ y $z = \lfloor n \cdot y \rfloor$, donde y es la constante de proporción que define cuántas iteraciones atrás se prueban. Los valores de x e y serán un dato de entrada del algoritmo.

De este modo, en la Figura 5.4 se muestra un ejemplo de los trabajos a repetir en cada iteración según los valores de los parámetros x y z. Como se puede ver, cuando nos encontramos en el tercer paso de construcción de la secuencia parcial (tercera fila de la matriz), se vuelven a probar la mitad de los movimientos más prometedores (x = 0.5) del paso anterior (z = 1), señalados en color rojo. De la misma manera, en la cuarta iteración se probarán el 100% de los movimientos (x = 1) para los 3 pasos anteriores (z = 3) correspondientes al color verde. Y por último, en la quinta iteración se realizarán de nuevo la mitad (x = 0.5) de los movimientos prometedores de cada una de las 2 filas anteriores (z = 2), marcados en azul.

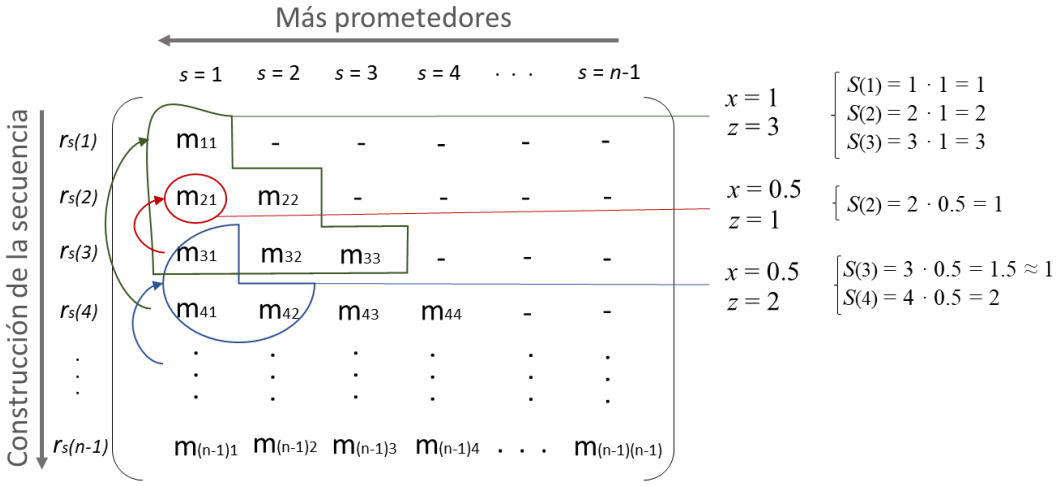

Figura 5.4. Matriz de movimientos prometedores para V.2. [Fuente: Elaboración propia]

Aunque en el ejemplo los valores de x y z varíen en cada iteración, en el algoritmo propuesto el valor de z será constante, repitiendo los movimientos del mismo número de iteraciones anteriores en todo el procedimiento.

El pseudocódigo correspondiente a esta versión (V.2) se presenta en la Figura 5.5.

Al repetir movimientos de iteraciones anteriores en el instante actual, se provoca que los mismos movimientos se vuelvan a evaluar repetidas veces en distintas iteraciones consecutivas. Lo que en principio puede evitar que se pierdan soluciones prometedoras durante la búsqueda, podría conllevar a veces un proceso de búsqueda ineficiente, incrementando notablemente el tiempo de cálculo al evaluar reiteradamente la misma combinación sin producir mejoras en cuanto a calidad de la solución. Por ello, se debe mantener un correcto equilibrio entre diversificación e intensificación de la búsqueda.

Para conseguirlo, al igual que en la versión anterior, es importante ajustar los valores de los parámetros x e y. Si se establece un valor pequeño del parámetro x se introducirían en la lista de movimientos únicamente las combinaciones de trabajos más prometedoras de cada iteración, aumentando la intensificación de la búsqueda, mientras que por el contrario, valores de x próximos a 1 diversificarán la búsqueda permitiendo explorar un espacio mayor de soluciones diversas.

Con esta variante, al probar los mejores movimientos de las z iteraciones anteriores, se intenta solucionar el efecto de la heurística anterior (V.1) en la que únicamente se reinsertaban los trabajos de la iteración anterior en la siguiente iteración próxima a la que pertenece el movimiento prometedor en cuestión. Esto provocaba que las soluciones prometedoras halladas en los primeros pasos de la construcción de la secuencia no se

tuviesen en cuenta en los pasos no contiguos. Aunque la versión V.2. trata de mejorar esto incluyendo más iteraciones, igualmente conlleva que se descarten movimientos más prometedores que los encontrados en una fase de construcción más avanzada, puesto que siguen manteniendo cierta caducidad en la lista en función del tiempo transcurrido en ella, y no en función de su calidad, al igual que ocurría en la versión anterior (V.1). Esto se tratará de variar en las siguientes versiones propuestas a continuación.

Entrada: datos de la instancia, parámetro *x*, parámetro *y*
Salida: Π_b, Obj_b
Inicio
  $P_{\gamma_j} = \sum_{i=1}^{m} p_{i\gamma_j}$;
  $J = \{\gamma_1, \ldots, \gamma_n\}$ verificándose que $P_{\gamma_1} \geq P_{\gamma_2} \geq \ldots P_{\gamma_n}$;
  Asignar a *Obj* el valor de la función objetivo para *J;*
  $z = n * y$
  $\Pi^{(1)} = \{\gamma_1\}$;
  Para $k = 2$ **hasta** n **hacer**
    $S^{(k)} = k \cdot x$;
    Insertar el trabajo γ_k en todas las posibles posiciones j de $\Pi^{(k-1)}$, con $j \in \{1, \ldots, k\}$;
    $\Pi^{(k)}$ = secuencia obtenida insertando γ_k en la posición de $\Pi^{(k-1)}$ con menor valor de la función objetivo (Obj_b);
    $r_s^{(k)}$ (con $s \in \{1, \ldots, S^{(k)}\}$) = trabajo anterior a γ_k , tras evaluar γ_k en la s-ésima posición con menor Obj_b;
    Si $k > 2$ **entonces**
      $h = 0$;
      Mientras $h < z$ **y** $k > h$ **hacer**
        Para $s = 1$ **hasta** $S^{(k-1-h)}$ **hacer**
          $\Pi'^{(k)}$ = permutación obtenida retirando el trabajo $\gamma_{(k-1-h)}$ de $\Pi^{(k)}$ y reinsertándolo detrás del trabajo $r_s^{(k-1-h)}$. Asignar a *Obj*' su valor de la función objetivo;
          Si *Obj*' < Obj_b **entonces**
            $\Pi_b^{(k)} = \Pi'^{(k)}$;
            $Obj_b = Obj'$;
          Fin
        Fin
        $h = h + 1$;
      Fin
    Fin
  Fin
  Si *Obj* < Obj_b **entonces**
    $Obj_b = Obj$;
    $\Pi_b = J$;
  Fin
  Devolver Π_b, Obj_b
Fin

Figura 5.5. Pseudocódigo procedimiento V.2. [Fuente: Elaboración propia]

5.4 Versión 3 (V.3)

A diferencia de las versiones 1 y 2, donde cada vez que se inserta un nuevo trabajo en la secuencia se repetían los movimientos más prometedores de las iteraciones anteriores, en esta tercera versión la lista contiene los movimientos más prometedores globales durante todo el procedimiento de búsqueda.

A diferencia de las dos versiones anteriores, un movimiento de la lista no tiene caducidad, su permanencia en la lista depende de la calidad de su solución en comparación con la mejor obtenida hasta el momento de su inclusión en la lista. Para determinar dicha calidad, se usa un incremento (I) que mide cuánto se desvía el valor de la función objetivo de la secuencia candidata a entrar en la lista ($Obj_{Candidato}$) respecto al mínimo valor encontrado hasta ese momento (Obj_b), como se muestra en la siguiente ecuación:

$$I = \frac{Obj_{Candidato} - Obj_b}{Obj_b} * 100 \qquad (5.2)$$

La capacidad de la lista, es decir, el número de movimientos que se almacenan en ella viene determinado por el tamaño de la lista. Este se determina en función del número de trabajos n a secuenciar que componen la instancia del problema, multiplicado por el parámetro t (tamaño = $\lfloor t * n \rfloor$). Con esto se pretende que el algoritmo se adapte a cada instancia según su volumen de trabajos, ya que aunque provoque un aumento notable del tiempo de cómputo a mayor cantidad de trabajos, es preferible a establecer un tamaño fijo de la lista común para cualquier n.

En la Figura 5.6 se muestra la nueva estructura de la lista que contiene los movimientos prometedores. Cada movimiento contenido en la lista se corresponde con la combinación del trabajo a reinsertar y el trabajo tras el cual se debe posicionar en la secuencia, y lleva asignada la desviación respecto a la mejor función objetivo mencionada anteriormente (I).

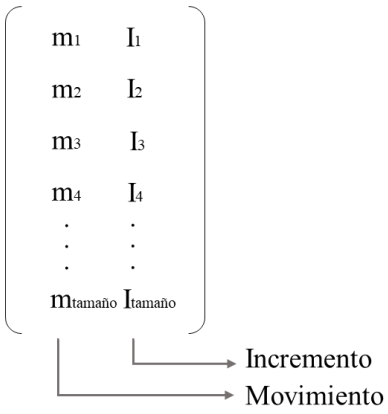

Figura 5.6. Lista de movimientos prometedores para V.3 y V.4. [Fuente: Elaboración propia]

Como la lista tiene una capacidad limitada por su tamaño, al insertar un nuevo movimiento cuando la ocupación de la lista alcance su capacidad máxima, se usará el valor de I para decidir su adición o no a la lista.

Cuando esto ocurre, la lista está completa y se desea introducir un nuevo movimiento prometedor en ella, se debe comprobar que no es peor que los ya contenidos en la lista. Para ello se comparan los incrementos de la lista hallando el máximo valor de ellos, y si el nuevo movimiento tiene un menor incremento se introduce en la lista reemplazando el lugar del movimiento con el máximo incremento.

En caso de empate, si el movimiento a introducir y el peor contenido en la lista son idénticos en cuanto a sus valores de incremento, no se introduce, ya que a priori no proporciona ninguna mejora. En concreto esto se debe a que el cálculo del incremento mediante la ecuación (5.2) favorece a los movimientos de iteraciones más avanzadas del algoritmo. A medida que se va construyendo la secuencia iteración a iteración con la inserción

de un nuevo trabajo, el valor de la mejor función objetivo hasta el momento (Obj_b) va incrementando, influyendo en que la desviación sea menor debido a que el denominador es mayor. Por tanto, cuando el valor de la función objetivo del movimiento candidato a entrar en la lista ($Obj_{Candidato}$) presente realmente igual desviación respecto al mejor (Obj_b) en dos iteraciones distintas, el cálculo del incremento proporcionará un valor menor para el movimiento de la iteración más reciente. Por este razonamiento, se decide que ante caso de empate, se da preferencia a los ya existentes en la lista para compensar esa diferencia.

Por último, respecto a la casuística que se sigue a la hora de decidir cuándo se realiza un movimiento de la lista, esta versión vuelve a reinsertar todos los movimientos contenidos en la lista tras la secuenciación de cada trabajo en la secuencia parcial. Cuando se produce una mejora con la repetición de uno de los movimientos, la lista se actualiza de modo que el movimiento realizado (que ahora forma parte de la mejor secuencia parcial) deje de formar parte de la lista, ocupando su lugar el movimiento recién desecho.

El pseudocódigo correspondiente al procedimiento descrito se muestra en la Figura 5.7.

5.5 Versión 4 (V.4)

Para la última de las versiones, se ha hecho uso como en el caso anterior, de una única lista que contiene los movimientos prometedores globales (Figura 5.6), pero en este caso no viene limitada por un tamaño, sino que su capacidad es infinita.

En esta modificación, la condición que deben cumplir los movimientos candidatos a introducirse en la lista viene definida por un incremento máximo. Este límite se impone de forma que un movimiento solo entrará en la lista si proporciona un valor de la función objetivo que se desvíe una cantidad máxima del mejor valor encontrado hasta el momento. Esta diferencia se mide en cada iteración, de la misma manera que en la versión anterior, mediante el incremento determinado por la ecuación (5.2)**.**

Para determinar el valor del incremento límite en cada instante, se tiene en cuenta como principal factor el tiempo de *set-up*. Puesto que se trata de un taller de flujo regular con tiempos de *set-up*, cada vez que se añade un nuevo trabajo en la secuencia parcial, el valor de la función objetivo se verá incrementado por la suma de los tiempos de proceso de dicho trabajo en cada una de las maquinas que compongan el taller, más los tiempos de *set-up* de ese trabajo para cada máquina. Estos últimos, al depender de la secuencia, varían en función del trabajo que se ha procesado antes en la máquina en cuestión, por lo que la variación principal del valor de la función objetivo de una iteración respecto a la siguiente vendrá dada por la variación de los tiempos de ajuste. Por este razonamiento, el incremento límite se calcula en cada iteración de la siguiente forma:

$$I_{\max} = \frac{a * \overline{S}}{Obj_b} * 100 \qquad (5.3)$$

Donde:

- a es el parámetro de entrada del algoritmo.
- $\overline{S}$ es la media aritmética de todos los tiempos de *set-up* S_{ijk}.
- Obj_b es el valor de la función objetivo para la mejor secuencia parcial encontrada hasta el momento.

De igual forma que en la tercera versión (V.3), se repiten todos los movimientos de la lista cada vez que se inserte un nuevo trabajo en la secuencia parcial, actualizando la mejor secuencia (y por tanto la lista) en caso de que se produzcan mejoras respecto a la función objetivo.

El pseudocódigo correspondiente a la heurística V.4 se muestra en la Figura 5.8.

Entrada: datos de la instancia, parámetro *t*
Salida: Π_b, Obj_b
Inicio
$P_{\gamma_j} = \sum_{i=1}^{m} p_{i\gamma_j}$;
$J = \{\gamma_1, \ldots, \gamma_n\}$ verificándose que $P_{\gamma_1} \geq P_{\gamma_2} \geq \ldots P_{\gamma_n}$;
Asignar a *Obj* el valor de la función objetivo para *J;*
$\Pi^{(1)} = \{\gamma_1\}$;
*capacidad_L = t * n;*
L = ∅; **comentario**: lista que contiene los movimientos prometedores, inicialmente vacía.
Para *k* = 2 **hasta** *n* **hacer**
Insertar el trabajo γ_k en todas las posibles posiciones *j* de $\Pi^{(k-1)}$, con $j \in \{1, \ldots, k\}$;
$\Pi^{(k)}$ = secuencia obtenida insertando γ_k en la posición de $\Pi^{(k-1)}$ con menor valor de la FO (Obj_b);
$CL_x^{(k)}$ (con $x \in \{1, \ldots, k\}$) = trabajo anterior a γ_k , tras evaluar γ_k en la posición *x*. Hacer que Obj_{CLx} denote tal valor de la función objetivo;
Para *x* = 2 **hasta** *k* **hacer**
$I_{CLx} = (Obj_{CLx} - Obj_b) / Obj_b * 100$;
Si *ocupación_L* < *capacidad_L* **entonces**
Asignar movimiento ($CL_x^{(k)}$, γ_k , I_{CLx}) a *L*;
Fin
Por el contrario
I_{max} = mayor incremento (*I*) contenido en *L*;
Si $I_{CLx} < I_{max}$ **entonces**
Reemplazar el movimiento contenido en *L* correspondiente al I_{max} por ($CL_x^{(k)}$, γ_k , I_{CLx});
Fin
Fin
Fin
Si *k* > 2 **entonces**
Para *h* = 0 **hasta** *ocupación_L* **hacer**
$\Pi'^{(k)}$ = permutación obtenida realizando el movimiento *h* de la lista *L* en $\Pi^{(k)}$. Asignar a *Obj*' su valor de la función objetivo;
Si *Obj*' < Obj_b **entonces**
$\Pi_b^{(k)} = \Pi'^{(k)}$;
$Obj_b = Obj'$;
Actualizar *L* (eliminar movimiento realizado de la lista e incluir el recién deshecho en su lugar);
Fin
Fin
Fin
Fin
Si *Obj* < Obj_b **entonces**
$Obj_b = Obj$;
$\Pi_b = J$;
Fin
Devolver Π_b, Obj_b
Fin

Figura 5.7. Pseudocódigo procedimiento V.3. [Fuente: Elaboración propia]

Entrada: datos de la instancia, parámetro *a*
Salida: Π_b, Obj_b
Inicio
  $P_{\gamma_j} = \sum_{i=1}^{m} p_{i\gamma_j}$;
  $J = \{\gamma_1, \ldots, \gamma_n\}$ verificándose que $P_{\gamma_1} \geq P_{\gamma_2} \geq \ldots P_{\gamma_n}$;
  Asignar a *Obj* el valor de la función objetivo para *J;*
  $\Pi^{(1)} = \{\gamma_1\}$;
  $L = \emptyset$; **comentario**: lista que contiene los *t* movimientos prometedores, inicialmente vacía.
  Para $k = 2$ **hasta** *n* **hacer**
    Insertar el trabajo γ_k en todas las posibles posiciones *j* de $\Pi^{(k-1)}$, con $j \in \{1, \ldots, k\}$;
    $\Pi^{(k)}$ = secuencia obtenida insertando γ_k en la posición de $\Pi^{(k-1)}$ con menor valor de la función objetivo (Obj_b);
    $CL_x^{(k)}$ (con $x \in \{1, \ldots, k\}$) = trabajo anterior a γ_k , tras evaluar γ_k en la posición *x*. Hacer que Obj_{CLx} denote tal valor de la función objetivo;
    $I_{limite} = (a * \bar{S}) / Obj_b * 100$;
    Para $j = 2$ **hasta** *k* **hacer**
      $I_{CLx} = (Obj_{CLx} - Obj_b) / Obj_b * 100$;
      Si $I_{CLx} < I_{limite}$ **entonces**
        Asignar movimiento ($CL_x^{(k)}$, γ_k , I_{CLx}) a *L*;
      Fin
    Fin
    Si $k > 2$ **entonces**
      Para $h = 0$ **hasta** *ocupación_L* **hacer**
        $\Pi'^{(k)}$ = permutación obtenida realizando el movimiento *h* de la lista *L* en $\Pi^{(k)}$. Asignar a *Obj*' su valor de la función objetivo;
        Si *Obj*' < Obj_b **entonces**
          $\Pi_b^{(k)} = \Pi'^{(k)}$;
          $Obj_b = Obj'$;
          Actualizar *L* (eliminar movimiento realizado de la lista e incluir el recién deshecho en su lugar);
        Fin
      Fin
    Fin
  Fin
  Si *Obj* < Obj_b **entonces**
    $Obj_b = Obj$;
    $\Pi_b = J$;
  Fin
  Devolver Π_b, Obj_b
Fin

Figura 5.8. Pseudocódigo procedimiento V.4. [Fuente: Elaboración propia]

6 EVALUACIÓN COMPUTACIONAL

A partir de la implementación de las heurísticas anteriores, se han evaluado bajo las mismas condiciones un conjunto de instancias en cada uno de los cuatro algoritmos aproximados propuestos para la resolución del problema analizado. Se presentan a continuación los bancos de pruebas e indicadores usados para evaluar su desempeño, y los resultados obtenidos en la comparación, comentándolos brevemente.

6.1 Bancos de pruebas

Para el problema objeto del estudio se definen los parámetros correspondientes al número de trabajos *n*, número de máquinas *m*, tiempos de proceso p_{ij}, y tiempos de *set-up* S_{ijk}. Se generarán para ello un conjunto de instancias que contengan los datos de entrada para el problema, similar a las propuestas por Vallada y Ruiz (2011).

El conjunto de instancias generadas se compone por la combinación de los siguientes números de trabajos y máquinas respectivamente: $n = \{50, 100, 150, 200, 250\}$ y $m = \{10, 15, 20, 25, 30\}$.

Respecto a los tiempos de proceso, se han generado aleatoriamente siguiendo una distribución uniforme con valores entre 1 y 99.

Para los tiempos de *set-up*, al igual que los tiempos de proceso, se han generado aleatoriamente según una distribución U (1, γ). El parámetro γ de la distribución uniforme toma los valores 9, 49, 99 y 124, generando así 4 subconjuntos de datos: U (1, 9), U (1, 49), U (1, 99) y U (1, 124), respectivamente.

Además, se han realizado 10 replicaciones para cada una de las posibles combinaciones anteriores, generando así un total de 1000 instancias para la comparación de las heurísticas propuestas.

6.2 Indicadores de desempeño

Puesto que cada una de las 4 heurísticas a comparar proporcionarán soluciones con distinto nivel de calidad en un tiempo de cómputo distinto, se usarán indicadores que midan ambas magnitudes, como son:

- *Average Relative Percentage Deviation* ($ARPD_h$): mide la media de la calidad de las soluciones obtenidas por cada heurística (RPD_{ih}) para el conjunto de iteraciones evaluadas.

$$ARPD_h = \sum_{i=1}^{I} \frac{RPD_{ih}}{I}, \forall h = 1, ..., H \qquad (6.1)$$

- *Average Computational CPU Times* (*ACT*)*:* mide el tiempo (*T*) que ha tardado cada heurística de media al evaluar las iteraciones.

$$ACT_h = \sum_{i=1}^{I} \frac{T_{ih}}{I}, \forall h = 1, ..., H \qquad (6.2)$$

Donde:

- H es el número de algoritmos bajo comparación ($h \in \{1, ..., H\}$)
- I es el número de instancias ($i \in \{1, ..., I\}$)

La calidad de las soluciones se mide con la desviación del valor de la función objetivo obtenido con la heurística *h* para la iteración *i* (Obj_{ih}) respecto al mejor valor obtenido entre todas las heurísticas para esa iteración, como se puede ver en la siguiente ecuación:

$$RPD_{ih} = \frac{Obj_{ih} - Mejor_i}{Mejor_i} * 100, \forall h = 1, ..., H \quad (6.3)$$

Para comparar los algoritmos aproximados en cuanto a su tiempo, se mide la desviación mediante el siguiente indicador:

- *Average Relative Percentage computation Time* (*ARPT*):

$$ARPT_h = 1 + \sum_{i=1}^{I} \frac{RPT_{ih}}{I}, \forall h = 1, ..., H \quad (6.4)$$

Donde:

$$RPT_{ih} = \frac{T_{ih} - ACT_i}{ACT_i}, \forall h = 1, ..., H \quad (6.5)$$

Siendo:

$$ACT_i = \sum_{h=1}^{H} \frac{T_{ih}}{I}, \forall i = 1, ..., I \quad (6.6)$$

6.3 Valores parámetros

Cada una de las heurísticas requiere como dato de entrada el valor de un parámetro. Al influir el parámetro directamente en el esfuerzo computacional del algoritmo, se evaluarán los algoritmos para diferentes valores de los parámetros (ver Fernández-Viagas et al., 2020 para enfoques similares). Para comparar valores diversos de los parámetros que sean representativos y proporcionen resultados distantes en cuanto a calidad y tiempo, se ha decidido establecer los siguientes rangos de valores:

- Versión 1: x = {0.2, 0.4, 0.6, 1}, siendo x el porcentaje de movimientos en tanto por uno de la iteración anterior que se reinsertan en cada iteración.
- Versión 2: y = {0.02, 0.05, 0.15, 0.5}; al parámetro x se le ha dado un valor constante de 0.2. Se recuerda que el parámetro y indica la proporción de iteraciones que se tienen en cuenta a la hora de reinsertar movimientos de la lista. Los movimientos de la lista tienen una caducidad de ($z = \lfloor y * n \rfloor$) iteraciones.
- Versión 3: t = {0.2, 0.8, 2.5, 5}, donde t determinaba el tamaño de la lista, (tamaño = $\lfloor t * n \rfloor$).
- Versión 4: a = {0.1, 0.3, 0.5, 1}, siendo a un factor que multiplica la media de los tiempos de *set-up* en el cálculo del incremento límite de la ecuación (5.3).

Así se compararán los 4 algoritmos con 4 parámetros distintos cada uno, resultando en 16 variantes de las heurísticas propuestas a comparar. Además, se ha incluido en la comparación la heurística NEH, para usarla como referencia en cuanto a los cambios producidos en calidad de la solución y tiempo de cómputo por el resto de las heurísticas propuestas en comparación con ella.

6.4 Resultados

Todas las heurísticas han sido codificadas en lenguaje de programación C# usando Visual Studio en un Intel Core i7-7700 con 3.6GHz, 16 GB RAM, con Microsoft Windows 10 64 bit, y usando las mismas funciones y librerías de forma común.

A continuación en la Tabla 6.1 se presentan los valores obtenidos de *ARPD* para cada heurística, agrupados por número de trabajos, máquinas, y parámetro γ.

Tabla 6.1. Valores detallados de *ARPD* agrupados por *n*, *m* y *γ*.

Heurística	*n*					*m*					*γ*				Media
	50	100	150	200	250	10	15	20	25	30	9	49	99	124	
NEH	2,46	2,48	2,54	2,60	2,74	3,11	2,68	2,56	2,29	2,18	1,79	2,39	2,87	3,21	2,564
V.1 (0,2)	2,04	2,09	2,17	2,15	2,21	2,52	2,25	2,08	1,95	1,86	1,54	1,96	2,40	2,63	2,132
V.1 (0,4)	2,03	2,02	2,10	2,15	2,18	2,52	2,17	2,06	1,93	1,81	1,49	1,96	2,34	2,60	2,098
V.1 (0,6)	1,99	2,03	2,12	2,14	2,20	2,54	2,15	2,04	1,95	1,79	1,51	1,99	2,35	2,53	2,096
V.1 (1)	2,00	2,05	2,14	2,17	2,25	2,51	2,19	2,05	1,95	1,91	1,56	1,95	2,37	2,60	2,121
V.2 (0,02)	2,04	1,79	1,64	1,58	1,47	2,05	1,78	1,64	1,55	1,51	1,28	1,56	1,88	2,10	1,704
V.2 (0,05)	1,96	1,38	1,22	1,05	1,04	1,61	1,40	1,32	1,18	1,13	0,96	1,17	1,43	1,76	1,328
V.2 (0,15)	1,35	0,89	0,69	0,59	0,48	0,98	0,81	0,76	0,76	0,68	0,50	0,70	0,88	1,12	0,799
V.2 (0,5)	1,06	0,56	0,37	0,26	0,19	0,54	0,49	0,51	0,46	0,44	0,21	0,43	0,58	0,73	0,488
V.3 (0,2)	1,99	1,85	1,93	1,95	2,04	2,38	2,04	1,90	1,77	1,66	1,45	1,75	2,18	2,42	1,950
V.3 (0,8)	1,45	1,39	1,30	1,34	1,42	1,62	1,48	1,34	1,26	1,18	1,13	1,24	1,50	1,64	1,377
V.3 (2,5)	0,95	0,87	0,86	0,82	0,88	0,99	0,88	0,81	0,86	0,84	0,70	0,76	0,97	1,07	0,876
V.3 (5)	0,71	0,54	0,59	0,57	0,58	0,74	0,62	0,62	0,49	0,53	0,50	0,53	0,66	0,71	0,599
V.4 (0,1)	2,14	2,06	2,12	2,06	2,10	2,48	2,19	2,03	1,96	1,83	1,61	1,91	2,33	2,54	2,098
V.4 (0,3)	1,73	1,48	1,44	1,35	1,38	1,68	1,50	1,41	1,43	1,36	1,44	1,32	1,50	1,64	1,475
V.4 (0,5)	1,32	1,10	1,07	0,99	0,94	1,15	1,09	1,12	1,05	1,01	1,24	0,97	0,97	1,16	1,084
V.4 (1)	0,95	0,55	0,43	0,36	0,41	0,56	0,54	0,51	0,56	0,54	0,91	0,41	0,37	0,48	0,541

En relación a estos resultados se puede destacar que:

- Las heurísticas funcionan mejor para una menor variabilidad de los tiempos de *set-up* (parámetro γ pequeño). A medida que la variabilidad de los tiempos de *set-up* aumenta, la calidad de la solución se ve reducida, excepto en el caso de la versión 4 (V.4), en la cual se ha tenido esto en cuenta a la hora de calcular el límite máximo (véase sección 5.5).

- Por lo general, a mayor valor de los parámetros de entrada se obtienen mejores resultados, aunque requiere más tiempo computacional, ya que se prueban más combinaciones. No es el caso de la versión 1 (V.1) que para valores altos del parámetro x (cuando se repiten gran cantidad de movimientos prometedores de la iteración anterior), no solo no se aprecian mejoras significativas, sino que la solución llega a dispersarse y se obtienen peores soluciones (falta de intensificación, exceso de diversificación). Por esto se deduce que en esta versión no es conveniente probar en cada iteración más del 50% de los movimientos prometedores de la iteración anterior, para obtener mejores soluciones y no aumentar el tiempo de cómputo innecesariamente.

- En cuanto a la versión 2 (V.2) puede observarse que por lo general funciona notablemente mejor para instancias con mayor número de trabajos. Esto se debe a que el parámetro *y* se ha establecido en relación a *n*, dando lugar a un mayor número de combinaciones cuanto mayor es *n*.

- Por lo general, las heurísticas funcionan mejor cuando el taller de flujo se compone de un número mayor de máquinas, ya que existe un mayor número de posibles soluciones.

En la Tabla 6.2 se presenta la recopilación de los indicadores de desempeño de calidad y tiempo.

Tabla 6.2. Resumen de los resultados computacionales.

Heurística	ARPD	ACT	ARPT
NEH	2,564	1,416	0,307
V.1 (0.2)	2,132	1,696	0,373
V.1 (0.4)	2,098	1,978	0,434
V.1 (0.6)	2,096	2,260	0,495
V.1 (1)	2,121	2,823	0,619
V.2 (0.02)	1,704	2,711	0,501
V.2 (0.05)	1,328	4,393	0,723
V.2 (0.15)	0,799	9,287	1,341
V.2 (0.5)	0,488	20,994	2,894
V.3 (0.2)	1,950	1,853	0,411
V.3 (0.8)	1,377	3,133	0,700
V.3 (2.5)	0,876	6,446	1,365
V.3 (5)	0,599	12,179	2,484
V.4 (0.1)	2,098	1,911	0,401
V.4 (0.3)	1,475	3,381	0,626
V.4 (0.5)	1,084	5,707	0,947
V.4 (1)	0,541	17,414	2,379

Como era previsto, los algoritmos conllevan un mejor tiempo para valores de sus parámetros más pequeños. Se puede observar en la tabla anterior cómo algunas heurísticas mejoran notablemente la calidad de la solución en comparación con la NEH sin incrementar el tiempo de cómputo en gran medida. Como se esperaba, el algoritmo más lento (V.2 para $y = 0,5$) es el que proporciona soluciones de mayor calidad.

Para visualizar gráficamente la relación calidad-tiempo que presenta cada heurística se expone la siguiente Figura 6.1 donde el eje de abscisas se representa la desviación en cuanto a tiempo y en el eje de ordenadas la desviación en cuanto a calidad.

Como se puede observar, todas las heurísticas propuestas mejoran en calidad a la solución proporcionada por la NEH para el problema estudiado, aunque requieren de mayor tiempo de cálculo. Por lo general, se obtienen mejores soluciones en un tiempo mayor. Dependiendo de la prioridad dada a cada aspecto (calidad de la función objetivo o esfuerzo computacional requerido), será más útil una heurística que proporcione resultados muy buenos en mayor tiempo, o se preferirá una solución rápida de una calidad intermedia.

Adicionalmente, en la Tabla 6.3 se presentan las relaciones de dominancia existentes entre las distintas versiones que se han comparado. Cabe destacar, por ejemplo, que V.3(2.5) está dominada por V.2(0.15) pero esta última no está dominada, es decir, ninguna otra heurística presenta menor valor de ambos objetivos (C_{max} y CPU time). Lo mismo ocurre con V.3(5) que está dominada por V.4(1), la cual no está dominada. Además, no hay ninguna heurística que domine a V.3(0.2), pero en cambio esta domina a V.1(0.4), V.1(0.6) y V.1(1). Otros dos casos más de heurísticas no dominadas son el de V.4(0.1), la cual domina a V.1(1) y V.1(0.4); y el de V.2(0.02), que domina a V.1(1).

Tabla 6.3. Relaciones de dominancia.

Heurística	Dominada por:	Domina a:
NEH	-	-
V.1 (0.2)	-	-
V.1 (0.4)	V.3(0.2), V.4(0.1)	V.1(1)
V.1 (0.6)	V.3(0.2)	V.1(1)
V.1 (1)	V.1(0.4), V.1(0.6), V.2(0.02), V.3(0.2), V.4(0.1)	-
V.2 (0.02)	-	V.1(1)
V.2 (0.05)	-	-
V.2 (0.15)	-	V.3(2.5)
V.2 (0.5)	-	-
V.3 (0.2)	-	V.1(0.4), V.1(0.6), V.1(1)
V.3 (0.8)	-	-
V.3 (2.5)	V.2(0.15)	-
V.3 (5)	V.4(1)	-
V.4 (0.1)	-	V.1(1), V.1(0.4)
V.4 (0.3)	-	-
V.4 (0.5)	-	-
V.4 (1)	-	V.3(5)

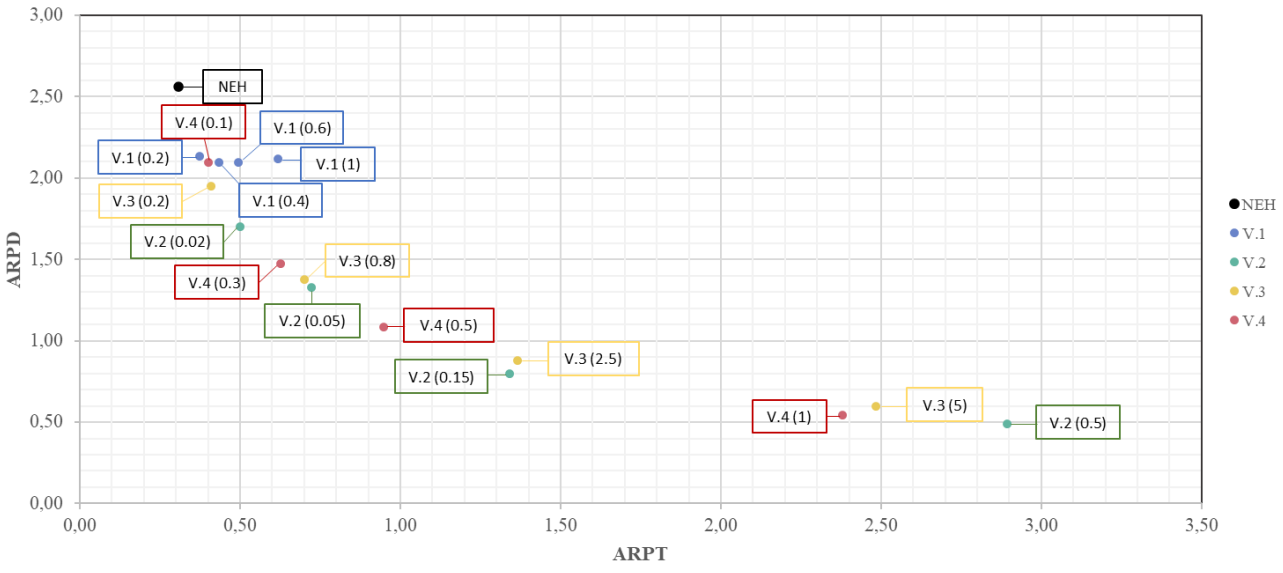

Figura 6.1. ARPD contra ARPT. [Fuente: Elaboración propia]

7 CONCLUSIONES

Para concluir, en este último capítulo del documento se comentan las conclusiones obtenidas con la realización del proyecto, incluyendo las posibles futuras líneas de investigación que podría tener.

7.1 Conclusiones

Tras el estudio del problema objeto de este proyecto ($F_m|prmu, S_{ijk}|C_{max}$), se concluye que por su complejidad para obtener el óptimo en un tiempo admisible, el procedimiento más adecuado de resolución es mediante métodos aproximados.

En vista de los resultados obtenidos, la combinación de una heurística constructiva con una búsqueda local de enfoque iterativo es una buena elección para resolver este problema. En concreto, la metodología propuesta para su resolución ha presentado mejoras en la calidad de las soluciones respecto a la heurística NEH usada como base para las modificaciones. Por lo general, existe relación entre calidad de la solución y tiempo computacional: las heurísticas desarrolladas proporcionan mejores soluciones en un mayor tiempo, mientras que las soluciones más rápidas son de menor calidad. Según el valor establecido para los parámetros de entrada a los algoritmos aproximados, la solución será mejor o peor, y requerirá de un mayor o menor tiempo. Valorando ambos criterios (tiempo y calidad), las mejores heurísticas se corresponden con: V.2 (0.05), V.3(0.8) y V.4(0.5). Además, la heurística V.4 proporciona muy buenos resultados para tiempos de *set-up* muy variables, ya que en su procedimiento trata de ajustar el criterio de movimientos prometedores en función de la media de estos tiempos.

Respecto a los algoritmos propuestos, se podría concluir que no interesa un tamaño de lista estático, ya que para instancias con distintos volúmenes de trabajos no sería igual de eficiente. Aunque esto se ha tratado de evitar ajustando el tamaño en función del número de trabajos, no ha resultado del todo efectivo, ya que el tiempo de cálculo aumentaría a mayor volumen de trabajos. Por ello quizá sea conveniente ajustar dinámicamente el tamaño de la lista o el número de movimientos que se repiten en cada iteración, modificándolo en función de algún parámetro en cada iteración del algoritmo o según otro criterio. Además, cuando el tamaño de tamaño de lista es demasiado grande se podrían crear problemas de memoria y tiempo de cómputo.

Por tanto, los resultados generados en el desarrollo de este proyecto se consideran satisfactorios, aunque existen diversas mejoras a realizar para los métodos de resolución propuestos que reducirían el tiempo de cómputo y quizá puedan proporcionar mejores soluciones.

7.2 Futuras líneas de investigación

Respecto a las posibles líneas de investigación que da lugar este proyecto, se podrían presentar variantes de los métodos propuestos que incluyan mejoras en cuanto a la inserción de movimientos prometedores de forma más eficiente, o un mejor ajuste de los tamaños de las matrices o listas que contienen dichos movimientos.

Además, como se ha comentado con anterioridad, en dos de las modificaciones planteadas (V.3 y V.4) se hace uso de un valor para medir la desviación respecto a la mejor función objetivo encontrada hasta el momento (I). El cálculo de dicho valor puede favorecer o perjudicar a las combinaciones de movimientos en función del instante en el que se calcula, provocando que en las primeras iteraciones del algoritmo el denominador sea mucho menor que en las últimas, causando una distorsión en la medida de la desviación. Sería conveniente estudiar esta medida, tratando de incluir en el cálculo un parámetro que compense este error en función de cada iteración.

También cabe destacar que para la versión 4 del algoritmo, el incremento límite se calcula en base a la media de todos los tiempos de ajuste que componen la instancia para hacer una simplificación. Convendría evaluar si calcular una media más ajustada (media de los tiempos de ajuste de cada trabajo, en lugar de una media global) proporciona mejores resultados sin incrementar el tiempo de cómputo en gran medida.

Adicionalmente, como se ha comentado en las conclusiones, se piensa que ajustar el tamaño de la lista de forma dinámica podría ser muy conveniente, pudiendo proporcionar soluciones de mayor calidad en menor tiempo, por lo que sería otro posible aspecto para analizar.

Por último, sería interesante abordar si las conclusiones obtenidas en el documento en base a las medidas obtenidas por los algoritmos pueden ser confirmadas estadísticamente mediante test.

REFERENCIAS

Abiri, M. B., Zandieh, M., & Alem-Tabriz, A. (2009). A Tabu Search Approach to Hybrid Flow Shops Scheduling with Sequence-Dependent Setup Times. *Journal for Applied Sciences, 9*(9), 1740-1745.

Adel, D., Ahcene, B., Abdelhakim, A., & Nadia Nouali, T. (2019). Efficient parallel tabu search for the blocking job shop scheduling problem. *Methodologies and application. Soft Computing*.

Chávez-Bosquez, O., Pozos-Parra, P., & Gómez-Ramos, J. L. (2015). Búsqueda Tabú con criterio de aspiración probabilístico aplicada a la generación de horarios escolares. *Revista de MAtemática: Teoría y Aplicaciones*.

Companys Pascual, R., & Ribas Vila, I. (July de 2012). El curioso comportamiento del método de inserción de la heurística NEH en el problema Fm|block|Cmax. *6th International Conference on Industrial Engineering and Industrial Management.* Vigo.

Dell'Amico, M., & Trubian, M. (1993). Applying tabu search to the job-shop scheduling problem. *Annals of Operations Research*, 231-252.

Fernández-Viagas, V. (s.f.). *Organización de la producción. Entornos de fabricación*. Obtenido de http://www.organizaciondelaproduccion.com/entornos-fabricacion.php

Fernández-Viagas, V., Costa, A., & Framiñán, J. M. (2020). *Hybrid flow shop with multiple servers: a computational evaluation and efficient divide-and-conquer heuristics.*

Fernández-Viagas, V., Molina-Pariente, J. M., & Framiñán, J. M. (2018). New efficient constructive heuristics for the hybrid flowshop to minimise makespan: A computational evaluation of heuristics. *Expert Systems With Applications*, 345-456.

Framiñan, J. M., Leisten, R., & Ruiz García, R. (2014). *Manufacturing Scheduling Systems An Integrated View on Models, Methods and Tools.* London: Springer.

Grabowski, J., & Wodecki, M. (2004). A very fast tabu search algorithm for the permutation flow shop problem with makespan criterion. *Computers & Operations Research*, 1891-1909.

Gupta, J. N. (2010). Designing a tabu search algorithm for the two-stage flow shop problem with secondary criterion. *Production Planning & Control: The Management of Operations*.

Nawaz, M., Enscore Jr, E. E., & Ham, I. (1983). "A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega, 11*(1), 91-95.

Nowicki, E., & Smutnicki, C. (2005). An advanced tabu search algorithm for the job shop problem. *Journal of Scheduling*, 145–159.

Pérez González, P., Fernández-Viagas Escudero, V., & Framiñán, J. M. (s.f.). Temario Programación y Control de la Producción. Sevilla: Escuela Técnica Superior de Ingeniería, Universidad de Sevilla.

Pinedo, M. L. (2008). *Scheduling. Theory, Algorithms, and Systems.* New York, NY, USA: Springer.

Ponsich, A., & Coello Coello, C. A. (2012). A hybrid Differential Evolution—Tabu Search algorithm for the solution of Job-Shop Scheduling Problems. *Applied Soft Computing*, 462-474.

Prot, D., & Bellenguez-Morineau, O. (2012). Tabu search and lower bound for an industrial complex shop scheduling problem. *Computers & Industrial Engineering*, 1109-1118.

Sauvey, C., & Sauer, N. (2020). Two NEH Heuristic Improvements for Flowshop Scheduling Problem with Makespan Criterion. *Algorithms*.

Vela, C. R., Afsar, S., Palacios, J. J., González-Rodríguez, I., & Puente, J. (2020). Evolutionary tabu search for flexible due-date satisfaction in fuzzy job shop shceduling. *Computers and Operations Research*.

Zanakis, S. H., & Evans, J. R. (1981). Heuristic "Optimization": Why, When, and How to Use It. *INTERFACES, 11*(5).