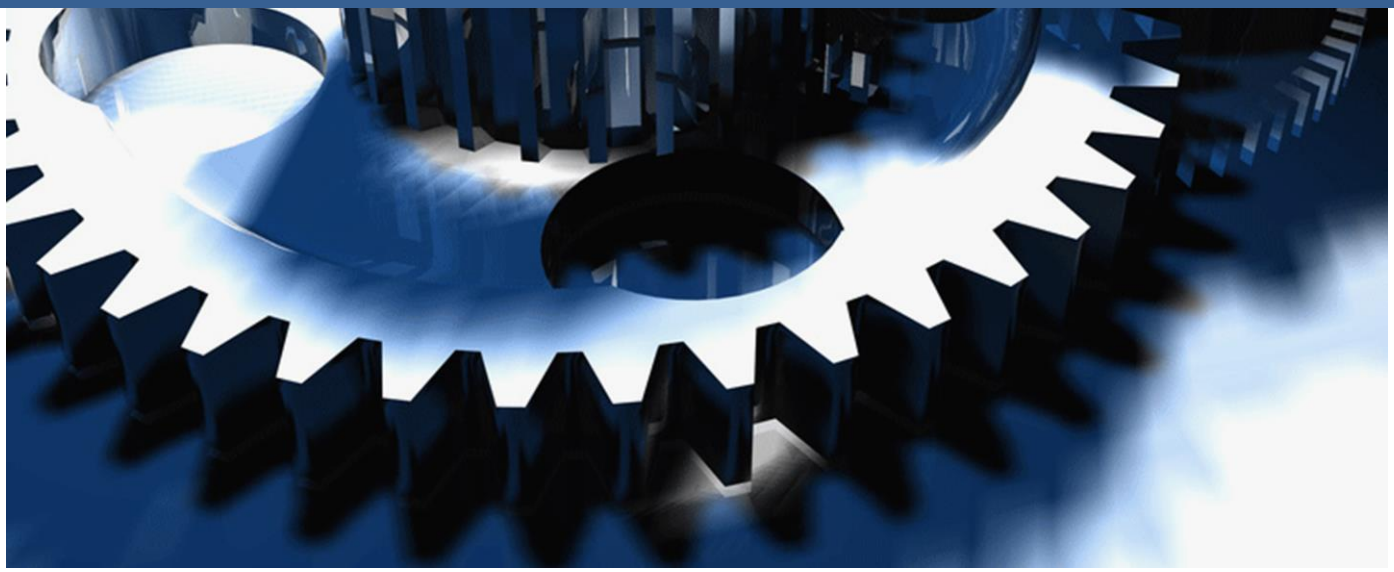
Universidad de Valladolid

ESCUELA DE INGENIERÍAS INDUSTRIALES

Algoritmos constructivos para la programación de operaciones en entornos *job shop* flexibles



UNIVERSIDAD DE VALLADOLID
ESCUELA DE INGENIERIAS INDUSTRIALES
Grado en Ingeniería en Organización Industrial

Valladolid 2015

Clara Mª González de Diego

Tutor: Araúzo Araúzo, J. Alberto

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

Universidad de Valladolid

ESCUELA DE INGENIERÍAS INDUSTRIALES

UNIVERSIDAD DE VALLADOLID

ESCUELA DE INGENIERIAS INDUSTRIALES

Grado en Ingeniería en Organización Industrial

Algoritmos constructivos para la programación de operaciones en entornos *job shop* flexibles

Autor:

González de Diego, Clara Mª

Tutor:

Araúzo Araúzo, J. Alberto

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

Valladolid, Septiembre 2015.

Agradecimientos

A mis padres, por el cariño, apoyo incondicional y paciencia con la que siempre me han tratado.

A mi tutor, J. Alberto Araúzo Araúzo, por la confianza que depositó en mí y todo el tiempo que me ha dedicado.

A la coordinadora de grado, Marta Posada Calvo, por su continuo interés y preocupación.

RESUMEN (ABSTRACT)

Debido a la complejidad que presentan los problemas de tipo *job shop* flexible en programación de operaciones, el desarrollo de métodos exactos que encuentren la solución óptima en un tiempo razonable es muy difícil. Por lo tanto, es necesario centrarse en el uso de métodos heurísticos que encuentren de manera eficiente soluciones lo más cercanas posibles al óptimo. Se implementará un programa que combine dos métodos heurísticos constructivos, el método de lanzamiento y el método de Giffler y Thompson, con distintas reglas de lanzamiento. Finalmente se evaluarán los resultados obtenidos para determinar cuál es la mejor combinación.

PALABRAS CLAVE (KEY WORDS)

Gestión de talleres, Programación de operaciones, *Job shop* flexible, métodos heurísticos constructivos, reglas de lanzamiento.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

Antecedentes

La historia del *job shop scheduling problem* se remonta a hace más de cincuenta años, pero sus orígenes no están del todo claros existiendo diversas opiniones.

Ya en 1930 Younger publicó un libro (quizás el primero dedicado a este campo de investigación) en el que se destacaba la importancia de la programación para la consecución de los objetivos de fabricación.

Sin embargo, debido a la complejidad del problema, no fue hasta la década de los 60 con el desarrollo de la informática, cuando las posibilidades de resolución con éxito empezaron a materializarse.

Destaca que en 1964 B. Roy y B. Sussman fueron los primeros en proponer la presentación del problema mediante un grafo disyuntivo. En 1969 E. Balas aplicó un acercamiento numérico basado en él. Sin embargo, ya existían múltiples trabajos anteriores. En 1960 B. Giffler y G. L. Thompsom propusieron un algoritmo de reglas de despacho de prioridad *o dispatching rules*. J.R. Jackson, en 1956, generalizó el algoritmo del *flow shop* descrito por S. M. Johnson dos años antes, al algoritmo de *job shop*. Cabe subrayar también que en 1955, S. Akers y J. Friedman aplicaron un modelo de álgebra booleana para representar secuencias de procesamiento.

Aunque, hasta ahora, existen dudas sobre quién propuso por primera vez el problema *job shop*, es una verdad aceptada de manera univeral que en el libro “Industrial Scheduling” editado en 1964 por Muth y Thompson se asientan las bases sobre el problema de las que partieron las investigaciones que siguieron en los sucesivos años.

Por otro lado, cabe destacar el trabajo publicado por Cheng y Sin en 1990 en el que se hace referencia a un estado del arte en el que se describen las investigaciones realizadas sobre la aplicación de *dispatching rules* en entornos *job shop* con máquinas en paralelo. Más recientemente, en el año 2008 Allahverdi, Cheng y Kovalyov publicaron un informe en el que se explican las diferentes clases de problemas de *job shop* flexible y las técnicas empleadas para su resolución.

Por lo tanto, se puede decir que el problema *job shop* flexible ha sido y es, en la actualidad, ampliamente estudiado.

Justificación y objetivos

Debido a la complejidad de este tipo de problemas, el desarrollo de métodos exactos que encuentren la solución óptima en un tiempo razonable es muy difícil. Por lo tanto, el esfuerzo se suele centrar en métodos heurísticos que encuentren de manera eficiente soluciones lo más cercanas posibles al óptimo.

El objetivo del presente trabajo se puede resumir, por tanto, en los siguientes pasos:

- Revisar la literatura de diferentes algoritmos o métodos existentes para programar talleres *job shop* flexibles.
- Desarrollar un programa usando Java como lenguaje de programación para implementar los distintos algoritmos constructivos documentados.
- Comparar los algoritmos implementados mediante *benchmark*ing.

En un entorno *job shop* flexible como el que se describe en este trabajo, nos encontramos con un problema de optimización combinatoria del tipo NP-hard, para el que no se conoce un algoritmo exacto que proporcione una solución óptima en un tiempo computacional razonable. Los métodos más usados dentro de los métodos heurísticos son los constructivos. Los algoritmos constructivos que serán analizados son polinómicos, por lo que, como se ha dicho antes, no proporcionan una solución óptima, aunque sí lo más cercana a ella, y lo que es más importante, de una manera muy rápida. Lo que se pretende, es discernir cuál de todos los algoritmos planteados funcionará mejor.

Estructura del proyecto

El resto del trabajo se estructurará en 6 capítulos.

En el capítulo 1 se plantea el problema general de programación de la producción, se estudia la clasificación de los principales problemas de

scheduling y su notación, para pasar a describir el problema *job shop* flexible y terminar estudiando su complejidad.

En el capítulo 2 se presentan los principales algoritmos existentes para resolver problemas de job shop flexible.

En el capítulo 3 se describirá el software programado y que se implementará para comparar diferentes algoritmos heurísticos constructivos.

En el capítulo 4 se analizarán los resultados obtenidos tras implementar el software descrito en el capítulo anterior.

En el capítulo 5 se hará un pequeño estudio económico del presente trabajo.

Por último, en el capítulo 6, a partir de los resultados obtenidos y analizados en el capítulo 4 se obtendrán las conclusiones más relevantes del amplio estudio realizado y se propondrán líneas futuras de trabajo.

CAPÍTULO 1. PROBLEMA DE PROGRAMACIÓN DE TALLERES

1.1. Introducción

Este capítulo está dedicado a la presentación del problema de programación de talleres u operaciones (*Job Shop Scheduling Problem*). Para ello, primero se define el problema de programación de operaciones aplicado al entorno de fabricación; a continuación se definen los principales tipos de configuraciones en entornos productivos, que van a dar origen a los principales tipos de problemas en programación de la producción; seguidamente, se clasifican los principales tipos de problemas de este tipo describiendo una de las propuestas de notación más utilizada en este ámbito; después se realiza una definición formal del problema *job shop* flexible; y, por último, se estudia el concepto de complejidad computacional, aplicándolo posteriormente a este tipo de problemas.

1.2. Programación de operaciones en la gestión de talleres

El término programación de operaciones hace referencia a la búsqueda de programas de producción óptimos o cercanos al óptimo, sujetos a una serie de restricciones. Aunque este problema puede ser aplicado en distintos ámbitos, en la gestión de la producción y en el entorno de fabricación trata de asignar recursos (máquinas, herramientas, personas y centros de trabajo) a los trabajos a realizar y fijar las fechas de comienzo. Con ello se pretende conseguir un funcionamiento del sistema de fabricación óptimo o suficientemente bueno. (Laviós Villahoz, 2013).

Dentro del proceso de planificación de la producción en la empresa, la programación de talleres se relaciona con el nivel más bajo de la planificación (planificación a muy corto plazo).

Los pedidos u órdenes de producción pueden tener su origen en:

- los MRP o cualquier otro sistema de planificación de la producción en sistemas bajo inventario
- los clientes en sistemas bajo pedido

Por lo tanto, la programación de operaciones puede verse como una fase de preparación de las actividades productivas, después de la planificación maestra y del cálculo de necesidades. El resultado es, por consiguiente, una asignación de los recursos a las operaciones a realizar y un calendario.

De esto se deriva la gestión del taller propuesta por (Companys Pascual, 1989) que se puede decir que, en el caso más general, consta de cuatro pasos o fases:

1. Revisión, autorización y priorización de los pedidos:
   - Comprobar si puede procederse a la emisión del pedido (u orden de producción) en función de :
     - Disponibilidad de materiales
     - Disponibilidad de capacidad en los centros de trabajo necesarios
   - Confeccionar el pedido con la información existente: ítem, cantidad, materiales, inspecciones, ruta, etc.
   - Emitir el pedido, el cual pasa a estar *en curso* hasta que se finaliza
2. Asignación de carga a los centros de trabajo: consiste en asignar los pedidos a los centros de trabajo. Para ello se debe indicar qué operaciones se realizarán en cada uno de esos centros de trabajo.
3. Secuenciación y temporización de las operaciones de los pedidos en los centros de trabajo:
   - Secuenciar: determinar el orden de ejecución de las operaciones en cada centro de trabajo
   - Temporizar: determinar el calendario de ejecución de las operaciones en cada centro de trabajo, indicando tanto su fecha de comienzo como su fecha de finalización.
4. Control (seguimiento y realimentación): consiste en obtener información del estado de ejecución de los pedidos, compararlo con lo que se había programado inicialmente y en el caso de que existiesen desviaciones significativas, reprogramar y tomar las medidas correctoras oportunas.

Los pasos intermedios, enumerados como 2 y 3, serían los correspondientes propiamente a la programación de operaciones en la gestión de talleres.

En el apartado 1.4 se verá una clasificación de los tipos de producción existentes, sin embargo, cabe destacar que, independientemente de la configuración del entorno de producción que se esté estudiando, un problema de programación de operaciones puede ser descrito utilizando la siguiente estructura, la cual fue propuesta por (Pinedo, 2009):

- Un conjunto de recursos. Hace referencia a las máquinas, personas, centros de trabajo o instalaciones que permiten realizar una determinada actividad. Son los sistemas físicos o lógicos capaces de realizar las operaciones de proceso sobre los trabajos.

- Un conjunto de trabajos (pedidos u órdenes), compuestos por una serie de operaciones que necesitan de los recursos para poder ser procesados.

- Unas restricciones o conjunto de condiciones a satisfacer. Las restricciones pueden hacer referencia:

  - a los recursos: principalmente referidas a su configuración y su capacidad.
  - a las operaciones: las principales son restricciones de sucesión y de localización temporal.

    - Restricciones de sucesión: existe una restricción de sucesión entre dos tareas Tar1 y Tar2, si la operación Tar1 no puede comenzar antes de que finalice la operación Tar2. Se denota por "Tar1>Tar2"
    - Restricción de localización temporal: hay una restricción de localización temporal para una tarea i cuando la fecha de su ejecución t_i está limitada inferiormente por un valor α ($t_i>\alpha$), superiormente por un valor β ($t_i<\beta$) o por los dos valores a la vez ($\alpha<t_i<\beta$). α y β son las fechas límites.

- Una función objetivo: es la función a evaluar y cuyo valor se desea optimizar. Los criterios generales más frecuentemente utilizados son la minimización de los siguientes cuatro conceptos:

  - La duración total de fabricación.
  - Los tiempos de inactividad de las máquinas.

- El coste de la programación de la producción
- Los retrasos incurridos.

Como se verá más adelante, este trabajo se centrara en los problemas cuya función objetivo consista en reducir *el makespan* (C_{max}) o tiempo de finalización de la última operación en ser acabada.

Finalmente, de forma general, se puede destacar que la programación de operaciones permite:

- ✓ Proporcionar visibilidad del sistema de manera que se pueda conocer de antemano cuál va a ser la capacidad utilizada o los medios a usar, permitiendo tomar las decisiones más adecuadas.
- ✓ Proporcionar grados de libertad para poder modificar el propio programa adaptándolo a las circunstancias cambiantes. Es de vital importancia prever de antemano ciertas holguras en los elementos más críticos.
- ✓ Aportar una función de control, permitiendo evaluar el trabajo realizado en producción (el programa sirve como referencia para comparar con el resultado final obtenido, por lo que es importante que el programa sea lo más realista posible).

Es necesario señalar que en general muchos de los problemas que aparecen en programación de operaciones pueden llegar a resultar muy complejos. El origen de esta dificultad puede provenir de diferentes aspectos. A continuación se describen algunos de ellos:

- La complejidad de los cálculos. Un problema de programación de operaciones en la gestión de talleres se caracteriza por su naturaleza combinatoria, que hace que la búsqueda de soluciones óptimas para problemas grandes sea una labor inabordable en un tiempo razonable.
- La naturaleza distribuida del problema. Se trata de repartir *n* tareas entre varios recursos, pudiendo existir dependencias entre ellos.
- El número de restricciones. El problema está sometido a una serie de restricciones que pueden llegar a ser demasiado numerosas. Estas restricciones pueden tener naturaleza tanto física (restricciones funcionales de una máquina-herramienta, de una tarea a ejecutar, etc.), como temporal (fecha de fin de un trabajo, duración de tratamiento de una operación, tiempo de cambio de herramienta de una máquina, etc.).

1.3. Tipos de producción

En este apartado se explicarán los diferentes tipos de producción existentes en función tanto del producto como del proceso. La gran variedad en las tipologías de productos y procesos hace que determinados productos sean adecuados para que su producción se lleve a cabo mediante determinados tipos de proceso, resultando de ello determinados tipos de producción. De esta idea surge una matriz producto–proceso que fue inicialmente desarrollada por Hayes y Wheelwright (1979) y más tarde (Cuatrecasas, Diseño avanzado de procesos y plantas de producción flexibles, 2009) hizo una distinción de los tipos de producción resultantes. Sin embargo, en la siguiente figura se puede observar cómo dicha matriz ha sido rediseñada y completada al haberse añadido la producción por proyectos, la cual faltaba en la original:

Figura 1. Matriz producto–proceso

De esta manera, a partir de la anterior matriz, se puede llevar a cabo una distinción pormenorizada de las distintas configuraciones de producción existentes:

- Producción por proyectos:

  Este tipo de configuración (situada en la matriz en la parte superior izquierda) se basa, generalmente, en la fabricación de productos únicos de cierta complejidad que requieren una gran cantidad de *inputs* o materia prima. Estos tipos de productos deben fabricarse en un lugar definido debido a que es difícil o casi imposible transportarlos una vez terminados. Como resultado, y a diferencia de cualquier otro proceso productivo, los recursos que comprende deben trasladarse al lugar de operación para que, de este modo, no exista un flujo de herramientas y materiales, sino que sean los recursos técnicos y humanos quienes acudan al lugar de trabajo. De esta manera, las actividades y los recursos se gestionan como un todo y, por lo tanto, su coordinación, adquiere un carácter crítico. Destaca el interés por el control de los costes y las fechas de terminación del proyecto.

- Producción funcional a medida (Job Shop):

  Este tipo de configuración (situada en la matriz en la segunda posición) se caracteriza por un *layout* orientado al proceso (base de la producción en masa que se dio en el siglo XX). La principal particularidad que presenta es que en ella se producen lotes que pueden variar desde la unidad y pocas unidades de una amplia variedad de productos de poca o nula estandarización (por lo general son pedidos hechos a medida para el cliente, de naturaleza muy poco repetitiva, y por tanto se trata de realizar operaciones personalizadas para cada caso). Se requieren operaciones poco especializadas, las cuales son realizadas por un mismo obrero o por un grupo muy pequeño de trabajadores. Por consiguiente, en esta configuración se emplean equipos de escasa especialización, los cuales suelen agruparse en talleres o centros de trabajo según la función a realizar (esto es lo denominado orientación al proceso). Como los productos a fabricar son tan diferentes, los recursos han de ser flexibles y versátiles. Sin embargo, hay que destacar, que los centros de trabajo suelen estar integrados por personal

altamente cualificado. Aunque los equipos, por lo general, suelen estar inactivos si no se usan, los operarios deben estar permanentemente ocupados. Para lograrlo suele haber, por un lado, una cartera de pedidos pendientes lo que, sin embargo, provoca un alargamiento de los plazos de entrega y, por otro lado, una gran cantidad de stock de materiales y trabajos en curso. No obstante, este tipo de producción acarrea fuertes desequilibrios al producirse los denominados "cuellos de botella" en determinados puestos de trabajo cuya carga es superior a los demás.

Los plazos de entrega de este tipo de producción suelen ser dilatados debido a la alta personalización del producto requerido por el cliente. Como ejemplos de actividades orientadas a este tipo de producción, destacan aquellas empresas enfocadas a la personalización, centrándose en lotes pequeños y no en grandes volúmenes consiguiendo la estandarización de los productos.

Con este tipo de producción se obtienen bajos costes fijos.

- Producción funcional en lotes:

  A continuación en la matriz producto–proceso aparece la producción funcional por lotes. En esta configuración se producen menos variedad de productos que en la anterior, fabricándose por consiguiente volúmenes más elevados. Este modo de producción ya no es a medida, apareciendo ciertas limitaciones aunque la variedad siga siendo relativamente amplia. El proceso de obtención requiere más operaciones y éstas son más especializadas. Por ello, aunque la automatización de los procesos sigua siendo baja, los centros de trabajo suelen disponer de equipos algo más sofisticados. Las instalaciones se dividen, al igual que en el caso anterior, en diferentes talleres o centros de trabajo en los cuales se agrupan los equipos con funciones similares. El flujo de material, a pesar de ser intermitente como en el job shop, en esta configuración es regular.

  Las características básicas de este tipo de implantación son las propias de la configuración de la producción orientada al proceso: diversos y largos recorridos para el producto, múltiples actividades de manipulación y transporte, esperas, volúmenes importantes de stock, tiempos de entrega largos…

Sin embargo, la diversificación del producto, aún existente, se consigue a base de tiempos largos de proceso, abundancia de stock y actividades sin valor añadido.
Como diferencia con la producción job shop, en esta configuración por lotes se pueden reunir pedidos suficientemente similares en una sola orden de fabricación. De ahí la producción en lotes.

- Producción en flujo o cadena (*flow shop*):

  Siguiendo la diagonal de la matriz aparecen los tipos de producción en flujo. Esta configuración está orientada al producto. Este proceso de fabricación, también conocida como producción en masa por algunos autores, se adopta, en esencia, cuando se trata de fabricación de lotes grandes de pocos productos diferentes pero técnicamente homogéneos. Se trata de productos cuyo proceso de obtención en el centro de trabajo requiere una secuencia similar de operaciones, de forma que los ítems pasan a través de todos las máquinas en el mismo orden, estando, por tanto, los puestos de trabajo y sus máquinas y equipos dispuestos en flujo, uno tras otro.
  Tras la fabricación de un lote de un tipo de producto, se han de preparar las máquinas, útiles, herramientas y materiales para la producción de otro lote distinto y así sucesivamente. La variedad de outputs pues, suele ser baja y de una calidad más controlada para conseguir una producción masiva. En esta modalidad de configuración de la producción, se diferencian a su vez dos tipos en función de si el ciclo productivo se encuentra controlado por el operario o por los equipos de producción. En el segundo caso, en el que la cadena está automatizada, se persigue un volumen superior de outputs con una calidad elevada a un coste menor.
  Otra característica propia del *flow shop* es que los procesos ya no solo pueden, sino que deben estar equilibrados como resultado de la homogeneidad de las rutas de operación de los productos y la ausencia de trabajo por lotes que hace que sea más fácil resolver los problemas conocidos como “cuellos de botella”. Sin embargo, y precisamente por la dependencia entre centros de trabajo, o incluso máquinas, que supone el equilibrado, cualquier incidencia puede parar la línea de producción. Se tiene un inventario de producto en proceso (WIP) relativamente bajo en relación a la salida u *output*.

En este tipo de configuración, los quipos suelen estar especializados según el tipo de operación a realizar sobre el ítem.

- Producción en flujo continuo:

  Por último, en el extremo inferior derecho de la diagonal se halla la producción en flujo continuo. Esta modalidad de producción es la menos flexible de todas (Características de un Proceso Productivo Flow Shop, GEO Tutoriales, 2015). En ella cada máquina y equipo están diseñados para realizar siempre la misma operación y, normalmente, preparados para aceptar de forma automática el trabajo que les es suministrado por una máquina precedente, la cual incluso puede haber sido especialmente diseñada para alimentar a la máquina que la sigue. El diseño del producto es muy estable y el flujo del material es continuo y sincronizado. De esta manera se pretende obtener un producto altamente estándar y un gran volumen de *outputs*, de una gran calidad y un coste muy bajo. Sin embargo, la variedad de los productos deberá ser muy pequeña, así como los cambios en el diseño de los mismos, teniendo en cuenta que se pretende conseguir la mínima intervención de operarios en la línea de producción, solo encargados por lo general de operaciones de control y de mantenimiento, así como de la alimentación y vaciado de las máquinas, cuando tampoco sean automáticos. (Sistema de Producción Flow Shop, Salazar, 2015).
  Como característica adicional, por tanto, se puede decir que el producto obtenido no puede medirse en unidades discretas sino en flujo continuo.
  Para que esta modalidad de producción sea costeable y eficiente, por lo general, se requieren jornadas laborales de 24 horas al día. (Configuraciones Productivas, Ibarra Mirón, 2015).
  (Sistemas Productivos, Cruells et al., 2015).

1.4. Los nuevos tipos de producción en el marco de la producción ajustada

Existen, además de los descritos en el apartado anterior, otra serie de modalidades de producción más avanzadas y que han sido recientemente desarrolladas. Estos tipos de producción buscan alcanzar dos objetivos principales:

- Por un lado, llevar a cabo el proceso de producción con el mínimo uso de recursos y con la realización del menor número de actividades posible.
- Por otro lado, trabajar con lotes de producción pequeños y con una elevada variedad de producto.

En la siguiente figura se muestra la matriz producto-proceso completa, que incluye los dos tipos de producción más avanzados y más recientemente desarrollados, como se ha dicho. En esta matriz, basada en la que diseñó (Cuatrecasas, Diseño avanzado de procesos y plantas de producción flexibles, 2009), se puede apreciar cómo los nuevos métodos de producción se colocan en la zona inferior izquierda. Esto es así por las dos siguientes razones:

- Los volúmenes de producto bajos con elevada variedad de producto, se hallan en la zona izquierda de la matriz producto-proceso.
- Para lograr el menor número de actividades innecesarias y, por tanto, el menor empleo de recursos, es necesario implantar procesos con orientación al producto (en flujo lineal), en los que se procesen los ítems de unidad en unidad evitando los lotes. En la matriz producto-proceso estos modos de producción se hallan en la zona inferior de la misma.

Así pues, como se observa a continuación, los nuevos tipos de gestión en línea con las tendencias de la producción ajustada que se van a explicar, están situados en la zona inferior izquierda de la matriz y, por tanto, fuera de la diagonal. Aparecen en un tono rojo para diferenciarlos de los ya explicados en el apartado anterior, los cuales se presentaban en tonalidades azuladas.

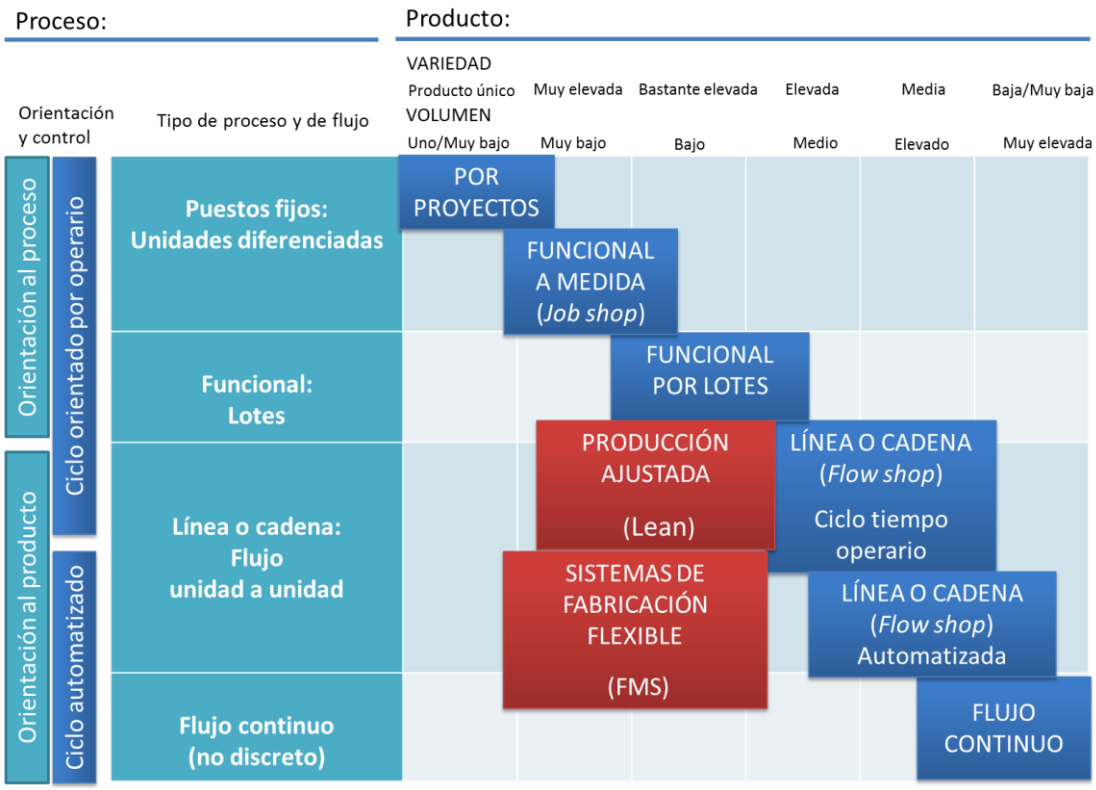


Figura 2. Matriz producto-proceso completa

Estas dos modalidades más recientes de producción son:

- Producción lean:

  Este tipo de producción es gestionada mediante el JIT (*Just in time*, Justo a Tiempo en español). Como bien dice (Cuatrecasas, Diseño de procesos de producción flexible, 1996), en la actualidad está ampliamente aceptado el gran potencial que tiene el enfoque JIT, el cual surgió en el seno de Toyota a partir de los años 50 y se le atribuye a Taiichi Ohno. La filosofía JIT se basa en producir justo lo que se quiere, cuando se necesita, consiguiendo una excelente calidad y sin desperdiciar recursos del sistema.
  La producción lean, también conocida por algunos autores como producción esbelta o producción ajustada, trata de abarcar eficientemente todas las características de competitividad:
  - Calidad. Se busca la calidad perfecta a la primera mediante la búsqueda de cero defectos que conlleva la detección y solución de los problemas en su mismo origen.

- Tiempo. El proceso de mejora continua al que se ve sometido este tipo de sistema productivo repercute, también, en el tiempo de producción necesario.
- Coste. Se intenta eliminar todas las actividades que no son de valor añadido.
- Flexibilidad. Se busca producir de manera rápida gran variedad de productos.
- Funcionalidad. Se habla de procesos *pull*, tirados por el cliente en el sentido de solicitados. Así solo se produce lo necesario, eliminando stocks. Se intenta también lograr una buena relación a largo plazo con los proveedores a partir de acuerdos para compartir riesgos e información.
- Se puede hablar incluso de innovación.

Las tres primeras características las presentan todos los tipos de producción con implantación en flujo existentes, y las tres últimas son características propias, dado que trabaja con pequeños lotes y alta variación del producto.

Como contrapartida, la organización que requiere implementar y su gestión presentan una alta complejidad.

- Sistemas de fabricación flexibles:

Estos métodos de producción también conocidos por su término en inglés *flexible manufacturing systems* (FMS), nacidos en la década de los 80 como respuesta al avance inexorable del JIT son, en la actualidad, una alternativa más entre los tipos de producción. Estos sistemas se basan en el uso intensivo de la tecnología por medio de máquinas y equipamientos automatizados y programables informáticamente, para lograr, con rapidez y de forma automática, adaptarse a las variaciones que puedan exigir los productos y los procesos. De este modo se optimiza la fabricación por lotes, se reduce el material en curso puesto que solo se fabrica lo necesario, y se mejora la gestión de la producción. Como desventaja, sin embargo, se puede hablar de un alto coste inicial tanto en equipos como en sistemas de transporte o software, y de la necesidad de organizar la producción por familias de piezas, lo que puede resultar algo tedioso y complicado.
Para entender mejor los sistemas de fabricación flexibles es necesario hablar de los sistemas *CAD* y los sistemas *CAM*, usados en este modelo de producción. Los sistemas *CAD* (*computer assited design)* permiten diseñar los productos, pudiendo introducir en ellos todas las

mejoras que sean necesarias. Por otro lado, los sistemas *CAM* (*computer assisted manufacturing*) pueden programar las tareas a realizar por los equipos o las máquinas (normalmente máquinas de control numérico, conocidas como *CNC*). Relacionando ambos conceptos se obtiene los conocidos como sistemas CAD-CAM. Mediante los mismos, se puede llevar a cabo la producción de productos diseñados informáticamente por medio de procesos programados también vía informática e introducir así toda la flexibilidad que se estime conveniente.

Finalmente, se puede hacer una evaluación conjunta de estos dos métodos de producción, hablando de características de producción que conciernen a las dos modalidades.

Para empezar, es necesario resaltar la idea de que la producción llevada a cabo en entornos fuertemente computarizados recibe el nombre de *CIM* (*computer integrated manufacturing*). Esto permite una gran flexibilidad, pudiendo alcanzar, de manera muy sencilla, la mayoría de las características de competitividad. Sin embargo, la fuerte complejidad derivada de estos sistemas, hace que en algunos casos se reserven para tipos de producción muy concretos.

Tanto la producción *lean* como los sistemas de producción flexible, se desarrollan mediante la implantación de células flexibles. Las células flexibles son implantaciones en flujo preparadas especialmente para:

- Llevar a cabo la producción de pequeños lotes de producto
- Cambiar rápidamente a otras variantes del mismo
- Poder ser utilizadas tanto para la fabricación como para el montaje.

Hay que tener en cuenta que en los sistemas tradicionales de producción en masa, solo existían cadenas de montaje. Este diseño de procesos obedece a los principios de producción en flujo, unidad a unidad. Cabe destacar, además, que en la línea de producción que compone la célula, ésta suele tener forma de U, para que el operario tenga cerca cualquier máquina u operación sin hacer movimientos inútiles.

En la siguiente figura podemos ver una línea en forma de U planteada por (Cuatrecasas, Diseño de procesos de producción flexible, 1996):

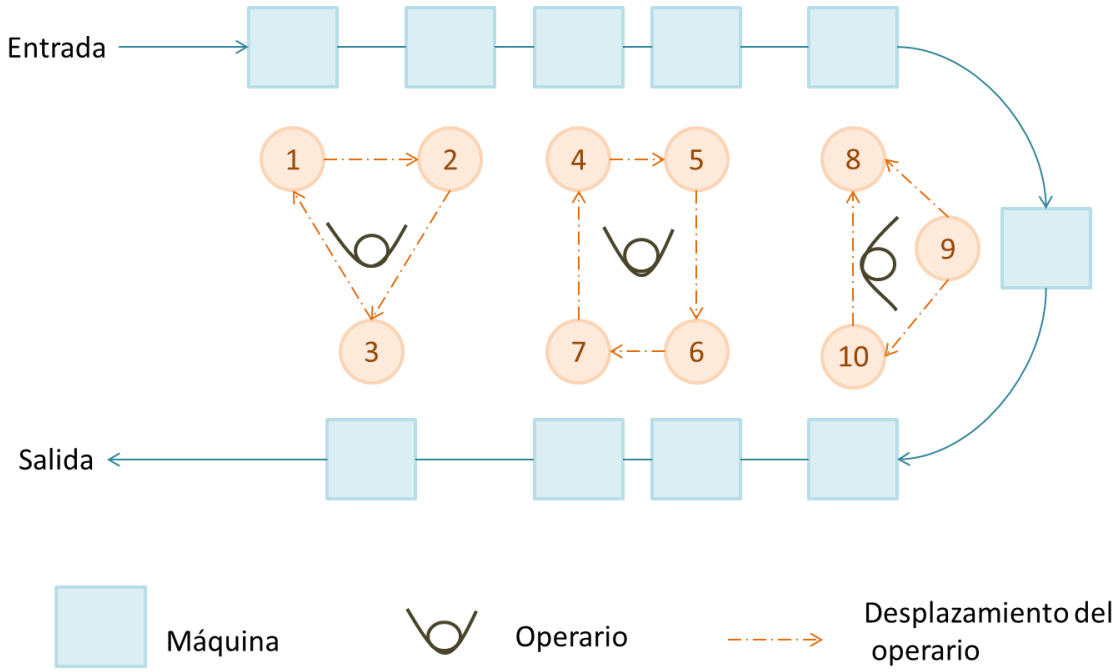


Figura 3. Línea en forma de U.

1.5. Clasificación de los problemas de programación de talleres

A continuación se hace referencia a la notación usada para plantear el problema de producción. Como base, se ha usado la propuesta por (Pinedo, 2009). En un problema de producción se dispone de un número finito *M* de máquinas en los que se han de realizar un conjunto *N* de trabajos (pedidos). Normalmente se utiliza el subíndice *h* para hacer referencia a una máquina mientras que el subíndice *j* hace referencia a un trabajo. Cada trabajo consiste en una serie de operaciones (tareas). El subíndice conjunto *ij* hace referencia a la operación i del trabajo j.

Cada trabajo o pedido *j* se describe por una serie de parámetros:

- Fecha de entrega comprometida (D_j): es la fecha comprometida con el cliente. La orden de trabajo puede ser completada

posteriormente pero incurriendo en una penalización en función del retraso. La D proviene del término inglés *due date.*

- Fecha más temprana de lanzamiento (r_j): es el momento a partir del cual el trabajo o pedido es lanzado a planta y sus operaciones pueden comenzar a ser ejecutadas. La r procede del término inglés *release time*, aunque también es conocida como *ready time*.
- Peso (w_j): es un factor de prioridad que indica la importancia del trabajo o pedido respecto a los demás. La w proviene de la palabra *weigth*, peso en inglés. Puede venir representado por el costo de mantener al pedido j en el sistema de fabricación durante una unidad de tiempo, el costo de posesión o de inventario, o puede ser incluso representativo del valor ya aportado a ese trabajo por las operaciones realizas sobre él.

Teniendo en cuenta las operaciones de cada pedido se tienen los siguientes parámetros:

- Tiempo de procesado (p_{ijh}): es el tiempo de ejecución de la operación i del pedido j en la máquina h. La p proviene del término anglosajón *processing time*.
- Duración (d_{ijh}): es la duración de la operación i del pedido j en la máquina h.

Un problema de producción se caracteriza mediante un triplete α/β/γ, donde:

- α: hace referencia a la configuración del taller. Tiene una entrada única.
- β: hace referencia a las características del proceso. Puede no tener ninguna entrada, una o varias.
- γ: es la función objetivo a conseguir (normalmente es de minimizar) y generalmente tiene una entrada.

A continuación se detalla una clasificación de los distintos problemas de programación de talleres atendiendo a diferentes aspectos: la configuración del taller, las características del proceso y la función objetivo a minimizar.

1.5.1. Según la configuración del taller (α)

Podemos clasificar las distintas configuraciones del taller por el número de operaciones que componen los trabajos o pedidos a programar:

- {o, P, Q, R} Cada trabajo consta de una sola operación que puede ser realizada en una o varias máquinas:
  - Una máquina (o): cada trabajo se realiza en una sola máquina.
  - Varias máquinas en paralelo, que pueden ser:
    - Idénticas (P_M): cada trabajo puede ser realizado en cualquiera de las *M* máquinas.
    - Uniformes (Q_M): hay *M* máquinas con diferentes velocidades. La velocidad de cada máquina es v_h. El tiempo que una máquina h tarda en realizar una operación i se calcula como tc_j/v_h
    - No relacionadas (R_M): hay *M* máquinas trabajando en paralelo. La velocidad de la máquina *h* para el trabajo o pedido j es v_{jh}. El tiempo que una máquina *h* tarda en realizar una operación i se calcula con la expresión p_j/v_{jh}.
- {G, F, FF, J, FJ, O}. En estos casos, cada trabajo j está formado por un conjunto de operaciones *(ij)*. Las operaciones están vinculadas entre sí mediante relaciones de precedencia. Las máquinas son dedicadas, es decir, especializadas en ciertas operaciones. En función de estas características se pueden dar los siguiente casos:
  - El modelo general (G): posee todas las características citadas anteriormente, siendo el resto son casos particulares de éste.
  - *Flow shop* (F_M): en este modelo hay *M* máquinas en serie. Cada trabajo ha de realizarse en cada una de las M máquinas y todos los trabajos siguen la misma ruta. Si además, los trabajos deben seguir la misma secuencia en todas las máquinas nos encontramos ante un subtipo del *flow shop* que se denomina *permutacional*. En ese caso el parámetro β tomaría el valor *prmu*.
  - *Flow shop flexible* (FFc): este modelo es la generalización del *flow shop* con máquinas en paralelo. En lugar de disponer de *M* máquinas, disponemos de C centros de trabajo en cada uno de los cuales hay varias máquinas funcionando en paralelo. Cada

trabajo ha de pasar por cada uno de estos centros de trabajo siguiendo siempre el mismo orden. En la literatura este tipo de problema también es conocido como *hibrid flow shop, taller de flujo híbrido* o *multiprocessor flow shop*.

- *Job shop* (J_M): en este caso se dispone de *M* máquinas dedicadas, es decir, especializadas en ciertas operaciones como se ha dicho anteriormente. Cada trabajo sigue una determinada ruta. Se puede distinguir, por tanto, entre los problemas en los que los trabajos o pedidos solo pasan una vez por cada máquina y aquellos en los que un trabajo puede pasar más de una vez por una determinada máquina. En este último caso el parámetro β deberá de indicar que se trata de un problema con recirculación *rcrc*.
- *Job shop flexible* (FJ_C): es una generalización del entorno *job shop* con máquinas en paralelo. En lugar de disponer de *M* máquinas, disponemos de *C* centros de trabajo y en cada centro se dispone varias máquinas trabajando en paralelo. Se puede hacer las mismas consideraciones que en el caso *job shop*. Es importante tener claro este concepto pues es el modelo en el que se concentrará el presente trabajo más adelante.
- *Open shop* (O_M): se dispone de *M* máquinas, teniendo cada trabajo tiene que pasar por todas ellas. El tiempo de proceso en algunas máquinas puede ser cero y a diferencia de los otros casos no existen restricciones de precedencia entre las operaciones. Se trata de encontrar la ruta para cada trabajo o pedido teniendo en cuenta que cada uno de ellos puede tener una ruta diferente.

1.5.2. Según las características del proceso (β)

Este parámetro define las características del proceso, tanto de los recursos como de los trabajos o pedidos. Hace referencia a las posibles restricciones y condiciones que pueden aparecer en estos problemas. Algunos de los posibles valores más importantes que puede tomar son:

- Fecha de lanzamiento (*r*): si este símbolo aparece, un trabajo j no podrá empezar a realizarse antes de r_i.
- Posibilidad de no continuidad o *preemptive* (*prmp)*: este símbolo indica que no es necesario completar una tarea en una determinada máquina una vez que ha comenzado. Si el símbolo *prmp* no está incluido las tareas han de realizarse de forma continua hasta su finalización una vez que han comenzado (*non-premptive*).
- Recirculación (rcrc): indica que un trabajo puede pasar por una misma máquina más de una vez.
- Restricciones de precedencia (*prec)*: indican que existe algún tipo de precedencia entre tareas en aquellos casos donde no estén ya definidas. Las precedencias pueden ser de varios tipos:
    - *Chain*: cuando una tarea puede tener como máximo una tarea precedente y una tarea sucesora
    - *Intree*: cuando una tarea puede tener más de una tarea precedente
    - *Outtree*: cuando una tarea puede tener varias tareas sucesoras
- Tiempos de preparación dependientes de la secuencia (*i)*: indica que el tiempo de preparación de una tarea depende de la operación que ha sido programada previamente en la misma máquina. En el caso de que este tiempo de preparación sea independiente de la secuencia, se incluirá en el tiempo de proceso de la operación.
- Familias de producto (*fmls*): en este caso los *N* trabajos están agrupados en *F* familias de producto. Lo que caracteriza a este sistema es que en una máquina, cuando se termina un producto y se comienza otro de la misma familia, no existen tiempos de preparación, mientras que si el producto no pertenece a la familia del producto procesado anteriormente hay que considerar tiempos de preparación.
- Producción por lotes (*batch (b)*): una máquina puede realizar *b* operaciones de forma simultánea. Los tiempos de ejecución de estas operaciones no tienen porqué ser iguales. El tiempo de operación del lote es el tiempo de la operación más larga.
- Paradas de máquina (*brkdwn*): también llamadas restricciones de disponibilidad de máquina. Esta restricción indica que las máquinas no van a estar disponibles de forma continua. Las paradas pueden ser previsibles y fijas, como en el caso de que existan turnos u

operaciones de mantenimiento programado, o imprevisibles, como en el caso de averías.

- Idoneidad de máquina (*Mij*): este caso se aplica a máquinas en paralelo. Cuando no todas las máquinas pueden realizar la operación *ij* y solo las máquinas del conjunto *Mij* pueden realizarlo.
- Permutación (*prmu*): esta restricción se da en el entorno *flow shop*. Indica que los trabajos deben seguir la misma secuencia en todas las máquinas.
- Bloqueo de máquina (*block*): se da cuando dos máquinas que operan de forma sucesiva tienen un *buffer* (cola) intermedio de capacidad limitada. Esta restricción impide que una operación pueda empezar en la máquina anterior al *buffer* hasta que éste no tenga capacidad libre.
- Sin esperas (*nwt*): esta restricción indica que dos operaciones sucesivas han de realizarse de forma continua, de forma que el tiempo de finalización de una operación coincida con el tiempo de comienzo de la siguiente.

1.5.3. Según la función objetivo a minimizar (γ)

Las funciones objetivo a optimizar son función de los tiempos de finalización de las operaciones (c_{ij}). Estos tiempos vienen determinados por el programa de producción. El tiempo de finalización de la última tarea de un trabajo es representado por C_j.

$$C_j = \max_i \ (c_{ij})$$

donde c_{ij} es el tiempo de finalización de cada operación o tarea *i.*

Es posible definir diferentes variables que dependen del tiempo de finalización de los trabajos. Con ellas definiremos distintas funciones objetivo. A continuación se citan las más representativas:

I. Tiempo de flujo (*Flow time)*: es el tiempo del pedido en el taller

$$F_j = C_j - r_j$$

II. Huelgo (*Lateness*): es la diferencia entre el tiempo de finalización de la última tarea del pedido o trabajo C_j y la fecha de entrega comprometida D_j. Puede tener tanto valor positivo como negativo.

$$L_j = C_j - D_j$$

III. Retraso (*Tardiness*): es la diferencia positiva entre el tiempo de finalización de la última tarea del pedido o trabajo C_j y la fecha de entrega comprometida D_j. Si no hay retraso vale cero.

$$T_j = \max(0, C_j - D_j) = max(0, L_j)$$

IV. Adelanto (*Earliness*): es la diferencia positiva entre la fecha de entrega comprometida y el tiempo de finalización del pedido. Si no hay adelanto vale cero.

$$E_j = \max(0, D_j - C_j)$$

V. Penalización unitaria: vale uno si hay retraso y cero en otro caso.

$$U_j = \begin{cases} 1 & si\ C_j - D_j > 0 \\ 0 & en\ otro\ caso \end{cases}$$

Las *funciones objetivo* se definen en base a las anteriores variables. Se puede clasificar las funciones objetivo como *regulares* o *no regulares*.

Las regulares son aquellas funciones que no decrecen cuando cualquiera de los C_j aumenta. Las más usadas son:

- *Makespan* (C_{max}): es el tiempo de finalización de la última operación en ser acabada.

$$C_{max} = \max_j\ (C_j)$$

- Retaso máximo *(T_{max})*:

$$T_{max} = \max_j\ (T_j)$$

- Tiempo de flujo máximo *(F_{max})*:

$$F_{max} = \max_j\ (F_j)$$

- Máximo huelgo (L_{max}): máxima separación de la finalización de los trabajos respecto su fecha de entrega.
- Tiempo de flujo medio o *avarage tardiness* (AFT). Su reducción implica la reducción del WIP (*work in process* o trabajo en curso).

$$AFT = \frac{1}{N}\sum F_j$$

- Retraso medio o *Avarage Tardiness* (AT).

$$AT = \frac{1}{N}\sum T_j$$

- Suma ponderada de los tiempos de finalización o *weighted flow time* (WFT).

$$WFT = \sum w_j F_j$$

- Suma ponderada de los retrasos o *total weighted tardiness* (WT).

$$WT = \sum w_j T_j$$

- Suma ponderada de los retrasos al cuadrado o *total weighted squared tardiness* (WST).

$$WST = \sum w_j T_j^2$$

- Suma ponderada de los trabajos retrasados o *Number of Tardy Jobs* (NTJ).

$$NTJ = \sum U_j$$

También son de interés algunas funciones no regulares. Una de las más utilizadas es la suma ponderada de los retrasos y los adelantos *Total Earliness and Tardiness* (TWET) en la que los pesos asignados al adelanto (w_i') y al retraso (w_i'') pueden ser diferentes.

$$TWET = \sum w_i' E_i + \sum w_i'' T_i$$

Como se ha indicado con anterioridad, en este trabajo se hará especial hincapié en la función objetivo de minimizar el $C_{máx}$.

1.6. Problema *job shop* flexible

Como también se ha dicho con anterioridad, este trabajo va a estar centrado en el problema de *job shop* flexible, más conocido por los términos en inglés *Flexible Job Shop Problem* de donde derivan las siglas *FJSP.*

Este problema es una extensión del clásico problema de *job shop*. Mientras que el problema *job shop* clásico (JSSP) se basa en que cada máquina es capaz de ejecutar una sola operación concreta, el FJSP permite realizar una operación en cualquier máquina de un centro de trabajo. El problema radica en asignar cada operación a la máquina correspondiente y priorizar las operaciones en dichas máquinas de manera que resulte óptimo, minimizado el tiempo máximo de finalización de todas las operaciones (*makespan*). (Flexible Job Shop Problem, Mastrolilli, 2015).

Existe una gran variedad de problemas que se dan en el mundo real y que pueden ser modelados como FJSP. Estos existen, por ejemplo, a la hora de optimizar las operaciones de una grúa o en la programación de sistemas de fabricación reales.

Se puede decir, de una manera clara y sencilla, que las características de un problema job shop flexible son las siguientes:

- El número de pedidos o trabajos y el número de máquinas vienen fijados.
- Cada pedido o trabajo consiste en na secuencia fija de operaciones.
- Ciertas operaciones pueden ser ejecutadas solamente en ciertas máquinas.
- El tiempo de procesado de una operación puede variar según sea ejecutada en una máquina o en otra.
- Cada máquina solo puede ejecutar cada operación de una en una.

1.7. Complejidad computacional

En este apartado se hablará de la complejidad computacional o estudio del número de recursos computacionales (tiempo, memoria…) necesarios para resolver diversos problemas de cómputo. La teoría de complejidad computacional es una rama de la teoría de la computación que estudia la clasificación de los problemas computacionales de acuerdo a su dificultad inherente en diversas clases de complejidad. Por tanto, se podría decir que las clases de complejidad son el conjunto de problemas que necesitan de unos recursos equivalentes para resolverlos. Determinar la complejidad de un problema no lo resolverá de manera directa, sin embargo va a permitir decidir qué tipo de método es el más adecuado para su resolución.

Esta teoría fue desarrollada inicialmente para el estudio de problemas algorítmicos, pero su aplicación ha sido de gran utilidad en optimización combinatoria. La optimización combinatoria (también llamada optimización discreta) se encarga del estudio de los problemas de optimización donde el espacio de soluciones factibles es discreto, y cuyo objetivo es encontrar la mejor solución.

1.7.1. Conceptos previos

Para entender dicha clasificación, es necesario tener claro dos conceptos: *problema de optimización* y *problema de decisión*.

- En los problemas de optimización la solución de una instancia del problema consiste en el valor de las variables que hacen óptima la función objetivo.
- Los problemas de decisión son aquellos para los que una solución a una instancia del problema puede tomar sólo dos valores: sí o no (o de manera más formal 1 ó 0). De forma poco ortodoxa se puede decir por tanto que este tipo de problemas está caracterizado por la condición de que solo admite dos variables como salida.

Otro término a definir, antes de abordar los tipos de clases de complejidad, es el de *algoritmo*. Se puede definir un algoritmo como un procedimiento paso a paso para resolver el problema que se está tratando. El

algoritmo más eficiente será el que nos proporciona la solución al problema con mayor rapidez ya que el tiempo es el recurso computacional que cobra a menudo mayor relevancia, aunque de una manera más general haga referencia a todos los recursos computacionales necesarios para resolver el problema.

1.7.2. La máquina de Turing

Para estudiar la complejidad de los problemas se suele usar un modelo abstracto de computador llamado máquina de Turing. La máquina de Turing es una máquina de cálculo teórica que sirve como modelo idealizado del cálculo matemático. Se caracteriza por ser una máquina que manipula símbolos en una cinta según una tabla de reglas. En la máquina de Turing se realiza una acción, ya sea lectura o escritura, a partir del estado actual y de las variables de entrada. Por tanto, se compone de una cinta infinitamente larga donde se pueden leer y escribir símbolos, una cabeza lectora/escritora que apunta a un posición concreta de la cinta y se puede mover, y un control de estado finito.

Según las reglas fijadas se puede conseguir que la máquina de Turing calcule diferentes funciones. Con este mecanismo tan sencillo se puede computar cualquier función computable en un lenguaje d programación (Java, C...).

Hay que distinguir entre dos tipos de máquina de Turing: *determinista* y *no determinista*. En el primer tipo existe una única acción posible a tomar por la computadora dados el estado actual y las variables de entrada. En el segundo tipo, puede, sin embargo, existir más de una posible combinación de actuaciones. Se podría definir una máquina de Turing no determinista como varias máquinas en paralelo que funcionaran de manera simultánea pero sin comunicarse entre sí en ningún momento.

1.7.3. Clases de complejidad

Una vez definida la máquina de Turing y atendiendo a que una clase de complejidad, como se ha indicado con anterioridad, es un conjunto de problemas que poseen la misma complejidad computacional, podemos distinguir entre:

- Clase P (*Polynomial)*. Se caracteriza por existir al menos un algoritmo polinómico para una máquina Turing que resuelva el problema.

- Clase NP (*Non – deterministic Polynomial*). Este conjunto de problema se caracteriza por poder ser resueltos en tiempo polinómico por una máquina de Turing no determinista. En esta clase se puede aplicar un algoritmo polinómico para comprobar si la solución es válida o no. Esta característica permite métodos de resolución donde se van aceptando o desestimando soluciones hipotéticas.
- Clase NP – *complete* o completos. Incluye problemas tipo NP de extrema complejidad. Muchos de los problemas que se plantean en dirección de operaciones son NP – Completos. Se cree que no existen algoritmos polinómicos para resolverlos, pero todavía no se ha demostrado, ya que los problemas de *job shop* crecen de forma exponencial (explosión combinatoria)
- Clase NP – *hard*. Esta clase está formada por el conjunto de problemas que son al menos tan difíciles como el problema más complejo de la clase NP. Los problemas de la clase NP no tiene porqué ser problemas de decisión, esta clase también incluye problemas de optimización u otros.

1.7.4. Relación entre clases y la complejidad de los problemas de *scheduling*

Uno de los grandes enigmas en ciencias de la computación consiste en responder al siguiente problema de decisión: ¿P=NP?, es la llamada conjetura P≠NP. Intuitivamente se puede pensar que P es un subconjunto de NP ya que

cada problema de decisión resuelto por un algoritmo determinista en tiempo polinómico también podría ser resuelto por un algoritmo no determinista. Lo más probable es que los problemas de clase NP - completo no formen parte de la clase de complejidad P debido a que, en ese casos si existiera una solución polinómica para algún problema NP – completo, todos los problemas de la clase NP tendrían también una solución en tiempo polinómico. Ésta es, por tanto en teoría, una de las cuestiones sin resolver de la complejidad computacional.

En la siguiente figura podemos ver cómo se puede representar la conjetura P≠NP de una manera visual.

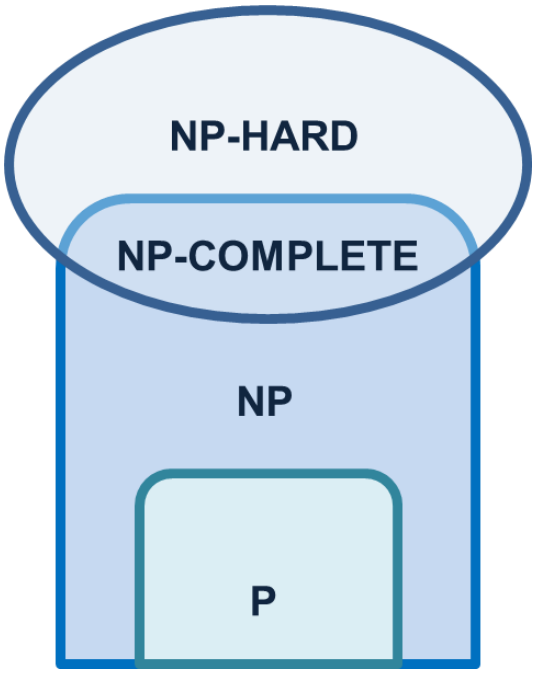


Figura 4. Clases de problemas si P≠NP.

Como se ha dicho en el apartado anterior, los problemas NP – completos incluyen a la mayoría de los problemas relacionados con la dirección de operaciones y además son NP – *hard*, es decir, tienen la propiedad de que todos los problemas NP pueden ser reducidos polinómicamente a ellos.

1.8. Conclusiones

Como se ha explicado en este capítulo, el problema de programación de operaciones dentro del ámbito de fabricación consiste en, dadas unas órdenes de trabajo o pedidos y un conjunto de recursos de los que se puede disponer, realizar los siguientes pasos principales:

- Asignar los recursos que van a realizar las operaciones que componen las órdenes de trabajo.
- Secuenciación y temporización de las operaciones de los pedidos en los centros de trabajo.

La matriz producto-proceso permite caracterizar las principales modalidades de producción. Esta, además, puede ser completada con los tipos de producción más recientemente desarrollados para, así, tener una información más completa.

Existiendo, por tanto, esa gran variedad de alternativas posibles, se han desarrollado propuestas de notación que facilitan su estudio y caracterización.

El problema *job shop* flexible, en el que, como se ha mencionado más veces, se centra este trabajo, es una extensión del clásico problema de *job shop*. El FJSP permite realizar una operación en cualquier máquina de un centro de trabajo. La dificultad puede surgir en asignar cada operación a la máquina correspondiente y priorizar las operaciones en dichas máquinas de manera que resulte óptimo, minimizado el tiempo máximo de finalización de todas las operaciones (*makespan*). Por lo tanto, uno de los aspectos más importantes a tener en cuenta en los problemas de programación de operaciones es su complejidad. Conocer la complejidad del problema nos va a permitir establecer de antemano qué tipo de estrategia debemos utilizar a la hora de buscar una solución al problema. Si el problema es fácil deberemos utilizar algún procedimiento exacto que permita obtener su solución, mientras, que si es difícil, esta estrategia no se puede usar. Es posible utilizar heurísticas que permitan encontrar una solución con rapidez o al menos encontrar una solución cercana al óptimo, a pesar de que no proporcionen una garantía formal de su eficacia. En el siguiente capítulo se presentarán los algoritmos documentados que se han propuesto para resolver este tipo de problemas.

CAPÍTULO 2. ALGORITMOS EXISTENTES PARA JOB SHOP FLEXIBLE

2.1. Introducción

Existen diversos métodos de resolución para los problemas de programación de operaciones. En este capítulo se realizará una descripción de los mismos, centrándonos siempre en el problema que se desea estudiar de una manera más exhaustiva en este trabajo, el problema de *job shop* flexible.

La complejidad de este problema de producción plantea dificultades en su resolución. Estas complejidades radican, principalmente, en que al ser NP-*hard*, hay serias sospechas de que no existen métodos para resolverlos de forma óptima en tiempo polinómico.

Este capítulo, por tanto, se estructura del modo que sigue: primero se distinguirá entre métodos exactos de optimización y métodos heurísticos, describiendo todas sus variantes; y, en segundo lugar, se explicará el concepto de robustez, tan importante a la hora de estudiar la resolución de los problemas de programación de operaciones.

2.2. Métodos exactos de optimización

Los métodos exactos son aquellos que garantizan la obtención de la solución que optimice un criterio de eficiencia concreto. (Araúzo Araúzo, 2007). Salvo para algunos casos determinados, estos métodos consisten en realizar una enumeración más o menos inteligente o guiada de todas las posibles soluciones existentes para el problema.

Sin embargo, como se ha comentado con anterioridad, debido a la complejidad de los problemas de programación de operaciones, el desarrollo de métodos exactos que encuentren la solución óptima en un tiempo razonable es muy difícil. En los métodos exactos el tiempo de resolución generalmente crece de manera exponencial respecto al tamaño del problema.

Dependiendo del escenario en el que se esté trabajando y del criterio de eficiencia que se desee optimizar, se puede optar entre diferentes procedimientos.

2.2.1. Programación lineal

Este procedimiento puede decirse que fue introducido durante la Segunda Guerra mundial. La programación lineal resuelve un problema indeterminado, formulado a través de ecuaciones lineales. Optimiza una función objetivo, la cual es también lineal, de forma que las variables de dicha función estén sujetas a una serie de restricciones expresadas mediante un sistema de inecuaciones lineales. Los programas lineales pueden resolverse en tiempo polinomial, lo que no ocurre con los enteros ni los mixtos. Los programas lineales enteros son aquellos que introducen la condición adicionas de que las variables deben tomar valores enteros. Sin embargo, si entre las variables hay valores enteros y reales, se habla de programa lineal mixto.

2.2.2. Ramificación y poda (*Branch&Bound*)

Esta técnica fue desarrollada en 1960 por Land y Doig. Hace referencia a un “árbol de soluciones”, donde cada rama lleva a una posible solución posterior a la actual. La ventaja que presenta este método respecto a otros es que el algoritmo detecta en qué ramificación las soluciones dadas ya no son óptimas para “podar” esa rama del árbol y no continuar desaprovechando recursos. La búsqueda comienza en el nodo raíz y continúa hasta el nodo hoja. Desde un nodo no seleccionado, la operación de

ramificación determina el siguiente conjunto de posibles nodos a partir del cual puede continuar la búsqueda. El procedimiento de poda selecciona la operación con la que continuar la búsqueda. Para ello, realiza una estimación usando el valor de la cota inferior y el valor de la mejor cota superior alcanzada hasta el momento.

De forma habitual, estos algoritmos toman como solución inicial una obtenida como resultado de algún método heurístico, lo que permite ahorrar un importante tiempo de computación.

2.2.3. Programación dinámica

Este método fue ideado en 1953 por Bellman. La programación dinámica plantea la solución como una sucesión de decisiones que intenta, reducir el tiempo computacional necesario para alcanzar la solución óptima. Se trata de resolver una serie de etapas (subdivisiones del problema). En cada una de ellas se van tomando decisiones simples hasta hallar una solución al problema original. Estas decisiones se toman en base a la información obtenida en todas las etapas anteriores, y su contribución a la función objetivo, consiguiendo así la solución óptima.

2.3. Métodos heurísticos

Un método heurístico es un conjunto de pasos que se deben realizar para obtener una solución de alta calidad en el menor tiempo posible para un determinado problema. Cualquier método de búsqueda que no tenga una garantía formal de encontrar la solución óptima puede ser considerado como heurística. Es adecuado aplicar estos métodos si no existen mejores alternativas. (Método Heurístico, Juarez, 2013).

Es en estos métodos heurísticos en los que se desea hacer más hincapié en el presente trabajo, por lo que se explicarán de manera más exhaustiva.

Se puede decir que los numerosos métodos heurísticos existentes comparten la característica de obtener soluciones aceptablemente buenas en tiempos razonables. A pesar de ello, cabe destacar que, a veces, no hay garantías de que la solución pueda hallarse en un tiempo razonablemente corto o incluso de que no se trate de una solución errónea. (Clasificación Métodos Heurísticos, Ramírez Izquierdo, 2015)

A modo aclaratorio, se puede decir que son necesarios los métodos heurísticos en las siguientes situaciones:

- Cuando la solución puede ser aproximada.
- Cuando el problema es de una naturaleza tal que no se conoce ningún método exacto para su resolución.
- Cuando, aunque exista un método exacto para resolver el problema, su uso computacional es demasiado costoso o inviable.
- Cuando es necesario un paso intermedio hacia otro procedimiento de resolución, normalmente un algoritmo metaheurístico.
- Cuando existen limitaciones de tiempo, espacio para almacenaje de datos, presupuesto, etc., que priorizan una solución rápida a costa de pérdida de precisión.

Existen, tal y como se presentan a continuación, dos tipos de métodos heurísticos: los constructivos y los de mejora.

2.3.1 Métodos heurísticos constructivos

Este tipo de heurísticas se utilizan para construir una solución que cumpla con las restricciones del problema mediante la elaboración de una única solución sin partir de una previa. Para ello se utilizan reglas que permitan la construcción de las soluciones de mayor calidad posible.

A continuación se destacan cuatro técnicas constructivas para la programación *job shop*: métodos de lanzamiento, algoritmos de inserción, programación hacia delante y hacia atrás, y métodos basados en cuellos de botella.

2.3.1.1 Métodos basados en reglas de lanzamiento

Cuando una máquina está disponible, se le debe asignar la operación que puede ser ejecutada en ella. Si hay más de una operación preparada para ser llevada a cabo, se debe seleccionar una de ellas de acuerdo a alguna regla que defina las prioridades entre operaciones. A estas reglas se les llama reglas de lanzamiento.

Las reglas de lanzamiento (*Dispatching Rules)*, en ocasiones también denominadas por algunos autores como reglas de prioridad o reglas de despacho, son reglas de muy sencillo funcionamiento y consisten en una serie de criterios que permiten priorizar las tareas que deben realizarse en cada máquina o en cada centro de trabajo.

Su denominación procede de su uso más inmediato: se deben tomar las decisiones sobre la operación que ha de ser llevada a cabo en un centro de trabajo o en una máquina, en ese mismo instante en el que la debe realizar. Cuando este procedimiento se utiliza para desarrollar programas previos a la fabricación, lo que se hace es una simulación del funcionamiento de la planta, bajo unos determinados criterios de toma de decisión. Las operaciones se irían añadiendo a la solución a continuación de las que ya han sido introducidas en pasos anteriores (como si las anteriores se hubieran ejecutado). Todo ello siempre respetando la disponibilidad de las máquinas y las relaciones de precedencia.

Sin embargo, es lícito mencionar que la calidad de los problemas obtenidos mediante reglas de lanzamiento es por lo general baja. Las reglas de prioridad o lanzamiento fueron los primeros intentos estudiados para resolver los problemas de programación de operaciones de una forma aproximada.

Uno de los primeros trabajos desarrollado sobre las reglas de lanzamiento fue el de Giffler y Thompson en 1960. Su algoritmo se convirtió en la base de la mayoría de las heurísticas basadas en reglas de lanzamiento que se desarrollaron con posteridad. Este método de Giffler y Thompson selecciona, dentro de las operaciones programables, aquellas pertenecientes a la máquina que antes puede terminar, y dentro de éstas, aquellas que pueden comenzar antes de esa fecha de finalización. Este método garantiza la obtención de cualquier problema activo, entre los que seguro se encuentra el óptimo, si se trabaja con funciones objetivo regulares.

Otro método también usado como base de las reglas de lanzamiento, es el método de lanzamiento. En este método, dentro de las operaciones

programables se elegirán aquellas que antes puedan comenzar. Se llama "de lanzamiento" porque su metodología es similar a la que se seguiría para ordenar, lanzar, la ejecución de las operaciones en los talleres en ausencia de programas: en el mismo momento en que una máquina queda libre, se selecciona una operación entre las que hay en cola pendientes de procesar, y se ordena su ejecución. Este método garantiza la obtención de programas sin esperas.

Para formalizar ambos métodos se ha de subdividir el conjunto de operaciones programables en M subconjuntos (E_1,E_2,..,E_M), uno por máquina. Cada subconjunto contendrá las operaciones programables de cada máquina.

En cada paso del método constructivo, en función de las operaciones ya programadas, se puede calcular:

- Para cada máquina:

  f_h: fecha en la que la máquina queda libre para realizar una nueva operación.

- Para cada operación *ij* programable:

  $r_{ij} = \begin{cases} r_j & si\ i = 1 \\ c_{i-1,j} & si\ i > 1 \end{cases}$: fecha de disponibilidad de la operación *ij*.

  $rp_{ij} = \max\left(r_{ij}, f_{h_{ij}}\right)$: fecha más temprana en la que puede comenzar la operación *ij*.

Ahora se distinguirán los pasos a seguir para cada uno de los dos métodos:

- <u>Método de lanzamiento</u>: en cada paso se seleccionará la máquina que antes pueda comenzar. De las operaciones que se realizan en esa máquina se tomarán como elegibles aquellas que antes puedan empezar.
  Para cada máquina *h* se calcula la fecha más temprana en que se puede comenzar una operación en la máquina *h*.

$$fp_h = \begin{cases} \min\limits_{ij \in E_h} \{rp_h\} & si\ E_h \neq 0 \\ \infty & si\ E_h = 0 \end{cases}$$

  Se selecciona la máquina h con menor fp_h (h_{lanz}):

$$fp^{min} = \min_{h}\{fp_h\} \qquad h_{lanz} = h \text{ tal que } fp_h = fp^{min}$$

De las operaciones que se realizan en h_{lanz} se toman aquellas que pueden comenzar en fp^{min} (aquellas cuyo $r_{ij} \leq fp^{min}$). Esas serán las operaciones elegibles:

$$E^* = \{ij \,/\, ij \in E_{h_{lanz}} \text{ y } r_{ij} \leq fp^{min}\}$$

- Método de lanzamiento: en cada paso se seleccionará la máquina que antes pueda finalizar. De las operaciones que se realizan en esa máquina se tomarán como elegibles aquellas que puedan comenzar antes de esa fecha más temprana de finalización.

  Para cada máquina *h* se calcula la fecha más temprana en que se puede finalizar una operación en la máquina *h*.

$$ff_h = \begin{cases} \min_{ij \in E_h}\{rp_h + d_{ij}\} & si\ E_h \neq 0 \\ \infty & si\ E_h = 0 \end{cases}$$

  Se selecciona la máquina h con menor fp_h (h_{GT}):

$$ff^{min} = \min_{h}\{ff_h\} \qquad h_{GT} = h \text{ tal que } ff_h = ff^{min}$$

  De las operaciones que se realizan en h_{GT} se toman aquellas que pueden comenzar antes de ff^{min}. Esas serán las operaciones elegibles:

$$E^* = \{ij \,/\, ij \in E_{GT} \text{ y } r_{ij} \leq ff^{min}\}$$

Al final del actual apartado aparece una tabla con el significado de la notación utilizada.

Una vez explicados los dos métodos, se procederá a describir las reglas de lanzamiento. Hay que tener en cuenta que existen un sinfín de ellas. A continuación se destacan las consideradas más representativas:

- FIFO (*First In First Out*): según esta regla se seleccionará aquella operación que antes haya llegado a la máquina. Esta operación es

la de menor r_{ij} (fecha de disponibilidad de la operación *(i)* de la orden *(j)*). Tiende a reducir el $F_{máx}$ (tiempo de flujo máximo, siendo el tiempo de flujo el tiempo del pedido en el taller).

- SPT (*Shortest Processing Time*): se trata de seleccionar la operación con menor tiempo de proceso. Secuencia primero los trabajos con menor duración, es decir, la operación con menor d_{ij} (duración de la operación *(i)* de la orden *(j)*). Esta regla tiende a minimizar el tiempo medio de permanencia en el taller de las órdenes (*AFT*: tiempo medio en el taller) y, por lo tanto, el stock medio de piezas en curso (*WIP*: *work in process*). En el caso de un solo centro de trabajo el resultado obtenido con esta regla es óptimo.

- LPT (*Longest Processing Time*): similar a la regla anterior, pero seleccionando la tarea con mayor tiempo de proceso, es decir, la operación con mayor d_{ij}.

- WSPT (*Weighted Shortest Processing Time*): es una variación de la SPT, aplicable en el caso de que cada orden tenga asociada un peso. La tarea con mayor prioridad será aquella con menor (d_{ij}/w_j), siendo (w_j) el peso de la orden (j).

- SRPT (*Shortest Remaining Process Time*): en esta regla se trata de seleccionar la operación con el tiempo de trabajo o proceso restante más largo (la operación con mayor RT_{ij}).

$$\mathrm{RTij} = \sum_{k=i+1}^{N_i} d_{kj}$$

- EDD (*Earliest Due Date*): esta regla da prioridad a los trabajos más urgentes (con menor fecha comprometida). Esta regla selecciona la operación que pertenezca al pedido con menor fecha de entrega D_j (*(i,j)* tal que D_j sea mínimo). De esta forma se pretende reducir los retrasos medios (*AT*). Cuando el problema se reduce a un solo centro de trabajo, la regla minimiza el retraso máximo.

- LNRO (*Largest Number of Remaining Operations*): esta regla consiste en seleccionar la operación *(i,j)* perteneciente al pedido con mayor número de operaciones pendientes (ro_{ij}).

$$ro_{ij} = N_j - i + 1$$

- WINQ (*Work In Next Queue*): usando esta regla tendrá prioridad la operación (*i,j*) – *operación i de la orden j* - con menor cantidad de trabajo en cola de la máquina donde se realiza siguiente operación (*i,j*+1) de la misma orden. Por trabajo en cola se entiende a la suma de los tiempos de las operaciones pendientes en una determinada máquina. Si (*i,j*) es la última operación de la orden, el trabajo en cola de la siguiente máquina se considera cero.

- MS (*Minimun Slack*): esta regla trata de seleccionar la operación con menor margen. El margen (*Slack*: s_{ij}) se define como el tiempo restante hasta la fecha comprometida, menos el trabajo restante, incluyendo la operación actual. Busca reducir los retrasos medios (*AT*).

  Margen o *slack* $s_{ij} = D_j - f_{h_{ij}} - d_{ij} - RT_{ij}$

- CR (*Critical Ratio*): regla que da preferencia a las tareas con menor razón crítica. Por razón crítica se entiende al cociente entre el tiempo disponible para realizar la orden (fecha de entrega comprometida del pedido *(j)* menos fecha actual: $D_j - f_{h_{ij}}$) y el plazo estimado para finalizar la orden ($TP_{ij} = RT_{ij} + d_{ij} + esperas$).

  Tal y como indica (Heizer & Render, 1993), un ratio critico bajo, por debajo de 1, indica que el trabajo va con retraso; un ratio crítico igual a 1 indica que justo el trabajo va según lo previsto; y un ratio crítico por encima de 1 indica que el trabajo va adelantado, existiendo, por tanto, un cierto margen o *slack*.

- SJT (*Shortest Job processing Time*): esta regla selecciona la operación de la orden con menor tiempo de proceso (el tiempo de proceso de la orden es la suma de todos los tiempos de procesos de sus correspondientes operaciones).

- COVERT (*Cost OVER Time*): en esta regla, la operación con mayor prioridad será la que mayor ratio (c_j / d_{ij}) tenga, siendo (c_j) un coste dinámico que depende del *slack* (s_{ij}) y del tiempo de espera estimado entre las operaciones que restan para finalizar la orden (esp_{ij}). Este coste será 1 si ($s_{ij} \leq 0$), ($esp_{ij} - s_{ij})/TP_{ij}$ si ($0 < s_{ij} \leq esp_{ij}$) y 0 si ($s_{ij} < esp_{ij}$).

- RANDOM: esta regla permite seleccionar de forma aleatoria la operación que se incorporará a la solución. Existen muchas variantes de este método. Se pueden asignar a las operaciones diferentes probabilidades de ser elegidas; también se puede disminuir el conjunto de operaciones candidatas mediante algún criterio heurístico como los anteriores. Estos métodos no suelen aportar buenas soluciones, pero en ocasiones son útiles para generar diferentes puntos de partida sobre los que aplicar los denominados métodos de mejora.

Las reglas de lanzamiento suelen clasificarse según diversos criterios. Así por ejemplo existen las reglas que pueden estar basadas en tiempo de proceso y las relacionadas con la fecha comprometida. Entre las primeras podríamos mencionar la SPT o la LPT; y entre las segundas la EDD, la MS o la CR.

Otro criterio de clasificación puede ser el de diferenciar reglas estáticas frente a dinámicas. En las reglas estáticas como SPT, LPT o EDD, el valor de la función heurística que establece la prioridad entre las operaciones no depende del momento del programa en el que se evalúe, sólo dependerá de la operación. En las dinámicas (MS, CR, WINQ o COVERT) ocurre lo contrario: la función heurística aplicada sobre una operación devuelve diferentes valores, dependiendo de la iteración en el que se evalúe.

Otro criterio bastante utilizado es el que hace referencia al ámbito de la información que se utiliza en la evaluación de la función heurística. Así nos podemos encontrar heurísticas locales versus globales. La regla WINQ, en la que es necesario tener en cuenta la cola de trabajo pendiente en todas las máquinas, sería del tipo global. Frente a ésta, otras como EDD, LPT o CR, las cuales sólo usan información de la máquina donde se va a realizar la operación a evaluar y de la orden a la que pertenece, se incluirían entre las locales.

En la siguiente tabla se muestran los términos usados tanto en los métodos explicados, como en las anteriores reglas de lanzamiento, para verlo todo de una forma más clara y visual:

r_{ij}	Fecha de disponibilidad de la operación i de la orden j
$F_{máx}$	Tiempo de flujo máximo
d_{ij}	Duración de la operación i de la orden j
AFT	Tiempo medio en el taller
WIP	Stock medio de piezas en curso
w_j	Peso de la orden j
RT_{ij}	Tiempo de trabajo o proceso restante de una operación
D_j	Fecha de entrega de la orden o pedido j
AT	Retrasos medios
TP_{ij}	Plazo estimado para finalizar la orden
ro_{ij}	Número de operaciones pendientes
$f_{h_{ij}}$	Fecha actual
c_j	Coste dinámico
s_{ij}	Slack
esp_{ij}	Tiempo de espera estimado entre las operaciones que restan para finalizar la orden

Tabla 1. Notación usada.

2.3.1.2 Algoritmos de inserción

A diferencia de las reglas de lanzamiento o *dispatching rules* descritas en el apartado anterior, en los algoritmos de inserción la adición y secuenciación de nuevas operaciones a la solución no se va a realizar tras las ya programadas. En los algoritmos de inserción lo que se hace es intercalar las nuevas operaciones entre las ya programadas.

Este tipo de métodos suele aportar mejores resultados que los algoritmos basados en reglas de lanzamiento, pero como desventaja precisan de mayores necesidades de cálculo.

2.3.1.3 Programación hacia delante y hacia atrás

Estos métodos simplifican de manera significativa el problema *job shop* al eliminar las restricciones de capacidad. Si se utiliza la programación hacia adelante se programa en primer lugar la primera operación de cada orden, a continuación la segunda, y así sucesivamente hasta terminar con la orden. Si por el contrario se emplea la programación hacia atrás, se comienza programando por la última operación de cada orden, de forma que termine justo antes del plazo.

Este tipo de programación es muy sencilla, tanto que se suele asignar a cada operación más tiempo del necesario para hacer a este método más realista, y se utiliza de forma frecuente en entornos *MRP* para calcular la capacidad necesaria para realizar la producción planificada. Generalmente, mientras que se recomienda el procedimiento de programación hacia adelante cuando se intenta dejar un margen de tiempo antes de la fecha de entrega para permitir absorber posibles retrasos, la programación hacia atrás se usa cuando se desea disminuir la cantidad de material o trabajo en curso (*WIP*).

2.3.1.4 Métodos basados en cuellos de botella

Tal y como se ha comentado con anterioridad, en los sistemas de producción se pueden dar los denominados cuellos de botella cuando una máquina procesa los ítems de manera más lenta que el resto de la cadena. Es muy importante localizarlos pues va a determinar la cantidad de piezas posibles a fabricar después de un determinado periodo de tiempo.

Estos métodos de programación, se basan en dichos cuellos de botella, estableciendo unas prioridades de producción que determinaran la

actividad del resto de las máquinas del sistema. La idea, se puede decir que es, por tanto, optimizar los recursos más críticos para, de esta forma, optimizar el rendimiento global del sistema.

Estos métodos, también conocidos por su término es inglés *Shifting Bottleneck Procedure (SBP)*, tratan de superar el inconveniente que supone que el cuello de botella no siempre corresponde a una sola máquina. Constan de los siguientes pasos o fases:

- Identificación de un subproblema.
- Selección del cuello de botella.
- Resolución del subproblema.
- Reoptimización del programa de producción o *scheduling*.

Cada vez que se programa una máquina previamente identificada, el resto de máquinas programadas con anterioridad son reoptimizadas resolviendo un problema de programación de una única máquina. Las máquinas se reoptimizan hasta que no se encuentre ninguna mejora. El algoritmo finaliza cuando todas y cada una de las máquinas han sido programadas.

2.3.2. Métodos de mejora

Los métodos heurísticos de mejora también son conocidos por algunos autores como métodos de búsqueda por entornos o como métodos metaheurísticos. La mayor parte de los métodos propuestos y desarrollados en las últimas décadas se incluyen en este tipo de métodos heurísticos. Se han ido desarrollando en estos últimos años para obtener mejores resultados que los obtenidos por los métodos heurísticos tradicionales.

El sufijo "meta" significa "más allá", "a un nivel superior". Las metaheurísticas son estrategias para diseñar o mejorar los procedimientos heurísticos en aras a conseguir un alto rendimiento. Se aplican generalmente en aquellos problemas en los cuales no existe ningún algoritmo específico satisfactorio o heurístico para su resolución; o cuando no es práctico utilizar tal método en ejecución.

Estos métodos se basan en, a partir de una solución inicial, buscar una solución más compleja y elaborada mediante pequeñas modificaciones. De este modo se van reordenando la secuencia de actividades que se van a

programar en cada máquina. A esto es a lo que se llama “movimiento básico”. Es necesario también, para entender mejor estos métodos, definir el término de “entorno”. El entorno será el conjunto de soluciones que se pueden obtener a partir de todos los movimientos básicos posibles.

La forma de seleccionar una solución perteneciente al entorno, a partir de la cual se continúa con la búsqueda, diferencia los posibles métodos existentes. A primera vista lo lógico podría parecer escoger la que fuera la mejor solución posible, de manera que si ésta no mejorara la solución de partida se finalizaría la búsqueda. Esta forma de proceder es conocida como “mejora iterativa”. Sin embargo, este procedimiento podría encerrar la búsqueda en óptimos locales, muy lejanos al óptimo global del problema, lo que llevaría a soluciones erróneas.

Para evitar esto, se han desarrollado diferentes sistemas de búsqueda que evitan el bloqueo en óptimos locales introduciendo una cierta aleatoriedad en la selección de soluciones intermedias,

Entre los métodos de mejora se distinguen dos tipos: los métodos metaheúristicos basados en trayectorias y los métodos metaheurísticos basados en poblaciones.

2.3.2.1 Metaheurísticas basados en trayectorias

Las metaheurísticas basadas en trayectorias parten de una solución inicial. Poco a poco se van explorando nuevas alternativas, proceso conocido por algunos autores como “exploración del vecindario”, de forma que la búsqueda finaliza cuando se alcanza un número máximo de iteraciones, se encuentra una solución con una calidad aceptable o se detecta un estancamiento del proceso.

Destacan las siguientes:

- Grasp

El acrónimo de este sistema de búsqueda proviene de las siglas en inglés de *Greedy Randomized Adaptive Search Procedures*. Este método fue introducido en 1995 por Feo y Resende. Este sistema hace referencia a una serie de métodos de búsqueda de soluciones que conjugan construcción

aleatoria de soluciones y mejoras de dichas soluciones. Es un método iterativo donde cada paso consiste en una fase de construcción y otra de mejora. Este algoritmo engloba los siguientes pasos:

- Construcción de una solución inicial.
- Aplicación de las modificaciones básicas aleatorias para construir nuevas soluciones.
- En el momento en que se encuentre una solución de mejora, se toma como mejor solución y se vuelve al paso anterior.

Los algoritmos de búsqueda GRASP resultan muy sencillos de aplicar a los problemas de *job shop*.

- Recocido simulado

Este sistema de búsqueda se inspira en la metalurgia y el tratamiento térmico que le da su nombre y fue indtroducido por Cemy y Kirkpatrick para la optimización de problemas combinatorios con mínimos locales. El recocido es una técnica en la que se tras un calentamiento del material, éste se somete a un lento enfriamiento. Las moléculas, de este modo, van adoptando poco a poco una configuración de mínima energía. Se obtiene, así, un sólido plástico, dúctil y tenaz.

En el recocido simulado los estados del sólido son la metáfora necesaria para compararlos con las posibles soluciones del problema. De esta manera, se podría decir que la búsqueda comenzara con “altas temperaturas” dónde un alto número de soluciones serán aún viables, y a medida que “la temperatura” disminuye solo se contemplaran pequeñas mejoras de la solución. Este rango de “temperaturas” y su control permitirán establecer las condiciones de búsqueda en cada paso del proceso.

- Búsqueda tabú

Este método fue presentado en 1986 por Glover. Para comenzar a explicar este método de búsqueda hay que hacer hincapié en que el método anterior, el de recocido simulado, no memoriza las soluciones que ya han sido estudiadas por lo que existe el peligro de caer en un bucle.

La búsqueda tabú soluciona este problema creando una lista en la que se almacenan todos los movimientos realizados y que, por tanto, no podrán

repetirse de nuevo. De este modo se restringe la búsqueda y se evitan los mínimos locales. Hay que destacar que, sin embargo, al igual que en el método anterior, es necesario partir de una solución inicial.

2.3.2.2 Metaheurísticas basadas en poblaciones

Las metaheurísticas basadas en poblaciones son métodos que van construyendo un conjunto de individuos que representarán al conjunto de soluciones. De forma general se puede decir que el procedimiento consiste en generar, seleccionar, combinar y reemplazar dicho conjunto. Su eficiencia y su resultado van a depender, fundamentalmente, de la forma con la que se manipula la población en cada iteración.

Destacan las siguientes:

- Algoritmos genéticos

Este método de búsqueda tiene su origen en la teoría de la evolución de Darwin. De este modo, introduciendo en los algoritmos de optimización procesos análogos a la transmisión genética, la reproducción sexual y la mutación, se puede conseguir soluciones bastante satisfactorias.

En este método, por tanto, las posibles secuencias o programas de producción se relacionan a los individuos que forman parte de una población. Cada individuo está caracterizado por su *fitness* (valor que se utiliza como criterio para el proceso de selección). El proceso es iterativo y a cada iteración se la denomina generación. La población de cada generación está formada por los individuos que sobreviven de la anterior más lo hijos de esa nueva generación. En cada generación, mediante los métodos de reproducción y de mutación, se obtienen individuos con cada vez mayor valor de *fitness*.

En este método no se parte de una solución inicial, a diferencia de en los anteriores. Aquí se comienza con un conjunto de soluciones que representan a la población inicial de la especie de una manera metafórica.

- Algoritmos basados en colonias de hormigas

En este método de búsqueda la metáfora se realiza al tomar como inspiración los mecanismos que regulan las sociedades de insectos, en especial la de las hormigas. Este tipo de algoritmos se conocen como algoritmos ACO (*Ant Colony Optimization*).

La idea de que las hormigas liberan feromonas las cuales sirven de guía al resto de hormigas de su colonia, permite diseñar metaheurísticas capaces de generar soluciones de buena calidad para problemas como el *job shop.*

De una manera breve se puede decir que el rastro de las feromonas se va reforzando cada vez que más hormigas pasan. Las hormigas tienden a seguir el rastro de feromona con una cierta probabilidad. Cuando una hormiga se desvía de ese rastro inicial, pero encuentra un camino más corto al hormiguero, este camino se verá reforzado haciendo que el primero, debido a la volatilidad de las feromonas vaya perdiendo "transeúntes". Esto permite que los caminos más eficientes sean los más transitados, y por tanto se refuerce su uso, mientras que los menos eficientes son abandonados.

Con este método se consiguen, progresivamente, soluciones que cada vez se acercan más al valor óptimo.

- Optimización por cúmulos de partículas

Este método es más conocido por su término inglés, *particle swarm optimization* (PSO). Fue desarrollado en 1995 por Kennedy y Eberhart.

Se basa en la metáfora de relacionarlo con los movimientos de los bancos de peces, bandadas de pájaros o enjambres de insectos. El movimiento de estos sistemas se sabe que está condicionado por un líder, aunque el movimiento individual dependa de la experiencia propia.

2.4. Robustez

Una vez presentados los procedimientos de resolución tanto exactos como heurísticos conviene destacar la importancia del concepto de robustez.

El concepto de robustez de una solución o programa se refiere a su capacidad para hacer frente a posibles perturbaciones aleatorias que ocurran en el tiempo de ejecución, permaneciendo estable y consiguiendo el mínimo

deterioro de la función objetivo. Por tanto, los métodos de resolución de problemas de programación de operaciones estudiados, no sólo deben proporcionar una solución factible, sino también robusta con la finalidad de hacer frente a las variaciones externas que puedan afectar a su rendimiento.

En la práctica, debido a las difíciles condiciones en las que se suele trabajar, normalmente inciertas y cambiantes, los planes desarrollados raramente se corresponden con la realidad y es necesario modificarlos a menudo a causa de sucesos aleatorios como por ejemplo, el que una máquina se averíe o el que haya que introducir un nuevo trabajo sin previo aviso.

De este modo, se puede decir, que cuanto más robusta es la secuenciación de operaciones, más sencillo es reprogramarla, por lo que resulta conveniente incorporar reglas de robustez en el propio desarrollo del método de resolución. Existen numerosas reglas a aplicar para conseguir programas robustos. A continuación se explican algunas de las más usadas:

- Insertar tiempos de máquina parada

Esta regla sugiere insertar periodos de paro de máquina en ciertos instantes de tiempo, de forma que las máquinas queden programadas por debajo de su capacidad.

La duración de estos tiempos ociosos introducidos en el programa dependerá de la naturaleza de las perturbaciones que puedan ocurrir. Sin embargo, la probabilidad de ocurrencia de estos sucesos inesperados suele ser mayor al principio de los programas. Debido a esto, la duración de los tiempos ociosos introducidos debe ir disminuyendo según se va avanzando en el programa.

- Programar los trabajos menos flexibles primero

Esta regla se basa en la convicción de que los trabajos menos flexibles deben tener mayor prioridad que los más flexibles. Si ocurriese un suceso inesperado, entonces los trabajos pendientes de procesar corresponderían a los más flexibles. La flexibilidad de un trabajo puede estar determinada tanto por el número de máquinas que pueden realizarlo, como por los tiempos de preparación que requiere, entre otras características.

- No posponer innecesariamente el procesamiento de ninguna operación

Desde el punto de vista de costes de inventario es deseable empezar las operaciones lo más tarde posible. Además, hay que tener en cuenta que se puede caer en penalizaciones por adelantos. Sin embargo, desde un punto de vista de robustez, lo deseable es empezar las operaciones tan pronto como sea posible. Por lo tanto, es necesario, en cada caso, buscar el equilibrio entre robustez y costes por inventario decidiendo qué es lo que más interesa mediante la ayuda de ponderaciones.

- Mantener siempre algún trabajo en la cola de las máquinas más utilizadas

Esta regla tiene el propósito de asegurar que una máquina que sea cuello de botella nunca deje de estar alimentada a pesar de la posible existencia de un suceso aleatorio que ocurra en alguna etapa anterior. De este modo, si no se mantiene un inventario frente a la máquina cuello de botella y la máquina que alimenta el cuello de botella de repente se avería, entonces el cuello de botella debe permanecer ocioso y puede no ser capaz de recuperar el tiempo perdido más adelante.

2.5. Conclusiones

La planificación de la producción es un proceso de toma de decisiones habitual en el día a día de la industria. Se trata de uno de los procesos de decisión más importantes en gestión de la producción y aporta enormes ventajas. La secuenciación de operaciones pertenece al nivel de planificación a corto plazo, y es un problema de optimización combinatoria para el que se han desarrollado numerosos métodos a partir de técnicas matemáticas y procedimientos heurísticos, con el objetivo de definir el orden o secuencia óptima en que se ejecutan las operaciones asignadas a los recursos productivos.

Las técnicas de resolución desarrolladas se pueden dividir en dos categorías: los métodos exactos, que proporcionan una solución óptima, pero pueden requerir un tiempo de computación muy alto, y los métodos

heurísticos, que proporcionan una solución razonablemente buena en un tiempo aceptable.

En la siguiente figura se puede ver la clasificación que se ha llevado a cabo de los diferentes métodos existentes, los exactos por un lado, y los heurísticos por otro.

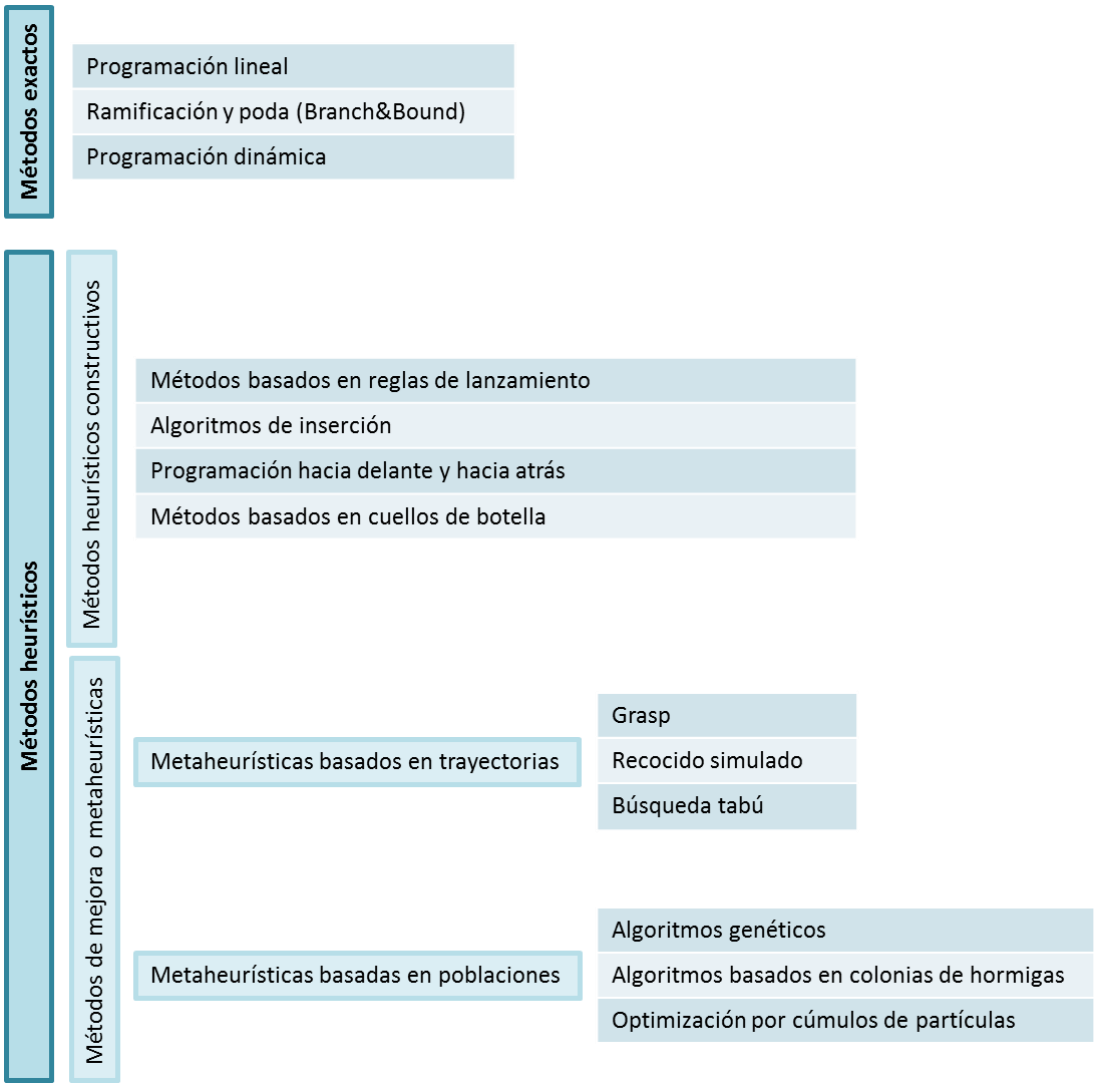


Figura 5. Esquema de los principales métodos de resolución.

La complejidad los problemas de producción plantea dificultades en su resolución. En primer lugar, el gran número de soluciones (programas) que ofrece la combinatoria hace inviable una exploración exhaustiva de las mismas. Por otro lado, al ser NP-hard hay serias sospechas de que no existen métodos para resolverlos de forma óptima en tiempo polinómico. Finalmente, otra dificultad añadida es la gran variedad de tipos de problemas que se presentan, siendo necesaria, en muchas ocasiones, la creación de nuevos procedimientos adaptados a la particularidad de cada caso.

Estos hechos han motivado la investigación y desarrollo de métodos útiles para la secuenciación en el FJSSP.

En la siguiente tabla se realiza una comparación entre los métodos disponibles referidos en los apartados anteriores, valorando los pros y contras de cada uno de ellos. Como se aprecia a continuación, en ella se resumen las principales ventajas e inconvenientes de los diferentes tipos de métodos existentes.

Métodos	Ventajas	Inconvenientes
Exactos	• Solución óptima • Exploración exhaustiva de las soluciones • Precisos	• Tiempo computacional elevado • No adecuados para problemas NP-hard • Limitados a Job-Shop reducidos • Dificultad de implementación
Heurísticos constructivos	• Tiempo computacional razonable • Adecuados en problemas NP-hard • Proporcionan varias soluciones • Sencillez de implementación	• No siempre garantizan el óptimo • Exploración de soluciones no exhaustiva
Metaheurísticos	• Tiempo computacional razonable • Adecuados para problemas NP-hard • Proporcionan varias soluciones • Sencillez de implementación • Mejoran los resultados heurísticos	• No siempre garantizan el óptimo • Exploración de soluciones no exhaustiva

Tabla 2. Comparativa de los distintos métodos de resolución.

En este capítulo además, se ha destacado la importancia de las reglas de robustez como una poderosa ayuda a la optimización bajo incertidumbre.

Dentro de los métodos heurísticos, que son los más utilizados por todas las razones que se han dado con anterioridad, este trabajo se va a centrar en los métodos heurísticos constructivos.

Por eso, como se verá en el siguiente capítulo, se probarán diferentes reglas de lanzamiento utilizando como base tanto el algoritmo de Giffler y Thompson, como el algoritmo de lanzamiento. Para realizar este programa en el que se verá la mejor solución a la secuenciación de operaciones, se utilizará Java como lenguaje de programación. En el siguiente capítulo se hará una descripción exhaustiva del programa implementado.

CAPÍTULO 3. DESCRIPCIÓN DEL SOFTWARE

3.1. Introducción

En este capítulo se llevará a cabo una descripción exhaustiva del programa implementado. Como ya se ha mencionado, el lenguaje de programación elegido ha sido *Java*. Esta elección fue debida a que se creyó que *Java* reunía todas las condiciones para realizar un programa de estas características. Asimismo, como entorno de programación, se decidió usar el programa informático *Eclipse*.

El objetivo del programa desarrollado es comparar diferente métodos heurísticos de resolución de problemas tipo *job shop* flexibles. Los algoritmos implementados se compararán mediante la técnica conocida como *benchmarking*.

Se ha decidido implementar dentro de los métodos heurísticos existentes, los constructivos. De este modo se van a tener dos modos de resolución, el método de lanzamiento y el método de Giffler y Thompson. A partir de ellos se aplicarán diversas reglas de lanzamiento, que ya fueron explicadas con detalle en el apartado 2.3.1.

Lo que se quiere discernir, es qué combinación de método y regla de lanzamiento proporcionan el mínimo valor de *makespan*. Minimizar el *makespan* significa conseguir la fecha de finalización de la última operación de la orden (y por consiguiente de todo la orden) más temprana.

Por lo tanto, el presente capítulo se estructura como sigue: primero se presentarán las ventajas que ofrece el lenguaje de programación Java, a continuación se estudiará un modelo o diagrama de clases para explicar el programa implementado y, después se describirá la colección de métodos usados en la clase del programa llamada *Scheduler.*

3.2. Ventajas del lenguaje de programación *Java*

Mediante un lenguaje de programación se representan cálculos y expresiones a realizar por un ordenador. Todo lenguaje de programación debe estar formado por un conjunto de palabras reservadas, símbolos y reglas sintéticas y semánticas que definen su estructura y el significado de sus elementos y expresiones. El proceso de programación consiste en la escritura, compilación y verificación del código fuente de un programa. (Lenguaje de Programación en Java, Tecolapa, 2013).

Java es un lenguaje de programación, diseñado como una mejora de C++, y desarrollado por *Sun Microsystems* (compañía actualmente absorbida por *Oracle*).

A continuación se indican sus principales ventajas por las que se ha elegido *Java* como lenguaje de programación para implementar el programa deseado en este trabajo:

- Es un lenguaje multiplataforma. Esto significa que los programas desarrollados podrán ejecutarse sin problemas en cualquier sistema operativo (Windows, Linux, Mac...).
- Permite crear programas ejecutables en dispositivos de muy distinta naturaleza.
- Es uno de los lenguajes más demandados por empresas. El conocimiento sobre tecnología *Java* está en alto crecimiento en el mercado.
- Es un lenguaje orientado a objetos, con las ventajas que eso supone a la hora de diseñar y mantener los programas.
- Soporta el manejo de *threads* para crear programas multitarea.
- Permite excepciones, como alternativa más sencilla para manejar errores, como ficheros inexistentes o situaciones inesperadas.
- Es más fiable y seguro que C++ al no existir los punteros.
- Cada vez va incorporando más facilidades para la creación de entornos basados en ventanas para la creación y manipulación de gráficos, para el acceso a bases de datos, etc.
- El sistema de *Java* tiene ciertas políticas que evitan que se puedan codificar virus con este lenguaje. Destaca, por tanto, por su seguridad.

3.3. Modelo de clases

A continuación se observa un modelo o diagrama de clases que resume la lógica seguida en el programa implementado (Tutorial de UML, 2015). Este diagrama, realizado en UML (de las siglas en inglés *Unified Modeling Language*), permitirá visualizar las relaciones entre las clases que configuran el programa desarrollado mediante el lenguaje de programación *Java.*

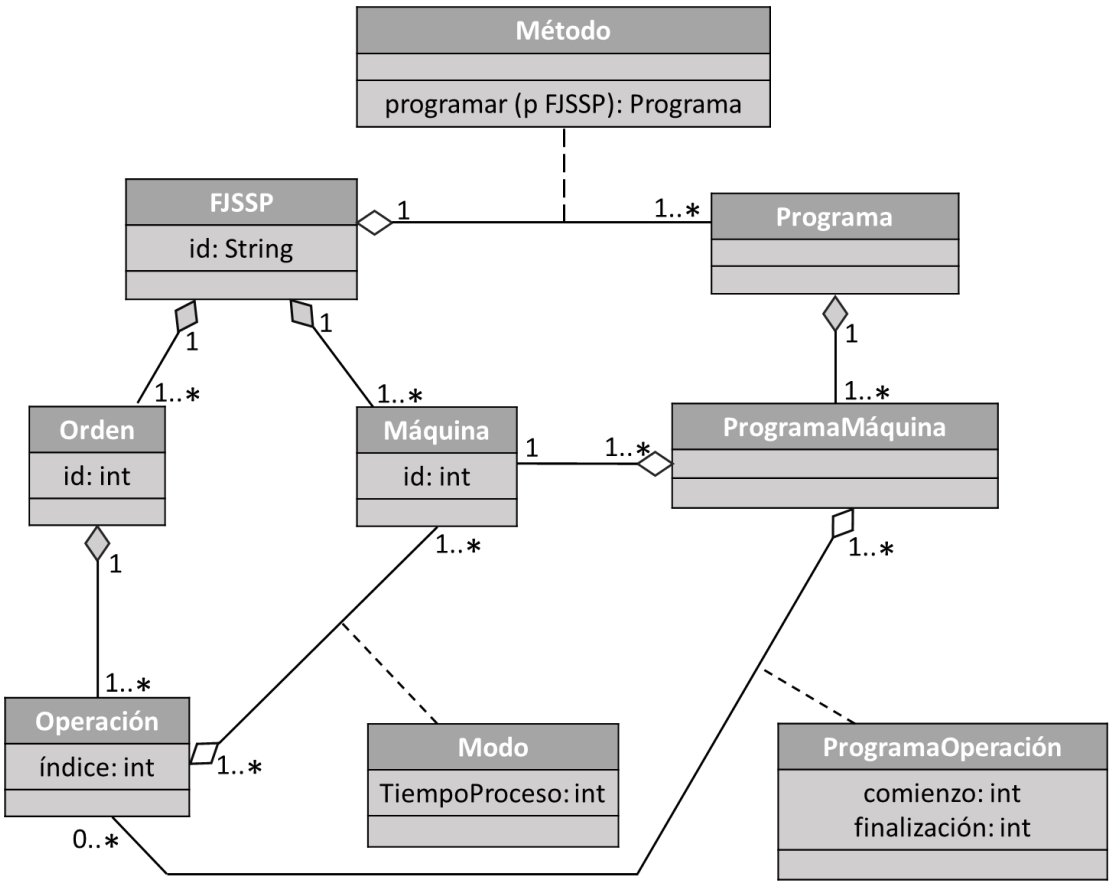


Figura 6. Diagrama de clases UML.

Para entender mejor el diagrama anterior es necesario destacar la siguiente notación usada:

FJSSP: Flexible Job Shop Scheduling Problem

id: identificador. Este elemento permite identificar el objeto.

índice: es la posición de la operación en la orden. Indica el orden de ejecución. Define la ruta.

_____ : esta línea discontinua sirve para representar relaciones cualificadas. Por ejemplo, la clase modo representa una relación entre una operación y una máquina.

Conviene, además, explicar todo lo concerniente al lenguaje UML aplicado a este diagrama. Un diagrama de clases se va a descomponer en clases y relaciones.

Una clase es la unidad básica que engloba toda la información correspondiente a un objeto o instancia de una clase. Cada clase, como se puede observar en la figura anterior, se representa por un rectángulo con tres divisiones. En la primera división aparece el nombre de la clase; en la segunda, los atributos o variables de instancia que caracterizan a esa clase; y por último, en la tercera, se muestran los métodos u operaciones que indican como interactúa el objeto con el entorno.

Una vez vista la representación de las clases, hay que definir el significado de las distintas relaciones usadas entre ellas. La cardinalidad de las relaciones, que indican el grado y el nivel de dependencia, pueden ser:

→ 1: un número fijo. En este caso es uno.

→ 1..*: uno o muchos (1..n).

→ 0..*: cero o muchos (0..n).

Finalmente es necesario destacar la diferencia entre los dos tipos de rombo:

Este símbolo indica que los objetos han sido compuestos por valor. Normalmente denominada por composición, este tipo de relación es estática, donde el tiempo de vida del objeto incluido está condicionado por el tiempo de vida del que lo incluye.

Este otro símbolo indica que los objetos han sido compuestos por referencia. A diferencia de la relación anterior, ésta es dinámica. Esto quiere decir que el tiempo de vida del objeto incluido es independiente

del que lo incluye. Comúnmente esta relación recibe el nombre de agregación.

Tanto en un caso como en el otro, los rombos van colocados en el objeto que posee las referencias.

Antes de proceder a la explicación del diagrama de clases se puede observar en el siguiente esquema una visión global de la estructura de datos:

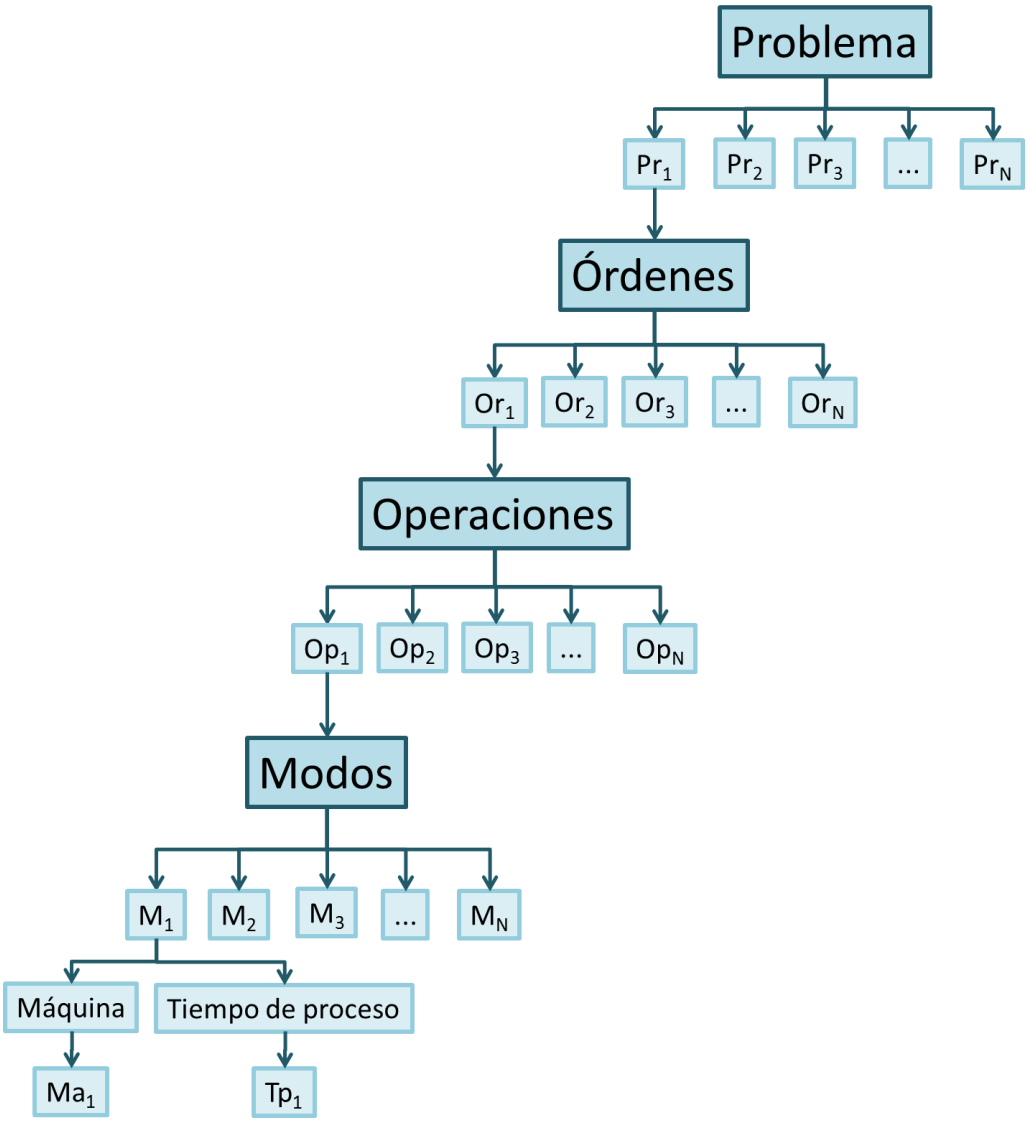


Figura 7. Esquema básico del modelo de datos.

En esta figura se puede apreciar de una manera muy clara como todo problema está compuesto por una serie de órdenes y cada orden posee una serie de operaciones. Hasta aquí se podría comparar con un problema *job shop* clásico. Sin embargo, al quererse implementar un programa que resuelva problemas tipo *job shop* flexible, hay que hablar de modos. Cada operación se puede realizar en una o varias máquinas, de ahí que una operación pueda poseer varios modos. Cada modo va a hacer, por tanto, referencia a una máquina y a un tiempo de proceso característico.

Esto mismo se puede observar en el diagrama de clases de la figura 6. La primera clase que aparece recibe el nombre de *Método* y tiene como operación *programar.* El programa va a hacer referencia al problema *job shop* flexible (FJSSP). Existe una dependencia de esta clase con otras dos: *FJSSP* y *Programa*. Esta dependencia viene indicada por una línea punteada que significa que la clase *Método* representa una relacion entre un problema *job shop* flexible y un programa. Esta relacion recibe el nombre de cualificada.

La clase *FJSSP* está relacionada con la clase *Programa* de una manera dinámica. Un problema de tipo job shop flexible va a poseer uno o muchos programas. La relación que presentan, denominada comunmente como agregación, indica que si se destruyese el objeto *FJSSP*, los objetos *Programa* asociados no se verían afectados.

Esta clase *FJSSP* tiene como atributo que permite identificarla un *string*. A su vez, esta clase está interrelacionada con otras dos: *Orden* y *Máquina*. La composición, tanto con *Orden* como con *Máquina,* es por valor e indica que si se destruyera el objeto *FJSSP,* los objetos *Orden* y *Máquina* existentes serían a su vez destruidos. Un problema de tipo *job shop* flexible puede poseer una o muchas ordenes y máquinas.

La clase *Orden* tiene como atributo un entero (int) que va a permitir identificarla. De nuevo, esta clase está compuesta por otra denominada *Operación*. Al igual que antes, si el objeto *Orden* fuera destruido, los objetos *Operación* tambien lo serían. Una orden posee una o muchas operaciones.

Existe otra clase, la clase *Modo*, que al igual que la clase *Método*, depende de otras dos: *Operación* y *Máquina*. Un modo representa una relación entre una máquina y una operación. Ya se ha explicado anteriormente que la existencia de varios modos es la esencia del problema *job shop* flexible. Con el otro esquema se ha visto claramente que cada operación se puede realizar en una o varias máquinas, pudiendo poseer, así, cada operación varios modos. La clase *Modo* va a tener como atributo el tiempo de proceso que vendrá definido como un entero.

Volviendo a la clase *Máquina*, cabe destacar que como atributo va a tener un entero que permitirá identificarla. Esta clase está relacionada con otras dos: *Operación* y *ProgramaMáquina*. En ambos casos la composición va a ser por referencia (relación también conocida como agregación). Al igual que ocurría en la relación de los objetos *Programa* y *FJSSP*, el destruir los objetos *Operación* o *ProgramaMáquina*, no va a suponer la destrucción del objeto *Máquina*. Una máquina puede poseer uno o muchos *programas máquina*, pero varias máquinas pueden poseer a su vez una o muchas operaciones. Esto último es así porque una misma operación se va a poder realizar, al tratarse del problema tipo *job shop* flexible, en varias máquinas a la vez.

La clase *Operación* va a tener como atributo lo que se ha denominado índice o posición de la operación de la orden. Este entero va a definir la ruta e indicar el orden de ejecución de la operación. Presenta relaciones con otras tres clases, con *Orden* y *Máquina*, que ya se han explicado, y con *ProgramaMáquina*. Esta composición va a ser también por referencia, presentando las mismas características indicadas anteriormente. Un problema máquina puede tener cero o varias operaciones.

En la clase *ProgramaMáquina* no se va a hablar de atributos, pues no posee identificadores. Esta clase se interrelaciona con otras tres: *Operación*, *Máquina* y *Programa*. Las dos primeras relaciones ya han sido explicadas. La composición con *Programa* es por valor, y por tanto, si se destruyera el objeto *Programa* los objetos *ProgramaMáquina* existentes serían a su vez destruidos. Cerrando por tanto el ciclo, se puede decir que un programa puede poseer uno o muchos *programas máquina.*

Y, por último, queda por definir la clase *ProgramaOperación*. Esta clase tiene dos atributos definidos como enteros: tiempo de comienzo y tiempo de finalización. Existe una dependencia de esta clase con otras dos: *Operación* y *ProgramaMáquina*. Esta dependencia viene indicada por una línea punteada que significa que la clase *ProgramaOperación* representa una relacion entre una operación y un *programa máquina*.

Finalmente, y para cerrar este apartado, conviene explicar el cometido de las clases *Programa*, *ProgramaMáquina* y *ProgramaOperación* debido a que no aparecían en el esquema general de la figura 7.

En la siguiente figura se pude observar un gráfico de Gant en el aparecen todos estos conceptos:

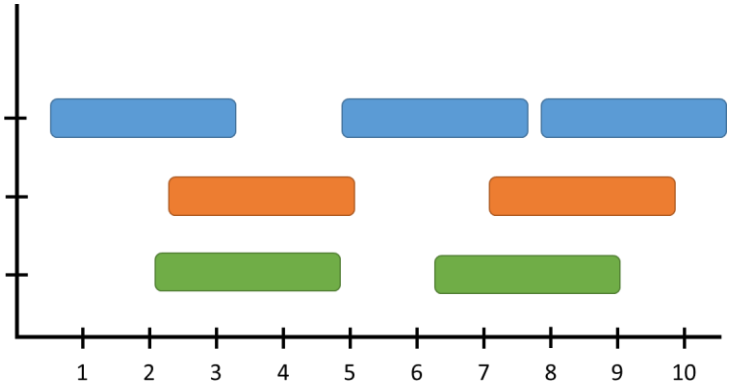


Figura 8. Programa, ProgramaMáquina y ProgramaOperación.

El conjunto de rectángulos hace referencia al Programa. Este Programa, por ejemplo, estaría compueto por tres ProgramaMáquina, cada uno de ellos representados en el diagrama de Gantt por el conjunto de rectángulos del mismo color. Cada uno de ellos a su vez, va a estar formado por varios ProgramaOperación (representados, por tanto, por cada uno de los rectángulos).

A modo de resumen, se puede decir de este modo, que cada programa representa el programa de las operaciones de cada una de las máquinas.

3.4. Descripción del *scheduler*: colección de métodos

Una vez explicado el programa en su totalidad, en este apartado se realizará un estudio de la clase *Scheduler*, describiendo todos los métodos que la componen. Antes de proceder a detallar la función de cada uno de ellos, conviene especificar cada paquete de clases usado en el programa.

En la siguiente figura se pueden observar los tres paquetes definidos para implementar el programa: el *scheduling*, el *scheduling.data* y el *scheduling.metodos*.

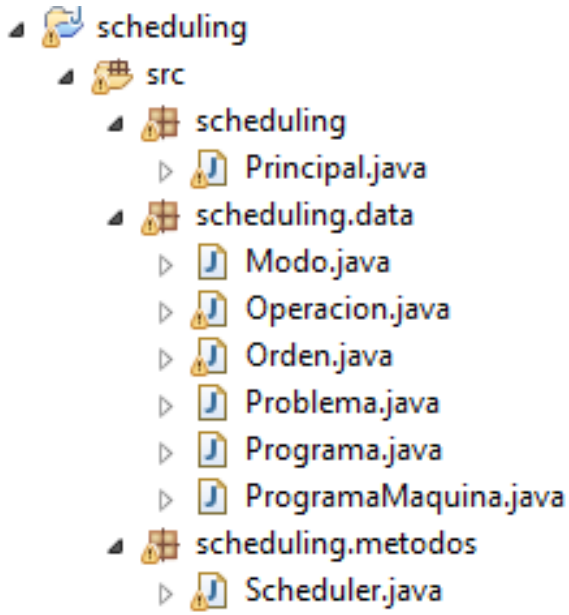


Figura 9. Paquetes del programa.

El paquete *scheduling* es la raíz y va a albergar a la clase denominada como *Principal* y desde la cual se leen los datos de los distintos problemas.

El paquete *scheduling.data* representa el diagrama de clases. Como se puede observar en la figura 8, contiene las siguientes clases: *Modo*, *Operación*, *Orden*, *Problema*, *Programa* y *ProgramaMaquina*.

Por último, el paquete *scheduling.metodos* contiene a la clase *Scheduler* y de lo que se va a encargar es de relacionar el programa con el problema.

En el anexo 2 se podrá observar, de una manera detallada, todo el código fuente correspondiente al programa.

A continuación se describen los distintos métodos que contiene la clase *Scheduler*:

3.4.1. Método para crear el programa

Este primer método de tipo programa tiene como argumento una variable de tipo "problema". Este es el método principal.

Lo primero que se debe hacer es crear una estructura para ir almacenando los datos de la forma representada en la siguiente figura:

Máquina 1	[Op11, Op21]
Máquina 2	[Op31, Op41, Op51]
Máquina 3	[Op61]
⋮	⋮

Figura 10. Ejemplo de estructura de las operaciones programables.

En el ejemplo de la figura se puede observar cómo, para una orden o trabajo, en este caso la 1, se van introduciendo en la máquina correspondiente las operaciones programables. En la máquina 1 se realizarían las operaciones 1 y 2; en la máquina 2, la 3, la 4 y la 5; y en la máquina 3, por último, la operación 6.

En Java, el lenguaje de programación usado, cada *array* de operaciones programables recibe el nombre de *HashSet*, sin embargo, para cada máquina, ese conjunto de operaciones programables recibe el nombre de *ArrayList*.

Se ha usado en este caso el *HashSet* por las ventajas que ofrece frente al *ArrayList*. Al quererse eliminar las operaciones programadas del conjunto de programables, estas operaciones son mucho más fáciles de encontrar en un *HashSet* ya que no supone el almacenamiento de cada elemento en una posición concreta. Se trata pues, de un conjunto no ordenado. En un ArrayList habría que recorrerlo posición por posición para encontrar la operación a eliminar.

Una vez se tiene definida la anterior estructura ya se puede crear el programa, al que es necesario aportar como variable el número de máquinas. Lo primero que se hace es, para cada orden, ver sus operaciones. Se selecciona su primera operación y se ven los modos en los que se puede

realizar dicha operación. Como se ha dicho con anterioridad, cada modo va a estar definido por una máquina y un tiempo de proceso determinado. Por lo tanto, se ve la máquina correspondiente a cada modo y se introduce la operación en el *HashSet* de dicha máquina. De esta manera ya se tiene la primera operación programable.

El siguiente paso es seleccionar las operaciones elegibles dentro de las programables. Ahora, por tanto, lo que se le pasa al programa, es el conjunto de operaciones elegibles. Para elegir estas operaciones existen dos métodos diferentes, tal y como se dijo en el apartado 2.3.1.1., el método de lanzamiento y el método de Giffler y Thompson. En los dos siguientes apartados se explicará cómo se han programado ambos métodos.

Una vez aportadas estas operaciones elegibles es necesario pasarle la operación a programar. Esta operación a programar también se elegirá según diferentes reglas de lanzamiento que han sido implementadas. Asimismo, los métodos usados para programar estas reglas se explicarán en el apartado 3.4.4.

Para finalizar este método, una vez se tiene la operación a programar, ésta se programa. Tras programarla, se debe eliminar de las operaciones programables. Se busca la operación sucesora a la programada y se añade al *HashSet* de operaciones programables correspondiente según el modo, y por tanto la máquina, en la que se deba realizar.

3.4.2. Método de lanzamiento

La finalidad de este método es obtener las operaciones elegibles. Para entenderlo bien, es necesario explicar que se basa en seleccionar las operaciones que antes pueden comenzar dentro del conjunto de operaciones programables.

Va a tener como argumentos tres variables: por un lado, una de tipo problema y una de tipo programa y, por otro, el *HashSet* de operaciones programables.

El primer paso de este método es crear un *ArrayList* donde se almacenarán las operaciones elegibles. Después se recorre el *HashSet* de operaciones programables teniendo en cuenta en qué máquina se han de realizar. Se obtiene, para cada una de esas operaciones, su índice, con el fin

de encontrar la fecha más temprana de disponibilidad de la operación. Se van a tener dos posibles opciones:

- Si el índice de la operación es mayor que cero. En este caso se debe buscar la orden de la operación para encontrar la precedente. La fecha más temprana de disponibilidad de la operación será el mínimo entre la fecha de disponibilidad de dicha operación y la fecha de finalización de la operación precedente.
- Si el índice de la operación es cero. En este caso, la fecha más temprana de disponibilidad de la operación va a ser cero.

Es necesario, además, tener en cuenta la fecha más temprana en la que la máquina puede empezar a procesar la operación. Esta fecha será el máximo entre la fecha más temprana en la que la máquina se queda disponible y la fecha más temprana de disponibilidad de la operación.

Tras definir una variable que almacene la fecha más temprana de comienzo de una operación, el siguiente paso es comprobar si la fecha más temprana de disponibilidad de la máquina es menor. En ese caso, se seleccionaría y esa sería la nueva fecha de comienzo de la operación.

Después, es necesario recorrer el *HashSet* de operaciones programables de la máquina seleccionada. Igual que se ha hecho con anterioridad, si el índice de la operación fuera mayor que cero, se buscaría la orden a la que pertenece para poder obtener la operación precedente. En este caso también tendríamos dos opciones:

- Si la fecha de finalización de la operación precedente es menor o igual que la fecha más temprana de comienzo, se añadiría la operación estudiada al conjunto de elegibles y se programaría con la fecha obtenida como el máximo entre la fecha de finalización de la operación precedente y la fecha de disponibilidad de la máquina correspondiente.
- Si la fecha de finalización de la operación precedente es mayor que la fecha más temprana de comienzo, directamente se añadiría al conjunto de operaciones elegibles y se programaría con la fecha obtenida como el máximo entre la fecha de disponibilidad de la máquina correspondiente y cero.

De esta manera ya se consigue el *ArrayList* de operaciones elegibles.

3.4.3. Método de Giffler y Thompson

Lo primero a explicar de este método es que selecciona, dentro del conjunto de operaciones programables, aquellas pertenecientes a la máquina que antes pueda terminar y, dentro de estas, las que pueden empezar antes de esa fecha de finalización.

Este es el segundo método programado para seleccionar las operaciones elegibles. Como se verá en el siguiente capítulo, se llevará a cabo un estudio en el que se comparará este método con el anterior para comprobar cuál de los dos es más eficaz.

Al igual que el anterior este método va a tener las mismas tres variables como argumentos: por un lado, una de tipo problema y una de tipo programa y, por otro, el *HashSet* de operaciones programables.

El primer paso de este método es crear un *ArrayList* donde se almacenarán las operaciones elegibles. Después se recorre el *HashSet* de operaciones programables teniendo en cuenta en qué máquina se han de realizar. Se obtiene, para cada una de esas operaciones, su índice, con el fin de encontrar la fecha más temprana de disponibilidad de la operación. Se van a tener dos posibles opciones:

- Si el índice de la operación es mayor que cero. En este caso se debe buscar la orden de la operación para encontrar la precedente y obtener su fecha de finalización. La fecha más temprana de disponibilidad de la operación será el mínimo entre la fecha de disponibilidad de dicha operación y la fecha de finalización de la operación precedente.
- Si el índice de la operación es cero. En este caso, la fecha más temprana de disponibilidad de la operación va a ser cero.

Después, es necesario recorrer el HashSet de operaciones programables de la máquina seleccionada. Igual que se ha hecho con anterioridad, si el índice de la operación fuera mayor que cero, se buscaría la orden a la que pertenece para poder obtener la operación precedente. En este caso también tendríamos dos opciones:

- Si la fecha de finalización de la operación precedente es menor que la fecha más temprana de finalización, se añadiría la operación estudiada al conjunto de elegibles y se programaría con la fecha obtenida como el máximo entre la fecha de finalización de la operación

precedente y la fecha de disponibilidad de la máquina correspondiente.

- Si la fecha de finalización de la operación precedente es mayor que la fecha más temprana de comienzo, directamente se añadiría al conjunto de operaciones elegibles y se programaría con la fecha obtenida como el máximo entre la fecha de disponibilidad de la máquina correspondiente y cero.

De esta manera ya se consigue el *ArrayList* de operaciones elegibles.

3.4.4. Reglas de lanzamiento

Los métodos que se explican en este apartado corresponden a las reglas de lanzamiento elegidas para comparar entre sí. Aplicando estas reglas se selecciona una operación del conjunto de elegibles calculado, bien con el método de lanzamiento, o bien con el método de Giffler y Thompson. A continuación se describe cada uno de ellos:

❖ SPT (*Shortest Processing Time*)

Esta regla selecciona la operación del conjunto de elegibles con el menor tiempo de proceso. Este método de tipo operación tiene como argumento cuatro variables: una de tipo "problema", otra de tipo "programa" y por último el *ArrayList* de operaciones elegibles y el de operaciones programables.
Para cada una de las operaciones elegibles, se obtienen sus modos. De esta manera, para cada modo de la operación se ve su tiempo de proceso. Se elige la operación de cada orden con el menor tiempo de proceso, teniendo en cuenta el modo en el que ese tiempo es menor.

❖ LPT (*Longest Processing Time*)

Esta regla es exactamente igual que la anterior exceptuando que selecciona la operación del conjunto de elegibles con el mayor tiempo de proceso. Al igual que el método anterior, este es de tipo operación y

tiene como argumento tres variables: una de tipo “problema”, otra de tipo “programa” y por último el *ArrayList* de operaciones elegibles y el de operaciones programables.
Para cada una de las operaciones elegibles, se obtienen sus modos. De esta manera, para cada modo de la operación se ve su tiempo de proceso. Se elige la operación de cada orden con el mayor tiempo de proceso, teniendo en cuenta el modo en el que ese tiempo es mayor.

- SRPT (*Shortest Remaining Process Time*)

  Esta regla selecciona la operación del conjunto de elegibles con el tiempo de trabajo o proceso restante más largo. Este método es de tipo operación tiene como argumento cuatro variables: una de tipo “problema”, otra de tipo “programa” y por último el *ArrayList* de operaciones elegibles y el *ArrayList* de operaciones programables.
  Tal y como se ha presentado en el capítulo 2, no se incluye el tiempo de la presente operación.
  Por lo tanto, este método recorre las operaciones elegibles y selecciona la que pertenezca a la orden con menor tiempo de proceso de sus operaciones restantes.

- SJT (*Shortest job Processing Time*)

  Esta regla selecciona la operación de la orden con menor tiempo de proceso (el tiempo de proceso de la orden es la suma de todos los tiempos de procesos de sus correspondientes operaciones). Al igual que los anteriores, este método es de tipo operación y tiene como argumento las mismas cuatro variables.
  Este método recorrerá el *array* de operaciones elegibles y seleccionará aquella que pertenezca a la orden con un tiempo de proceso total más pequeño.

- LNROandSPT (*Largest Number of Remaining Operations and Shortest Processing Time*)

  Se ha decidido combinar la regla LNRO con la SPT debido a los buenos resultados obtenidos de experimentos realizados con esta regla.
  La regla de lanzamiento LNRO selecciona la operación perteneciente a la orden con mayor número de operaciones pendientes. Se decidió combinar esta regla debido a la seguridad de que podrían existir

empates. En el caso de que haya dos operaciones pertenecientes a una orden con el mismo número de operaciones pendientes se seleccionará la de menor tiempo de proceso. Es ahí donde aparece la regla SPT.
Como una operación se puede realizar en varias máquinas hay que seleccionar la maquina con menor cola. Para encontrarla, es necesario recorrer todos los modos.
Este método implementado es de tipo operación, teniendo las mismas cuatro variables de argumento que los anteriores.

- WINQandSPT (*Work In Next Queue and Shortest Processing Time*)

Por los mismos motivos que en la anterior regla de lanzamiento, se decidió combinar la WINQ con la SPT, de la que tan buenos resultados se habían obtenido en los experimentos realizados.
La regla de lanzamiento WINQ es la más compleja de todas las implementadas y selecciona la operación con menor cantidad de trabajo en cola de la máquina donde se realiza siguiente operación de la misma orden. Por trabajo en cola se entiende a la suma de los tiempos de las operaciones pendientes en una determinada máquina. Si la operación presente es la última operación de la orden, el trabajo en cola de la siguiente máquina se considera cero. Se usará, por tanto, la regla de lanzamiento SPT como criterio de desempate, eligiendo la operación con un menor tiempo de proceso.
Este método implementado, al igual que se ha dicho en todos los anteriores, es de tipo operación tiene como argumento cuatro variables: una de tipo “problema”, otra de tipo “programa” y por último el ArrayList de operaciones elegibles y el de operaciones programables.

En el anexo 2 se podrá observar el código fuente desarrollado para implementar cada uno de los métodos correspondientes a estas reglas de lanzamiento.

3.5. Conclusiones

En este capítulo se ha visto de una manera muy completa cuál ha sido la lógica seguida en el programa implementado. Gracias a un diagrama o modelo de clases se ha podido comprobar las relaciones existentes entre cada una de las clases. Posteriormente, se ha visto una descripción de cada método desarrollado en la clase *Scheduler.*

En el siguiente capítulo se mostrarán los resultados obtenidos de aplicar las distintas reglas de lanzamiento desarrolladas, tanto para el método de lanzamiento, como para el de Giffler y Thompson.

CAPÍTULO 4. RESULTADOS Y EXPERIMENTOS COMPUTACIONALES

4.1. Introducción

En este cuarto capítulo se verá una presentación detallada de los resultados obtenidos al implementar las diferentes reglas de lanzamiento a los dos métodos programados: el método de lanzamiento y el método de Giffler y Thompson. Para poder realizar este análisis se han aplicado trescientos problemas diferentes utilizando la técnica conocida como *benchmarking*. Se denomina *benchmarking* al proceso sistemático y continuo para evaluar comparativamente los resultados obtenidos al resolver los problemas.

La secuenciación de tareas o actividades con limitación de recursos es un problema clásico de optimización combinatoria que ha sido ampliamente estudiado a lo largo de todos estos años. Como ya se ha dicho con anterioridad, el problema consiste en determinar el programa compatible de menor duración del conjunto de operaciones que lo compone. Estas operaciones, de duraciones conocidas, van a presentar una serie de restricciones que habrá que tener en cuenta (las restricciones de este tipo de problemas ya fueron explicadas en el apartado 1.5.2.).

Por tanto, el objetivo de este estudio radica en observar que combinación de método y regla de lanzamiento minimiza el *makespan* o fecha de finalización de la última operación del problema. De esta manera, la función objetivo que se tendrá presente será la de minimizar el *makespan*.

La aplicación de este tipo de métodos heurísticos ofrece soluciones aceptables, tanto más cuanto mayor sea el número de aspectos que combina la regla de lanzamiento y el método programado. Sin embargo, y aunque no exista evidencia de que haya una única regla que supere a todas las demás en cualquier tipo de problema, en este apartado y gracias al programa implementado, se intentará discernir la mejor combinación para el problema objeto de estudio, el *job shop* flexible.

Para realizar la presentación de resultados y experimentos computacionales de una forma sencilla y visual, se compararán todas las

combinaciones posibles de método y regla de lanzamiento, llegando así a concluir, de la manera adecuada, qué combinación reduce el *makespan*. Antes, sin embargo se plantean los resultados obtenidos en experimentos previos realizados.

4.2. Experimentos previos

Antes de proceder a la implementación de las seis reglas de lanzamiento, se probaron la SPT y la LPT en los mismos trescientos problemas.

Al aplicar ambas reglas al método de lanzamiento los resultados fueron los siguientes:

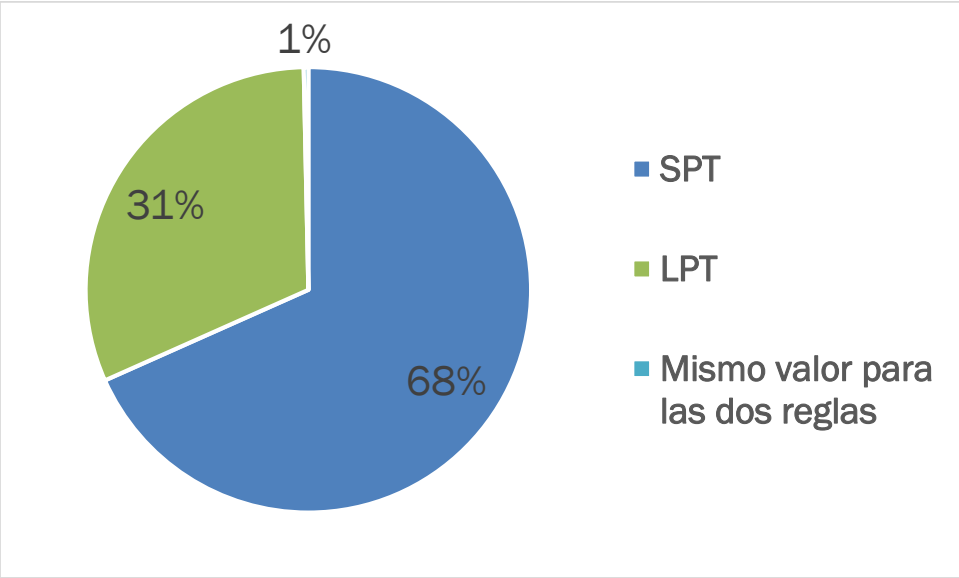


Figura 11. Comparación resultado reglas de lanzamiento SPT y LPT en el método de lanzamiento.

De los 300 problemas en los que se han aplicados estas reglas, en 205 el valor del makespan con la regla de lanzamiento SPT era menor; y en 94, era menor con la regla de lanzamiento LPT. En uno de ellos, el valor obtenido por ambas reglas es el mismo.

Tras la evidencia de los resultados anteriores, se pudo concluir que en comparación con la regla LPT, la regla SPT consigue mejores resultados teniendo como objetivo la minimización del *makespan*.

Ahora se presentas los resultados que se obtuvieron al aplicar el método de Giffler y Thomson con las mismas reglas de lanzamiento:

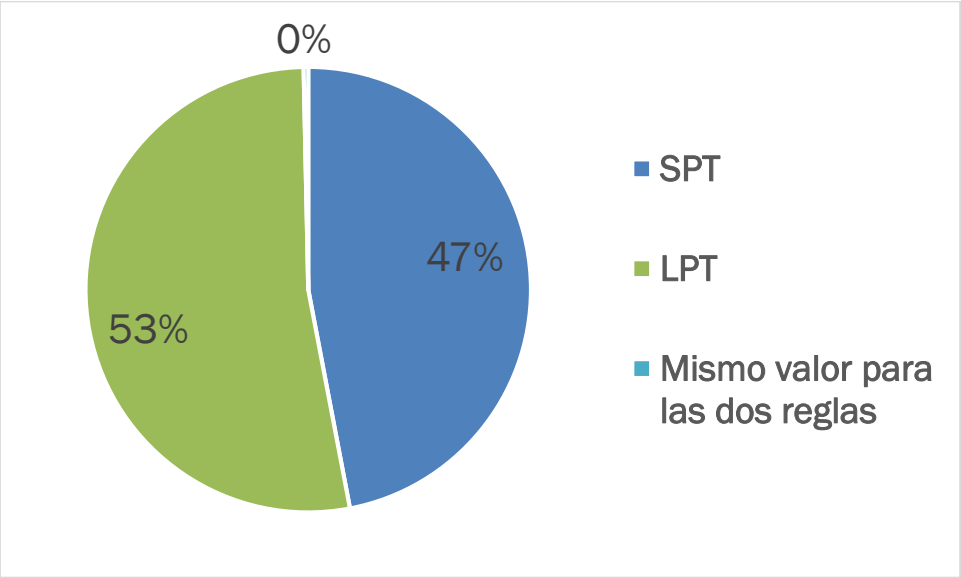


Figura 12. Comparación resultado reglas de lanzamiento SPT y LPT en el método de Giffler y Thompson.

No existe una evidencia tan clara como ocurría en el método de lanzamiento de que aplicar una regla u otra suponga minimizar el *makespan*. Los porcentajes están más equilibrados. Aun así, los resultados obtenidos de aplicar la regla de lanzamiento LPT son mejores con un 53 % del total. Por lo tanto, resumiendo, al contrario de lo ocurrido en el método de lanzamiento, se obtienen valores más pequeños del *makespan* al aplicar la regla de lanzamiento LPT.

Sin embargo, ante el éxito rotundo de la combinación de la regla de lanzamiento SPT y el método de lanzamiento, se decidió, como se ha indicado en el capítulo anterior, usar esta regla como criterio de desempate para las reglas de lanzamiento LNRO y WINQ.

En el siguiente apartado se realiza un estudio exhaustivo de los resultados finales obtenidos.

4.3. Resultados obtenidos

En el siguiente gráfico se pueden observar los resultados obtenidos de aplicar las distintas reglas de lanzamiento tanto al método de lanzamiento como al método de Giffler y Thompson para cada uno de los trescientos problemas estudiados. En el anexo 1 se pueden consultar los valores específicos obtenidos para cada problema.

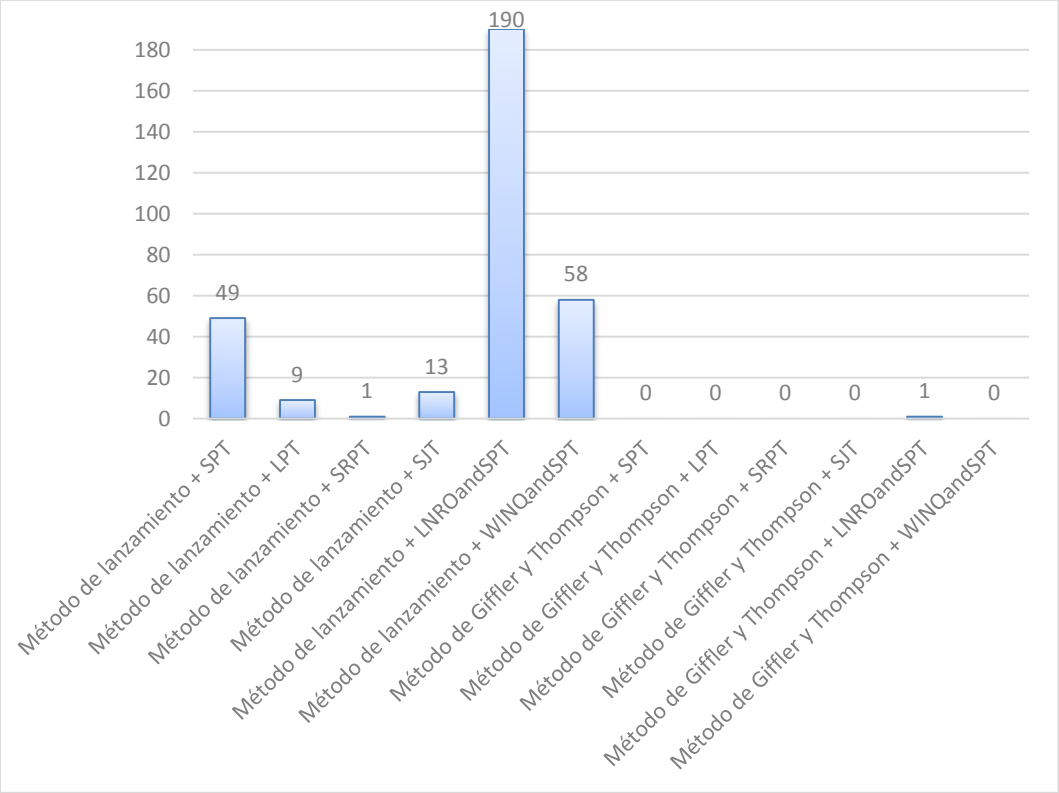


Figura 13. Resultados obtenidos.

	Mejor resultado	Frecuencia
Método de lanzamiento + SPT	49	0,163333333
Método de lanzamiento + LPT	9	0,03
Método de lanzamiento + SRPT	1	0,003333333
Método de lanzamiento + SJT	13	0,043333333
Método de lanzamiento + LNROandSPT	190	0,633333333
Método de lanzamiento + WINQandSPT	58	0,193333333
Método de Giffler y Thompson + SPT	0	0
Método de Giffler y Thompson + LPT	0	0
Método de Giffler y Thompson + SRPT	0	0
Método de Giffler y Thompson + SJT	0	0
Método de Giffler y Thompson + LNROandSPT	1	0,003333333
Método de Giffler y Thompson + WINQandSPT	0	0

Tabla 3. Correspondencia mejor resultado obtenido.

Tras las evidencias anteriores se pude asegurar que el método de lanzamiento ofrece un mejor resultado frente al método de Giffler y Thompson. Esto es así, debido a que, como se explicó en el capítulo 2, el método de lanzamiento garantiza la obtención de programas sin esperas.

Antes de evaluar todas las reglas de lanzamiento implementadas, se realizaron experimentos con las reglas SPT y LPT. Tras los buenos resultados obtenidos al aplicar la primera de ellas, se decidió combinar tanto la LNRO como la WINQ, con dicha regla de lanzamiento como criterio de desempate. El éxito obtenido ha sido rotundo.

Para el método de lanzamiento, en 49 de los 300 problemas la regla SPT ha logrado minimizar el *makespan*. Sin embargo, en 190 de esos 300 problemas ha sido la combinación de LNRO y SPT la que ha conseguido obtener un mejor valor. Seguidamente, la combinación de WINQ y SPT ha sido la que ha alcanzado un valor mínimo de *makespan* en 58 de los 300 problemas.

Después de estos resultados más prometedores, estaría la combinación de método de lanzamiento y SJT con la que se ha obtenido el mínimo en 13 de los 300 problemas. Por último, hay que señalar la combinación del método de lanzamiento con la regla SRPT y del método de

Giffler y Thompson con LNROandSPT, con la obtención del mejor resultado en una ocasión, tanto en uno como en otro.

Cabe destacar, a modo aclaratorio, que la suma de los valores anteriores no es de 300 debido a que ha habido veintiuna ocasiones en las que se han obtenido el mismo valor al aplicar dos reglas de lanzamiento distintas.

En la tabla anterior los valores de la frecuencia indican las veces que cada combinación de método y regla de lanzamiento ha obtenido el mínimo en los trescientos problemas estudiados.

4.4. Conclusiones

Como criterio de evaluación de las distintas combinaciones de métodos y reglas de lanzamiento, se decidió tomar como función objetivo minimizar el *makespan* o el tiempo de finalización de la última operación del problema.

Tras realizar experimentos previos a la implementación de las definitivas seis reglas de lanzamiento utilizadas, con la SPT (*Shortest Processing Time*) y la LPT (*Longest Processing Time*), se vieron los maginificos resultados que se obtenían con la primera de ellas. Es por eso, que se decidió combinarla con la LNRO y la WINQ, usándola como criterio de desempate.

Tras evaluar los trescientos problemas a estudiar mediante la técnica conocida como *benchmarking*, se han obtenido los mejores resultados al combinar el método de lanzamiento con la regla de lanzamiento LNROandSPT.

El éxito rotundo del método de lanzamiento frente al de Giffler y Thompson se debe a que no introduce esperas en el programa.

En el capítulo 6 se podrán ver con más detalle las conclusiones del estudio llevado a cabo en este trabajo fin de grado.

CAPÍTULO 5. ESTUDIO ECONÓMICO

5.1. Introducción

El estudio económico es una parte fundamental de cualquier proyecto técnico debido a que determina la viabilidad económica de llevarlo a cabo en la práctica. En este capítulo se hace repaso del coste económico que habría tenido la realización de cada una de las diferentes etapas del proyecto, en el caso de que hubiesen sido contratadas a un equipo profesional cualificado.

A continuación, en el siguiente apartado, se describen las distintas etapas o fases que se consideran en los cálculos realizados para hallar los costes de este TFG. Posteriormente, se explicará cómo se han calculado los diferentes costes (de personal, de amortización, de material) y los gastos generales. Por último se indicarán los costes generales y el precio de venta del presente TFG.

5.2. Etapas de desarrollo del trabajo

En este apartado se hace un desglose del tiempo invertido en la realización del presente trabajo, teniendo en cuenta las distintas tareas que lo componen. El tiempo total invertido en el presente trabajo se ha estimado en cinco meses.

Por tanto, es necesario mostrar a continuación las etapas del TFG desde el punto de vista del análisis económico:

- Definición del trabajo

  En esta primera etapa se realizó el primer contacto con la programación de talleres y con el problema a tratar en este trabajo, el *job shop* flexible (FJSSP). Para ello fue necesario realizar una

búsqueda exhaustiva y un estudio de la literatura existente. Es en este momento en el que se decidió que hay campo de trabajo suficiente para realizar un trabajo fin de grado interesante y enriquecedor. Dentro de esta área de investigación, se recopiló información suficiente sobre las distintas heurísticas aplicables al problema tipo *job shop*, y se decidió aplicarlas al problema tipo *job shop* flexible para poder comparar su eficiencia. Una vez se dispuso qué métodos heurísticos comparar (en este caso se llegó a la conclusión de que lo mejor era centrarse, dentro de los métodos heurísticos constructivos, en los basados en las distintas reglas de lanzamiento), el siguiente paso fue elegir qué lenguaje de programación era el más apropiado para implementarlas. Se llegó a la conclusión de que *Java* reunía todas las condiciones para realizar un programa de estas características.

- Familiarización con el programa informático *Eclipse* y con el lenguaje de programación *Java*

  En esta segunda etapa se llevó a cabo un profundo estudio tanto de las instrucciones del propio lenguaje, como de las funcionalidades que *Eclipse* permite para facilitar la labor de programación.

- Recopilación de la información necesaria para programar el algoritmo

  Esta tercera etapa, se dedicó a recopilar toda la información necesaria sobre los métodos heurísticos constructivos y las reglas de lanzamiento elegidas, antes de llevar a cabo la programación. De esta manera se conocieron de manera adecuada todas las variables que intervenían y se pensó de qué manera habría de fluir la información entre las distintas partes del programa. Esta etapa permitió tener una idea muy clara de lo que se quería hacer y cómo conseguirlo, lo que ahorró, sin duda, una cantidad de tiempo importante en el desarrollo del problema que cuenta, de por sí, de una gran complejidad.

- Programación del algoritmo

  En esta cuarta etapa, y a partir del modelo pensado en la etapa anterior, se programó el algoritmo añadiendo numerosas características, que en un principio no se visualizaron, y mejorándolo considerablemente. Sin lugar a dudas, esta es la etapa que más

tiempo llevó, necesitando de un esfuerzo importante tanto de análisis de los distintos fallos, como de rigor y atención al detalle.

- Realización de experimentos y análisis de resultados

  Una vez realizado el programa y habiéndose cerciorado de su correcto funcionamiento, se procedió a la realización de numerosos experimentos para la obtención de los resultados necesarios. La posterior comparación de dichos resultados permitió valorar los distintos métodos programados.

- Edición de la documentación

  En esta sexta etapa, se redactó el tomo del presente trabajo fin de grado. En él se condensó toda la información generada en el transcurso del mismo. Se trató en todo momento de conseguir un libro lo más ameno posible, no perdiendo en ningún momento el rigor. Además se intentó otorgarle un marcado espíritu instructivo y de divulgación, teniendo en cuenta que apenas existe información editada en nuestro idioma de esta índole.

5.3. Cálculo de los costes

Para llevar a cabo de una manera correcta este estudio económico, se debe realizar el cálculo de las horas efectivas anuales para poder, después, determinar las tasas por hora del salario. Los días efectivos de trabajo anual deberán ser calculados en función de datos históricos recopilados de años anteriores. De esta manera, ya se podrá conocer el número de horas por año que trabajarían los profesionales involucrados en la realización de este trabajo, y, así, determinar el coste laboral generado en función de las horas utilizadas.

En la siguiente tabla se puede observar el cálculo de las horas efectivas trabajadas a lo largo de los cinco meses del año:

Nº días por año	**150**
- sábados y domingos	- 40
- días efectivos de vacaciones	-10
- días festivos reconocidos	-5
- media de días perdidos por enfermedad	-4
Total estimado de días efectivos	**91**
Total horas (a 8 horas por día)	**728**

Tabla 4. Días efectivos trabajados al año.

5.3.1. Costes de personal

En el presente trabajo vamos a suponer que los costes de personal (también llamados costes salariales) son los correspondientes a los honorarios de un ingeniero que realiza el estudio y de un administrativo que fue contratado para la edición del presente tomo.

Los costes de personal en este trabajo van a ser los más importantes puesto que se trata de un trabajo de investigación en el que los requerimientos son fundamentalmente humanos.

El desarrollo del TFG, por tanto, se supone llevado a cabo por un Ingeniero Industrial en Organización Industrial, con conocimientos de programación en *Java*. Es este ingeniero el que se va a encargar de:

- Definir los objetivos del trabajo.
- Realizar la programación mediante el programa *Eclipse.*
- Llevar a cabo el análisis de los resultados y realizar los experimentos necesarios.
- Llegar a unas conclusiones finales que condensen todo el conocimiento generado.

Además de con este ingeniero, se va a contar, como se ha dicho, con un administrativo encargado de:

- Realizar los distintos informes.
- Editar la documentación correspondiente.

En la siguiente tabla se pueden apreciar las horas empleadas por cada uno de los dos técnicos en cada etapa del presente trabajo:

ETAPAS	INGENIERO	ADMINISTRATIVO
1. Definición del trabajo	132	-
2. Familiarización con el programa informático Eclipse y con el lenguaje de programación Java	55	-
3. Recopilación de la información necesaria para programar el algoritmo	60	-
4. Programación del algoritmo	280	-
5. Realización de experimentos y análisis de resultados	46	-
6. Edición de la documentación	124	128
TOTAL HORAS	**697**	**128**

Tabla 5. Desglose del tiempo empleado en cada etapa (en horas)

Una vez definido este desglose de horas empleadas por etapa del trabajo, ya se puede calcular los salarios de los dos técnicos. El salario neto anual estimado para el ingeniero es de 24.000 € y para el administrativo, de 9.500 €. El salario total y la retribución horaria están reflejados en la siguiente tabla:

REMUNERACIÓN	INGENIERO	ADMINISTRATIVO
Sueldo bruto anual	24.000	9.500
Sueldo bruto 5 meses	9.863	3.904,1
Seguridad social (35%)	3.452,05	1.366,44
TOTAL 5 MESES	**13.315,05**	**5.270,54**
TOTAL COSTE HORARIO	**18,29**	**7,24**

Tabla 6. Salario del equipo de profesionales (en euros).

Se puede observar cómo la retribución horaria ha sido obtenida dividiendo la retribución anual entre las horas de trabajo anuales.

Lo siguiente es asignar a cada etapa o actividad el número de horas dedicadas por cada técnico y, así, calcular el coste por actividad. Además, se puede calcular el porcentaje de cada actividad sobre el total. Todo ello se puede observar en la siguiente tabla:

ETAPAS	COSTE INGENIERO	COSTE ADMINISTRATIVO	COSTE POR ETAPA	% SOBRE EL TOTAL
1. Definición del trabajo	2.414,28	-	2.414,28	17,65%
2. Familiarización con el programa informático y con el lenguaje de programación	1.005,95	-	1.005,95	7,36%
3. Recopilación de la información necesaria para programar el algoritmo	1.097,4	-	1.097,4	8,02%
4. Programación del algoritmo	5.121,2	-	5.121,2	37,45%
5. Realización de experimentos y análisis de resultados	841,34	-	841,34	6,15%
6. Edición de la documentación	2.267,96	926,72	3.194,68	23,36%
COSTES TOTALES	**12.748,13**	**926,72**	**13.674,85**	**100%**

Tabla 7. Coste de las horas efectivas trabajadas por los dos profesionales (en euros). Se incluye total y porcentaje relativo de cada etapa.

5.3.2. Costes de amortización

En este apartado se calcularán las amortizaciones de los equipos utilizados en la elaboración del presente trabajo fin de grado. Para su realización fue necesario el uso tanto de un ordenador personal, como de un conjunto de periféricos y consumibles.

En la siguiente tabla se puede ver el hardware y el software utilizado. Se ha aplicado una amortización lineal de tres años con valor residual nulo.

EQUIPO INFORMÁTICO	COSTE
Ordenador HP Pavilion 15-p203ns Intel Core i7-5500U	899
Windows 8.1 Pro	139
Microsoft Office 2013	119
Adobe Acrobat	0
Licencia Eclipse	0
Licencia Java	0
Impresora HP DESKJET 1050	59
TOTAL	**1.216**
Amortización anual del equipo	**405,33**
Amortización del equipo (5 meses)	**166,58**

Tabla 8. Coste y amortización de los equipos utilizados (en euros).

5.3.3. Costes de material

Tras haber calculado los costes de personal y los costes de amortización, se puede pasar a evaluar los gastos correspondientes a los materiales que han sido necesarios para la realización del presente trabajo. Como se puede observar en la siguiente tabla, las partidas más importantes son las correspondientes a los consumibles de informática

MATERIALES	COSTE
Material de oficina	30
CD´s	20
Encuadernación	10
Consumibles impresora	90
TOTAL	**150**

Tabla 9. Costes de material (en euros)

5.3.4. Gastos generales

Los costes generales, también conocidos como indirecto, vienen representados en la siguiente tabla:

GASTOS GENERALES	COSTE
Uso de oficina	375
Electricidad	55
Internet	35
Teléfono	50
Agua	7
TOTAL	**522**

Tabla 10. Distribución de gastos generales (en euros)

5.4. Costes totales

El coste del presente trabajo fin de grado se obtiene como suma de los costes totales de cada una de las partidas definidas en los apartados anteriores. En la siguiente tabla se refleja todos estos costes y el porcentaje de cada partida sobre el total de los costes:

CONCEPTO	COSTE	% sobre el total
Costes de personal	13.674,85	94,22%
Costes de amortización	166,58	1,15%
Costes de material	150	1,03%
Gastos generales	522	3,6%
COSTE TOTAL	**14.513,43**	**100%**

Tabla 11. Cálculo del coste total del trabajo (en euros).

En la siguiente gráfica se puede observar de una manera más visual cómo, efectivamente, los mayores costes del trabajo son los del personal:

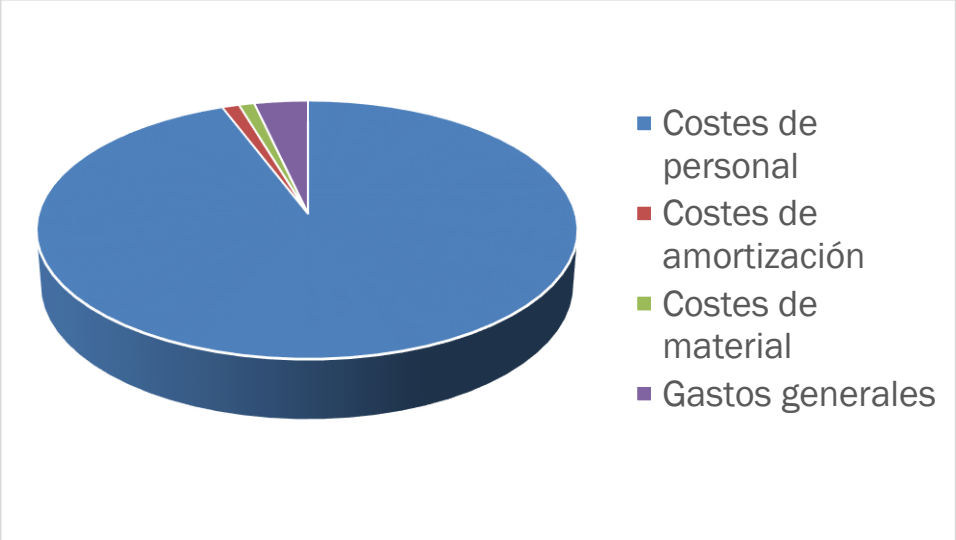


Figura 14. Distribución de los costes del trabajo.

5.5. Cálculo del precio de venta del trabajo

Como se ha visto en el apartado anterior, el importe total de la realización del trabajo es de 14.513,43 €.

Para, finalmente, calcular el precio de venta se ha sumado un beneficio de un 20% sobre el importe del trabajo y se ha sumado un 21% de IVA. En la siguiente tabla se pueden observar todos estos cálculos:

Importe total sin impuestos	14.513,43
Beneficio del 20%	2.902,7
Importe + Beneficio	17.416,12
Impuestos (21%)	3.657,38
PRECIO DE VENTA FINAL	**21.073,5**

Tabla 12. Cálculo precio de venta final (en euros).

CAPÍTULO 6. CONCLUSIONES Y LÍNEAS DE TRABAJO FUTURAS

6.1. Conclusiones

En este trabajo fin de grado se ha demostrado la conveniencia de utilizar algoritmos o métodos heurísticos para resolver problemas complejos de producción como es el *job shop* flexible. A menudo, la dificultad inherente a estos problemas provoca que sean inabordables mediante métodos exactos de resolución y que se deba recurrir a métodos aproximados como son los heurísticos.

Existen una gran cantidad de métodos heurísticos debido a su naturaleza compleja y variada. A menudo estos métodos se desarrollan para un problema en particular y es imposible su aplicación a otros problemas, aunque sean similares.

En concreto, en este trabajo, se ha decidido aplicar los métodos heurísticos constructivos al problema conocido como *job shop* flexible. Con este fin se ha desarrollado un programa para aplicar diversas reglas de lanzamiento a los dos métodos existentes, el método de lanzamiento y el método de Giffler y Thompson, evaluando, así, su eficiencia. Se han elegido los métodos heurísticos constructivos por tratarse de métodos deterministas y estar basados en la mejor solución de cada iteración, al irse construyendo paso a paso la solución del problema.

Para llevar a cabo este estudio y como criterio de evaluación de las distintas combinaciones de métodos y reglas de lanzamiento, se decidió tomar como función objetivo minimizar el *makespan* o el tiempo de finalización de la última operación del problema.

Por lo tanto, como se puede observar, para la realización de este trabajo lo primero y fundamental fue establecer los objetivos que se plantearon en la introducción. A partir de ellos, se ha ido desarrollando todo el estudio sin perder de vista el objetivo principal: comparar lo distintos algoritmos constructivos existentes aplicables al problema de tipo *job shop* flexible mediante *benchmark*ing.

Algunas de las conclusiones principales que se pueden desprender de este trabajo son las siguientes (la mayoría de ellas han sido extraídas del capítulo 4 titulado “Resultados y experimentos computacionales”):

- Los resultados obtenidos con el método de lanzamiento han sido mucho mejores que los obtenidos con el método de Giffler y Thompson. Esto se debe a que el método de lanzamiento evita las esperas en caso de tener las máquinas trabajo disponible.
- La regla SPT (*Shortest Processing Time*) ofrece muy buenos resultados. Es por ello que se decidió usarla como criterio de desempate al combinarla con las reglas de lanzamiento LNRO (*Largest Number of Remaining Operations*) y WINQ (*Work In Next Queue*).
- La combinación representada por el método de lanzamiento y la regla de lanzamiento LNROandSPT ha sido la que mejores resultados ha proporcionado. De los trescientos problemas estudiados, en ciento noventa ha conseguido un *makespan* menor al obtenido con las otras reglas.

6.2. Líneas de trabajo futuras

A la hora de realizar el presente trabajo fin de grado me he centrado en la implementación de dos métodos: el método de lanzamiento y el método de Giffler y Thompson. Estudios futuros podrían desarrollarse a partir del uso de otros métodos heurísticos existentes.

Otra dirección que podrían tomar futuros trabajos podría ser la de implementar otras funciones objetivos diferentes. Este trabajo se ha basado en minimizar el *makespan.* Podrían aplicarse distintas reglas de lanzamiento diseñadas para otras funciones objetivos como puede ser minimizar la fecha de entrega comprometida (en inglés *Due Date*).

)

REFERENCIAS

- Araúzo Araúzo, J. (2007). *Control distribuido de sistemas de fabricación flexibles. Un enfoque basado en agentes.*
- Bautista, J., Pereira, J., de la Rosa, M., & Companys, R. (2001). Resolución del problema RCPS mediante lógica neuro-difusa con aprendizaje evolutivo. *IV Congreso de Ingeniería de Organización: Sevilla, 13-14 de Septiembre de 2001*.
- Companys Pascual, R. (1989). *Planificación y programación de la producción.* Barcelona: Marcombo.
- Cuatrecasas, L. (1996). *Diseño de procesos de producción flexible.* Barcelona: Productivity.
- Cuatrecasas, L. (2009). *Diseño avanzado de procesos y plantas de producción flexibles.* Barcelona: Profit editorial.
- Heizer, J., & Render, B. (1993). *Production and Operations Management. Strategies and Tactics.* Needham Heights: Allyn and Bacon.
- Herrmann, J. (2006). *A history of production scheduling, in Handbook of Production.* New York: Springer.
- Laviós Villahoz, J. J. (2013). *Análisis de la Relajación Lagrangiana como método de programación de talleres flexibles en un entorno multiagente.* Burgos.
- Martí Cunquero, R. (2003). Algoritmos Heurísticos en Optimización Combinatoria.
- Min Hee, K., & Yeong-Dae, K. (1994). Simulation-Based Real-Time Scheduling in a Flexible Manufacturing System. *Journal of Manufacturing Systems*.
- Nhu Binh, H., & Joc Cing, T. (2005). Evolving Dispaching Rules for solving the Flexible Job-Shop Problem.
- Peña, V., & Zumelzu, L. (2006). Estado del Arte del Job Shop Scheduling Problem.

- Pinedo, M. (2009). *Planning and Scheduling in Manufacturing and Services.* New York: Springer.
- Vela, C., González, M., & Varela, R. (2013). An efficient Memetic Algorithm for the Fexible Job Shop with Setup TImes. *Twenty-Third International Conference on Automated Planning and Scheduling*.

REFERENCIAS WEB

- Características de un Proceso Productivo Flow Shop (Producción en Masa).
  Autor: Geo Tutoriales.
  http://www.gestiondeoperaciones.net/procesos/caracteristicas-de-un-proceso-productivo-flow-shop-produccion-en-masa/
  Última visita: julio 2015.

- Configuraciones Productivas. Conceptos y tipologías fundamentales.
  Autor: Santiago Ibarra Mirón.
  http://www.monografias.com/trabajos16/configuraciones-productivas/configuraciones-productivas.shtml
  Última visita: julio 2015.

- Flexible Job Shop Problem.
  Autor: Monaldo Mastrolilli.
  http://people.idsia.ch/~monaldo/fjsp.html
  Última visita: agosto 2015.

- Guía de colecciones en Java.
  Autor: Héctor Luaces Novo.
  Año: 2013.
  http://www.luaces-novo.es/guia-de-colecciones-en-java/
  Última visita: agosto 2015.

- Método Heurístico.
  Autor: Javier Juarez.
  Año: 2013.
  http://es.slideshare.net/profjavierjuarez/metodo-heurstico-1
  Última visita: agosto 2015.

- Sistema de Producción Flow Shop.
  Autor: Bryan Salazar López.
  http://www.ingenieriaindustrialonline.com/herramientas-para-el-ingeniero-industrial/producci%C3%B3n/sistema-de-produccion-flow-shop/
  Última visita: julio 2015.

- Sistemas Productivos.
  Autores: Javier Cruells y Alejandra Difiori.
  http://www.rinconcreativo.com.ar/sistproductivo/Tema%2018_02.htm
  Última visita: julio 2015.

- Clasificación Métodos Heurísticos.
  Autor: Brito Ramírez Izquierdo.
  http://psmheuristica.webnode.com.ve/clasificacion-metodos-heuristicos/.
  Última visita: agosto 2015.

- Tutorial de UML – Modelo de clases
  http://users.dcc.uchile.cl/~psalinas/uml/modelo.html
  Última visita: agosto 2015.

- Lenguaje de Programación en Java.
  Autor: Marco Tecolapa.
  Año: 2013.
  http://www.academica.mx/blogs/lenguaje-programaci%C3%B3n-en-java
  Última visita: agosto 2015.

ANEXO 1. Resultados

En este anexo se detallan todos los valores obtenidos al comparar las reglas de lanzamiento aplicadas a los dos métodos implementados: el método de lanzamiento y el método de Giffler y Thompson.

A continuación, se pueden observar en forma de tabla los resultados descritos en el capítulo 4 del presente trabajo:

	Método de lanzamiento						Método de Giffler y Thompson					
Problema	SPT	LPT	SRPT	SJT	LNRO And SPT	WINQ And SPT	SPT	LPT	SRPT	SJT	LNRO And SPT	WINQ And SPT
1	3006	3060	3417	3521	2770	2899	4231	3677	4778	4667	3301	3286
2	2478	2701	2961	2671	2425	2504	4577	3830	4159	4445	2874	3695
3	3289	2506	3058	2853	2263	2673	4846	4008	5560	4914	2769	4026
4	2952	3152	3530	3501	2770	3100	4207	4280	4398	4654	3254	3604
5	2833	2795	3133	2841	2421	2574	4273	3760	4309	4616	2963	3482
6	2605	2713	3173	2864	2299	2474	4412	4163	5570	5356	2829	4001
7	2982	3108	3487	3529	2752	2897	4824	4197	5213	4673	3196	3263
8	2446	3022	2712	3070	2220	2521	4309	4243	5010	5348	2690	4179
9	2612	2395	2744	2644	2092	2399	4964	4426	6003	5823	2601	4249
10	2954	3233	3536	3464	2658	2925	4548	4427	4963	4805	3204	3477
11	2576	2660	2852	2729	2232	2633	4297	4131	5134	5298	2668	3748
12	2278	2509	2618	2701	2097	2683	5142	4542	5940	5660	2564	4054
13	3083	3339	3239	3253	2669	2849	4813	4162	5256	5105	3113	3746
14	2687	2847	3043	2900	2250	2466	4946	5189	6208	5760	2664	3995
15	2875	2725	2699	2754	2272	2591	6148	5522	6867	6864	2687	4378
16	3307	3520	3357	3179	2558	2710	4834	4184	5273	4734	3149	4258
17	2661	2532	2941	2881	2252	2587	4680	4698	5806	4971	2676	4400
18	2437	3016	2755	2740	2201	2762	5620	5718	7197	6849	2575	4492
19	1352	1555	1735	1624	1371	1420	1886	1935	1922	2145	1466	1760
20	1352	1555	1735	1624	1371	1319	1886	2043	1922	2145	1466	1650

Tabla 13. Resultados.

	Método de lanzamiento						Método de Giffler y Thompson					
Problema	SPT	LPT	SRPT	SJT	LNRO And SPT	WINQ And SPT	SPT	LPT	SRPT	SJT	LNRO And SPT	WINQ And SPT
21	1120	1521	1633	1293	1040	1220	2093	2016	2482	2543	1270	1510
22	1324	1562	1602	1361	1281	1495	1777	2078	2138	2236	1455	1566
23	952	1132	959	951	947	946	1936	2307	2315	2375	1040	1913
24	1097	1207	1162	1124	981	1006	1437	1719	1835	1458	1129	1226
25	946	976	943	875	921	881	1651	1532	1852	1870	1001	1641
26	1085	1207	1239	1088	1067	1159	1760	1355	1503	1716	1177	1308
27	862	755	848	924	809	755	1649	1370	1647	1653	875	1499
28	847	903	1032	1020	765	832	1164	1049	1208	1253	885	851
29	745	675	916	789	601	663	1247	1146	1444	1367	736	913
30	876	818	943	895	734	780	1154	1252	1422	1341	870	937
31	608	582	665	715	520	552	1557	1302	1792	1920	610	1081
32	884	947	1021	924	776	801	1042	1216	1357	1305	866	952
33	722	761	976	760	686	649	1185	1162	1433	1428	761	1127
34	857	873	890	995	751	796	1222	1119	1329	1250	876	1054
35	598	673	751	728	538	566	1370	1455	1885	1834	648	1060
36	917	963	1071	917	829	834	1174	1217	1354	1161	986	1034
37	748	719	816	848	700	670	1152	1138	1329	1501	782	1027
38	891	928	969	907	786	844	1234	1163	1327	1222	950	1033
39	608	613	703	724	538	572	1426	1333	1736	1741	619	1164
40	7798	10336	10344	8848	7798	9208	11159	13663	12963	8848	11159	12297
41	6320	6465	6335	6923	6821	6044	8964	8754	9527	9452	7344	8280
42	7251	8894	9765	8291	7451	8189	11491	11144	12774	10475	11919	13219
43	6985	6424	7918	6815	5355	5408	7725	8040	10939	9449	6276	7681
44	7741	10330	10206	9166	7741	8643	10680	12704	13257	9166	10680	13821
45	6724	7059	7482	6448	7263	7140	9266	9971	12707	10538	7955	8226
46	7329	8767	8817	7909	8325	8815	10723	12414	11051	9213	9932	11358
47	8198	6504	6926	6865	6144	6329	10191	8876	10143	10037	7524	8291
48	8237	11272	9710	9298	8237	9640	11898	14382	11479	9298	11898	13557
49	7992	7186	7580	7189	6706	6433	10229	9961	12994	12077	7978	9627
50	8026	10921	9200	9132	8066	8778	9558	13593	12518	9647	12288	13987
51	7056	6536	7036	7062	6139	6639	10871	9358	11647	12352	6771	9354
52	8423	11840	12009	10684	8423	10072	13079	14627	13340	10684	13079	12741
53	8088	8303	8539	9352	8264	8451	10079	9764	12689	11660	10208	10749
54	8845	11328	9948	8907	8488	8441	12434	13093	11015	10368	12766	11469
55	7383	7602	7232	7182	7573	7700	9320	9555	12578	11673	8048	8525
56	9028	11582	10231	9265	9028	9273	10483	14788	12816	9265	10483	12937
57	7912	7560	7262	7716	6624	7395	10638	11000	10060	10409	8941	10002
58	8175	9698	10314	9148	8108	8520	9682	12832	12848	9931	11411	13420
59	5870	6255	6380	6240	6140	6659	9313	9105	12825	12238	6690	7873

Tabla 14. Resultados (continuación).

	Método de lanzamiento						Método de Giffler y Thompson					
Problema	**SPT**	**LPT**	**SRPT**	**SJT**	**LNRO And SPT**	**WINQ And SPT**	**SPT**	**LPT**	**SRPT**	**SJT**	**LNRO And SPT**	**WINQ And SPT**
60	9718	10858	9982	9815	9718	9426	12893	13307	12350	9815	12893	11447
61	7765	8043	7381	7627	7290	7363	10974	12799	12485	12295	8461	10481
62	8837	9507	9493	10016	8837	9973	12644	13872	11417	9815	12215	10650
63	6254	6514	6762	7012	5974	5865	11225	11453	14040	12401	6793	10445
64	7191	8901	8153	7992	7191	8217	9201	10152	10400	7992	9201	12276
65	5535	7003	5464	5502	5044	5626	8092	8992	10526	10886	7296	7395
66	6925	8326	8502	8777	6925	7975	8283	10351	11217	8777	9201	9873
67	4959	6197	5548	5792	5054	6194	8410	7373	10886	10849	6339	8962
68	9466	10342	10009	10037	9466	10388	12296	11585	11069	10037	12296	12316
69	6887	7250	7535	6933	6355	6613	9514	10003	10237	10916	8419	9680
70	9666	9223	10390	9522	9859	8929	10629	13058	12677	10577	12904	10704
71	5208	5388	5707	5782	4973	5037	10312	8822	12050	9991	6339	9316
72	751	792	933	933	732	794	1185	978	1117	1254	788	1048
73	760	875	748	762	649	683	855	1049	1125	1200	698	934
74	747	790	957	931	694	714	974	1222	1170	1096	791	813
75	670	722	750	793	634	630	982	1135	1177	1184	748	959
76	821	990	786	761	810	704	1132	1164	1133	981	904	950
77	700	676	687	637	641	606	1000	832	1344	1103	671	716
78	837	927	840	825	809	762	1227	1211	1101	987	859	947
79	687	575	701	776	581	613	907	997	1315	1165	736	818
80	672	825	806	811	653	694	795	993	1149	930	786	770
81	596	681	625	629	509	693	833	960	1213	941	588	837
82	684	703	722	712	639	606	900	1028	884	962	786	812
83	564	572	606	585	515	524	841	920	1017	1046	567	726
84	711	818	758	874	720	786	1047	1142	1048	950	819	929
85	618	797	728	645	547	568	1014	933	1042	1141	724	871
86	668	735	742	778	666	729	1038	1106	1158	1032	824	930
87	613	625	620	627	557	559	884	813	1186	963	664	752
88	610	693	670	677	593	593	1148	827	996	956	653	751
89	782	593	727	704	487	523	1013	767	1111	931	540	737
90	581	668	745	657	562	677	903	1016	1003	1009	633	752
91	686	641	705	797	570	548	791	915	1173	950	545	717
92	1200	1125	1246	1012	937	973	1439	1289	1665	1331	1026	1188
93	991	920	1067	1065	854	857	1480	1331	1514	1621	904	1128
94	1182	1172	1136	1015	942	968	1662	1347	1616	1375	960	1063
95	1100	874	980	1047	854	881	1478	1299	1684	1510	888	1074
96	1034	1069	1178	1105	944	1062	1241	1463	1418	1330	1003	1118
97	914	937	993	996	809	937	1347	1350	1618	1362	885	1176
98	1021	908	1009	1029	878	939	1254	1399	1421	1341	983	1163
99	971	1040	1014	900	784	777	1155	1333	1565	1376	822	1001
100	942	1035	995	995	904	951	1332	1440	1374	1487	1018	1387

Tabla 15. Resultados (continuación).

	Método de lanzamiento						Método de Giffler y Thompson					
Problema	SPT	LPT	SRPT	SJT	LNRO And SPT	WINQ And SPT	SPT	LPT	SRPT	SJT	LNRO And SPT	WINQ And SPT
101	1081	902	1113	1031	845	866	1439	1257	1534	1364	912	1143
102	1021	1111	1136	1152	956	1034	1416	1273	1603	1427	1070	1258
103	898	923	901	938	822	895	1360	1322	1666	1493	887	1113
104	1045	1183	1335	1220	951	951	1569	1570	1578	1555	985	1202
105	1040	1001	1145	1111	891	946	1453	1395	1747	1588	945	1262
106	1019	1102	1237	1158	964	925	1344	1432	1734	1531	985	1260
107	908	924	1073	990	897	909	1319	1343	1829	1593	954	1115
108	1049	1121	1324	1312	958	959	1487	1270	1536	1383	1008	1191
109	1067	902	1366	925	878	871	1547	1327	1668	1689	889	1174
110	978	993	1105	1043	929	937	1559	1201	1725	1629	1007	1183
111	956	956	1001	1041	875	910	1344	1157	1583	1418	894	1116
112	1496	1467	1446	1446	1222	1328	1835	1722	1747	1774	1322	1514
113	1310	1221	1344	1386	1199	1157	2070	1515	2260	1858	1175	1386
114	1386	1378	1596	1370	1184	1146	1715	1703	1876	1898	1283	1463
115	1268	1146	1241	1266	1136	1124	1760	1570	2120	1805	1137	1521
116	1305	1240	1365	1358	1039	1039	1474	1564	1788	1732	1088	1260
117	1093	1084	1181	1131	968	1012	1520	1521	1808	1833	1068	1291
118	1158	1234	1244	1485	1007	1108	1438	1416	1742	1782	1099	1179
119	1076	1081	1090	1121	981	974	1555	1361	1706	1718	1021	1164
120	1474	1230	1396	1267	1150	1207	1761	1827	1806	1662	1250	1471
121	1189	1160	1404	1338	1141	1125	1622	1489	1808	2195	1172	1324
122	1314	1301	1372	1335	1184	1278	1728	1461	1986	1739	1184	1305
123	1173	1258	1209	1154	1062	1103	1715	1685	1937	1820	1187	1423
124	1427	1434	1701	1646	1292	1292	1772	1755	1973	1878	1293	1483
125	1197	1241	1334	1229	1185	1139	1655	1557	1880	1715	1186	1261
126	1519	1357	1679	1447	1276	1124	1926	1734	1980	1957	1338	1442
127	1227	1176	1294	1198	1112	1112	1973	1636	1996	1971	1140	1450
128	1339	1612	1728	1660	1385	1455	1732	1798	1964	1798	1623	1546
129	1229	1256	1300	1332	1320	1277	1836	1753	2083	1832	1413	1487
130	1273	1485	1596	1598	1331	1300	1815	1670	2057	1773	1575	1569
131	1138	1386	1281	1272	1135	1123	1943	1714	2247	1987	1201	1421
132	1265	1229	1158	1268	1096	1203	1466	1486	1524	1498	1302	1187
133	839	998	893	917	824	915	1269	1362	1434	1890	1035	1304
134	1085	1154	1042	1240	1057	1008	1276	1351	1598	1397	1112	1198
135	751	769	814	766	736	747	1505	1302	1831	1746	859	1194
136	924	1082	1237	1120	873	929	1124	1091	1511	1486	954	1223
137	918	839	854	898	775	904	1252	1121	1559	1308	822	965
138	966	946	1047	1047	800	974	1167	1173	1291	1342	906	1061
139	704	679	761	669	701	725	1259	1222	1784	1576	802	1134
140	981	1114	1131	1111	954	947	1268	1391	1534	1502	1035	1264
141	734	868	1071	822	788	856	1705	1091	1532	1591	889	1253
142	987	1183	1051	1102	954	1041	1195	1364	1552	1506	1045	1337

Tabla 16. Resultados (continuación).

	Método de lanzamiento						Método de Giffler y Thompson					
Problema	**SPT**	**LPT**	**SRPT**	**SJT**	**LNRO And SPT**	**WINQ And SPT**	**SPT**	**LPT**	**SRPT**	**SJT**	**LNRO And SPT**	**WINQ And SPT**
143	667	663	731	724	668	686	1276	1293	1589	1538	764	1036
144	940	1062	986	981	975	980	1212	1340	1309	1236	1106	1169
145	875	850	877	1060	839	851	1410	1453	1427	1608	1115	1381
146	972	1278	922	907	1012	943	1271	1250	1298	1419	1057	1214
147	650	690	686	668	647	692	1557	1447	1706	1675	829	1125
148	1000	1272	1251	1201	999	1127	1512	1281	1437	1561	1114	1200
149	890	910	968	968	940	990	1412	1390	1668	1569	957	1512
150	1007	1107	1206	1072	989	1132	1385	1500	1666	1347	1114	1075
151	849	762	763	817	808	784	1478	1307	1726	1706	864	1147
152	1324	1451	1651	1541	1255	1411	1836	1827	2045	1928	1467	1423
153	1144	1213	1377	1353	1009	1031	1607	1744	1936	1907	1176	1620
154	1662	1237	1578	1457	1224	1326	1932	1741	1975	2068	1381	1558
155	986	937	1052	1075	912	930	1997	1986	2298	2265	1091	1459
156	1180	1315	1479	1563	1188	1271	1629	1622	1916	1847	1346	1590
157	993	1243	1268	1157	941	980	1803	1486	2155	2048	1178	1435
158	1147	1401	1460	1503	1132	1404	1552	1622	2100	1854	1232	1474
159	848	941	1108	1086	796	1098	1603	1588	2289	2014	977	1617
160	1162	1302	1577	1254	1166	1132	1726	1818	1971	1790	1341	1264
161	1082	1024	1169	1298	925	1063	1722	1743	2430	1936	1155	1489
162	1235	1394	1536	1294	1080	1090	1826	1821	1773	1846	1315	1563
163	979	1045	1104	1159	846	1131	1942	1771	2605	2122	1012	1567
164	1203	1245	1231	1517	1124	1095	1682	1696	1915	1677	1266	1328
165	1123	1193	1158	1154	955	1041	1838	1533	2169	1961	1204	1500
166	1111	1181	1364	1188	1080	1140	1864	1500	1890	1867	1255	1618
167	951	845	1183	1111	842	859	1910	1955	2460	2400	1015	1660
168	1449	1374	1335	1499	1209	1192	1691	1536	2027	1977	1374	1627
169	1112	1136	1353	1202	1063	1048	1802	1821	1869	1995	1134	1570
170	1175	1354	1418	1308	1067	1281	1742	1697	1983	1979	1309	1647
171	1036	895	1202	1188	876	939	1708	1742	2436	2430	1015	1651
172	1498	1741	1826	1754	1406	1424	2140	2050	2535	2372	1669	1862
173	1402	1358	1620	1584	1151	1290	2162	2182	2451	2571	1477	1897
174	1494	1477	1781	1679	1319	1511	2174	2111	2209	2257	1622	1817
175	1261	1260	1310	1396	1108	1244	2491	2363	3057	2872	1260	1719
177	1269	1512	1596	1665	1257	1479	2388	2382	2712	2407	1438	1651
178	1695	1734	1869	1932	1441	1348	2355	2300	2676	2551	1706	1844
179	1441	1395	1397	1353	1141	1291	2770	2111	2810	3026	1289	1938
180	1541	1844	1822	1529	1473	1572	2247	2370	2432	2243	1658	1947
181	1668	1423	1640	1736	1269	1315	2642	2038	3003	2536	1373	2003
182	1514	1592	1799	1646	1305	1539	1932	1973	2337	2367	1600	1619
183	1550	1419	1388	1311	1111	1142	2609	2167	2975	2734	1274	1829
184	1536	1759	1826	1724	1341	1568	2293	2289	2352	2201	1555	1747
185	1383	1281	1573	1400	1117	1330	2165	1986	2360	2259	1370	2011

Tabla 17. Resultados (continuación).

	Método de lanzamiento						Método de Giffler y Thompson					
Problema	SPT	LPT	SRPT	SJT	LNRO And SPT	WINQ And SPT	SPT	LPT	SRPT	SJT	LNRO And SPT	WINQ And SPT
186	1470	1652	1748	1699	1338	1492	2270	2030	2439	2225	1536	1718
187	1419	1252	1295	1425	1070	1131	2550	2176	2895	2574	1187	1742
188	1792	1902	2014	1963	1565	1562	2228	2019	2638	2316	1758	1855
189	1582	1478	1517	1635	1252	1357	2447	2060	2629	2333	1540	1840
190	1626	1559	1778	1802	1486	1560	2136	2363	2519	2706	1713	1797
191	1360	1500	1378	1442	1110	1214	2495	2230	3041	2808	1270	1884
192	2001	2245	2359	2270	1948	1910	2977	2690	3129	3139	2178	2389
193	1797	1959	1835	1831	1650	1760	2824	2733	2991	3207	1866	2320
194	1847	2167	2174	2218	1828	1860	3016	3163	2948	2952	2066	2188
195	1901	1786	1803	2055	1587	1841	3389	3102	4132	3839	1675	2456
196	2357	2437	2436	2327	1868	1995	2984	3044	3281	3147	2195	2666
197	1946	2067	2128	2093	1792	2008	3131	3006	3691	3546	1988	2635
198	2160	2226	2402	2591	1896	1903	3079	3167	3580	3286	2175	2594
199	2050	1880	2084	2093	1726	1808	3731	3164	4179	4171	1857	2643
200	1901	2316	2474	2142	1810	1909	2544	2893	3284	2699	2065	2115
201	2047	1773	1926	1954	1580	1884	2861	2894	3274	3162	1776	2435
202	2082	2262	2226	2227	1762	1782	2597	2866	3250	2900	1952	2244
203	1661	1720	1780	1860	1539	1594	3238	3186	3939	3714	1646	2404
204	2070	2330	2177	2110	1806	1887	2944	2766	3362	3068	2146	2402
205	1881	1807	1973	1919	1636	1825	3109	2430	3381	3114	1845	2290
206	1907	2383	2163	2256	1817	2052	2857	2667	3010	2867	2126	2785
207	1778	1663	1763	1762	1597	1620	3481	2954	4112	3713	1687	2628
208	2135	2395	2370	2383	2069	2217	2825	2913	3318	3009	2527	2683
209	1762	1874	2022	1960	1778	1827	2787	2828	3264	3212	1975	2471
210	2054	2382	2306	2411	1943	2233	2813	2869	3374	3290	2348	2601
211	1835	1710	1811	1953	1603	1644	3297	3189	3953	4113	1739	2623
212	1799	1949	1879	1894	1451	1706	2315	1993	2132	2782	1584	1691
213	1296	1628	1475	1507	1304	1262	2532	1792	2449	2284	1406	1848
214	1671	1579	1725	1605	1401	1441	1999	1747	2486	2301	1523	1743
215	978	998	1049	1038	1003	1012	2379	2144	2714	3139	1184	1730
216	1669	1944	2058	1745	1663	1615	2047	2199	2562	2430	1766	1742
217	1334	1425	1611	1415	1379	1298	2429	2097	2742	2363	1498	1982
218	1809	1907	1934	1782	1661	1558	2000	2173	2695	2503	1773	2023
219	1044	1037	1050	1032	1036	1070	2677	2369	2972	2798	1273	2275
220	1404	1732	1752	1838	1435	1431	2135	1766	2180	1941	1563	1772
221	1331	1364	1309	1377	1295	1359	2090	1943	2374	2461	1344	1720
222	1510	1690	1726	1575	1409	1592	1949	2076	2133	2240	1475	1709
223	962	989	1045	984	972	968	2481	1937	2801	2889	1144	1752
224	1604	1625	1848	1602	1525	1579	1907	2105	2438	2239	1615	1937
225	1297	1352	1445	1363	1252	1279	2331	2025	2409	1993	1383	1849
226	1541	1638	1775	1602	1363	1601	2022	2153	2406	2245	1669	1823
227	1008	976	1039	1021	994	978	2422	2384	2714	2804	1144	1817

Tabla 18. Resultados (continuación).

	Método de lanzamiento						Método de Giffler y Thompson					
Problema	**SPT**	**LPT**	**SRPT**	**SJT**	**LNRO And SPT**	**WINQ And SPT**	**SPT**	**LPT**	**SRPT**	**SJT**	**LNRO And SPT**	**WINQ And SPT**
228	1476	1840	1557	1613	1393	1472	1823	2203	2122	1944	1651	1570
229	1159	1325	1310	1665	1241	1168	1937	2270	2524	2459	1360	1719
230	1342	1705	1539	1512	1309	1344	2108	2060	2417	2399	1662	1694
231	975	993	1069	1066	979	1015	2173	1999	2812	2716	1135	1969
232	55	64	60	60	55	54	77	86	119	85	52	88
233	47	55	66	55	36	46	73	78	77	91	38	59
234	274	254	336	337	223	217	444	426	396	421	222	302
235	90	126	123	123	74	107	116	146	169	143	90	106
236	198	224	230	203	189	208	308	269	322	318	206	239
237	108	117	124	113	88	108	121	197	228	183	84	162
238	242	262	266	262	182	204	264	348	349	350	183	268
239	614	565	623	569	523	526	824	751	785	798	552	680
240	507	449	492	491	352	455	748	697	741	745	385	592
241	416	448	415	379	244	341	562	650	706	631	273	515
242	88	75	70	83	68	75	94	92	100	100	62	76
243	50	53	59	59	50	62	70	88	96	71	57	66
244	83	72	83	83	59	67	89	83	98	98	73	83
245	54	47	55	55	51	47	83	79	94	94	54	67
246	1074	1295	1334	1262	1173	1152	1429	1500	1525	1477	1345	1233
247	905	913	895	876	919	929	1252	1450	1439	1425	932	1205
248	1124	1147	1094	1157	1051	1039	1345	1400	1500	1349	1316	1260
249	666	737	685	761	658	696	1503	1185	1815	1586	816	1073
250	1023	1258	1334	1244	1102	1257	1371	1474	1679	1530	1234	1277
251	1144	1224	1367	1224	1065	1189	1371	1517	1679	1571	1191	1364
252	1060	1209	1306	1248	1173	1143	1429	1480	1461	1530	1256	1348
253	1060	1295	1342	1248	1141	1122	1389	1480	1460	1530	1251	1426
254	1060	1295	1342	1248	1141	1159	1389	1480	1460	1530	1251	1204
255	1055	1247	1122	1170	1097	1081	1417	1487	1298	1444	1256	1258
256	1012	1188	1053	1166	977	1091	1421	1219	1356	1294	1147	1162
257	1267	1631	1422	1324	1565	1395	1809	1726	1906	1624	1839	1844
258	1284	1281	1335	1350	1199	1279	1923	1655	1859	1754	1362	1402
259	1360	1613	1364	1307	1538	1392	1843	1884	2004	1855	1750	1749
260	1237	1110	1334	1275	1052	1055	1781	1458	1977	1686	1075	1223
261	1478	1410	1423	1458	1344	1213	1481	1758	1524	1714	1522	1393
262	937	1037	966	945	919	919	1445	1125	1508	1767	1034	1273
263	1170	1395	1306	1299	1267	1247	1534	1586	1659	1576	1486	1391
264	749	795	760	765	780	748	1532	1279	1584	1783	881	1346
265	1113	1254	1222	1162	1219	1281	1639	1672	1458	1780	1560	1231
266	941	1116	954	1033	930	1061	1343	1582	1934	1667	1040	1253
267	1071	1341	1270	1090	1162	1119	1523	1540	1525	1486	1565	1357
268	752	694	757	735	759	702	1564	1489	1762	1968	834	1320
269	1175	1293	1170	1166	1005	1116	1251	1413	1539	1347	1152	1413

Tabla 19. Resultados (continuación).

	Método de lanzamiento						Método de Giffler y Thompson					
Problema	SPT	LPT	SRPT	SJT	LNRO And SPT	WINQ And SPT	SPT	LPT	SRPT	SJT	LNRO And SPT	WINQ And SPT
270	826	875	1180	1050	791	824	1258	1293	1495	1419	1029	1123
271	1001	1296	1140	1134	992	1144	1331	1558	1482	1422	1252	1260
272	735	700	732	680	735	832	1435	1311	1815	1476	861	1025
273	1179	1430	1350	1295	1376	1257	1561	1862	1779	1425	1665	1383
274	944	1061	957	941	877	812	1421	1487	1628	1625	1109	1290
275	1152	1357	1346	1281	1269	1240	1376	1684	1637	1484	1559	1469
276	770	808	866	741	773	758	1239	1488	2204	1648	771	1381
277	1236	1415	1455	1351	1224	1190	1618	1595	1686	1634	1371	1335
278	990	936	1085	953	917	1120	1517	1555	1850	1514	1121	1250
279	1302	1399	1377	1337	1179	1227	1550	1648	1591	1600	1293	1575
280	838	792	806	861	781	801	1516	1688	1802	1600	821	1329
281	1128	1099	1106	1198	1043	1058	1118	1355	1437	1520	1139	1289
282	896	925	934	925	795	886	1319	1123	1464	1429	914	1172
284	606	629	644	643	601	639	1257	1292	1673	1448	772	998
285	1190	1298	1378	1343	1201	1242	1527	1696	1667	1743	1454	1388
286	954	917	932	989	1021	1009	1412	1638	1654	1478	1090	1200
287	1231	1341	1148	1257	1246	1230	1492	1558	1700	1766	1300	1392
288	753	715	732	761	727	719	1556	1417	1734	1490	840	1178
289	504	470	562	487	516	456	666	684	656	645	557	565
290	384	459	427	426	357	368	554	614	770	661	436	582
291	528	555	575	474	458	464	637	676	709	674	532	557
292	318	296	311	323	319	287	687	715	756	753	367	521
293	1107	1176	1343	1237	1304	1203	1274	1578	1707	1574	1402	1220
294	777	981	996	1028	891	842	1228	1347	1447	1348	986	1185
295	1044	1155	1278	1136	1168	1100	1327	1538	1597	1485	1350	1346
296	606	692	728	725	601	713	1158	1300	1573	1640	781	1304
297	1262	1268	1400	1205	1164	1119	1524	1389	1727	1508	1190	1058
298	915	985	1071	901	905	933	1589	1476	1438	1371	995	1352
299	1152	1283	1247	1190	1182	1185	1557	1478	1661	1460	1173	1301
300	764	735	852	748	731	740	1794	1274	1906	1610	806	1156

Tabla 20. Resultados (continuación).

Se puede observar en amarillo con qué combinación de método y regla de lanzamiento se obtiene el menor valor de *makespan* para cada uno de los trescientos problemas.

ANEXO 2. Código fuente

En este segundo anexo se describe el código fuente del programa implementado en su totalidad.

Principal

Siguiendo el esquema de la figura 9, primero se presenta la clase *Principal* del paquete *scheduling*. Como se observa a continuación, esta es la clase en la que se leen los datos de los trescientos problemas que se han comparado para realizar el estudio.

```
package scheduling;


import java.io.*;
import java.util.StringTokenizer;

import scheduling.data.*;
import scheduling.metodos.Scheduler;

public class Principal {

	public static Problema muestraContenido(String archivo) {

		String cadena;
		Problema problema=null;

	  try{
			FileReader f = new FileReader(archivo);
			BufferedReader b = new BufferedReader(f);
			cadena=b.readLine();
			StringTokenizer st=  new StringTokenizer(cadena);
		    int nOrd=Integer.parseInt(st.nextToken());
		    int nMaq=Integer.parseInt(st.nextToken());
		    float mpo=Float.parseFloat(st.nextToken());
		    problema=new Problema(nMaq);
		    for(int i=0;i<nOrd;i++){
		     cadena = b.readLine();
				Orden orden=new Orden(0,0);
				problema.addOrden(orden);
			   st=  new StringTokenizer(cadena);
			   int nOp=Integer.parseInt(st.nextToken());
```

```
                    for(int j=0;j<nOp;j++){
                           int nMod=Integer.parseInt(st.nextToken());
                           Operacion op=new Operacion(i,j);
                           orden.addOperacion(op);
                           for(int k=0;k<nMod;k++){
                            int maq=Integer.parseInt(st.nextToken());
                            int pt=Integer.parseInt(st.nextToken());
                            op.addModo(new Modo(maq-1,pt));
                           }
                    }

                }

                b.close();
            }catch(Exception e){
               System.out.println(e);
            }

          return problema;
        }


    public static void main(String[] args) throws IOException {

       File dir= new File("/Users/Clara G/Desktop/Ficheros");
       String[] ficheros = dir.list();
       for(int x=0; x<ficheros.length;x++){
       Problema p=muestraContenido(dir+"/"+ficheros[x]);
       Programa prg=Scheduler.schedule(p);
         System.out.println(prg.cMax());

       }
    }

}
```

Modo

La clase *modo* ya pertenece al paquete *scheduling.data* que representa el diagrama de clases. El que en un problema existan modos es la base del *job shop* flexible. Esta clase, simplemente se encarga de representar los modos de una determinada operación, que implicarán el que se realice en una u otra máquina y, por consiguiente, que tenga uno u otro tiempo de proceso. A continuación se observa cómo se ha programado:

```
package scheduling.data;

public class Modo {
	private int maquina;
	private int tiempoProceso;

	public Modo(int maquina, int tiempoProceso){
		this.maquina = maquina;
		this.tiempoProceso = tiempoProceso;
	}

	public int getMaquina() {
		return maquina;
	}

	public int getTiempoProceso() {
		return tiempoProceso;
	}
}
```

Operación

Esta clase, incluida también en el paquete *scheduling.data,* representa todas las operaciones de un determinado problema. Por lo tanto, esta clase ha sido programada de la siguiente manera:

```
package scheduling.data;

import java.util.ArrayList;
import java.util.Collection;
import java.util.HashMap;


public class Operacion {

	private HashMap<Integer, Modo> modos=new HashMap<Integer,
Modo>();
	private int indice;
	private int orden;
	private int fechaComienzo;
	private int fechaFinalizacion;
	private int maquina;

	public Operacion(int orden, int indice){
		this.orden=orden;
```

```
		this.indice=indice;
	}



	public void programar(int maquina, int fechaComienzo){
		this.maquina=maquina;
		this.fechaComienzo=fechaComienzo;
		fechaFinalizacion=fechaComienzo +
modos.get(maquina).getTiempoProceso();

	}

	public int getFechaComienzo() {
		return fechaComienzo;
	}

	public int getFechaFinalizacion() {
		return fechaFinalizacion;
	}

	public int getMaquina() {
		return maquina;
	}

	public int getOrden(){
		return orden;
	}

	public int getIndice(){
		return indice;
	}
	public void addModo(Modo m){
		modos.put(m.getMaquina(),m);
	}

	public ArrayList<Modo> getModos(){
		return new ArrayList<Modo>(modos.values());
	}

	public Modo getModo(int maquina){
		return modos.get(maquina);
	}

	public int minTiempoProceso(){
		int minTiempoProceso=Integer.MAX_VALUE;
		for(Modo modo: modos.values()){
			minTiempoProceso=Math.min(minTiempoProceso,
modo.getTiempoProceso());
		}

		return minTiempoProceso;
	}

}
```

Orden

Siguiendo con las clases del paquete *scheduling.data*, esta clase representa a las órdenes de un determinado problema. A continuación se puede observar cómo ha sido programada dicha clase:

```
package scheduling.data;

import java.util.ArrayList;
import java.util.HashMap;

public class Orden {
	private int dueDate;
	private int peso;
	private ArrayList<Operacion> operaciones=new
ArrayList<Operacion>();



	public Orden(int dueDate, int peso){
	this.dueDate = dueDate;
	this.peso = peso;
	}

	public int getDueDate() {
	return dueDate;
	}

	public int getPeso() {
	return peso;
	}

	public void addOperacion(Operacion o){
	operaciones.add(o);
	}

	public ArrayList<Operacion> getOperaciones(){
	return operaciones;
	}

	/*Devuelve el mínimo tiempo restante sin contar el tiempo de la
operación indicada
	El mínimo es porque se selecciona el modo de menor duración*/
	public int minTiempoRestante(int operacion){
		int tiempoRestante=0;
		if(operacion>=operaciones.size()-1){
			return 0;
		}

		for(int i=operacion+1;i<operaciones.size();i++){
```

```
		tiempoRestante=tiempoRestante+operaciones.get(i).minTiempoProceso
();
			}

			return tiempoRestante;
		}

		//Devuelve el mínimo tiempo de proceso total de la orden
		public int minTiempoProcesoTotal(){
			return minTiempoRestante(-1);
		}

		//Devuelve el número de operaciones restantes sin incluir la
operación indicada
		public int operacionesRestantes(int operacion){
			if(operacion>=operaciones.size()-1){
				return 0;
			}

			return operaciones.size()-1-operacion;
		}

}
```

Problema

Al igual que las tres clases anteriores, esta también pertenece al paquete *scheduling.data*. Hace referencia a los distintos problemas existentes:

```
package scheduling.data;

import java.util.*;

public class Problema {
		private ArrayList<Orden> ordenes=new ArrayList<Orden>();
		private int numMaquinas;

		public Problema(int numMaq){
			this.numMaquinas=numMaq;
		}


		public int getNumMaquinas() {
			return numMaquinas;
		}
```

```
	public void addOrden(Orden o){
		ordenes.add(o);
	}

	public int numOrdenes(){
		return ordenes.size();
	}

	public Orden getOrden(int orden){ // Devuelve la orden que hay
almacenada en la posición i del array
		return ordenes.get(orden);
	}

	public ArrayList<Orden> getOrdenes(){
		return ordenes;
	}
}
```

Programa

La siguiente clase del paquete *scheduling.data* hace referencia al programa. El programa, como se dijo en el capítulo 3, representa el programa de las operaciones de cáda máquina. A continuación se puede observar cómo ha sido programada:

```
package scheduling.data;

import java.util.ArrayList;

public class Programa {
	private ArrayList <ProgramaMaquina> programaMaquinas= new
ArrayList <ProgramaMaquina>();

	public Programa(int numMaq){
		for(int i=0;i<numMaq;i++){
			programaMaquinas.add(new ProgramaMaquina());
		}
	}

	public void addOperacion(Operacion op){
		int maq=op.getMaquina(); // Máquina donde se ha programado
la operación op
		ProgramaMaquina pm=programaMaquinas.get(maq); /Programa de
la máquina maq
		pm.addOperacion(op);
	}

	public void addProgramaMaquina(ProgramaMaquina p){
```

```
			programaMaquinas.add(p);
		}

		public ArrayList<ProgramaMaquina> getProgramaMaquinas(){
			return programaMaquinas;

		}

		public int fechaDisponibilidad(int maquina){
			return programaMaquinas.get(maquina).fechaDisponibilidad();
		}

		public void imprimir(){
			for(ProgramaMaquina pm: programaMaquinas){
				pm.imprimir();
			}
		}

		public int cMax(){
			int cMax=0;
			for(ProgramaMaquina pm: programaMaquinas){
				cMax=Math.max(cMax, pm.fechaFinalizacion());
			}
			return cMax;
		}

}
```

ProgramaMáquina

Ésta es la última clase del paquete *scheduling.data*. Esta clase, que representa a los distintos programas máquina, ha sido programada como sigue:

```
package scheduling.data;

import java.util.ArrayList;

public class ProgramaMaquina {

		private ArrayList<Operacion> operaciones= new
ArrayList<Operacion>();


		public void addOperacion(Operacion o){
			operaciones.add(o);
		}
```

```
	public ArrayList<Operacion> getOperaciones(){
		return operaciones;
	}

	public int fechaDisponibilidad() {
		if(operaciones.size()==0)
			return 0;
		Operacion op = operaciones.get(operaciones.size()-1);
		return op.getFechaFinalizacion();

	}

	public void imprimir(){
		String s="";
		for(Operacion op: operaciones){

	s=s+"["+op.getIndice()+"."+op.getOrden()+"("+op.getFechaComienzo(
)+", "+op.getFechaFinalizacion()+")]";
		}
		System.out.println(s);
	}

	public int fechaFinalizacion(){
		if(operaciones.size()==0){
			return 0;
		}

		return operaciones.get(operaciones.size()-
1).getFechaFinalizacion();
	}
}
```

Scheduler

Esta clase corresponde ya al paquete denominado *scheduling.metodos*. En ella han sido programados el método de lanzamiento, el método de Giffler y Thompson y las distintas reglas de lanzamiento tal y como se explicó en el capítulo 3. A continuación se puede observar cómo ha sido llevada a cabo su implementación:

```
package scheduling.metodos;

import java.util.ArrayList;
import java.util.HashSet;

import scheduling.data.*;
```

```
public class Scheduler {

	// Método para crear programas

	public static Programa schedule(Problema problema){
		int nm=problema.getNumMaquinas(); // nm es el número de
máquinas
		ArrayList<HashSet<Operacion>> opProgramables=new
ArrayList<HashSet<Operacion>>(); // Se crea el hashet

		for(int i=0;i<nm;i++){
			opProgramables.add(new HashSet<Operacion>());//
Hashset para poder almacenar las operaciones programables de cada
máquina (vacío)
		}

		Programa programa=new Programa(problema.getNumMaquinas());

		for(int i=0;i<problema.numOrdenes();i++){
			ArrayList<Operacion>
ops=problema.getOrden(i).getOperaciones(); // Se recorre cada orden y se
cogen las operaciones
			Operacion op=ops.get(0); // Máquina en la que se
realiza la primera operación de la orden
			ArrayList<Modo> modos=op.getModos();
			for(Modo m: modos){
				int maqOp=m.getMaquina();
				opProgramables.get(maqOp).add(op);// Almacena
la operación programable en el HashSet de la maquina donde se realiza
			}
		}

		ArrayList<Operacion>
elegibles=seleccionarElegiblesGT(problema, programa, opProgramables);

		while(elegibles.size()>0){
			Operacion opAP=selecOpSRPT(problema, programa,
elegibles, opProgramables);

			// Programar opAP
			programa.addOperacion(opAP);

			// Eliminar opAP de programables

			for(HashSet<Operacion> set: opProgramables){
				set.remove(opAP);
			}

			// Buscar sucesora de opAP y añadirla a programables

			Orden orden= problema.getOrden(opAP.getOrden());

			if(orden.getOperaciones().size()-1 >
opAP.getIndice()){
				Operacion
opSuc=orden.getOperaciones().get(opAP.getIndice()+1);
				ArrayList<Modo> modos=opSuc.getModos();
```

```
                        for(Modo m: modos){
                              int maqOp=m.getMaquina();// Máquina en
la que se realiza la primera operación de la orden

                              opProgramables.get(maqOp).add(opSuc);//
Almacena la operación programable en el HashSet de la máquina donde se
realiza
                        }
                  }

                  elegibles=seleccionarElegiblesGT(problema, programa,
opProgramables);
            }

            return programa;
      }

      private static ArrayList<Operacion>
seleccionarElegiblesLanz(Problema problema, Programa programa,
ArrayList<HashSet<Operacion>> programables){
            ArrayList<Operacion> elegibles=new ArrayList<Operacion>();
            int maquina=0;
            int fechaTemC=Integer.MAX_VALUE;

            for(int i=0;i<programables.size();i++){ // Se recorre el
array de operaciones programables (i es cada máquina)

                  HashSet<Operacion> opProM=programables.get(i);
                  int fechaTemDO=Integer.MAX_VALUE;

                  for(Operacion operacion: opProM){

                        if(operacion.getIndice()>0){// Si el índice de
la operación es mayor que 0, busco la orden de la operación

                              Orden orden=
problema.getOrden(operacion.getOrden());
                              Operacion
opPrec=orden.getOperaciones().get(operacion.getIndice()-1);// busco la
operación precedente

                              fechaTemDO=Math.min(fechaTemDO,
opPrec.getFechaFinalizacion()); // Se elige la operación que antes puede
empezar, min disponibilidad o fecha finalización

                        }else{
                              fechaTemDO=0; // Fecha más temprana de
disponibilidad de la operación
                        }
                  }

                  int
fechaTemCM=Math.max(programa.fechaDisponibilidad(i),fechaTemDO); //Fecha
más temprana de comienzo
```

```
                        if(fechaTemCM<fechaTemC){
                                maquina=i; // Si encontramos una fecha de
comienzo más temprana la seleccionamos y es la nueva fecha

                                fechaTemC=fechaTemCM;
                        }
                }

                for(Operacion operacion: programables.get(maquina)){// Se
recorre el hashset de operaciones programables de la maquina
seleccionada

                        if(operacion.getIndice()>0){// Si el índice de la
operación es mayor que 0, busco la orden de la operación

                                Orden orden=
problema.getOrden(operacion.getOrden());
                                Operacion
opPrec=orden.getOperaciones().get(operacion.getIndice()-1);// Busco la
operación precedente
                                if(opPrec.getFechaFinalizacion()<=fechaTemC){
                                        elegibles.add(operacion);
                                        operacion.programar(maquina,
Math.max(opPrec.getFechaFinalizacion(),programa.fechaDisponibilidad(maqu
ina)));
                                }
                        }else{
                                elegibles.add(operacion);
                                operacion.programar(maquina,
Math.max(0,programa.fechaDisponibilidad(maquina)));
                        }
                }


                return elegibles;
        }

        private static ArrayList<Operacion>
seleccionarElegiblesGT(Problema problema, Programa programa,
ArrayList<HashSet<Operacion>> programables){

                ArrayList<Operacion> elegibles=new ArrayList<Operacion>();
                int maquina=0;
                int fechaTemF=Integer.MAX_VALUE; // Fecha en la que antes
puede finalizar sumando la duración de la propia operación

                for(int i=0;i<programables.size();i++){ // Se recorre el
array de operaciones programables (i es cada máquina)

                        HashSet<Operacion> opProM=programables.get(i);
                        int fechaTemFO=Integer.MAX_VALUE; // Fecha
disponibilidad de la operación

                        for(Operacion operacion: opProM){

                                int fechaTemDO=0;
                                if(operacion.getIndice()>0){
```

```
                                    Orden orden=
problema.getOrden(operacion.getOrden());
                                    Operacion
opPrec=orden.getOperaciones().get(operacion.getIndice()-1);

      fechaTemDO=opPrec.getFechaFinalizacion();
                              }

                              int fechaTemCO=Math.max(fechaTemDO,
programa.fechaDisponibilidad(i));
                              int
fechaFO=fechaTemCO+operacion.getModo(i).getTiempoProceso();;
                              fechaTemFO=Math.min(fechaTemFO, fechaFO);
                        }

                        if(fechaTemFO<fechaTemF){
                              maquina=i; // Si encontramos una fecha de
comienzo más temprana la seleccionamos y es la nueva fecha
                              fechaTemF=fechaTemFO;
                        }
                  }

                  for(Operacion operacion: programables.get(maquina)){//
Recorro el hashset de operaciones programables de la maquina
seleccionada

                        if(operacion.getIndice()>0){// Si el índice de la
operación es mayor que 0, busco la orden de la operación
                              Orden orden=
problema.getOrden(operacion.getOrden());
                              Operacion
opPrec=orden.getOperaciones().get(operacion.getIndice()-1);// Se busca
la operación precedente
                              if(opPrec.getFechaFinalizacion()<fechaTemF);{
                                    elegibles.add(operacion);
                                    operacion.programar(maquina,
Math.max(opPrec.getFechaFinalizacion(),programa.fechaDisponibilidad(maqu
ina)));
                              }
                        }else{
                              elegibles.add(operacion);
                              operacion.programar(maquina,
Math.max(0,programa.fechaDisponibilidad(maquina)));
                        }
                  }

                  return elegibles;
      }



      static private Operacion selecOpSPT(Problema problema, Programa
programa, ArrayList<Operacion> elegibles, ArrayList<HashSet<Operacion>>
opProgramables){
                  int minPT = Integer.MAX_VALUE;
```

```
		Operacion op=null;

		for(int i = 0; i<elegibles.size(); i++){
			Operacion opi =elegibles.get(i);
			int pt =
opi.getModo(opi.getMaquina()).getTiempoProceso();
			if(pt<minPT){
				minPT=pt;
				op=opi;
			}
		}
		return op;
	}


	static private Operacion selecOpLPT(Problema problema, Programa
programa, ArrayList<Operacion> elegibles, ArrayList<HashSet<Operacion>>
opProgramables){
		int maxPT = Integer.MIN_VALUE;
		Operacion op=null;

		for(int i = 0; i<elegibles.size(); i++){
			Operacion opi =elegibles.get(i);
			int    pt =
opi.getModo(opi.getMaquina()).getTiempoProceso();
			if(pt>maxPT){
				maxPT=pt;
				op=opi;
			}
		}
		return op;
	}

	//Shortest Remaining Process Time (no se incluye el tiempo de la
operación indicada)
	static private Operacion selecOpSRPT(Problema problema, Programa
programa, ArrayList<Operacion> elegibles, ArrayList<HashSet<Operacion>>
opProgramables){
		int minRPT = Integer.MAX_VALUE;
		Operacion op=null;

		for(int i = 0; i<elegibles.size(); i++){
			Operacion opi =elegibles.get(i);
			int rpt =
problema.getOrden(opi.getOrden()).minTiempoRestante(opi.getIndice());
			if(rpt<minRPT){
				minRPT=rpt;
				op=opi;
			}
		}
		return op;
	}

	//Shortest Job process Time
	static private Operacion selecOpSJT(Problema problema, Programa
programa, ArrayList<Operacion> elegibles, ArrayList<HashSet<Operacion>>
opProgramables){
```

```
		int minJPT = Integer.MAX_VALUE;
		Operacion op=null;

		for(int i = 0; i<elegibles.size(); i++){
			Operacion opi =elegibles.get(i);
			int jpt =
problema.getOrden(opi.getOrden()).minTiempoProcesoTotal();
			if(jpt<minJPT){
				minJPT=jpt;
				op=opi;
			}
		}
		return op;
	}

	//Largest Number of Remaining Operations (como criterio de
desempate he incluido SPT)
	//Es muy fácil empatar con este criterio
	static private Operacion selecOpLNROandSPT(Problema problema,
Programa programa, ArrayList<Operacion> elegibles,
ArrayList<HashSet<Operacion>> opProgramables){
		int maxNRO = Integer.MIN_VALUE;
		int minPT = Integer.MAX_VALUE;
		Operacion op=null;

		for(int i = 0; i<elegibles.size(); i++){
			Operacion opi =elegibles.get(i);
			int    nro =
problema.getOrden(opi.getOrden()).operacionesRestantes(opi.getIndice());
			if(nro>maxNRO){
				maxNRO=nro;

	minPT=opi.getModo(opi.getMaquina()).getTiempoProceso();
				op=opi;
			}else if(nro==maxNRO){
				int pt =
opi.getModo(opi.getMaquina()).getTiempoProceso();
				if(pt<minPT){
					minPT=pt;
					op=opi;
				}
			}
		}
		return op;
	}

	//WINQ (Work In Next Queue) (Supongo que mínimo)
	//La he combinado tambien con SPT como criterio de desempate
	static private Operacion selecOpWINQandSPT(Problema problema,
Programa programa, ArrayList<Operacion> elegibles,
ArrayList<HashSet<Operacion>> opProgramables){
		int minWINQ = Integer.MAX_VALUE;
		int minPT = Integer.MAX_VALUE;
		Operacion op=null;

		for(int i = 0; i<elegibles.size(); i++){
```

```
                        Operacion opi =elegibles.get(i);
                        Orden orden=problema.getOrden(opi.getOrden());
                        if(opi.getIndice()<orden.getOperaciones().size()-1){
                                Operacion
sig=orden.getOperaciones().get(opi.getIndice()+1);
                                ArrayList<Modo> modos=sig.getModos();
                                //Como una operación se puede realizar en
varias máquinas hay que seleccionar la maquina con menor cola
                                //Hay que recorrer todos los modos para
encontrarla
                                for(Modo modo: modos){
                                        int winq =
opProgramables.get(modo.getMaquina()).size();
                                        if(winq<minWINQ){
                                                minWINQ=winq;
                                                op=opi;
                                        }else if(winq<minWINQ){
                                                int pt =
opi.getModo(opi.getMaquina()).getTiempoProceso();
                                                if(pt<minPT){
                                                        minPT=pt;
                                                        op=opi;
                                                }
                                        }
                                }

                        }else if(minWINQ == Integer.MAX_VALUE){
                                int pt =
opi.getModo(opi.getMaquina()).getTiempoProceso();
                                if(pt<minPT){
                                        minPT=pt;
                                        op=opi;
                                }
                        }
                }
                return op;
        }
}
```