UNIVERSIDAD DE BURGOS

DEPARTAMENTO DE INGENIERÍA CIVIL

TESIS DOCTORAL

ANÁLISIS DE LA RELAJACIÓN LAGRANGIANA COMO MÉTODO DE PROGRAMACIÓN DE TALLERES FLEXIBLES EN UN ENTORNO MULTIAGENTE

Autor: Juan José Laviós Villahoz

Directores: Dr. Ricardo del Olmo Martínez

Dr. José Alberto Araúzo Araúzo

Abril, 2013

Agradecimientos

La finalización de este trabajo no hubiera sido posible sin la colaboración de muchas personas que, de una u otra manera, me han acompañado en este camino.

En primer lugar quiero agradecer a mis directores de tesis la dedicación y disponibilidad constante que han tenido. A Ricardo del Olmo por su confianza en mi, paciencia y ayuda, sin cuya revisión y conocimiento no hubiera sido posible la finalización de esta tesis. A Alberto Araúzo por su generosidad al proponerme una línea de investigación que él comenzó en su tesis doctoral, y de la que el presente trabajo es continuación. Sus aportaciones han sido fundamentales en la realización de este trabajo.

A Miguel Ángel Manzanedo a quien nunca he encontrado el momento de agradecer las oportunidades que me ha dado para colaborar con él y por hacer posible que pueda estar realizando este trabajo tan apasionante.

A Luis Izquierdo por introducirme a trabajar con Netlogo, revisar pacientemente mis primeros pasos y resolver todas las dudas que le he consultado, sus orientaciones y consejos me han sido de gran utilidad. A Josema Galán y a Nacho Santos su apoyo y estímulo en los momentos en los que estaba más perdido. A Lourdes Saiz, por darme un consejo adecuado en el momento preciso.

También quiero agradecer a mis compañeros del Edificio A, Susana, Michel, Juanma, Nacho por el interés y ánimos que me han transmitido en todo este tiempo. A Teresa Peña, mi compañera de asignatura y a pesar de ello mi amiga, por su #optimismo y #pasion, por complementarme en mi trabajo y darme su apoyo moral.

A mis amigos por compartir los buenos momentos que hacen la vida más llevadera, en especial a Fran, cuyas conversaciones me ayudan a poner los pies en la tierra.

A mi familia, a mis padres, por ser ejemplo de superación en los momentos difíciles y por su apoyo siempre que lo he necesitado.

Índice

Introducción

Uno de los principales problemas de la mayoría de las empresas, en la actualidad, es como hacer frente a las demandas de un mercado cada vez más exigente. Las empresas han dejado de actuar dentro de un entorno local para pasar a competir en un mercado global. Esto ha hecho necesario aumentar la competitividad de los sistemas productivos de forma que se ajusten a las necesidades de los consumidores.

Los modelos clásicos de producción en masa que primaban bajos costes frente a flexibilidad, variedad de gama y rapidez de respuesta, han quedado obsoletos en los mercados actuales. Las empresas deben reducir los tamaños de lote, el ciclo de vida de los productos, así como los plazos de entrega; y todo ello aumentando la variedad de productos. Por ello, se proponen sistemas mucho más flexibles y ágiles, capaces de dar una rápida respuesta ante los cambios del mercado, proporcionando productos y servicios más personalizados y adaptados al cliente, pues éstos participan cada vez más en la definición de las especificaciones de los productos, exigiendo unos plazos de entrega cada vez más cortos.

En el entorno actual, los sistemas clásicos de gestión de la producción dejan de ser eficientes a pesar de su visión global del sistema y de su capacidad para generar buenos programas de producción. Los sistemas clásicos de planificación y control de la producción, basados en estructuras centralizadas y altamente jerarquizadas, tienen dos grandes inconvenientes. El primero es la dependencia exclusiva de un planificador central que realiza todos los cálculos y toma las decisiones, provocando un cuello de botella que limita la capacidad y ralentiza la toma de decisiones. El segundo, es su rigidez que hace que la modificación del sistema de control, altamente jerarquizado,

sea muy costosa en términos de tiempo y de trabajo de programación (Ouelhadj y Petrovic 2009). Las empresas, en consecuencia, deben evolucionar hacia estructuras distribuidas donde las decisiones se toman de forma descentralizada, principalmente para mejorar su respuesta ante los cambios y dotarlas de una mayor flexibilidad.

Para hacer frente a estas exigencias, se buscan nuevos sistemas de gestión, altamente automatizados, capaces de ofrecer características como robustez, estabilidad, adaptabilidad y uso eficiente de los recursos, mediante el diseño de sistemas modulares y distribuidos que proporcionen estas características dando una respuesta rápida y poco traumática a los cambios que se van produciendo. Esto se debe realizar a todos los niveles dentro del subsistema de producción, desde el diseño del producto y de los procesos, al nivel más operativo de la planta de producción como es la programación de operaciones.

Antecedentes y justificación

La toma de decisiones distribuida es considerada una alternativa a los sistemas de programación operaciones puramente centralizados, ya que permiten la incorporación de objetivos locales, preferencias y restricciones de cada recurso en el proceso de decisión (Kutanoglu y Wu 1999). El problema de programación de operaciones es un problema de toma de decisiones relacionado con la asignación de recursos a tareas sobre períodos de tiempo, y su fin es la optimización de uno o varios objetivos.

Los sistemas de producción son uno de los campos de aplicación más conocidos de este problema, donde las tareas son operaciones de distintos procesos y los recursos suelen ser los centros de trabajo o máquinas de un taller. Este problema, de gran interés práctico en la industria, se caracteriza por su complejidad debido en parte a su naturaleza combinatoria, y, por ello, es uno de los campos de investigación más activos dentro de la dirección de la producción (Shen 2002, Pinedo 2008). Se busca un método para la programación de sistemas de producción en un entorno dinámico. Este trabajo se centra en el problema de programación de talleres flexibles.

El paradigma de sistemas multiagente ha permitido que se desarrollen propuestas de sistemas de programación de la producción distribuidos. Las características propias de estos sistemas, como la autonomía, robustez y su naturaleza distribuida, hacen que sean uno de los paradigmas más adecuados para construir sistemas complejos y

robustos (Parunak 1996, Shen et al 2001, Brennan y Norrie 2001). Los sistemas multiagente están constituidos por unidades software autónomas y proactivas denominadas agentes. Los agentes se utilizan para encapsular entidades físicas y lógicas, e incluso funciones del sistema de producción. Mediante la interacción y la negociación entre los distintos agentes que conforman el sistema, se pretende resolver problemas de alta complejidad para los que las técnicas clásicas no han aportado soluciones totalmente satisfactorias. En las dos últimas décadas, este paradigma ha tenido un gran desarrollo en el entorno de planificación de operaciones (Shen et al 2006, Lee y Kim 2008).

Desde este punto de vista, el problema de programación de operaciones puede ser representado mediante un sistema multiagente compuesto por los siguientes elementos: (1) un conjunto de agentes que representan cada recurso o máquina. Cada máquina está caracterizada por una serie de habilidades que le permite realizar distintos tipos de operaciones, así como de su capacidad de producción que limita el número de operaciones simultáneas que puede realizar; (2) un conjunto de agentes que representan cada orden de trabajo.

Cada orden de trabajo tiene un determinado objetivo propio, que puede ser la finalización en una fecha determinada, minimización del coste de producción o del coste de penalización por retraso u otros, y está compuesta por una serie de tareas interrelacionadas mediante relaciones de precedencia. Para poder realizar las tareas es necesario utilizar los recursos que dispongan de las habilidades correspondientes. Este conjunto de agentes deberá coordinarse para que cada uno alcance sus propios objetivos, consiguiendo realizar el conjunto de órdenes de trabajo pendientes.

En estos sistemas se busca el comportamiento emergente a partir de las decisiones tomadas por las entidades individuales que lo componen. La combinación de los procesos de resolución individual de problemas y los protocolos de coordinación y negociación es uno de los retos de investigación en esta área, dado que lo que se gana en flexibilidad y rapidez de respuesta en los sistemas distribuidos, se pierde en eficiencia frente a los clásicos sistemas centralizados. Además, la dificultad para conseguir que su comportamiento sea predecible ha frenado su aplicación en entornos reales. Por esta razón, es necesario estudiar los mecanismos de coordinación que permiten su funcionamiento.

Uno de los mecanismos de coordinación utilizados en la programación distribuida de operaciones es la subasta. Una subasta “es un mecanismo de mercado que se caracteriza porque la asignación del bien intercambiado y sus precios se determinan mediante un conjunto de reglas en función de las pujas de los participantes” (McAfee

y McMillan 1987). Este método de coordinación implica crear un programa de producción basado en los precios que se generan a partir de un proceso de subasta, en el que los productos que deben ser fabricados -las órdenes de trabajo lanzadas en el plan de producción- pujan por el uso de los recursos o máquinas.

Así, el problema de programación de operaciones puede ser modelado como una subasta donde los bienes intercambiados son las unidades de tiempo en que se divide el horizonte de programación de cada recurso. Las órdenes de trabajo pujan en la subasta por aquellos intervalos de tiempo del horizonte de programación de los recursos que le permiten realizar su operación, tratando de maximizar sus propios objetivos (p.e. finalizar el producto dentro de plazo o minimizar la penalización por retraso).

Este mecanismo sigue los principios de los sistemas distribuidos, de forma que la información relevante de cada orden de trabajo que participa, como la fecha de entrega comprometida o la penalización por retraso, permanece oculta para el resto de participantes en la subasta. Ninguna de las órdenes de trabajo conoce cuáles son los objetivos de las otras órdenes de trabajo ni si están pujando por los mismos recursos.

Los precios actúan como señales que indican las preferencias del conjunto de participantes, no de forma individual, sobre cada ítem y permiten que se coordinen. Cada orden de trabajo propone un programa de producción de sus propias operaciones que optimice sus objetivos con los precios dados. De esta forma, la complejidad de cálculo se distribuye entre los distintos participantes, dividiendo el problema en otros más fáciles que pueden ser resueltos en paralelo (Wellman et al 2001).

La idea de subasta aparece de forma intuitiva como un mecanismo eficaz para coordinar elementos con intereses propios. Sin embargo, existen dos dificultades principales en su aplicación: (1) se va a subastar de forma simultánea un elevado número de ítems (cada periodo de tiempo en que se divide el horizonte de programación de cada recurso); y (2) los ítems subastados presentan complementariedades, de modo que la valoración de un determinado ítem depende de la combinación de productos a la que pertenezca. Debido a las restricciones de precedencia y capacidad que caracterizan a los problemas de programación de operaciones un ítem será valorado en función de los otros ítems que se adquieran al mismo tiempo en la subasta.

El modelado del proceso de programación de la producción como una subasta combinatoria permite tener en cuenta las dependencias originadas por las restricciones de precedencia y de capacidad que caracterizan a los problemas de programación de

operaciones. La subasta combinatoria es adecuada como medio de negociación y coordinación en aquellos problemas en los que se han de considerar las distintas valoraciones que se tienen sobre bienes interdependientes.

Hay distintos tipos de subasta combinatoria. En las de una ronda, los participantes envían una sola vez las distintas valoraciones de las posibles combinaciones de bienes (en nuestro problema intervalos de tiempo de uso de las máquinas) y el subastador asigna los bienes de modo que se maximiza el objetivo global. La alternativa es realizar la valoración de forma iterativa, de forma que los precios se fijan después de múltiples rondas.

Las subastas combinatorias iterativas tienen ventajas respecto a la de una sola ronda. En primer lugar, los participantes no necesitan realizar pujas sobre todo el conjunto de posibles combinaciones. En segundo lugar, en las subastas iterativas los participantes revelan en cada iteración sus preferencias, a través de los precios que se generan. En tercer lugar, las subastas iterativas son adecuadas en entornos dinámicos, donde los participantes entran y salen en cualquier momento.

Así mismo, hay diferencias en cuanto a la forma de realizar las pujas en la subasta. De esta forma, aparecen dos tipos principales de subastas iterativas. En un tipo, los participantes envían el precio que están dispuestos a pagar por las posibles combinaciones de bienes, y el subastador asigna los bienes a cada participante en base a esas valoraciones (subastas con fijación de cantidad). En el segundo tipo, los participantes envían como puja el conjunto de bienes que estarían dispuestos a adquirir dados unos precios. En cada iteración el subastador actualiza los precios en función de la demanda (subastas de fijación de precios).

El mecanismo puede ser descrito como sigue: existe un conjunto de recursos, cada uno representado por un agente. Cada recurso dispone de una serie de habilidades diferentes. El horizonte de programación de cada recurso está dividido en partes iguales, que son puestas a la venta en una subasta. Por otra parte, existe un conjunto de órdenes de trabajo que deben ser completadas. Cada orden de trabajo también está representada por un agente. Para completar cada orden de trabajo es necesario realizar un conjunto de tareas, ligadas entre sí por un conjunto de relaciones de precedencia. Para poder realizar las tareas es necesario utilizar los recursos que dispongan de las habilidades correspondientes. Las órdenes de trabajo tratan de minimizar su función de coste. Esta función de coste es la suma de la penalización por retraso más el coste de uso de los recursos, valorados con los precios generados en la subasta. Hay un agente que actúa como nodo central, que hace las funciones de subastador. El subastador recibe las pujas realizadas por las órdenes de trabajo y actualiza los precios

de cada unidad temporal de los recursos. Una vez conocidos los precios, las órdenes de trabajo reelaborarán sus pujas, repitiéndose el proceso. De forma iterativa, este ciclo continúa hasta que los precios se estabilizan (Kutanoglu y Wu 1999).

El precio de cada una de las unidades de tiempo aumenta o disminuye siguiendo un mecanismo walrasiano, de forma que el subastador compara la demanda sobre cada una de las unidades de tiempo con la capacidad del recurso. Si hay un exceso de demanda, el subastador sube el precio, mientras que si hay un exceso de capacidad, el subastador lo baja. Existen distintas formas de actualizar los precios: pueden actualizarse con un valor constante, de forma proporcional a la demanda, o de forma proporcional al exceso de demanda. La forma de actualizar los precios es importante, ya que va a determinar que el proceso iterativo tienda o no hacia el equilibrio.

Se puede establecer una analogía entre el algoritmo de Relajación Lagrangiana aplicado al problema de programación de la producción (Hoitomt et al 1993) y las subastas combinatorias. El proceso de actualización de precios es similar al proceso iterativo de actualización de los multiplicadores de Lagrange (Dewan y Joshi 2002). De este modo, los métodos de actualización utilizados en el método de Relajación Lagrangiana pueden ser usados para actualizar los precios en la subasta iterativa. Este mecanismo aporta el soporte matemático que proporciona las características y propiedades del método y ha sido utilizado para resolver diferentes problemas de programación de tareas.

Las decisiones tomadas por los agentes tienen gran influencia en la eficiencia del sistema. Los precios que aparecen el mecanismo de subasta aportan información en cada instante de la demanda que tiene cada uno de los recursos, y adaptará sus propuestas en función de dichos precios. El resultado final deberían ser programas más eficientes. Las principales dificultades en la aplicación del mecanismo de subasta como mecanismo de coordinación en un problema de programación de operaciones son las siguientes:

- Diseñar un mecanismo de coordinación cuyo comportamiento sea predecible. El mecanismo debe hacer que el sistema tienda hacia el equilibrio y que el funcionamiento que se consiga esté cercano al óptimo.
- Lograr una buena convergencia del método, de forma que la transición de un modo a otro de funcionamiento, cuando existen cambios en los parámetros del sistema sea rápida.
- Eliminar los elementos de coordinación central en los cálculos y su papel necesario en el mecanismo de tipo síncrono.

Objetivos

Esta tesis está relacionada con la programación de operaciones de tipo distribuido y analiza el método de Relajación Lagrangiana para su aplicación como mecanismo de generación de precios en el contexto de las subastas combinatorias iterativas. Distintos trabajos han aplicado estas técnicas para la resolución de distintos problemas dentro de este ámbito. En este trabajo se pretende estudiar distintos mecanismos de programación distribuidas basados en subastas y las distintas opciones que presentan, con el objeto de poder estudiar sus características y limitaciones para ser implementados en sistemas distribuidos de programación y control de la producción basados en agentes.

Se partirá del sistema desarrollado por Araúzo (2007) donde se implementó un sistema de programación y control distribuido en fabricación basado en agentes. En él se demostró la viabilidad de estos métodos implementados mediante estándares FIPA, pero también se vio la necesidad de profundizar sobre la metodología de coordinación basada en subastas, de estudiar sus características y sus límites. Ya que, como indica Shen (2002), el aumento del uso de protocolos de negociación basados en mercado o basados en subastas hace necesario la investigación y desarrollo de mecanismos más sofisticados de negociación. Los sistemas de negociación basados en subastas combinatorias tienen un alto interés en este campo.

El objetivo perseguido con el presente trabajo es:

"Estudiar la adecuación de los métodos de programación basados en subastas para ser implementados en sistemas multiagente de programación y control de la producción. Todo ello orientado al diseño de métodos que teniendo las ventajas de los procedimientos distribuidos (flexibilidad, robustez,…) presenten, a la vez, algunas de las bondades de los métodos centralizados (optimalidad y estabilidad)".

Para intentar alcanzar el éxito, este objetivo se ha desglosado en los siguientes objetivos parciales:

- Contextualizar el problema de programación de talleres flexibles definiendo el problema y revisando los principales métodos que se han aplicado a su resolución.
- Estudiar el método de coordinación basado en subastas, revisando los trabajos prácticos que lo han implementado como medio de coordinación de sistemas multiagente en entornos de producción, realizando su clasificación y caracterización en función de distintos criterios. Estudiar la relación entre las subastas combinatorias de tipo iterativo y el método de Relajación Lagrangiana

como método de resolución del problema de programación de la producción. Esta relación proporciona alternativas al método de actualización de precios. Analizar las propiedades del método de Relajación Lagrangiana que permitan determinar su capacidad de aplicación en entornos distribuidos.

- Determinar los principales parámetros de funcionamiento del método de Relajación Lagrangiana y sus valores óptimos en su aplicación al problema de programación de talleres flexibles.
- Estudiar alternativas al método del subgradiente que permitan aplicar el método de forma asíncrona y reducir el tiempo de computación en los cálculos locales.
- Estudiar las diferentes variantes del método de construcción de programas factibles, incidiendo sobre su calidad y su capacidad de distribución.
- Comparar el método de Relajación Lagrangiana con el método Walrasiano no adaptativo para la actualización de los precios. En estas comparaciones se tendrá en cuenta la calidad de la solución propuesta y la velocidad de convergencia del método.

Organización del documento

El documento se ha organizado en una Introducción, siete Capítulos, Conclusiones y Bibliografía.

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 de programación de talleres flexible y terminar estudiando su complejidad.

Los principales métodos para la resolución de la programación de operaciones en producción, desde un punto de vista estático y dinámico, se presentan en el capítulo 2.

En el capítulo 3 se analizan los sistemas de programación de operaciones de tipo distribuido y se vinculan con los sistemas multiagente, paradigma que ha permitido potenciar su desarrollo. El capítulo finaliza estudiando el modelado del problema de programación de operaciones desde un enfoque distribuido basado en agentes.

El siguiente capítulo, capítulo 4, recoge los mecanismos de subastas como mecanismos de coordinación en sistemas distribuidos y su aplicación a problemas de programación de operaciones en producción.

Los dos siguientes capítulos se centran en la Relajación Lagrangiana como método de resolución de problemas de programación entera, primero de forma general, estudiando sus principales propiedades y aplicaciones a la programación de operaciones (capítulo 5), para pasar a centrarse sobre el problema de programación de talleres y plasmando su relación con las subastas combinatorias (capítulo 6).

En el capítulo 7 se realizan una serie experimentos que permiten estudiar la aplicación de las subastas combinatorias en la resolución del problema de programación de talleres flexibles y la utilización del método de Relajación Lagrangiana como base de los métodos de actualización de precios en la subasta. Estos experimentos se agruparán en tres bloques. Se estudiarán, en el primer bloque, los aspectos relacionados con la actualización de precios en la subastas. En el segundo bloque, el análisis se centrará en los algoritmos para obtener soluciones factibles a partir de los programas propuestos en la subasta. En el último bloque, se comparará el un método de actualización de precios derivado del método de Relajación Lagrangiano con un método de actualización de precios no adaptativo.

A partir de los resultados obtenidos, se obtendrán las conclusiones del trabajo realizado y se propondrán líneas futuras de trabajo, finalizando este trabajo citando toda la bibliografía utilizada para su realización.

Capítulo 1. Programación de operaciones y el problema de programación de talleres

Este capítulo está dedicado a la definición del problema de programación de talleres u operaciones con enrutamiento flexible (*job shop scheduling problem*). Este es el problema sobre el que se va a aplicar el sistema de programación de operaciones basado en subastas. El capítulo se estructura como sigue: primero se define el problema de programación de operaciones aplicado al entorno de fabricación; después se explican los tipos más importantes de configuraciones en entornos productivos, que van a originar los tipos fundamentales de problemas en programación de la producción; a continuación, se clasifican los principales tipos de problemas de este tipo describiendo una de las propuestas de notación más utilizada en este ámbito; seguidamente se realiza una definición formal de los problemas de programación de talleres simples y con enrutamiento flexible; y, por último, se estudia el concepto de complejidad computacional, aplicándolo posteriormente a estos problemas.

1.1. Programación de operaciones

El término programación de operaciones (*scheduling*) se refiere a la búsqueda de programas de producción óptimos o cercanos al óptimo, sujetos a una serie de restricciones. Este problema puede aplicarse en distintos ámbitos: en gestión de producción trata de programar las operaciones de fabricación asignándolas a unas determinadas máquinas; en gestión de proyectos se programan las fases de realización del proyecto asignando cada uno a los distintos equipos y recursos; en informática versa sobre la ejecución de programas asignando los recursos de memoria y del procesador; o en logística, donde el transporte de materiales es asignado a los distintos medios de transporte y personal (Pinedo 2009).

La programación de operaciones permite: (1) proporcionar visibilidad del sistema de modo que se pueda conocer de antemano cual va a ser la capacidad utilizada o los medios a usar, permitiendo tomar decisiones adecuadas; (2) proporcionar grados de libertad para poder modificar el propio programa adaptándolo a las circunstancias cambiantes, previendo ciertas holguras en los elementos más críticos; y por último, (3) aportar una función de control, permitiendo evaluar el trabajo realizado en producción (el programa sirve como referencia para comparar con el resultado obtenido finalmente, por lo que es importante que el programa sea lo más realista posible).

En general, muchos de los problemas que aparecen en programación de operaciones son complejos. El origen de esta dificultad proviene de: (1) la complejidad de los cálculos: un problema de programación de la producción se caracteriza por su naturaleza combinatoria, que hace que la búsqueda de soluciones óptimas para problemas grandes sea una labor inabordable en tiempo razonable; (2) la naturaleza distribuida del problema: se trata, en efecto de repartir *n* tareas entre varios recursos, pudiendo existir dependencias entre ellos; (3) el número de restricciones: el problema está sometido a una serie de restricciones que pueden ser numerosas. Estas restricciones pueden tener naturaleza física (restricciones funcionales de una máquina-herramienta, de una tarea a ejecutar,...) o temporal (como la fecha de fin de un trabajo, duración de tratamiento de una operación o el tiempo de cambio de herramienta de una máquina,...).

En el entorno de fabricación, la programación de operaciones consiste en asignar recursos (máquinas, herramientas o personas), a los trabajos a realizar, fijando sus fechas de comienzo, de forma que se consiga un funcionamiento óptimo o suficientemente bueno del sistema de fabricación. Se trata del nivel más bajo de

planificación (planificación a muy corto plazo) dentro del proceso de planificación de la producción en la empresa. Puede verse como una fase de preparación de las actividades productivas, después de la planificación maestra y del cálculo de necesidades. El resultado es una asignación y un calendario.

En el caso más general la programación de operaciones requerirá de tres procesos (Companys 1989):

- Asignación o carga: de las operaciones a los centros de trabajo.
- Secuenciación: de las operaciones asignadas a un centro de trabajo estableciendo en qué orden van a ser ejecutadas.
- Temporización: determinación de los momentos de comienzo y de fin de cada operación.

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 (Pinedo 2009):

- Un conjunto de recursos: hace referencia a las máquinas, personas e instalaciones que permiten realizar una determinada actividad. Son los sistemas físicos o lógicos que son capaces de realizar las operaciones de proceso sobre los trabajos.
- Un conjunto de trabajos, compuestos por una serie de operaciones que necesitan de los recursos para poder ser procesados.
- Unas restricciones, que son un conjunto de condiciones a satisfacer. Las restricciones pueden hacer referencia a:
  - Los recursos: principalmente referidas a su configuración y su capacidad.
  - Las operaciones: las principales son restricciones de sucesión y de localización temporal.
    + *Restricciones de sucesión*: existe una restricción de sucesión entre dos tareas T1 y T2, si la operación T1 no puede comenzar antes de que finalice la operación T2. Se denota por "T1>T2".
    + *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 ($t_i>\alpha$), superiormente ($t_i<\beta$) o las dos a la vez ($\alpha<t_i<\beta$) donde α y β son las fechas límites.

- Una función objetivo: es la función a evaluar y cuyo valor se quiere optimizar. Los criterios generales más frecuentemente utilizados son la minimización de la duración total de fabricación, de los tiempos de inactividad de las máquinas, del coste de la programación de la producción (p.e. el número de productos en curso puede llevar a costes de almacenamiento suplementarios) o incluso de los retrasos incurridos. En ciertos casos particulares, pueden ser necesarios criterios más específicos como, por ejemplo, la minimización del efecto de cuello de botella producido por un recurso.

Se puede encontrar una descripción detallada del problema, de su desarrollo y de las principales técnicas de resolución en Brucker (2007), Blazewicz et al (2007) y Pinedo (2008, 2009).

1.2. Configuraciones del entorno de fabricación y su influencia en la programación

Dentro del ámbito de la fabricación, el abanico de configuraciones distintas que se puede encontrar es muy amplio. Puede ir desde un caso tan simple como secuenciar un conjunto de tareas sobre una máquina determinada, hasta entornos más complejos como el *job shop* donde varias tareas relacionadas mediante precedencias han de ser programadas en distintas máquinas y donde pueden existir rutas alternativas. Entre esos extremos se sitúan problemas con una cierta complejidad donde existen varias máquinas iguales trabajando en paralelo, o casos algo más complicados como el que resulta de programar un conjunto de trabajos sobre un sistema configurado en varias etapas, donde todas las tareas deben ser procesadas en las mismas máquinas y con una secuencia igual (*flow shop*).

Las configuraciones productivas están asociadas al tipo de producto que se produce, a su variedad y al volumen de producción. De acuerdo con Cuatrecasas (2009), existe una relación directa entre el tipo de producto y el proceso donde se fabrica. Los principales tipos de proceso son: el funcional por lotes y su variante por puestos fijos, por una parte, y la producción en flujo o cadena y su variante en flujo continuo, por otra. La otra variable a analizar es el volumen producido y la variedad. Se puede distinguir entre producción de volúmenes muy bajos y variedad muy elevada, o de productos estandarizados producidos en grandes volúmenes.

Es posible realizar una matriz combinando las dimensiones de proceso y volumen de producción, que es la llamada matriz producto-proceso (Figura 1). En esta matriz, inicialmente desarrollada por Hayes y Wheelwright (1979), la combinación producto y proceso da lugar a una diagonal.

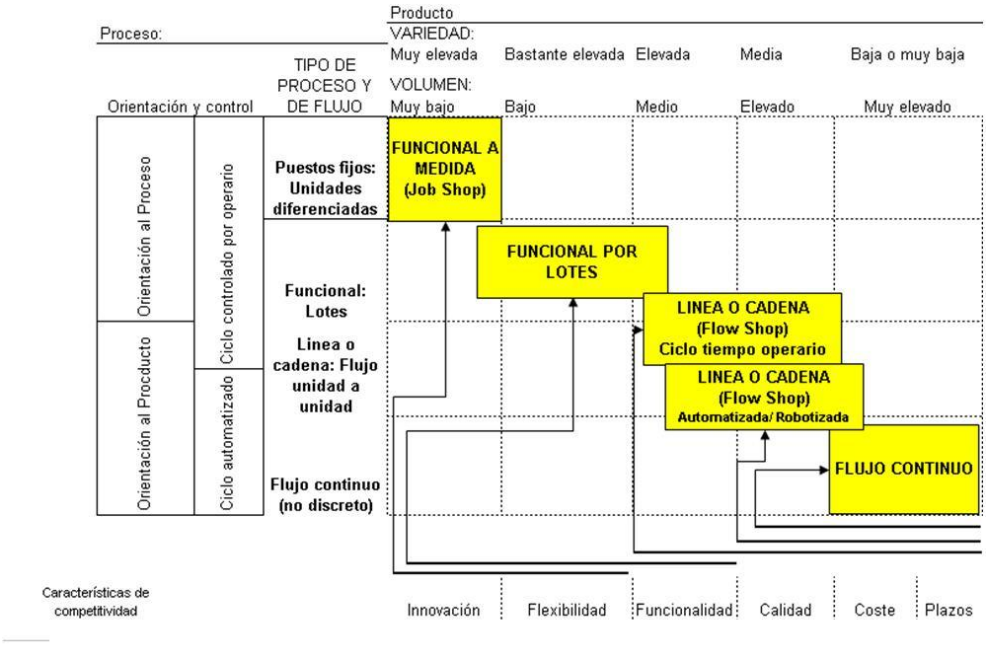

Figura 1. Matriz producto-proceso convencional (Cuatrecasas 2009)

Dentro de la matriz producto-proceso convencional, se define los siguientes tipos de producción (Cuatrecasas 2009):

- Producción en flujo continuo: en esta modalidad cada máquina y equipo están diseñados para realizar siempre la misma operación. Suelen ser procesos automatizados donde el ítem que procesa una máquina es suministrado por la máquina precedente. Se pretende obtener un gran volumen de producción, de una gran calidad y a un coste muy bajo. Por el contrario, la variedad de los productos es pequeña, y el diseño es muy estandarizado. En esta configuración la programación de operaciones se limita a la determinación del tiempo de funcionamiento de la línea.
- Producción en flujo o cadena (flow shop): es un sistema de producción orientado a producto. Esta modalidad se adopta cuando se deben fabricar lotes grandes de pocos productos diferentes, pero técnicamente homogéneos, usando para ello las mismas instalaciones. Se trata de productos cuyo proceso de

obtención en el centro de trabajo requiere una secuencia similar de operaciones, aunque algunos de ellos puedan necesitar alguna operación diferente. Los puestos de trabajo y sus máquinas y equipos se disponen siguiendo el flujo de operaciones que sigue el producto. Los equipos suelen estar especializados para el tipo de operación y de producto y se hallan dedicados por completo a los mismos. En esta configuración, la programación de operaciones consiste en determinar la secuencia de entrada a la cadena.

- Producción funcional por lotes: en este caso el producto ya no es a medida, sino que se dispone de una gran variedad de versiones entre las que ha de elegir el consumidor. Los lotes suelen ser de un volumen mayor, incluso muy grandes. La distribución en planta está orientada al proceso: recorridos diversos y largos para el producto, muchas actividades de manipulación y transporte, altas esperas de productos en colas y volúmenes importantes de stock. Esto hace que aparezcan muchas actividades sin valor añadido (tiempos largos de proceso, stock…). En este caso la programación de operaciones abarca los procesos de asignación o carga (si existe la posibilidad de realizar la tarea en más de una máquina), secuenciación y temporización.
- Producción funcional a medida (*job shop*): en este tipo de configuración se producen lotes más o menos pequeños de una amplia variedad de productos de poca o nula estandarización. Los equipos de producción estarán poco especializados. La distribución de planta suele estar orientada al proceso, es decir, los equipos suelen agruparse en talleres o centros de trabajo según la función que desarrollan. Suelen ser versátiles y permiten ejecutar operaciones diversas. Es habitual que exista una cartera de pedidos pendientes (con el consecuente alargamiento de los plazos de entrega) y gran cantidad de stocks de materiales y trabajos en curso. Este tipo de producción acarrea fuertes desequilibrios, se crean cuellos de botella en determinados puestos de trabajo y se acumula mucho material en las colas internas. Al igual que en el caso anterior, la programación de operaciones abarca la asignación, secuenciación y temporización.

En la actualidad, las nuevas tendencias en gestión de los sistemas productivos en el marco de la producción ajustada han hecho que se rompa el principio de la diagonal. Nuevos modelos pretenden abarcar eficientemente las características de competitividad: calidad, tiempo y coste, de los tipos de producción en línea; y la flexibilidad, la funcionalidad y alta variación de producto, características propias de los entornos enfocados a proceso. Estos nuevos tipos de gestión se encuentran fuera de

la diagonal que aparecía en la matriz producto-proceso original (Figura 2). Algunos de estos nuevos conceptos son:

- La producción *lean* o ajustada: ha aportado nuevas y avanzadas formas de gestionar los sistemas productivos tratando de alcanzar dos objetivos de forma simultánea. Por una parte, persigue reducir al máximo el empleo de recursos y el número de actividades necesarias para realizar la producción, utilizando procesos con orientación al producto (zona inferior de la matriz producto-proceso). Por otra, busca lotes de producción pequeños y elevada variación de producto para alcanzar una mayor flexibilidad (zona izquierda de la matriz producto-proceso) (Womack et al 2007).
- Sistemas de fabricación flexible (FMS - *flexible manufacturing systems*): se trata de sistemas altamente automatizados, que permiten alcanzar altos niveles de utilización y eficiencia, manteniendo la flexibilidad de los equipos poco especializados. Por sus bajos tiempos de preparación estos sistemas son adecuados para producciones de pequeños lotes, que permiten cambiar rápidamente a distintas variantes del producto. La configuración de estos sistemas va desde la más cercana a las cadenas de producción (pero incorporando fabricación y no sólo de montaje), a las configuraciones de taller (Vollmann et al 1997).

Dada la flexibilidad de estas nuevas configuraciones, se utilizan los tres procesos de programación de operaciones: asignación, secuenciación y temporización.

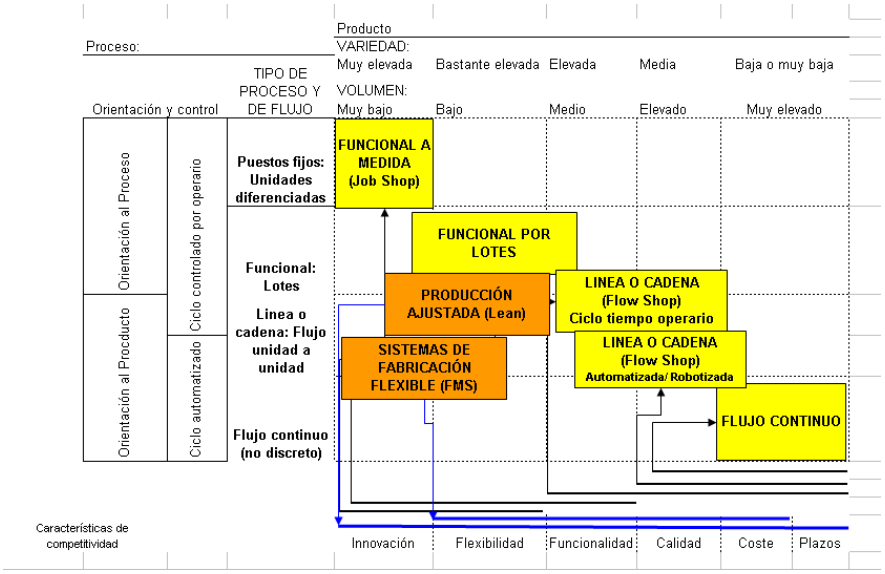

Figura 2. Matriz producto-proceso completa (Cuatrecasas 2009)

1.3. Clasificación de los problemas de scheduling: notación

Para facilitar el estudio y caracterización de los problemas de programación de operaciones, Graham et al (1979) realizaron una propuesta de notación que ha sido adoptada por numerosos autores. En este trabajo se sigue una adaptación de esta notación propuesta por Brucker (2007), Blazewicz et al (2007) y Pinedo (2009).

En un problema de producción se dispone de un número finito *m* de máquinas (recursos) en los que se han de realizar un conjunto *n* de trabajos. Normalmente se utiliza el subíndice *h* para hacer referencia a una máquina mientras que el subíndice *i* hace referencia a un trabajo (J_i). Cada trabajo consiste en una serie de operaciones (tareas). El subíndice conjunto *ij* hace referencia a la operación j del trabajo i (O_{ij}).

Cada trabajo *i* se describe por una serie de parámetros:

- Fecha de entrega comprometida (D_i): es la fecha comprometida con el cliente. La orden de trabajo puede ser completada posteriormente pero incurriendo en una penalización por retraso.
- Momento de lanzamiento (r_i): es el momento a partir del cual el trabajo es lanzado a planta y sus operaciones pueden ser realizadas.
- Peso (w_i): es un factor de prioridad que indica la importancia del trabajo respecto a los demás.

Un problema de producción se caracteriza mediante un triplete $\alpha/\beta/\gamma$, donde:

- α se refiere 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 el objetivo a minimizar, generalmente una entrada.

- *Configuración del taller (α):*

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

- {o, P, Q, R} Cada trabajo consiste en una sola operación que puede ser realizado en una o varias máquinas:
  - Una máquina (o): cada trabajo se realiza en una sola máquina.

- Varias máquinas en paralelo, que pueden ser:
  + Idénticas (Pm): cada trabajo puede ser realizado en cualquiera de las *m* máquinas.
  + Uniformes (Qm): hay *m* máquinas con diferentes velocidades. La velocidad de cada máquina es v_h. El tiempo que una máquina i tarda en realizar una operación se calcula por p_i/v_h
  + No relacionadas (Rm): hay *m* máquinas trabajando en paralelo. La velocidad de la máquina *h* para el trabajo i es v_{ih}. El tiempo que una máquina *h* tarda en realizar una operación se calcula por p_i/v_{ih}.

❖ {G, X, O, J, F}. En estos casos, cada trabajo i está formado por un conjunto de operaciones $\{ij\}$. Las operaciones están vinculadas mediante relaciones de precedencia. Las máquinas son dedicadas, es decir, especializadas en ciertas operaciones. En base a estas características se pueden dar los siguiente casos:

- Modelo general (G): posee todas las características citadas anteriormente, el resto son casos particulares de éste.
- *Flow shop* (Fm): 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 tomará el valor *prmu*.
- *Flow shop flexible* (FFc): es la generalización del *flow shop* con máquinas paralelas. En lugar de haber *m* máquinas existen *c* etapas en cada una de las cuales hay varias máquinas funcionando en paralelo. Cada trabajo ha de pasar por cada una de estas etapas siguiendo siempre el mismo orden. En la literatura este tipo problema también se conoce como *hibrid flow shop* y como *multiprocessor flow shop*.
- *Job shop* (Jm): en este caso se dispone de *m* máquinas dedicadas. Cada trabajo sigue una determinada ruta. Se puede distinguir entre los problemas en los que los trabajos 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 indicará que se trata de un problema con recirculación *rcrc*.

- *Job shop flexible* (FJc): 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*.
- *Open shop* (Om): se dispone de *m* máquinas, cada trabajo tiene que pasar por todas las máquinas, aunque 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 teniendo en cuenta que cada trabajo puede tener una ruta diferente.

- Valores del parámetro β:

Este parámetro define las características de los recursos y de los trabajos. Hace referencia a las posibles restricciones y condiciones que pueden aparecer. Algunos de los posibles valores que puede tomar son:

- Fechas de lanzamiento (r_i): si este símbolo aparece, un trabajo j no podrá empezar a realizarse antes de r_i.
- Restricciones 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 comenzadas (*non-premptive*).
- 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: (1) *Chain*, cuando una tarea puede tener como máximo una tarea precedente y una tarea sucesora, (2) *Intree*, cuando una tarea puede tener más de una tarea precedente, (3) *Outtree*, cuando una tarea puede tener varias sucesoras.
- Tiempos de preparación dependientes de la secuencia (i_{jk})*:* indica que el tiempo de preparación de una tarea depende de la tarea que ha sido programada previamente. En el caso de que este tiempo de preparación sea independiente de la secuencia, se incluirá en el tiempo de proceso de la tarea.
- 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 pueden ser diferentes. 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 sólo 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* 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.
- Recirculación (*rcrc*): indica que un trabajo puede pasar por una misma máquina más de una vez.

- 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 lo representamos por C_i:

$$C_i = \max_j(c_{ij}) \tag{1.1}$$

donde c_{ij} es el tiempo de finalización de cada tarea j

Es posible definir distintas variables que dependen del tiempo de finalización de los trabajos:

- Huelgo (*Lateness*): es la diferencia entre el tiempo de finalización del trabajo y la fecha de entrega comprometida. Puede tener un valor positivo o negativo.

$$L_i = C_i - D_i \tag{1.2}$$

- Retraso: la diferencia positiva entre el tiempo de finalización del trabajo y la fecha de entrega comprometida.

$$T_i = \max(0, C_i - D_i) = max(0,\ L_i) \tag{1.3}$$

- Adelanto: la diferencia positiva entre la fecha de entrega comprometida y el tiempo de finalización del trabajo.

$$E_i = \max(0, D_i - C_i) \tag{1.4}$$

- Penalización unitaria

$$L_i = \begin{cases} 1 & si\ C_i - D_i > 0 \\ 0 & en\ otro\ caso \end{cases} \tag{1.5}$$

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 crece cualquiera de los C_i. Las principales son:

- *Makespan* (C_{max}): tiempo de finalización de la última operación en ser acabada.
- Máximo huelgo (L_{max}): máxima separación de la finalización de los trabajos respecto su fecha de entrega.
- Suma ponderada de los tiempos de finalización o *weighted flow time* (WFT).

$$WFT = \sum w_i C_i \tag{1.6}$$

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

$$WT = \sum w_i T_i \tag{1.7}$$

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

$$WST = \sum w_i T_i^2 \tag{1.8}$$

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

$$NTJ = \sum w_i U_i \tag{1.9}$$

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 distintos.

$$TWET = \sum w_i' E_i + \sum w_i'' T_i \tag{1.10}$$

1.4. Definición del problema de programación de talleres

El problema de programación de talleres, o problema *job shop*, es un caso particular del problema general de programación al cual se añaden una serie de hipótesis restrictivas. Se busca la gestión óptima de recursos típica del ámbito de la gestión de la producción (Blazewicz et al 2007).

En el caso del problema de programación de talleres, las *tareas* son las operaciones elementales de fabricación (ensamblaje, torneado, amolado...) y los *recursos* son las máquinas donde se efectúan estas operaciones. Un trabajo (*job*) corresponde entonces a una lista de operaciones elementales (operaciones de fabricación) necesaria para su realización. Es frecuente la asociación tarea con etapa de fabricación de un producto, y de trabajo con producto (o estructura del producto). Una máquina de producción se considera un recurso renovable ya que una vez utilizada puede estar disponible de nuevo. Por el contrario, las materias primas, por ejemplo, se denominan recursos consumibles.

Los problemas de programación de talleres precisan igualmente de las restricciones presentadas en la sección anterior:

- Restricción de precedencia: restricción de sucesión entre dos tareas de un mismo trabajo.

- Restricción de localización temporal: se puede definir un intervalo temporal (τ, ω) en el que la tarea debe ser programada. La definición de las fechas τ o β provienen de las dificultades prácticas encontradas en las situaciones reales. Estas pueden provenir de los objetivos especificados en una orden de fabricación (n productos deben ser entregados antes de la fecha ω) o resultar de la organización del plan de carga de la máquina (la máquina i no está disponible antes de la fecha τ).
- Restricción de capacidad: una máquina no puede tratar más que una tarea a la vez.

Además, los problemas de programación de talleres presentan las características siguientes (Blazewicz et al 1996):

- Restricción de independencia de trabajos: no existe ninguna restricción de precedencia entre operaciones (tareas) de trabajos diferentes.
- Hipótesis de no interrupción de las tareas (*non preemption*): una vez comenzada, una tarea no puede ser interrumpida. No se podrá recurrir a una segmentación de la ejecución de la tarea.
- Restricción disyuntiva: un trabajo no puede ser efectuado más que por una sola máquina a la vez, la ejecución de las tareas de un mismo trabajo deben ser secuenciales y no paralelas.

Existen diversas formas de definir el problema general *job shop*, como son a través de grafos disjuntos, lenguajes de programación como el OPL o mediante la formulación matemática. En los siguientes apartados describiremos brevemente estas formas de representación.

1.4.1. Grafo disjunto

Esta representación se debe a Roy y Sussmann (1964). Consiste en un grafo donde cada nodo representa una operación. Existen dos nodos adicionales que representan el comienzo y el fin del programa de producción. Entre las tareas consecutivas de un mismo trabajo existe un arco directo. El nodo inicio del grafo representa el "inicio de tareas", es una operación ficticia de duración cero. El nodo final representa "finalización de todos los trabajos" e igualmente es una operación ficticia de duración cero.

Para cada par de operaciones que necesitan la misma máquina (h), existen dos arcos, que tienen direcciones opuestas (arco doble o arco disjunto). Por tanto, los arcos simples representan las relaciones de precedencia, mientras que los arcos dobles representan la limitación de capacidad de las máquinas. Cada arco ij tiene asociado un peso p_{ij} que indica la duración de la operación que representa el nodo i del que parte el arco (Figura 3).

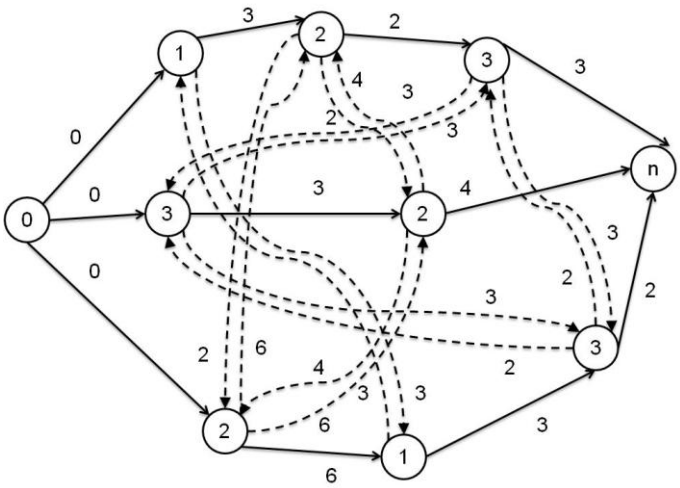

Figura 3. Gráfico disjunto representando un problema *job shop* (Blazewicz et al 1996)

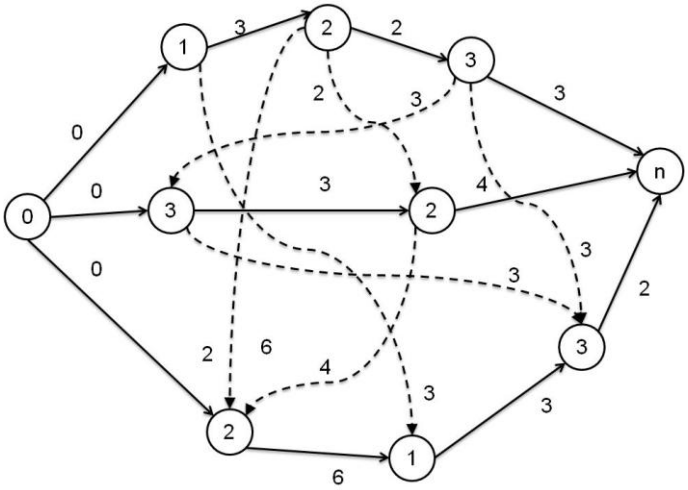

Figura 4. Representación de una solución factible del problema *job shop* (Blazewicz et al 1996)

El problema *job shop* consiste en encontrar el orden de programación de las tareas en cada máquina que optimiza la función objetivo. Utilizando esta representación, el problema consiste en seleccionar uno de cada par de arcos disjuntos, de modo que el grafo no sea cíclico (Figura 4) y la suma de los pesos del camino más largo sea mínima. Un grafo cíclico representa un programa no factible, porque en tal caso existiría un conflicto de precedencia entre las operaciones.

1.4.2. Lenguaje OPL

En la última década del siglo XX se desarrolló el *Optimization Programming Language* (OPL) para resolver problemas de optimización mediante una combinación de técnicas de programación de restricciones y programación matemática (Pinedo 2008).

En el siguiente ejemplo se muestra la definición del problema $Jm \parallel Cmax$ en el lenguaje OPL:

```
1   int nbMachines = . . .;
2   range Machines 1..nbMachines;
3   int nbJobs = . . .;
4   range Jobs 1..nbJobs;
5   int nbOperations = . . .;
6   range Operations 1..nbOperations;
7   Machines resource[Jobs,Operations]= . . .;
8   int+ duration[Jobs,Operations] = . . . ;
9   int totalDuration=sum(j in Jobs, o in Operations)duration[j,o];
10  scheduleHorizon=totalDuration;
11  Activity operation[j in Jobs, o in Operations](duration[j,o]);
12  Activity makespan(0);
13  UnaryResource tool[Machines];
14  Minimize
15  Makespan.end
16  subject to {
17  forall(j in Jobs)
18  Operation[j,nbOperations] precedes makespan;
19  forall(j in Jobs)
```

```
20          forall(o in 1,..nbOperations-1)
21          Operation[j,o] precedes Operation[j,o+1];
22          forall(j in Jobs)
23          forall(o in Operations)
24          Operation[j,o] requires tool[resource[j,o]];
25          };
```

1.4.3. Formulación matemática del problema de programación de talleres

Como se mencionó al comienzo del apartado 1.4, el problema de la programación de talleres (*job shop)* es un caso particular del problema general de programación al cual se le añaden una serie de hipótesis restrictivas (Blazewicz et al 2007). Este problema puede ser formulado como un problema de optimización sujeto a restricciones mediante programación entera mixta o mediante programación entera. A continuación se muestran ambas opciones:

- Problema *job shop* definido mediante programación entera mixta

Se busca secuenciar las operaciones de forma que cumplan los requisitos de precedencia y las restricciones de capacidad. Una de las primeras propuestas de debe a Manne (1960). En su caso utiliza el *makespan* como función objetivo. Utiliza la siguiente notación:

$\Im$	conjunto de trabajos J
I	conjunto de operaciones
M	conjunto de máquinas
A_j	Conjunto de pares de operaciones consecutivas $(J_r, J_{r+1}), 1 \leq r \leq \lvert J \rvert \; del \; trabajo \; J \in \Im$
B	conjunto de operaciones $i, j \in I, i \neq j$ que se realizan en la misma máquina $k, \; k \in M$
p_i	tiempo de proceso de la operación i
H	número positivo muy grande

y_{ij} $= \begin{cases} 1 & si\ i\ precede\ a\ j\ (no\ necesariamente\ de\ forma\ inmediata) \\ 0 & en\ otro\ caso \end{cases}$

para dos operaciones cualesquiera que se realicen sobre la misma máquina $i, j \in I$, $(i,j) \in B$

x_i tiempo de comienzo de la operación $i \in I$

x_τ tiempo de finalización de la actividad ficticia final (se sigue la definición del grafo disyuntivo

El problema queda definido como:

$$\min x_\tau \text{ , sujeto a:} \tag{1.11}$$

$$x_j - x_i \geq p_i \quad para\ todo\ (i,j) \in A_J, J \in \Im \tag{1.12}$$

$$x_j + H(1 - y_{ij}) - x_i \geq p_i \quad para\ todo\ (i,j) \in \mathcal{B} \tag{1.13}$$

$$x_i + Hy_{ij} - x_j \geq p_{ji} \quad para\ todo\ (i,j) \in \mathcal{B} \tag{1.14}$$

$$x_\tau - x_i - p_i \geq 0 \quad para\ todo\ i \in I \tag{1.15}$$

$$y_{ij} \in \{0,1\} \quad para\ todo (i,j) \in \mathcal{B} \tag{1.16}$$

$$x_i \geq 0 \quad para\ todo\ i \in I \tag{1.17}$$

$$x_\tau \geq 0 \tag{1.18}$$

donde (1.11) es la función objetivo; las restricciones (1.12) definen que cada operación solo puede empezar cuando ha terminado su operación precedente; las restricciones (1.13) y (1.14) definen las restricciones disyuntivas que no permiten realizar dos operaciones simultáneamente en una misma máquina; la restricción (1.15) hace que la operación ficticia final tenga lugar cuando hayan finalizado todas las anteriores. Las últimas tres restricciones sirven para fijar el dominio para cada variable.

Las restricciones (1.13) y (1.14) son equivalentes a la siguiente expresión disyuntiva:

$$x_j \geq x_i + p_i \quad \vee \quad x_i \geq x_j + p_j \qquad para\ todo\ (i,j) \in \mathcal{B} \tag{1.19}$$

- Problema *job shop* definido mediante programación entera

La definición anterior del problema *job shop* no resulta satisfactoria para nuestros objetivos, por lo que seguiremos la propuesta de Hoitomt et al (1993): el problema se representa como $J_m/r_j/\sum w_i T_i$ y puede ser caracterizado como sigue: existen *n* trabajos que deben ser procesados en *m* máquinas en un taller. Cada trabajo implica una serie de tareas que deben realizarse, donde cada tarea necesita utilizar una máquina determinada. Cada máquina tiene una capacidad limitada por lo que no pueden ser programadas en cada instante más operaciones de las que su capacidad permite. Cada operación sólo se puede realizar en una máquina y está caracterizada por su duración.

La formulación matemática se corresponde con un problema de optimización sujeto a restricciones. Para formularlo es necesario definir una función y una serie de restricciones. Las restricciones están relacionadas: (1) con las máquinas, son las que reflejan su capacidad limitada (restricciones de capacidad); (2) con las órdenes, las cuales reflejan (2.1.) la limitación debida a la fecha de lanzamiento de la orden de trabajo (restricciones de localización temporal); (2.2.) el orden en que deben ejecutarse las tareas de cada orden de trabajo (restricciones de precedencia); (2.3.) que dos tareas de una misma orden no pueden realizarse en paralelo (restricciones disyuntivas); (2.4.) que la realización de una determinada tarea no puede interrumpirse ni ser divida en distintos intervalos de tiempo (restricciones de continuidad).

Para formular el problema utilizaremos la siguiente notación:

i	índice de cada trabajo. i = 1,...,n donde n es el número total de trabajos
J	conjunto de trabajos, J = { J_i } i : 1,..., N
O_i	número de tareas del trabajo J_i
j	índice de cada tarea dentro de cada trabajo
O_{ij}	tarea i del trabajo j
ij	índice de la tarea i del trabajo j
$G(J_i)$	conjunto de operaciones que componen el trabajo J_i, $G(J_i)$= { O_{ij} } j:1,…, O_i
k	periodo de tiempo, k = 1,..., K donde K es el horizonte de programación
h	índice de cada máquina, h = 1,..., m donde m es el número total de máquinas
H	conjunto de máquinas, H = {h} h : 1,..., M
$F(O_{ij})$	conjunto de máquinas que puede realizar la tarea O_{ij}. En el caso *job shop* cada tarea solo se puede realizar en una máquina, $F(O_{ij}) = h_{ij}$
d_{ijh}	tiempo proceso de la tarea Oij en la máquina h

D_i fecha de entrega comprometida para el trabajo i

B_i fecha de comienzo mínima para el trabajo i

b_{ii} fecha inicio programada de la tarea O_{ij}

c_{ij} fecha finalización programada de la tarea O_{ij}

h_{ij} máquina donde se ha asignado la tarea ij. Esta máquina debe pertenecer al conjunto de máquinas $F(O_{ij})$ que pueden realizar la tarea ij

T_i retraso del trabajo i

$$= \begin{cases} 1 & si\ la\ tarea\ se\ programa\ en\ el\ recurso\ h\ en\ el\ instante\ k \\ 0 & en\ caso\ contrario \end{cases}$$

M_h capacidad de la máquina S_h

l_{ij} conjunto de tareas de las que es precedente la tarea Oij

A continuación se presentan tanto la función objetivo como las restricciones del problema.

1.4.3.1. Función objetivo

La función objetivo hace referencia al criterio utilizado para medir la calidad de una solución del problema planteado. Los métodos de programación de la producción buscan soluciones óptimas o satisfactorias bajo este criterio. Las funciones objetivo más utilizadas son la minimización de la duración total de fabricación, los tiempos de inactividad de las máquinas, el coste de la programación de la producción (p.ej. el número de productos en curso puede llevar a costes de almacenamiento suplementarios) o incluso respecto las fechas de finalización de las tareas a conseguir. En ciertos casos particulares pueden necesitarse criterios más específicos como, por ejemplo, la minimización del efecto de cuello de botella producido por un recurso. En el punto 1.2 se enumeraron las principales.

A pesar de que los algoritmos tradicionales se han centrado en la minimización del *makespan*, o fecha de finalización de todos los trabajos, cada vez es más frecuente el estudio de problemas de programación de tareas relacionados en sus objetivos con la fecha de entrega debido a que, en la actualidad, la entrega a tiempo es un factor crítico a la hora de conseguir la satisfacción del cliente y de mantener la reputación de la empresa (Jayamohan y Rajendran 2004). Nosotros seguiremos esa tendencia y estudiaremos el problema cuya función objetivo depende del retraso.

$$\min_{\delta} F(T_i) \tag{1.20}$$

donde $F(T_i)$ puede tomar los siguientes valores:

$$F(T_i) = \sum_{i=1}^{N} w_i T_i \tag{1.21}$$

, o

$$F(T_i) = \sum_{i=1}^{N} w_i T_i^2 \tag{1.22}$$

1.4.3.2. Restricciones de capacidad

Las restricciones de capacidad limitan el número de operaciones (i) que pueden ser programados en una máquina (h) en un instante determinado (k). Para el caso del problema *job shop* $M_h = 1$, es decir, que cada máquina no puede realizar más de una operación simultáneamente:

$$\sum_{i=1}^{N} \sum_{j=1}^{O_i} \delta_{ijhk} \leq 1 \qquad \begin{matrix} k = 1,2,\dots,K \\ h = 1,2,\dots,M \end{matrix} \tag{1.23}$$

1.4.3.3. Restricciones de localización temporal

Indican que la fecha de inicio de cualquier operación será posterior a la fecha de comienzo de la orden de trabajo:

$$b_{ij} \leq B_i \qquad \begin{matrix} i = 1,2,\dots,N \\ j = 1,2,\dots,O_i \end{matrix} \tag{1.24}$$

1.4.3.4. Restricciones de precedencia

Indican que la fecha de finalización de una determinada operación ij (c_{ij}) tiene que ser anterior a la fecha de inicio de las operaciones posteriores (b_{il}, con $l \in I_{ij}$):

$$c_{ij} \leq b_{il} \qquad \begin{matrix} i = 1,2,\dots,N \\ j = 1,2,\dots,O_i \\ l \in I_{ij} \end{matrix} \tag{1.25}$$

1.4.3.5. Restricciones disyuntivas

En cada instante no se puede realizar más de una tarea perteneciente a una misma orden de trabajo a la vez. La ejecución de las tareas de un mismo trabajo debe ser secuencial y no se permite la ejecución en paralelo de las tareas de una misma orden de trabajo:

$$\sum_{j=1}^{O_i} \delta_{\mathrm{ijhk}} \leq 1 \qquad \begin{matrix} i = 1,2,\dots,N \\ k = 1,2,\dots,K \\ h = \mathrm{h_{ij}} \end{matrix} \tag{1.26}$$

1.4.3.6. Restricciones de no interrupción de las tareas (non-preemption)

Las relaciones entre los instantes de comienzo b_i y finalización c_i de una operación i, si la operación se realiza en una máquina en la que se tarda en realizar un tiempo d_i, se muestra en la expresión:

$$\mathrm{c_{ij}} = \mathrm{b_{ij}} + \mathrm{d_{ijh}} - 1 \qquad \begin{matrix} i = 1,2,\dots,N \\ j = 1,2,\dots,O_i \\ h = \mathrm{h_{ij}} \end{matrix} \tag{1.27}$$

1.4.3.7. Relación entre variables del problema

Además existe un conjunto de restricciones que no son propias del problema, sino que se utilizan para poder establecer la relación entre las verdaderas variables, las δ_{ijhk} , y las variables que muestran el principio y el fin de cada tarea, b_{ij} y c_{ij}, que se utilizan para definir el programa de forma más simplificada:

$$\delta_{\mathrm{ijhk}} = \begin{cases} 1 & \text{si } \mathrm{k} \in [\mathrm{b_{ij}}, \mathrm{c_{ij}}] \wedge \mathrm{h} = \mathrm{h_{ij}} \\ 0 & \text{en caso contrario} \end{cases} \qquad \begin{matrix} i = 1,2,\dots,N \\ j = 1,2,\dots,O_i \\ k = 1,2,\dots,K \\ h = 1,2,\dots,M \end{matrix} \tag{1.28}$$

1.4.3.8. Formulación del problema

En resumen, el problema puede formularse como un programa en variables temporales discretas de la siguiente forma:

$$(P) \quad \min_{\delta} \sum_{i=1}^{n} w_i T_i$$

sujeto a:

$$T_i = \max\left(0, \max_j(c_{ij}) - D_i\right) \qquad \mathrm{i} = 1,2,\ldots.\mathrm{n}$$

$$\sum_{\mathrm{i}=1}^{\mathrm{n}} \sum_{\mathrm{j}=1}^{O_i} \delta_{\mathrm{ijhk}} \leq 1 \qquad k = 1,2,\ldots,K \quad h = 1,2,\ldots,m$$

$$b_{ij} \leq B_i \qquad i = 1,2,\ldots,n \quad j = 1,2,\ldots,O_i$$

$$c_{ij} \leq b_{il} \qquad i = 1,2,\ldots,n \quad j = 1,2,\ldots,O_i \quad l \in I_{ij}$$

$$\sum_{j=1}^{O_i} \delta_{\mathrm{ijhk}} \leq 1 \qquad i = 1,2,\ldots,N \quad k = 1,2,\ldots,K \quad h = \mathrm{h_{ij}}$$

$$\mathrm{c_{ij}} = \mathrm{b_{ij}} + \mathrm{d_{ijh}} - 1 \qquad i = 1,2,\ldots,N \quad j = 1,2,\ldots,O_i \quad h = \mathrm{h_{ij}}$$

$$\delta_{\mathrm{ijhk}} = \begin{cases} 1 & \text{si } \mathrm{k} \in [\mathrm{b_{ij}}, \mathrm{c_{ij}}] \wedge \mathrm{h} = \mathrm{h_{ij}} \\ 0 & \text{en caso contrario} \end{cases} \qquad i = 1,2,\ldots,n \quad j = 1,2,\ldots,O_i \quad k = 1,2,\ldots,K \quad h = 1,2,\ldots,m$$

1.5. Problema de programación de talleres con enrutamiento flexible

De forma similar al problema clásico *job shop*, el problema *job shop* con enrutamiento flexible requiere la asignación óptima de cada trabajo a cada máquina con tiempos de comienzo y finalización conocidos. Sin embargo, este problema es más complejo que el clásico ya que necesita de una correcta selección de la máquina

entre un conjunto de máquinas que pueden procesar las operaciones de cada trabajo. Además, puede tener la opción de recirculación. Esta opción permite que un trabajo pueda realizar más de una operación sobre la misma máquina.

El problema *job shop* con enrutamiento flexible se puede definir a partir del problema *job shop* modificando alguna de sus restricciones:

1.5.1.1. Restricciones de capacidad

En el caso más general esta capacidad será un número real (M_h). Este sería un problema *job shop* flexible.

$$\sum_{\mathrm{i}=1}^{\mathrm{N}} \sum_{\mathrm{j}=1}^{O_i} \delta_{\mathrm{ijhk}} \leq M_{hk} \qquad \begin{matrix} k = 1,2,\dots,K \\ h = 1,2,\dots,M \end{matrix} \tag{1.29}$$

1.5.1.2. Restricciones de asignación de los recursos

El problema *job shop* flexible puede ser clasificado en dos categorías:

- Problema *job shop* flexible total (T-FJSP), donde cada operación puede ser realizada en cualquiera de las máquinas, es decir:

$$F(O_{i,j}) = H \tag{1.30}$$

- Problema *job shop* flexible parcial (P-FJSP), donde cada operación puede ser realizada en sólo un subconjunto de todas las máquinas, es decir: H:

$$F(O_{i,j}) \subset H \tag{1.31}$$

En general, se define esta restricción como:

$$F(O_{i,j}) \subseteq H \tag{1.32}$$

Si se permite la recirculación, un trabajo puede visitar la misma máquina en más de una ocasión. Formalmente, esto implica que (Ho & Tay 2005):

$$\exists\, i, j_1, j_2 : F\left(O_{i,j_1}\right) \cap F\left(O_{i,j_2}\right) \neq \emptyset \tag{1.33}$$

1.5.1.3. Formulación del problema

En resumen, el problema puede formularse como un programa en variables temporales discretas de forma similar a la formulación mostrada para el problema *job shop*. La diferencia en cuanto a su formulación respecto al problema *job shop* está en que, en éste último, sólo se puede realizar cada tarea $O_{i,j}$ en una sola máquina: $F(O_{i,j}) = h_{ij}$.

$$(P) \qquad \min_{\delta} \sum_{i=1}^{N} w_i T_i$$

sujeto a:

$$T_i = \max\left(0, \max_j(c_{ij}) - D_i\right) \qquad \mathrm{i} = 1,2,\ldots.\mathrm{N}$$

$$\sum_{\mathrm{i}=1}^{\mathrm{N}} \sum_{\mathrm{j}=1}^{O_i} \delta_{\mathrm{ijhk}} \leq M_{hk} \qquad \begin{matrix} k = 1,2,\ldots,K \\ h = 1,2,\ldots,M \end{matrix}$$

$$b_{ij} \leq B_i \qquad \begin{matrix} i = 1,2,\ldots,N \\ j = 1,2,\ldots,O_i \end{matrix}$$

$$c_{ij} \leq b_{il} \qquad \begin{matrix} i = 1,2,\ldots,N \\ j = 1,2,\ldots,O_i \\ l \in I_{ij} \end{matrix}$$

$$\sum_{j=1}^{O_i} \delta_{\mathrm{ijhk}} \qquad \begin{matrix} i = 1,2,\ldots,N \\ k = 1,2,\ldots,K \\ h = \mathrm{h_{ij}} \end{matrix}$$

$$\mathrm{c_{ij}} = \mathrm{b_{ij}} + \mathrm{d_{ijh}} - 1 \qquad \begin{matrix} i = 1,2,\ldots,N \\ j = 1,2,\ldots,O_i \\ h = \mathrm{h_{ij}} \end{matrix}$$

$$h_{ij} \in F(O_{ij}) \qquad F(O_{ij}) \subseteq H$$

$$\exists\, i, j_1, j_2 : F\left(O_{i,j_1}\right) \cap F\left(O_{i,j_2}\right) \neq \emptyset$$

$$\delta_{\mathrm{ijhk}} = \begin{matrix} 1 & \text{si } \mathrm{k} \in \left[\mathrm{b_{ij}}, \mathrm{c_{ij}}\right] \wedge \mathrm{h} = \mathrm{h_{ij}} \\ 0 & \text{en caso contrario} \end{matrix} \qquad \begin{matrix} i = 1,2,\ldots,N \\ j = 1,2,\ldots,O_i \\ k = 1,2,\ldots,K \\ h = 1,2,\ldots,M \end{matrix}$$

1.6. Complejidad del problema

La teoría de la complejidad computacional es la rama de la teoría de la computación que estudia los recursos requeridos durante el cómputo de un algoritmo destinado a resolver un problema, proporcionando un marco teórico que permite clasificar los problemas en fáciles o difíciles. Esta teoría fue desarrollada inicialmente para el estudio de los problemas algorítmicos y su aplicación ha sido de utilidad en optimización combinatoria (Pinedo 2008).

Para poder caracterizar un problema por su complejidad se hace necesaria la definición de las clases de complejidad. Las clases de complejidad son el conjunto de problemas que necesitan de unos recursos equivalentes para resolverlos. En este punto se definen las bases de la teoría de la complejidad computacional y las principales clases de complejidad, para poder aplicarlo a los problemas de programación de operaciones.

1.6.1. Teoría de la complejidad computacional

La teoría de la complejidad computacional trata de medir el rendimiento de los algoritmos respecto de su tiempo de computación (Brucker 2007). La complejidad computacional estima la cantidad de recursos que se necesita para resolver diferentes problemas. Determinar la complejidad de un problema no lo resuelve directamente, sin embargo va a permitir decidir qué tipo de método es más adecuado para su resolución. Por ejemplo, no vale la pena aplicar un método exacto de resolución si sabemos que el problema es complejo (Tovey 2002).

Uno de los problemas donde la teoría de complejidad tiene más aplicación es en la optimización combinatoria. La optimización combinatoria u optimización discreta, hace referencia al estudio de los problemas de optimización donde el espacio de soluciones factibles es discreto, y cuya meta es encontrar la mejor solución.

Se puede definir una instancia de un problema de optimización combinatoria como un par (S, f), donde S es el conjunto de soluciones factibles y $f: S \rightarrow \mathbb{R}$ es la función de coste. El problema consiste en encontrar una solución $s^* \in S$ tal que (Papadimitriou and Steiglitz, 1998):

$$f(s^*) \leq f(s) \;\; \forall s \in S \tag{1.34}$$

Es habitual que el espacio de soluciones S esté expresado en forma implícita, es decir, no se dispone directamente de una lista de soluciones con sus costes respectivos.

Un algoritmo es cualquier procedimiento para resolver un problema (Blazewicz et al 2007). Se dice que un algoritmo es exacto para un problema dado si devuelve una solución factible óptima. Una algoritmo trivial es aquel que considera todas y cada una de las soluciones del espacio de soluciones factibles y escoge la solución con coste mínimo. Sin embargo, en este tipo de problemas es habitual que la enumeración de todas las soluciones posibles sea imposible en la práctica, ya que el espacio de soluciones factibles suele estar expresado de modo implícito y su tamaño crece exponencialmente con el tamaño del problema.

Veamos como ejemplo el *2-partition problem* (Korf 1998): dado un conjunto A de n números enteros, encontrar un subconjunto X tal que X y su complementario A\X estén lo más balanceado posible. Es decir, el problema consiste en calcular $\min c(X)$ con $c(X) = \left|\sum_{a \in X} a\right| - \left|\sum_{a \in \mathrm{A} \setminus \mathrm{X}} a\right|$. Se puede ver que, aunque se parte de una lista de n números, existen 2^n soluciones factibles, por lo que el algoritmo de búsqueda exhaustiva es demasiado costoso incluso para valores pequeños de n.

1.6.1.1. Máquina de Turing

En el análisis computacional se requiere un modelo formal de la computadora que va a realizar los cálculos. El objetivo es estudiar la complejidad del problema independientemente de la máquina que lo resuelva. La máquina de Turing es una máquina de cálculo teórica que sirve como modelo idealizado del cálculo matemático. Es una materialización de la idea de que la computación consiste en aplicar reglas mecánicas para manipular números, donde la persona-máquina que manipula dispone de algún medio que le permite memorizar los resultados intermedios (Arora y Barak 2009).

En la máquina de Turing se realiza una acción (lectura o escritura) a partir del estado actual y de las variables de entrada. Utiliza una función de transición que, a partir de un estado inicial y una cadena de caracteres en una cinta, va leyendo una celda de la cinta, borrando el símbolo, escribe el nuevo símbolo y finalmente avanza a la izquierda o a la derecha (sólo una celda a la vez). El procedimiento se formula en forma de etapas muy simples del tipo: *"si se encuentra en el estado 30 y el símbolo leído es '0', entonces remplazar este símbolo por un '1', pasar al estado 14, y mover a*

una celda adyacente (a derecha o izquierda)". Este proceso se repite hasta que de detiene en un estado final o de aceptación, representando así la salida.

Se distingue entre máquina de Turing determinista y no determinista. En la máquina de Turing determinista existe una única acción posible que la computadora puede tomar, dados el estado actual y las variables de entrada. Por el contrario, en la máquina de Turing no determinista, puede existir más de una posible combinación de actuaciones. Podemos entender la máquina de Turing no determinista como varias máquinas de Turing en paralelo, funcionando de forma simultánea pero sin posibilidad de comunicarse entre sí. Esto permite resolver problemas de complejidad exponencial en tiempo polinómico (Papadimitriou y Steiglitz 1998).

1.6.1.2. Notación

Para formalizar el concepto de complejidad se modelarán aspectos que intervienen en la resolución de los problemas como el tiempo de procesamiento o el espacio de almacenamiento necesario en máquinas de computación estándares como la máquina de Turing.

Se define la complejidad de un algoritmo como el número de pasos básicos que tendría que realizar el algoritmo en función del tamaño de la entrada. Un paso básico puede ser una comparación, un cálculo o cualquier manipulación de datos. De forma simplificada, podemos representar un paso básico como una operación lo suficientemente sencilla para que pueda ser realizada de forma exacta y en tiempo finito por un hombre usando lápiz y papel. Para cada input se define el tamaño del input *n*. La eficiencia del algoritmo se mide como una cota superior del número de pasos que utiliza el algoritmo con un input de tamaño *n*.

Se define la complejidad de un problema como la complejidad del algoritmo más eficiente que lo resuelve. Como no es necesario un valor exacto de esa cota se utiliza la cota superior asintótica. Ésta tiene gran importancia en la teoría de la complejidad computacional a la hora de definir las clases de complejidad. La notación utilizada es la llamada notación de asintótica O mayúscula O() (Brassard y Bratley 1996). Es una función f(n) que sirve de cota superior a otra función h(n) cuando el argumento tiende a infinito. Se utiliza la notación O(f(n)) para hacer referencia a las funciones acotadas superiormente por f(n). Por ejemplo, en lugar de decir que la complejidad del algoritmo está acotada por $T(n) = n^4 + 9n + 2$, decimos que es $O(n^4)$ (Brucker 2007).

Una definición precisa de la notación O(), cota de tiempo de procesamiento, es que un algoritmo tiene una cota de tiempo O(f(n)) si existe constantes N y k tales que para cada entrada de tamaño $n \geq N$ el algoritmo no requiere un tiempo de más de $k \cdot |f(n)|$ (Tovey 2002). Gráficamente se puede ver en la Figura 5.

$$O(f(n)) = \begin{cases} h(n): existen\ \ N, & K\ tales\ que \\ \forall n \geq N: 0 \leq |h(n)| \leq k \cdot |f(n)| \end{cases} \tag{1.35}$$

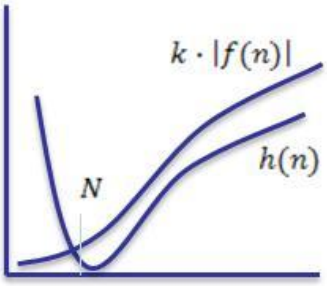

Figura 5. Cota superior asintótica de una función h(n)

Generalmente lo que interesa distinguir es si los algoritmos necesitan un tiempo de procesamiento polinómico o no. Un algoritmo polinómico es aquel cuya función de complejidad es $O(p(n))$, donde p es algún polinomio y n es el tamaño del input. Se dice que un algoritmo es eficiente si es polinómico (Brassard y Bratley 1996). Los algoritmos para los que no es posible acotar su complejidad de esta forma se denominan algoritmos exponenciales. Cualquier función exponencial es mayor que una función polinómica si n es suficientemente grande.

Un problema es resoluble en tiempo polinómico si existe un polinomio p(n) tal que $T(n) \in O(p(n))$ para cualquier input de tamaño n posible del problema. (Brucker 2007). En general, sólo es factible buscar una solución para los problemas resolubles en tiempo polinómico. Para un problema que, por ejemplo, requiera 2^n operaciones para su resolución; para una fuente de información relativamente pequeña, n=100; y asumiendo una computadora de alta capacidad de cálculo, se tardaría mucho más tiempo que la actual edad del universo para resolver el problema.

1.6.1.3. Problemas de optimización- Problemas de decisión

La teoría de la complejidad computacional tiene importantes aplicaciones en problemas de optimización combinatoria o discreta. Cuando en estos problemas se

concretan los parámetros que los definen con valores determinados, el resultado es una instancia. Un problema puede ser visto como la generalización de muchas instancias.

Podemos distinguir dos grupos de problemas: de decisión y de optimización (Blazewicz et al 2007):

- Los problemas de decisión son aquellos para los que una solución a una instancia del problema puede tomar sólo dos valores: sí o no.
- 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.

Todo problema de optimización se puede asociar a un problema de decisión. Por ejemplo: cualquier problema de minimización es posible plantearlo como problema de decisión en el que la respuesta es si existe una solución con el valor de la función objetivo f menor o igual a algún valor determinado k. La pregunta a formular sería: ¿existe una solución factible S tal que $f(S) \leq k$? (Brucker 2007). Por ejemplo, en un problema *job shop* del tipo Jm//Cmax en el que debe ser minimizada la fecha de finalización de la última tarea en ser terminada (*makespan*), en el problema de decisión asociado se cuestiona si existe un programa de producción cuyo *makespan* es menor de un valor determinado *z*.

Una propiedad importante es que si tenemos un problema de decisión asociado a un problema de optimización, si el problema de decisión es difícil, entonces el problema de optimización será también difícil, ya que un problema de decisión no puede ser más complicado que el problema de optimización del que deriva. Esto es debido a que si un problema de optimización puede ser resuelto en tiempo polinómico, también será posible resolver el correspondiente problema de optimización, ya que tan solo hay que comparar el valor óptimo de la función objetivo con una constante k (operación que no aumenta la complejidad del problema) (Blazewicz et al 2007).

1.6.1.4. Reducción

Uno de los conceptos fundamentales en teoría de complejidad computacional es el de reducción de un problema. Se dice que un problema P es reducible a un problema Q si existe una función polinómica que transforma P en Q. De modo más formal: se define la transformación polinómica del problema P en el problema Q y se representa por $P \propto Q$, si una función f asigna el conjunto de todas las instancias de P en el conjunto de instancias de Q, satisfaciendo las siguientes condiciones (Blazewicz et al 2007):

- Para cada instancia I de P la respuesta es "sí" si y solo si para f(I) de Q es siempre "sí".
- f es computable en tiempo polinómico (dependiendo del tamaño de la instancia I) por una máquina de Turing determinista.

También se puede definir el concepto de reducción para dos problemas de decisión de la siguiente forma: para dos problemas de decisión P y Q, decimos que P es reducible a Q ($P \propto Q$) si existe una función g computable en tiempo polinómico que transforma inputs para P en inputs para Q de tal forma que x es un input "sí" para P si y solo si g(x) es un input "sí" para Q (Brucker 2007).

La reducción sirve para demostrar que un problema pertenece a una determinada clase de complejidad. La idea básica es que si se conoce que un problema P no es más complejo que otro problema Q, y se conoce que el problema P es complejo, entonces Q será complejo.

1.6.2. Clases de complejidad

1.6.2.1. Clase P

La clase P está formada por los problemas de decisión para los que existe al menos un algoritmo para una máquina de Turing determinista que resuelva el problema tal que el número de pasos necesario para resolverlo está acotado por un polinomio en *n*, donde *n* es la longitud de la entrada (Pinedo 2008). Es decir, está compuesto por los problemas que pueden ser resueltos por algún algoritmo polinómico (Brassard y Bratley 1996).

Sea un algoritmo h que transforma una entrada x en una salida h(x), que es la respuesta del problema resuelto por el algoritmo. Se dice que el algoritmo es polinómico si h(x) puede ser computado como máximo en O(p(|x|)), siendo p una función polinómica y |x| el tamaño de la entrada de una entrada.

Sean P y Q problemas de decisión y P es reducible a Q ($P \propto Q$), entonces si $Q \in P \Rightarrow P \in P$ y de igual modo si $P \notin P \Rightarrow Q \notin P$

1.6.2.2. Clase NP

En teoría de la complejidad computacional, NP es el acrónimo en inglés de Polinómico no determinista (Non-Deterministic Polynomial-time). Es el conjunto de problemas que pueden ser resueltos en tiempo polinómico por una máquina de Turing no determinista.

Una definición equivalente a la anterior es: la clase NP contiene todos los problemas de decisión para los que existe un certificado con el que la respuesta correcta puede ser verificada por una máquina de Turing en un número de pasos acotado por un polinomio en función del tamaño de la entrada (Pinedo 2008). Se llama certificado a la información adicional que nos permite probar un problema NP en tiempo polinómico. Podemos decir que es el conjunto de problemas de decisión para los que, dada una posible solución del problema, podemos determinar en tiempo polinómico si la solución dada es correcta si conocemos un certificado.

1.6.2.3. Clase NP-complete

En teoría de la complejidad computacional, un problema pertenece a la clase *NP-Complete* si cualquier problema NP se puede reducir a él. Es decir: la clase de complejidad NP-completo es el subconjunto de los problemas de decisión en NP tal que todo problema en NP se puede reducir en cada uno de los problemas de *NP-complete*. Un problema de decisión Q es *NP-complete* si $Q \in NP$ y, para cualquier otro problema de decisión $P \in NP$, tenemos que $P \propto Q$. Es decir, que cualquier problema de NP no es más difícil que un problema de *NP-complete*. Por tanto, los problemas de NP-completo son los problemas más difíciles de NP.

Para probar que un determinado problema P pertenece a la clase *NP-completo*, es suficiente que un problema $Q \in NP$ pueda ser transformado polinomicamente en P, $P \propto Q$ (Blazewicz et al 2007).

Podemos decir que un problema es NP-completo si es a la vez NP (verificable de forma no determinista en tiempo polinómico) y NP-hard (cualquier problema NP puede ser transformado en ese problema).

Por ejemplo, el problema de la suma de subconjuntos (*subset sum problem*) es *NP-complete* (Garey y Johnson 1990). Este problema se enuncia como sigue: dado un conjunto S de enteros, ¿existe un subconjunto no vacío de S cuyos elementos sumen cero? Podemos verificar fácilmente si una respuesta es correcta, pero para obtener la

solución exacta no se conoce mejor solución que explorar todos los 2^n subconjuntos posibles hasta encontrar uno que cumpla con la condición.

1.6.2.4. Clase NP-hard

La clase *NP-hard* 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 tienen por qué ser problemas de decisión, también incluye problemas de optimización u otros.

Dado un problema P, que puede ser un problema de decisión o un problema de optimización, se dice que es NP-hard si toda la clase de problemas NP reduce polinómicamente a P (Pinedo 2008).

Un problema de optimización es NP-hard si puede ser transformado en un problema de decisión que sea NP-completo (Brucker 2007). Como ya se ha comentado, los problemas de decisión no son más complejos que los correspondientes problemas de optimización. Por tanto, para demostrar que un problema de optimización es NP-hard es suficiente con transformarlo polinómicamente en algún problema conocido de la clase NP-completo.

Por el contrario, para probar que un problema de optimización es fácil, es suficiente con construir un algoritmo polinómico que lo resuelva. Existen un conjunto de problemas para los que todavía no se ha demostrado que sean NP-completo, ni se ha encontrado un algoritmo que los resuelva polinómicamente (Blazewicz et al 2007).

1.6.2.5. Relación entre clases

La clase P es una subclase de NP, $P \subseteq NP$. Uno de los grandes problemas en ciencias de la computación consiste en responder al siguiente problema de decisión: ¿P=NP?, es la llamada conjetura P≠NP.

La Figura 6 muestra las relaciones entre casos en caso de que P≠NP. Es muy probable que los problemas de la clase *NP-completo* no formen parte de la clase de complejidad *P* ya que, en tal caso, si se tuviese una solución polinómica para algún problema de NP-completo, todos los problemas de NP tendrían también una solución en tiempo polinómico. Ésta es una de las cuestiones sin resolver en teoría de la complejidad, si NP=P.

La clase *NP-completo* juega un papel importante para resolver esta cuestión. Si se demuestra que un problema perteneciente a *NP-completo* puede ser resuelto en tiempo polinómico entonces P=NP. Es decir, que si un problema Q perteneciente a *NP-completo* pudiera ser resuelto en tiempo polinómico, entonces todos los problemas de NP podrían resolverse en tiempo polinómico, y por tanto tendríamos que P=NP. En ese caso existiría una gran cantidad de problemas que pueden ser resueltos en tiempo polinómico para los que todavía no se ha encontrado ningún algoritmo polinómico que los resuelva. Esto subraya la importancia conceptual de la clase NP-completo (Brucker 2007).

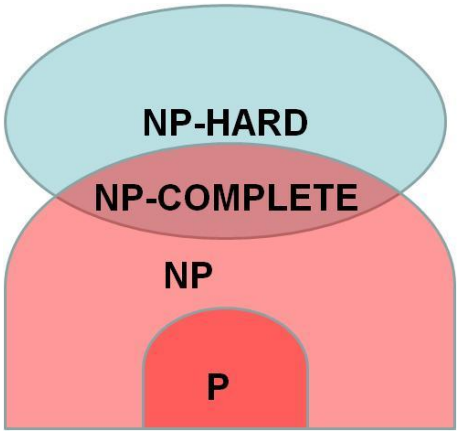

Figura 6. Clases de problemas si P≠NP (Tovey 2002)

Otra forma de dar una respuesta a la conjetura es demostrar que para un problema NP-completo no existe solución en tiempo polinómico, ya que entonces ninguno de los problemas NP tendría solución.

1.6.3. Complejidad de los problemas de scheduling

Los problemas de *scheduling* son problemas de optimización combinatoria, por lo que resulta fundamental el estudio de su complejidad a la hora de definir métodos que los resuelvan.

Un problema de programación de la producción puede estar caracterizado por datos como el número de trabajos, duración de los trabajos, número de entregas, o el tiempo de entrega comprometido para cada trabajo. Sin embargo, para la determinación de su

complejidad la aproximación que determina el tamaño de una instancia de un problema de producción por el número de trabajos (n) es suficiente (Pinedo 2008).

Existen tablas para los problemas de *scheduling* que permiten estudiar la complejidad de un problema a partir del tipo de problema y de su función objetivo. Estas tablas se construyen aplicando el concepto de reducción. Algunas de estas reducciones son sencillas. Por ejemplo: el problema de *flow shop* (F) reduce al problema *job shop* (J) ($F \propto J$) ya que el primero es un caso particular del segundo. También en cuanto a la función objetivo se pueden establecer reducciones. Por ejemplo $\sum f_j$ puede reducirse a $\sum w_j f_j$ haciendo $w_j = 1$ para todo j. C_{max}, $\sum C_j$, $\sum w_j C_j$ reduce a L_{max}, $\sum T_j$, $\sum w_j T_j$ respectivamente haciendo $d_j = 0$ para todo j (Brucker 2007).

En las Figuras 7 y 8 se muestran alguna de las reducciones más frecuentes para problemas de programación de la producción. La forma de interpretarlas es que el origen de la flecha es reducible al final de la flecha. Por ejemplo, en la Figura 8 $F \rightarrow J$ es equivalente a $F \propto J$.

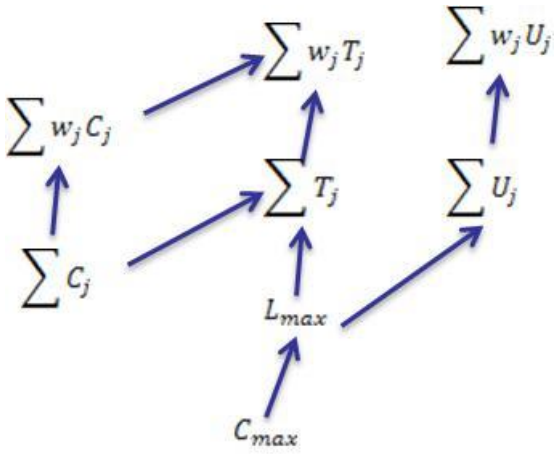

Figura 7. Relaciones de reducción entre funciones objetivo

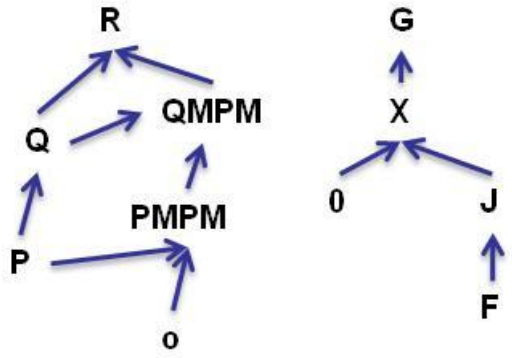

Figura 8. Reducciones básicas por tipos de problema (Brucker 2007)

En Brucker et al (1997) se hace un estudio de la complejidad del problema *job shop* flexible. Se puede encontrar un listado con la complejidad de numerosos problemas de *scheduling* en (Brucker 2009).

1.7. Conclusiones del capítulo

El problema de programación de operaciones dentro del ámbito de fabricación consiste en, dadas unas órdenes de trabajo y un conjunto de recursos disponible: (1) asignar los recursos que van a realizar las operaciones que componen las órdenes de trabajo; (2) determinar la secuencia en que van a ser realizadas las operaciones en cada recurso; (3) calcular los tiempos de comienzo y de finalización de las operaciones; todo ello, de forma que se optimice una función objetivo dada.

La matriz producto-proceso permite caracterizar los principales tipos de entornos de producción. A cada configuración se le asocia un problema de programación diferente. Dada la gran variedad de alternativas posibles, se han realizado propuestas de notación que facilitan su estudio y caracterización, siendo la de (Graham et al 1979) una de las más seguidas. El problema de programación de talleres es un caso particular del problema general de programación de operaciones, al que se le añaden una serie de hipótesis. Este problema está ligado a la producción funcional a medida, que se caracteriza por producir una amplia variedad de productos con una estandarización mínima, donde los equipos están poco especializados y la distribución en planta suele estar orientada a procesos.

Uno de los aspectos más importantes en los problemas de programación de operaciones es su complejidad. Conocer si un problema es fácil o difícil 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. Si el problema es difícil no tiene sentido utilizar la misma estrategia. La mayor parte de los problemas de programación de la producción son NP-Hard. La teoría de complejidad indica que es improbable que existan algoritmos exactos y eficientes para los problemas de la clase NP-Hard. Esto implica que no podemos encontrar una solución sin utilizar un algoritmo de tipo enumerativo, y el tiempo de computación aumenta exponencialmente con el tamaño del problema. Salvo para problemas de pequeño tamaño, no existen métodos exactos que permitan su resolución en tiempos prácticos. Por tanto, si queremos obtener eficiencia en la resolución de problemas difíciles debemos renunciar al óptimo. Es posible utilizar heurísticas que permitan encontrar una solución de forma temprana 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. Métodos para la generación de programas de producción

En el capítulo anterior se indicaba como el problema de programación de operaciones consistía en realizar una asignación de recursos escasos a tareas de producción dentro de un horizonte de programación determinado (Pinedo 2008). Se trata de buscar programas de producción (*schedules*) óptimos, o cercanos al óptimo, dada una serie de restricciones. Es una actividad fundamental dentro de la programación y control de talleres, ya que el programa de producción tiene una incidencia directa en el desempeño del sistema (Toptal y Sabuncuoglu 2009). En la mayoría de casos, el problema de producción es complejo (NP-Hard) y se complica aún más si consideramos los aspectos dinámicos de los entornos reales.

Desde los años 40 y 50 del pasado siglo se han desarrollado un gran número de alternativas para resolver este problema. En la mayoría no se tienen en cuenta las características de los entornos actuales, como dinamismo, restricciones complejas o las frecuentes interrupciones no esperadas, lo que lleva a soluciones poco viables (Ouelhadj y Petrovic 2009). Se puede distinguir dos escuelas: una que afronta el problema desde un punto de vista estático y la otra que lo hace desde un punto de vista dinámico. Los sistemas deterministas asumen que una vez realizado el programa de producción, la secuencia establecida va a seguir siendo válida en relación al

desempeño del sistema durante el horizonte de programación establecido (muy corto plazo). Sin embargo, es bastante probable que esto no ocurra debido a incidencias diversas que puedan producirse como fallos de máquinas, problemas de calidad o la llegada de nuevos trabajos (Aytug et al 1994), aspectos tenidos en cuenta en los sistemas dinámicos.

En los siguientes apartados se describen primero los métodos utilizados en la programación estática, que serán clasificados en exactos y aproximados. A continuación se trata la programación dinámica, que es la que considera que el entorno de programación es cambiante. Se estudiará la naturaleza del entorno, los objetivos buscados, los principales enfoques de programación utilizados, la aplicación de los métodos de resolución en entornos dinámicos y la influencia de la arquitectura del sistema de programación y control.

2.1. Métodos de resolución del problema de programación de talleres desde un punto de vista estático

La programación estática ha sido ampliamente estudiada. El conjunto de métodos desarrollados para su resolución van desde las más simples reglas de lanzamiento aplicadas a la realización de programas de forma predictiva hasta las más sofisticadas técnicas ligadas a los cuellos de botella. Algunas de estas técnicas (métodos exactos) buscan encontrar el óptimo del problema, otras (métodos aproximados) buscan un valor suficientemente cercano al óptimo en un tiempo razonable.

En los métodos exactos se incluyen los métodos eficientes, los métodos de descomposición, la programación dinámica y el *branch and bound*. En los métodos aproximados se incluyen las reglas de lanzamiento, las heurísticas basadas en cuellos de botella, las técnicas de inteligencia artificial, como la satisfacción de restricciones, las redes neuronales, la optimización con colonias de hormigas y los métodos basados en cúmulos de partículas, y los métodos de búsqueda local, como el GRASP, el recocido simulado, la búsqueda tabú, los algoritmos genéticos y los método basados en subastas. La aplicación de estos métodos a los problemas de scheduling aparece en Jain y Meeran (1999), Blazewicz et al (2007) y en Pinedo (2008).

2.1.1. Métodos de programación exactos

Los métodos exactos son aquellos que garantizan la obtención de la solución óptima. Salvo para algunos casos concretos, estos métodos consisten en realizar una enumeración más o menos inteligente o guiada de todas las posibles soluciones del problema. Como se ha comentado en el capítulo anterior, debido a la complejidad de los problemas de programación de operaciones, en los métodos exactos el tiempo de resolución crece exponencialmente respecto al tamaño del problema, excepto en un conjunto restringido de versiones.

2.1.1.1. Métodos eficientes

Los algoritmos eficientes resuelven un problema de forma óptima con requerimientos que crecen polinómicamente respecto al tamaño del input. Estos métodos obtienen soluciones óptimas a partir de los datos del problema siguiendo una serie de reglas simples que logran determinar la solución del problema de forma exacta. Son los primeros problemas de programación que se resolvieron de forma óptima. Estos métodos son aplicables a ciertos problemas con hipótesis muy restrictivas: en la mayoría de los casos son algoritmos de resolución para problemas con una o dos máquinas. Uno de los más conocidos son la regla de Johnson (1954) que se usa para resolver el problema de *flow shop* con 2 máquinas para minimizar el máximo tiempo de flujo (n/2/F/Fmax); o también, para el *job shop*, los desarrollados por Akers (1956) para el problema 2xm y por Jackson (1956) para el problema nx2, donde cada trabajo no tiene más de dos operaciones.

2.1.1.2. Métodos de descomposición

Buscan dividir el problema de optimización a resolver en problemas más pequeños, más fáciles, de forma que, una vez resueltos, puedan integrar sus soluciones para obtener una solución al problema inicial. Los principales enfoques que se han aplicado han sido el método de descomposición de Dantzig-Wolfe (Dantzig y Wolfe 1960), la descomposición de Benders (Benders 1962) y la relajación lagrangiana (Fisher 1981). Normalmente estos métodos no son capaces de encontrar una solución directamente a partir de la descomposición inicial sino que utilizan un procedimiento iterativo para obtenerla.

Algunos ejemplos de aplicación son los realizados por Davies (2005) que estudia la adecuación de los métodos de descomposición de Dantzig-Wolfe y de Benders para solucionar de forma distribuida problemas de asignación combinatoria; Gélinas y Soumis (2005) aplican el método de descomposición de Dantzig-Wolfe a la resolución del problema *job shop*; Hoitomt et al (1993), Wang et al (1997), Kaskavelis y Caramanis (1998) y Chen et al (1998) entre otros aplican la relajación lagrangiana al problema *job shop*.

2.1.1.3. Programación dinámica

El fundamento de este método es dividir el problema en varias etapas y encontrar una función recursiva que permita resolver el problema. Se basa en el Principio de Bellman, que dice que *"dada una secuencia óptima que resuelve un problema, cualquier subsecuencia es, a su vez, óptima respecto al problema que resuelve"* (Bellman, 1957). Este principio no es aplicable a todos los problemas, pero existe un número grande de problemas donde se puede utilizar.

Este método trata de buscar un conjunto de subproblemas que a través de una función de relación recursiva compongan el problema a resolver. La relación entre los subproblemas determina el orden en que deben ser resueltos: comenzando desde el más sencillo, que no depende de ningún otro para ser resuelto, y siguiendo el orden marcado en la relación. Si se aplica la programación dinámica a un problema combinatorio, para calcular el valor óptimo de un subconjunto de tamaño *k*, es necesario calcular el valor óptimo de cada subconjunto de tamaño *k-1*. Por tanto, si el problema está caracterizado por *n* elementos, el número de subconjuntos considerado es 2^n. Esto quiere decir que estos algoritmos tienen complejidad computacional exponencial. Sin embargo, para algunos problemas es posible construir algoritmos pseudopolinomiales (Blazewicz et al 2007).

2.1.1.4. Branch and bound

Se basa en la consideración de la totalidad de soluciones posibles del problema, tratando de descartar conjuntos grandes de candidatos cuyo valor sobrepase una determinada cota de la solución. A partir de este estado inicial se va pasando por distintos estados, aplicando dos técnicas de forma conjunta. La primera es la técnica de ramificación (*branching*) que trata de ir subdividiendo el espacio de soluciones en otros de menor tamaño. La segunda es la técnica de acotamiento (*bounding*) que

elimina aquellas ramas que no sean susceptibles de contener la solución. Para ello va calculando para cada rama las cotas superior e inferior. Si la cota inferior de una rama A es mayor que la cota superior de otra rama B, la rama A puede ser eliminada puesto que no contendrá la solución. Este proceso se va repitiendo hasta llegar a conjuntos unitarios que representan una solución del problema. Esta técnica tiene una complejidad computacional exponencial respecto al tamaño del problema y en muchos casos es inviable su utilización (Burke y Kendall 2005).

Ha sido aplicada por Brucker et al (1994) para resolver el problema *job shop* con función objetivo *makespan*. También lo utilizan Singer y Pinedo (1998) para minimizar el makespan.

2.1.2. Métodos aproximados

Cualquier método de búsqueda que no tenga una garantía formal de encontrar la solución óptima puede ser considerado una heurística. Estos métodos son adecuados si no existen alternativas mejores (Brucker 2007). Aunque los métodos aproximados no garantizan que se encuentre una solución exacta, son capaces de encontrar soluciones cercanas al óptimo en un tiempo razonable.

En la actualidad existen distintas heurísticas que están ampliamente establecidas y que se utilizan en problemas reales de programación de la producción. Como son genéricas, pueden aplicarse a casi cualquier tipo de problema de programación de la producción (Pinedo 2008). Un estudio más detallado de estas heurísticas se puede encontrar en Blum y Roli (2003) y en Burke y Kendall (2005).

2.1.2.1. Heurísticas constructivas

Este tipo de heurísticas se utilizan para construir una solución que cumpla con las restricciones del problema. Se busca la solución que tenga la mejor calidad posible. Destacamos las siguientes:

- Reglas de prioridad

Las reglas de prioridad o reglas de lanzamiento han sido frecuentemente utilizadas por la sencillez en su funcionamiento; sin embargo, la calidad de los programas obtenidos es generalmente baja. Son los primeros intentos que se realizaron para resolver el problema de programación de operaciones de forma aproximada. Uno de

los primeros trabajos, es el desarrollado por Giffler y Thompson (1960), cuyo algoritmo representa la base de la mayoría de heurísticas basadas en reglas de lanzamiento desarrolladas posteriormente, ya que genera programas activos, entre los que se encuentra la solución del problema (Blazewicz et al 2007). Son algoritmos de tipo constructivo, donde se parte de una situación inicial en la que no existe ningún programa y el programa de producción se va construyendo paso a paso. Podemos clasificar las reglas de lanzamiento según diferentes criterios (Pinedo 2008):

- Dependencia del tiempo:
  - ❖ Estáticas: no dependen del tiempo, son sólo función de las características propias de las tareas o de los trabajos. Algunos ejemplos son el tiempo de proceso más corto o la fecha de entrega más temprana.
  - ❖ Dinámicas: dependen del tiempo, un ejemplo es la regla de mínimo slack, que se define como la diferencia entre el tiempo que queda hasta la fecha comprometida del trabajo (d_i) y el tiempo de proceso restante para terminar el trabajo:

$$s_{ij}^{t} = D_i - t - \sum_{q=j}^{n} p_{iq} \tag{2.1}$$

- Ámbito de la información:
  - ❖ Basadas en información local: son aquellas que sólo utilizan la información relacionada con las tareas que se encuentran en la cola que se quiere ordenar.
  - ❖ Basadas en información global: estas reglas utilizan información relacionada con otras máquinas.

En la Tabla 2.1 se muestran algunas de las reglas más conocidas. La primera columna representa la abreviación y el nombre de la regla, mientras que la segunda describe qué operación o qué trabajo en conflicto tiene la prioridad más alta.

Distintos trabajos han tratado de encontrar cual es la regla de lanzamiento más adecuada, como por ejemplo Blackstone et al (1982), Haupt (1989), Ramasesh (1990) y Rajendran y Holthaus (1999). Sin embargo, no se ha encontrado ninguna regla que sea mejor para todos los casos o para todas las medidas del rendimiento. En ella influyen variables como el grado de utilización de las máquinas o el grado de ajuste de las fechas de entrega comprometidas.

Cuando se quiere optimizar más de un objetivo se utilizan reglas de decisión compuestas, ya que las reglas de decisión simples funcionan relativamente bien cuando se quiere optimizar un único objetivo pero no son adecuadas con múltiples objetivos.

Abreviatura	Prioridad	Dependencia del tiempo	Ámbito de la información
SIRO (service in random order)	La operación se elige de forma aleatoria.		
FCFS (First come first served)	La primera operación en la cola de los trabajos que están esperando para la misma máquina.	dinámica	local
SPT (shortest processing time)	El trabajo con el tiempo total de proceso menor.	estática	local
LPT (longest processing time)	El trabajo con el tiempo total de proceso mayor.	estática	local
LRPT (longest remaining processing time)	La operación con el tiempo de trabajo restante más largo.	estática	global
STPT (shortest remaining processing time)	La operación con el tiempo de trabajo restante más corto.	estática	global
LOS (longest operation successor)	La operación cuya sucesora tenga el tiempo de proceso más largo.	estática	global
SNRO (smallest number of remaining operations)	La operación que tenga el menor número de operaciones posteriores.	estática	global
LNRO (largest number of remaining operations)	La operación con el número mayor de operaciones posteriores.	estática	global
SQNO(Shortest queue at the next operation)	La operación cuya sucesora tenga una cola de espera más corta.	dinámica	global
MS (Minimum Slack)	La operación con el mínimo slack.	dinámica	local

Tabla 2.1. Reglas de prioridad. Adaptado de Blazewicz et al (1996) y (Pinedo 2008)

- Heurísticas basadas en cuello de botella

Existen varios métodos en programación de la producción que tienen como base el recurso cuello de botella del sistema de producción. La idea es optimizar los recursos más críticos para optimizar el rendimiento del sistema.

Los métodos de cuello de botella cambiante (*shifting bottleneck*) tratan de superar el inconveniente que supone que el cuello de botella no siempre corresponde a una sola máquina. Este método se caracteriza por las fases de identificación del subproblema, selección del cuello de botella, resolución del subproblema y reconstrucción del programa de producción. El procedimiento seguido es una mezcla de técnicas constructivas y de mejora. La idea es resolver para cada recurso uno a uno como el recurso cuello de botella de entre los recursos aún no secuenciados. Cada vez que se programa una máquina identificada, el resto de máquinas previamente programadas son reoptimizadas resolviendo un problema de programación de una sola máquina. Las máquinas se reoptimizan hasta que no se encuentre ninguna mejora. El algoritmo finaliza cuando todas las máquinas han sido programadas. (Adams *et al.* 1988). Podemos ver una aplicación de esta técnica a la resolución del problema *job shop* en Balas y Vazacopoulos (1998), así como en Pinedo y Singer (1999) donde minimiza la suma ponderada de los retrasos.

2.1.2.2. Heurísticas basadas en trayectorias

La mayoría de heurísticas aplicables a la resolución de problemas de naturaleza combinatoria desarrollados en los últimos años entran en esta categoría. En estos métodos el proceso de búsqueda que realizan se caracteriza por seguir una trayectoria en el espacio de búsqueda: a diferencia de los métodos de tipo constructivo, que empiezan de cero y van construyendo el programa de producción incorporando una tarea cada vez, estas metaheurísticas empiezan desde el primer momento con un programa construido mediante cualquier técnica constructiva, que puede seleccionarse de forma aleatoria y que posteriormente se va mejorando mediante un proceso iterativo (Blum y Roli 2003).

Este tipo de algoritmos no garantizan la consecución de una solución óptima. En cada paso se busca un programa vecino que mejore el programa actual. Un programa es vecino si puede ser obtenido a partir del primero mediante una modificación básica o movimiento básico. En el proceso se van aceptando o rechazando posibles candidatos en base a un criterio determinado de aceptación. Los métodos se diferencian entre sí por (Pinedo 2008):

- El modelo de representación del programa de producción.
- La forma de definir el vecindario.
- El proceso de búsqueda entre el vecindario.
- El criterio de aceptación o rechazo.

Uno de los problemas que aparece en estas técnicas es el estancamiento en un óptimo local. Para evitar este inconveniente, los métodos utilizan alguna de estas estrategias: (1) incorporar aleatoriedad al proceso de forma que se exploren regiones alejadas de la solución actual con una cierta probabilidad, (2) utilizar la memoria, almacenando información sobre las regiones ya exploradas. Destacamos los siguientes:

- Recocido simulado

El recocido simulado (*simulated annealing*) utiliza como modelo el proceso de recocido que se utiliza en metalurgia. El recocido es un tratamiento térmico en el que tras un calentamiento inicial de un metal y un enfriamiento lento posterior, se consigue reordenar su estructura a nivel molecular de forma que el estado de energía es inferior al estado inicial.

En los problemas de programación de la producción se establece una analogía entre la energía del sólido y la función a optimizar. En el proceso de enfriamiento se aceptan incluso soluciones que empeoran la solución actual. Con ello se trata de evitar que el algoritmo se estanque en un óptimo local. Esto se controla mediante el parámetro de temperatura, de forma que a altas temperaturas se aceptan soluciones peores con una determinada probabilidad. A medida que la temperatura disminuye, la probabilidad de aceptar estas soluciones que empeoran la solución actual va decreciendo. Kolonko (1999), Ponnambalam et al (1999), Wang y Wu (2000) y Wu et al (2005), aplican esta técnica al problema de *job shop*.

- Búsqueda tabú

El método de recocido simulado no memoriza las soluciones que ya han sido exploradas, lo que puede provocar que se entre en bucles donde se visita siempre la misma secuencia de soluciones. Para solucionar este problema se utilizan listas donde se almacenan las soluciones ya visitadas. En el proceso de búsqueda se irán aceptando los pasos que mejoren la solución actual y que no se encuentren en la lista tabú (Glover et al 1993). La diferencia entre ambos radica principalmente en sus criterios

de aceptación o rechazo. En el recocido simulado este proceso es probabilístico, mientras que en la búsqueda tabú se trata de un proceso determinista.

Se pueden encontrar aplicaciones del método de búsqueda tabú a la resolución del problema *job shop* en Dell'Amico y Trubian (1993), Brandimarte (1993), Nowicki y Smutnicki (1996, 2005), Armentano y Scrich (2000), Pezzella y Merelli (2000), Scrich et al (2004), Grabowskil y Wodecki (2005) y en Liang y Huang (2005).

- GRASP

Greedy Randomized Adaptative Search Procedure (GRASP) engloba un conjunto de métodos de búsqueda de soluciones con funcionamiento similar: primero se realiza una construcción aleatoria de una posible solución que luego se utiliza como base para ser mejorada (Resende y Ribeiro 2003). El algoritmo se basa en los siguientes pasos: (1º) se construye una solución inicial, (2º) se aplican modificaciones básicas aleatorias para construir nuevas soluciones, (3º) en el momento que se encuentra una solución que mejora la actual se toma como mejor solución y se vuelve al segundo paso. Aiex et al (2003) y Rajkumar et al. (2011) lo han aplicado al problema *job shop*.

2.1.2.3. Metaheurísticas basadas en poblaciones

Las metaheurísticas basadas en poblaciones basan su funcionamiento en utilizar en cada iteración un conjunto de soluciones, en lugar de una única solución como en los casos anteriores. El funcionamiento de la heurística dependerá de cómo evolucionan esas poblaciones en la exploración del espacio de soluciones. Las más conocidas son los algoritmos genéticos, la optimización con colonias de hormigas y la optimización por cúmulos de partículas.

- Algoritmos genéticos

El tratamiento de la programación de operaciones mediante algoritmos genéticos se modela de la siguiente forma: las posibles secuencias o programas de producción se ven como individuos que forman parte de una población. Cada individuo se caracteriza por su *fitness*, que es el valor que se utiliza como criterio en el proceso de selección. El procedimiento se repite de forma iterativa, y cada iteración se denomina generación. La población de cada generación está formada por los individuos de la generación anterior que sobreviven más los hijos de la nueva generación. Los hijos se obtienen como resultado de la reproducción y la mutación de los individuos de la

generación anterior. Una mutación sería un cambio en la secuencia que representa un determinado individuo. En cada generación se reproducen los individuos con un valor más alto de *fitness*, mientras que mueren los que tienen un valor más bajo (Pinedo 2009).

Se puede considerar que la técnica de algoritmos genéticos es una generalización de las técnicas de recocido simulado y de búsqueda tabú. Estas últimas pueden ser vistas como casos particulares del primero. En cara iteración de la técnica de algoritmos genéticos se generan muchas secuencias que componen la población, mientras que en el recocido simulado y en la búsqueda tabú se genera una sola secuencia en cada iteración. Estas técnicas pueden ser vistas como algoritmos genéticos con población formada por un único individuo.

Encontramos algunos ejemplos de la aplicación del método de algoritmos genéticos al problema *job shop* en Cheng et al (1996, 1999), Pérez (2000), Park et al (2003), Mattfeld y Bierwirth (2004), Tanev et al (2004), Gonçalves et al (2005), Watanabe et al (2005), Essafi et al (2008), Zhou et al *(*2009a).

- Optimización con colonias de hormigas

También conocidas como *Ant Colony Optimization* (ACO). Este procedimiento utiliza como modelo el mecanismo mediante el cual las hormigas consiguen encontrar el camino más corto entre la fuente de comida y el hormiguero. Las hormigas caminan aleatoriamente en busca de comida. Cada hormiga va dejando a su paso un rastro de feromona. Esta feromona es volátil y desaparece con el tiempo. Cuando una hormiga encuentra comida regresa al hormiguero, siguiendo su propio rastro de feromona. El rastro de feromona se va reforzando en el camino cada vez que una hormiga pasa. Las hormigas tienden a seguir el rastro de feromona con una cierta probabilidad. Cuando una hormiga se desvía del rastro inicial, pero encuentra un camino más corto al hormiguero este camino se verá reforzado debido a la volatilidad de la feromona, que hace que cuanto menos frecuente es el paso de una hormiga por un punto del camino, menor será el rastro que queda. 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.

Utilizando como modelo este mecanismo se construyen algoritmos en los que se generan soluciones con una componente aleatoria, acercándose progresivamente a las combinaciones de valor óptimo. Se puede encontrar un trabajo detallado en esta área en Dorigo y Stützle (2004). Esta técnica ha sido aplicado al problema *Job shop* por Lo

y Hsu (1993), Fonseca y Navaresse (2002), Blum y Sampels (2004) y Wang et al (2004).

- Optimización por cúmulos de partículas

En la literatura se conoce como *particle swarm optimization* (PSO). Fue presentado por Kennedy y Eberhart (1995). Estos sistemas están basados en la analogía con los movimientos que realizan los bancos de peces, bandadas de aves o los enjambres de insectos. El movimiento del sistema está condicionado por un líder, pero el movimiento de cada individuo depende además de su propia experiencia. En Sha y Hsu (2006), Xia y Wu (2005), Lian et al (2006) se aplica esta técnica a la resolución del problema *job shop*.

2.1.2.4. Otros enfoques

- Redes neuronales

El paradigma de las redes neuronales artificiales utiliza la estructura organizativa del sistema nervioso de los animales como base para realizar los procesos de aprendizaje y procesamiento automático. Las redes neuronales artificiales (RNA), *Artificial Neural Systems* (ANS), constan de un conjunto de procesadores elementales (neuronas) conectadas en red. Una red neuronal está compuesta de varias capas de procesadores elementales (neuronas) o nodos. Estos nodos están conectados entre sí. Cada neurona es estimulada, bien desde fuera del sistema o bien por otras neuronas, proporcionando una respuesta, que puede ser una salida del sistema o estimular a otras neuronas. Las redes neuronales artificiales se caracterizan por su capacidad de aprendizaje. A cada arco de la red se le asocia unos pesos y unos umbrales máximos de señal, que constituyen los grados de libertad del sistema. Las redes pueden ajustas los pesos y los umbrales en un proceso de aprendizaje o entrenamiento.

En Jain y Meeran (1999) y en Akyol y Bayhan (2007) aparecen las principales aplicaciones de redes neuronales artificiales a los problemas de programación de la producción. Se han utilizado redes de búsqueda (redes de Hopfield) (Jeng y Chang 1997), redes probabilísticas (máquina de Boltzman) (Satake et al 1994), redes de corrección del error (perceptron multicapa) (Feng et al 2003), redes competitivas (Chen y Huang 2001) y redes autoorganizadas (McMullen 2001).

- Satisfacción de restricciones (constraint satisfaction)

Se trata de una tecnología software desarrollada para la descripción y resolución de problemas combinatorios en los que las restricciones tienen gran importancia. Es una técnica que proviene del campo de la inteligencia artificial y que, en los últimos años, se ha combinado con técnicas de investigación operativa para mejorar sus resultados. Su objetivo inicial es obtener soluciones factibles que cumplan todas las restricciones; sin embargo, estas soluciones no tienen por qué ser óptimas (Pinedo 2008).

El problema de producción se modela como un conjunto de variables que pueden tomar valores de unos dominios predefinidos y un conjunto de restricciones que acota los valores que pueden ser asignados a los subconjuntos de variables (Beck *et al.* 2003). Se define una terna (X, D, C) donde: X es un conjunto de variables $\{x_i\}$, D es un conjunto de dominios $\{d_i\}$, donde cada dominio d_i es el conjunto de valores posible que puede tomar la variable x_i, y C es un conjunto de restricciones $\{c_i\}$ que limitan los valores que pueden tomar las variables simultáneamente. La solución de estos problemas consiste en una asignación de valores a las variables de forma que se satisfagan las restricciones.

Estas técnicas tratan de reducir el tamaño del espacio de búsqueda aplicando restricciones que van acotando el orden de selección de las variables y la secuencia en la que los posibles valores se asignan a cada variable. En este proceso van apareciendo inconsistencias que son eliminadas progresivamente. Si el problema se ha resuelto, se pasa a una exploración sistemática del espacio de soluciones hasta que se encuentra una solución o se prueba que no existe.

En los años 90 de la pasada centuria, se desarrolló el Optimization Programming Lenguaje (OPL) (ver apartado 1.4.2), creado para modelar problemas de optimización mediante una combinación de programación con restricciones y técnicas de programación matemática. Se puede encontrar la base teórica respecto a cómo representar los problemas de programación de la producción utilizando esta metodología en Baptiste et al (2001)

- Métodos basados en subastas

Estos métodos toman las subastas como modelo asignación. Su desarrollo ha ido en paralelo con el desarrollo de los modelos basados en agentes (*agent-based models ABM*). Estas técnicas son consideradas como una de las heurísticas más prometedoras, principalmente en problemas complejos que se desarrollan en entornos distribuidos (Barbati et al 2012). Los métodos basados en subastas utilizan los precios que se

obtienen mediante la aplicación de protocolos de mercado para obtener los programas de producción.

Básicamente consiste en la siguiente idea: se crea una economía de mercado en la que agentes que representan órdenes de trabajo y máquinas negocian entre sí para contratar los servicios que permitan completar las órdenes, obteniendo un beneficio por su realización (Pinedo 2008). Existen distintas formas de implementar esta idea: Wellman et al (2001) define el problema de asignación de recursos en un entorno de fabricación donde un conjunto de bienes indivisibles (los intervalos de tiempo), cuyo poseedor inicial se llama agente máquina, que debe ser asignados a un conjunto de compradores, los agentes órdenes de trabajo. El problema consiste en encontrar la asignación que permite maximizar la suma de la utilidad obtenida por los compradores y por el vendedor respectivamente por los bienes asignados. El proceso de asignación puede ser analizado como un mercado en el que los agentes y vendedor intentan maximizar su utilidad. Esta situación se da cuando se llega a unos precios de equilibrio.

Otra de las alternativas es aplicar al problema de scheduling definido mediante programación matemática técnicas de descomposición, como las de Danztig-Wolfe y Benders (Davies 2005) o la relajación lagrangiana (Luh y Hoitomt 1993, Kaskavelis y Caramanis 1998). Estas técnicas permiten obtener del problema original subproblemas que pueden ser asignados a agentes que tratan de maximizar su utilidad. Se puede establecer una analogía entre el proceso de resolución que utilizan y los mecanismos de subasta, de forma que las variables adicionales que utilizan, multiplicadores en la relajación lagrangiana y variables duales en la descomposición de Danztig-Wolfe y Benders, pueden ser interpretadas como precios.

Una de las dificultades en la aplicación de este enfoque a problemas como el *job shop* es que, debido a la existencia de complementariedades, el equilibrio no tiene por qué existir, y en caso de que exista las subastas simples no son capaces de encontrarlo. Estas complementariedades se dan cuando la valoración de un bien no es independiente del resto de bienes que se adquieren. En los problemas de scheduling encontramos complementariedades debidas, por ejemplo, a las restricciones de continuidad de las tareas o a las relaciones de precedencia. En este caso es necesario acudir a mecanismos más complejos, como son las subastas combinatorias, que tienen como inconveniente su complejidad computacional potencial. Estos aspectos serán desarrollados en el capítulo 4.

2.2. Programación en entornos dinámicos

A pesar de la naturaleza dinámica y cambiante de los entornos productivos, la gran parte de los estudios realizados sobre programación de operaciones lo han ignorado y se han centrado en la optimización de programas de modo estático. La mayor parte de estos trabajos tratan de encontrar soluciones óptimas o cercanas al óptimo, utilizando distintas medidas como el tiempo total de proceso o el coste mínimo de producción. Estos modelos suponen de modo implícito un entorno estático y con mínimas variaciones respecto a la situación de partida. Se asume que las diferencias en la ejecución respecto al programa no afectan a la calidad del mismo.

Una de las posibles causas de que así haya sido es que históricamente una de las principales fuentes de incertidumbre en los entornos productivos ha sido la falta de información exacta respecto a la situación real del sistema en cada instante. Sin embargo, la llegada de los sistemas informáticos y redes de comunicación ha hecho posible el seguimiento tanto de las máquinas como de los trabajos, permitiendo desarrollar sistemas de producción más dinámicos.

Los sistemas de producción reales necesitan de sistemas de programación ágiles, que sean capaces de adaptarse rápidamente a los cambios en las condiciones de partida (Rajabinasab y Mansour 2011). Los sistemas tradicionales de planificación tratan de realizar de antemano un programa que optimice las operaciones de producción para el siguiente periodo. Pero estos programas, en la práctica, son difíciles de mantener debido a que las condiciones cambian con frecuencia. Como resultado, algunas operaciones se manejan sin un programa detallado. Nunca se consigue el óptimo planificado ya que las condiciones con que se había realizado la programación no se mantienen.

Estas perturbaciones pueden estar relacionadas: (1) con los recursos internos: averías de máquinas, baja de operarios, fallos o no disposición de herramientas, límites de carga, retraso en la llegada o falta de materiales, materiales defectuosos o con especificaciones incorrectas; o (2) con los trabajos: urgencias, cancelaciones, cambios de fechas de entrega, llegada temprana o con retraso, cambios en la prioridad o cambios en el tiempo de proceso (Ouelhadj y Petrovic 2009).

La programación adaptativa forma parte del sistema de programación y control. Su objetivo es rehacer el programa de producción de forma que se adapte mejor a las condiciones existentes en cada momento. El sistema de control detecta la existencia de algún suceso que puede hacer que el programa existente sea obsoleto y propone un

nuevo programa que sustituya al anterior. La información en tiempo real hace que en todo momento deban ser revisados y reconsiderados los programas de producción establecidos.

Los sistemas de control deben utilizar métodos de programación que operen en un entorno dinámico y que resuelvan de modo efectivo y eficiente el problema de programación de la producción con un mínimo de información global y que consideren las siguientes características (Liu et al 2007):

- Robustez: capacidad del sistema para mantener su efectividad en presencia de disrupciones.
- Adaptabilidad: capacidad del sistema para generar programas de producción que se puedan acomodar fácilmente ante la llegada de un rango amplio de distintos tipos de eventos no esperados.

2.2.1. Frecuencia de revisión de los programas

En este punto estudiamos las estrategias de generación de programas para controlar la producción en entornos dinámicos. Se debe decidir si se considera la programación de las tareas en un periodo futuro o si sólo se consideran las tareas que se van a realizar de forma inmediata (Raheja y Subramaniam 2002, Ouelhadj y Petrovic 2009).

2.2.1.1. Sistemas predictivos puros

En muchos sistemas de fabricación la forma de realizar la programación es asumir que no va a ocurrir ningún tipo de imprevisto, pasando posteriormente el programa al personal de planta. El programa servirá de guía para la toma de decisiones. Esto es lo que se conoce como programación predictiva (*predictive scheduling*) o programación fuera de línea (*offline scheduling*).

La programación predictiva forma parte de la mayoría de los sistemas de planificación de la producción. En ella se asume que la información relativa a los trabajos y la disponibilidad de las máquinas se conocen antes de la realización del programa de producción. El programa predictivo sirve como plan global sobre el que se basan las actividades del taller. Permite una mejora de actividades, identificar posibles conflictos entre recursos, controlar la salida de órdenes de trabajo a planta y

asegurar que los pedidos de materias primas sean realizados a tiempo; lo que posibilita el aumento de la productividad y la minimización de los costes de operación.

No obstante, el hecho de que la mayoría de sistemas productivos operen en entornos dinámicos en los que son frecuentes los sucesos en tiempo real totalmente impredecibles, hace que los programas óptimos calculados estén ya obsoletos en el momento de lanzarlos a producción (Ouelhadj y Petrovic 2009), evidenciando la existencia de un gap entre la teoría de programación de operaciones y la programación en la práctica por su naturaleza dinámica (MacCarthy y Liu 1993).

Podemos calificar la programación de la producción de estática o programación predictiva si se calcula a partir de datos previstos. El cálculo se lanza en modelo fuera de línea o desconectado (*off-line*), en el sentido de que los datos sobre los recursos no se obtiene en tiempo real y que las soluciones calculadas no se aplican inmediatamente. Por el contrario, la programación de la producción dinámica o programación reactiva se efectúa en tiempo real, es decir, las tareas son programadas a medida que van llegando.

2.2.1.2. Programación dinámica pura

Un sistema de programación dinámica es aquel que utiliza la información en el momento en el que se genera (Cowling y Johansson 2002). También se denominan sistemas de programación en línea (*online scheduling*). Aunque se utiliza el término "programación", en realidad estos sistemas se caracterizan porque no se genera ningún programa de antemano sino que las decisiones se van tomando de forma local en tiempo real. En cada momento se selecciona el siguiente trabajo a lanzar a planta siguiendo una regla de asignación que elige el trabajo con mayor prioridad. La prioridad se calcula en base a los atributos de las máquinas y de los trabajos.

Las ventajas de este criterio están en su baja carga de procesado, y en la facilidad de ser explicado a sus usuarios. Estos enfoques no utilizan verdaderos "programas" puesto que no existe una previsión de lo que va a ocurrir, sino que actúan reaccionando ante las circunstancias cambiantes. Aunque parezca que este enfoque sólo considera la información local de cada elemento, sí que tienen en cuenta la información a más largo plazo, puesto que en muchos de los criterios de decisión se utiliza (como p.e. la fecha de entrega de un determinado pedido).

2.2.1.3. Programación predictiva-reactiva:

Parece que ninguno de los dos enfoques anteriores es universalmente adecuado. Por ejemplo, en el caso de que los tiempos de ejecución sean completamente aleatorios, parece que un enfoque reactivo funciona mejor que uno predictivo, pero esto no es así en otro tipo de problemas, como cuando introducimos tiempos medios de fallos de máquina. Por tanto, en entornos con mucha incertidumbre los sistemas predictivos van a funcionar peor que los sistemas reactivos.

En este tipo de sistemas se realiza la programación de las tareas futuras, siendo susceptible de ser modificada. Se plantea una programación inicial desde un punto de vista predictivo, que será seguida hasta que exista una interrupción, en que se modificará el programa. El problema de programación de la producción así planteado conduce a no eliminar las herramientas de programación de la producción que pueden existir en un taller de producción. Se propone una reprogramación que se base en las soluciones obtenidas en un sistema de programación de la producción previsional. Este enfoque es complementario y no alternativo a un sistema de programación previsional clásico.

Sin embargo, en la mayoría de los sistemas reales, en los que la incertidumbre es elevada, es poco probable que la programación inicialmente planteada pueda llevarse a cabo: las interrupciones van a hacer que el programa inicialmente lanzado no pueda ser realizado por lo que será necesario cambiarlo para permitir su ejecución y quizá aprovechar para realizar mejoras considerando la nueva situación. Esto es lo que se llama *rescheduling.* La reacción puede ser la realización de un nuevo programa o bien la modificación del anterior, que se seguiría hasta una nueva interrupción.

La reprogramación se realizaría cuando se produjera un evento con potencial suficiente para producir una alteración importante en el sistema. El reconocimiento de la misma se puede realizar de forma continua, o bien estableciendo periodos de revisión. Podemos clasificar los distintos enfoques, en base a este criterio, como:

- Dirigida por eventos (*Event-driven*): el control se realiza de forma continua. Es el caso extremo: se crea un nuevo programa cada vez que se produzca un determinado evento. La naturaleza online de la programación por eventos y los requisitos asociados de respuesta en tiempo real hacen que el programa de producción deba ser computado dentro del periodo de tiempo para el que el programa sigue siendo válido. Éste puede ser muy corto en ciertos casos (Raheja y Subramaniam 2002).

- Periodificación del espacio temporal o *rolling time horizon*: Agrupa dos enfoques principales: (1) Periódica: se establecen periodos de revisión (T) en los que se revisa si durante ese periodo ha tenido lugar algún evento que haga necesario rehacer la programación (Raman y Talbot 1993). (2) Híbrida: es un caso intermedio entre ambas, donde la revisión del programa se realiza periódicamente, salvo que tenga lugar algún evento importante que haga necesario revisar la programación.

Un sistema ideal de programación en un entorno dinámico debería llegar a un compromiso entre los recursos utilizados y la estabilidad del sistema. El sistema de revisión continuo puede consumir muchos recursos informáticos, mientras que en el sistema periódico puede ocurrir que en el momento de revisar la programación del sistema, ésta resulte ya irrealizable. En cuanto a la estabilidad del sistema, los sistemas de revisión periódica son más estables, mientras que los sistemas de revisión continua se caracterizan por lo contrario.

La tendencia es hacia sistemas híbridos, con periodos de revisión establecidos, pero en caso de que se produzca una incidencia importante también se revisa el programa. Esto es lo que se realiza en la práctica: se revisan los programas en un horizonte determinado que puede ser de un día o un turno y en caso de que ocurra alguna incidencia destacable se modifica el programa.

2.2.2. Métodos de generación de programas

Otro aspecto a considerar es, si a la hora de realizar un nuevo programa se parte de cero o si, por el contrario, se modifica el programa de producción ya existente. Esto da lugar a dos enfoques en la programación de la producción:

- Reelaboración completa del programa. Cada vez que se hace necesario modificar el programa existente se parte desde cero. Las técnicas de programación utilizadas son las utilizadas en la programación estática.
- Modificación del programa existente (reprogramación). En lugar de partir de cero, se modifica el programa existente reprogramando las tareas afectadas por el evento. La reprogramación trata de aprovechar el coste de cálculo efectuado para realizar la programación inicial cuando las condiciones iniciales se han modificado. Aunque se emplea en los sistemas de programación de la producción, está considerado como el "pariente pobre" de la programación. A pesar de no existir una definición clara de en qué consiste, sin embargo, por su

orientación práctica constituye un objeto de estudio importante dado que se acerca a los problemas industriales reales (Miyashita et al 1996). Este enfoque trata de dar la posibilidad a los primeros actores, que son afectados por una perturbación, de reaccionar lo más pronto posible con el fin de minimizar las consecuencias de una perturbación. Esto se traduce en términos objetivos en una voluntad de preservar en lo posible la programación inicial, limitando los cambios. Se trata de encontrar la adaptación continua y mejor del programa de producción anteriormente calculado (Raheja y Subramaniam 2002).

2.2.3. Robustez de los programas

La programación realizada debe intentar disminuir el impacto de posibles alteraciones futuras en el programa. La cuestión es: cómo programar para que en caso de interrupción se reduzca al mínimo el impacto producido en el programa. Algunos procedimientos proponen probar con diferentes soluciones hasta encontrar la que se adapta mejor a distintos escenarios posibles. Otras buscan minimizar la función de degradación, que mide la diferencia entre el valor objetivo de la programación predictiva y la realizada.

Atendiendo a la robustez de los programas se puede hablar de:

- Programación predictiva-reactiva simple: Es el enfoque más utilizado en sistemas de producción. Este enfoque se basa en la revisión de los programas como respuesta a los sucesos en tiempo real. Se trata de un proceso en dos pasos: primero se genera un programa predictivo con el objetivo de optimizar el funcionamiento de la planta, sin considerar ninguna interrupción en el taller. Este programa se modifica durante la ejecución en respuesta a los sucesos disruptivos (Yamamoto y Nof 1985, Bean *et al.* 1991; Church y Uzsoy 1992; Leon *et al.* 1994a).
- Programación predictiva-reactiva robusta: La programación predictiva-reactiva se basa en simples ajustes que consideran solamente la eficiencia de la planta de producción. Esto puede afectar a otras actividades ya programadas en el programa original y provocar un rendimiento pobre del programa. La programación predictiva-reactiva robusta busca generar programas que minimicen los efectos de la disrupción en el programa. La solución típica a este problema ha sido la reprogramación considerando tanto la eficiencia de la planta como la desviación respecto al programa inicial. Se buscan soluciones

bicriterio que optimicen la eficiencia del programa y la estabilidad del mismo, por ejemplo minimizando respectivamente el makespan y la desviación respecto al momento de comienzo de los trabajos (Leon et al 1994b; Wu et al 1999).

- Programación proactiva robusta: Algunos trabajos de investigación proponen gestionar de antemano las perturbaciones potenciales, es decir, establecer "soluciones de recambio" antes de que lleguen. Se centra en construir programas predictivos que satisfagan los requerimientos predictibilidad de un entorno dinámico (Mehta 1999, Lou et al 2012).

En caso de que se dé una elevada incertidumbre en el sistema, puede ser más conveniente estudiar los factores que la causan y tratar de eliminarlos. Una posible dirección de estudio sería desarrollar metodologías para determinar qué grado de incertidumbre en el sistema hace aconsejable el uso de un sistema reactivo frente a uno predictivo. Otro aspecto es desarrollar un estudio sistemático de los costes de reconfiguración del sistema.

2.2.4. Técnicas utilizadas en programación en entornos dinámicos

El problema de programación dinámica de la producción ha sido ampliamente estudiado para entornos *job shop* y sistemas de producción flexible. Hay que destacar que los trabajos de investigación en el dominio de la programación de la producción se realizan frecuentemente como respuesta a un problema concreto encontrado en la industria. En la mayoría de los casos, los métodos utilizados son los mismos que en el caso estático, adaptándolos para que recojan la naturaleza dinámica del sistema.

Se pueden encontrar algunos ejemplos en Rajendran y Holthaus (1999) donde se realiza una comparativa entre reglas de lanzamiento en un entorno dinámico. Liu et al (2005) utilizan búsqueda tabú, Zhou et al (2009b) utilizan optimización basada en colonias de hormigas para resolver el problema *job shop* en un entorno dinámico. Otros trabajos utilizan algoritmos genéticos, como es el caso de Bierwirth et al (1995) Mesghouni et al (1999), Qi et al (2000), Rossi y Dini (2000) y Chryssolouris y Subramaniam (2001). En todos ellos se realiza una reelaboración completa del programa.

Wu y Wysk (1989), Ishii y Talavage (1991), Shaw et al (1992), Cho y Wysk (1993), Jeong y Kim (1998), Kim et al (1998), Arzi y Iaroslavitz (1999), Aydin y Öztemel (2000), Raheja y Subramaniam (2002) y Shnits et al (2004), utilizan algoritmos de aprendizaje para seleccionar la regla de prioridad más apropiada basada en los datos actualizados en planta de producción. Para ello se divide el horizonte de programación en pequeños intervalos y al inicio de cada uno se elige cual es la regla que funciona mejor. El sistema utilizado consiste en modelar primero el sistema productivo a estudiar, ver su comportamiento utilizando diferentes reglas, y por último aplicar un algoritmo de aprendizaje inductivo, mediante redes neuronales, para predecir la regla a seguir cuando el sistema tenga unas determinadas condiciones, o aprendizaje genético para seleccionar un conjunto de reglas para una configuración de sistema dada.

Uno de los enfoques que ha tenido más desarrollo en los últimos años en la resolución de problemas de programación dinámicos es utilizar los sistemas basados en mercado como técnica de optimización teniendo como base para su implementación el modelado basado en agentes Este enfoque aprovecha las características de los sistemas multiagente que los hace adecuados para resolver problemas complejos y en entornos distribuidos. Estos son sistemas modulares, lo que permite una división más fácil en subproblemas. Esta división permite que los subproblemas a resolver sean más sencillos que el problema de partida. Además tienen una alta capacidad de reacción ante los frecuentes cambios que se producen en el entorno de producción. Este enfoque ha sido aplicado a la resolución del problema *job shop* en entornos dinámicos por Dewan y Joshi (2002) y por Araúzo (2007).

2.3. Conclusiones del capítulo

En este capítulo se han revisado los principales métodos de programación de operaciones. Las primeras propuestas consideran entornos de producción estáticos, donde las condiciones de programación son conocidas y no cambian con el tiempo. Esto es lo que se denomina programación predictiva. Sin embargo, los entornos de producción reales se caracterizan por un alto dinamismo. En cada instante las condiciones para las que se había definido el programa pueden cambiar, haciendo que éste deje ya de ser válido.

En los últimos años una parte de la investigación en programación de operaciones se está centrando en idear métodos de programación que permitan reaccionar ante los

cambios que se producen en el momento de ejecutar las órdenes de producción. La programación dinámica estudia la programación de la producción considerando los cambios en las condiciones que se producen. Con ello aparecen nuevas cuestiones a resolver: ¿qué se hace con el antiguo programa de producción?, ¿deja de ser válido en su totalidad o es posible recomponerlo?, ¿cómo afectan los cambios en el programa a los objetivos? Además aparecen aspectos relevantes relacionados con el tipo de estructura organizativa que sustenta estas técnicas (más centralizada o más descentralizada) y las implicaciones que tiene respecto a la coordinación de los elementos.

Una de las técnicas de optimización con más desarrollo en los últimos años son los sistemas basados en subastas, que tienen como base el modelado basado en agentes. Los próximos capítulos se centran en estas técnicas. En el capítulo 3 se hablará del modelado basado en agentes y la programación distribuida, y en el capítulo 4 se mostrará las subastas como mecanismos de coordinación en sistemas distribuidos.

Capítulo 3. Programación de operaciones distribuida y sistemas multiagente

Como ya se ha comentado en anteriores capítulos, la necesidad de disponer de sistemas de producción robustos, flexibles y con capacidad de respuesta ante los cambios ha hecho surgir en los últimos años nuevos enfoques de la programación de operaciones que tienden hacia los sistemas distribuidos. Este hecho se ha visto reforzado por el desarrollo e implementación de los sistemas multiagente, paradigma software que aparece en los años 80 como una rama de la inteligencia artificial distribuida. Su funcionamiento se basa en que mediante la interacción de elementos independientes se puede alcanzar un comportamiento global determinado. La implementación de estos sistemas en los entornos de planificación y control de la producción trata de conseguir sistemas distribuidos que proporcionen una mejor respuesta en entornos cada vez más dinámicos y cambiantes.

Este capítulo se divide en tres partes: en la primera se introduce el concepto de programación de operaciones distribuida, en la segunda parte se analizan los sistemas multiagente como soporte tecnológico de implementación de este tipo de sistemas, y en la tercera se plantean los aspectos básicos en un sistema de programación de operaciones basado en agentes.

3.1. Programación de operaciones distribuida

La programación de operaciones distribuida trata de buscar métodos en los que se generen programas globales eficientes integrando los programas locales desarrollados por cada agente involucrado. Esto facilita la inclusión de los objetivos y restricciones locales en el problema global. En la programación de operaciones distribuida las decisiones que permiten resolver el problema no se toman por un único decisor, sino que son tomadas por decisores locales independientes entre sí. Como estos decisores se guían por sus propios objetivos, necesitan de un mecanismo de comunicación que resuelva los posibles conflictos y los coordine (Toptal y Sabuncuoglu 2009). Las decisiones locales en las diferentes partes del sistema son integradas mediante mecanismos de coordinación y comunicación.

El programa global se construye como combinación de los programas individuales de cada uno de los elementos que componen el sistema. La base de estos sistemas son: por una parte, los mecanismos de control y programación interna de cada individuo; y por otra, los mecanismos de coordinación que permiten obtener un programa global. El resultado es un programa de producción que se actualiza en tiempo real cuando las condiciones en el sistema cambian. La clave está en encontrar una asignación aceptable considerando que los agentes locales no pueden, o no desean, compartir toda la información de la que disponen.

Si los comparamos con los sistemas centralizados, los sistemas de programación distribuidos producen, en general, peores resultados cuando las condiciones son estables. Sin embargo, ocurre lo contrario en entornos muy dinámicos: dado que la mayoría de los programas de producción en entornos de fabricación reales no pueden ejecutarse según lo establecido, debido a la aparición de cambios en las condiciones iniciales del sistema (una nueva orden de trabajo, cancelación de una orden no programada, una máquina averiada, falta de material, ...), la obtención de programas no óptimos no es un factor tan relevante (Frey et al 2003, Babiceanu et al 2005).

Uno de los problemas que se ha encontrado con los sistemas clásicos de programación de operaciones es el gap existente entre la investigación y la resolución de problemas reales. La programación de operaciones distribuida trata de acercarse más a la resolución de estos problemas. Entre los objetivos que se buscan está la posibilidad de obtener respuestas más rápidas a los distintos sucesos que pueden tener lugar.

- Ventajas e inconvenientes

Los sistemas distribuidos tienen como ventaja que pueden ser aplicados en entornos no colaborativos, es decir, en sistemas compuestos por entidades con objetivos en conflicto que pueden funcionar de un modo competitivo. Esto puede ser aplicado, por ejemplo, a las distintas subcontratas que tiene una empresa, cada una con distintas capacidades y que proponen sus programas para completar las tareas pendientes.

Debido a su modularidad los sistemas distribuidos son más flexibles, pudiendo adaptarse a los cambios en el entorno. Además esta modularidad permite un mantenimiento del sistema menos costoso, aunque el diseño de todo el sistema puede complicarse en la definición de las interrelaciones de los elementos que lo componen.

Por otra parte, aunque tienen un mantenimiento más fácil debido a su modularidad, su desarrollo es mucho más complejo que en los sistemas centralizados, principalmente pon los aspectos ligados a la comunicación.

Además, los sistemas distribuidos son más inestables que los sistemas de tipo centralizados. Una de las líneas de investigación en estos sistemas trata de encontrar mecanismos de coordinación que garanticen el funcionamiento del sistema.

- Implementación

El desarrollo de los sistemas multiagente ha permitido aplicar esta tecnología a sistemas tradicionalmente centralizados como la programación de operaciones. Los sistemas basados en agentes permiten representar los recursos productivos como agentes inteligentes, que pueden estar interconectados a través de una intranet. En estos sistemas es posible utilizar estructuras diferentes a los sistemas centralizados clásicos. Permiten definir un sistema distribuido de programación donde los agentes que representan a un determinado recurso o trabajo pueden gestionar el plan local y conjuntamente construir un programa global mediante negociación y coordinación con otros agentes (Shen et al 2006).

En el siguiente punto se introducen los conceptos de agentes y sistemas multiagente y su relación con los sistemas de programación de operaciones distribuidos.

3.2. Sistemas multiagente

Los sistemas multiagente son una rama de la inteligencia artificial distribuida que nace como medio para resolver problemas complejos. Este paradigma aparece al principio de los años 80 del pasado siglo, siendo a partir de mediados de los 90 cuando comienza su mayor desarrollo. En ellos se aplica la máxima de *divide y vencerás*. La idea básica radica en que es más sencillo diseñar y mantener muchas unidades de software que resuelvan pequeños problemas que un único software que resuelva un problema complejo. Consideran la presencia de varios responsables en la toma de decisiones, que interactúan y cooperan para alcanzar un objetivo global. Utilizan modelos de las ciencias sociales y naturales, como la biología, la física o la sociología y permiten construir sistemas para resolver problemas complejos de forma distribuida. Los sistemas multiagente han sido aplicados en diversos ámbitos que van desde el comercio electrónico, robótica o la comunicación, a la producción industrial (Wooldridge 2002).

La programación distribuida de operaciones ha estado ligada al desarrollo de los sistemas multiagente. Su naturaleza distribuida, así como el hecho de que posean propiedades tales como flexibilidad, fiabilidad, adaptabilidad y reconfigurabilidad, hacen de ellos unos candidatos adecuados como herramienta de modelado e implementación de sistemas de programación de operaciones distribuida. La aplicación de los avances en sistemas multiagente ha potenciado su desarrollo y aplicación. (Giret y Botti 2009, Rajabinasab y Mansour 2011). Shen (2002) destaca su adecuación debido a que poseen las siguientes características:

- Es un paradigma que utiliza la computación paralela, lo que puede aportar eficiencia y robustez.
- Permite integrar los distintos niveles de planificación, lo que permite una optimización más completa del sistema.
- Los agentes pueden obtener de forma directa datos de las entidades a las que representan, lo que permite una mejor respuesta en entornos dinámicos.
- Los mecanismos utilizados por estos sistemas son escalables, permitiendo utilizar los mismos mecanismos utilizados para programar la producción para coordinar los programas dentro de la cadena de suministro.
- Es posible compensar los beneficios locales por el beneficio global a través de sistemas de cooperación.

- Estos sistemas permiten incorporar otras técnicas relacionadas con la toma de decisión y aprendizaje.

Los sistemas multiagente han sido utilizados en ámbitos como el diseño de producto y la planificación de procesos, planificación y programación de la producción, control de la producción y sistemas holónicos de producción, así como en colaboración entre empresas. Las principales aplicaciones de los sistemas basados en agente en los sistemas de producción han sido descritas por Shen (2002), Monostori et al (2006), Shen et al (2006) y Lee y Kim (2008). En la Tabla 3.1 se muestran algunos ejemplos.

Ámbito de aplicación	**Aplicaciones**
Diseño de producto y planificación de procesos	PACT (Cutkosky et al 1993), SHADE (Mcguire et al 1993), Fist-link (Park et al 1994), ABCDE (Balasubramanian et al 1996), RAPID (Parunak et al 1999), Facilitator (Sun et al 2001). En planificación de procesos (Feng et al 2005).
Planificación y programación de la producción	YAMS (Parunak 1987, 1996), Metamorph (Maturana y Norrie 1996), (Liu y Sycara 1998), (Sun *et al.* 2001), (Caridi y Cavalieri 2004), (Shen et al 2006), (Babiceanu y Chen 2006), (Araúzo 2007), (Leitão 2009)
Control de la producción y sistemas holónicos de producción (HMS: Holonic Manufacturing Systems)	(Van Brussel et al 1998), (Bussmann y Schild 2001), (Wang et al 2001), (Brennan et al 2002), (Valckenaers y Van Brussel 2005) , (Trentesaux 2009).
Colaboración entre empresas: cadena de suministro, empresa extendida y empresa virtual	(Swaminathan et al 1998), (Fox et al 2000), (Sanz 2008).

Tabla 3.1. Aplicaciones de los sistemas basados en agente en el ámbito de la producción

En lo que sigue se revisarán los conceptos fundamentales utilizados en el paradigma de sistemas basados en agente, desarrollado de forma más detallada en Wooldridge y Jennings (1995), Weiss (1999), Ferber (1999), Luck et al (2005) y en Wooldridge (2002).

3.2.1. Concepto de agente y propiedades

Se pueden encontrar numerosas definiciones del concepto de agente en la literatura. Las diferencias entre ellas provienen principalmente del dominio de aplicación y de la complejidad del sistema:

- "Un agente es una entidad física o virtual movida por una serie de tendencias, objetivos de diseño o una función de utilidad a optimizar, que posee determinados recursos y tiene una representación parcial de su entorno. Su comportamiento se dirige a satisfacer estos objetivos, teniendo en cuenta los recursos que posee y sus percepciones, sus representaciones y sus comunicaciones" (Ferber 1999).
- "Un agente es un sistema computacional situado en un entorno y que es capaz de realizar una acción de modo autónomo dentro de ese entorno para conseguir sus propios objetivos" (Jennings y Wooldridge 1998).
- "Un agente es cualquier objeto computacional situado en un determinado entorno que es capaz de actuar de forma flexible y robusta para conseguir sus objetivos" (Jennings 2000).
- "Un agente es una entidad situada en un entorno que recibe información del mismo y, a partir de esa información, realiza acciones que implican cambios en el entorno" (Belmonte et al 2005).
- En el entorno de los sistemas distribuidos de producción: "Un agente es un software que se comunica y coopera con otros para resolver un problema que está más allá de las posibilidades de resolución de cada agente individual" (Shen et al 2006).

A la hora de definir los agentes, también se les otorga una serie de propiedades. De ellas, las principales son (Nwana 1996):

- ❖ Autonomía: los agentes pueden operar sin intervención humana ni de otros agentes.
- ❖ Capacidades sociales: un agente puede interactúan con otros agentes, comunicándose de forma síncrona o asíncrona.
- ❖ Racionalidad: un agente es capaz de utilizar la información percibida para establecer los planes con los que aproximarse a sus objetivos.
- ❖ Reactividad: es capaz de percibir los estímulos del exterior y responder en un tiempo adecuado a cambios del entorno en el que se encuentra situado.
- ❖ Proactividad: su comportamiento se orienta a conseguir sus objetivos.

3.2.2. Concepto de sistemas multiagente

Un sistema multiagente está formado por una red de agentes que interactúan y se comunican entre sí. Weiss (1999) lo define como: "Un conjunto de agentes que evolucionan en un entorno común y que se relacionan para conseguir unos objetivos comunes o realizar un conjunto de acciones". En un sistema multiagente se busca crear sinergias a partir del comportamiento individual de los agentes que lo componen, de forma que el sistema desarrolle propiedades que no pueden ser derivadas de forma directa de las propiedades de cada agente. Este comportamiento emergente que resulta de las interacciones entre agentes, permite resolver problemas que están fuera del alcance de resolución del agente individual. La capacidad para resolver un problema proviene de la emergencia de un comportamiento global a partir de las acciones individuales de los agentes (Monostori et al 2006).

Estos sistemas se caracterizan por: (1) cada agente no dispone de información completa sobre el problema ni tiene capacidades suficientes para resolverlo, (2) no hay un control global del sistema, (3) los datos están descentralizados, y (4) la computación es asíncrona (Jennings et al 1998).

3.2.3. Herramientas y estándares

Para el desarrollo de los sistemas basados en agentes es fundamental disponer de metodologías de desarrollo que proporcionen los procedimientos a seguir para construir soluciones de forma sistemática. Estas metodologías se centran principalmente en el análisis y diseño de sistemas multiagente. Sus objetivos son la definición y descripción del problema, la construcción de la solución y la comprobación de su funcionamiento. Algunas de las más importantes son AgentUML (Odel et al 2001), Gaia (Wooldridge et al 2000), MaSE (De Loach y Wood 2001), MAS-CommonKADS ((Iglesias et al 1998), INGENIAS (Pavón et al 2005).

Asimismo, también es necesario disponer de plataformas de diseño e implementación que dispongan de las funcionalidades básicas necesarias (Shen et al 2006). La Tabla 3.2 muestra algunas de las diferentes plataformas que se han desarrollado y que más aceptación tienen en la actualidad. Estas plataformas proporcionan los medios para poder implementar de forma más sencilla un sistema multiagente.

Algunas de estas plataformas de desarrollo siguen los estándares marcado por FIPA (*Foundation for Intelligente Physical Agents).* FIPA es una organización dedicada a la estandarización y cuyo objetivo es el desarrollo y la especificación de la tecnología de los agentes. Las especificaciones FIPA proporcionan un conjunto de estándares para que los agentes puedan interactuar a diferentes niveles. Estas especificaciones incluyen las referentes a la arquitectura (*FIPA Agent Architecture Specification*) y el modelo de gestión de la plataforma (*FIPA Agent Management Specification*).

Nombre	Lenguaje	Estándar	¿Libre?	Referencia
JADE	**Java**	**FIPA**	**Sí**	**(Bellifemine et al 2008)**
Jadex	**Java**	**FIPA**	**Sí**	**(Pokahr et al 2005)**
Cougaar	**Java**	**No**	**Sí**	**(Helsinger y Wright, 2005)**
Agent factory	**Java**	**FIPA**	**Sí**	**(Ross et al 2005)**
JACK	**Java**	**No**	**Sí**	**(Winikoff 2005)**

Tabla 3.2. Plataformas de desarrollo de agentes (Vallejo et al 2010)

También existen herramientas de simulación que se pueden utilizar para comprobar el comportamiento de sistemas multiagente. Algunas como Swarm (Swarm 2012) y Mason (MASON 2012), proporcionan librerías que pueden ser usadas en los modelos de programación. Repast (Repast Suite 2012), ofrece herramientas para la construcción rápida de modelos basados en agentes. Netlogo (Netlogo 2012) está formado por un lenguaje de programación y un conjunto de librerías, así como por un entorno de programación. Netlogo proporciona servicios de programación, como también hacen RePast y Swarm, pero además ofrece un entorno gráfico que permite construir interfaces rápidamente para poder ejecutar los modelos (Berryman y Angus 2010). Por su facilidad de programación, y sus buenos resultados en relación a los objetivos fijados, en este trabajo se utilizará Netlogo como herramienta de desarrollo.

3.3. Modelado basado en agentes de un problema de programación de operaciones distribuido

A la hora de modelar un sistema de programación de operaciones mediante agentes, las principales cuestiones a concretar son: (1) ¿cómo se va a descomponer el problema principal en distintos subproblemas?, (2) ¿cuáles son las relaciones jerárquicas que se establecen entre las distintas unidades que componen el sistema?, y (3) ¿cuáles son los mecanismos que utilizan las unidades del sistema para interactuar? (Toptal y Sabuncuoglu 2009). A continuación se desarrollan estos aspectos.

3.3.1. Descomposición del problema en subproblemas

Un aspecto fundamental que define un sistema distribuido es cómo se divide en subproblemas. Básicamente existen dos tipos de descomposición: la descomposición funcional y la descomposición física. Ambas no son excluyentes y es posible encontrar sistemas en los que se ha realizado una descomposición mixta (Shen 2002):

- Descomposición funcional: los agentes tienen asignadas una de las funciones necesarias para que el sistema funcione, adquisición de órdenes, planificación, programación, gestión de materiales o gestión del transporte. No hay una relación directa entre el agente y ninguna entidad física. Aunque este tipo de descomposición es menos frecuente, se ha aplicado por parte de Ryu y Jung (2003) que, utilizando el modelo de sistemas fractales, diseñan un sistema compuesto por un conjunto de agentes (observer, analyzer, resolver, reporter y organizer) caracterizados por realizar una función específica de cada fractal. Por su parte, Leitão y Restivo (2008) descomponen las funciones de programación en distintas entidades orientadas a la reprogramación, utilizando un mecanismo que evoluciona dinámicamente para combinar estrategias centralizadas y distribuidas.
- Descomposición física: se vale de los agentes para representar las entidades presentes en el sistema físico, como máquinas, herramientas, operaciones o componentes. Existe una relación directa entre el agente y la entidad física. Este tipo de descomposición se ha seguido en trabajos como los de Parunak (1987), Fox et al (2000), Shen et al (2000), Parunak et al (2001), Sadeh et al (2003),

Wallace (2003) y Yen y Wu (2004). Destaca la descomposición utilizada en la arquitectura PROSA (Van Brussel et al 1998), en la que la descomposición hace referencia a: (1) los recursos, donde las funciones asignadas están relacionadas con un recurso o un grupo de ellos; (2) las tareas, donde la representación está ligada a una o un grupo de operaciones que buscan un recurso al que ser asignado para conseguir un determinado objetivo.

3.3.2. Relaciones jerárquicas

Hacen referencia a la arquitectura del sistema que proporciona el marco de relaciones entre las unidades de decisión que lo componen. En los sistemas dinámicos la arquitectura tiene gran importancia, ya que va a determinar el tiempo de respuesta y la aproximación al óptimo de la planta. La clasificación más sencilla distingue entre sistemas jerárquicos y sistemas heterárquicos. Entre esos dos extremos se encuentran los sistemas cuasi-heterárquicos (Toptal y Sabuncuoglu 2009). Se pueden clasificar las arquitecturas en función de las relaciones jerárquicas que utilizan. Los principales tipos son:

3.3.2.1. Arquitecturas de tipo jerárquico

Existe un decisor global que desarrolla un programa para todo el sistema, mientras que otros decisores ejecutan instrucciones. Esta jerarquía puede desarrollarse en un solo nivel o en varios niveles. Muchos de los sistemas distribuidos basados en agentes son sistemas jerárquicos, ya que replican la descomposición funcional de los sistemas que representan.

Asume que hay una relación maestro/esclavo entre los niveles superior e inferior. Existe un flujo ascendente, desde los niveles inferiores a los superiores, con los datos obtenidos por los sensores que sirven de base para que los niveles superiores tomen las decisiones y envíen las instrucciones en un flujo descendente hacia los niveles inferiores (Figura 9).

Estos sistemas tienen como inconveniente que ignoran elementos tales como la complejidad y la incertidumbre que caracterizan el mundo real. La lenta respuesta que tienen, hace que los datos utilizados para la toma de decisiones no tengan nada que ver (y por tanto no sirvan para caracterizar) con los que existen realmente en el momento de la aplicación de la decisión. Además, las arquitecturas centralizadas son inflexibles,

costosas de mantener y lentas para poder gestionar problemas con perturbaciones de distintos tipos.

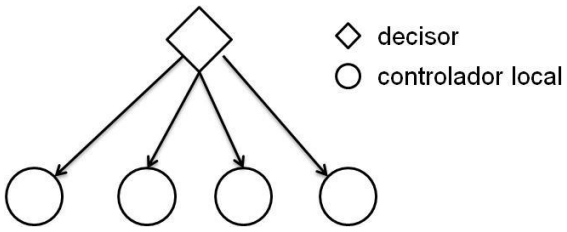

Figura 9. Estructura jerárquica en un solo nivel

Sistemas como CIM (*Computer Integrated Manufacturing*) son un ejemplo de este tipo de arquitectura. Con CIM se busca la completa automatización de la planta de producción integrando las funciones de producción y de negocio mediante la aplicación de la tecnología. Todos los procesos son controlados mediante sistemas informáticos. Sin embargo, la planificación y programación de tareas es totalmente centralizada y secuencial. Los sistemas CIM implican grandes inversiones y estructuras muy complejas que pueden llevar a la generación de sistemas extremadamente rígidos (Christo y Cardeira 2007).

3.3.2.2. Arquitecturas de tipo heterárquico

Es el sistema opuesto al centralizado, donde cada decisor local toma sus propias decisiones y ha de coordinarse directamente con el resto de decisores locales del sistema para elaborar un programa común. En ellas no existe una jerarquía directa sino que los decisores se comunican bien a través de un coordinador, bien por pares para llegar a un acuerdo.

En los sistemas puramente heterárquicos los decisores locales no comparten información. Además de tomar sus decisiones considerando solo información local, compiten por optimizar sus propios objetivos, sin tener en cuenta el objetivo global.

Las arquitecturas heterárquicas se caracterizan por considerar únicamente las relaciones entre las unidades de un determinado nivel (para mejorar la flexibilidad), ignorando las relaciones existentes entre las unidades de los niveles superiores e inferiores (Figura 10).

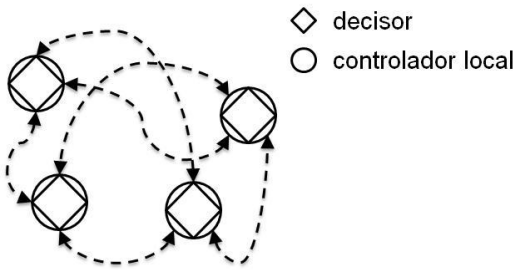

Figura 10. Estructura heterárquica básica

Este enfoque puede aplicarse en las situaciones en las que en el proceso de asignación intervienen distintos sujetos (agentes), cada uno representando a distintos trabajos, y no existe una autoridad central que pueda resolver los posibles conflictos en el uso de los recursos que puedan presentarse. En este caso el modelo, más que buscar soluciones optimas, tratará de proporcionar medios para el proceso de negociación que permitan una asignación aceptable y estable (Agnetis et al 2007).

Los mayores inconvenientes de este tipo de arquitectura son la falta de predictibilidad en su funcionamiento y la falta de visión global, ya que por definición no existe ningún elemento que posea información global.

3.3.2.3. Arquitecturas cuasi-heterárquicas

Estas estructuras se encuentran a medio camino entre las dos anteriores. En ellas los decisores locales elaboran sus propios programas. Existe un decisor *maestro* que recoge los distintos programas locales y resuelve los conflictos potenciales que se puedan producir. Incorpora las ventajas de los sistemas heterárquicos, como son la capacidad de reacción ante eventos no esperados, así como un alto nivel de rendimiento. Por otro lado, se incluyen elementos de tipo central para coordinar al resto de entidades locales. Estos sistemas desarrollan el concepto de jerarquía en sistemas distribuidos. La toma de decisiones individual por parte de las entidades individuales se mantiene, pero éstas pueden pedir consejo a la entidad central. Con ello se consigue combinar la reactividad de las entidades locales con la eficiencia de utilizar sistemas centralizados (Figura 11).

Estas estructuras pueden desarrollarse en un único nivel o en múltiples niveles. Las estructuras en un solo nivel son similares a las estructuras heterárquicas, pero en ellas los decisores locales disponen de suficiente información global para resolver

conflictos en el sistema. Una de las primeras propuestas es la de Ramos (1994) que crea un agente mediador de tareas y otro mediador de recursos que son los que negocian la ejecución de las tareas.

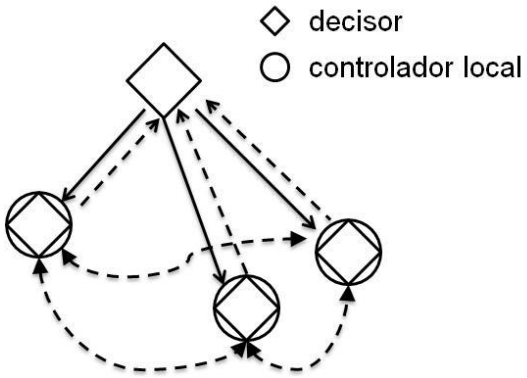

Figura 11. Estructura cuasi-heterárquica básica

Dentro de los sistemas cuasi-heterárquicos multinivel pueden englobarse arquitecturas alternativas que han surgido en los últimos años como son los sistemas holónicos, los sistemas biónicos y los sistemas fractales (Christo y Cardeira 2007, Tharumarajah 1996).

- Sistemas fractales

Los sistemas fractales de producción tienen su origen en el concepto matemático de fractal que, básicamente, sirve para describir geometrías complejas cuya estructura básica se repite a varias escalas. El sistema de producción se describe en estos términos, utilizando unidades fractales autosimilares. Cada unidad fractal se caracteriza por las propiedades de autoorganización, autosimilitud y autooptimización. Cada unidad fractal actúa de forma independiente para conseguir sus propios objetivos, pero también deberá mantener coherencia con la meta global del sistema mediante la cooperación continua con las otras unidades (Ryu y Jung 2003).

- Sistemas biónicos

Los sistemas biónicos de producción están basados en la biología. Estos sistemas buscan un paralelismo entre los organismos vivos y los sistemas de producción. Los organismos vivos son vistos como sistemas naturales organizados formados por

unidades básicas (las células). La coordinación entre las células se produce mediante procesos químicos activados mediante las enzimas. Con ello se consigue llevar a cabo la función del órgano del que forman parte estas células. Este proceso es tomado como modelo y reproducido para gestionar un sistema de producción (Leitão, 2004).

- Sistemas holónicos

Los sistemas holónicos son un paradigma que traslada al ámbito de la producción los conceptos desarrollados por el filósofo Arthur Koestler (Koestler 1967) acerca de los seres vivos y los organismos. El sistema productivo se caracteriza mediante agrupaciones organizadas llamadas holarquías de entidades independientes y cooperativas relacionadas entre sí llamadas holones (Figura 11). La holarquía define las reglas mínimas para la cooperación entre holones y por tanto limita su autonomía. Un holón es un elemento identificable de un sistema de producción para transformar, transportar, almacenar o validar información o elementos físicos. El holón está formado por una serie de subelementos y puede formar parte de otro holón (Van Brussel et al 1998).

Los holones se caracterizan por: (1) su autonomía, o capacidad de una entidad para crear y controlar sus propios planes y estrategias sin necesidad de recibir órdenes de una entidad superior; y (2) la cooperación, lo que permite que varias entidades creen y ejecuten un plan de mutuo acuerdo para conseguir un objetivo común.

Este paradigma se fundamenta en dos observaciones básicas: (1) los sistemas complejos evolucionan más rápido desde sistemas simples cuando aparecen formas intermedias estables, que si estos estados intermedios no se dan; y (2) no existen en un sentido absoluto los conceptos de "todo" y de "parte", ya que los holones son *todos* formados por distintas partes, y ellos a su vez forman partes de otro sistema (otro *todo*) de nivel superior (Valckenaers et al 1994, Bongaerts et al 2000, Valckenaers y Van Brussel 2005).

3.3.3. Mecanismos de coordinación utilizados

Uno de los aspectos fundamentales a la hora de diseñar un sistema distribuido es definir la forma en que se relacionan las distintas unidades de decisión que lo componen. Estas relaciones son las que van a permitir la coordinación entre las decisiones individuales que conducirán a un programa global. A continuación se muestran los mecanismos de coordinación más utilizados:

- Dominación:

En este tipo de mecanismo un decisor toma las decisiones sobre otro agente, que implementará las acciones. Este mecanismo se utiliza tanto en sistemas centralizados como en muchos de los sistemas descentralizados. El grado de dominación en un sistema descentralizado depende del grado de descentralización que se ha establecido (Toptal y Sabuncuoglu 2009).

- Utilización de reglas

Los agentes utilizan reglas que son las que marcan la pauta de funcionamiento. El programa aparece al aplicar estas reglas tanto al proceso de asignación como de secuenciación. Las reglas de prioridad han sido ampliamente estudiadas en distintos trabajos como hemos hecho referencia en el capítulo 2. También el proceso de *stigmergy* de los métodos de optimización basados en colonias de hormigas se basa en conseguir el funcionamiento global del sistema a partir del comportamiento de los individuos, basado en reglas simples (Hadeli et al 2004, Xiang y Lee 2008) .

- Cooperación:

En un mecanismo de cooperación, agentes del mismo nivel colaboran para obtener un mejor funcionamiento del sistema. En este mecanismo se ponderan los objetivos locales con el objetivo global. La coordinación se produce cuando varios agentes interactúan para conseguir un objetivo común. Estos agentes deben tener objetivos interrelacionados, recursos compartidos y un entorno común. Además deben ser benevolentes, de modo que los agentes tratarán de ayudarse siempre que sea posible. Los agentes deben coordinar sus actividades para conseguir maximizar el objetivo marcado estando limitada la cantidad de información que pueden intercambiarse. Esta cooperación puede llevarse a cabo compartiendo planes, o intercambiando tareas. Existen situaciones en la negociación en las que se pueden formar coaliciones, que incluyan a algunos de los agentes mientras que excluyan a otros y que, de esta forma, sea posible que los agentes que forman la coalición compartan sus recursos y coordinen sus acciones, aumentando así sus beneficios como proponen Nof y Weill (1992). Kouiss et al (1997) realizan una propuesta de este tipo donde los agentes recurso seleccionan de forma colaborativa las reglas de prioridad más adecuadas en un entorno dinámico. Lu y Yih (2001) proponen un mecanismo cooperativo mediante el intercambio de distintos índices calculados por cada agente. Archimede y Coudert (2001) proponen un marco colaborativo que puede ser aplicado

a diferentes problemas de planificación y programación de operaciones. El-Bouri (2012) propone elegir las reglas de prioridad a aplicar de forma cooperativa.

- Negociación:

En el mecanismo de negociación, agentes del mismo nivel jerárquico se comunican para intercambiar información que les permita obtener mejores programas para sus subproblemas. El objetivo principal es la optimización de los objetivos locales, normalmente incompatibles entre sí, sin tener en cuenta el funcionamiento global del sistema. Los agentes no son benevolentes y se coordinan mediante mecanismos de negociación por la existencia de algún conflicto que debe ser resuelto de una forma descentralizada. La investigación en este campo se centra en diseñar mecanismos que modifiquen el comportamiento de cada agente individual de forma que se genere el comportamiento global deseado.

Muchos de los protocolos de negociación implementados están basados en el *contract net protocol* (Smith 1980). Éste es un protocolo de interacción para la resolución de problemas de forma distribuida entre agentes que posibilita la asignación dinámica de tareas, permite a los agentes hacer ofertas sobre múltiples tareas al mismo tiempo. De forma resumida se puede describir como sigue: (1) un agente manager quiere subcontratar un trabajo y envía un mensaje a otros agentes (subcontratantes), (2) los agentes interesados en hacer el trabajo envían un mensaje comunicando sus ofertas, y (3) pasado un tiempo el agente manager selecciona los agentes que hayan realizado las mejores ofertas y les envía un mensaje comunicando su aceptación.

Uno de los enfoques más utilizados es plantear el problema de *scheduling* como una economía cerrada en la que participan tanto las órdenes de trabajo como las máquinas. Wellman et al (2001) utilizan la teoría del equilibrio general walrasiano, que proporciona un método distribuido para una asignación eficiente de bienes y recursos entre los agentes basada en precios de mercado. El modelo equipara el sistema multiagente con un mercado. Para ello las actividades de interés son clasificadas en términos de producción y consumo de bienes. Los agentes pueden adquirir recursos en diferentes mercados. El proceso finaliza cuando los mercados alcanzan un equilibrio general y los precios se mantienen. El equilibrio general se alcanza si cada productor maximiza sus beneficios y cada consumidor maximiza sus preferencias dados los precios, y el mercado se vacía, es decir, la producción iguala al consumo. Por otra parte, autores como Lin y Solberg (1992), Dewan y Joshi (2002) y

Araúzo (2007) utilizan el mecanismo de subasta para resolver el problema. Este mecanismo será revisado con más detalle en el siguiente capítulo.

3.4. Conclusiones del capítulo

Los sistemas distribuidos de programación de operaciones aparecen como una alternativa a los sistemas clásicos de tipo jerárquico y centralizado. Estos sistemas proporcionan una mejor respuesta en entornos dinámicos, lo que permite salvar la distancia existente hasta hace pocos años entre la investigación y los problemas reales. Los sistemas multiagente son una herramienta adecuada para el desarrollo de sistemas distribuidos de programación de operaciones, y el desarrollo de la investigación en ambos ha ido en paralelo, potenciándose mutuamente.

Una de las características que se busca en los sistemas multiagente es el comportamiento emergente. Este comportamiento aparece por la interrelación de los agentes, no siendo asignable dicho comportamiento a ninguno de ellos de forma específica.

Es importante el estudio de las relaciones entre agentes ya que una de las condiciones deseables para estos sistemas es que su comportamiento sea predecible. En el siguiente capítulo nos centraremos en el estudio de las subastas como medio de interrelación entre los elementos del sistema multiagente. Las subastas son un mecanismo de mercado que permite coordinar los distintos agentes a través de los precios. Los precios que se generan en la subasta son la expresión de los intereses de los agentes y sirven como señales en la toma de decisiones.

Capítulo 4. Mecanismos de asignación distribuidos basados en subastas

La asignación basada en mecanismos de mercado es una de las ramas de la programación de tareas distribuida más activa en los últimos años. Trata de establecer un programa de producción basándose en los precios que emergen a partir de un proceso de pujas realizadas por una tarea o recurso para conseguir un objetivo individual (Kutanoglu y Wu 1999). Este paradigma ha sido aplicado para resolver problemas de asignación de recursos de distinto tipo, como programación de los aterrizajes y despegues en un aeropuerto en Rassenti et al (1982), problemas de flujos de mercancías en Bertsekas (1990) y gestión de los tramos de la red de ferrocarriles en Parkes y Ungar (2001).

También autores como Kutanoglu y Wu (1999), Wellman et al (2001) y Dewan y Joshi (2002) han propuesto el uso de mecanismos de mercado como medio de coordinación para determinar programas de producción de forma distribuida. Las subastas poseen una serie de características adecuadas para ser utilizadas como mecanismos de asignación en este tipo de sistemas, caracterizados por ser distribuidos y complejos. Lee et al (2003) proponen las siguientes razones para utilizar las subastas como mecanismo de coordinación en el modelado de sistemas de producción basados en agentes:

- Tanto los sistemas multiagente como las subastas son distribuidos por naturaleza.
- Cada elemento participante en el sistema de producción puede ser modelado como un agente auto-interesado, que busca sus propios objetivos y solo tiene acceso a su propia información.
- Los mecanismos basados en subastas, comparados con otras alternativas, reducen la carga de comunicación de la negociación. La información intercambiada está limitada en forma de pujas -de los agentes al subastador- y en forma de precios -del subastador a los agentes- (Agnetis et al 2007).
- Tanto el sistema basado en subastas como los sistemas multiagente permiten la modularidad, lo que es una ventaja en su implementación y mantenimiento.

La idea que subyace es que la asignación se realice por la interacción entre los agentes, recursos y tareas, que participan en un mercado creado ad-hoc y que establecen sus precios a través de un proceso iterativo de búsqueda del equilibrio. Esto permite que los cálculos complejos puedan ser distribuidos entre los distintos agentes participantes y realizarlos en paralelo, por lo que mejora la rapidez en su resolución, y aumenta la robustez del sistema (Wellman et al 2001).

Para definir el proceso de programación de operaciones como una subasta hay que responder a las siguientes preguntas: ¿qué se subasta? ¿quién puja por lo que se subasta? ¿qué tipo de subasta se va a implementar? ¿cómo se expresan las preferencias? ¿cómo se establecen los precios?. En este capítulo se dan respuestas a estas cuestiones: en primer lugar se estudian los principales tipos de subasta, después se revisan los principales trabajos relacionados con la aplicación de subastas a la programación de operaciones, a continuación se justifica el uso de las subastas combinatorias como modelo adecuado para representar un problema de programación de operaciones y, por último, se define el modelo y su algoritmo de resolución.

4.1. Tipos de subasta

Una *subasta* se define como "un mecanismo de envío de información, junto con reglas para asignar ítems y pagos a los participantes de acuerdo con la información enviada" (McAfee y McMillan 1987). Se pueden encontrar estudios sobre subastas en McAfee y McMillan (1987), Milgrom (1989), Klemperer (1999) y

Krishna (2009). Existen muchos tipos de subastas que se pueden caracterizar teniendo en cuenta aspectos como el número de participantes, si los bienes intercambiados son divisibles o no, el número de bienes, si los bienes son iguales o si están diferenciados, si las pujas que se realizan son conocidas o no por el resto de participantes, y por el precio al que finalmente se intercambia el bien, tal como se muestra en la Tabla 4.1.

Número de participantes	**Tipos de artículos**	**Número de artículos**	**Diferen-ciación**	**Forma en la que se realizan las pujas**	**Precio al que se intercambia**
▪ Un vendedor, muchos compradores (directa) ▪ Un comprador, muchos vendedores (inversa) ▪ Muchos compradores y muchos vendedores (doble)	▪ Divisible ▪ Discreto	▪ Un solo artículo (single) ▪ Varios artículos (múltiple)	▪ Idénticos ▪ Diferen-ciados	▪ Abiertas ▪ Cerradas	▪ Al primer precio ▪ Al segundo precio

Tabla 4.1. Caracterización de las subastas por el número de participantes y de artículos vendidos. Adaptado de Anandalingam et al (2005)

- Según el número de participantes, las subastas con un solo vendedor y muchos compradores se conocen como subastas directas (*forward auctions* o *demand auctions*), frente a éstas están las subastas con un solo comprador y varios vendedores que se conocen como subastas inversas o subastas de aprovisionamiento (*reverse auctions*, *supply or procurement auction*). Las subastas con varios compradores y varios vendedores son conocidas como subastas dobles (Figura 12).
- En cuanto a la forma en que se realizan las pujas (*bids*), las subastas pueden clasificarse en abiertas (*open auction*) cuando los participantes pueden pujar de forma repetida y conocen en todo momento cuales han sido las pujas previas en la subasta. En las subastas cerradas (*close auction*) los compradores (o vendedores) envían sus pujas, que no son conocidas por el resto de participantes en la subasta.

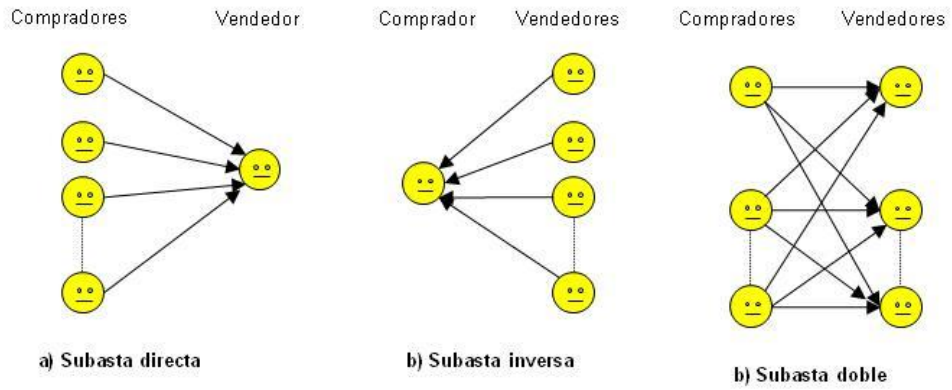

Figura 12. Caracterización de las subastas por el número de participantes y de artículos vendidos (Anandalingam et al 2005)

- En cuanto al precio al que se vende o compra el artículo subastado, puede ser fijado al primer precio más alto (*first price*) o al segundo precio más alto (*second price)*.
- Uno de los criterios más utilizados a la hora de clasificar las subastas es el número de artículos vendidos. Se distingue entre las subastas de un solo artículo (*single-item auctions*) y las de varios artículos (*multiple-item auction*)

En la Figura 13 se muestran los principales tipos de subasta con sus características básicas que se explican a continuación.

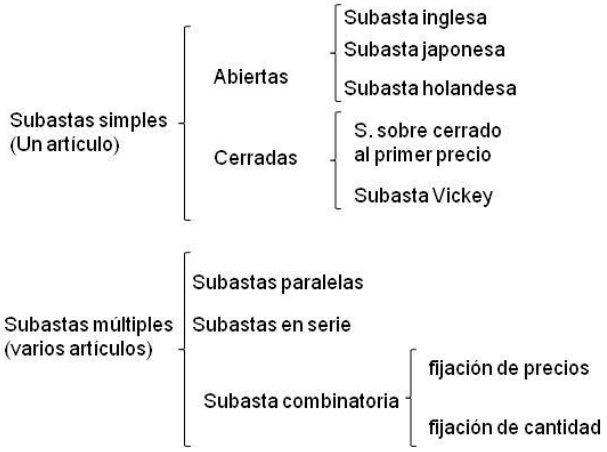

Figura 13. Principales tipos de subasta (adaptado de McAfee y McMillan (1987) y De Vries y Vohra (2003))

4.1.1. Subastas de un solo artículo

La subasta inglesa *(english auction or open ascending price auction*) es una subasta abierta y ascendente. En esta subasta, el precio aumenta sucesivamente hasta que permanezca en la puja un solo participante. El precio puede ser anunciado por el subastador o por los propios participantes. En este último caso, cualquier postor puede pujar con un precio mayor que el precio de puja actual. Cuando nadie más puja por un precio mayor, el último postor gana la subasta y paga por el precio pujado. A la versión en la que el subastador va aumentando el precio de la puja de forma gradual hasta que los participantes abandonan, se le denomina subasta japonesa. Un participante puede abandonar la subasta en cualquier momento si el precio es demasiado alto para él. La subasta finaliza cuando solo queda un postor, que gana y paga la puja al precio final.

La subasta holandesa (*dutch auction or open descending price auction*) es una subasta abierta y descendente. La subasta comienza a un precio alto que se va bajando gradualmente. Un postor puede demandar el ítem en cualquier momento al precio anunciado. La subasta finalizará en ese momento.

La subasta de primer precio en sobre cerrado (*first-price sealed-bid auction*) es una subasta de una sola ronda en la que inicialmente cada postor envía su puja de forma privada al subastador. Gana la puja más alta y paga por el precio pujado.

La subasta *Vickrey* es una subasta de segundo precio en sobre cerrado (*second-price sealed-bid auction*). Es idéntica a la subasta de primer precio, excepto que el precio que paga el ganador (el postor con la puja más alta) es el segundo precio más alto.

4.1.2. Subastas de varios artículos

En general, se puede decir que en las subastas de varios artículos es mucho más difícil obtener un resultado eficiente que en las subastas simples (Klemperer 1999). Al igual que las subastas simples, las subastas de varios artículos (*multiple-item auctions*) pueden ser directas, inversas o dobles. Una consideración importante es si los bienes intercambiados son independientes entre sí. En ese caso, los bienes pueden ser subastados de forma simultánea (subastas paralelas) o pueden ser subastados progresivamente en subastas que van teniendo lugar a medida que se termina la subasta del bien anterior (subastas secuenciales). Si los bienes no son independientes y

se adquieren por lotes, el valor de cada ítem está determinado por el resto de ítems del lote adquirido. Para este caso se utilizan las subastas combinatorias.

4.2. Aplicación de las subastas a problemas de producción

Las subastas se han aplicado a la resolución del problema general de programación de la producción, programación de sistemas de fabricación flexible, programación *job shop*, programación de proyectos, programación simultánea de varias plantas de producción, problema de secuenciación de una sola máquina, etc.

A la hora de modelar el sistema de programación de la producción de forma distribuida, la mayoría de autores utilizan una división física en la que las entidades representan trabajos y máquinas. Se pueden distinguir dos enfoques según el papel que realiza cada una de estas entidades en la subasta. En el primero, las máquinas tienen un papel activo y participan en la subasta para conseguir tareas que les permitan maximizar su utilidad. En el segundo enfoque, cada trabajo se representa como una entidad que participa en la subasta para conseguir las máquinas que necesita.

En la Tabla 4.2 se muestran algunos trabajos, indicando el enfoque que hacia el que se han orientado y el problema que resuelven. Algunos de ellos utilizan el marco conceptual de los sistemas holónicos: Márkus et al (1996) y de los sistemas multiagente: Lee et al (2003), Wu et al (2003), Siwamogsatham y Saygin (2004), Anussornnitisarn et al (2005), Agnetis et al (2004, 2007), Alvarez (2007).

A la hora de implementar el mecanismo de mercado, algunos trabajos, como los de Shaw (1988), Lin y Solberg (1992, 1994), Baker (1996), Wu et al (2003) y Anussornnitisarn et al (2005), utilizan el Contract Net Protocol propuesto por Smith (1980), Smith y Davis (1981) y Davis y Smith (1983). En este protocolo los participantes pujan por el ítem que se intercambia, por lo que también se conoce como protocolo basado en subastas de sobre cerrado. Por su parte, Wellman (1993) desarrolla para problemas de asignación de recursos de forma distribuida el mecanismo de subasta WALRAS, en honor a Léon Walras (1834-1910), primer economista en analizar, con un modelo matemático, el problema del equilibrio general de la competencia perfecta.

Problema resuelto	Las máquinas pujan por las tareas	Las tareas o los trabajos pujan por las máquinas
Problema general de programación de la producción	Lin y Solberg (1992, 1994), Márkus et al (1996),	Lin y Solberg (1992, 1994), Baker (1996), Wellman et al (2001), Wu et al (2003), Anussornnitisarn et al (2005)
Programación de sistemas de fabricación flexible	Siwamogsatham y Saygin (2004)	Shaw (1988), Tiwari y Allada (2004), Araúzo (2007)
Problema *Job shop*		Kutanoglu y Wu (1999), Dewan y Joshi (2001, 2002), Liu et al (2007), Jeong y Yim (2009)
Programación de proyectos	Lee *et al.* (2003)	Kumara et al (2002), Araúzo et al (2010)
Problema de secuenciación de una sola máquina		Dewan y Joshi (2000), Jeong y Leon (2005), Agnetis et al (2007)
Programación de una planta de producción real		Gou et al (1998)
Programación simultánea de varias plantas de producción		Vytelingum et al (2009), Tiwari et al (2010)

Tabla 4.2. Trabajos en el ámbito de la programación de la producción que utilizan las subastas como mecanismo de coordinación

Es interesante analizar el tipo de programación que se ha asociado a cada uno de los enfoques. En el primer enfoque las máquinas pujan por aquellas tareas que se van activando conforme se van completando las tareas precedentes, por lo que nos encontramos con un tipo de programación puramente reactivo. Sin embargo, en el segundo enfoque, la mayor parte de trabajos dividen el horizonte de fabricación en intervalos de tiempo iguales (*slots*) que son el bien por el que pujan los trabajos (Figura 14). Esto permite realizar una programación que realice con antelación las funciones de carga y temporización de tareas en cada máquina, por lo que se estará utilizando un enfoque reactivo-predictivo.

El modelo en el que los trabajos pujan por las máquinas es adecuado cuando la función objetivo del problema de programación es un sumatorio de una función

dependiente de variables asociadas con cada trabajo. En ese caso la función de utilidad de cada trabajo se puede definir a partir de la descomposición del sumatorio en cada uno de los sumandos asociados a cada uno de ellos.

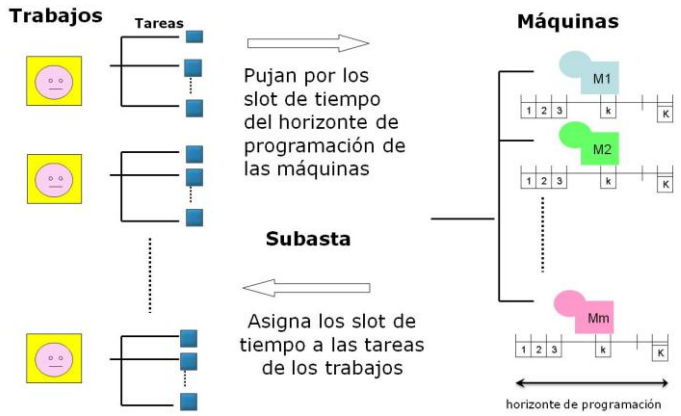

Figura 14. Subasta en la que las órdenes de trabajo pujan por las máquinas.

4.3. Subastas combinatorias y su aplicación a problemas de producción

Distintos autores como Duffie (1990), Kutanoglu y Wu (1999) y Wellman et al (2001) han propuesto las subastas combinatorias como medio de negociación y coordinación en los sistemas multiagente. Las subastas combinatorias son un tipo de mecanismo de mercado que se caracteriza porque los participantes realizan pujas por combinaciones de productos para las que tienen distintas valoraciones. Este tipo de subastas es adecuado para los casos en los que existan complementariedades entre los ítems, de forma que el valor de cada ítem dependerá de cuales sean los productos adquiridos en el mismo lote. Han sido aplicadas en la resolución de problemas distribuidos de asignación de recursos como en Rassenti et al (1982) o en Parkes y Ungar (2001). Un estudio completo de este tipo de subastas se puede encontrar en De Vries y Vohra (2003), Cramton et al.(2006) y Abrache et al (2007).

Si se define el problema de programación de operaciones como una subasta en la que las órdenes de trabajo (compradores) pujan por conseguir los *slot* de tiempo (bienes intercambiados) en que se divide el horizonte de programación de cada máquina (vendedores), entonces este problema presenta complementariedades, ya que

los elementos que se intercambian están ligados por las restricciones de programación. Estas complementariedades son debidas a:

1. Restricción de continuidad o de no interrupción de la tarea (*non-preemption*). Esta restricción obliga a que, una vez comenzada la realización de una operación, ésta permanecerá en la máquina hasta que se haya completado. Ninguna otra tarea puede ser programada en una máquina entre los momentos de comienzo y de fin de una operación. Para poder realizar una operación se ha de pujar sobre conjuntos de *slot* continuos y de la misma máquina (Figura 15).
2. Restricciones de precedencia, que determinan el orden en que han de ser realizadas las distintas operaciones de un trabajo. Una tarea no puede ser comenzada hasta que no se hayan realizado sus tareas precedentes (Figura 16).

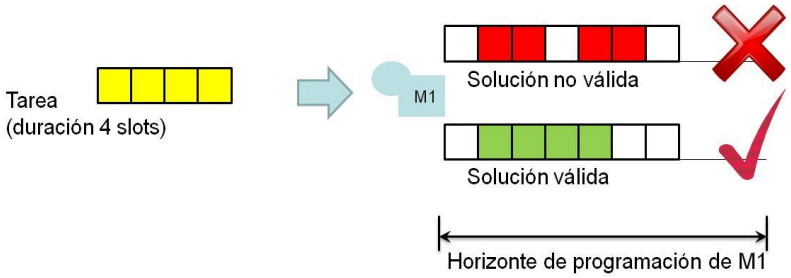

Figura 15. Complementariedad debida a las restricciones de no interrupción de la tarea (non-preemption)

4.3.1. Características de las subastas combinatorias

Las subastas combinatorias son un tipo de subastas de varios artículos en las que las pujas se realizan sobre lotes de productos y en las que existen complementariedades. En ese caso, las subastas combinatorias permiten superar las ineficiencias económicas de las subastas paralelas y secuenciales (Sandholm 2000). El estudio de estas subastas es complejo ya que las complementariedades entre los ítems hacen que la función de utilidad no sea regular (Anandalingam et al 2005).

Al igual que en las subastas simples, las subastas combinatorias se pueden clasificar en las directas -participan un solo vendedor y múltiples compradores-, o inversas -participan un solo comprador y múltiples vendedores- y dobles -en las que participan múltiples compradores y múltiples vendedores-.En este último caso, las

subastas combinatorias se denominan intercambios combinatorios (*combinatorial exchanges*).

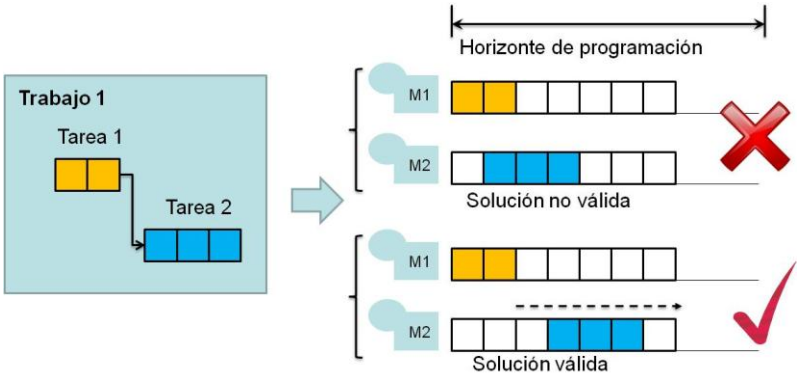

Figura 16. Complementariedad debida a las relaciones de precedencia entre tareas de un mismo trabajo.

En cuanto al número de rondas que se realizan, se distingue entre subastas combinatorias en una sola ronda (de sobre cerrado), donde los participantes dan sus valoraciones sobre las combinaciones de productos una sola vez y las subastas combinatorias en varias rondas o iterativas.

Las subastas combinatorias iterativas tienen ventajas sobre las de una sola ronda: en primer lugar no es necesario que cada participante realice pujas sobre todo el conjunto de combinaciones posible, sino que puede hacerlo únicamente sobre aquellas en las que esté interesado. Además, cada participante termina revelando sus preferencias –que es información privada relevante para el resto de participantes-. Además se adaptan bien a los entornos dinámicos -como es el caso de los entornos productivos- donde tanto los participantes como los elementos a subastar van entrando y saliendo en la subasta en momentos diferentes (De Vries y Vohra 2003).

Existen dos tipos básicos de subastas combinatorias iterativas: las de fijación de cantidades y las de fijación de precios. La diferencia está en la forma en que los licitadores expresan sus preferencias:

- Subastas iterativas de fijación de cantidades

En las subastas iterativas de fijación de cantidades, los participantes expresan sus preferencias sobre las distintas combinaciones de ítems subastados enviando el precio que estarían dispuestos a pagar por cada combinación. El subastador realiza una

asignación provisional de los artículos que depende de los precios enviados. Los participantes van ajustando los precios en cada iteración. Este es el método que utiliza *ibundle* de Parkes (1999). Existen varios lenguajes que permiten expresar las preferencias de los licitadores como los lenguajes OR y XOR (Cramton et al 2006).

- Subastas iterativas de fijación de precios

En las subastas iterativas de fijación de precios, el subastador asigna un precio a cada uno de los artículos a subastar. Cada participante envía como puja la combinación de elementos que pretende adquirir con los precios dados. El subastador va adaptando los precios en cada iteración en base a la demanda de cada artículo. Aquí aparecen distintas formas de establecer los precios:

- Subasta ascendente, también denominada *clock auction,* en la que el subastador anuncia precio, uno para cada uno de los ítems vendidos. Los participantes en la subasta indican las cantidades que quieren adquirir de cada ítem al precio dado. Se incrementan los precios de los ítems para los que exista un exceso de demanda. Los participantes en la subasta expresan de nuevo las cantidades que quieren adquirir de cada ítem al nuevo precio. La subasta continua mientras exista exceso de demanda para alguno de los ítems subastados. La subasta llega a su fin cuando existe una puja para cada unidad de cada ítem disponible y los bienes son entregados a los precios fijados en la iteración. En estas subastas el precio nunca decrece (Ausubel et al 2006, Porter et al 2003). Forde y Doyle (2008) proponen un algoritmo que implementa estos pasos y se muestra en la Figura 17. El atractivo de las *clock auctions* es su simplicidad y los requerimientos mínimos de procesado computacional. Sin embargo, este tipo de subasta no asegura la consecución del equilibrio, ni proporciona cotas a la solución del problema (Cramton et al 2006).

- Subasta en la que los precios varían en función del exceso de demanda o de oferta, aumentando o disminuyendo en cada ronda. El valor en que se varían los precios puede establecerse utilizando un método de cálculo de precios walrasiano no adaptativo o mediante un proceso de ajuste adaptativo. En el primer caso, el incremento del precio permanece constante en todas las iteraciones. En el segundo, el incremento del precio se establece como función del exceso de oferta o demanda. En este caso tampoco existe garantía de que se

consiga el equilibrio pero, en determinadas condiciones, es posible determinar cotas para el mismo.

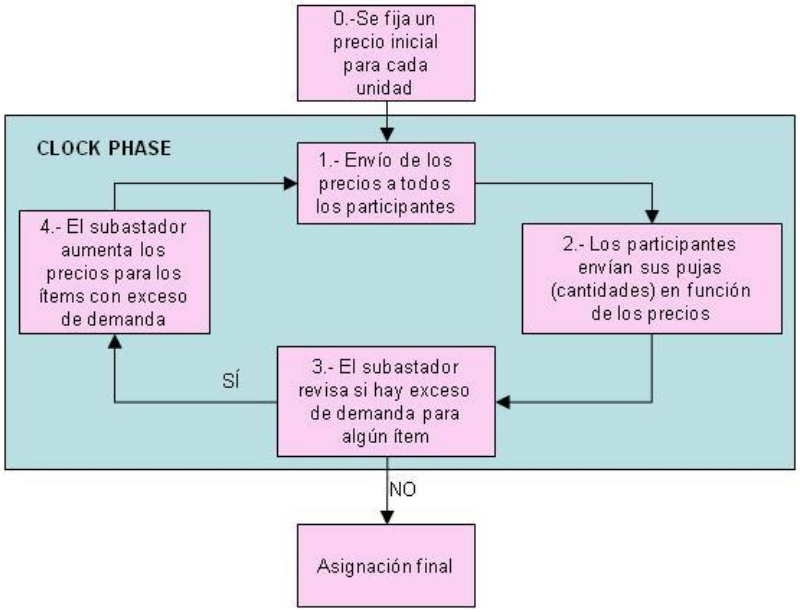

Figura 17. Proceso de la Combinatorial Clock auction (adaptado de Forde y Doyle (2008))

Por último, tenemos la cuestión de si las subastas combinatorias son capaces de llegar al equilibrio en un problema de producción. El mecanismo ideal de subasta combinatoria debería tener las siguientes tres características: (1) un mecanismo eficiente de determinación del ganador de la subasta, (2) un mecanismo de fijación de precios en la puja compatible con los incentivos, y (3) debe permitir imputar precios a los artículos.

Cuando todos los artículos son divisibles, el problema de determinación del ganador en el equilibrio es un problema de programación lineal. Se sabe que el modelo dual del problema lineal es asintóticamente compatible con los incentivos (Fan et al 2003). Los valores de las variables duales de la programación lineal también proporcionan precios individuales, de forma que se satisfacen las tres condiciones (asintóticamente).

Sin embargo, en el caso indivisible esto solamente se cumple en casos especiales, uno de los cuales se produce cuando la solución entera es también una solución al problema de programación conseguido cuando se relaja la condición de variables enteras. Se han propuesto algoritmos basados en precios que pueden ser aplicados a

subastas combinatorias, pero estos pueden encontrar un conjunto de precios con las características deseadas o bien terminar en un ciclo infinito (Xia et al 2004).

En principio, se trataría de buscar mecanismos de mercado que llevaran a precios de equilibrio. Esto es posible en el caso de que se cumplan ciertas condiciones. Una de las condiciones principales es que no existan complementariedades entre los ítems vendidos. Como se ha indicado en el punto anterior en cualquier problema de programación de tareas existen complementariedades, por lo que no se puede esperar que ningún mecanismo de formación de precios conduzca necesariamente al equilibrio (Agnetis et al 2007).

4.3.2. Aplicación de las subastas combinatorias a problemas de programación de la producción

Distintos trabajos han aplicado el concepto de subasta combinatoria a los sistemas programación de operaciones. En la mayor parte de los casos las subastas han sido utilizadas como sistema coordinación dentro de un sistema multiagente. Se busca que puedan ser aplicadas en entornos distribuidos (Kutanoglu y Wu 1999), dinámicos (Dewan y Joshi 2000) y que cumplan las condiciones de robustez y adaptabilidad (Liu et al 2007).

En la Tabla 4.3 se presentan algunos de estos trabajos. En todos los casos su utilización se explica como mecanismo para la resolución distribuida del problema y son implementadas en un sistema multiagente. La mayor parte de estos trabajos siguen el enfoque realizado por Kutanoglu y Wu (1999) y Dewan y Joshi (2002) y utilizan una subasta combinatoria iterativa con fijación de precios. Los trabajos de Wellman et al (2001), Tiwari y Allada (2004) y Tiwari et al (2010) usan la subasta combinatoria iterativa con fijación de cantidad. Los trabajos que utilizan la subasta combinatoria con fijación de precio emplean como base de resolución el algoritmo de Relajación Lagrangiana. En Jeong y Leon (2002) se utiliza una variación del método de Relajación Lagrangiana para el desarrollo de un método distribuido que necesite menos información global, el protocolo CICA. Este método ha sido aplicado para la resolución de programación de una sola máquina (Jeong y Leon 2005) y del problema *job shop* (Jeong y Yim 2009).

Ámbito de aplicación	Subasta combinatoria con fijación de precio	Subasta combinatoria con fijación de cantidad
Problema genérico de programación de la producción	Jeong y Leon (2002),	Wellman et al (2001)
Job shop	Kutanoglu y Wu (1999), Dewan y Joshi (2001, 2002), Liu et al (2007), Jeong y Yim (2009)	
Sistema de fabricación flexible	Araúzo (2007)	Tiwari y Allada (2004)
Programación de proyectos	Lee et al (2003), Araúzo et al (2009)	
Programación de una planta de producción real	Gou et al (1998)	
Programación de una sola máquina	Dewan y Joshi (2000), Jeong y Leon (2005)	
Programación de varias plantas de producción		Tiwari et al (2010)

Tabla 4.3. Aplicaciones de las subastas combinatorias a programación de la producción

4.4. Implementación de las subastas en problemas de programación de operaciones

Para representar un problema de programación de operaciones como una subasta es necesario dar respuesta a las siguientes cuestiones: ¿qué se subasta?, ¿quién puja por lo que se subasta?, ¿qué tipo de subasta?, ¿cómo se expresan las preferencias? y ¿cómo se establecen los precios? Una vez definido el tipo de subasta y sus características, se debe establecer el protocolo de resolución que tenga como resultado la asignación de los bienes subastados entre los participantes en la subasta.

4.4.1. Definición de la subasta

En este punto se da respuesta a estas preguntas teniendo en cuenta las justificaciones que se han ido realizando a lo largo del capítulo.

- *¿Qué se subasta?*

El problema de programación de operaciones puede ser visto como como el problema de asignar intervalos de tiempo de un determinado recurso a una serie de órdenes de trabajo que compiten entre sí por los recursos, que son limitados. El horizonte de programación de cada máquina está dividido en unidades de tiempo o *slots* (Figura 18), que son los bienes por los que se puja. La adquisición de un determinado *slot* de una máquina da derecho a su uso durante ese intervalo de tiempo (Dewan y Joshi 2002).

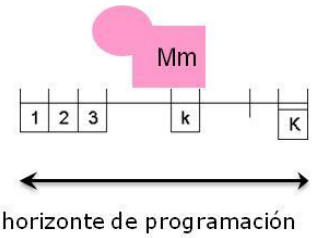

Figura 18. División del horizonte de programación de la máquina *m* en *slots*.

- *¿Quién puja por lo que se subasta?*

Cada una de las órdenes de trabajo que están en el sistema puja por los *slot* de tiempo de los recursos. Cada orden de trabajo tratar de conseguir los *slots* de tiempo de aquellos recursos que le permitan completar sus tareas minimizando su coste, que es la suma de los precios pagados por los *slots* de tiempo seleccionados más la penalización por retraso.

- *¿Qué tipo de subasta?*

Las subastas combinatorias son adecuadas para modelar este problema puesto que permiten considerar las complementariedades debidas a las restricciones de no interrupción de los trabajos y de precedencia de tareas que existen en un problema de producción. Dentro de estas, en las subastas combinatorias de tipo iterativo cada participante conoce a lo largo del proceso las preferencias del resto de participantes de modo indirecto a través de los precios que se establecen, lo que facilita que adapten sus pujas incorporando esta información.

- *¿Cómo se expresan las preferencias?*

Para que la subasta combinatoria sea susceptible de ser implementada en sistemas multiagente es necesario que el procedimiento siga los principios de los sistemas distribuidos, de forma que la información relevante de cada uno de los agentes participantes, como el tiempo de proceso de cada operación, recursos necesarios, fecha de entrega comprometida, penalización por unidad de tiempo de retraso, no sea directamente accesible al resto de agentes. Se busca un sistema distribuido en el que se consiga que no exista un elemento que posea toda la información del sistema, sino que ésta se encontrará distribuida entre los distintos agentes que componen el mismo. Las preferencias de los participantes se conocerán de modo indirecto, expresadas en las pujas que realicen en la subasta. Los participantes sólo tienen acceso a los precios de los ítems de cada iteración, sin saber quiénes ni cuántos participan en la puja, ni cuáles son los ítems por los que cada uno de ellos está pujando. Los precios de cada unidad de tiempo de uso de cada recurso resultantes de la subasta serán los indicadores del impacto que tiene cada *slot* temporal en la consecución de objetivos globales (Duffie 1990).

Cada trabajo encapsula su proceso de producción (rutas y tiempos de operación), la fecha de entrega comprometida, y el momento de llegada de la orden de trabajo. Las máquinas pueden encapsular datos como la disponibilidad (momentos en la que está ocupada por mantenimiento, ...) y número de unidades iguales o capacidad (que puede variar en cada *slot*), pero no disponen de información de forma previa a la subasta respecto a qué trabajos la necesitan, y de ésta sólo mantienen información sobre el último precio de cada unidad de tiempo del horizonte de programación. Esta estructura de datos facilita la adición de nuevas entidades en el sistema, y proporciona robustez ante el posible fallo de alguna de las máquinas (Dewan y Joshi 2001).

Existen dos alternativas genéricas para la expresión de las preferencias de los participantes. En la primera los participantes en la subasta expresan el precio que estarían dispuestos a pagar por cada una de las posibles combinaciones de ítems que se subastan (subasta combinatoria iterativa con fijación de cantidad). En la segunda el subastador fija un precio para cada ítem y los participantes determinan cual es la combinación de ítems que prefieren dados esos precios (subasta combinatoria iterativa con fijación de precios).

En la segunda opción, la información intercambiada entre los participantes y el subastador es mucho menor. En la primera, cada participante tiene que enviar la valoración que realiza de cada posible combinación de ítems (*slots* de tiempo); mientras que en la segunda, sólo envía la combinación de ítems (*slots*) que prefiere. El

número de combinaciones posibles puede aumentar considerablemente cuanto mayor sea el número de ítems (*slots*) subastados.

Dado el conjunto de *slots* seleccionados por el trabajo (B_i), cada trabajo tendrá una función de beneficio o de penalización $F_i(B_i)$ que dependerá de los *slots* que adquiera y tendrá un coste por el uso de los recursos $C_i(B_i)$. Es posible definir una función de utilidad $U_i(B_i)$ de cada trabajo *i* que incluya ambos términos, la función de beneficio y el coste por uso:

$$U_i(B_i) = F_i(B_i) - C_i(B_i) \tag{4.1}$$

Para el conjunto de todas las posibles pujas (β_i) la mejor puja B^*_i para un determinado trabajo es aquella que maximiza su función de utilidad:

$$B_i^* = \underset{B_i \in \beta_i}{\operatorname{argmax}}\, U_i(B_i) \tag{4.2}$$

Durante la subasta cada agente resuelve un problema de maximización de su función de utilidad tratando de encontrar la mejor combinación de unidades de tiempo de cada recurso (B^*_i). Cada agente envía su puja al subastador, quien recoge las pujas de todos los agentes, las evalúa y calcula los nuevos precios de cada unidad de tiempo (*slot*) de cada recurso, pasando a la siguiente iteración.

- *¿Cómo se establecen los precios?*

En una subasta iterativa los precios se van ajustando de forma progresiva. Con la actualización de los precios se trata de eliminar los conflictos que se produzcan. Aparecen conflictos cuando el número de trabajos que solicitan un determinado *slot* de un recurso supera su capacidad. En ese caso, se deberán ajustar los precios de acuerdo al exceso de demanda, mediante un aumento de los mismos.

En cada iteración *r*, cada *slot k* de la máquina *h* tiene asociado un precio λ^r_{hk}, para todo el horizonte de programación y todas las máquinas. Una posible estrategia es variar los precios en cada iteración utilizando una función f(D) que depende exceso de oferta o demanda de cada slot *k* de la máquina *h* (D^r_{hk}) , es decir:

$$\lambda^{r+1}_{hk} = \max\{0, \lambda^r_{hk} + f(D^r_{hk})\} \tag{4.3}$$

La función f(D) determinará los distintos protocolos de subasta. Éstos pueden ser implementados utilizando un método de cálculo de precios walrasiano no adaptativo o mediante un proceso de ajuste adaptativo. Uno de los protocolos no adaptativo es llamado *standard walrasian tâtonnement* (Richter y Wong 1999), en el que la función se define como una constante (*s*) por el valor del exceso de demanda en un determinado *slot k* de la máquina *h*:

$$f(D_{\mathrm{hk}}^{r}) = s \cdot D_{\mathrm{hk}}^{r} \tag{4.4}$$

Existen otras funciones de tipo adaptativo en las que el factor que se aplica al exceso de demanda se va modificando, siendo inicialmente alto y disminuyendo conforme avanzan las iteraciones para ajustar la calidad de la solución. Existen métodos adaptativos derivados de la técnica de Relajación Lagrangiana (Fisher 1981) utilizados para actualizar precios de forma adaptativa en las subastas combinatorias iterativas dada la similitud existente en sus protocolos de funcionamiento. Autores como Kutanoglu y Wu (1999), Dewan y Joshi (2002), Lim y Tang (2005), Attanasio et al (2006), Guo y Lim (2006), Agnetis et al (2007), Hsieh (2007), Liu et al (2007) y Hsieh (2010), han mostrado la similitud existente entre el protocolo de resolución de una subasta iterativa combinatoria con fijación de precios y el algoritmo de Relajación Lagrangiana para la resolución de problemas complejos. En el capítulo 5 se muestra esta relación aplicada al problema de programación de talleres.

4.4.2. Protocolo de resolución de la subasta combinatoria

Una vez definida la subasta, el siguiente paso es describir su funcionamiento. Se trata de una subasta de tipo iterativo, por lo que existirá un ciclo que se repetirá en cada iteración hasta que se cumpla alguna condición de parada. Cada ciclo estará compuesto por una serie de pasos en los que los precios se irán ajustando y actuarán como señales para que los compradores modifiquen sus demandas. La subasta se realizará siguiendo los pasos que muestra el siguiente protocolo (ver Figura 19):

- *Paso 0. Inicialización de variables*: el subastador comienza fijando unos precios iniciales para los slot de tiempo de cada uno de los recursos $\lambda_{\mathrm{hk}}^{0}$. Lo habitual es comenzar con un valor de cero para el conjunto de *slots* de tiempo.
- *Paso 1. Cálculo de la puja a enviar por parte de cada trabajo*: cada uno de los trabajos maximiza su utilidad dados los precios (λ^{r}). Su objetivo es que la suma

de la función de penalización por el retraso más el valor del coste de uso de los recursos, dado por la suma de los precios de los *slots* de los recursos seleccionados, sea mínimo. Para ello cada trabajo evaluará todas las alternativas escogiendo la de menor valor. Por último enviará al subastador su puja con el conjunto de *slots* seleccionados.

- *Paso 2. Evaluación de las pujas enviadas:* el subastador combina los programas propuestos por los trabajos y evalúa las necesidades de capacidad de cada recurso que existen en cada instante de tiempo.
- *Paso 3. Construcción del programa factible*: en general la solución propuesta a partir de las pujas enviadas por las tareas va a resultar no factible porque no se cumplirán las restricciones de capacidad de los recursos. Es improbable que las preferencias entre los participantes en la subasta sean complementarias y no se generen conflictos en los recursos limitados Se construye una solución factible a partir de las propuestas enviadas por los trabajos, de forma que se eliminen los conflictos.
- *Paso 4. Criterio de parada:* en el caso de que el sistema llegue al equilibrio o bien los precios se estabilicen, el proceso se parará y la solución será la asignación realizada en el paso 3. En caso contrario, se continuará con el paso 5.

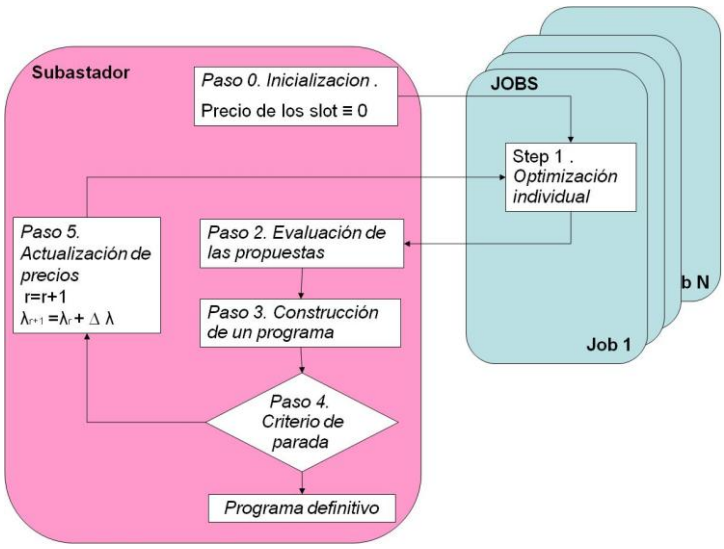

Figura 19. Subasta combinatoria iterativa con fijación de precios

- *Paso 5. Actualización de los precios:* el objetivo de actualizar los precios es reducir los conflictos entre los agentes. Así, en los periodos de tiempo en los que exista una demanda superior a la capacidad del recurso el precio aumentará. Si coinciden demanda y capacidad del recurso, el precio no variará. Por último, si la demanda de un recurso en un determinado *slot* es inferior a su capacidad, el precio disminuirá. El precio no puede ser negativo.

4.5. Conclusiones del capítulo

En este capítulo se han estudiado las subastas como medio de coordinación. Se han descrito los principales tipos de subasta especialmente las subastas combinatorias. El uso de este tipo de subastas es adecuado cuando se subastan varios ítems y la valoración de cada uno depende del resto de ítems adquiridos, es decir, existen complementariedades entre los ítems subastados.

Si se modela el problema de programación de la producción como una subasta donde las órdenes de trabajo pujan por conseguir los *slots* de tiempo en que se divide el horizonte de programación de cada recurso (ítems subastados), estos ítems presentaran complementariedades. Éstas son debidas a algunas de las restricciones que definen el problema, como son las restricciones de no interrupción de las tareas (non-preemption) y las restricciones de precedencia entre ellas. Dentro de las distintas opciones para modelar el problema se ha seleccionado el modelo de subasta iterativa con fijación de precios. La elección está justificada porque de este modo se permite que cada agente conozca las preferencias de los demás, expresadas a través de los precios que se van generando y porque el modo de fijación de precios presenta una menor carga de información.

Para utilizar estos procedimientos es preciso seleccionar algún método de actualización de precios. Lo deseable es que conduzca hacia el equilibrio si existen complementariedades entre los ítems intercambiados, éste no tiene por qué existir. Por tanto, bastará con que el método de actualización de precios tenga un comportamiento predecible. Una opción es utilizar la semejanza entre las subastas combinatorias y el método de Relajación Lagrangiana que, dado que parte de una formulación matemática rigurosa como procedimiento de optimización, permitirán una mejor predicción de su comportamiento. En los siguientes capítulos se estudia el método de Relajación Lagrangiana y su aplicación al problema *job shop*.

Capítulo 5. Método de Relajación Lagrangiana: principales propiedades

En el capítulo anterior se justificaba la necesidad de encontrar métodos de actualización de precios para las subastas combinatorias que permitan una mejor predictibilidad de su comportamiento así como la posibilidad de mejorar el procedimiento. Como ya se comentó, distintos autores han mostrado la similitud existente entre el protocolo de resolución de una subasta iterativa combinatoria con fijación de precios y el algoritmo de Relajación Lagrangiana para la resolución de problemas complejos (apartado 4.4).

En este capítulo se estudian los fundamentos y propiedades de la técnica de Relajación Lagrangiana, lo que nos proporcionará una mejor comprensión de sus posibilidades a la hora de ser implementada en un entorno distribuido. En primer lugar se ofrece un conjunto de definiciones y se estudian las propiedades de la Relajación Lagrangiana. Seguidamente se presentan los métodos del subgradiente, del subgradiente conjugado y del gradiente surrogado como parte de la resolución del método. A continuación, el método de Relajación Lagrangiana se aplica al caso particular de resolución de un problema separable de programación entera. Por último, se revisan los trabajos previos en los que se aplica la Relajación Lagrangiana como método de resolución de problemas de programación de operaciones.

5.1. Relajación Lagrangiana

En los métodos de resolución mediante relajación, la idea básica es simplificar el problema, eliminando, normalmente, restricciones que hacen que el problema sea complejo, de modo que el problema resultante (problema relajado) sea mucho más sencillo de resolver. Posteriormente, se transforma la solución obtenida en el problema relajado en otra que cumpla con las restricciones del problema inicial.

La Relajación Lagrangiana pretende transformar un problema de programación entera, cuya complejidad proviene de un determinado conjunto de restricciones, en otro problema relativamente más fácil. El problema transformado se denomina Problema Relajado. En él, las restricciones complicadas son eliminadas e incorporadas a la función objetivo afectadas por un determinado peso denominado multiplicador de Lagrange. Cada peso representa una penalización que se añade a una solución que no satisface una determinada restricción (Fisher 1985).

Aunque la idea de eliminar las restricciones para resolver el nuevo problema y deducir la solución del problema original a partir de la solución del problema relajado puede parecer muy atractiva, lo cierto es que la solución del problema relajado no suele ser una solución factible del problema original, lo que hace necesario transformarla en una solución que cumpla con todas las restricciones. Esto no garantiza que sea la solución óptima del problema original.

A pesar de ello, la utilización de los métodos lagrangianos tiene ventajas tales como: (1) la amplia gama de problemas a los que pueden ser aplicados, y (2) dado que las técnicas de relajación proporcionan una cota inferior del problema original, con la Relajación Lagrangiana es posible obtener un indicador de la calidad de la solución calculada mediante la medición de la diferencia (gap) entre una cota superior y la cota inferior que proporciona el método (Beasley 1993).

5.2. Definiciones y propiedades en la Relajación Lagrangiana

En este apartado se presentan las definiciones fundamentales y las propiedades básicas relacionadas con el método de Relajación Lagrangiana. El fundamento teórico lo podemos encontrar en Geoffrion (1974), Fisher (1981, 1985), Wolsey y Nemhauser (1999), Guignard (2003) y Lemaréchal (2007), entre otros.

- Relajación de un problema P

Para definir el concepto de relajación, utilizaremos dos funciones genéricas f(x) y g(x) a minimizar sometidas, respectivamente, a las restricciones determinadas por V y W, conjuntos de restricciones de cada uno de los problemas. Sean los problemas $(P_{\min})$ (5.1) y $(RP_{\min})$ (5.2):

$$(P_{min}) \quad \min_x f(x) \quad sujeto\ a \quad x \in V \tag{5.1}$$

$$(RP_{min}) \quad \min_x g(x) \quad sujeto\ a \quad x \in W \tag{5.2}$$

El problema (RP_{min}) es una relajación de (P_{min}) si se cumplen las siguientes condiciones:

- El conjunto factible de (RP_{min}) contiene al de (P_{min}):

$$W \supseteq V \tag{5.3}$$

- Sobre el conjunto factible de P_{min} la función objetivo de RP_{min} domina (es mejor que) la de P_{min}:

$$\forall x \in V, g(x) \leq f(x) \tag{5.4}$$

- Función Lagrangiana

Consideramos ahora un problema general de optimización (P) definido como:

$$(P) \quad \min_x f(x) \tag{5.5}$$

$$sujeto\ a \quad Ax \leq b \tag{5.6}$$

$$Cx \leq d \tag{5.7}$$

$$x \in X \tag{5.8}$$

donde (5.6) contiene las restricciones que se consideran complicadas, de modo que sin ellas el problema sería mucho más fácil de resolver. Las restricciones (5.7) y (5.8) se mantendrán en el problema. La expresión (5.8) contiene las restricciones de signo y de integridad (solución entera o binaria).

Sea λ el vector no negativo de pesos ($\lambda \geq 0$), denominado vector de multiplicadores de Lagrange. La Función Lagrangiana es función de la variable $x \in X$ pero también de

la variable $\lambda \in \mathbb{R}^n$, y se define según (5.9) de modo que las restricciones complicadas (5.6) son relajadas, o dualizadas, siendo los multiplicadores de Lagrange (λ) las variables duales.

$$L(x, \lambda) = f(x) + \lambda(Ax - b) \tag{5.9}$$

- Relajación Lagrangiana

Se denomina *Relajación Lagrangiana de (P)*, al problema de minimizar la Función Lagrangiana $L(x, \lambda)$ sobre la variable $x \in X$ sujeto a las restricciones no relajadas del problema *(P)*:

$$\begin{aligned} (LR_\lambda) \quad & \min_x \{f(x) + \lambda(Ax - b)\} = \min_x L(x, \lambda) \\ & sujeto\ a \quad Cx \leq d \\ & \qquad\quad x \in X \end{aligned} \tag{5.10}$$

La denominación del problema (LR_λ) de Relajación Lagrangiana de (P) con respecto a las restricciones complicadas (5.6) proviene de que el vector λ juega el mismo rol que juegan los multiplicadores de Lagrange en los problemas de optimización continuos. Se puede comprobar que el problema (LR_λ) es una relajación de (P), es decir, cumple las condiciones (5.3) y (5.4).

- Función dual asociada

Al resolver el problema (LR_λ) (5.10), se obtiene un valor dependiente del vector *λ*. A este valor se denomina función dual asociada a las restricciones complejas, *θ(λ)*:

$$\begin{aligned} & \theta(\lambda) = \min_x L(x.\lambda) = \min_x (f(x) + \lambda(Ax - b)) \\ & sujeto\ a \quad Cx \leq d \\ & \qquad\quad x \in X \end{aligned} \tag{5.11}$$

O también:

$$\theta(\lambda) = v\,(LR_\lambda) \tag{5.12}$$

- Dualidad débil

El teorema de la dualidad débil (Guignard 2003) demuestra que para cualquier valor de los multiplicadores de Lagrange ($\lambda \geq 0$), el valor de la función dual *θ(λ)* (5.11)

proporciona una cota inferior al valor mínimo de la función objetivo del problema original (P) (5.5).

Sea x(λ) una solución de (LR_λ), para algún $\lambda \geq 0$, conocida como solución Lagrangiana. Aunque esta solución sea factible en el problema original, en la mayoría de los casos no es la óptima. Sin embargo, se puede afirmar que:

1.- Si x(λ) es una solución de (LR_λ) para algún ($\lambda \geq 0$) para algún $\lambda \geq 0$, entonces se cumple que el valor de la función dual $\theta(\lambda)$ es una cota inferior del valor del óptimo del problema:

$$\theta(\lambda) = f\big(x(\lambda)\big) + \lambda(Ax(\lambda) - b) \leq v(P) \tag{5.13}$$

Para demostrar (5.13) se define el problema (P') como:

$$(P') \quad \min_x \{f(x) + \lambda(Ax - b)\} \tag{5.14}$$
$$sujeto\ a \quad Ax \leq b$$
$$Cx \leq d$$
$$x \in X$$

Teniendo en cuenta que el valor óptimo del problema (P) (5.5), *v*(P), no es menor que el valor óptimo del problema (P') (5.14), *v*(P'), y el valor óptimo del problema (P') no es menor que el valor óptimo de la Relajación Lagrangiana de (P), ya que al eliminar una de las restricciones este valor óptimo sólo puede llegar a ser menor:

$$v(LR_\lambda) \leq v(P') \leq v(P) \tag{5.15}$$

2.- Si además x(λ) es factible para (P), el valor óptimo de (P), *v*(P), se encuentra acotada inferiormente por el valor de la función dual asociada en λ y superiormente por el valor de la función objetivo del problema original (P) en x(λ):

$$\theta(\lambda) = f\big(x(\lambda)\big) + \lambda(Ax(\lambda) - b) \leq v(P) \leq f\big(x(\lambda)\big) \tag{5.16}$$

Se comprueba que el hecho de encontrar una solución óptima del problema relajado (LR_λ) que sea factible, no implica que esa solución sea óptima para (P). Para que esto ocurra, además se ha de cumplir la siguiente condición.

3.- Si además se cumple:

$$\lambda(\mathrm{Ax}(\lambda) - b) = 0 \tag{5.17}$$

entonces x(λ) es una solución óptima de (P), y valor del mínimo del problema (P) se corresponde con el valor de la función objetivo para x(λ):

$$v(P) = f\big(\mathrm{x}(\lambda)\big) \tag{5.18}$$

El punto 3 aporta una condición suficiente de optimalidad, pero no es una condición necesaria. Es posible encontrar una solución óptima de (LR_λ) que sea factible en (P) que no cumpla la expresión (5.17), y que sea una solución óptima de (P).

- Problema dual

Con objeto de encontrar la mayor cota inferior de la solución del problema (P), el método de la Relajación Lagrangiana trata de resolver el llamado Problema Dual Lagrangiano (LD) que busca el máximo de la función dual asociada a las restricciones complicadas:

$$\text{(LD)} \qquad \max_{\lambda \geq 0} v\,(LR_\lambda) = \max_{\lambda \geq 0} \theta(\lambda) \tag{5.19}$$

LD es un problema en el espacio de los multiplicadores lagrangianos, mientras que LR_λ es un problema en *x*.

- Cota lagrangiana

Sea λ* la solución del problema dual (LD). Llamamos cota Lagrangiana al valor del óptimo del problema dual Lagrangiano *v*(LD):

$$v(\mathrm{LD}) = \theta(\lambda^*) \tag{5.20}$$

Una consideración importante es que *v*(LD) va a proporcionar una cota inferior de la solución del problema P, pero si llamamos x_p^* a la solución de (P) y x_{LD}^* a la solución de (LD), no se puede asegurar que $x_p^* = x_{LD}^*$. Generalmente la solución obtenida de la Relajación Lagrangiana es no factible. Por eso, es necesario transformar la solución obtenida para obtener una cota superior para el problema original.

Sea F una función que transforma una solución de la Relajación Lagrangiana del problema (P), (LR_λ) , en una solución de (P):

$$\begin{aligned} F\colon \Delta &\longrightarrow \Delta_{\mathrm{p}} \\ \mathrm{x} &\longrightarrow \Delta_{\mathrm{p}} \end{aligned} \tag{5.21}$$

En general la solución transformada de la solución del problema dual (LD) no será el óptimo de (P):

$$F(x_{LD}^*) \neq x_{LD}^* \tag{5.22}$$

- Gap de dualidad (duality gap)

Se define el gap de dualidad como la diferencia (no negativa) entre los valores óptimos de (P) y de (LD), como muestra el ejemplo de la Figura 20.

$$GAP = f(x_p^*) - v(LD) \tag{5.23}$$

Cuanto más pequeño sea el gap de dualidad, mejor será el resultado obtenido mediante el proceso de Relajación Lagrangiana.

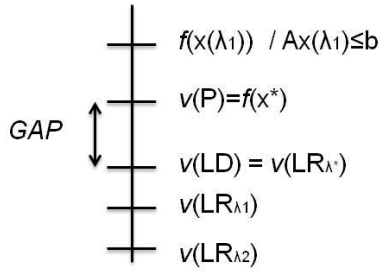

Figura 20. Ejemplo ilustrativo de los principales puntos en la relajación lagrangiana.

5.3. Resolución del problema dual

Existen distintos métodos para la resolución del problema dual de la expresión (5.19). Los principales son el método del subgradiente (Held et al 1974), el método de los planos cortantes (Tomastik y Luh 1993) y el método de empaquetamiento (*bundle*) (Hiriart-Urruty y Lemarechal 1996). El método del subgradiente se caracteriza por su sencillez de implementación y su buena convergencia, por lo que es el que se aplica en este trabajo. En este apartado se describe el método del subgradiente y dos métodos derivados, el método del subgradiente conjugado y el método del gradiente surrogado.

5.3.1. Método del subgradiente

El método del subgradiente es un método iterativo que se estructura en dos niveles (Figura 21): el nivel de resolución del problema de Relajación Lagrangiana (LR_λ) (5.24) y el de resolución del problema dual (LD) (5.25):

$$\begin{aligned} &(LR_\lambda) \quad L(\lambda) = \min_x \{f(x) + \lambda(Ax - b)\} \\ &sujeto\ a \quad Cx \leq d \\ &\qquad\quad x \in X \end{aligned} \tag{5.24}$$

$$(\text{LD}) \quad \max_{\lambda \geq 0} v\,(LR_\lambda) = \max_{\lambda \geq 0} \theta(\lambda) \tag{5.25}$$

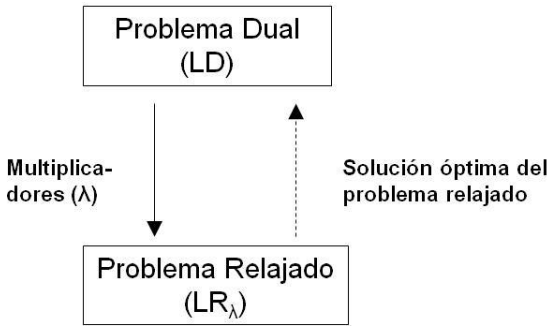

Figura 21. Resolución del problema dual por el método del subgradiente

En cada iteración se va avanzando paso a paso a lo largo de la dirección negativa del subgradiente de la función dual asociada a las restricciones complicadas $\theta(\lambda)$. Esta función es cóncava pero no diferenciable. El subgradiente de una función es el equivalente para las funciones no diferenciables del gradiente de las funciones diferenciables. Para una iteración *r*, el subgradiente de la función dual asociada a las restricciones complicadas se calcula mediante (5.26), que se puede interpretar como el vector de incumplimiento de las restricciones relajadas.

$$g^r = Ax(\lambda^r) - b \tag{5.26}$$

Los multiplicadores de Lagrange se actualizan en cada iteración *r* siguiendo la dirección negativa del subgradiente, según (5.27). Se utiliza la proyección positiva del vector dado que los multiplicadores de Lagrange no son negativos.

$$\lambda^{r+1} = \max(0, \lambda^r + \alpha^r g^r) \tag{5.27}$$

El valor α^r es el tamaño del paso en cada iteración. Para que el método converja α^r debe ir disminuyendo. Si α^r se calcula como:

$$\alpha^r = \gamma_r \frac{\theta(\lambda^*) - \theta(\lambda^r)}{\|g^r\|^2} \qquad \text{con} \quad 0 < \gamma_r \leq 2 \tag{5.28}$$

los multiplicadores convergen a la solución óptima. En esta expresión se calcula la diferencia entre la cota Lagrangiana y el valor de la función dual asociada en esa iteración, dividido por la norma del subgradiente en la iteración. En esas condiciones la dirección dada por el subgradiente está en ángulo agudo con la dirección hacia el óptimo λ^* :

$$0 \leq \theta(\lambda^*) - \theta(\lambda^r) \leq (\lambda^* - \lambda^r) g(\lambda^r) \tag{5.29}$$

por lo que la distancia entre los multiplicadores de Lagrange en la iteración actual (λ^n) y en su valor óptimo (λ^*) puede reducirse paso a paso (Held et al 1974).

Dado que no se conoce a priori el valor de la cota Lagrangiana $\theta(\lambda^*)$, ha de ser estimado en el procedimiento de cálculo del tamaño del paso con algún valor que se aproxime. El método de estimación más utilizado es sustituir la cota Lagrangiana por una cota superior. Ésta puede calcularse como el mejor valor obtenido hasta el momento en la función original $V[f(x_p)]$, donde x_p es una solución factible del problema original (P). Esta cota se acercará cada vez más al valor de $\theta(\lambda^*)$.

$$\alpha^r = \gamma^r \frac{(V[f(x_p)] - \theta(\lambda^r))}{\|g^r\|^2} \qquad con\ 0 < \gamma < 2 \tag{5.30}$$

Es habitual que la secuencia de valores de γ_r se obtenga comenzando por $\gamma_0 = 2$ y se vaya reduciendo a la mitad si el valor de la función dual asociada $\theta(\lambda^r)$ no ha mejorado su valor en un determinado número de iteraciones (Fisher 1981). En todos los trabajos este factor toma valores entre $0 < \gamma < 2$. No existe un acuerdo en la frecuencia con la que se ha de actualizar este factor, siendo utilizados valores que van desde 5, 10, 20 o más iteraciones.

- Método de resolución

El método del subgradiente puede ser descrito mediante el siguiente algoritmo:

- *Inicialización de valores:* se asignan unos valores iniciales a los multiplicadores de Lagrange (λ^0). Habitualmente se les da el valor cero, pero en ocasiones puede ser más conveniente asignar un valor relacionado con el problema a resolver.

- *Paso 1. Resolución del problema relajado:* dados los multiplicadores de Lagrange ($\lambda^r \geq 0$) se resuelve el problema relajado (LR_λ) (5.10), obteniendo así la solución óptima $x(\lambda^r)$.
- *Paso 2. Cálculo del subgradiente:* se calcula el subgradiente, que representa el incumplimiento de las restricciones, mediante (5.26).
- *Paso 3. Generación de una solución factible:* en general la solución obtenida en el paso 2 no será compatible con las restricciones del problema original (P). Se busca una solución cumpla con las restricciones del problema mediante algún método adecuado.
- *Paso 4. Criterio de parada:* se establece un criterio de parada para el algoritmo, de modo que si el paso calculado en la iteración es muy pequeño o si coinciden la solución de los problemas relajado y original el algoritmo se para. En caso de contrario, continúa con el paso 5.
- *Paso 5. Mejora del valor de la función dual:* se busca mejorar el valor de la función dual. Para ello se actualizan los multiplicadores de Lagrange mediante (5.27). Se actualiza el contador *r* y continúa con el paso 1.

5.3.2. Método del subgradiente conjugado

Este método está basado en el método del subgradiente, pero en lugar de actualizar los multiplicadores de Lagrange en la dirección del subgradiente, se busca una dirección que es combinación lineal del subgradiente con los subgradientes utilizados en iteraciones anteriores.

El método del subgradiente conjugado busca tener una mejor convergencia que el método del subgradiente porque reduce las oscilaciones que caracterizan a este método (Brännlund 1995, Camerini et al 1975). La única diferencia con el método del subgradiente es la forma de calcular la dirección de actualización de los multiplicadores de Lagrange en el paso 5. En lugar de seguir la dirección del subgradiente, los multiplicadores son actualizados siguiendo una dirección que se obtiene como combinación lineal entre la dirección del subgradiente de la iteración actual y la dirección de actualización de la iteración anterior:

$$d^r = \begin{cases} g^0 & si\ r = 0 \\ \beta g^r + (1-\beta) d^r\ con\ 0 < \beta < 1 & en\ \ otro\ caso \end{cases} \tag{5.31}$$

Los multiplicadores de Lagrange se actualizan mediante:

$$\lambda^{r+1} = \max(0, \lambda^r + \alpha^r d^r) \tag{5.32}$$

El tamaño del paso α^r también se modifica respecto al método del subgradiente puesto que el divisor es la norma de la dirección de actualización de la iteración y se calcula mediante:

$$\alpha^r = \gamma^r \frac{(V[f(x_p)] - \theta(\lambda^r))}{\|d^r\|^2} \qquad con\ 0 < \gamma < 2 \tag{5.31}$$

Otra variante es la de Brännlund (1995), que propone calcular la dirección en cada iteración según:

$$d^r = \begin{cases} g^0 & si\ r = 0 \\ \beta^r g^r + (1 - \beta^r) d^r & en\ \ otro\ caso \end{cases} \tag{5.32}$$

$$\text{con } \gamma^r = \beta^r = \begin{cases} \frac{(d^{r-1})^T d^{r-1}}{(d^{r-1})^T d^{r-1} - (g^r)^T d^{r-1}} & si\ (g^r)^T d^{r-1} < 0 \\ 1 & \text{en caso contrario} \end{cases} \tag{5.33}$$

Los precios también se calculan según (5.32) y el valor del tamaño del paso con (5.33).

- Método de resolución

El algoritmo de resolución del método del subgradiente conjugado es igual al del método del subgradiente del apartado 5.3.2, excepto en el paso 5, en el que la dirección de actualización se calcula mediante la expresión (5.31), los multiplicadores de Lagrange se actualizan mediante la expresión (5.32), y el tamaño del paso mediante la expresión (5.31).

5.3.3. Método del gradiente surrogado

El método del gradiente surrogado también se basa en el método del subgradiente. A diferencia de éste, no busca la solución del problema relajado en cada iteración, sino que utiliza una aproximación que mejore el valor de la función relajada para los multiplicadores de Lagrange respecto a la solución de la iteración anterior. Se ha comprobado que este método tiene una mejor estabilidad que el método del subgradiente y reduce la oscilación provocada por problemas homogéneos (Lorena y

Narciso 1996), y su principal ventaja es que permite actualizar los multiplicadores sin tener que calcular el óptimo del problema relajado (Zhao et al 1999).

Se define la función dual surrogada a partir de la función dual $\theta(\lambda)$ como:

$$\begin{aligned} &\tilde{L}(\lambda, x) = f(x) + \lambda^r (Ax - b) \\ &con \qquad Cx \leq d \\ &\qquad\quad x \in X \end{aligned} \tag{5.34}$$

Comparándola con la función dual se puede comprobar que, al contrario que ésta, la función surrogada no exige la optimización para un determinado valor de los multiplicadores de Lagrange λ (por eso es una función en λ y en las variables del problema original *x*, $\tilde{L}(\lambda, x)$), mientras que la función dual es una función en λ, es decir, $\theta(\lambda)$.

En este método los multiplicadores de Lagrange se actualizan mediante (5.35), donde $\tilde{g}$ es el gradiente surrogado que se calcula con la expresión (5.36).

$$\lambda^{r+1} = \max(0, \lambda^r + \alpha^r \tilde{g}^r) \tag{5.35}$$

$$\tilde{g}^r = \tilde{g}(x^r) = Ax^r - b \tag{5.36}$$

Al utilizar el método del subgradiente surrogado ya no es posible asegurar que el coste dual surrogado sea siempre una cota inferior del coste óptimo. Sin embargo, el hecho de no tener que calcular el valor óptimo del problema relajado puede compensar este inconveniente.

Para que el algoritmo converja al óptimo, en cada iteración se tiene que cumplir que, dados los multiplicadores de Lagrange (λ^r) de la nueva iteración, el valor de la función lagrangiana surrogada, para el valor de la variable de la nueva iteración (x^r), mejore el valor de la función lagrangiana surrogada del valor de la variable de la iteración anterior (x^{r-1}) y los multiplicadores de Lagrange de la nueva iteración (λ^r), es decir:

$$\tilde{L}(\lambda^r, x^r) < \tilde{L}(\lambda^r, x^{r-1}) \tag{5.37}$$

En Zhao et al (1999) se demuestra que, en estas condiciones, el subgradiente surrogado es una dirección correcta (se encuentra en un ángulo agudo respecto la dirección correcta hacia λ^*). Para el cálculo del paso se utiliza una expresión similar a la utilizada en el método del subgradiente, en la que se sustituye el subgradiente por el gradiente surrogado y la función dual asociada por la función dual asociada en el punto x^r:

$$\alpha^r = \gamma^r \cdot \frac{\theta(\lambda^*) - \tilde{\mathrm{L}}(x^r, \lambda^r)}{\|\tilde{g}(x^r)\|^2} \qquad \text{con } 0 < \gamma^r < 1 \tag{5.38}$$

Al igual que en el método del subgradiente, no se conoce la solución del problema dual $\theta(\lambda^*)$, por lo que se aproxima su valor al de la cota superior calculada como el mejor valor obtenido hasta el momento en la función original $V[f(x_p)]$, donde x_p es una solución factible del problema original (P). La expresión se transforma en:

$$\alpha^r = \gamma^r \frac{(V[f(x_p)] - \tilde{\mathrm{L}}(x^r, \lambda^r)}{\|\tilde{g}(x^r)\|^2} \qquad con\ 0 < \gamma^r < 1 \tag{5.39}$$

- Método de resolución

El método de resolución sigue una estructura similar al del método del subgradiente, pero se modifican algunos de los pasos. A continuación se describe el algoritmo, indicando las diferencias respecto al método del subgradiente.

- *Inicialización de valores:* se asignan unos valores iniciales a los multiplicadores de Lagrange (λ^0) y se realiza una primera iteración siguiendo el método del subgradiente.
- *Paso 1. Mejora de la función Lagrangiana surrogada:* dados los multiplicadores de Lagrange actualizados (λ^r), se trata de buscar un valor de las variables del problema (x^r) que cumpla que, con los multiplicadores actualizados (λ^r), se mejore el valor de la función lagrangiana surrogada (5.34) respecto al valor de esta función con los valores de las variables del problema de la iteración anterior (x^{r-1}) y los nuevos multiplicadores de Lagrange (λ^r), condición (5.37).
- *Paso 2. Cálculo del subgradiente surrogado:* una vez calculado el valor de las variables del problema (x^r) que cumplen (5.37), se calcula el subgradiente surrogado que representa el incumplimiento de las restricciones relajadas mediante la expresión (5.36).
- *Paso 3. Generación de una solución factible:* este paso es igual al del método del subgradiente.
- *Paso 4. Criterio de parada:* el criterio de parada en este método es que si el vector de multiplicadores de Lagrange ha variado muy poco de una iteración a la siguiente el algoritmo se para (5.40), en caso contrario continúa con el paso 5.

$$\|\lambda^{r+1} - \lambda^r\| < \varepsilon_2\ con \quad \varepsilon_2 \to 0 \tag{5.40}$$

- *Paso 5. Mejora del valor de la función dual:* se busca mejorar el valor de la función dual. Para ello se actualizan los multiplicadores de Lagrange mediante la expresión (5.35), teniendo en cuenta que la dirección de actualización será la del gradiente surrogado (5.36) y el tamaño del paso se calcula con (5.39). Se actualiza el contador *r* y continúa con el paso 1.

- Variantes

Chen y Chu (2003) proponen una alternativa al cálculo del paso. La razón es que el dual surrogado puede llegar a tener un valor superior al mínimo del problema original. Por esta razón se utiliza un método de cálculo del paso adaptativo que estima el valor de la cota Lagrangiana $[\theta(\lambda^*)]$ basándose en el valor dual surrogado (5.42), en lugar de utilizar el mejor valor de la función objetivo del problema original obtenido hasta ese momento. De esta forma, el paso en cada iteración se calcula con (5.41).

$$\alpha^r = \gamma^r \cdot \frac{\widehat{\mathrm{D}}^* - \tilde{\mathrm{L}}(x^r, \lambda^r)}{\|\tilde{g}(x^r)\|^2} \quad \text{con} \quad 0 < \gamma^r < 1 \tag{5.41}$$

$$\widehat{\mathrm{D}}^* = \left(1 + \frac{\varpi}{\theta^p}\right) \cdot V[\tilde{\mathrm{L}}^{[r]}] \tag{5.42}$$

donde:

$\tilde{\mathrm{L}}(x^r, \lambda^r)$ es el valor de la función lagrangiana surrogada en la iteración *r*.

$V[\tilde{\mathrm{L}}^{[r]}]$ es el mejor valor de la función lagrangiana surrogada hasta la iteración *r*

$\varpi \in [0.1 ; 1.0]$ y $\rho \in [1.1 ; 1.5]$

$$\theta^r = \begin{cases} \max(1;\ \theta^{r-1} - 1) & \tilde{\mathrm{L}}(x^r, \lambda^r) > V[\tilde{\mathrm{L}}^{[r]}] \\ \theta^r = \theta^{r-1} + 1 & en\ otro\ caso \end{cases}$$

5.4. Aplicación del método de Relajación Lagrangiana a un problema separable de programación entera

Un caso particular de aplicación de la Relajación Lagrangiana se da cuando en el problema (P) de (5.5), la función objetivo se puede descomponer en varias funciones relacionadas con las variables de decisión (Zhao et al 1997). Supongamos que la

función objetivo f(x) se puede definir como suma de I funciones $f_i\ (x_i)$, que pueden ser no lineales:

$$f(x) = \sum_{i=1}^{I} f_i\ (x_i) \tag{5.43}$$

donde:

- $x = [x_1, x_2, \dots, x_n]^T$ es el vector de variables de decisión de dimensión nx1 con $n = \sum_{i=1}^{I} n_i$, compuesto por I vectores x_i de dimensión $n_i \times 1$.
- $A = [A_1, A_2, \dots, A_I]^T$ es una matriz mxn, donde A_i es una matriz $m \times n_i$
- b es un vector $m \times 1$
- C_i es una matriz $s \times n_i$
- d_i es un vector $s \times 1$

El problema (P) queda definido como:

$$(P) \qquad \min_x \sum_{i=1}^{I} f_i\ (x_i) \tag{5.44}$$

$$sujeto\ a \quad Ax \leq b \tag{5.45}$$

$$C_i x_i \leq d_i\ , \quad i = 1, \dots, I \tag{5.46}$$

$$x_i \in Z^{n_i}\ , \quad i = 1, \dots, I \tag{5.47}$$

Las m restricciones (5.45) son las que complican el problema (restricciones complicadas), puesto que interrelacionan las variables de los diferentes grupos x_i. El resto de restricciones relacionan solo variables de uno de los grupos x_i. Al aplicar la Relajación Lagrangiana a las restricciones complicadas (5.45) el problema (P) se transforma en:

$$(LR_\lambda) \qquad \min_x \left\{ \sum_{i=1}^{I} f_i\ (x_i) + \lambda(Ax - b) \right\} \tag{5.48}$$

$$sujeto\ a \quad C_i x_i \leq d_i\ , \quad i = 1, \dots, I$$

$$x_i \in Z^{n_i}\ , \quad i = 1, \dots, I$$

El segundo término de la función objetivo de la Relajación Lagrangiana (LR_λ) de (5.48) puede ser expresado como sumatorio de los términos en x_i según:

$$\lambda(\mathrm{Ax}-\mathrm{b}) = \sum_{i=1}^{I} \lambda(\mathrm{a_i x_i}) - \lambda \mathrm{b} \tag{5.51}$$

Si se sustituye en (5.48), quedaría como:

$$(LR_\lambda) \qquad \min_x \left\{ \sum_{i=1}^{I} \mathrm{f_i}\,(x_i) + \sum_{i=1}^{I} \lambda(\mathrm{a_i x_i}) - \lambda \mathrm{b} \right\} \tag{5.49}$$
$$sujeto\ a \quad C_i x_i \le d_i\ , \quad i = 1, \dots, I$$
$$x_i \in Z^{n_i}\ , \quad i = 1, \dots, I$$

Con ello se consigue agrupar en los términos que dependen de *i*.

En (5.49) el término λb es independiente de la variable a minimizar, por lo que se puede sacar de la función de minimización. Es decir:

$$(LR_\lambda) \qquad \min_x \left\{ \sum_{i=1}^{I} \mathrm{f_i}\,(x_i) + \sum_{i=1}^{I} \lambda(\mathrm{a_i x_i}) \right\} - \lambda \mathrm{b} \tag{5.50}$$
$$sujeto\ a \quad C_i x_i \le d_i\ , \quad i = 1, \dots, I$$
$$x_i \in Z^{n_i}\ , \quad i = 1, \dots, I$$

Ahora, al eliminar las restricciones que ligan los I grupos de variables x_i, la expresión (5.50) se puede poner como:

$$(LR_\lambda) \qquad \sum_{i=1}^{I} \min_{x_i} \{\mathrm{f_i}\,(x_i) + \lambda(\mathrm{a_i x_i})\} - \lambda \mathrm{b} \tag{5.51}$$
$$sujeto\ a \quad C_i x_i \le d_i\ , \quad i = 1, \dots, I$$
$$x_i \in Z^{n_i}\ , \quad i = 1, \dots, I$$

Por lo tanto, la Relajación Lagrangiana puede ser reescrita en términos de I subproblemas individuales cuya función dual asociada es $\theta_i(\lambda)$:

$$\theta_i(\lambda) = \min_{x_i} \{\mathrm{f_i}\,(x_i) + \lambda(\mathrm{a_i x_i})\} \tag{5.52}$$
$$sujeto\ a \quad C_i x_i \le d_i$$
$$x_i \in Z^{n_i}$$

Sustituyendo (5.52) en (5.51), la función dual asociada a las restricciones complejas del problema (P) queda:

$$\theta(\lambda) = \sum_{i=1}^{I} \theta_i(\lambda) - \lambda \mathrm{b} \tag{5.53}$$

Con ello se consigue transformar la Relajación Lagrangiana en I subproblemas independientes. Esto es adecuado cuando se busca una estrategia de distribución del problema para su resolución.

Por último, queda por encontrar los valores de los multiplicadores de Lagrange (λ) que proporcionan una mejor cota inferior de la solución al problema (P). Este es el problema dual lagrangiano:

$$(\mathrm{LD}) \qquad \max_{\lambda \geq 0} \theta(\lambda) \tag{5.54}$$

La solución óptima a este problema viene dada por:

$$\theta^* = \theta(\lambda^*), \quad \text{con} \quad \lambda^* = \arg\max_{\lambda} \theta(\lambda) \tag{5.55}$$

5.5. Revisión de los trabajos previos de aplicación de la Relajación Lagrangiana a programación de operaciones.

El algoritmo de Relajación Lagrangiana ha sido utilizado para la resolución de distintos problemas de programación de operaciones. La mayor parte de los trabajos realizan la formulación del problema de un modo similar, diferenciándose únicamente en las restricciones que se relajan. En algunos solo se relajan las restricciones de capacidad, mientras que en otros también se relajan las restricciones de precedencia. También aplican distintos métodos para la resolución del problema dual tal como se muestra en la Tabla 5.1. En los siguientes párrafos se muestran los principales trabajos, agrupados por el tipo de problema resuelto:

- Problema de programación de una sola máquina

Sun et al (1999) utilizan el algoritmo de Relajación Lagrangiana para resolver el problema de programación de una máquina con tiempos de preparación dependientes de la secuencia. La función objetivo a minimizar que utiliza es la suma ponderada de los cuadrados del retraso de los trabajos.

En Dewan y Joshi (2000) se realiza la programación de una sola máquina. Utiliza la Relajación Lagrangiana para descomponer el problema en varios subproblemas asociados a cada orden de trabajo (resolución de cada uno de los problemas que resulta de descomponer el problema relajado) y en un problema de optimización por parte de la máquina (resolución del problema dual). La máquina determina unos precios en cada iteración para los *slot* de tiempo en que se divide su horizonte de programación y las órdenes de trabajo realizan su optimización dados esos precios. Esto se plantea en un entorno dinámico donde nuevas órdenes de trabajo pueden entrar en cualquier momento. Cada vez que una orden se incorpora al sistema los precios se recalculan, tomando como precios iniciales los últimos precios previos a la llegada de la nueva orden.

Jeong y Leon (2005) resuelven el problema de programar un recurso compartido por tres subsistemas productivos sin utilizar un sistema de planificación central, aplicando una variación en la resolución del problema dual. Se evita la necesidad de un valor global de la función objetivo que sirva como referencia para calcular la actualización de los precios. Esto es importante cuando se quiere conseguir un problema completamente distribuido.

Tang et al (2007) estudian el problema de programación de una sola máquina con relaciones de precedencia en los trabajos minimizando la suma ponderada del retraso. Desarrollan un algoritmo de programación dinámica hacia delante y hacia atrás para resolver el problema relajado. Utilizan el método de Relajación Lagrangiana, relajando las restricciones de capacidad. El método de resolución del problema dual es el método del subgradiente.

Jiang y Tang (2008) utilizan la Relajación Lagrangiana para resolver el problema programación de tareas, donde los trabajos tiene fecha de emisión, mediante una variable continua en lugar de discretizar la variable tiempo en *slots* iguales.

- Problema de programación de máquinas paralelas

Luh et al (1990) aplican la Relajación Lagrangiana para programar tareas en varias máquinas idénticas. Realizan una revisión de los trabajos realizados hasta ese momento, definen el algoritmo iterativo a utilizar y actualizan los precios mediante el método del subgradiente. El programa factible lo obtienen mediante una heurística que utiliza la secuencia de tareas obtenida en el problema relajado y la penalización incremental de las tareas como criterio secundario.

Edis et al (2008) tratan de resolver el problema de programación de máquinas paralelas con restricciones adicionales de idoneidad de las máquinas (*machine eligibility restrictions*). En estos problemas distintas máquinas pueden realizar un mismo trabajo, pero con diferente rendimiento. Utilizan el método del subgradiente para actualizar los precios. Desarrollan una heurística para construir programas factibles.

Tang y Zhang (2009) plantean el problema de reprogramación de tareas en un entorno de máquinas paralelas. Dentro de las posibles incidencias que se pueden presentar en un entorno dinámico, se estudia la avería o parada de una de las máquinas.

Problema	subgradiente	Subgradiente surrogado	Subgradiente conjugado
Programación de una sola máquina	Sun et al (1999); Jeong y Leon (2005); Tang et al (2007); Jiang y Tang (2008)		
Programación en máquinas paralelas	Luh et al (1990); Edis et al (2008); Tang y Zhang (2009)		
Job shop	Hoitomt et al (1993); Chen et al (1995, 1998); Wang et al (1997); Dewan y Joshi (2002); Luh y Feng (2003); Liu et al (2004, 2007); Baptiste et al (2008); Jeong y Yim (2009)	Kaskavelis y Caramanis (1994); Zhao et al (1999); Fang et al (2000); Chen y Luh (2003)	Czerwinski y Luh (1994)
Flow shop	Tang et al (2006); Nishi et al (2007)		
Problema general de programación de la producción	Luh y Hoitomt (1993); Jeong y Leon (2002)		
Programación de entornos complejos	Masin et al (2007) ; Araúzo (2007)	Zhang et al (2000); Zhang et al (2001); Chen et al (2003); Sun et al (2006)	Gou et al (1998)
Programación y gestión de stocks	Tempelmeier y Derstroff (1996)		Czerwinski y Luh (1994)
Programación de proyectos	Araúzo et al (2010)	Ni et al (2008)	
Cadena de suministro		Chen y Chu (2003); Luh et al (2003)	

Tabla 5.1. Métodos de resolución del problema dual utilizados.

El objetivo es reprogramar las tareas con las mínimas modificaciones sobre el problema original, pero teniendo en cuenta la eficacia del programa obtenido. Para ello se plantea una nueva función objetivo que incorpora, además de la función objetivo original, unos términos de penalización a las desviaciones (adelanto o retraso) de las tareas programadas respecto al programa original.

- Problema *job shop*

Es el problema que ha recibido una mayor atención, aplicando la técnica de Relajación Lagrangiana. A continuación que se enumeran los trabajos más representativos:

En Hoitomt et al (1993) se plantea la resolución de un problema *job shop* mediante la relajación de las relaciones de precedencia y de las restricciones de capacidad de las máquinas. Proponen la utilización de la Relajación Lagrangiana aumentada, que consiste en añadir un término de penalización a la función de coste que induzca a un coste mayor por la violación de las restricciones. El coeficiente de penalización puede ser aumentado, de modo que se acerque a la solución del problema original.

La aportación principal de los trabajos de Chen et al (1995) y de Czerwinski y Luh (1992) es la utilización de técnicas de programación dinámica para resolver el problema relajado a nivel de órdenes de trabajo. De esta forma se evita la relajación adicional de las restricciones de precedencia, que otros trabajos señalan como causa de una oscilación excesiva. Aplican esta propuesta a distintos problemas e indican que una de las líneas a seguir es el desarrollo de métodos de construcción de programas factibles eficientes.

Wang et al (1997) proponen resolver el problema *Job shop* con una función objetivo que pondera el adelanto y el retraso de las órdenes de trabajo (suma ponderada de los cuadrados de estos valores). Utilizan el algoritmo de Relajación Lagrangiana. Para la resolución de los subproblemas individuales usan la programación dinámica hacia atrás (*backward dynamic programming*). Para la resolución del problema dual proponen y comparan los principales métodos. Finalmente utilizan e implementan el método del subgradiente conjugado intercalado.

En Gou et al (1998) se utiliza la Relajación Lagrangiana para resolver el problema *Job shop* y aprovecha la relajación de las restricciones de capacidad y precedencia para estructurar el problema y organizar una jerarquía siguiendo una estructura holónica cuasi-distribuida.

Chen et al (1998) se centran principalmente en la aplicación de la técnica de Relajación Lagrangiana a la resolución del problema *job shop*, mejorando los trabajos previos. En la aplicación hecha anteriormente a este mismo problema se relajaban tanto las restricciones de capacidad como las restricciones de precedencias entre variables. La relajación de estas últimas restricciones provocaba fuertes oscilaciones en las soluciones, lo que no garantizaba la convergencia del método. En este trabajo proponen no relajar las restricciones de precedencia, utilizando la técnica de programación dinámica para resolver cada uno de los subproblemas de optimización por parte de cada orden de trabajo que aparecen. Al no eliminar las relaciones de precedencia entre tareas, también proponen la utilización de este método para resolver el problema *job shop* con minimización del *makespan* como función objetivo.

En Kaskavelis y Caramanis (1998) se presentan mejoras sobre el algoritmo de Relajación Lagrangiana. Una es la aplicación del coste dual surrogado para el cálculo de la dirección del paso y también la actualización frecuente de los multiplicadores de Lagrange (*método del subgradiente intercalado)*. La segunda de las mejoras que plantea es el cálculo del tamaño del paso de cada iteración.

En Luh et al (1998) resuelven el problema *job shop* con algunas particularidades como considerar costes de preparación dependientes de la secuencia.

En Zhao et al (1999) también se aplica el método del subgradiente intercalado a partir del método del subgradiente surrogado. La ventaja de este método es que, a pesar de que las direcciones que se obtienen para resolver el problema dual son peores que en el método del subgradiente, el tiempo necesario para resolver los subproblemas es menor que en el segundo método, ya que no se debe encontrar el óptimo de cada subproblema, sino sólo una mejor solución. Desarrollan este método para un problema de optimización entera separable y lo aplican a un ejemplo con el problema *job shop*.

Fang et al (2000) revisan el algoritmo de Relajación Lagrangiana aplicado a la resolución del problema *job shop* para mejorar su funcionamiento en problemas de tamaño grande. Estas mejoras se centran, por una parte, en la rapidez en la resolución del problema dual, desarrollando un método para regular el tamaño del paso de forma que se mejore la convergencia del método, y, por otra parte, en la construcción de programas factibles, utilizando con éxito una heurística que utiliza la regla de prioridad SPT/CR.

La principal aportación de Dewan y Joshi (2002) es la relación que establecen entre la teoría sobre subastas y la Relajación Lagrangiana. Justifican la necesidad de la aplicación de sistemas distribuidos en los sistemas de planificación y control de la

producción. Utilizan el concepto de sistemas multiagente como base para su implantación que pretende resolver el problema de forma distribuida. Plantean el problema como una o varias subastas combinatorias donde los ítem vendidos son los *slot* de tiempo en los que se divide el horizonte de planificación de cada uno de los recursos a programar. Para resolver el problema de asignación de la subasta se apoyan en la Relajación Lagrangiana.

En Chen y Luh (2003) se plantea resolver el problema *job shop* utilizando un algoritmo de Relajación Lagrangiana, relajando las restricciones de precedencia en lugar de las restricciones de capacidad. Los subproblemas a resolver son equivalentes a la resolución de un problema de programación de una sola máquina o de máquinas paralelas. Para la resolución del problema dual utilizan el método del subgradiente surrogado, que permite resolver los subproblemas de forma aproximada. La ventaja de este método sobre el del subgradiente es que el método del subgradiente necesita calcular el valor óptimo de todos los subproblemas, que requiere utilizar el método de programación dinámica. La complejidad computacional de este método y el número de multiplicadores son proporcionales al horizonte de programación, por lo que puede ser muy costoso para horizontes de programación largos.

Kutanoglu y Wu (2004) tratan de mejorar la robustez del método de Relajación Lagrangiana combinando el análisis estocástico con la adaptación dinámica del sistema.

Liu et al (2004, 2007) definen un sistema de programación reactivo en un entorno *job shop*, donde se van realizando subastas cada vez que queda un *slot* de tiempo libre en una máquina. Pretenden utilizar un sistema predictivo mediante órdenes de fabricación ficticias previstas, pero todavía no reales. El sistema también pretende ser descentralizado, realizando una subasta por cada centro de trabajo. Sin embargo, para resolver el problema de actualización de precios necesita información global.

En Baptiste et al (2008) se trata de resolver el problema *job shop* minimizando la suma ponderada de los retrasos y los adelantos. Se prueban dos tipos distintos de Relajación Lagrangiana. En el primero, se relajan exclusivamente las restricciones de precedencia manteniendo las de capacidad. En el segundo, se relajan las restricciones de capacidad como se ha hecho en trabajos anteriores. Para calcular el programa factible utilizan el algoritmo Giffler-Thompson pero indican que no obtienen resultados cercanos al óptimo. Luego lo mejoran con búsqueda local.

Jeong y Yim (2009) aplican la Relajación Lagrangiana para resolver el problema *job shop*, pero extrapolado al caso de empresa virtual con agentes que controlan una o varias órdenes de trabajo y máquinas. El cálculo de los planes se realiza por separado en

los distintos subsistemas de producción, cada uno con sus correspondientes recursos, y son coordinados a su vez por un coordinador central. Estudian el efecto del tamaño del horizonte de programación sobre la calidad de la solución y del tiempo de cálculo. La experimentación realizada muestra que la calidad de la solución no se reduce si se reduce el horizonte de planificación, mientras que el tiempo de cálculo sí que se reduce significativamente.

- Problema *flow shop*

Tang et al (2006) aplican la Relajación Lagrangiana para el problema de programación de un sistema *flow shop* híbrido. Este sistema está compuesto por distintas etapas de producción, y en cada una de estas etapas existe un conjunto de máquinas en paralelo. Todas las órdenes de trabajo siguen la misma secuencia. En este trabajo se relajan las restricciones de precedencias, no las de capacidad. Con ello se pretende descomponer el problema por máquina en lugar de descomponerlo por órdenes de fabricación. Diseñan un algoritmo de programación dinámica para resolver los problemas locales. La dirección de actualización de los multiplicadores de Lagrange se calcula mediante el método del subgradiente.

En Nishi et al (2007) se pretende resolver el problema *flow shop* aplicando una versión del algoritmo de Relajación Lagrangiana. Se trata de aplicar de forma sucesiva el algoritmo de Relajación Lagrangiana, incorporando cada vez nuevas restricciones. Realizan propuestas para la mejora del paso de programación dinámica. Utilizan el método del subgradiente para resolver el problema dual.

- Problema general de programación de la producción

En Luh y Hoitomt (1993) se propone una metodología genérica que utiliza la Relajación Lagrangiana para resolver un problema de programación de la producción (*scheduling*). Lo aplican a la programación de tres tipos distintos de entornos. El primero es el de operaciones individuales que deben ser programadas en máquinas paralelas idénticas. El segundo es el de programación de órdenes de trabajo compuestas por varias operaciones que deben ser programadas en máquinas paralelas idénticas. El tercer problema es el de órdenes de trabajos con varias operaciones relacionadas con relaciones de precedencia generales que deben programarse en máquinas diversas.

Jeong y Leon (2002) desarrollan un protocolo llamado CICA basado en la Relajación Lagrangiana como marco para resolver problemas de asignación, entre ellos

el problema de programación de la producción, de forma distribuida, utilizado un entorno basado en agentes.

- Problema de programación de entornos complejos

Gou et al (1998) utilizan como base el método de Relajación Lagrangiana para distribuir el problema de programación de la producción en una planta de producción real. Lo implementan utilizando un sistema multiagente.

En Zhang et al (2000), el algoritmo de Relajación Lagrangiana se utiliza para realizar la programación en líneas de ensamblaje mixtas que permiten el ensamblaje de productos distintos con tiempos de preparación y cambio pequeños. Utilizan el método del gradiente surrogado intercalado para resolver el problema dual. Uno de los problemas que se revela es que aparecen oscilaciones cuando existen varios lotes iguales con los mismos pesos en la función objetivo. La solución que proponen es modificar ligeramente los pesos de forma que ya no sean iguales, con lo que el método converge mejor.

Zhang et al (2001) proponen utilizar también el algoritmo de Relajación Lagrangiana para realizar la programación donde la función objetivo penaliza los retrasos, la finalización anterior a la fecha de entrega, el tiempo total de proceso (lead-time) y la sobrecarga de los recursos. Se permiten montajes (varios componentes son necesarios en una operación) y que una operación tenga varias operaciones inmediatamente posteriores. Esto hace que las estructuras de proceso consideradas sean complejas, lo mismo que la resolución de los subproblemas.

Chen et al (2003) aplican el método de Relajación Lagrangiana para un problema de programación de múltiples recursos, tiempos de preparación y lotes de transferencia. Utilizan un método de programación dinámica para la resolución de los subproblemas relacionados con los lotes de transferencia más sencillo que el usado en trabajos anteriores.

También se utiliza la Relajación Lagrangiana en Cixing et al (2005) como propuesta de resolución del problema de asignación de recursos en un entorno de fabricación e-manufacturing, utilizando una subasta combinatoria como estructura básica de gestión.

En Sun et al (2006) se estudia el caso de estructuras de proceso complejo, con montajes y desmontajes y acoplamiento de máquinas. Esto hace que la resolución sea compleja. Se han desarrollado soluciones que utilizan el método de programación dinámica, pero solo funciona bien en el caso de que solo existan montajes o desmontajes. Proponen relajar únicamente alguna de las relaciones de precedencia,

puesto que surgen problemas de oscilación. Para mejorar su convergencia utilizan una función auxiliar de penalización. El problema dual se resuelve aplicando una modificación del método del subgradiente surrogado, el subgradiente surrogado normalizado.

Araúzo (2007) implementa un mecanismo de subastas para el control de sistemas de fabricación flexibles. En el proceso de definición de la subasta utiliza la analogía entre subastas y relajación lagrangiana, lo que proporciona un método matemático robusto en la actualización de los precios.

- Problema de programación y gestión de stocks

En Czerwinski y Luh (1994) se propone integrar la gestión de componentes con la programación de la producción, mejorando la tradicional gestión mediante sistemas MRP y reglas heurísticas para realizar la programación de la producción. Para resolver el problema dual utilizan el método del subgradiente conjugado.

Tempelmeier y Derstroff (1996) aplican el algoritmo de Relajación Lagrangiana al cálculo de tamaños de lotes en estructuras multinivel con tiempos de preparación. Con ello pretenden superar las limitaciones que tiene el MRP en la previsión de necesidades. Utiliza el método del subgradiente para resolver el problema dual.

- Problema de programación de proyectos

En Ni et al (2008) se aplica el algoritmo de Relajación Lagrangiana a la programación de proyectos de diseño. Uno de los aspectos en los que incide es la definición del problema y las relaciones entre tareas. Los subproblemas del problema relajado los resuelve mediante el algoritmo de programación dinámica hacia atrás y hacia delante (BFDP). El problema dual lo resuelve utilizando el método del subgradiente surrogado.

Araúzo et al (2010) proponen una metodología de programación dinámica de la cartera de proyectos de una empresa basada en agentes. Proponen un sistema multiagente donde los proyectos negocian la adquisición de los recursos mediante una subasta. Para la definición de la subasta utilizan la analogía que existe con el método de Relajación Lagrangiana. Este enfoque permite la gestión de los proyectos actuales de la empresa y proporciona criterios de decisión para la aceptación o el rechazo de nuevos proyectos.

- Cadena de suministro

Chen y Chu (2003) parten del trabajo de Tempelmeier y Derstroff (1996), y utilizan el método del gradiente surrogado para resolver el problema dual.

Luh et al (2003) plantean la aplicación de la Relajación Lagrangiana a un problema de cadena de suministro asemejándolo a un problema *job shop*. Utilizan una variación del Contract Net como protocolo de comunicación entre los agentes y el método del gradiente surrogado para resolver el problema dual.

5.6. Conclusiones del capítulo

En este capítulo se han presentado los fundamentos básicos del método de Relajación Lagrangiana mediante sus definiciones y principales propiedades. Esto permite un mejor conocimiento del método de cara a su aplicación en sistemas de programación de operaciones distribuidos.

Una de las propiedades más importantes es que la solución del problema relajado es siempre menor que la solución del problema original. Esto permite obtener una cota inferior a la solución del problema original. El proceso iterativo va acercando la solución del problema Lagrangiano a la solución del problema original. Esto es la base del método: se centra principalmente en la obtención de la cota inferior más cercana a la solución del problema original (cota Lagrangiana). Una de las grandes ventajas del método es que disponiendo de la cota inferior de la solución es posible evaluar la calidad de la solución obtenida.

El estudio realizado también ha permitido detectar las limitaciones del método. La principal radica en que el procedimiento no busca directamente el óptimo del problema original, sino que es necesario utilizar un método de transformación de soluciones del problema relajado en soluciones del problema original. Así mismo, debe destacarse el hecho de que si una solución al problema relajado cumple con las restricciones del problema original (es factible), esto no asegura que la solución relajada sea el óptimo, solo lo será en determinados casos -si se cumple la condición (5.17)-, siendo ésta una condición suficiente pero no necesaria de optimalidad. La transformación de la solución del problema dual no garantiza la obtención de la solución del problema original, sin embargo puede estar cerca y en cualquier caso acotada por el método.

La estructura de determinados problemas hace que al realizar la Relajación Lagrangiana de unas determinadas restricciones, el problema relajado pueda dividirse

en un número limitado de subproblemas independientes entre sí. Esto supone una ventaja cuando se utiliza una estrategia de distribución del problema para su resolución.

Por último, también se han revisado los trabajos que han aplicado este método a la resolución de la programación de operaciones en los últimos años, que ha sido muy variada y abarca la mayoría de los problemas tipo. En cuanto a la aplicación del método, cabe resaltar que se ha hecho relajando distintos tipos de restricciones, aunque destacamos que en la mayor parte de los casos se han relajado las restricciones de capacidad ya que son las que más complican el problema. En cuanto a la resolución del problema dual, los trabajos revisados no se han restringido a la utilización del método del subgradiente, sino que también se han propuesto variantes de este método que buscan mejorar la rapidez de resolución, la estabilidad de las soluciones obtenidas y la facilidad de resolución de los problemas.

En el siguiente capítulo se realiza la aplicación del método de Relajación Lagrangiana al problema de programación de talleres y se estudia su capacidad para generar métodos de programación de operaciones distribuidos.

Capítulo 6. Problema *job shop*: método de Relajación Lagrangiana y subastas

En el capítulo anterior se ha revisado el método de Relajación Lagrangiana para la resolución de problemas complejos. Es un método iterativo que trata de obtener la solución del problema a partir de su cota inferior mediante la resolución de su problema dual (LD). También se ha estudiado el caso en el que la aplicación de la Relajación Lagrangiana permite distribuir el problema en un número limitado de subproblemas independientes, cuya resolución conduce a la solución del problema principal.

En este capítulo se plantea la aplicación de la Relajación Lagrangiana como método de descomposición del problema *job shop* definido en el apartado 1.4. Esta descomposición permite dividirlo en subproblemas relacionados con las órdenes de trabajo, lo que supone una ventaja para su implementación en sistemas distribuidos como los sistemas multiagente.

El problema dual asociado al problema relajado puede ser resuelto siguiendo distintos métodos. Se analiza la aplicación al problema *job shop* de los principales métodos de resolución ya vistos en el capítulo anterior: el método del subgradiente, el del subgradiente conjugado y el del gradiente surrogado. Para este último, se plantean dos variantes que aprovechan la estructura distribuida en subproblemas (método del gradiente intercalado).

También se analizan, para el caso particular del problema *job shop*, los métodos para tratar dos cuestiones que se plantean en el algoritmo: la primera es la resolución del problema de minimización de la función lagrangiana asociada a cada subproblema, y la segunda es la construcción de soluciones factibles para el problema original a partir de la solución del problema relajado.

Por último se compara el método de Relajación Lagrangiana para el problema *job shop* con los algoritmos basados en subastas. Se establece la relación entre ambos métodos y se identifican las mejoras que el método de la Relajación Lagrangiana puede aportar a los mecanismos basados en subastas.

6.1. Aplicación del método de Relajación Lagrangiana al problema *job shop*

El problema *job shop* consiste en encontrar el programa óptimo de forma que se minimice una determinada función objetivo que depende de las fechas de inicio y de fin de las tareas, de forma que se cumplan las restricciones de programación. La formulación del problema ya se explicó en el capítulo 1. El problema a resolver (P) queda definido como:

$$(P)\ \min_{\delta_{ijhk}} \sum_{i=1}^{n} f_i(\delta_{ijhk}) \tag{6.1}$$

$$sujeto\ a \sum_{\mathrm{i}=1}^{\mathrm{n}} \sum_{\mathrm{j}=1}^{O_i} \delta_{ijhk} \leq M_{hk} \qquad \begin{matrix} k = 1,2,\dots,K \\ h = 1,2,\dots,m \end{matrix} \tag{6.2}$$

$$b_{ij} \leq B_i \qquad \begin{matrix} i = 1,2,\dots,n \\ j = 1,2,\dots,O_i \end{matrix} \tag{6.3}$$

$$c_{ij} \leq b_{il} \qquad \begin{matrix} i = 1,2,\dots,n \\ j = 1,2,\dots,O_i \\ l \in I_{ij} \end{matrix} \tag{6.4}$$

$$\sum_{j=1}^{O_i} \delta_{ijhk} \leq 1 \qquad \begin{matrix} i = 1,2,\dots,n \\ k = 1,2,\dots,K \end{matrix} \tag{6.5}$$

$$c_{ij} = b_{ij} + d_{ij} - 1 \qquad \begin{matrix} i = 1,2,\dots,n \\ j = 1,2,\dots,O_i \end{matrix} \qquad (6.6)$$

$$h_{ij} \in F(O_{ij}) \qquad F(O_{ij}) \subseteq H \qquad (6.7)$$

$$\exists\, i, j_1, j_2 : F\left(O_{i,j_1}\right) \cap F\left(O_{i,j_2}\right) \neq \emptyset \qquad (6.8)$$

$$\delta_{ijhk} = \begin{cases} 1 & \text{si k} \in \left[\mathrm{b_{ij}}, \mathrm{c_{ij}}\right] \wedge \mathrm{h} = \mathrm{h_{ij}} \\ 0 & \text{en caso contrario} \end{cases} \qquad \begin{matrix} i = 1,2,\dots,n \\ j = 1,2,\dots,O_i \\ k = 1,2,\dots,K \\ h = 1,2,\dots,m \end{matrix} \qquad (6.9)$$

En la expresión (6.1), la función $\sum_{i=1}^{n} f_i\left(\delta_{ijhk}\right)$ es la función objetivo del problema (P) y se calcula como el sumatorio de los valores que toma la función $f_i\left(\delta_{ijhk}\right)$ en cada una de las órdenes de trabajo del problema. La función f_i depende de las variables binarias δ_{ijhk}, que representan el programa de operaciones. En el caso más referenciado, esta función depende de los retrasos (T_i) , que son la diferencia entre la fecha de finalización de la orden i y la fecha de entrega comprometida, siempre que el valor de la diferencia sea positivo:

$$T_i\left(\delta_{ijhk}\right) = \max\left(0, \max_j\left(c_{ij}\left(\delta_{ijhk}\right)\right) - D_i\right) \qquad (6.10)$$

Las funciones más habituales son la suma ponderada de los retrasos (6.11) y la suma ponderada de los retrasos al cuadrado (6.12).

$$f_i\left(\delta_{ijhk}\right) = w_i T_i\left(\delta_{ijhk}\right) \qquad (6.11)$$

$$f_i\left(\delta_{ijhk}\right) = w_i T_i^2\left(\delta_{ijhk}\right) \qquad (6.12)$$

Como ya se ha comentado, la idea básica de la Relajación Lagrangiana es relajar aquellas restricciones consideradas complejas o difíciles, de modo que el problema resultante sea mucho más fácil de resolver. Para ello se eliminan esas restricciones y se incorporan a la función objetivo, como un término que penaliza el incumplimiento de la restricción.

En el caso del problema *job shop* las restricciones complejas son las de capacidad, ya que establecen dependencias entre las variables de programación de cada una de las órdenes de trabajo, que de otro modo serían independientes. Esto es lo que hace que el problema sea complejo. Si en la expresión (6.1) se introducen las *M* x *K* restricciones de capacidad (6.2) multiplicadas cada una por un escalar positivo ($\lambda_{hk} \geq 0$), se obtiene la

Relajación Lagrangiana del problema original (Luh et al 1999), donde las restricciones de la orden *i* hacen referencia a las restricciones de la (6.3) a la (6.9):

$$(LR_\lambda) \quad \min_{\delta_{ijhk}} \left(\sum_{i=1}^{N} f_i(\delta_{ijhk}) + \sum_{h=1}^{M} \sum_{k=1}^{K} \lambda_{hk} \left(\sum_{i=1}^{N} \sum_{j=1}^{O_i} \delta_{ijhk} - M_{hk} \right) \right) \tag{6.13}$$
$$\lambda_{hk} \geq 0$$
$$\text{sujeto a restricciones de la orden i} \qquad i=1,2,\ldots, N$$

Se trata de un problema separable de programación entera, por lo que, como se mostró en el capítulo anterior, es posible reagrupar términos en función de las órdenes:

$$(LR_\lambda) \quad \min_{\delta_{ijhk}} \left(\sum_{i=1}^{N} \left(\mathrm{f_i}(\delta_{ijhk}) + \sum_{h=1}^{M} \sum_{k=1}^{K} \sum_{j=1}^{O_i} \lambda_{hk} \delta_{ijhk} \right) - \sum_{h=1}^{M} \sum_{k=1}^{K} \lambda_{hk} M_{hk} \right) \tag{6.14}$$
$$\lambda_{hk} \geq 0$$
$$\text{sujeto a restricciones de la orden i} \qquad i=1,2,\ldots, N$$

El problema (6.14) está compuesto por el sumatorio de *N* términos correspondientes a cada orden, más otro término que es el sumatorio de los *M* x *K* términos independientes del problema, los multiplicadores de Lagrange (λ_{hk}) multiplicados por las constantes que representan la capacidad de cada slot (M_{hk}), por lo que se puede expresar como:

$$(LR_\lambda) \quad \sum_{i=1}^{N} \left(\min_{\delta_{ijhk}} \left(\mathrm{f_i}(\delta_{ijhk}) + \sum_{h=1}^{M} \sum_{k=1}^{K} \sum_{j=1}^{O_i} \lambda_{hk} \delta_{ijhk} \right) \right) - \sum_{h=1}^{M} \sum_{k=1}^{K} \lambda_{hk} M_{hk} \tag{6.15}$$
$$\text{sujeto a} \quad \lambda_{hk} \geq 0$$
$$\text{restricciones de la orden } i \qquad i = 1, 2, \ldots, N$$

Dados unos multiplicadores de Lagrange (λ_{hk}), el problema se descompone en *N* subproblemas de minimización, cada uno de ellos asociado a una orden de trabajo (6.16) menos la suma de los *M* x *K* términos independientes, obteniendo (6.17).

$$\min_{\substack{\delta_{ijhk} \\ \lambda_{hk} \geq 0}} \left(\mathrm{f_i}(\delta_{ijhk}) + \sum_{h=1}^{M} \sum_{k=1}^{K} \sum_{j=1}^{O_i} \lambda_{hk} \delta_{ijhk} \right) = \min_{\delta_i} \mathrm{L_i}(\lambda, \delta_{ijhk}) = \theta_\mathrm{i}(\lambda) \tag{6.16}$$
$$\text{sujeto a restricciones de la orden } i$$

$$(LR_\lambda) \quad \sum_{i=1}^{n}\left(\theta_i(\lambda)\right) - \sum_{h=1}^{M}\sum_{k=1}^{K}\lambda_{hk}M_{hk} \quad = \theta(\lambda) \qquad \lambda_{hk} \geq 0 \tag{6.17}$$

El método de Relajación Lagrangiana busca soluciones a un problema mediante la transformación de soluciones obtenidas en el problema relajado. El problema relajado proporciona una cota a la solución del problema original (apartado 5.2) y se espera que la cotas más cercanas proporcionen mejores soluciones al problema original. El problema dual (6.18) busca los valores de los multiplicadores de Lagrange (λ_{hk}) que proporcionan una mejor cota inferior de la solución al problema (P). Esto proporcionará la cota lagrangiana más ajustada a la solución del problema original [v(P)].

$$(LD) \quad \max_{\lambda_{hk}}\left(\sum_{i=1}^{n}\left(\theta_i(\lambda)\right) - \sum_{h=1}^{M}\sum_{k=1}^{K}\lambda_{hk}M_{hk}\right) \qquad \text{sujeto a} \quad \lambda_{hk} \geq 0 \tag{6.18}$$

El problema dual (LD) es un problema en el espacio dual de los multiplicadores lagrangianos (λ_{hk}), mientras que el problema relajado (LR_λ) es un problema en las variables δ_{ijhk}. Como se vio en el capítulo anterior, existen distintos métodos para resolver este problema. Elegimos el método del subgradiente y sus derivados por la sencillez en su resolución y su convergencia, así como por su similitud con los mercados, como se desarrollará al final del capítulo. En el siguiente punto se desarrolla la aplicación de estos métodos al problema *job shop*.

6.2. Resolución del problema dual Lagrangiano en el problema *job shop*

En este apartado se presentan los métodos de resolución del problema dual Lagrangiano en los términos específicos del problema *job shop* que se ha definido. Primero se presentará el método del subgradiente, como método principal de resolución del problema, el método del subgradiente conjugado como método derivado del anterior, y por último se presentarán dos variantes del método del gradiente surrogado denominadas método del gradiente intercalado.

6.2.1. Método del subgradiente

Para resolver el problema dual se deben conocer los valores de las variables δ_{ijhk} que hacen mínimo el problema relajado, pero para obtener éstos es necesario conocer previamente los multiplicadores de Lagrange (λ_{hk}). Para calcular ambas variables (apartado 5.3.1) se propone un procedimiento iterativo denominado método del subgradiente, que aquí se particulariza para el problema de programación de talleres. El algoritmo se estructura en dos niveles: (1) el nivel local donde, dados unos valores de los multiplicadores de Lagrange (λ_{hk}) se resuelven los *N* subproblemas θ_i de (6.16), que en este problema están asociados a las órdenes de trabajo, (2) el nivel general donde se resuelve el problema dual, obteniendo en cada iteración el valor de los multiplicadores de Lagrange (λ_{hk}) a partir de la información generada en la resolución de los subproblemas θ_i. De esta forma se avanza hacia la resolución del problema dual (LD) (6.18) que depende los multiplicadores de Lagrange (λ_{hk}), tal y como se muestra en la Figura 22. El algoritmo es el siguiente:

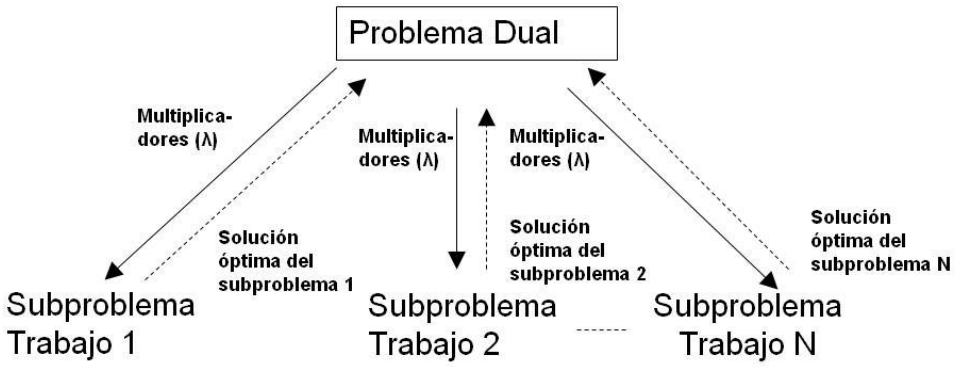

Figura 22. Estructura de resolución del método del subgradiente para el problema *job shop*.

- *Inicialización de valores:* se asignan unos valores iniciales a los multiplicadores de Lagrange (λ_{hk}).

$$\lambda_{hk} = 0 \qquad \begin{matrix} h = 1,2,\dots,H \\ k = 1,2,\dots,K \end{matrix} \tag{6.19}$$

- *Paso 1. Resolución del problema relajado:* dados unos λ_{hk} se resuelve el problema relajado resolviendo los *N* subproblemas correspondientes a cada orden:

$$\theta_i(\lambda^r) = \min_{\delta_{ijhk}} \left(\mathrm{f_i}(\delta_{ijhk}) + \sum_{h=1}^{M} \sum_{k=1}^{K} \sum_{j=1}^{O_i} \lambda_{hk}^r \delta_{ijhk} \right) \quad (6.20)$$
$$sujeto\ a\ restricciones\ de\ la\ orden\ i$$

obteniendo así las soluciones δ_{ijhk}^{*RL} de los subproblemas relajados $\theta_i(\lambda^r)$:

$$\delta_i^{*RL} = \arg\min_{\delta_{ijhk}} \left(\mathrm{f_i}(\delta_{ijhk}) + \sum_{h=1}^{M} \sum_{k=1}^{K} \sum_{j=1}^{O_i} \lambda_{hk}^r \delta_{ijhk} \right) \quad (6.21)$$
$$sujeto\ a\ restricciones\ de\ la\ orden\ i$$

La resolución de este problema se tratará en el apartado 6.3.

- *Paso 2. Cálculo del subgradiente:* se calcula el desajuste entre la capacidad de los recursos y la demanda en cada intervalo de tiempo *hk*. Dadas las soluciones de los subproblemas relajados $\theta_i(\lambda^r)$ calculados en el paso anterior, los incumplimientos de las restricciones de capacidad (6.2) constituirán el vector subgradiente:

$$\mathrm{g_{hk}^r} = \sum_{\mathrm{i=1}}^{\mathrm{n}} \sum_{\mathrm{j=1}}^{O_i} \delta_{\mathrm{ijhk}}^{*r\,RL} - M_{hk} \qquad \begin{matrix} k = 1,2,\dots,K \\ h = 1,2,\dots,m \end{matrix} \quad (6.22)$$

- *Paso 3. Generación de una solución factible:* en general, la solución obtenida como agregación de las soluciones de los subproblemas no cumplirá las restricciones de capacidad del problema original que habían sido eliminadas en el problema relajado. A partir de la solución obtenida se busca una solución δ_{p} que cumpla con las restricciones del problema mediante algún método constructivo F que transforme las soluciones del problema relajado:

$$\begin{matrix} F\colon \Delta \to \Delta_{\mathrm{p}} \\ \delta \to \delta_{\mathrm{p}} \end{matrix} \quad (6.23)$$

Este paso se desarrolla en el apartado 6.4.

- *Paso 4. Criterio de parada:* se establecen los criterios de parada para el algoritmo. En caso de que no se cumplan, sigue en el Paso 5. Los criterios más habituales de parada son:

  - El valor calculado del gap de dualidad es muy pequeño. Como los valores óptimos de (P) y de (LD) son desconocidos a priori, el gap de dualidad se calcula como la diferencia (6.27) entre el mejor valor de la función calculado $f(\delta_{\mathrm{p}})$, que será la cota superior (CS) de la solución del problema

original (6.24) y el mejor valor del problema relajado calculado, que será la cota inferior (CI) de la solución del problema original (6.25). Esta diferencia proporciona una cota superior del gap de dualidad. El valor proporciona una medida de la suboptimalidad del programa factible respecto al óptimo.

$$CS^{r+1} = V\left[f\left(\delta_p^{r+1}\right)\right] = \min\left[CS^r, f\left(\delta_p^r\right)\right] \tag{6.24}$$

$$CI^{r+1} = V[\theta(\lambda^{r+1})] = \max[CI^r, \theta(\lambda^r)] \tag{6.25}$$

donde

$$\theta(\lambda^r) = \sum_{i=1}^{n}\left(\theta_i(\lambda)\right) - \sum_{h=1}^{M}\sum_{k=1}^{K}\lambda_{hk}M_{hk} \tag{6.26}$$

$$GAP = CS - CI < \varepsilon_1 \; con \quad \varepsilon_1 \to 0 \tag{6.27}$$

- El vector de multiplicadores de Lagrange ha variado muy poco de una iteración a la siguiente:

$$\|\lambda^{r+1} - \lambda^r\| < \varepsilon_2 \; con \quad \varepsilon_2 \to 0 \tag{6.28}$$

- La cota inferior (CI) de (6.25) no ha sido actualizada en un determinado número de iteraciones.

❖ *Paso 5. Actualización de los multiplicadores de Lagrange:* en este paso se busca mejorar el valor de la función dual. Para ello se actualizan los multiplicadores de Lagrange que se utilizarán en la siguiente iteración (λ_{hk}^{r+1}) . Básicamente consiste en sumar a los multiplicadores actuales un múltiplo del vector subgradiente:

$$\lambda^{r+1} = \max(0, \lambda^r + \alpha^r g^r) \tag{6.29}$$

donde el tamaño del paso se calcula como:

$$\alpha^r = \gamma_r \frac{\left(V[f(x_p)] - \theta(\lambda^r)\right)}{\|g^r\|^2} \qquad \text{con } 0 < \gamma_r \leq 2 \tag{6.30}$$

A continuación se muestran algunas variantes del método. Estas variantes afectan a la forma de calcular el vector subgradiente o la dirección de actualización de los multiplicadores de Lagrange.

- Método del subgradiente conjugado

El método del subgradiente conjugado busca reducir la oscilación en las soluciones proporcionadas por el método del subgradiente ponderando en el cálculo de la dirección de actualización de los multiplicadores de Lagrange (6.31) la dirección del subgradiente con los subgradientes de iteraciones anteriores (6.32), como se explicó en el apartado 5.3. Para el caso particular de resolución del problema *job shop*, el método del subgradiente conjugado coincide en todos los pasos con el método del subgradiente excepto en el paso 5.

$$\lambda^{r+1} = \max(0, \lambda^r + \alpha^r d^r) \tag{6.31}$$

$$d^r = \begin{cases} g^0 & si\ n = 0 \\ \beta g^r + (1-\beta) d^{r-1} & en\ \ otro\ caso \end{cases} \tag{6.32}$$

$$con\ 0 < \beta < 1$$

El cálculo del tamaño del paso se calcula como en el método del subgradiente (6.30) sustituyendo el vector subgradiente por la dirección de actualización calculada:

$$\alpha^r = \gamma^r \frac{(V[f(x_p)] - \theta(\lambda^r))}{\|d^r\|^2} \qquad con\ 0 < \gamma < 2 \tag{6.33}$$

- Método del subgradiente normalizado

Utilizado por Kaskavelis y Caramanis (1998) y Sun et al (2006). Este método busca determinar un tamaño de paso diferente para cada recurso. Esto se justifica por el hecho de que cada recurso no va a estar igualmente ocupado, por lo que la velocidad de actualización de sus precios podría ser diferente.

El tamaño del paso del cada máquina h se computa según:

$$\alpha_{\mathrm{h}}^{\mathrm{r}} = \gamma^r \frac{V[f(\delta_p)] - \theta(\lambda^r)}{\left\| g^r(\lambda_{\mathrm{h}}^{\mathrm{r}}) \right\|^2} \qquad con\ 0 < \gamma^n < 2 \tag{6.34}$$

Esta expresión es similar a la utilizada en el método del subgradiente (6.30) sustituyendo el vector subgradiente por el subgradiente de la máquina *h:*

$$g_h^r = g^r(\lambda_n^r) = \{g_{hk}\ \ con\ k = 1,2, \dots, K\} \tag{6.35}$$

La actualización de precios se realiza mediante:

$$\lambda_{\mathrm{hk}}^{r+1} = \max(0, \lambda_{\mathrm{hk}}^r + \alpha_{\mathrm{h}}^{\mathrm{r}} g_{\mathrm{hk}}^r) \qquad \begin{matrix} k = 1,2, \dots, K \\ h = 1,2, \dots, m \end{matrix} \tag{6.36}$$

- Actualización de los multiplicadores

Beasley (1993) y Wang (2003) sugieren mejorar el cálculo del tamaño del paso utilizando un subgradiente modificado. Este método permite aumentar la rapidez de convergencia dado que, cuando el horizonte de programación es grande, hay muchas restricciones de capacidad y muchas de ellas están inactivas (como estas restricciones no son violadas los multiplicadores serán cero). Las restricciones inactivas corresponden a los *slot* que no han sido activados en el problema relajado $[\theta_i(\lambda^r)]$ de ninguna orden de trabajo i y en ninguna iteración r. Por tanto, para estos *slots* se cumple que:

$$\delta^r_{\mathrm{hk}} = 0 \qquad \begin{matrix} k = 1,2,\dots,K \\ i = 1,2,\dots,N \\ j = 1,2,\dots,O_i \end{matrix} \tag{6.37}$$

En los *slots* inactivos (Figura 23) se cumple que el gradiente es igual a la capacidad de la máquina en valor negativo ($g^r_{hk} = -M_{hk}$) y los multiplicadores de Lagrange son los iniciales ($\lambda^k_{hk} = \lambda^0_{hk} = 0$). Dado el subgradiente g^r y λ^r el vector multiplicadores de Lagrange:

$$g^r = [g^r_{hk}] \qquad \lambda^{\mathrm{r}} = [\lambda^r_{hk}] \qquad \begin{matrix} \mathrm{h} = 1,2,\dots,\mathrm{m} \\ \mathrm{k} = 1,2,\dots,\mathrm{K} \end{matrix} \tag{6.38}$$

se define el subgradiente modificado según (6.39) en el que se da valor 0 a los *slots* inactivos y al resto el valor del subgradiente.

$$\hat{g}^r = [\hat{g}^r_{hk}] \qquad \begin{matrix} \mathrm{h} = 1,2,\dots,\mathrm{m} \\ \mathrm{k} = 1,2,\dots,\mathrm{K} \end{matrix} \tag{6.39}$$

$$\text{con} \quad \hat{g}^k_{hk} = \begin{cases} 0 & si\ g^k_{hk} = -\mathrm{M_{hk}}\ y\ \ \lambda^k_{hk} = 0 \\ g^k_{hk} & en\ otro\ caso \end{cases}$$

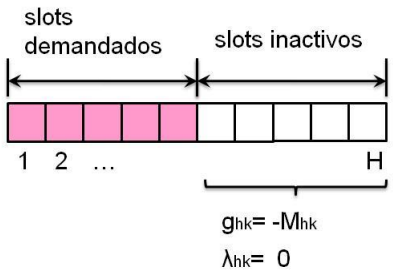

Figura 23. *Slots* inactivos del horizonte de programación de la máquina h

En Wang (2003) se demuestra que si los multiplicadores de Lagrange se actualizan mediante:

$$\lambda^{r+1} = \max(0, \lambda^r + \alpha^r \hat{g}^r) \tag{6.40}$$

y si el tamaño del paso α^r cumple:

$$\alpha^r = \gamma_r \frac{\theta(\lambda^*) - \theta(\lambda^r)}{\|\hat{g}^r\|^2} \qquad \text{con} \quad 0 < \gamma_r \leq 2 \tag{6.41}$$

entonces se verifica:

$$\|\lambda^{r+1} - \lambda^*\| < \|\lambda^r - \lambda^*\| \tag{6.42}$$

con lo que se asegura la convergencia, ya que la solución va estando más cercana en cada iteración.

Con ello se consigue aumentar el rango en el tamaño del paso ya que la norma del vector subgradiente modificado $\|\hat{g}^r\|$ no es mayor que la norma del vector subgradiente $\|g^r\|$ y, por tanto, la velocidad de convergencia puede aumentar. Las restricciones inactivas aumentan innecesariamente la norma del subgradiente, lo que provoca que el tamaño del paso sea mucho menor del que debiera ser. Al ajustar los tamaños del paso se puede reducir este efecto.

6.2.2. Método del gradiente intercalado

En este punto se presentan dos variantes del método del gradiente intercalado derivadas del método del gradiente surrogado (apartado 5.3.3), particularizado para el problema *job shop* objeto de estudio. La característica de este método frente al método del subgradiente es que para garantizar su convergencia no es necesario que se realice la optimización del problema relajado en cada iteración, sino que se exige que se mejore el valor de la función surrogada con los multiplicadores de Lagrange actualizados respecto al valor obtenido con la solución de la iteración anterior. Esto se representa matemáticamente con las variables del problema *job shop* mediante:

$$\tilde{L}(\lambda^r, \delta^r) < \tilde{L}(\lambda^r, \delta^{r-1}) \tag{6.43}$$

donde la función Lagrangiana surrogada $\tilde{L}(\lambda, \delta)$ se define mediante (6.44). Esta función está compuesta por *n* sumandos relacionados con las órdenes de trabajo del

problema denominadas subfunciones surrogadas relacionadas con las órdenes de trabajo $\tilde{L}_{\mathrm{i}}(\lambda, \delta_i)$ definidas en (6.45).

$$\tilde{L}(\lambda, \delta) = \tilde{L}(\lambda_{hk}, \delta_{ijhk}) = \sum_{i=1}^{n} \left(\tilde{L}_{\mathrm{i}}(\lambda_{hk}, \delta_{ijhk}) \right) - \sum_{h=1}^{M} \sum_{k=1}^{K} \lambda_{hk} M_{hk} \qquad (6.44)$$

$$\lambda_{hk} \geq 0$$

$$\text{con} \quad \tilde{L}_{\mathrm{i}}(\lambda_{hk}, \delta_{ijhk}) = \mathrm{f_i}(\delta_{ijhk}) + \sum_{h=1}^{M} \sum_{k=1}^{K} \sum_{j=1}^{O_i} \lambda_{hk} \delta_{ijhk} \qquad (6.45)$$

$$\lambda_{hk} \geq 0$$

$$\text{sujeto a restricciones de la orden } i$$

Este método ha sido utilizado tanto por Zhao et al (1997) como por Kaskavelis y Caramanis (1998), con alguna diferencias en la definición de la función Lagrangiana surrogada y en la estructura del método de resolución.

- Primera variante del gradiente intercalado

Una de las variantes del método es la desarrollada en Zhao et al (1997). Aprovechan que el problema relajado del problema *job shop* se puede descomponer en *N* subproblemas relacionados con las órdenes de trabajo. Proponen que se cumpla la condición de convergencia del método del gradiente surrogado de la expresión (6.43) optimizando tan sólo uno de los subproblemas, mientras que el resto no varía. Esto implica actualizar tanto el valor de los multiplicadores como el tamaño del paso en cada iteración.

Se define la función Lagrangiana surrogada según (6.46), de forma que en cada iteración solo se optimice el subproblema relacionado con la orden de trabajo (i_s (6.48), mientras que el resto permanece igual que en la iteración anterior (6.47).

$$\tilde{L}^{\mathrm{r}} = \tilde{L}(\lambda_{hk}^{r}, \delta_{\mathrm{ijhk}}^{\mathrm{r}}) = \sum_{i=1}^{N} \left(\tilde{L}_{\mathrm{i}}(\lambda_{hk}^{r}, \delta_{\mathrm{ijhk}}^{\mathrm{r}}) \right) - \sum_{h=1}^{M} \sum_{k=1}^{K} \lambda_{hk}^{r} M_{hk} \qquad (6.46)$$

$$\text{con} \quad \tilde{L}_{\mathrm{i}}^{\ \mathrm{r}} = \begin{cases} \theta_{\mathrm{i}}^{\mathrm{r}}(\lambda_{hk}^{r}) = \min\limits_{\delta_{ijhk}} \mathrm{L_i}(\lambda^{r}, \delta_{ijhk}) & \text{si i} = \mathrm{i_s} \\ \tilde{L}_{\mathrm{i}}(\lambda_{hk}^{r}, \delta_{\mathrm{ijhk}}{}^{r-1}) & \text{si i} \neq \mathrm{i_s} \end{cases} \qquad (6.47)$$

$$\min_{\delta_{\mathrm{i_s}jhk}} \mathrm{L_{i_s}}(\lambda^{r}, \delta_{\mathrm{i_s}jhk}) = \min_{\delta_{\mathrm{i_s}jhk}} \left(\mathrm{f_{i_s}}(\delta_{\mathrm{i_s}jhk}) + \sum_{h=1}^{M} \sum_{k=1}^{K} \sum_{j=1}^{O_i} \lambda_{hk}^{r} \delta_{\mathrm{i_s}jhk} \right) \qquad (6.48)$$

y las soluciones en cada iteración son:

$$\delta_{ijhk}^{r} = \begin{cases} \arg\min_{\delta_{ijhk}} \mathrm{L_i}(\lambda^r, \delta_{ijhk}) & \text{si i} = \mathrm{i_s} \\ \delta_{ijhk}^{r-1} & \text{si i} \neq \mathrm{i_s} \end{cases} \quad (6.49)$$

La ventaja de este método es que no es necesario que la optimización de los subproblemas se realice de forma ordenada, sino que puede realizarse al azar hasta completar el ciclo. Esto tiene consecuencias respecto al asincronismo a la hora de implementar el método en un sistema distribuido.

En esta variante del método del gradiente surrogado (apartado 5.3.3), la condición de mejora del valor de la función Lagrangiana surrogada (*paso 1*) aprovecha la descomposición del problema *job shop* mediante la Relajación Lagrangiana que permite optimizar en cada iteración tan solo uno de los subproblemas resultantes. A continuación se muestra el algoritmo, particularizado para el problema *job shop*:

- *Inicialización de valores:* al igual que en el método del gradiente surrogado general, se asignan unos valores iniciales a los multiplicadores de Lagrange (λ^r) y se realiza una primera interación siguiendo el método del subgradiente.
- *Paso 1. Mejora de la función Lagrangiana surrogada:* en este paso se optimiza el subproblema asociado a la orden i_s, teniendo como resultado el programa asociado a la orden ($\delta_{\mathrm{i_s}jhk}^{r}$). Los programas asociadas al resto de órdenes se mantienen (6.49). De esta forma mejora el valor de la función Lagrangiana surrograda (6.46), cumpliendo la condición de convergencia (6.43).
- *Paso 2. Cálculo del subgradiente surrogado:* una vez calculado el valor de las variables del problema ($\delta_{\mathrm{i}jhk}^{r}$), se calcula el subgradiente surrogado que representa el incumplimiento de las restricciones relajadas:

$$\hat{\mathrm{g}}_{\mathrm{hk}}^{\mathrm{r}} = \sum_{\mathrm{i}=1}^{\mathrm{N}} \sum_{\mathrm{j}=1}^{O_i} \delta_{\mathrm{ijhk}}^{r} - M_{hk} \qquad \begin{matrix} k = 1,2,\dots,K \\ h = 1,2,\dots,m \end{matrix} \quad (6.50)$$

- *Paso 3. Generación de una solución factible:* este paso es igual al del método del subgradiente (apartado 6.2.1).
- *Paso 4. Criterio de parada:* el criterio de parada en este método es que si el vector de multiplicadores de Lagrange ha variado muy poco de una iteración a la siguiente el algoritmo se para (6.51), en caso contrario continúa con el paso 5.

$$\|\lambda^{r+1} - \lambda^r\| < \varepsilon_2 \; con \quad \varepsilon_2 \to 0 \quad (6.51)$$

❖ *Paso 5. Actualización de los multiplicadores de Lagrange:* se actualizan los multiplicadores de Lagrange mediante:

$$\lambda^{r+1} = \max(0, \lambda^r + \alpha^r \tilde{g}^r) \tag{6.52}$$

teniendo en cuenta que la dirección de actualización será la del gradiente surrogado (6.50) y el tamaño del paso se calcula como:

$$\alpha^r = \gamma^r \frac{V[f(\delta_p)] - \tilde{\mathrm{L}}(\lambda^r, \delta^r)}{\|\hat{\mathrm{g}}(\delta^r)\|^2} \quad con\ 0 < \gamma^r < 1 \tag{6.53}$$

El cálculo del valor de la función Lagrangiana surrogada ($\tilde{L}^r$) se realiza mediante la expresión (6.46). Se actualiza el contador *r* según (6.54) y el contador $\mathrm{i_s}$ según (6.55) y se continúa con el paso 1.

$$\mathrm{r} = \mathrm{r} + 1 \tag{6.54}$$

$$\mathrm{i_s} = \begin{cases} \mathrm{i_s} + 1 & \text{si } \mathrm{i_s} < n \\ 1 & \text{si } \mathrm{i_s} = \mathrm{n} \end{cases} \tag{6.55}$$

- Segunda variante del gradiente intercalado

Método propuesto por Kaskavelis y Caramanis (1998). Al igual que en la primera variante, la idea básica consiste en actualizar los multiplicadores después de resolver cada subproblema en lugar de hacerlo tras resolver todos los subproblemas.

Esta propuesta difiere de la anterior en que el cálculo del paso se actualiza cada vez que se han resuelto todos los subproblemas. De esta forma, cuando se resuelve uno de los subproblemas se actualizan los precios, y son esos precios los que utiliza el siguiente subproblema para calcular la nueva solución. Esto se realiza para todos los subproblemas, de modo que cada uno se resuelve con unos precios distintos. El tamaño del paso se actualiza cuando todos los subproblemas se han actualizado, y la función Lagrangiana se calcula como suma de las subfunciones de cada subproblema calculadas con los precios que han ido recibiendo.

El algoritmo sigue una estructura diferente a los anteriores, puesto que se define un ciclo principal y un subciclo de actualización de los multiplicadores. El ciclo principal está asociado al cálculo del tamaño del paso y de la solución del problema. El subciclo está asociado a la actualización de los multiplicadores y a la optimización de uno de los subproblemas $\theta_i(\lambda)$. Para completar un ciclo principal es necesario que todos los subproblemas a nivel de orden de trabajo hayan sido resueltos una vez.

Se define la función lagrangiana surrogada según la expresión (6.56), en la que *r* es el contador asociado al ciclo principal e i_s el contador asociado al subciclo.

$$\hat{L}^{r,i_s} = \sum_{\mathrm{i}=1}^{i_s} \theta_{\mathrm{i}}(\lambda^{r,i}) + \sum_{\mathrm{i}=i_s+1}^{\mathrm{N}} \theta_{\mathrm{i}}(\lambda^{r-1,\mathrm{i}}) - \sum_{h=1}^{M}\sum_{k=1}^{K} \lambda_{hk}^{r,i_s} M_{hk} \tag{6.56}$$

$$con \quad \theta_i(\lambda^{r,i}) = \min_{\delta_{ijhk}} \left(\mathrm{f_i}(\delta_{ijhk}) + \sum_{h=1}^{M}\sum_{k=1}^{K}\sum_{j=1}^{O_i} \lambda_{hk}^{r,i}\delta_{ijhk} \right) \tag{6.57}$$

$$sujeto\ a\ restricciones\ de\ la\ orden\ i$$

Es decir, la función lagrangiana surrogada está compuesta por la suma de:

- Para los subproblemas ya resueltos en la iteración *r* del ciclo principal: el valor de la función lagrangiana surrogada del subproblema *i* con los multiplicadores de la subiteración *i* y del ciclo *r*

$$\sum_{\mathrm{i}=1}^{i_s} \theta_{\mathrm{i}}(\lambda^{r,i}) \tag{6.58}$$

- Para los subproblemas todavía no resueltos en la iteración *r* del ciclo principal: el valor de la función lagrangiana surrogada del subproblema *i* calculada con los multiplicadores de la subiteración *i* y del ciclo *r*-1

$$\sum_{\mathrm{i}=i_s+1}^{\mathrm{N}} \theta_{\mathrm{i}}(\lambda^{r-1,\mathrm{i}}) \tag{6.59}$$

- Sumatorio de los multiplicadores de la subiteración $\mathrm{i_s}$ del ciclo *r*

$$\sum_{h=1}^{M}\sum_{k=1}^{K} \lambda_{hk}^{r,i_s} M_{hk} \tag{6.60}$$

El gradiente surrogado en la iteración i_s se calcula con las soluciones al problema θ_{i} calculado en la subiteración correspondiente:

$$\hat{\mathrm{g}}\left(\lambda_{\mathrm{hk}}^{\mathrm{r},i_s}\right) = \left[\sum_{\mathrm{i}=1}^{i_s}\sum_{\mathrm{j}=1}^{O_i} \delta_{\mathrm{ijhk}}^{r,i} + \sum_{\mathrm{i}=i_s+1}^{\mathrm{n}}\sum_{\mathrm{j}=1}^{O_i} \delta_{\mathrm{ijhk}}^{r-1,i} \right] - M_{hk} \tag{6.61}$$

$$para \quad k = 1,2,\dots,K$$
$$h = 1,2,\dots,m$$

Visto de otro modo, en cada subiteración se calcula la subfunción lagrangiana correspondiente al problema i_s, $\theta_i(\lambda^{r,i_s})$.

Como dentro de una iteración *r* del ciclo principal, un determinado multiplicador λ_{hk} puede ser actualizado varias veces en las distintas subiteraciones *i*, el tamaño del paso se normaliza mediante F_h para asegurar que en una iteración completa todos los multiplicadores sean actualizados en una proporción comparable, donde F_h es el número de trabajos que utilizan la máquina *h*.

$$\lambda_{hk}^{r,i+1} = \lambda_{hk}^{r,i} + \alpha_h^r \frac{\hat{g}(\lambda_{hk}^{r,i})}{F_h} \quad , \lambda_{hk}^{r,i+1} \geq 0 \tag{6.62}$$

El ciclo principal está compuesto por los siguientes pasos:

- *Inicialización de valores:* al igual que en el método del gradiente surrogado general, se asignan unos valores iniciales a los multiplicadores de Lagrange (λ^r) y se realiza una primera interación siguiendo el método del subgradiente.
- *Paso 1. Subciclo de actualización de los multiplicadores de Lagrange:* se trata de encontrar el valor de las variables del problema (δ_i^r). En cada iteración del subciclo se actualiza el valor de los multiplicadores de Lagrange y se optimiza uno de los subproblemas, manteniendo las soluciones del resto. Los pasos que componen el subciclo se explican después del ciclo principal.
- *Paso 2. Cálculo del tamaño del paso*: se calcula igual que en la primera variante del método mediante la expresión (6.53), donde al haberse completado el ciclo de actualización de multiplicadores de Lagrange, el valor de la función Lagrangiana surrogada se calcula mediante:

$$\hat{L}^{r,n} = \sum_{i=1}^{n} \theta_i(\lambda^{r,i}) - \sum_{h=1}^{M} \sum_{k=1}^{K} \lambda_{hk}^{r,n} M_{hk} \tag{6.63}$$

- *Paso 3. Generación de una solución factible:* este paso es igual al del método del subgradiente (apartado 6.2.1).
- *Paso 4. Criterio de parada:* si el vector de multiplicadores de Lagrange varía muy poco de una iteración a la siguiente el algoritmo se para (6.64). En caso contrario se actualiza el contador del ciclo principal (6.65) y continúa con el paso 1.

$$\|\lambda^{r+1} - \lambda^r\| < \varepsilon_2 \; con \quad \varepsilon_2 \to 0 \tag{6.64}$$

$$\mathrm{r} = \mathrm{r} + 1 \tag{6.65}$$

El subciclo para la actualización de los multiplicadores se realiza en el paso 1 del ciclo principal. Este paso no se completa hasta que se han resuelto todos los subproblemas relacionados con las órdenes de trabajo, utilizando el contador $i_s \in \{1,2,\dots,\mathrm{n}\}$. El subciclo estará compuesto por los siguientes pasos:

- *Paso 1.1. Cálculo del subgradiente surrogado:* una vez calculado el valor de las variables del problema ($\delta^{r}_{\mathrm{ijhk}}$), se calcula el subgradiente surrogado que representa el incumplimiento de las restricciones relajadas mediante la expresión (6.61).
- *Paso 1.2. Actualización de los multiplicadores de Lagrange:* se actualizan los multiplicadores de Lagrange mediante la expresión (6.62), teniendo en cuenta que la dirección de actualización será la del gradiente surrogado de (6.61) y el tamaño del paso es el calculado en el ciclo principal mediante la expresión (6.53). El cálculo del valor de la función Lagrangiana surrogada se realiza mediante la expresión (6.56).
- *Paso 1.3. Resolución del subproblema i_s*: en cada iteración del subciclo se resuelve el problema correspondiente a la orden de trabajo i_s que se define como:

$$\theta_{i_s}\left(\lambda^{r,i_s}\right) = \min_{\delta_{\mathrm{i_s}jhk}} \left(\mathrm{f_{i_s}}\left(\delta_{\mathrm{i_s}jhk}\right) + \sum_{h=1}^{M}\sum_{k=1}^{K}\sum_{j=1}^{O_i} \lambda_{hk}^{r,\mathrm{i_s}}\delta_{\mathrm{i_s}jhk} \right) \tag{6.66}$$

- *Paso 1.4. Criterio de parada del subciclo*: si se han optimizado los subproblemas correspondientes a todas las órdenes, el subciclo se para y se continua el ciclo principal. En caso contrario avanza el contador i_s:

$$\begin{cases} \text{si } \mathrm{i_s} < n & \mathrm{i_s} + 1 \\ \text{si } \mathrm{i_s} = \mathrm{n} & \text{ir al paso 2 del ciclo principal} \end{cases} \tag{6.67}$$

6.3. Selección del programa local por parte de cada orden de trabajo

En el método del subgradiente utilizado para resolver el problema dual Lagrangiano se debe calcular el óptimo del problema relajado en cada iteración, una

vez que se han actualizado los multiplicadores de Lagrange. Ya se vio que, dada la estructura del problema de programación *job shop*, es posible distribuir el problema relajado en tantos subproblemas de optimización como órdenes de trabajo lo componen (apartado 6.1). El objetivo de este apartado es definir un método que resuelva el subproblema de optimización relacionado con cada orden de trabajo.

Cada subproblema está compuesto por la función objetivo, que en este caso es la subfunción lagrangiana relacionada con cada orden $\mathrm{L_i}\left(\lambda_{hk}^{r}, \delta_{\mathrm{ijhk}}^{\mathrm{r}}\right)$ sujeto a las restricciones de la orden (6.68). Recordemos que cada orden *i* está compuesta por $\mathrm{O_i}$ tareas que deben ser realizadas, siguiendo un determinado orden de precedencia. Las tareas puede ser realizadas por una o varias máquinas distintas y tienen una duración determinada (tiempo de proceso). El resultado son los valores δ_{ijhk} que optimizan la subfunción lagrangiana relacionada con cada orden $\mathrm{L_i}$, dados unos determinados valores de los multiplicadores de Lagrange (λ_{hk}^{r}). El problema a resolver por parte de cada orden es:

$$(RL_i)\quad \min_{\delta_{\mathrm{ijhk}}} \mathrm{L_i^r}\left(\lambda_{hk}^{r}, \delta_{\mathrm{ijhk}}^{\mathrm{r}}\right) = \min_{\delta_{\mathrm{ijhk}}} \left(\mathrm{f_i}\left(\delta_{\mathrm{ijhk}}\right) + \sum_{h=1}^{M}\sum_{k=1}^{K}\sum_{j=1}^{O_i} \lambda_{hk}^{r}\delta_{ijhk} \right) \qquad (6.68)$$

sujeto a *las restricciones de la orden i*

Este problema es equivalente al problema de encontrar la ruta con la distancia mínima en un grafo y puede ser resuelto por el método de programación dinámica (Chen et al 1998). La programación dinámica es un método enumerativo en el que se evalúan todas las posibles soluciones, quedándose cada vez con el mejor valor y obteniendo de esta forma el óptimo del problema. El uso de una función recursiva hace que se acote la búsqueda de las posibles soluciones, permitiendo que su cálculo sea factible desde un punto de vista computacional.

Es posible aplicar la técnica de programación dinámica cuando en una secuencia óptima de operaciones cualquier subsecuencia también es óptima. Es lo que se conoce como principio de Bellman (Bellman 1957). Esto permite que la solución sea calculada en varios pasos sucesivos, donde en cada paso se va calculando el óptimo utilizando la solución calculada en el paso anterior. La solución final se obtiene en el último paso.

Para el desarrollo de este método se define la función auxiliar Lagrangiana $\Pi_{ij}(h,k)$ para una tarea O_{ij} que se programa en la máquina h y finaliza en el instante k_f, como:

$$\Pi_{ij}(h,k) = \begin{cases} +\infty & si\ k < \mathrm{k}_{\mathrm{i},j}^{\min} \\ \sum_{k=k_f-d_{ijh}+1}^{k_f} \lambda_{hk} & si\ j \le O_i\ y\ k \ge \mathrm{k}_{\mathrm{i},j}^{\min} \\ f_i\left(T_{k_f}\right) + \sum_{k=k_f-d_{ijh}+1}^{k_f} \lambda_{hk} & si\ j = O_i\ y\ k \ge \mathrm{k}_{\mathrm{i},j}^{\min} \end{cases} \quad (6.69)$$

$$para\ k = 1,2,\dots,K \quad j = 1,2,\dots,O_i \quad h \in F(O_{ij})$$

donde k_{ij}^{min} es el slot de finalización más temprano:

$$\mathrm{k}_{\mathrm{i},j}^{\min} = \mathrm{B_i} + \sum_{\mathrm{j}=1}^{\mathrm{j}} \min_{h \in F(O_{ij})} d_{ijh} - 1 \quad (6.70)$$

Si la tarea no es la última de la orden, el valor de la función auxiliar es la suma de los multiplicadores de Lagrange correspondientes a los intervalos de tiempo donde se programa la tarea. Si la tarea es la última de la orden, se sumará también el valor de la función objetivo para la orden de trabajo. Si la tarea se programa de forma que finalice antes del slot de finalización más temprano k_{ij}^{min} se da a la función auxiliar un valor muy grande.

El problema de optimización de cada orden (6.68) puede reescribirse utilizando la función auxiliar Lagrangiana definida como:

$$\min_{h,k} \sum_{j=1}^{O_i} \Pi_{ij}(h,k) \quad (6.71)$$
$$sujeto\ a\ las\ restricciones\ de\ la\ orden\ i$$

Se define el problema de encontrar en un determinado horizonte de programación K el coste mínimo de completar no más tarde de *k* una operación O_{ij_l} y sus predecesoras (6.72), es decir $\{O_{ij}\}$ con j=1,2.. j_l, de modo que la operación j sea completada no más tarde dentro del periodo de tiempo k ($c_{ij_l} \le k$).

$$JP_{ij_l}(k) \quad tal\ que\ c_{ij_l} \le k \quad (6.72)$$

con parámetros

$\{O_{ij}\}\ \ j = 1,2,\dots,j_l$	tareas precedentes de O_{ij_l} de la orden de trabajo *i*
$0 \le k \le K$	momentos de tiempo
c_{ij_l}	slot de finalización de la tarea O_{ij_l}

Se define la función recursiva $J_{ij}(x)$ que proporciona el valor a optimizar del problema $JP_{ij_l}(k)$, es decir, el valor mínimo de la función auxiliar para una operación O_{ij_l} y de sus predecesoras:

$$J_{i,j}(k) = min\left\{J_{i,j}(k-1), \min_{h\in F(O_{ij})}\{\Pi_{ij}(h,k) + J_{i,j-1}(k-d_{ijh})\}\right\} \quad (6.73)$$

de forma que:

- Dado un determinado *k*, en caso de que no exista solución para un determinado *k* entonces la función $J_{ij}(x)$ tomará el valor $+\infty$.
- Sean $c^*_{ij_l}$ y $h^*_{ij_l}$, respectivamente, el tiempo de finalización de la operación y la máquina utilizada por la operación O_{ij_l} correspondientes a la solución óptima. Existen dos casos posibles:

  - ❖ Si $c^*_{ij_l} < k$ el momento óptimo de finalización de la tarea será anterior a *k* por lo que $J_{ij}(k) = J_{ij}(k-1)$.
  - ❖ Si $c^*_{ij_l} = k$, considerando las relaciones de precedencia, se tiene que para j=1,2,.., $j_l - 1$ se cumple que $c^*_{ij} \leq c^*_{ij_l} - d_{ij_l h^*_{ij_l}} = k - d_{ij_l h^*_{ij_l}}$. La función de coste total será la suma de la función de coste de la tarea O_{ij_l} si se programa de forma que finalice en *k* $[\Pi_{ij_l}(h,k)]$ y la función de coste de las tareas precedentes si finalizan antes del momento de inicio de la tarea O_{ij_l}, por tanto $J_{ij}\left(k - d_{ij_l h^*_{ij_l}}\right)$. Es decir, que

$$J_{i,j}(k) = \min_{h\in F(O_{ij})}\{\Pi_{ij}(h,k) + J_{i,j-1}(k-d_{ijh})\} \quad (6.74)$$

- El valor de $J_{i,j-1}(k-d_{ijh})$ se calcula de manera semenjante, y así sucesivamente hasta llegar a j=1, donde:

$$J_{i,1}(k) = min\left\{J_{i,1}(k-1), \min_{h\in F(O_{i1})}\{\Pi_{ij}(h,k)\}\right\} \quad (6.75)$$

El valor óptimo para la orden de trabajo viene dado por $J_{i,j}(K)$.

El problema (6.71) se resuelve mediante el siguiente algoritmo:

- *Paso 1. Ordenar operaciones*: en primer lugar se ordenan las operaciones que componen la orden i en orden no creciente de número de operaciones posteriores,

obteniendo el conjunto ordenado de tareas de la orden *i* (6.76). Al ordenar las operaciones por el número decreciente de sucesores se asegura que, al calcular el valor de la función para una operación O_{ij}, cualquiera los valores de la función recursiva han sido calculados previamente para todas las operaciones.

$$\{O_{ij}\} \text{ con } j = 1,2,\dots,O_i \tag{6.76}$$

- *Paso 2. Cálculo de la función $J_{ij}(k)$ para cada operación:* para cada una de las operaciones siguiendo el orden de las tareas establecido en el punto anterior. Para cada tarea O_{ij_s} desde $j_s = 1$ hasta O_i.

  - *Paso 2.1. Calcular el tiempo de finalización más temprano:* se calcula el tiempo de finalización más temprano de la operación O_{ij_s} mediante (6.77), donde B_i es el momento de lanzamiento de la orden de trabajo i y d_{ijh} es la duración de la tarea si se realiza en la máquina h.

$$k_{i,j_s}^{min} = B_i + \sum_{j=1}^{j_s} \min_{h \in F(O_{ij})} d_{ijh} - 1 \tag{6.77}$$

  - *Paso 2.2. Calcular el valor de la función $J_{i,j_s}(k))$ en cada instante:* desde $k = k_{i,j_s}^{min}{}_n$ hasta K:

$$\begin{aligned} &J_{i,j_s}(k) = min\left\{J_{i,j_s}(k-1), \min_{h \in F(O_{ij_s})}\left\{\Pi_{ij_s}(h,k) + J_{i,j_s-1}\left(k - d_{ij_s h}\right)\right\}\right\} \\ &\text{con } J_{i,j_s}\left(k_{i,j_s}^{min} - 1\right) = +\infty \\ &k = k + 1 \end{aligned} \tag{6.78}$$

  - *Paso 2.3. Actualizar el contador de tareas:* $j_s = j_s + 1$

- *Paso 3. Definición del programa óptimo*: el programa óptimo está definido en cada tarea por el *slot* más temprano con el valor mínimo de la función auxiliar $J_{i,j}(k)$ que cumpla con las relaciones de precedencia, empezando desde la última operación y calculando sucesivamente las tareas precedentes.

6.4. Métodos de construcción de programas factibles

En cualquiera de las versiones del método de Relajación Lagrangiana, para la resolución de un problema de optimización entero (P), es necesario construir una solución que cumpla con las restricciones del problema original (P).

Para el problema *job shop*, la solución es un programa de producción que cumpla las condiciones definidas en el problema. La mayoría de los métodos de optimización utilizan un método constructivo de generación de programas de producción, puesto que en algún paso es necesario generar un programa que defina una secuencia de operaciones en cada uno de los recursos disponibles, así como los momentos de inicio de estas.

En un método constructivo, las tareas, durante el proceso de generación de programas, van pasando por cada uno de los siguientes estados: no programables (N) cuando las tareas inmediatamente precedentes no han sido programadas todavía; elegibles (E), en el caso de que sus tareas precedentes han sido ya programadas; y programadas (P). Al final del proceso todas las tareas deben ser programadas.

La mayor parte de los algoritmos de generación de programas de producción (factibles) se componen de dos procesos principales. El primero es la selección de operaciones candidatas a ser programadas. El segundo es la selección de la tarea a programar (Companys 1989).

El proceso por el que las tareas se seleccionan y se les asignan unos tiempos de realización en un determinado recurso da origen a los distintos métodos de construcción de programas. En este proceso hay que tener en cuenta que, dada una determinada secuencia de operaciones asignadas a una o varias máquinas, el número de programas distintos es infinito si se tiene en cuenta la temporización -determinación de los momentos de inicio y de fin- de cada tarea.

En lo que sigue se describen los principales tipos de programas y las condiciones para que un programa de producción sea válido, así como los principales métodos para generarlos.

6.4.1. Tipos de programas

Dado un conjunto de tareas a realizar y un conjunto de recursos disponibles, programar es asignar recursos a las tareas, secuenciar las tareas en cada recurso y definir la temporización de las mismas, de forma que se cumplan las restricciones del problema. Se dice que un programa de producción es *factible* cuando cumple con las restricciones. Será *no factible* si no cumple con al menos una de ellas.

Programar no es solo secuenciar una serie de tareas en una máquina, sino también realizar la temporización de las mismas, determinando el tiempo de comienzo y de fin de cada una de estas operaciones. De hecho, a cada secuencia posible de operaciones en una máquina se le pueden asociar infinitas temporizaciones (calendarios o programas), pero sólo una de ellas corresponde a un programa válido.

Para una determinada secuencia de operaciones, un *desplazamiento limitado a la izquierda* consiste en adelantar el comienzo (y fin) de una operación, respetando las relaciones de precedencia y sin modificar el orden de la secuencia en la misma.

Dada una secuencia de operaciones, un *programa semiactivo* es aquel en el que no se pueden realizar desplazamientos limitados a la izquierda (Figura 24). Dicho de otro modo, es aquel programa que no posee tiempos muertos innecesarios entre todos los posibles programas definidos por la secuencia de operaciones.

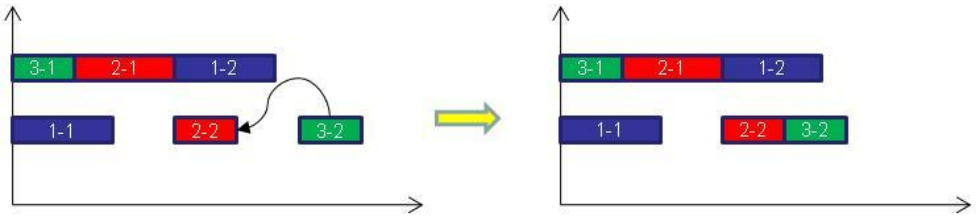

Figura 24. Transformación de un programa en semiactivo mediante un desplazamiento limitado a la izquierda

Si además se permite modificar la secuencia de operaciones en la máquina, se define un *desplazamiento a la izquierda en sentido amplio (no limitado)* que permite saltar una operación en la misma máquina, siempre que antes quede un hueco de tiempo muerto suficiente para la realización de la operación que se desliza, y se sigan respetando las relaciones de precedencia.

Dada una secuencia de operaciones, un *programa activo* es aquel en el que no se pueden realizar desplazamientos a la izquierda en sentido amplio (Figura 25).

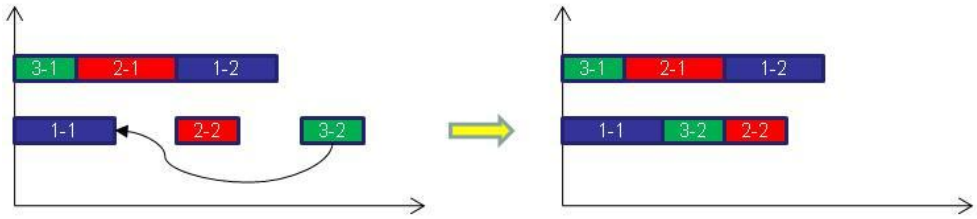

Figura 25. Transformación de un programa en activo mediante un desplazamiento no limitado a la izquierda

Como principales propiedades podemos destacar que:

- Un programa activo es semiactivo, pero no a la inversa.
- Todo programa no semiactivo es dominado (tiene un valor igual o peor de la función objetivo) por uno activo.
- Si la medida de eficacia es regular, el conjunto de programas activos contiene algún programa óptimo, ya que siempre existirá un programa óptimo activo de calidad igual o superior a un programa cualquiera, porque la trasformación a programa activo no puede empeorar la medida de eficiencia.

Un *programa sin retrasos* es aquel en el que ninguna máquina espera sin trabajar en ninguna operación si existe una que podría realizarla (Figura 26). Un programa sin retrasos es un programa activo, pero no todo programa activo es sin retrasos ya que pueden existir programas activos con retrasos. En ocasiones resulta útil tener en espera alguna operación si esto produce una mejora en la medida de eficacia, al permitir incluir otras posibles operaciones.

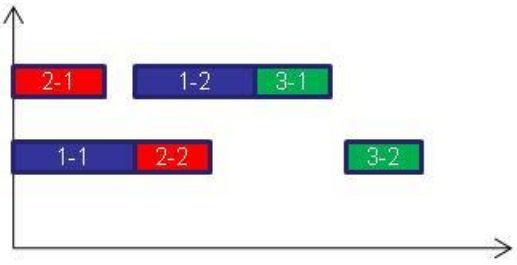

Figura 26. Programa activo con retrasos

6.4.2. Algoritmo de Giffler y Thompson

Este algoritmo fue desarrollado por Giffler y Thompson (1960) para la generación de programas de producción. Tiene como característica principal que genera programas activos. Esto es una ventaja sobre otros métodos, al reducir el campo de búsqueda.

En este método se evalúa el tiempo de finalización más temprano entre todas las tareas que cumplen las condiciones para ser programadas -por sus precedencias y fecha de lanzamiento de la orden de trabajo-. A continuación, se escoge la máquina que puede quedar libre más temprano y se genera el conjunto de tareas candidatas a ser programadas, compuesto por aquellas tareas que pueden ser programadas en dicha máquina y además su tiempo de comienzo más temprano es inferior al momento de finalización de la tarea elegida en un primer momento.

Para explicar el método (Figura 27), se realiza una serie de definiciones:

E(h)	Conjunto de operaciones elegibles, es decir, operaciones que se realizan en la máquina h que están disponibles para ser programadas.
FFS(h)	Es el *slot* de tiempo más temprano en que puede quedar libre la máquina h si se programa en ella cualquiera de las tareas elegibles.
$k_{i,j}^{min}$	Momento de comienzo más temprano de la tarea O_{ij} teniendo en cuenta sus relaciones de de precedencia.
d_{ijh}	Duración de la tarea O_{ij} en la máquina h.

El algoritmo se desarrolla en dos fases: primero se selecciona el recurso a programar y, una vez elegido, se selecciona la área a programar:

- *Paso 1. Selección de la máquina a programar:* se calcula el instante más próximo en que puede quedar libre cada máquina *FFS(h)* si se programan las tareas elegibles (6.79). Posteriormente se elige el recurso h_s con menor *FFS(h)*.

$$\begin{aligned} FFS(h) &= min\{k_{i,j}^{min} + d_{ijh}\} \mid O_{ij} \in E(h) \quad si\ E(h) \neq \emptyset \\ FFS(h) &= \infty \qquad\qquad\qquad\qquad\qquad\qquad\quad si\ E(h) = \emptyset \end{aligned} \tag{6.79}$$

- *Paso 2. Selección de la tarea:* se seleccionan las tareas candidatas a ser programadas, y después se elige una entre estas:

  - *Selección del conjunto de candidatas:* para elegir la operación, consideraremos un conjunto de candidatos reducido. Las operaciones a considerar son aquellas pertenecientes al conjunto de tareas pendientes de programar cuya fecha de

posible comienzo, $k_{i,j}^{min}$, es menor que el instante más próximo en el que puede quedar libre la máquina, *FFS(h_s)*.

$$E'(h_s) = O_{ij} \in E(h_s) \mid rp(O_{ij}) < min\{FFS(h_s)\} \quad (6.80)$$

- *Elección de la tarea:* si existe una sola operación candidata a programar será la elegida. En el caso de que existan varias tareas a programar, se seleccionará la tarea mediante alguna regla de decisión. En nuestro caso, la regla de decisión estará relacionada con el programa lagrangiano no factible -tiempo de inicio o de finalización del problema relajado-, complementado con alguna otra regla en caso de empate.

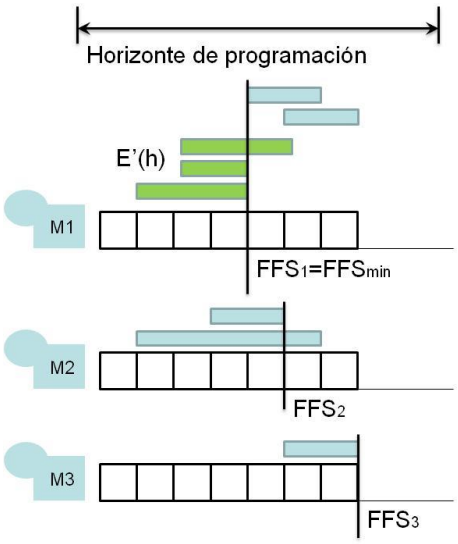

Figura 27. Método de programación de Giffler-Thompson

6.4.3. Algoritmo de lanzamiento de tareas

La característica de este método es que la tarea a programar se elige entre todas las tareas elegibles en un momento determinado con el tiempo de comienzo o de finalización más temprano. Este método produce programas sin retraso.

El algoritmo es el siguiente:

- *Paso 1. Selección de candidatas:* de entre todas las tareas elegibles, se eligen aquellas con el tiempo de comienzo más temprano.

- *Paso 2. Elección de la tarea:* la tarea a programar se selecciona entre todas las tareas candidatas mediante un criterio determinado (reglas de prioridad). Uno de los criterios es programar todas las tareas por su tiempo de finalización, primero la tarea cuyo tiempo de finalización sea menor. En caso de empate, elegir aquella que tenga un tiempo de comienzo menor. En caso de empate en las dos anteriores, complementarla con alguna otra regla de selección. Un criterio alternativo es coger como criterio primario el momento de comienzo y como criterio secundario el momento de finalización de la tarea.

6.4.4. Algoritmo voraz en función de la penalización marginal por retraso

Este algoritmo ha sido definido por Hoitomt et al (1993). El algoritmo parte de la lista de tareas ordenadas por el tiempo de comienzo definido en el programa relajado. En primer lugar intenta programar la tarea en los recursos disponibles antes o en la fecha de comienzo inicialmente programada (b). En caso de que no sea posible, busca otra tarea que tenga el mismo tiempo de comienzo que la primera (b). Si no la encuentra, modifica las fechas de comienzo de las tareas que no han encontrado recursos libres y las retrasa una unidad de tiempo, para reordenar la lista de tareas y repetir el proceso.

Se define la función penalización marginal $Pm(O_{ij})$ para el trabajo i como el incremento del coste cuando la tarea se retrasa una unidad de tiempo:

$$Pm(O_{ij}) = \begin{cases} f_i(T_i + 1) - f_i(T_i) & Si\ \Delta T_i > 0\ cuando\ O_{ij}\ se\ retrasa \\ 0 & en\ caso\ contrario \end{cases} \quad (6.81)$$

Inicialmente se construye una secuencia U a partir de la información de la solución del problema relajado, donde las tareas estén ordenadas por el instante de inicio y, en caso de empate, por la penalización marginal por retraso. Aparecerá antes en la lista aquella tarea que tenga mayor penalización marginal. Es decir, una secuencia en la que se cumpla que si la operación O_{ij} se encuentra antes que la operación O_{uv}, se verifica:

$$\begin{aligned} & b_{ij} \leq b_{uv} \\ & o\ si \quad b_{ij} = b_{uv} \ \ entonces\ Pm(O_{ij}) \geq Pm(O_{uv}) \end{aligned} \quad (6.82)$$

Se definen una serie de índices y variables requeridos por el pseudocódigo.

$F(O_{ij})$	Conjunto ordenado de máquinas que puede realizar la operación (O_{ij}).
t	Índice del conjunto de máquinas.
h_t	-ésimo elemento del conjunto de máquinas.
k	Índice del horizonte de programación.
m_{ij}	Ruta óptima del programa inicial.
temp	Variable temporal que almacena la ruta óptima.
U	Secuencia de tareas programables.
E	Conjunto de operaciones no programada que no pueden ser programadas entre el momento n y su propio momento de comienzo bij.
Md_h	Capacidad disponible de la máquina h.

El algoritmo voraz se define a continuación:

- *Paso 0. Inicialización:* se parte de la lista U que contiene la secuencia inicial de tareas y del conjunto de tareas E, que inicialmente estará vacío E = Ø. Sigue en el Paso 1.
- *Paso 1. Coger la primera operación de la lista U:* se coge la primera operación de la lista U. El tiempo de comienzo y la máquina asignada serán los del programa original. Hacer m:=m_{ij} , b:=b_{ij}, temp:= m y r:=1, ir al Paso 2.
- *Paso 2. Determinar el momento disponible más temprano de la máquina:* se determina el momento más temprano *l* para el que la capacidad disponible de la máquina es mayor que cero, es decir, que $Md_{hk} > 0$. Hacer k:= *l* e ir al Paso 3.
- *Paso 3. Comprobar la disponibilidad de la máquina:* se comprueba si la máquina puede realizar la operación completa de forma continua a partir del instante *l*, es decir, si $Md_{hk} > 0$ para k= *l*, *l* +1,…, *l* +d_{ijh}-1, ir al Paso 4. En otro caso, ir al Paso 5.
- *Paso 4. Modificar la disponibilidad de máquina:* si se cumplen las restricciones de precedencia vinculadas a la operación, programar la operación O_{ij} para que empiece en el momento *l*, programar la operación y actualizar la capacidad de la máquina utilizada. Es decir, $Md_{hk} = Md_{hk} - 1$ para k= *l*, *l* +1,…, *l* +d_{ijh}-1, e ir al Paso 9. En otro caso, ir al Paso 5.
- *Paso 5. Avanzar una unidad de tiempo k*: se avanza una unidad de tiempo en el horizonte de programación (k:=k+1). Si k es posterior a la fecha de inicio de la tarea, k>b, ir al Paso 6. En otro caso, volver al Paso 3.

- *Paso 6. Seleccionar otra máquina:* si $h_t = temp$, entonces hacer t:=t+1 . Si ya se han comprobado todas las máquinas $t > |F(O_{ij})|$, hacer $h_{ij} \coloneqq temp$ e ir al Paso 7. En otro caso, hacer h= h_t, t=t+1, e ir al Paso 2.
- *Paso 7. Coger la siguiente operación en la lista:* si la segunda operación en la lista U tiene un tiempo de comienzo b, se elimina la primera operación de la lista U y se reordena, y por último se incluye en la lista E. U:=U-{(O_{ij}} y $E \coloneqq E \cup \{O_{ij}\}$ e ir al Paso 1. En otro caso, ir al Paso 8.
- *Paso 8. Actualizar la secuencia U:* para cualquier operación no programada $O_{ij} \in E$ tal que su tiempo de comienzo sea menor o igual que b $(b_{ij} \leq b)$ se modifica el tiempo de comienzo de la operación $b_{ij} \coloneqq b + 1$; además se revisan todas las operaciones siguientes del trabajo *i* y se modifican los tiempos de inicio que violen las relaciones de precedencia. Las tareas del conjunto E pasan al conjunto U y se reordena. Hacer $U \coloneqq U \cup$E y reordenarlo. El conjunto E quedará vacío (E=Ø) e ir al Paso 1.
- *Paso 9. Criterio de parada:* se elimina la tarea O_{ij} de la lista U y se comprueba si quedan operaciones por programar. Set U=U-{O_{ij}}. Si U=Ø parar. En otro caso, ir al Paso 1.

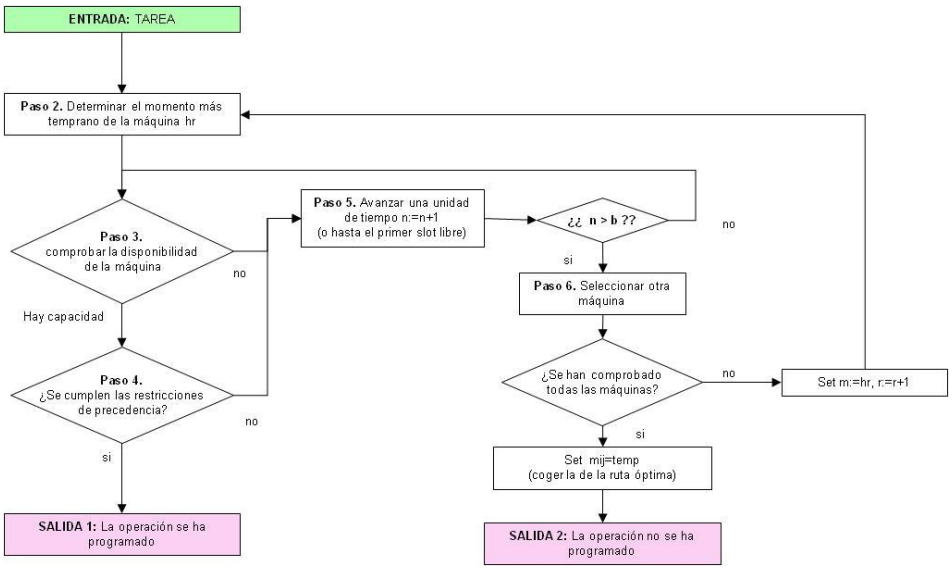

Figura 28. Función de búsqueda de recursos libres por parte de la tarea a partir del utilizado en Hoitomt et al (1993)

En el algoritmo podemos agrupar los pasos que hay entre el 2 y el 6, y representarlos como una función de búsqueda de recursos libres por parte de la tarea (Figura 28). Eso permite simplificar la representación el algoritmo (Figura 29).

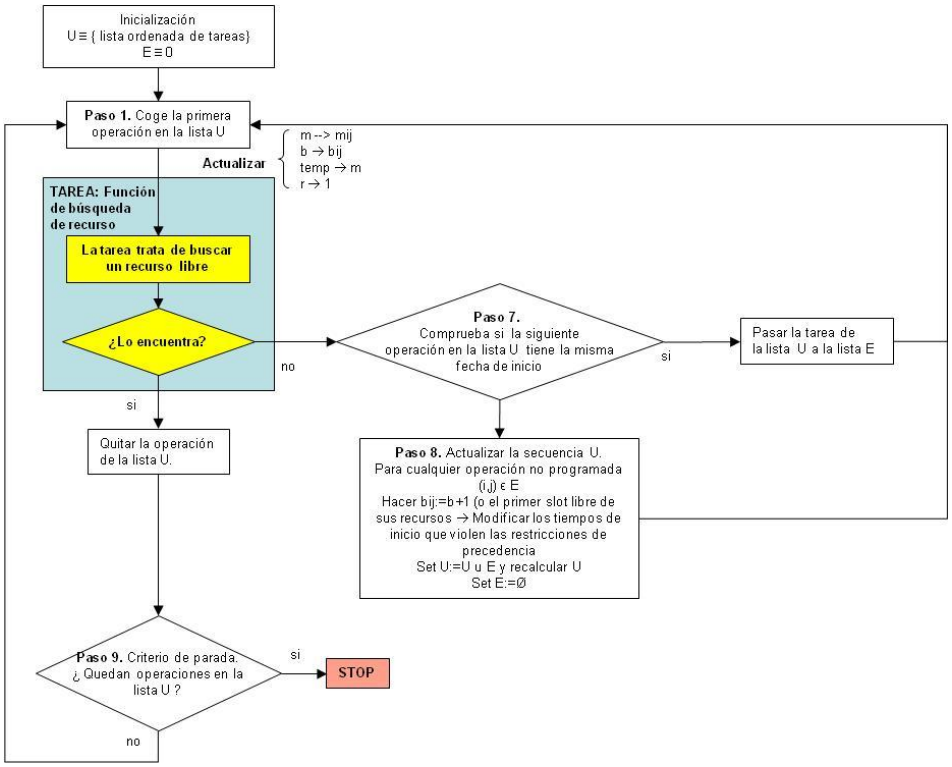

Figura 29. Algoritmo simplificado de construcción de programas factibles a partir del utilizado en Hoitomt et al (1993)

6.4.5. Selección de tareas

Un paso clave en los métodos descritos en el apartado anterior es la selección de la tarea a programar. El proceso comienza con el establecimiento previo de las tareas candidatas a ser programadas. Este proceso es característico de cada método, aunque en todos ellos se cumple la condición de que para que una tarea sea candidata a ser programada, sus tareas precedentes han debido ser programadas previamente. La mayor parte de los trabajos aplican diferentes reglas de prioridad para elegir una entre las tareas candidatas a ser programadas. No es posible considerar que ninguna de las reglas funciona mejor que el resto, ya que su rendimiento depende en

gran medida de la estructura de la instancia del problema a resolver. En el capítulo 2 se citan distintos trabajos que estudian el funcionamiento de estas reglas.

En los trabajos estudiados, para seleccionar la tarea a programar se utilizan varios criterios ordenados de forma jerárquica, de modo que primero se aplica un criterio, el principal, y en caso de empate entre varias tareas, se utiliza el criterio secundario. Muchos de los trabajos de construcción de programas factibles a partir de la solución del problema de relajación Lagrangiana siguen los desarrollos de Hoitomt et al (1993), que utiliza la fecha de comienzo de la tarea de la solución del problema relajado como criterio principal, y la penalización incremental al retrasarse una unidad de tiempo como criterio secundario. La Tabla 6.1 recoge los criterios utilizados los trabajos que lo indican de modo explícito.

	Criterio principal	
Criterio secundario	**fecha de comienzo**	**fecha de fin**
incremento de la función de penalización	Hoitomt et al (1993), Czerwinski y Luh (1994), Kaskavelis y Caramanis (1994), Chen et al (1995, 1998), Wang et al (1997), Gou et al (1998), Zhang et al (2001), Luh y Feng (2003), Ni et al (2008)	Chen y Luh (2003)
ratio critico	Fang et al (2000), Sun et al (2006)	
duración de la tarea		Masin et al (2007)
no especificado	Dewan y Joshi (2002), Jeong y Yim (2009)	Tang et al (2006), Baptiste et al (2008)

Tabla 6.1. Aplicación de los criterios principal y secundario para la construcción de programas factibles

Definiremos, a continuación, algunos criterios que incorporan información del programa. Estos son la función de penalización marginal, el ratio crítico, el tiempo de finalización deseado y el coste de retraso aparente.

- Función de penalización marginal

El criterio de función de penalización marginal utiliza como índice el aumento de la función de penalización de la orden de la tarea cuando la tarea se retrasa una unidad de tiempo.

$$\begin{aligned} Pm(O_{ij}) &= 0 && si\ la\ tarea\ tiene\ holgura \\ Pm(O_{ij}) &= f_i(t+1) - f_i(t) && si\ produce\ un\ retraso\ en\ la\ orden \end{aligned} \tag{6.83}$$

- Ratio crítico

El ratio crítico es uno de los índices más utilizado en programación de tareas. Este índice se calcula como el cociente entre el tiempo que queda para la fecha de entrega comprometida (D_i) y el tiempo de trabajo restante para completar el trabajo:

$$RC_{ij} = \frac{D_i - k}{\sum_{q=j}^{O_i} d_{ij}} \tag{6.84}$$

- Tiempo de finalización deseado

El tiempo de finalización deseado de la tarea se calcula teniendo en cuenta el (S_{ij}^{+}) o tiempo de holgura que tiene la tarea. Lo que hace este índice es repartir la holgura total de la tarea entre ésta y todas las actividades restantes para completar la orden de trabajo.

$$TFD_{ij} = \frac{S_{ij}^{+}}{O_i - j} \tag{6.85}$$

donde:

TSL es el tiempo de comienzo en el programa propuesto por la orden.

$S_{ij}^{+} = D_i - k - \sum_{q=j}^{O_i} d_{iqh}^{LR\ q}$ es el slack u holgura.

$(O_i - j)$ es el número de tareas restantes para completar la orden.

- Coste de retraso aparente (apparent tardiness cost atc)

Este criterio ha sido definido en Vepsalainen y Morton (1987), y destaca por su buen comportamiento en los problemas de minimización del retraso ponderado. Evalúa el slack (Sij+) o tiempo de holgura que tiene la tarea. Si el slack es cero el criterio se comporta como el de mínima duración de la tarea (SPT). Si el slack es positivo entonces el índice se ve afectado por un factor de urgencia (U_{ij}), que es una función exponencial negativa del slack, de modo que cuanto mayor es el slack de la tarea la prioridad de ésta va disminuyendo de modo exponencial. En la fórmula original se utilizan tiempos estimados de comienzo de las tareas futuras, que nosotros

sustituimos por la información que proporcionan los programas propuestos por las órdenes de trabajo:

$$ATC_{ij} = \frac{w_i}{p_{ij}} \times e^{-(\frac{S_{ij}^{+}}{Kd_{avg}})} \tag{6.86}$$

donde:

$S_{ij}^{+} = D_i - k - \sum_{q=j}^{O_i} d_{iqh}^{LR\ q}$ es el slack u holgura.

w_i es el peso de la órden

d_{ijh} duración de la tarea en la máquina *h*

d_{avg} duración media de las tareas en el recurso

$K \in \{2,3\}$

6.5. Relación del método de Relajación Lagrangiana con las subastas combinatorias

A lo largo de este capítulo se ha visto la aplicación del método de Relajación Lagrangiana para la resolución del problema *job shop* (apartado 1.4.3), donde el horizonte de programación se divide en K intervalos de tiempo o *slots*. La solución es un programa de producción definido mediante las variables binarias $\{\delta_{ijhk}\}$. El valor 1 representa que la tarea j de la orden de trabajo i está programada en el instante k de la máquina h y el valor 0 representa que la tarea no está programada en dicho *slot*. Por las restricciones de continuidad del problema, cada tarea necesitará ser programada en un número de *slot* consecutivos igual a su duración en la máquina (d_{ijh}).

El método de Relajación Lagrangiana puede entenderse como una subasta en la que las órdenes de trabajo pujan por conseguir los *slot* de tiempo que necesitan del horizonte de programación de cada máquina. Los ítems intercambiados en la subasta son estos *slot* de tiempo, definidos por la máquina a la que pertenece *h* y el momento en el horizonte de programación que representan *k*. Por eso, cada *slot* se denomina con los índices *hk*. En concreto, el algoritmo para la resolución del problema lagrangiano puede interpretarse como la resolución de una subasta combinatoria iterativa con fijación de precios. Para verlo, se puede comparar el algoritmo definido en el apartado

4.4.2, donde se muestra el protocolo de resolución del problema *job shop* mediante una subasta combinatoria de ese tipo, con el algoritmo de resolución del Problema Lagrangiano (Tabla 6.2 y Figura 30). La utilización del método del subgradiente de resolución del problema dual de la Relajación Lagrangiana, como método de actualización de precios en las subastas combinatorias, proporciona un entorno escalable, distribuido y con un comportamiento predecible (Masin et al 2007).

Protocolo de subasta combinatoria (apartado 4.4.2)	**Algoritmo para la resolución del problema dual lagrangiano (apartado 6.2)**
Paso 0. Se parte de unos precios iniciales.	Paso 0. Inicialización de valores.
Paso 1. Cálculo de la puja a enviar por parte de cada trabajo, optimizando su función de beneficio.	Paso 1. Resolución de cada uno de los problemas que resulta de descomponer la Relajación Lagrangiana del problema $\theta_i^r(\lambda^r)$.
Paso 2. Evaluación de las pujas enviadas. Comparación de la demanda de cada *slot* de tiempo con la capacidad disponible.	Paso 2. Cálculo del subgradiente.
Paso 3. Construcción del programa factible.	Paso 3. Generación de una solución factible.
Paso 4. Criterio de parada.	Paso 4. Criterio de parada.
Paso 5. Actualización de los precios en función del exceso de demanda.	Paso 5. Mejora del valor de la función dual. Actualización de los multiplicadores.

Tabla 6.2. Relación entre el protocolo de subasta combinatoria y el algoritmo de Relajación Lagrangiana

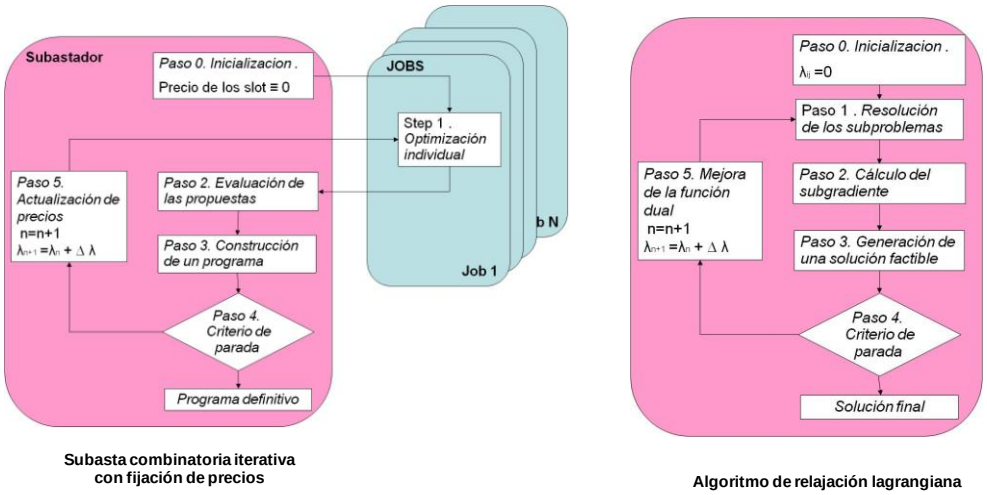

Figura 30. Comparación entre el protocolo de subasta combinatoria y el algoritmo de Relajación Lagrangiana

En el problema *job shop* cada *slot* de tiempo *hk* dispone de una determinada capacidad que limita el número de tareas que pueden ser programadas en él. Estas restricciones son las que se relajan al aplicar el método de Relajación Lagrangiana. A cada restricción de capacidad del *slot* de tiempo *hk* se asocian unos valores λ_{hk} llamados multiplicadores de Lagrange que penalizan el incumplimiento de estas restricciones. Este valor puede ser interpretado como el precio por utilizar el *slot hk*.

Volviendo a la subasta, cada orden de trabajo tiene que tratar de minimizar una función de coste suma de la función objetivo que penaliza el retraso de la orden de trabajo ($f_i\left(T_i(\delta_{ijhk})\right)$ y el coste de uso de los *slot* en las máquinas correspondientes para poder realizar las tareas de la orden de trabajo ($\sum_{j=1}^{O_i}\sum_{h=1}^{M}\sum_{k=1}^{\mathrm{K}}\lambda_{hk}\delta_{ijhk}$). El valor de esta función es equivalente a la función Lagrangiana asociada a cada orden de trabajo *i*:

$$\min_{\delta_{\mathrm{i}}}\left(\mathrm{f_i}(\delta_i)+\sum_{j=1}^{O_i}\sum_{h=1}^{M}\sum_{k=1}^{\mathrm{K}}\lambda_{hk}\delta_{ijhk}\right)=\min_{\delta_{\mathrm{i}}}\mathrm{L_i}(\lambda,\delta_{\mathrm{i}}) \tag{6.87}$$

En el método de Relajación Lagrangiana este paso se corresponde con la resolución del problema relajado. El resultado de este proceso será el programa propuesto por la orden de trabajo *i* ($\{\delta_{ijhk}^{RL}\}$) que se enviará al subastador.

En la subasta, el subastador evalúa todas las pujas recibidas y modifica el precio en función de la relación oferta-demanda. Si la demanda de un *slot* es superior a la oferta, el precio del *slot* aumenta. Si la oferta es superior a la demanda, el precio del *slot* disminuye. En el capítulo 4 se vio como el precio se podía actualizar mediante:

$$\lambda_{hk}^{r+1}=\max\{0,\lambda_{\mathrm{hk}}^{r}+f(D_{\mathrm{hk}}^{r})\} \tag{6.88}$$

donde D_{hk}^{r} es el exceso de oferta ($D_{\mathrm{hk}}^{r}<0$) o demanda ($D_{\mathrm{hk}}^{r}>0$) de cada *slot k* de la máquina *h*:

$$D_{\mathrm{hk}}^{r}=\sum_{\mathrm{i}=1}^{\mathrm{N}}\sum_{\mathrm{j}=1}^{\mathrm{O_i}}\delta_{\mathrm{ijhk}}^{\mathrm{RL}}-\mathrm{M_{hk}} \tag{6.89}$$

La actualización de los multiplicadores de Lagrange según el método del subgradiente (6.29) hace lo mismo: aumenta el valor del multiplicador cuando hay más demanda ($\sum_{\mathrm{i}=1}^{\mathrm{N}}\sum_{\mathrm{j}=1}^{\mathrm{O_i}}\delta_{\mathrm{ijhk}}^{\mathrm{RL}}$) que oferta (capacidad de la máquina en el *slot* ($\mathrm{M_{hk}}$)) y disminuye cuando ocurre lo contrario. La expresión del cálculo del subgradiente (6.22) es idéntica a la expresión (6.89) , por lo que ambos procesos son equivalentes.

Este proceso se repetirá hasta que se cumplan las condiciones de parada. Desde el punto de vista de la subasta, cuando el subastador compruebe que se cumplen estos criterios realizará la asignación definitiva y dará por finalizada la subasta. Este proceso se ha explicado sobre la base del subgradiente. Sin embargo, también es posible utilizar otras métodos de resolución utilizados en el método de Relajación Lagrangiana como método de actualización de precios (Attanasio et al 2006).

El método del gradiente intercalado también puede interpretarse como una subasta, de forma similar a como se ha hecho con el método del subgradiente. En la versión de Zhao et al (1997) los precios se envían de forma secuencial a la orden i correspondiente. Esta orden *i* devuelve el programa óptimo local, que sirve para actualizar el tamaño del paso a partir de los precios y los programas calculados, de forma los nuevos precios calculados se enviarían a una nueva orden. En la versión de Kaskavelis y Caramanis (1998) cada orden *i* envía de forma secuencial el programa local y el valor de la subfunción Lagrangiana calculada con los precios utilizados. Los precios se actualizan y se envían a una nueva orden que repite el proceso. El tamaño del paso se actualiza cuando todas las órdenes han enviado el programa local y el valor de la subfunción Lagrangiana con la que se calcula el valor de la función Lagrangiana para poder calcular el nuevo tamaño de paso del ciclo.

6.6. Conclusiones del capítulo

Partiendo de la definición del problema *job shop* como un problema de optimización en variables enteras, la técnica de Relajación Lagrangiana permite eliminar las restricciones complejas, en este caso las restricciones de capacidad de las máquinas, simplificando notablemente la resolución del problema. El grupo de restricciones eliminadas se incorpora a la función objetivo con un multiplicador, de forma que las restricciones que no se cumplan –porque hay más demanda de un *slot* temporal en una máquina que su capacidad- son penalizadas. La nueva función objetivo es la llamada función lagrangiana. Para obtener los multiplicadores más adecuados, el método del subgradiente penaliza el incumplimiento de las restricciones relajadas, aumentando el valor de los multiplicadores en función de su grado de incumplimiento.

La aplicación del método de Relajación Lagrangiana a problemas separables de programación entera, como es el caso del problema *job shop*, permite la distribución de los cálculos necesarios para su resolución. Esto lo hace muy atractivo para su

utilización como paradigma de distribución en sistemas multiagente. Los cálculos serán repartidos entre distintos agentes que representan órdenes de trabajo.

Se han revisado los principales métodos de actualización de los multiplicadores de Lagrange para la resolución del problema dual: el método del subgradiente es el más utilizado para resolver el problema de maximización de la función dual. La dirección del subgradiente se obtiene después de resolver todos los subproblemas en los que se divide el problema relajado. Los multiplicadores se actualizan utilizando este subgradiente. El cálculo de todos los subproblemas puede ser muy costoso computacionalmente en caso de problemas de tamaño grande, por lo que sería deseable poder obtener una dirección para poder actualizar los multiplicadores sin necesidad de resolver todos los subproblemas. El método del gradiente conjugado utiliza la dirección de actualización de los multiplicadores en iteraciones anteriores para calcular la nueva dirección de actualización. Con ello se busca reducir las oscilaciones que caracterizan el método del subgradiente.

En el método del subgradiente surrogado no es necesario que se optimicen todos los subproblemas para encontrar una dirección adecuada, sólo se necesita mejorar la solución de la iteración anterior dados los nuevos multiplicadores. Esta condición asegura la convergencia del método hacia el óptimo. Es posible conseguir que se cumpla de diversas maneras, lo que convierte a este método en generador de otros. De hecho, el método del subgradiente intercalado es uno de ellos. Este método se caracteriza porque sólo necesita que se resuelva un problema por iteración para obtener la dirección de actualización de los multiplicadores de Lagrange.

Los algoritmos que se utilizan en todos estos métodos siguen un esquema básico común. Las variaciones en los métodos provienen de:

- La forma de resolver los problemas locales. Las opciones encontradas son:

  (1) Se optimizan todos los subproblemas.
  (2) Se optimiza uno de los subproblemas.
  (3) Se mejora el valor de la función lagrangiana.

- La dirección de actualización de los multiplicadores de Lagrange:

  (1) No se tiene en cuenta la dirección de actualización de las iteraciones anteriores.
  (2) Se tiene en cuenta la dirección de actualización de las iteraciones anteriores (subgradiente conjugado).

Además, existen algunas variaciones en cuanto a la forma de calcular el peso de las direcciones anteriores al calcular la dirección de actualización en el método del subgradiente conjugado, en el cálculo del tamaño del paso y en el método para actualizar los multiplicadores. Creemos que es interesante poder establecer qué alternativas funcionan mejor.

Asimismo, se ha descrito la implementación de los pasos que completan la resolución del problema dual en el método de Relajación Lagrangiana, como son la resolución de los subproblemas asociados y los métodos de construcción de programas factibles. Para el problema *job shop*, el método utilizado para la resolución de los subproblemas es el método de programación dinámica. Este método es muy dependiente del tamaño del horizonte de programación, por lo que es importante ajustarlo al máximo para reducir el tiempo de cálculo. En cuanto a los métodos de construcción de programas, hemos estudiado tres métodos con distintas variaciones en función de los criterios utilizados. Consideramos interesante determinar cuál de estos métodos tiene un mejor comportamiento en este tipo de problemas

Por último, se ha establecido la relación entre el método de Relajación Lagrangiana y el método de resolución basado en subastas. La utilización del método del subgradiente de resolución del problema dual de la Relajación Lagrangiana como método de actualización de precios en las subastas combinatorias, proporciona un entorno escalable y distribuido con un comportamiento predecible. También es posible utilizar otros métodos de resolución utilizados en el método de Relajación Lagrangiana como método de actualización de precios en las subastas.

En el próximo capítulo estudiaremos distintas alternativas tanto de los métodos como de los parámetros utilizados en el método de Relajación Lagrangiana, con el objetivo de ser implementados en sistemas distribuidos de programación de tareas. Se plantearán aspectos relacionados con la resolución del problema dual, o del proceso de actualización de precios cuando la programación de tareas se modela bajo el paradigma de subastas. Se revisarán distintos métodos y variantes que comparar, atendiendo a su capacidad de utilización desde un punto de vista distribuido, de convergencia y calidad de la solución obtenida. También se estudiarán, bajo estos mismos criterios, los métodos de construcción de programas factibles que transforman los programas relajados.

Capítulo 7. Experimentación y análisis de resultados

En los capítulos anteriores se ha presentado un método de resolución distribuido basado en la Relajación Lagrangiana para la resolución del problema de programación de talleres flexibles, además de presentar las distintas alternativas en cada paso. En el presente capítulo se muestran los experimentos realizados donde se analizan algunas de las variantes más relevantes respecto al método base (punto 6.2.1).

En primer lugar se estudiarán los aspectos relacionados con la actualización de multiplicadores (precios desde el punto de vista de subastas). Éstos están relacionados con la actualización del mejor valor del problema relajado, es decir, con el cálculo de la cota inferior (Figura 31). También se compara el método de actualización de precios walrasiano no adaptativo con el método del subgradiente como alternativa, y se incidirá sobre los aspectos de sincronismo del método. En este primer bloque se comparan los principales métodos para la actualización de precios y se estudian los parámetros de funcionamiento de los métodos como la frecuencia de actualización del factor de corrector del tamaño del paso, y el peso de la información de iteraciones anteriores en el proceso de actualización de precios.

En segundo lugar, también se estudian los aspectos relacionados con la elaboración del programa factible a partir de la información proporcionada por el problema

relajado (Figura 32). Se comparan distintos métodos de construcción de programas factibles, así como las reglas de prioridad utilizadas.

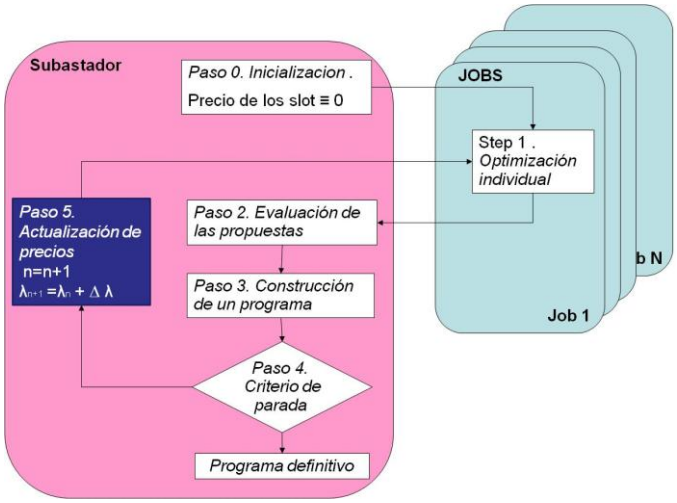

Figura 31. Subasta combinatoria iterativa con fijación de precios. Estudio de la actualización de precios.

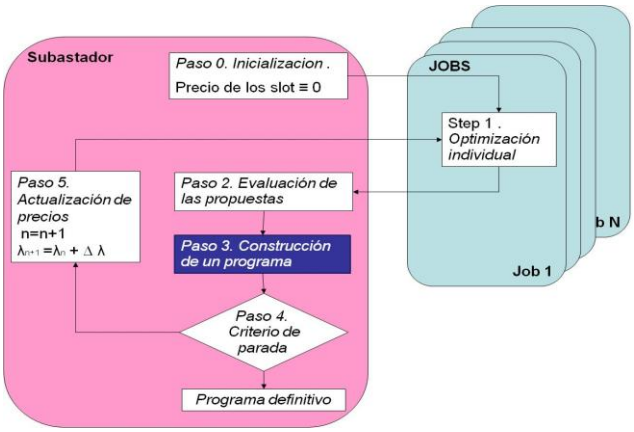

Figura 32. Subasta combinatoria iterativa con fijación de precios. Estudio de la construcción de programas factibles.

Para ello se va a utilizar la plataforma Netlogo, en la que implementaremos un modelo basado en agentes. Uno de los aspectos que pretendemos estudiar es la capacidad del método para ser distribuido y en qué medida la utilización de métodos más distribuidos ofrece resultados diferentes.

Este capítulo se estructura como sigue: en primer lugar se explica la implementación del problema de *job shop* mediante un modelo basado en agentes. A continuación, se definen las instancias del problema *job shop flexible* que van a servir como base para la experimentación. En último lugar, se describe cada experimento realizado y se muestran sus resultados.

7.1. Implementación del problema *job shop* en la plataforma Netlogo.

Los diferentes mecanismos de programación de talleres se han implementado utilizando la plataforma Netlogo (versión 4.1.3). Netlogo forma parte de un amplio rango de plataformas que han sido desarrolladas para la generación de modelos basados en agente. Este tipo de plataformas, entre las que se encuentra Swam, Repast, Netlogo y Mason, proporcionan distintos servicios y librerías que pueden ser utilizados en los modelos de programación.

Al igual que RePast y Swarm, Netlogo proporciona servicios de programación, pero además ofrece un entorno gráfico que permite construir interfaces rápidamente para poder hacer correr los modelos (Berryman y Angus 2010). Una de las ventajas de Netlogo es su interfaz, que permite una mejor interacción con el usuario puesto que es posible agregar fácilmente elementos como botones, parámetros del modelo o gráficos que permiten monitorizar los resultados. Estos elementos pueden ser ubicados y configurados a gusto del programador.

7.1.1. Modelado de las entidades del problema

Se ha realizado la implementación del modelo de programación de la producción basado en subastas combinatorias definido en el capítulo 4. Para ello, se han definido cuatro tipos principales de agentes: un tipo representa a los recursos, otro tipo representa a las órdenes de fabricación, el tercer tipo representa a las tareas que componen cada orden, y, por último, el agente subastador que actúa como elemento central (Figura 33). En Netlogo existe un agente global al que se asigna el papel de subastador dentro del protocolo definido. Este agente realiza la monitorización de lo que ocurre en el modelo. Los agentes pueden interactúan siguiendo el protocolo de subasta combinatoria definido anteriormente.

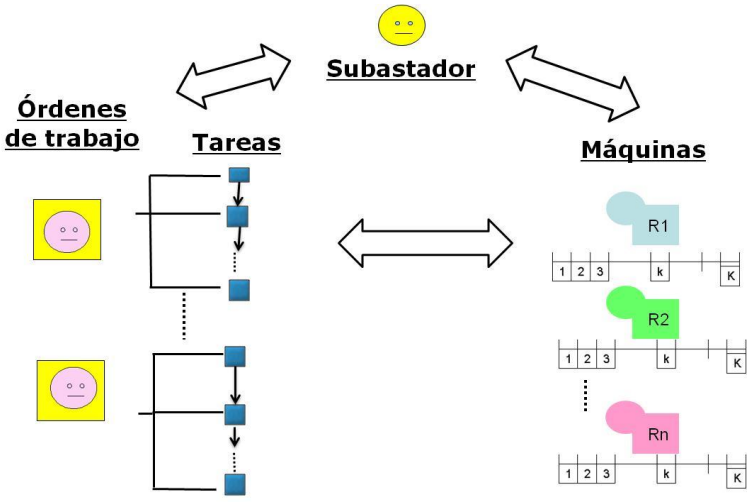

Figura 33. Modelo basado en agentes de la subasta

Agente recurso

Cada *agente recurso* representa una máquina o varias máquinas idénticas. Se definen distintas variables propias como son: las habilidades de que dispone, su capacidad -número de tareas que puede realizar al mismo tiempo- y la lista de multiplicadores de Lagrange para cada *slot* del horizonte de programación.

Agente orden

Cada orden de trabajo estará representada por un *agente orden*, que contendrá la información relevante sobre la orden de trabajo almacenada en las variables propias. Las principales son la fecha de entrada de la orden en el sistema; la fecha de entrega comprometida a partir de la cual se activa la función de penalización por retraso; y el peso, que afecta a la función de penalización y representa la prioridad de la orden de trabajo. Cada orden de trabajo está compuesta por un conjunto de tareas que hay que realizar siguiendo una determinada secuencia.

Agente tarea

Existirá un agente por cada tarea asociada con cada orden de trabajo que se cree. Las relaciones de precedencia entre las variables se definen mediante vínculos (*links*) dirigidos entre *agentes tarea*. La información básica respecto a cada tarea estará almacenada en las variables propias. Las principales son: la orden a la que pertenece la tarea; la habilidad necesaria para realizarla; la carga de trabajo que supone su

realización; y todas las variables relacionadas con la programación de la tarea, como tiempos de comienzo y de fin de la tarea y recurso asignado.

Agente subastador

Las funciones del *agente subastador* pueden ser asignadas al agente observador. El agente subastador es quien utiliza toda la información de los agentes individuales y establece la programación definitiva. A lo largo de la subasta se encarga de actualizar los precios de los *slot* de tiempo de los recursos.

7.1.2. Distribución del método de Relajación Lagrangiana mediante subastas

En los siguientes párrafos se presenta el desarrollo del método del subgradiente de forma distribuida entre los distintos agentes que componen el modelo. Se plantea un sistema distribuido utilizando un sistema multiagente en el que se resuelve el problema *job shop flexible* utilizando el método de Relajación Lagrangiana. Se toma el mecanismo de subasta como modelo para su distribución y utilizará el método de Relajación Lagrangiana como base del proceso de actualización de los precios. En la Figura 34 se muestra el esquema del método. A continuación se ilustra cada una de las fases con una breve descripción, indicando cuales son las principales entradas y resultados de cada una de las fases.

Inicialización de valores

Las *máquinas* asignan un valor inicial a los multiplicadores de Lagrange asociados a los intervalos de programación (*slots)*. Normalmente $\lambda_{hk} = 0$.

Fase 1. Resolución del problema relajado

Cada *orden* optimiza su función de utilidad (6.68) mediante el método de programación dinámica del apartado 6.3. En este método, que muestra la Figura 35, las *tareas* asociadas a la *orden* calculan la función recursiva que permite obtener el programa óptimo. Se desarrolla en dos fases principales:

1º Cálculo de la función auxiliar: la *orden* activa la primera *tarea*. Esta *tarea* calcula la función auxiliar para el cálculo de la función de coste de la orden según (6.78). Este proceso tiene como resultado un vector con los valores de la función auxiliar para cada *slot.* Al terminar el cálculo, la *tarea* activa la siguiente tarea para que se

repita el proceso. El proceso finaliza cuando la última *tarea* calcule el valor de la función auxiliar.

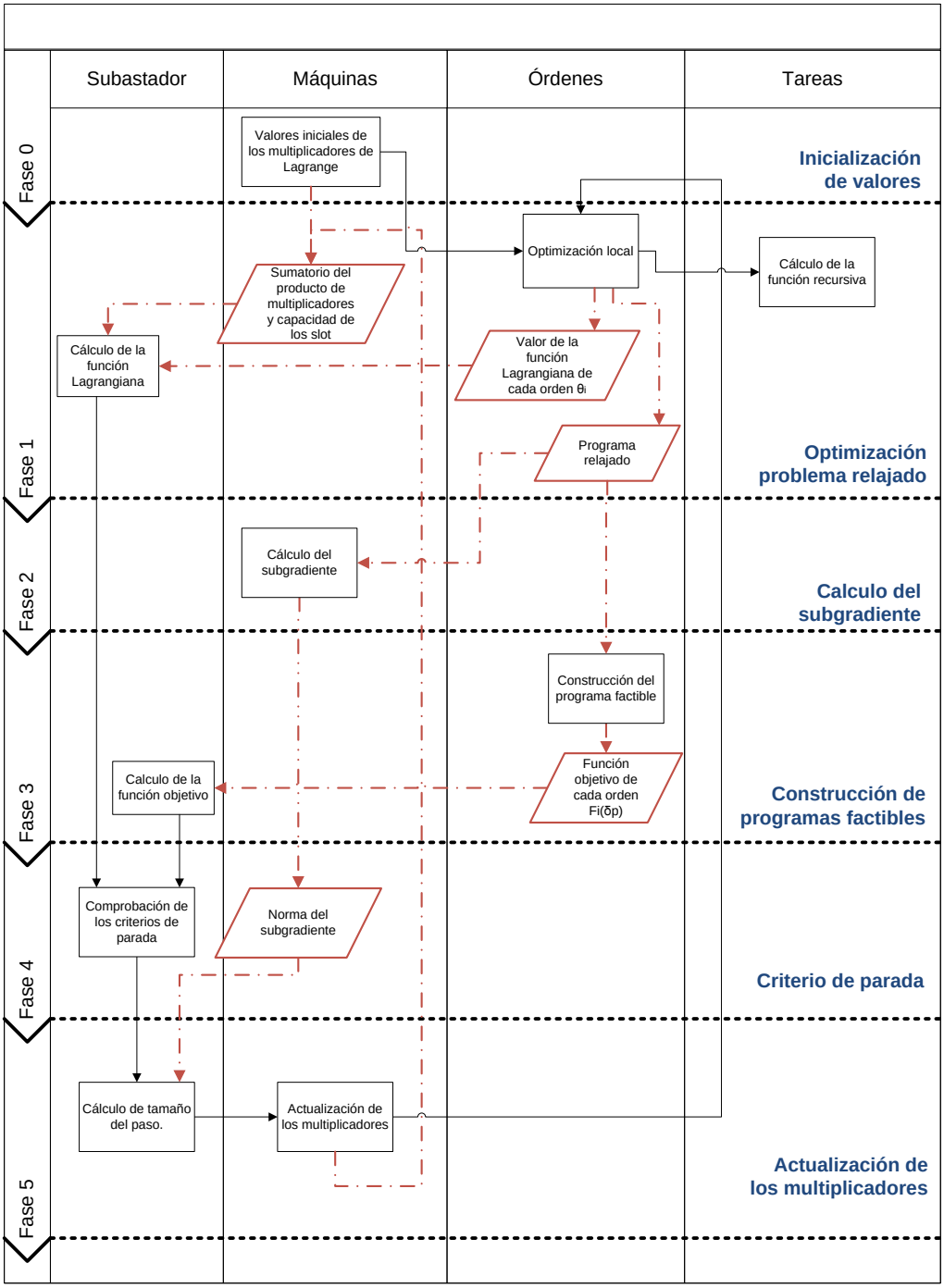

Figura 34. Método de Relajación Lagrangiana distribuido.

2º Selección del programa óptimo: la última *tarea* elige el *slot* de tiempo más temprano para el que la función auxiliar es mínimo. Este valor determina el momento de finalización de la tarea c_{i,j_l}^{RL}. La *tarea* activa su tarea precedente y le comunica el momento de finalización más tardío que puede programar ($k_{max} = c_{i,j_l}^{RL} - d_{ijh}$. La *tarea* precedente buscará el *slot* más temprano con el valor mínimo de la función hasta k_{max} y lo determina como momento de finalización de la tarea. El proceso se repite hasta la primera *tarea*.

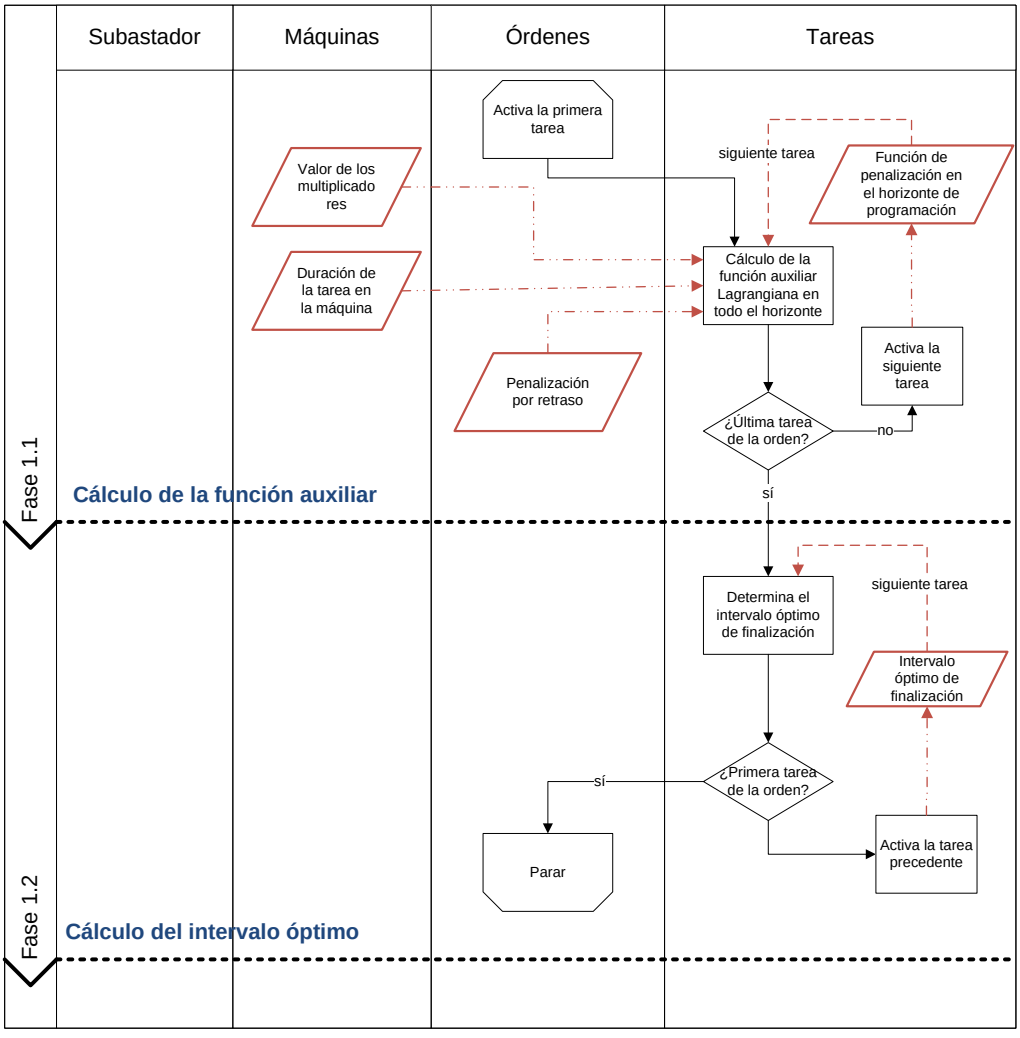

Figura 35. Cálculo de la función recursiva de optimización del problema relajado local.

La información que necesita cada *tarea* para realizar estos cálculos es:

- Su tarea precedente.
- Los precios de los *slots* (λ_{hk}) y duración de la tarea en la máquina h (d_{ijh}). Las obtiene de cada una de las máquinas disponibles.
- La última tarea necesita la función de penalización por el retraso $f_i\left(T_i(\delta_i)\right)$. Ésta es propia de la orden de trabajo.

Como resultado de este paso se obtiene:

- El problema propuesto por las tareas de cada orden de trabajo (δ_i^{*RL}).
- El valor de la función dual asociada a la orden de trabajo [$\theta_i(\lambda^r)$].

Fase 2. Cálculo del subgradiente

Cada *máquina* calcula la demanda de cada *slot* agregando las solicitudes de cada orden. Con esta información construye el subgradiente según (6.22).

La información que necesita cada máquina para realizar estos cálculos es:

- La capacidad en cada *slot* (M_{hk}). Esta información es propia de cada máquina.
- Los programas propuestos por las órdenes (δ_i^{*RL}). Se agregan las demandas en cada *slot hk* de las propuestas de las órdenes de trabajo.

Como resultado de este paso se obtiene la norma del vector subgradiente correspondiente a cada máquina $\|g_h^r\|^2$. Se envía al subastador, que sumará las enviadas por todas las máquinas.

Fase 3. Construcción del programa factible

Esta fase se plantea mediante los tres métodos revisados en el apartado 6.4. El método de lanzamiento de tareas sin retraso y el método de Giffler-Thompson siguen una estructura común que puede ser distribuida, como se observa en la Figura 36. Difieren en el paso de selección de la tarea.

La *orden* activa la primera *tarea*. La tarea *activa* solicita ser programada en la máquina asignada en el programa relajado. La *máquina* elige la tarea entre las elegibles y la programa lo antes posible. La *tarea* es programada y activa la siguiente tarea. Indicará su fecha de fin, que determinará la fecha de comienzo más temprana de la siguiente. En el método Giffler-Thompson, en el paso de selección de la tarea, la *máquina,* de entre las *tareas* elegibles, calcula el momento más cercano en el que la tarea puede quedar libre (FFS). Se queda con aquellas *tareas* cuyo tiempo de comienzo sea menor que FFS. En el método de lanzamiento de tareas sin retraso, de

entre todas las tareas elegibles se escoge aquella programada antes en el programa relajado.

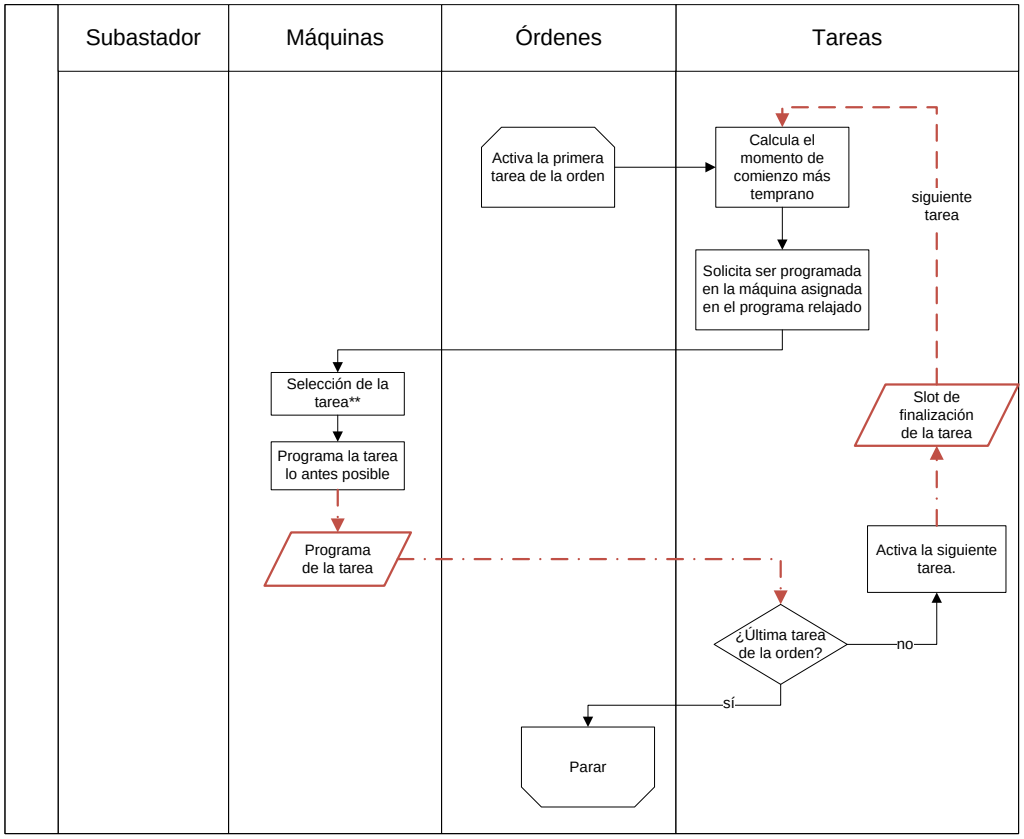

Figura 36. Método de construcción de programas

En el método del algoritmo voraz de Hoitomt et al (1993) se construye un programa que cumpla con las restricciones de capacidad a partir del programa agregado a partir de las propuestas de las órdenes de trabajo. Este método no es distribuible por lo que será realizado por el *subastador*.

La información que necesita cada máquina para realizar estos cálculos es qué tareas son elegibles y la información respecto a ellas (comienzo y fin en el programa relajado δ_i^{*RL}, fecha de comienzo más temprano, penalización marginal por retraso…).

Como resultado de este paso se obtiene el programa factible (δ_p^r), que se envía a las máquinas y a las órdenes de trabajo (tareas).

Fase 4. Criterio de parada

El *subastador* comprueba si se cumplen las condiciones para parar el proceso iterativo de cálculo (apartado 6.2.1).

La información que necesita el *subastador* para realizar estos cálculos es:

- Valor de la función Lagrangiana $\theta(\lambda^r)$ (6.26).
- Valor de la función objetivo $f\left(\delta_p^r\right)$.

Fase 5. Actualización de multiplicadores

1º Cálculo del tamaño del paso: en primer lugar el *subastador* calcula el tamaño del paso (α^r) con (6.29).

La información que necesita cada máquina para realizar estos cálculos es:

- El mejor valor de la función objetivo del problema con los programas factibles calculados hasta el momento ($V[f\left(x_p\right)]$).
- El valor de la función dual en la iteración $\theta(\lambda^r)$ (6.26). Las operaciones envían las subfunciones duales asociadas a cada una $\theta_i(\lambda^r)$ (6.20) y cada máquina h enviará la suma del producto en cada *slot* de los precios por la capacidad ($\sum_{k=1}^{K} \lambda_{hk} M_{hk}$).
- El valor de la norma del subgradiente al cuadrado $\|g^r\|^2$. Se obtiene como suma de las normas al cuadrado del subgradiente enviadas por las máquinas.

Como resultado de este paso se obtiene:

- El tamaño del paso (α^r) (6.30), que se envía a las máquinas para que puedan actualizar los precios.

2º Actualización de los multiplicadores: cada *máquina* actualiza los multiplicadores según (6.29).

La información que necesita cada *máquina* para realizar estos cálculos es:

- El valor del multiplicador en cada *slot* antes de actualizar (λ_{hk}^r) y el valor del subgradiente en dicho *slot* (g_{hk}^r).
- El tamaño del paso (α^r), que es enviado por el *subastador*.

Como resultado de este paso se obtiene el valor del multiplicador en cada *slot* actualizado (λ_{hk}^{r+1}), que permanecerá almacenado en cada máquina.

7.2. Test de prueba

En este punto se definen las instancias del problema *job shop* que han sido utilizadas para la realización de los experimentos. La mayor parte de los trabajos que estudian el problema *job shop* utilizan el criterio del makespan (Cmax), o tiempo de finalización de la totalidad de las tareas que se calcula mediante:

$$\min(max\ c_{ij}) \tag{7.1}$$

De hecho, la mayoría de colecciones de problemas utilizados como benchmark han sido definidas para la minimización de este objetivo (Jain y Meeran 1999). Dentro de este ámbito, existe un conjunto de colecciones de problemas que se utilizan como referencia en muchos trabajos (OR-LIBRARY). Los problemas que las componen se denominan mediante un sufijo y un número de orden que las identifica. Los sufijos hacen referencia al trabajo donde han sido desarrollados. Se denominan *abz* a los definidos en Adams et al (1988), *la* los definidos en Lawrence (1985), *mt* los definidos en Muth y Thompson (1963), y *or* los definidos en Applegate y Cook (1991).

En los últimos años se ha incrementado el número de trabajos que utilizan la suma ponderada del retraso de las órdenes de trabajo como función a optimizar, por lo que se han propuesto fórmulas para adaptar los problemas utilizados en la minimización del makespan a este segundo problema. Pinedo y Singer (1999) se basan en los benchmark anteriormente referidos para el criterio del makespan, y los transforman para utilizarlos en el problema *job shop* cuyo objetivo es la minimización del retraso total ponderado. Esta forma de construir nuevos problemas, en base a los desarrollados para el makespan, ha sido utilizado en Kreipl (2000), Asano y Ohta (2002), Essafi et al (2008), Zhou et al (2009a), Bülbül (2011) y Kachitvichyanukul y Sitthitham (2011).

Para la definición de cada caso se asigna un peso y una fecha de entrega comprometida a cada orden de trabajo del problema original, siguiendo las siguientes reglas:

- Pesos: se asigna al primer 20% de las órdenes un peso de 4 (muy importante), al siguiente 60% se le asigna un peso de 2 y al resto de órdenes se le asigna un peso de 1 (no importante).
- Fechas de entrega: la fecha de entrega dd_j del trabajo j es la suma de todos las duraciones de las operaciones que componen la orden p_{ij} multiplicado por el factor de ajuste de la fecha de entrega (*f*). El factor *f* permite controlar lo ajustadas que

están las fechas de entrega respecto a la duración mínima de las órdenes de trabajo (Baker 1984).

$$d_i = r_i + \mathrm{f} \cdot \sum_{j=1}^{o_i} \mathrm{p}_{ij} \tag{7.2}$$

De forma similar a lo que ocurre en el caso del problema *job shop*, la mayoría de los trabajos que estudian el problema *job shop flexible* se centran en el makespan como función objetivo. Para transformar estos problemas en problemas que estudien la suma ponderada del retraso es necesario crear unas fechas de entrega y unos pesos. Kreipl (2000) utiliza una variante de la transformación propuesta por Pinedo y Singer (1999). Esta transformación tiene las siguientes características:

- Los valores tomados para el factor de ajuste de la fecha de entrega son $f \in [0.3, 0.6, 0.9]$.
- Para las fechas de entrega se considerará la suma de la media de los tiempos de proceso en las distintas máquinas ($\overline{p}_{ij}$).

Como base de pruebas usaremos los problemas *job shop flexible* propuestos por Brandimarte (1993), cuyas características se recogen en la Tabla 7.1.

Denominación	**Tamaño (Máquinas x Trabajos)**
mk01-mk02	6x10
mk03-mk04	8x15
mk05	4x15
mk06	10x10
mk07	5x20
mk08-mk10	10x20

Tabla 7.1. Problemas *job shop* flexible (Brandimarte 1993)

En la Tabla 7.2 se muestra como ejemplo el problema *mk01*. Para cada problema se calculan las fechas de entrega de las órdenes utilizando el factor de ajuste con los valores: $f \in [0.3, 0.6, 0.9]$. En total se estudian 30 problemas diferentes, tres por cada problema, para los que utilizará la notación: *mkXX_f*.

		MAQUINAS					
Orden	**TAREAS**	**1**	**2**	**3**	**4**	**5**	**6**
1	**1**	5		4			
1	**2**		1	5		3	
1	**3**			4			2
1	**4**	1	6				5
1	**5**			1			
1	**6**			6	3		6
2	**1**		6				
2	**2**			1			
2	**3**	2					
2	**4**		6		6		
2	**5**	1	6				5
3	**1**		6				
3	**2**			4			2
3	**3**	1	6				5
3	**4**		6	4			6
3	**5**	1				5	
4	**1**	1	6				5
4	**2**		6				
4	**3**			1			
4	**4**		1	5		3	
4	**5**			4			2
5	**1**		1	5		3	
5	**2**	1	6				5
5	**3**		6				
5	**4**	5		4			
5	**5**		6		6		
5	**6**		6	4			6

		MAQUINAS					
Orden	**TAREAS**	**1**	**2**	**3**	**4**	**5**	**6**
6	**1**			4			2
6	**2**	2					
6	**3**		6	4			6
6	**4**		6				
6	**5**	1	6				5
6	**6**	3			2		
7	**1**						1
7	**2**	3			2		
7	**3**		6	4			6
7	**4**	6	6			1	
7	**5**			1			
8	**1**			4			2
8	**2**		6	4			6
8	**3**	1	6				5
8	**4**		6				
8	**5**		6		6		
9	**1**						1
9	**2**	1				5	
9	**3**			6	3		6
9	**4**	2					
9	**5**		6	4			6
9	**6**		6		6		
10	**1**			4			2
10	**2**		6	4			6
10	**3**		1	5		3	
10	**4**						1
10	**5**		6		6		
10	**6**	3			2		

Tabla 7.2 Problema "mk01". Duración tareas en unidades de tiempo.

7.3. Planificación de la experimentación

Comentamos, en este apartado, el conjunto de experimentos diseñados para estudiar el efecto de las distintas variantes del algoritmo. Dichos experimentos se han implementado sobre el conjunto de problemas definido. Para su definición se han realizado una serie de pruebas previas que han permitido acotar el rango de estudio de las variables. Cada experimento se ha definido variando la variable de estudio sobre el modelo base que ha servido como estándar (Tabla 7.3). El estudio se estructura en dos partes, una referente a los métodos de actualización de precios y otra referente a los métodos de construcción de programas factibles.

Actualización de precios	
Método de actualización	Subgradiente
Frecuencia de actualización del factor de corrector del tamaño del paso (γ)	50
Nº de iteraciones máximo	500
Construcción de programas factibles	
Método de construcción de programas:	Método de lanzamiento de tareas sin retraso.
Criterio principal para la construcción de programas factibles:	Momento de finalización de la tarea en el programa relajado.
Criterio secundario para la construcción de programas factibles:	Función de penalización por retraso

Tabla 7.3. Parámetros definidos para el método estándar

La primera parte se centra en el estudio de los métodos de actualización de precios y de los parámetros de funcionamiento de los métodos. El objetivo es poder estudiar qué métodos proporcionan mejores cotas inferiores a la solución del problema y si, a partir de cotas más ajustadas, se obtienen mejores soluciones. Para los métodos que no proporcionan una cota inferior, el objetivo es comparar la calidad de la solución obtenida. Los experimentos propuestos son los siguientes:

- La frecuencia de actualización del factor corrector del tamaño del paso.
- El método mejorado del cálculo del subgradiente.
- El cálculo del tamaño del paso diferenciado por recursos frente al cálculo global del tamaño del paso.

- La influencia del peso de la información de iteraciones anteriores en el proceso de actualización de precios.
- La propuesta alternativa en el cálculo del tamaño del paso en la actualización de precios.
- Métodos que no garantizan la cota inferior.
- Comparación del método del subgradiente con el método walrasiano no adaptativo.
- Estudio de métodos asíncronos.

La segunda parte se centra en los métodos de construcción de programas factibles. En este bloque de experimentos se utilizará como método de actualización de precios el método del subgradiente estándar definido en la Tabla 7.3. El objetivo es comparar los resultados que proporcionan los distintos métodos de construcción de programas a partir de cotas inferiores semejantes. También se estudiarán las diferencias en los resultados cuando se utiliza el método de lanzamiento de tareas sin retraso con distintos criterios de selección de tarea. Los experimentos propuestos en este bloque son los siguientes:

- El método constructivo a utilizar.
- El criterio principal de selección de las tareas.
- El método secundario de selección de las tareas en caso de empate.

Se realizan 50 repeticiones por experimento. En cada experimento realizado se compara para cada problema la cota inferior, la cota superior y el gap. La cota inferior hace referencia al mejor resultado la función dual $\theta(\lambda^r)$ (6.25) obtenido en el transcurso de las iteraciones. La cota superior hace referencia al mejor resultado de la función objetivo $f\left(\delta_p^r\right)$ (6.24) obtenido en el transcurso de las iteraciones. El gap es la diferencia entre la cota superior y la cota inferior (6.27), expresada en términos relativos respecto al valor de la cota superior.

Para poder analizar de forma agregada los resultados de los diferentes problemas estudiados en cada experimento se han definido las siguientes variables normalizadas: la cota inferior normalizada (7.3), la cota superior normalizada (7.4) y el gap normalizado (7.5).

$$CI\ norm = \frac{CI - \overline{CI}}{\sigma_{CI}} \tag{7.3}$$

$$CS\ norm = \frac{CS - \overline{CS}}{\sigma_{CS}} \tag{7.4}$$

$$GAP\ norm = \frac{gap - \overline{gap}}{\sigma_{gap}} \tag{7.5}$$

También se compara la velocidad de convergencia de los métodos estudiando la evolución de las cotas superior e inferior en el transcurso de las iteraciones. Se analiza, en cada iteración, el porcentaje que ha alcanzado en cada iteración la cota correspondiente respecto a la cota obtenida finalmente.

7.4. Análisis de los aspectos que afectan a la actualización de los precios

Estudiaremos los distintos métodos de actualización de precios basados en el método del subgradiente que resuelve el problema dual en el método de Relajación Lagrangiana. También compararemos los resultados obtenidos utilizando los métodos derivados del método del gradiente surrogado, que no garantizan una cota inferior. Por último, se comparará el método del subgradiente con el método walrasiano no adaptativo.

Cada uno de los puntos siguientes se estructura del siguiente modo: primero se explica el método o el parámetro que se quiere estudiar y su papel en el algoritmo, después se presentan los resultados agregados con las variables normalizadas, así como los resultados de cada problema de forma individual, se estudia la velocidad de convergencia en los diferentes casos, y por último se presentan las conclusiones.

7.4.1. Frecuencia de actualización del factor corrector del tamaño del paso

En el método del subgradiente (apartado 5.3.1) el cálculo del tamaño del paso en cada iteración está afectado por un factor de modificación γ que disminuye el valor del paso si el valor de la cota inferior o de la cota superior no ha mejorado en las últimas r_s iteraciones. En cada iteración, los precios (multiplicadores de Lagrange) se actualizan en función del grado de utilización de cada *slot* mediante (7.6), donde el tamaño del paso en cada iteración α^r se calcula con (7.7).

$$\lambda^{r+1} = max[0, \lambda^r + \alpha^r g(\lambda^r)] \tag{7.6}$$

$$\alpha^r = \gamma^r \cdot \frac{(UB^r - V(P_D(\lambda^r))}{\|g^r\|^2} \quad \text{con} \quad 0 < \gamma < 2 \tag{7.7}$$

Con este experimento se quiere estudiar el efecto que tiene la frecuencia de actualización del factor corrector del tamaño del paso (γ), tanto en la mejor solución obtenido del problema original como en el valor de la cota inferior calculada. En los trabajos estudiados hay una gran diversidad de valores para esta variable. Muchos de ellos no concretan el valor utilizado, o si han utilizado este factor corrector. En este experimento, se varía el valor de la frecuencia de actualización del factor de modificación del tamaño del paso, que toma los valores r_s {2, 10, 20, 50, 75, 100, 200, 300, 500}. Se ha realizado el conjunto de experimentos para la batería de problemas.

Comentamos los resultados de los experimentos realizados para los distintos problemas. Las conclusiones obtenidas, en base a aquellos, son independientes del tamaño del problema estudiado. Se aprecia un efecto significativo de la frecuencia con la que se ajusta el valor del paso sobre el valor de la cota inferior. En las Tablas 7.4, 7.5 y 7.6, se muestran en negrita los intervalos para los que el valor es máximo o mínimo, respectivamente, en cada problema. Para estos intervalos se considera que la diferencia entre los valores no es significativa.

Respecto a la cota inferior obtenida, los resultados globales muestran que existe una tendencia a obtener resultados diferentes para cada nivel de la variable r_s. En la Figura 37 se observan diferencias entre los valores extremos y los valores centrales. Los mejores valores se obtienen para el rango de valores entre 50 y 75, o más ampliado, entre 10 y 100. Al alejarse de este intervalo se va produciendo un deterioro del valor de la cota inferior.

Esto se confirma en la Figura 38 y en la Tabla 7.4, donde se presentan los resultados pormenorizados por problema. Con los valores extremos, que implican una reducción más frecuente, o por el contrario que prácticamente no exista la reducción en el tamaño del paso, se obtienen resultados mucho peores. Para los valores más bajos de la frecuencia de actualización de γ, el tamaño del paso disminuye su valor demasiado pronto. Para valores que implican una actualización más frecuente de γ, el valor del tamaño del paso no se llega a ver afectado y los valores del programa relajado oscilan. El porcentaje medio de la diferencia respecto al mejor valor es pequeño en el rango entre 10 y 100 –menor del 2%- como muestra la Tabla 7.7.

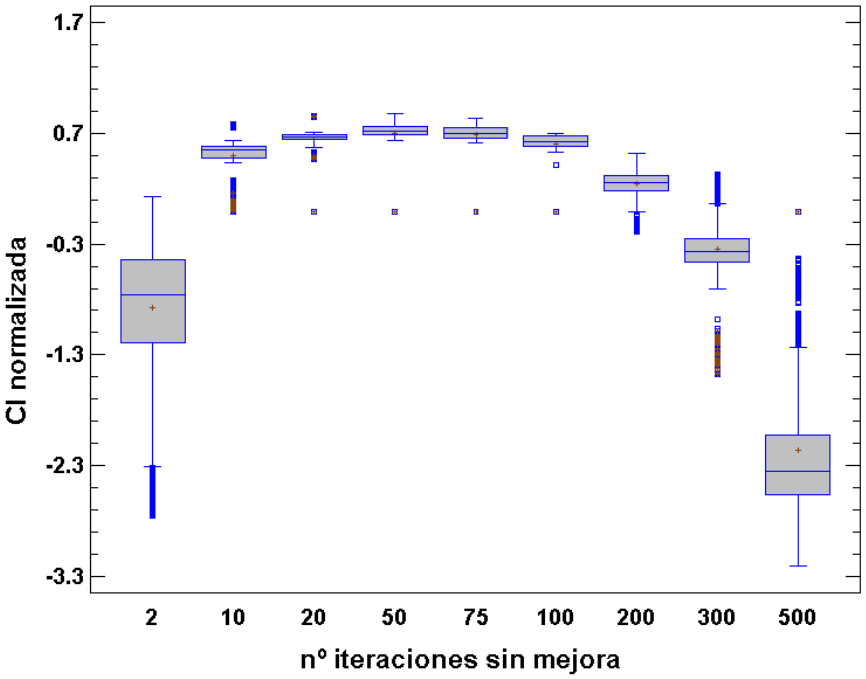

Figura 37. Cota inferior normalizada por el número de iteraciones (r_s) sin mejora.

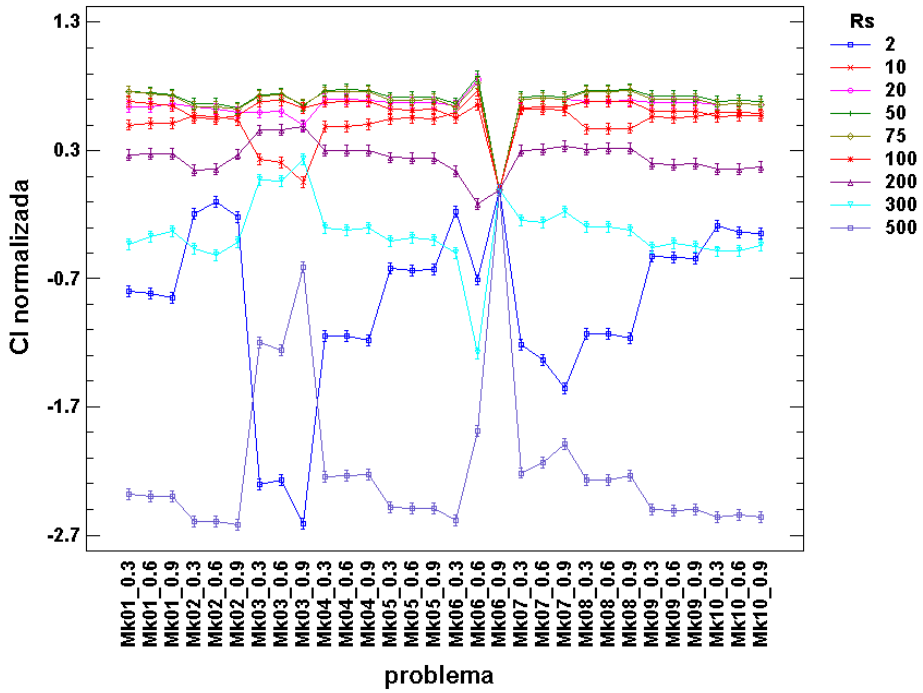

Figura 38. Cota inferior normalizada por el número de iteraciones sin mejora (r_s) para cada problema.

código	F	2	10	20	50	75	100	200	300	500
Mk01	0.3	174.0	180.3	181.0	**181.6**	**181.6**	181.2	179.2	175.8	166.4
	0.6	110.8	117.5	118.1	**118.6**	**118.6**	118.2	116.2	113.0	102.9
	0.9	54.5	60.8	**61.6**	**61.9**	**61.8**	61.5	59.8	57.0	47.4
Mk02	0.3	122.1	127.2	**127.7**	**127.8**	**127.7**	127.1	124.4	120.2	105.9
	0.6	57.8	62.1	**62.5**	**62.8**	**62.6**	62.0	59.5	55.2	41.7
	0.9	11.6	15.2	**15.4**	**15.6**	**15.5**	15.3	13.9	10.7	0.3
Mk03	0.3	1124.9	1299.3	1324.5	**1334.1**	**1333.6**	1330.5	1315.0	1288.3	1200.6
	0.6	696.3	860.4	886.6	**895.6**	**895.2**	892.2	876.9	850.9	763.1
	0.9	319.6	533.8	568.9	**581.5**	**581.8**	**579.7**	568.2	548.1	480.2
Mk04	0.3	384.0	410.5	413.9	**415.0**	**414.8**	413.5	407.4	397.8	366.5
	0.6	258.6	284.5	287.8	**289.0**	**288.7**	287.5	281.5	271.7	241.3
	0.9	140.8	166.9	169.8	**170.9**	**170.8**	169.6	163.8	154.5	124.8
Mk05	0.3	1237.0	1290.1	**1296.0**	**1297.7**	**1296.9**	1293.5	1276.4	1246.7	1151.5
	0.6	1022.4	1076.4	**1081.9**	**1083.7**	**1082.9**	1079.4	1062.2	1033.8	937.7
	0.9	803.5	857.2	**863.0**	**864.8**	**864.1**	860.6	843.3	814.0	717.6
Mk06	0.3	224.4	245.8	**247.3**	**247.6**	**247.0**	244.5	233.2	215.7	158.2
	0.6	17.5	39.4	**40.7**	**41.0**	**40.2**	37.7	26.3	8.9	0.0
	0.9	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
Mk07	0.3	1104.1	1218.4	**1223.7**	**1224.6**	**1223.4**	1219.1	1198.8	1164.7	1042.3
	0.6	805.7	922.2	**927.1**	**928.2**	**927.1**	922.9	903.5	869.5	758.5
	0.9	538.7	**662.6**	**667.4**	**668.6**	**667.7**	**664.0**	646.7	617.8	514.4
Mk08	0.3	4318.7	4553.8	4585.9	**4598.8**	**4597.4**	4585.9	4531.2	4441.9	4151.1
	0.6	3481.5	3717.4	3749.2	**3761.8**	**3760.5**	3749.1	3694.5	3604.4	3314.1
	0.9	2643.0	2881.7	2912.9	**2925.5**	**2924.1**	2912.4	2859.2	2766.2	2486.6
Mk09	0.3	2722.1	2899.5	2918.3	**2926.2**	**2922.7**	2907.1	2840.4	2733.6	2398.6
	0.6	1889.6	2069.3	2088.2	**2096.2**	**2092.7**	2077.4	2009.1	1907.6	1565.0
	0.9	1071.1	1248.6	1267.7	**1275.4**	**1271.8**	1256.7	1189.9	1086.6	754.4
Mk10	0.3	1984.5	2130.3	**2141.2**	**2144.8**	**2141.0**	2125.5	2057.6	1951.5	1607.4
	0.6	1206.0	1360.4	**1371.3**	**1374.9**	**1370.9**	1355.5	1286.9	1181.8	842.6
	0.9	513.4	662.8	**674.9**	**678.2**	**674.6**	660.9	597.1	498.5	160.0

Tabla 7.4. Cota inferior por el número de iteraciones sin mejora. Mediana por problema.

código	F	2	10	20	50	75	100	200	300	500
Mk01	0.3	**204.7**	**204.3**	**203.8**	**204.3**	**205.7**	206.1	207.5	209.2	209.5
	0.6	**140.9**	**141.4**	**140.8**	**141.5**	**141.6**	142.9	144.5	145.7	147.3
	0.9	84.4	**82.3**	**83.3**	**81.9**	**81.9**	**83.0**	85.0	85.8	88.2
Mk02	0.3	**160.3**	**159.1**	**160.6**	162.0	162.5	162.8	165.2	165.1	167.9
	0.6	94.0	**91.2**	**91.8**	**93.0**	94.1	95.7	96.8	98.2	99.7
	0.9	30.9	**27.9**	**27.9**	29.4	29.5	29.8	32.4	37.1	41.7
Mk03	0.3	**1492.6**	1520.1	1508.9	**1503.0**	1506.3	1518.1	1526.2	1532.9	1534.8
	0.6	**1054.1**	1068.6	1066.2	**1063.5**	1065.5	1073.5	1081.9	1087.3	1093.6
	0.9	**668.9**	**686.2**	**689.9**	**697.6**	702.3	703.4	712.4	718.6	724.9
Mk04	0.3	493.2	**486.1**	**488.7**	**488.4**	**487.3**	**489.8**	492.7	493.7	496.4
	0.6	366.0	**360.7**	**362.3**	**362.2**	**362.7**	**363.3**	**365.7**	368.3	370.3
	0.9	242.2	**236.8**	**236.9**	**239.7**	241.5	243.3	245.3	246.2	249.0
Mk05	0.3	1520.2	**1469.2**	**1472.2**	1481.6	1483.3	1489.8	1510.5	1526.4	1550.9
	0.6	1305.0	**1255.0**	**1255.6**	1268.0	1275.6	1279.6	1296.7	1307.5	1337.9
	0.9	1084.4	**1039.0**	**1038.6**	**1046.6**	**1048.6**	1057.8	1076.1	1090.9	1121.4
Mk06	0.3	**380.0**	**379.5**	**377.8**	**382.7**	**381.3**	**384.2**	392.2	396.7	405.5
	0.6	**174.7**	**173.8**	**177.0**	182.3	183.2	182.5	190.2	191.7	194.8
	0.9	34.5	33.7	36.0	38.7	39.4	38.7	40.3	**38.4**	40.8
Mk07	0.3	1504.6	**1441.1**	**1436.1**	**1440.9**	1456.3	1468.3	1494.9	1508.4	1538.8
	0.6	1198.9	**1145.3**	**1150.1**	**1157.8**	**1160.1**	1170.4	1187.9	1206.9	1223.7
	0.9	907.7	**839.0**	**842.4**	857.5	858.8	870.5	891.1	901.1	931.0
Mk08	0.3	5263.7	**5242.9**	**5237.3**	**5236.6**	**5250.2**	**5259.7**	5276.4	5296.6	5348.4
	0.6	4427.4	**4410.7**	**4395.3**	**4411.3**	**4416.4**	**4417.9**	4445.5	4455.8	4510.0
	0.9	3586.7	**3570.2**	**3558.0**	**3568.5**	**3580.0**	3589.8	3588.4	3631.4	3656.3
Mk09	0.3	**3691.4**	**3705.9**	**3715.3**	3726.8	3735.6	3758.7	3763.1	3802.0	3842.4
	0.6	**2854.6**	**2879.3**	2884.3	2903.3	2906.4	2914.0	2949.2	2948.5	3011.2
	0.9	**2035.7**	**2048.8**	**2051.9**	**2062.7**	2070.4	2080.1	2118.6	2133.6	2171.8
Mk10	0.3	**2887.8**	**2875.8**	**2889.3**	2904.7	2926.7	2949.1	2986.6	3026.2	3087.0
	0.6	**2109.8**	**2108.9**	**2124.9**	2147.6	2154.2	2170.1	2225.2	2249.6	2305.2
	0.9	**1326.8**	**1320.8**	**1327.0**	1362.3	1379.7	1403.4	1450.7	1483.9	1555.0

Tabla 7.5. Cota superior por el número de iteraciones sin mejora. Mediana por problema.

código	F	2	10	20	50	75	100	200	300	500
Mk01	0.3	0.149	**0.117**	**0.111**	**0.111**	**0.117**	0.121	0.136	0.159	0.206
	0.6	0.213	**0.169**	**0.161**	**0.162**	**0.162**	**0.173**	0.196	0.224	0.301
	0.9	0.353	**0.258**	**0.259**	**0.243**	**0.243**	**0.258**	0.296	0.335	0.461
Mk02	0.3	0.238	**0.200**	**0.205**	0.211	0.214	0.219	0.247	0.271	0.369
	0.6	0.384	**0.317**	**0.318**	**0.324**	**0.333**	0.351	0.385	0.437	0.581
	0.9	0.618	**0.455**	**0.443**	0.467	0.470	0.485	0.568	0.710	0.991
Mk03	0.3	0.246	0.145	0.122	**0.112**	**0.115**	0.124	0.138	0.159	0.218
	0.6	0.339	0.194	0.168	**0.158**	**0.160**	0.169	0.189	0.217	0.302
	0.9	0.522	0.222	**0.175**	**0.166**	**0.171**	**0.176**	0.202	0.237	0.337
Mk04	0.3	0.221	**0.155**	**0.153**	**0.150**	**0.149**	**0.156**	0.173	0.194	0.262
	0.6	0.293	**0.211**	**0.205**	**0.202**	**0.204**	**0.208**	0.230	0.262	0.348
	0.9	0.418	**0.294**	**0.283**	**0.286**	**0.292**	0.303	0.332	0.372	0.498
Mk05	0.3	0.186	**0.122**	**0.120**	**0.124**	0.126	0.132	0.155	0.183	0.257
	0.6	0.216	**0.142**	**0.138**	**0.145**	0.151	0.156	0.181	0.209	0.299
	0.9	0.258	**0.175**	**0.169**	**0.174**	**0.176**	0.186	0.216	0.254	0.360
Mk06	0.3	0.408	**0.352**	**0.345**	**0.353**	**0.352**	0.363	0.405	0.456	0.610
	0.6	0.899	**0.772**	**0.769**	**0.775**	0.780	0.793	0.861	0.953	1.000
	0.9	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**
Mk07	0.3	0.266	**0.154**	**0.148**	**0.150**	**0.160**	0.170	0.198	0.228	0.322
	0.6	0.327	**0.194**	**0.193**	**0.198**	**0.201**	0.211	0.239	0.279	0.379
	0.9	0.406	**0.209**	**0.207**	**0.220**	**0.222**	0.237	0.273	0.313	0.446
Mk08	0.3	0.179	0.131	**0.124**	**0.122**	**0.124**	0.128	0.141	0.161	0.224
	0.6	0.214	0.157	**0.147**	**0.147**	**0.148**	**0.151**	0.169	0.191	0.265
	0.9	0.263	0.193	**0.181**	**0.180**	**0.183**	0.189	0.203	0.238	0.320
Mk09	0.3	0.262	**0.218**	**0.214**	**0.215**	**0.218**	0.226	0.245	0.281	0.376
	0.6	0.338	**0.281**	**0.276**	**0.278**	**0.280**	0.287	0.319	0.353	0.480
	0.9	0.473	**0.390**	**0.382**	**0.381**	**0.386**	0.396	0.438	0.490	0.652
Mk10	0.3	0.313	**0.259**	**0.259**	**0.261**	0.268	0.279	0.311	0.355	0.479
	0.6	0.428	**0.355**	**0.355**	**0.360**	**0.363**	0.375	0.421	0.474	0.634
	0.9	0.613	**0.498**	**0.491**	**0.502**	0.511	0.529	0.588	0.664	0.896

Tabla 7.6. Gap por el número de iteraciones sin mejora. Mediana por problema.

	2	10	20	50	75	100	200	300	500
Dif CI	14.0%	1.7%	0.5%	0.0%	0.2%	1.0%	4.7%	10.9%	27.1%
Dif CS	1.9%	0.4%	0.8%	1.8%	2.2%	2.7%	4.3%	5.3%	7.2%
Dif gap	29.7%	4.7%	1.2%	1.5%	3.0%	6.7%	16.5%	26.6%	44.7%

Tabla 7.7. Diferencias respecto a los valores óptimos en cada problema por el número de iteraciones sin mejora.

Respecto a la cota superior, los mejores valores globales medios se obtienen para el rango de valores entre 2 y 75 (Figura 39). Esto se confirma para cada uno de los problemas estudiados, como se observa en la Figura 40. Las diferencias medias respecto al mejor valor son más bajas que respecto la cota inferior –hasta el 2% para el rango entre 2 y 75-.

Los valores del gap (Figuras 41 y 42, y Tabla 7.6) tienen un comportamiento similar, si bien las diferencias medias respecto al mejor valor son más acusadas en los extremos. Los valores de la cota inferior están poco dispersos, al contrario de lo que ocurre con los valores de la cota superior, donde los valores aparecen mucho más dispersos. En cuanto al valor de la cota superior, el efecto de la frecuencia de ajuste del valor del paso es menos significativo que en la cota inferior. El rango de valores es más amplio, entre 10 y 100. En este caso se obtienen mejores valores para las frecuencias más altas, entre 10 y 20. El efecto de la frecuencia de ajuste del paso sobre el gap se comporta de modo similar a la cota inferior. Los mejores valores se obtienen para los valores 50 y 75. A medida que nos alejamos de este intervalo los valores obtenidos son peores.

En cuanto a la convergencia del método, las Figuras 43 y 44 muestran la evolución de la cota superior e inferior respectivamente con el paso del tiempo. Estas gráficas muestran que los valores de frecuencia entre 2 y 20 tienen una convergencia más rápida hacia el valor óptimo, de modo que se acercan más a este valor en las primeras iteraciones que el resto de valores. Es interesante observar como en el valor de 2 es el que mejores valores obtiene en las primeras iteraciones, mientras que el valor de la cota inferior es peor que los valores 10 y 20. Esto es debido a que en esos primeros instantes con una mayor frecuencia de actualización del paso se puede explorar una región más amplia de soluciones, encontrando antes una solución mejor. Sin embargo, con el paso de las iteraciones, una frecuencia tan alta en la corrección del paso hace que el método se estanque de forma temprana y no pueda avanzar.

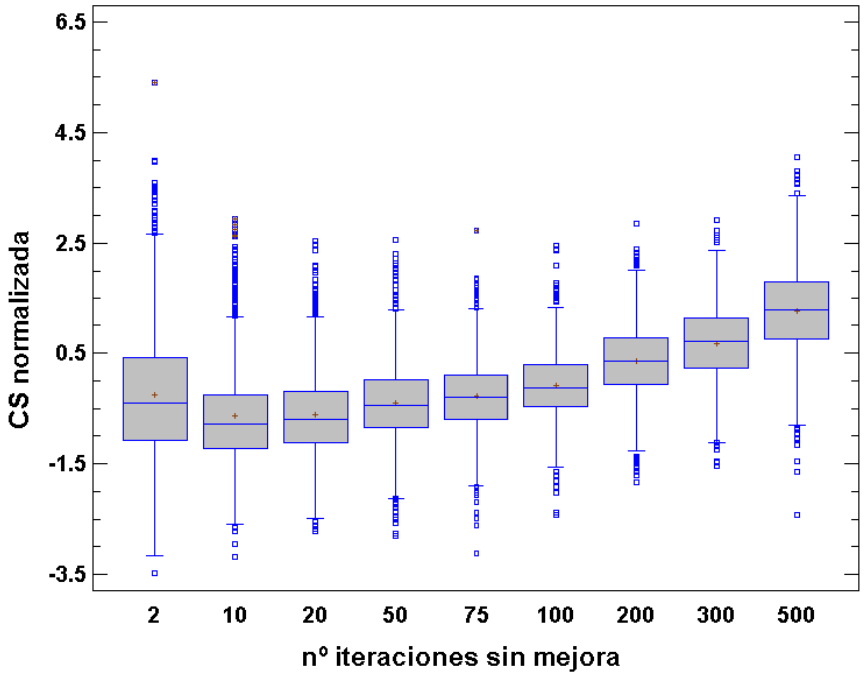

Figura 39. Cota superior normalizada por el número de iteraciones (r_s) sin mejora.

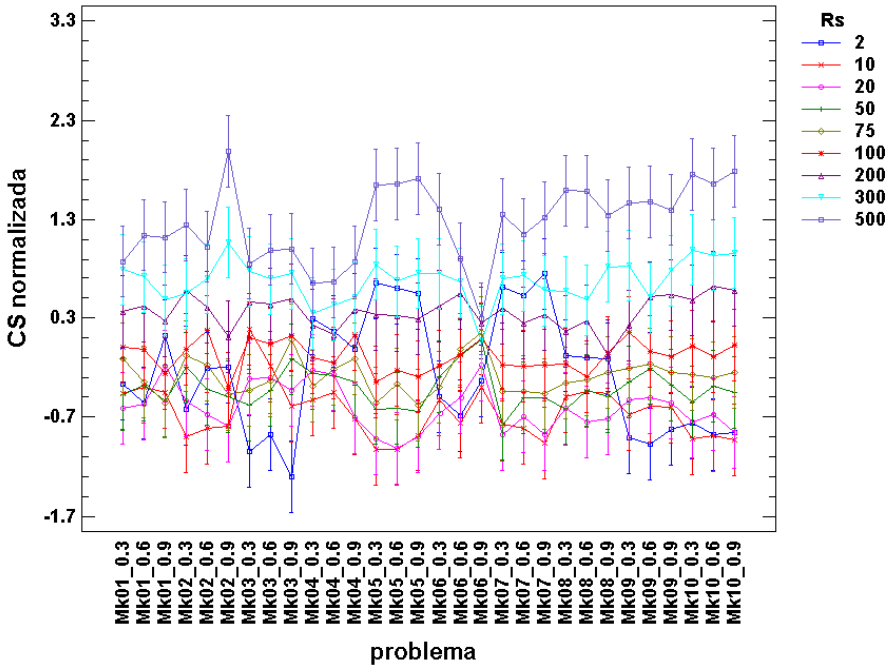

Figura 40. Cota superior normalizada por el número de iteraciones sin mejora (r_s) para cada problema.

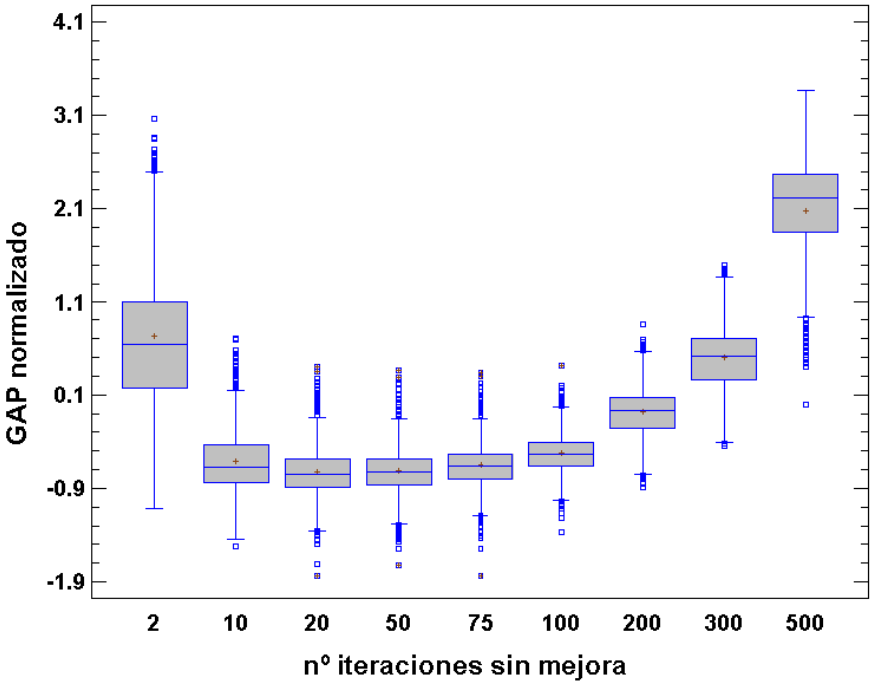

Figura 41. Gap normalizado por el número de iteraciones (r_s) sin mejora.

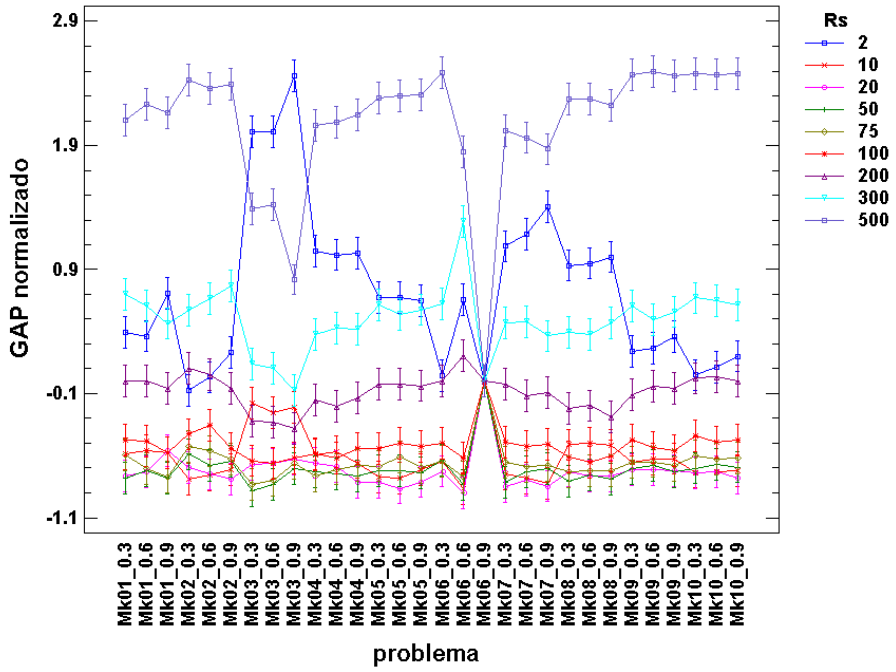

Figura 42. Gap normalizado por el número de iteraciones sin mejora (r_s) para cada problema.

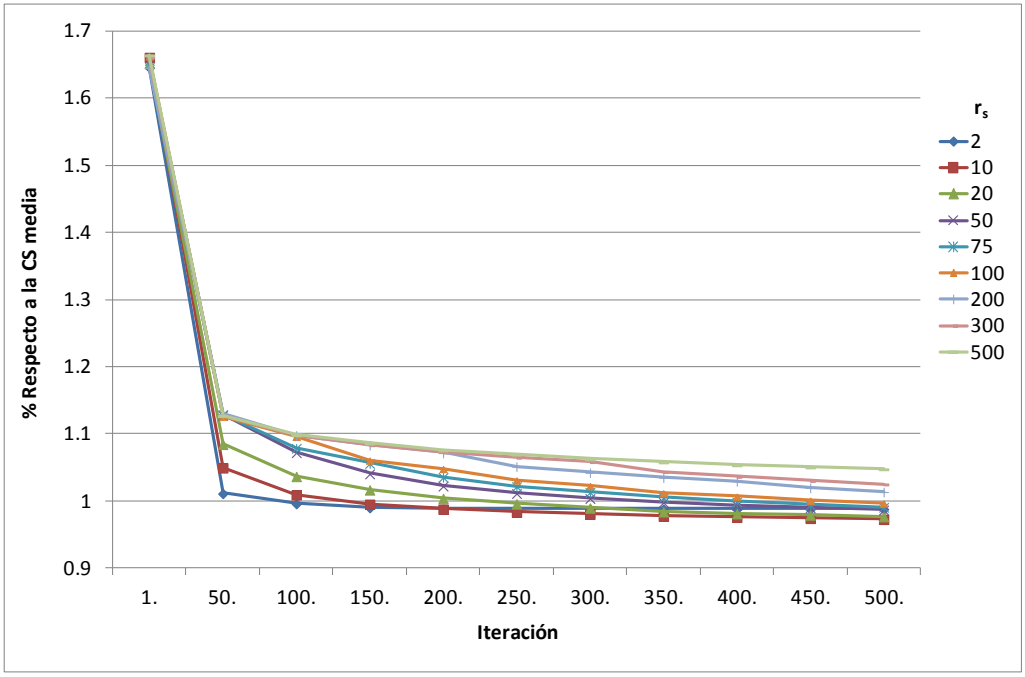

Figura 43. Porcentaje de la cota superior en cada iteración respecto a la cota superior media final por el número de iteraciones sin mejora.

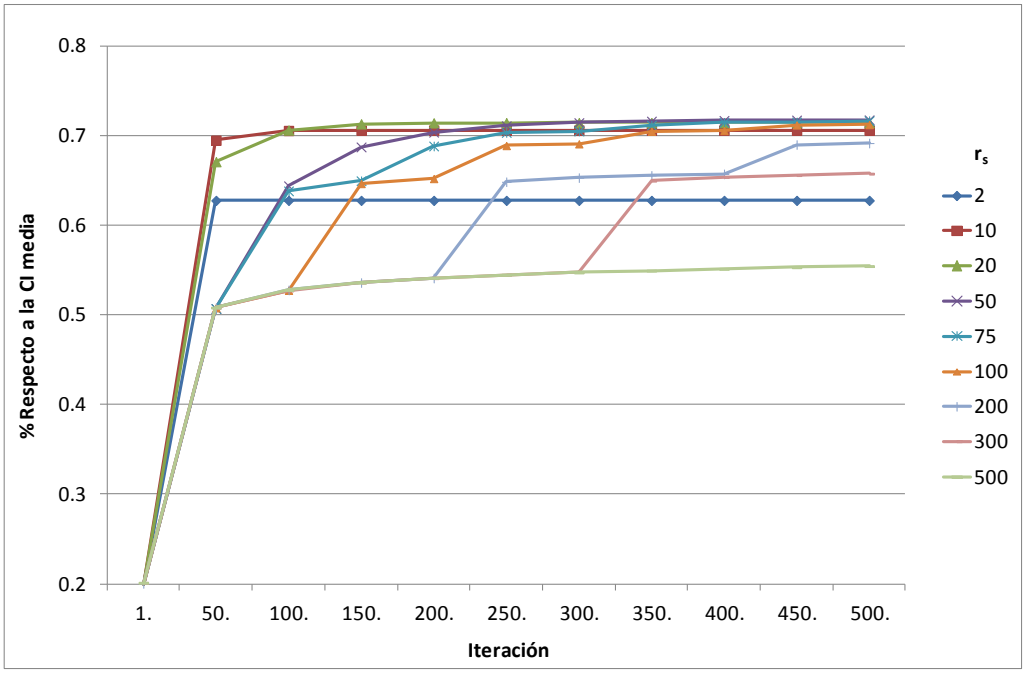

Figura 44. Porcentaje de la cota inferior en cada iteración respecto a la cota inferior media final por el número de iteraciones sin mejora.

7.4.2. Método mejorado del cálculo del subgradiente

Uno de los problemas de la utilización del método de Relajación Lagrangiana y de su resolución mediante el método del subgradiente es que, dado que el horizonte de programación puede ser grande, podrían existir muchas restricciones relajadas inactivas, ligadas a los *slots* de tiempo que no han sido solicitados por ninguna orden de trabajo. El valor del subgradiente en ese punto tiene un valor negativo y un valor absoluto igual a la capacidad del recurso. Esto provoca que el módulo del gradiente sea muy grande -más grande cuanto mayor sea el horizonte de programación- y mayor sea el número de *slots* inactivos. Como en la expresión que permite calcular el tamaño del paso aparece el módulo del subgradiente en el denominador, si el denominador es grande, el tamaño del paso será más pequeño. Como estos *slots* están inactivos, la información que aportan en el cálculo del subgradiente es innecesaria, por lo que parece adecuada su eliminación en el cálculo. Beasley (1993) y Wang (2003) proponen eliminarlos haciendo cero el valor del subgradiente en aquellos *slots* con subgradiente negativo y cuyos multiplicadores de Lagrange sean cero (apartado 6.2.1). El objeto de este experimento es estudiar el efecto de utilizar la mejora propuesta en el cálculo del subgradiente. En este experimento se varía el método utilizado en el cálculo del subgradiente. Se utiliza el método estándar y el método mejorado.

Respecto a la cota inferior los resultados globales muestran un mejor comportamiento cuando se aplica el método mejorado del cálculo del subgradiente (Figura 45), tendencia que se confirma cuando se analiza el comportamiento en cada problema, como muestra la Figura 46. El valor en porcentaje de la diferencia relativa de cada opción respecto al mejor valor en cada problema es de un 8% en la cota inferior cuando se aplica el cálculo del subgradiente por el método original (Tabla 7.9). En cuanto a la cota superior (Figuras 47 y 48), las diferencias entre la opción del método original y el método mejorado son menos acusadas que en la cota inferior. En una tercera parte de los problemas no existen diferencias significativas entre ambos métodos y las diferencias en porcentaje respecto al mejor valor en ambos métodos son bajas, alrededor del 1% como se muestra en la Tabla 7.9.

Para estudiar las diferencias entre ambos métodos se ha aplicado la prueba U de Mann-Whitney para comparar medianas con un nivel de confianza del 95%. En la Tabla 7.8 se recogen los resultados de la mediana de cada problema para la cota inferior, la cota superior y el gap. Se han resaltado en negrita los valores óptimos correspondientes, destacando ambas opciones cuando no hay diferencias significativas. En general todos los datos obtenidos son muy homogéneos, teniendo

coeficientes de variación para cada problema y opción menores del 1% en el caso de la cota inferior y menores del 2% en el caso de la cota superior.

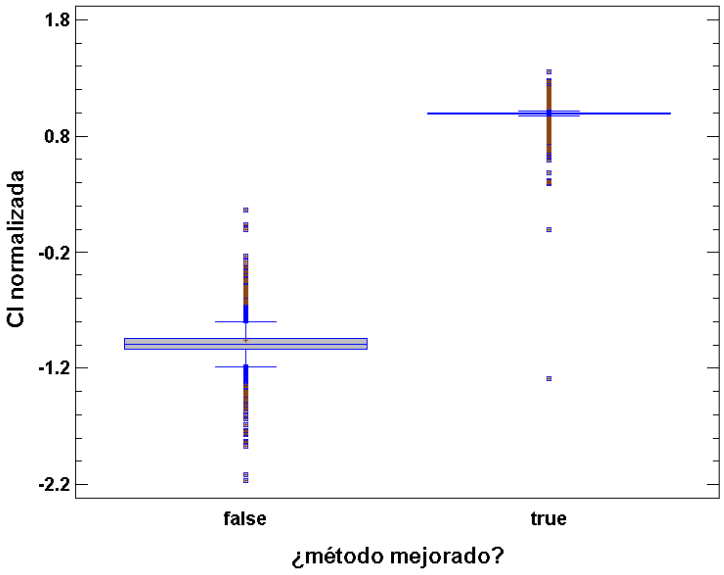

Figura 45. Cota inferior normalizada. Comparación entre el método del subgradiente estándar (false) y el mejorado (true).

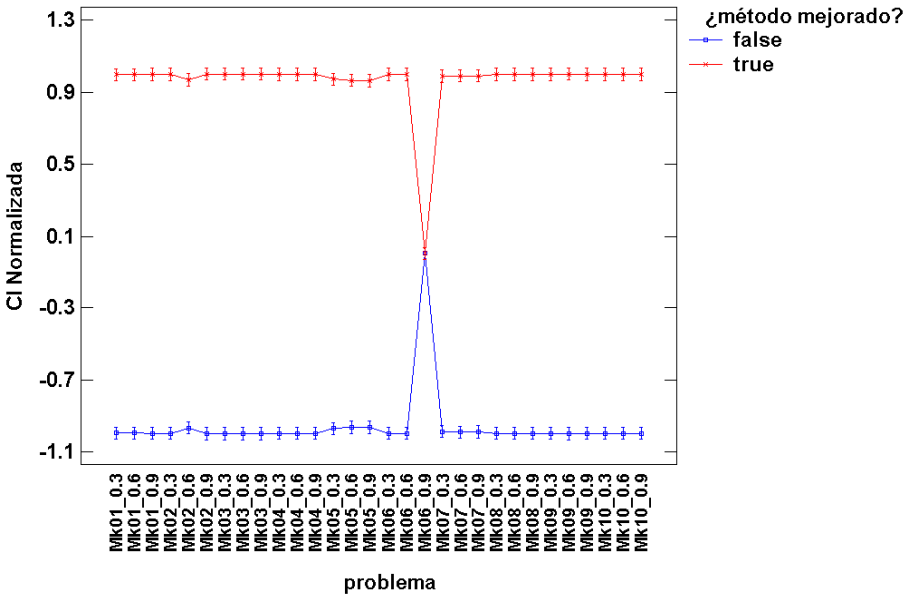

Figura 46. Cota inferior normalizada por problemas. Comparación entre el método del subgradiente estándar (false) y el mejorado (true).

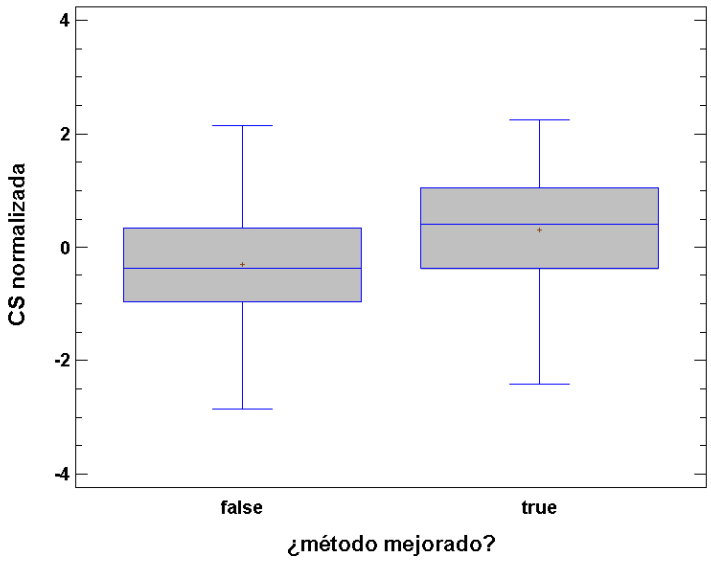

Figura 47. Cota superior normalizada. Comparación entre el método del subgradiente estándar (false) y el mejorado (true).

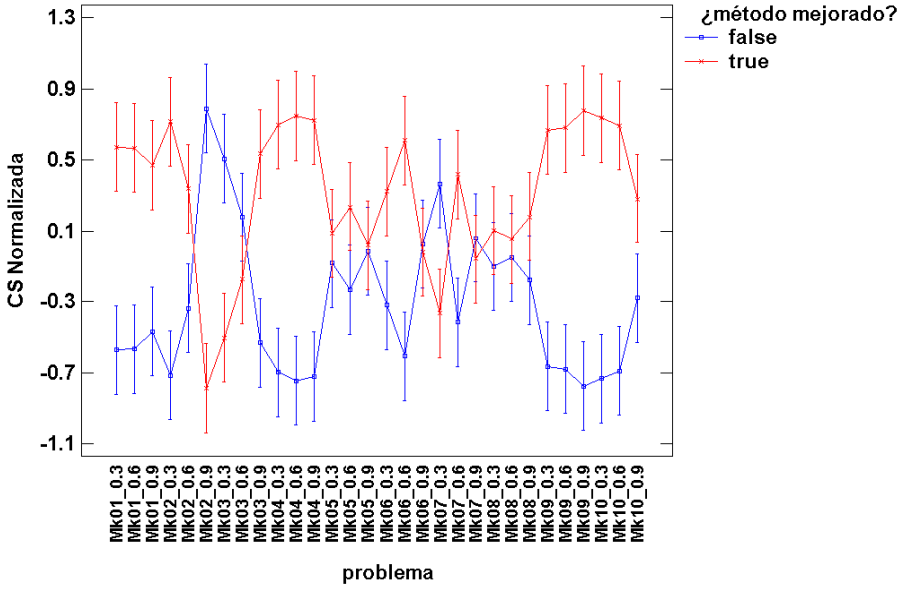

Figura 48. Cota superior normalizada por problemas. Comparación entre el método del subgradiente estándar (false) y el mejorado (true).

Los resultados obtenidos, en ambos métodos, para el gap normalizado se representan en las Figuras 49 y 59.

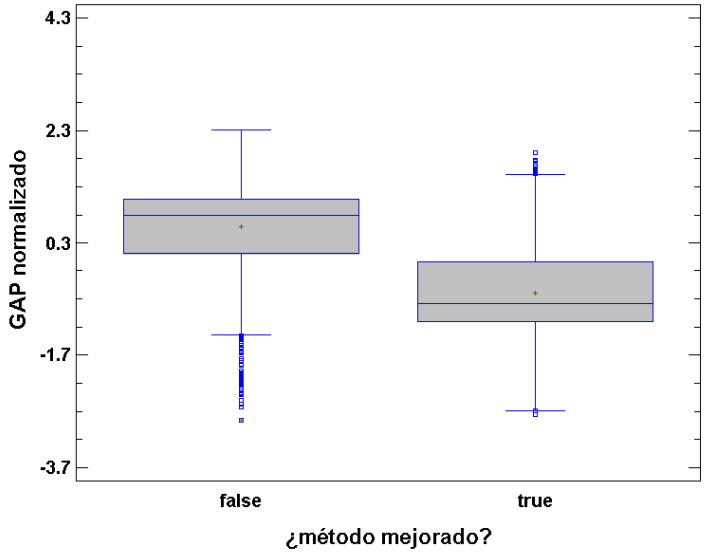

Figura 49. Gap normalizado por problemas. Comparación entre el método del subgradiente estándar (false) y el mejorado (true).

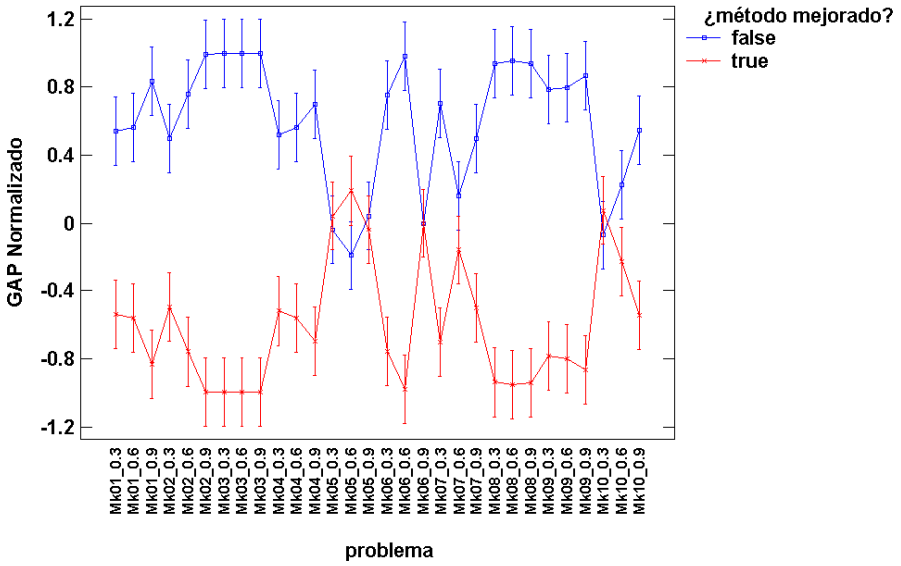

Figura 50. Gap normalizado por problemas. Comparación entre el método del subgradiente estándar (false) y el mejorado (true).

En las Figuras 51 y 52 se observa la influencia de la utilización del método mejorado del tamaño del paso sobre la velocidad de convergencia de la cota superior e inferior, respectivamente. Respecto la cota inferior se observa que inicialmente el método estándar de cálculo del subgradiente tiene una convergencia más rápida, pero, con el paso de las iteraciones, el valor de la cota obtenido es peor que cuando se utiliza

el método mejorado del tamaño del paso. Sin embargo, este mejor valor no tiene su reflejo en una mejora de la cota superior, siendo semejantes los valores obtenidos con ambos métodos.

		CI prom		CS prom		GAP prom	
cod	F	false	true	false	true	false	true
Mk01	0.3	177.1	**181.6**	**201**	204	0.119	0.110
	0.6	114.3	**118.6**	**139**	142	0.215	0.206
	0.9	57.1	**61.9**	**78**	80	0.259	0.110
Mk02	0.3	124.0	**127.8**	**158**	161	0.156	0.146
	0.6	59.7	**62.8**	**91**	92	0.124	0.124
	0.9	7.3	**15.6**	32	**29**	0.365	0.347
Mk03	0.3	1122.1	**1334.1**	1513	**1499**	0.160	0.148
	0.6	680.8	**895.6**	**1067**	**1064**	0.145	0.124
	0.9	363.3	**581.3**	**680.5**	**694.5**	0.229	0.216
Mk04	0.3	404.2	**415.0**	**479**	486	0.263	0.264
	0.6	278.2	**289.0**	**352**	361	0.178	0.164
	0.9	161.4	**170.9**	**231**	238	0.348	0.321
Mk05	0.3	**1297.1**	**1297.7**	**1481**	**1481.5**	0.364	0.158
	0.6	**1083.1**	**1083.7**	**1264**	**1269**	0.210	0.200
	0.9	**864.1**	**864.8**	**1048**	**1047**	0.143	0.146
Mk06	0.3	238.7	**247.6**	**376**	379	0.830	0.770
	0.6	29.1	**41.0**	**171**	178	0.193	0.192
	0.9	**0.0**	**0.0**	**39**	39	0.173	0.147
Mk07	0.3	1212.6	**1224.6**	1445	**1437**	0.294	0.275
	0.6	915.5	**928.2**	**1136**	1149	0.361	0.357
	0.9	660.8	**668.6**	**848**	**847.5**	0.268	0.227
Mk08	0.3	4484.0	**4598.8**	**5243**	**5247**	0.773	0.462
	0.6	3646.8	**3761.7**	**4410**	**4409**	0.466	0.164
	0.9	2811.1	**2925.5**	**3565**	**3568.5**	0.300	0.282
Mk09	0.3	2843.6	**2926.2**	**3689.5**	3732	0.175	0.174
	0.6	2013.3	**2096.2**	**2852.5**	2893	1.000	1.000
	0.9	1190.7	**1275.4**	**1999.5**	2053.5	0.220	0.211
Mk10	0.3	2106.1	**2144.8**	**2857.5**	2913	0.212	0.180
	0.6	1335.8	**1374.8**	**2089**	2139.5	0.404	0.379
	0.9	655.2	**678.3**	**1342.5**	1359	0.512	0.501

Tabla 7.8. Comparación entre los resultados del método del subgradiente estándar (false) y el mejorado (true). Mediana por problema.

	false	true
% Dif CI	8.0%	0.0%
% Dif CS	0.4%	1.2%
% Dif gap	12.4%	0.1%

Tabla 7.9. Diferencias respecto a los valores óptimos en cada problema. Comparación entre el método del subgradiente estándar (false) y el mejorado (true).

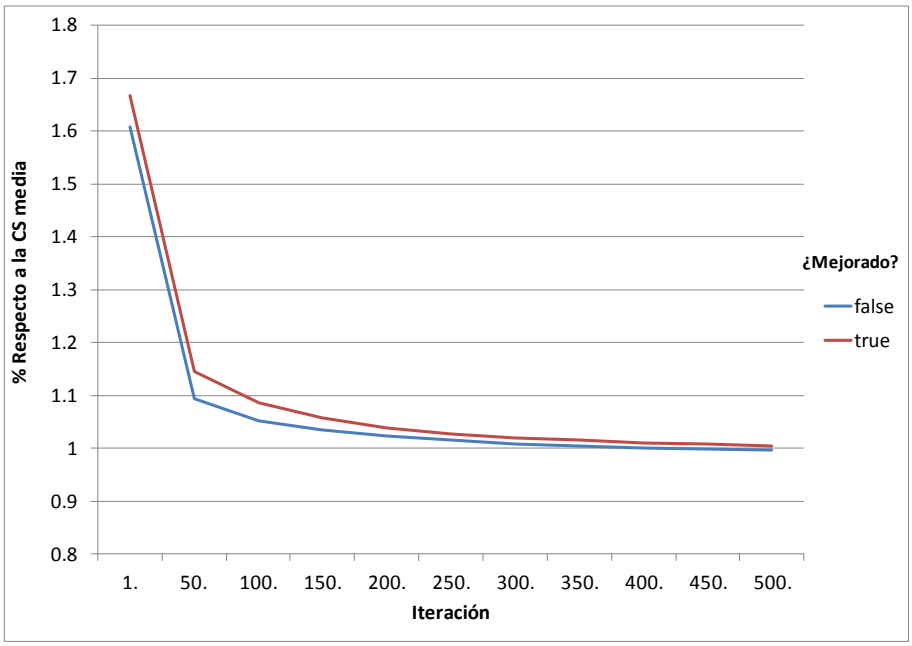

Figura 51. Porcentaje de la cota superior en cada iteración respecto a la cota superior media final. Comparación entre el método del subgradiente estándar (false) y el mejorado (true).

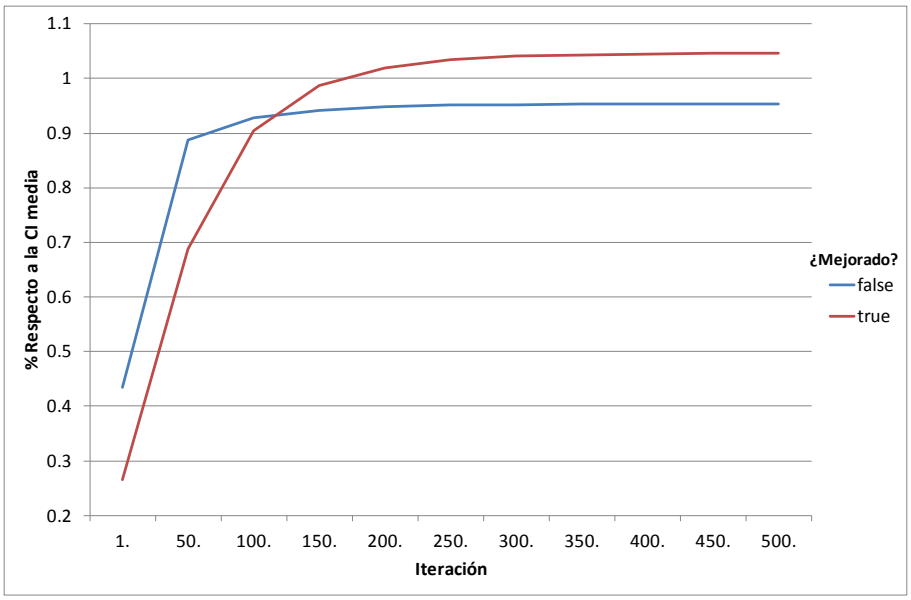

Figura 52. Porcentaje de la cota inferior en cada iteración respecto a la cota inferior media final. Comparación entre el método del subgradiente estándar (false) y el mejorado (true).

7.4.3. Cálculo del tamaño del paso diferenciado por recurso frente al cálculo global del tamaño del paso

Se quiere probar el efecto del gradiente normalizado (apartado 6.2.1) utilizado por Kaskavelis y Caramanis (1998) y Sun et al (2006). Este método busca calcular un tamaño de paso diferente para cada recurso, en función del subgradiente correspondiente a cada recurso, de forma que se adapte la velocidad de ajuste de los precios en función de la ocupación de cada recurso. Además permite distribuir por recursos el cálculo del subgradiente, aunque siga necesitando información global como el valor de la mejor solución y el valor de la función dual en la iteración.

La actualización de precios se realizará mediante:

$$\lambda_{\mathrm{hk}}^{r+1} = \max(0, \lambda_{\mathrm{hk}}^{r} + \alpha_{\mathrm{h}}^{\mathrm{r}} g_{\mathrm{hk}}^{r}) \qquad \begin{matrix} k = 1,2,\dots,K \\ h = 1,2,\dots,m \end{matrix} \tag{7.8}$$

donde el tamaño del paso normalizado $\alpha_{\mathrm{h}}^{\mathrm{r}}$ para cada recurso *h* se calcula con:

$$\alpha_{\mathrm{h}}^{\mathrm{r}} = \gamma^{r} \frac{(V[f(\delta_p)] - \theta(\lambda^{r}))}{\left\| g_h^r \right\|^2} \qquad con\ 0 < \gamma^{n} < 2 \tag{7.9}$$

y g_h^r, subgradiente correspondiente a los *slot* de la máquina *h*, según:

$$g_h^r = g^r(\lambda_n^r) = \{g_{hk}\ \ con\ k = 1,2,\dots,K\} \tag{7.10}$$

En este experimento se varía el método utilizado en el cálculo del subgradiente. Se utiliza el método estándar y el método normalizado. Se crea una variable booleana en la que el valor verdadero (*true*) indica que se utiliza el cálculo del paso normalizado, y el valor falso (*false*) indica que se utiliza el método de cálculo del paso estándar.

Se han comparado los resultados obtenidos en ambos métodos: la cota inferior normalizada (Figuras 53 y 54), la cota superior normalizada (Figuras 55 y 56) y el gap normalizado (Figuras 57 y 58), aplicando la prueba U de Mann-Whitney para comparar medianas. Debido a que el valor -P es menor que 0.05, existe una diferencia estadísticamente significativa entre las medianas con un nivel de confianza del 95%. Se han comprobado las diferencias existentes para cada problema.

En la Tabla 7.10 se resaltan en negrita los mejores valores obtenido. En los problemas en los que no se han encontrado diferencias significativas aparecen ambos valores en negrita. Los resultados muestran que en la mayor parte de los problemas se obtienen resultados significativamente diferentes en la cota inferior y superior. Se obtienen mejores resultados cuando no se utiliza el cálculo del paso normalizado. Las

diferencias obtenidas en la cota inferior están alrededor del 2,9%. Respecto a la cota superior, las diferencias encontradas se sitúan alrededor del 4,2% (Tabla 7.11).

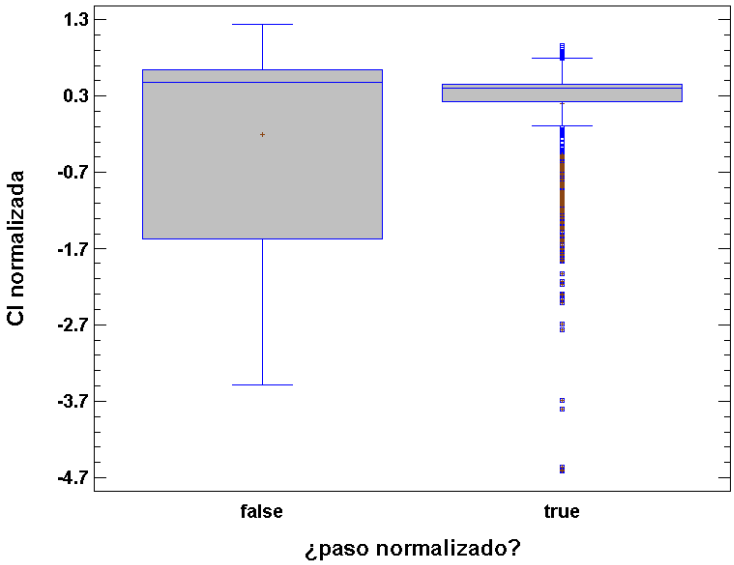

Figura 53. Cota inferior normalizada. Comparación entre el método del subgradiente estándar (false) y el normalizado (true).

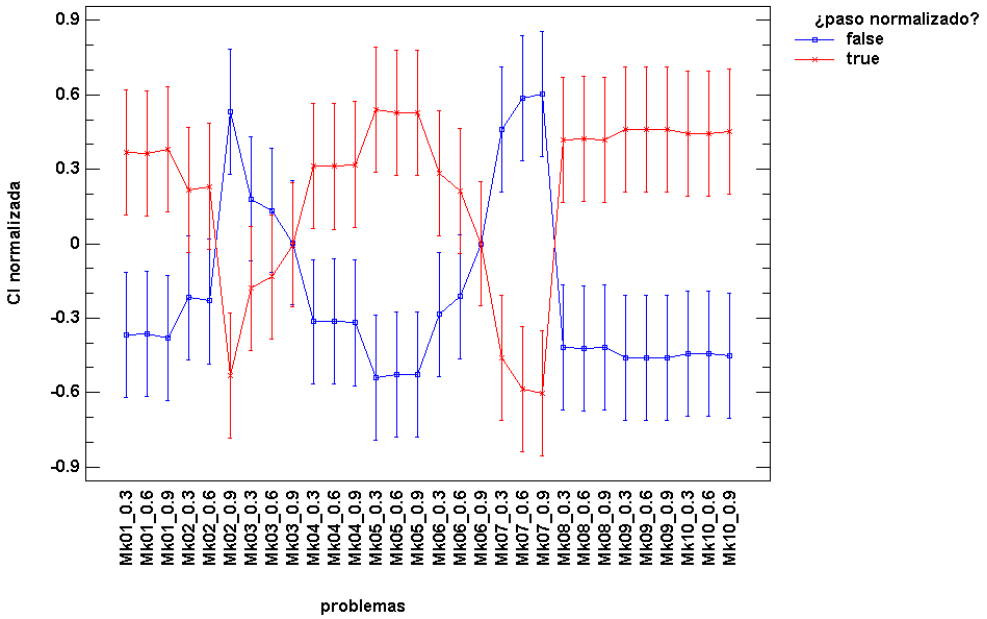

Figura 54. Cota inferior normalizada por problemas Comparación entre el método del subgradiente estándar (false) y el normalizado (true).

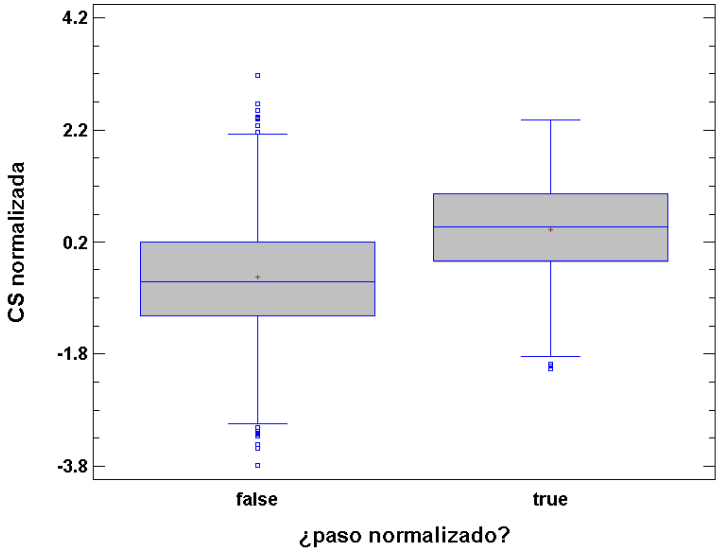

Figura 55. Cota superior normalizada. Comparación entre el método del subgradiente (false) y el normalizado (true).

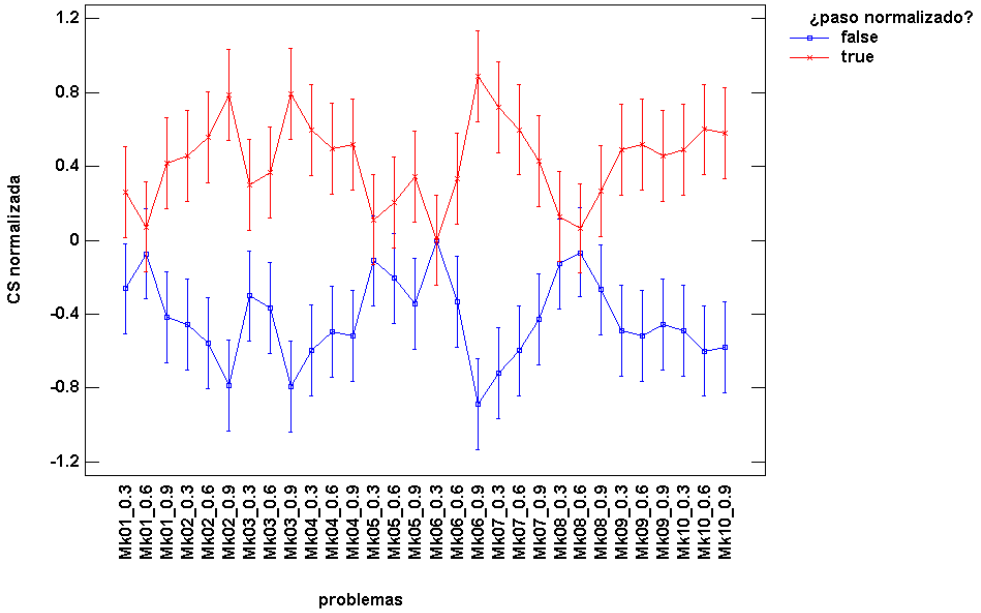

Figura 56. Cota superior normalizada por problemas Comparación entre el método del subgradiente estándar (false) y el normalizado (true).

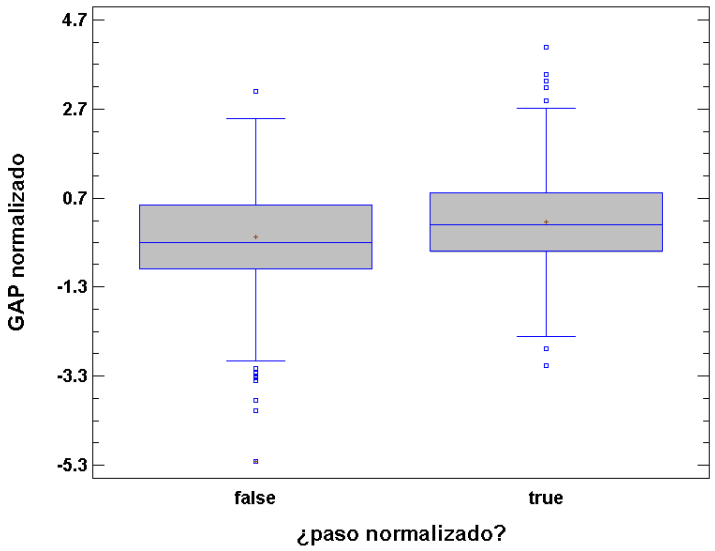

Figura 57. Gap normalizado. Comparación entre el método del subgradiente estándar (false) y el normalizado (true).

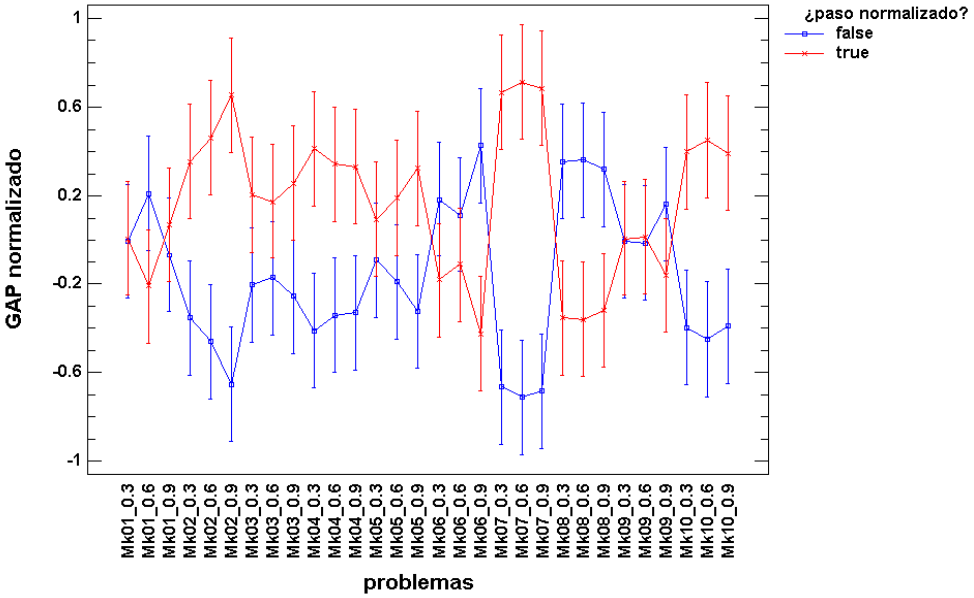

Figura 58. Gap normalizado por problemas Comparación entre el método del subgradiente estándar (false) y el normalizado (true).

		CI prom		CS prom		Gap prom	
código	**F**	**false**	**true**	**false**	**true**	**false**	**true**
Mk01	0.3	**181.6**	181.4	**201**	204	0.114	0.111
	0.6	**118.6**	118.4	**140**	142	0.170	0.165
	0.9	**61.9**	61.7	**78**	80	0.231	0.230
Mk02	0.3	**127.8**	127.3	**160.5**	163	**0.215**	0.220
	0.6	**62.8**	62.3	**91**	94.5	**0.332**	0.344
	0.9	**15.6**	9.5	**30**	40.5	**0.480**	0.771
Mk03	0.3	**1334.1**	1160.9	**1509**	1514.5	**0.121**	0.236
	0.6	**895.5**	804.0	**1064.5**	1080.5	**0.170**	0.256
	0.9	**580.4**	566.2	**677**	753.5	**0.181**	0.287
Mk04	0.3	**414.9**	413.7	**481**	493	**0.153**	0.161
	0.6	**289.0**	287.7	**356**	365	**0.205**	0.212
	0.9	**170.9**	169.8	**234.5**	241	**0.290**	0.297
Mk05	0.3	**1297.7**	**1297.8**	**1483**	1485	**0.125**	**0.126**
	0.6	**1083.6**	**1083.8**	**1261**	1268	**0.141**	0.145
	0.9	**864.8**	**864.9**	**1044**	1052.5	**0.172**	0.178
Mk06	0.3	**247.6**	246.2	**378**	**377.5**	**0.353**	0.348
	0.6	**41.0**	38.5	**170.5**	178	**0.777**	0.784
	0.9	**0.0**	**0.0**	**36**	66	**1.000**	**1.000**
Mk07	0.3	**1224.6**	1206.8	**1445**	1470	**0.156**	0.180
	0.6	**928.2**	901.4	**1135**	1166.5	**0.188**	0.225
	0.9	**668.6**	645.4	**842**	859.5	**0.213**	0.247
Mk08	0.3	**4598.7**	4593.7	**5228**	5252	0.129	0.125
	0.6	**3761.5**	3756.7	**4398**	4412.5	0.152	0.149
	0.9	**2925.3**	2920.3	**3555.5**	3578.5	0.187	0.184
Mk09	0.3	**2926.1**	2925.5	**3705**	3746.5	**0.220**	**0.219**
	0.6	**2096.1**	**2095.4**	**2855.5**	2906	0.282	0.279
	0.9	**1275.3**	**1274.9**	**2032.5**	2075.5	**0.388**	**0.386**
Mk10	0.3	**2144.8**	2143.9	**2891.5**	2928.5	**0.260**	0.268
	0.6	**1374.8**	1373.9	**2114.5**	2163.5	**0.358**	0.365
	0.9	**678.2**	**677.9**	**1332.5**	1384.5	**0.504**	0.510

Tabla 7.10 Comparación de los resultados entre el método del subgradiente estándar (false) y el normalizado (true). Mediana por problema.

	false	true
% Dif CI	0.0%	2.9%
% Dif CS	0.0%	4.2%
% Dif gap	0.5%	7.7%

Tabla 7.11. Diferencias respecto a los valores óptimos en cada problema. Comparación de los resultados entre el método del subgradiente estándar (false) y el normalizado (true).

En cuanto a la convergencia del método, en las Figuras 59 y 60 se observa un mejor comportamiento del método estándar que el método normalizado, tanto en la cota superior como en la cota inferior.

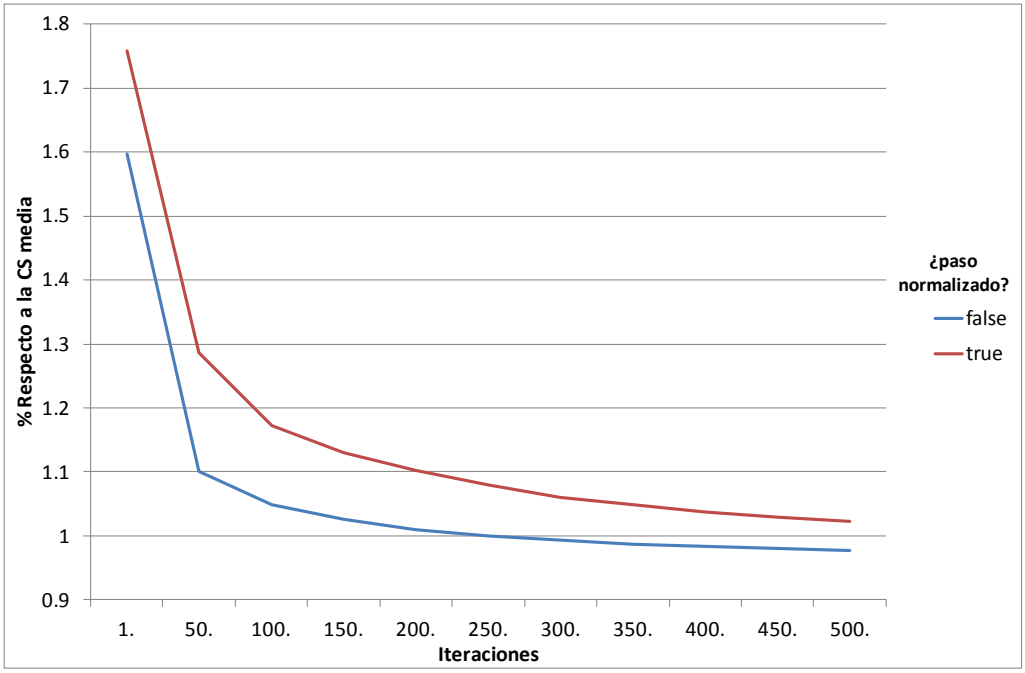

Figura 59. Porcentaje de la cota superior en cada iteración respecto a la cota superior media final. Comparación de los resultados entre el método del subgradiente estándar (false) y el normalizado (true).

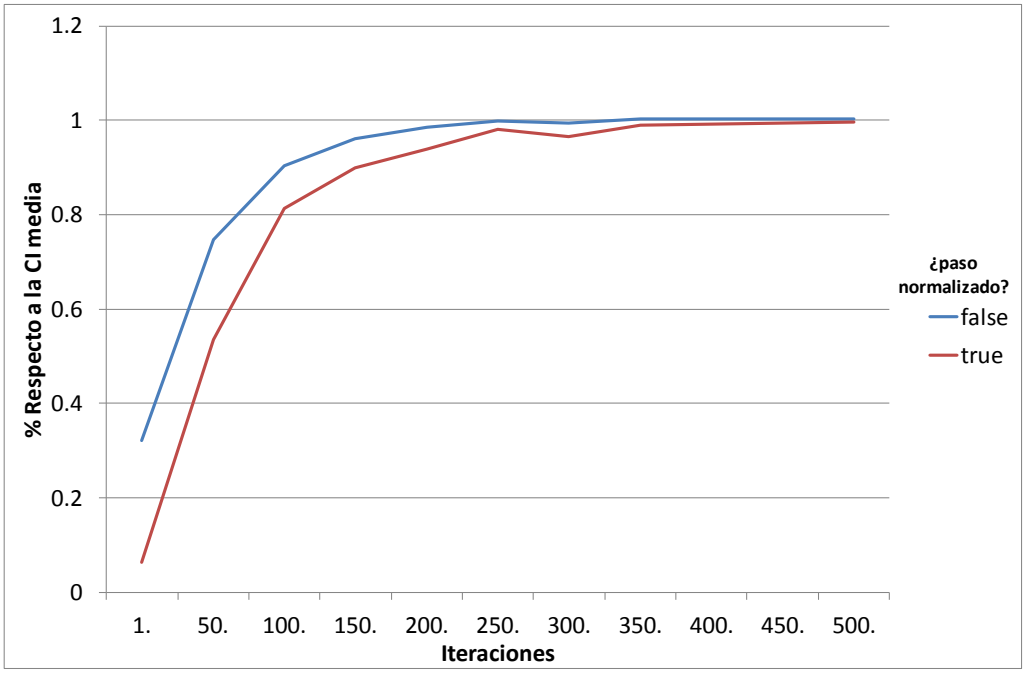

Figura 60. Porcentaje de la cota inferior en cada iteración respecto a la cota inferior media final. Comparación de los resultados entre el método del subgradiente estándar (false) y el normalizado (true).

7.4.4. Peso de la información de iteraciones anteriores en el proceso de actualización de precios

En este apartado estudiamos la variante de actualización de precios mediante el método del gradiente conjugado, en el que la dirección de actualización de los precios tiene en cuenta tanto la información obtenida en la iteración actual, como la obtenida en iteraciones anteriores. Además, se analiza la influencia del factor β en los resultados, siendo β el peso que el subgradiente de la iteración actual tiene en la dirección de actualización. También se estudia el efecto sobre el mejor valor obtenido en la cota superior, variando el valor del peso $\beta \in \{0.1, 0.2, 0.4, 0.5, 0.6, 0.8, 0.9, 1\}$.

Respecto a la cota inferior, los resultados globales estudiados mediante la variable normalizada (Figura 61) muestran la tendencia que luego será confirmada por los datos para cada problema mostrados en la Figura 62 y en la Tabla 7.12. En ellos se muestran que los mejores valores de la cota inferior se obtienen para los valores de β dentro del intervalo de [0.7, 1]. Para los distintos problemas no existen diferencias significativas en este intervalo.

Para la cota superior existe, en este caso, una correspondencia entre los mejores valores y los mejores valores obtenidos en la cota inferior. Esto se muestra en los resultados globales mediante la variable normalizada (Figura 63), así como en los resultados obtenidos en cada problema (Figura 64 y Tabla 7.13). Aquí también se observa que los resultados en la cota superior obtenidos de soluciones de peor calidad de la cota inferior son, asimismo, de peor calidad. Las diferencias relativas obtenidas en el intervalo de β [0.7, 1] son mínimas y son muy importantes para los valores de β menores de 0.5 (Tabla 7.15). Los valores del gap se recogen en la Tabla 7.14 y las Figuras 65 y 66 muestran el gap normalizado.

En cuanto a la convergencia del método respecto al valor de β, las Figuras 67 y 68 muestran que el valor con mejor convergencia, tanto en la cota superior como en la cota inferior, es β=1. Este valor de β se corresponde con el método del subgradiente. Cuando β va aumentando la convergencia va siendo más lenta, aunque, finalmente, entre los valores de β entre 0.7 y 1 el método converge hacia el mismo valor.

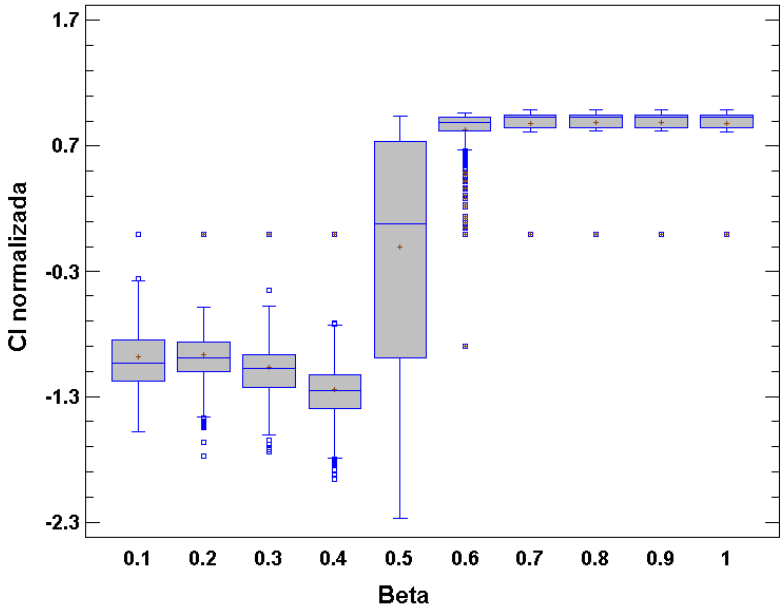

Figura 61. Cota inferior normalizada por el peso del subgradiente en la dirección de actualización.

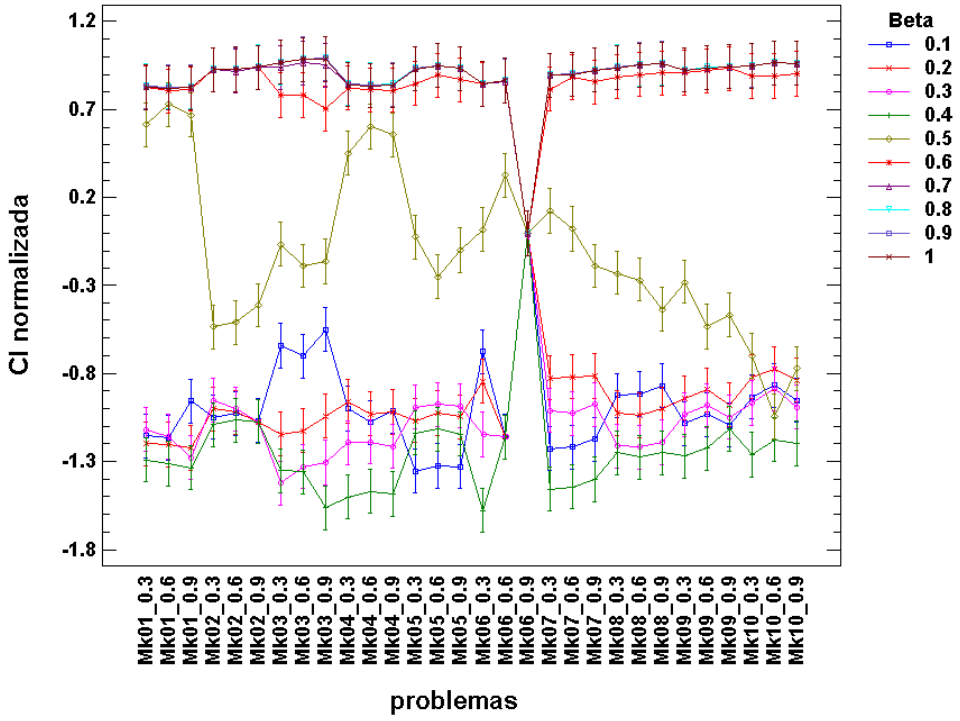

Figura 62. Cota inferior normalizada por el peso del subgradiente en la dirección de actualización y por problema.

código	F	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
Mk01	**0.3**	153.5	152.8	154.2	151.5	181.3	181.7	**181.7**	**181.7**	**181.7**	**181.6**
	0.6	90.3	90.2	90.5	87.8	118.2	118.6	**118.7**	**118.7**	**118.7**	**118.6**
	0.9	38.9	34.9	34.1	34.6	61.3	61.9	**61.9**	**61.9**	**61.9**	**61.9**
Mk02	**0.3**	103.6	103.8	104.9	103.5	102.8	127.8	**127.8**	**127.8**	**127.8**	**127.8**
	0.6	38.0	38.3	38.5	37.8	37.7	62.8	**62.8**	**62.8**	**62.8**	**62.8**
	0.9	0.0	0.0	0.0	0.0	0.0	15.6	**15.6**	**15.6**	**15.6**	**15.6**
Mk03	**0.3**	1045.4	950.8	902.3	921.7	1158.7	1301.4	**1330.0**	**1334.2**	**1334.2**	**1334.1**
	0.6	591.8	515.6	483.5	474.2	687.2	863.3	**892.5**	**895.7**	**895.8**	**895.5**
	0.9	290.8	201.8	152.7	92.5	371.1	531.9	**576.3**	**582.2**	**582.2**	**581.8**
Mk04	**0.3**	342.3	343.5	334.2	322.5	411.2	414.5	**415.2**	**415.2**	**415.1**	**415.0**
	0.6	213.2	216.0	209.6	199.1	285.2	288.4	**289.2**	**289.2**	**289.1**	**289.0**
	0.9	102.9	102.5	95.2	85.3	167.6	170.1	**171.2**	**171.1**	**171.0**	**170.9**
Mk05	**0.3**	808.6	870.5	886.1	857.4	1140.8	1296.8	**1297.7**	**1297.8**	**1297.7**	**1297.7**
	0.6	593.2	656.6	668.1	635.3	815.1	1083.5	**1083.7**	**1083.8**	**1083.7**	**1083.7**
	0.9	376.9	437.8	449.5	416.3	694.1	864.4	**864.9**	**864.9**	**864.9**	**864.8**
Mk06	**0.3**	198.6	192.7	184.0	169.4	247.4	247.5	**247.6**	**247.6**	**247.6**	**247.6**
	0.6	0.0	0.0	0.0	0.0	40.8	40.9	**40.9**	**41.0**	**41.0**	**41.0**
	0.9	0.0	0.0	0.0	0.0	0.0	0.0	**0.0**	**0.0**	**0.0**	**0.0**
Mk07	**0.3**	887.0	947.6	917.9	850.4	1156.6	1223.7	**1224.5**	**1224.6**	**1224.6**	**1224.6**
	0.6	580.8	648.8	610.4	529.5	823.4	927.9	**928.2**	**928.2**	**928.2**	**928.2**
	0.9	327.6	389.5	358.1	303.5	520.8	668.2	**668.6**	**668.6**	**668.6**	**668.6**
Mk08	**0.3**	3581.3	3514.2	3427.3	3395.9	4151.3	4573.9	**4601.4**	**4601.6**	**4600.3**	**4598.7**
	0.6	2747.7	2681.3	2585.0	2546.5	3166.7	3737.6	**3764.4**	**3764.5**	**3763.3**	**3761.8**
	0.9	1924.0	1851.1	1749.4	1711.9	2197.9	2903.1	**2928.0**	**2928.2**	**2926.9**	**2925.4**
Mk09	**0.3**	2267.6	2311.6	2281.9	2210.8	2662.6	2923.9	**2926.7**	**2926.9**	**2926.7**	**2926.3**
	0.6	1437.8	1485.2	1453.9	1375.0	1338.5	2093.6	**2096.7**	**2096.9**	**2096.8**	**2096.2**
	0.9	655.2	683.4	660.0	643.1	753.8	1273.3	**1275.8**	**1276.0**	**1275.9**	**1275.5**
Mk10	**0.3**	1735.6	1756.0	1720.5	1661.8	1746.5	2142.7	**2144.7**	**2144.9**	**2145.0**	**2144.8**
	0.6	963.3	984.9	957.1	898.5	828.9	1372.4	**1374.8**	**1374.9**	**1375.0**	**1374.8**
	0.9	265.7	290.2	255.7	217.6	248.8	675.8	**678.1**	**678.3**	**678.3**	**678.3**

Tabla 7.12. Cota inferior por el peso del subgradiente en la dirección de actualización. Medianas por problema.

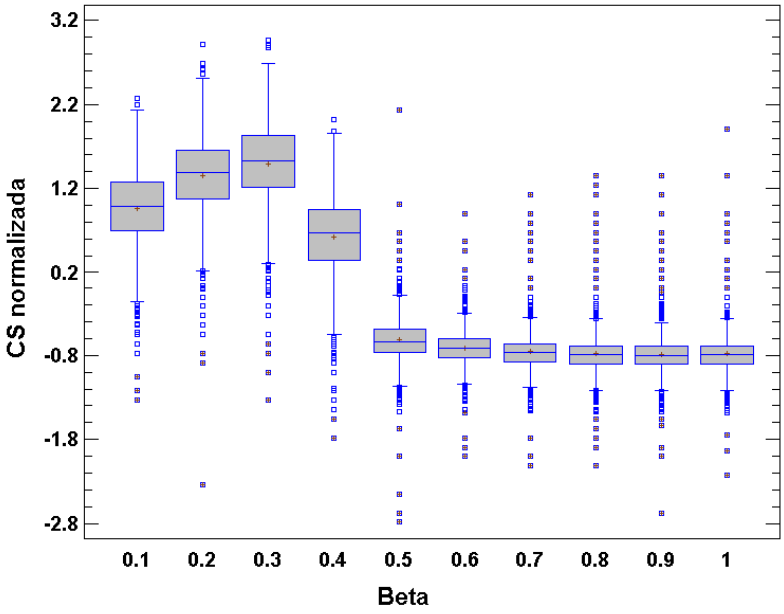

Figura 63. Cota superior normalizada por el peso del subgradiente en la dirección de actualización.

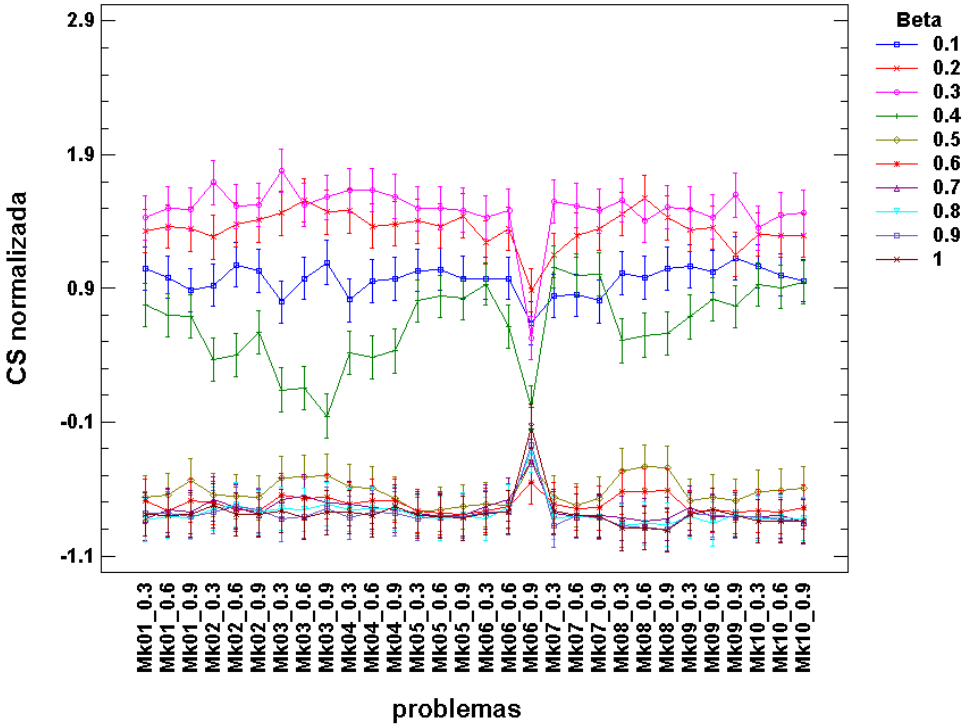

Figura 64. Cota superior normalizada por el peso del subgradiente en la dirección de actualización y por problema.

código	F	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
Mk01	**0.3**	230.5	234	235	226	205	205	**203**	**203**	**204**	**204**
	0.6	166.5	170	174	161.5	142	141	**141**	**140**	**141**	**141**
	0.9	101	107.5	111	100	83	81	**81**	**80**	**80**	**80**
Mk02	**0.3**	179	183	187	173	162	161	**161**	**161**	**161**	**161**
	0.6	116	119.5	120	107	92	91	**92**	**91.5**	**92**	**90.5**
	0.9	53	57	60	47	30	29	**29**	**29**	**29**	**28.5**
Mk03	**0.3**	1627	1685	1700.5	1572	1518.5	1510.5	**1507.5**	**1500**	**1493.5**	**1498.5**
	0.6	1189.5	1227	1233.5	1128.5	1082.5	1072.5	**1072**	**1065**	**1064**	**1064**
	0.9	814	845	862	743.5	709	699	**698.5**	**694**	**694.5**	**692**
Mk04	**0.3**	535.5	558.5	563	522	490.5	487	**486**	**485**	**484**	**484.5**
	0.6	410	423	437	393	364.5	362	**360**	**360.5**	**358**	**357**
	0.9	289.5	304	311.5	270	240	241	**237**	**238**	**237.5**	**238**
Mk05	**0.3**	1777	1844	1853	1745	1486	1485.5	**1482**	**1479**	**1477**	**1480**
	0.6	1565.5	1628	1645	1536.5	1274	1270.5	**1265.5**	**1263**	**1267**	**1266**
	0.9	1329.5	1409	1417.5	1312.5	1058.5	1050	**1046.5**	**1048.5**	**1049**	**1043**
Mk06	**0.3**	433	443.5	450	433	379	376.5	**378.5**	**375.5**	**377**	**377**
	0.6	222	232.5	237.5	210	178	178	**180**	**177.5**	**175**	**178**
	0.9	45	48	45.5	39.5	36	33.5	**36.5**	**37**	**38.5**	**37.5**
Mk07	**0.3**	1663	1707	1770	1693	1454	1450	**1441**	**1439**	**1427.5**	**1443**
	0.6	1376.5	1431.5	1442.5	1383	1159	1160	**1150**	**1150.5**	**1150**	**1153.5**
	0.9	1053.5	1119	1149.5	1076	862.5	853	**846**	**845**	**842.5**	**847**
Mk08	**0.3**	5740.5	5851.5	5850.5	5597.5	5355.5	5308.5	**5257**	**5239.5**	**5250**	**5241.5**
	0.6	4891	5046.5	5001.5	4774	4521	4472.5	**4416**	**4409.5**	**4398.5**	**4406**
	0.9	4067.5	4166	4179.5	3955	3674	3627.5	**3584.5**	**3570**	**3560.5**	**3563**
Mk09	**0.3**	4120.5	4172	4213	4022.5	3743.5	3727.5	**3738.5**	**3726**	**3732**	**3726**
	0.6	3273	3336	3348.5	3228.5	2933	2910	**2898**	**2878**	**2889.5**	**2903**
	0.9	2468	2460	2548	2377	2085	2060.5	**2055.5**	**2060**	**2060.5**	**2055**
Mk10	**0.3**	3358	3411.5	3431.5	3326	2954.5	2921	**2909.5**	**2908**	**2908**	**2896.5**
	0.6	2570.5	2629.5	2685	2547	2184.5	2143	**2145.5**	**2143**	**2137**	**2133**
	0.9	1819	1915.5	1950.5	1807	1416	1382.5	**1352**	**1354**	**1354**	**1348**

Tabla 7.13. Cota superior por el peso del subgradiente en la dirección de actualización. Medianas por problema.

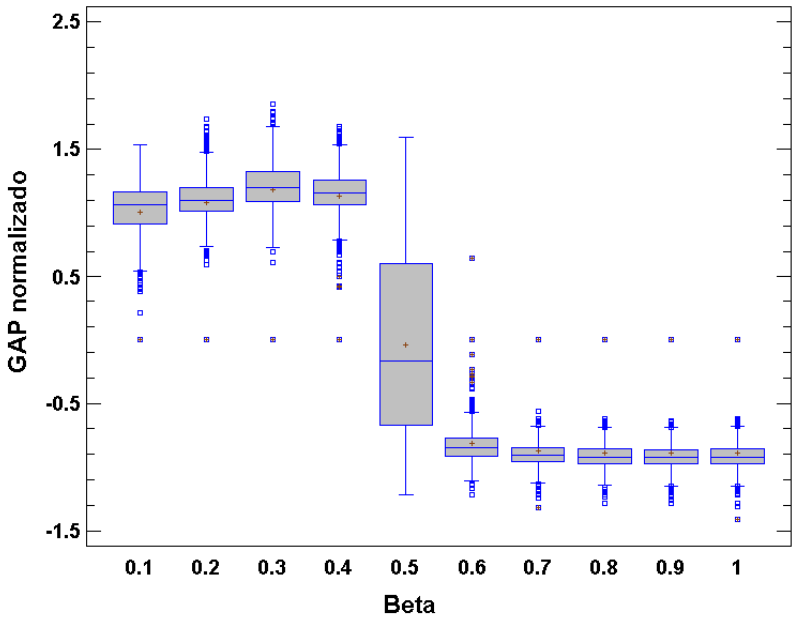

Figura 65. Gap normalizado por el peso del subgradiente en la dirección de actualización.

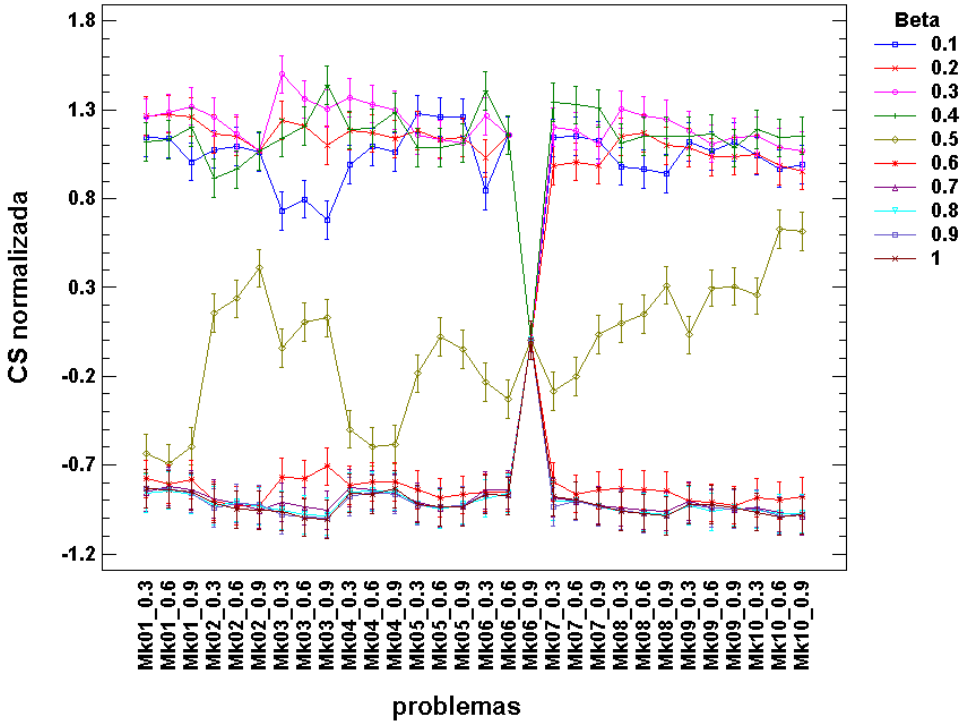

Figura 66. Gap normalizado por el peso del subgradiente en la dirección de actualización y por problema.

código	F	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
Mk01	**0.3**	0.333	0.343	0.346	0.328	0.117	0.114	**0.105**	**0.105**	**0.109**	**0.110**
	0.6	0.454	0.475	0.478	0.453	0.172	0.158	**0.158**	**0.152**	**0.158**	**0.159**
	0.9	0.618	0.668	0.689	0.652	0.266	0.236	**0.235**	**0.226**	**0.226**	**0.227**
Mk02	**0.3**	0.421	0.433	0.441	0.400	0.361	0.206	**0.206**	**0.206**	**0.206**	**0.206**
	0.6	0.666	0.678	0.682	0.642	0.589	0.318	**0.318**	**0.318**	**0.318**	**0.310**
	0.9	1.000	1.000	1.000	1.000	1.000	0.462	**0.462**	**0.462**	**0.462**	**0.453**
Mk03	**0.3**	0.361	0.434	0.473	0.421	0.236	0.139	**0.118**	**0.111**	**0.107**	**0.110**
	0.6	0.502	0.581	0.609	0.579	0.366	0.202	**0.168**	**0.159**	**0.158**	**0.158**
	0.9	0.642	0.760	0.823	0.872	0.484	0.242	**0.175**	**0.163**	**0.162**	**0.159**
Mk04	**0.3**	0.358	0.385	0.408	0.383	0.164	0.149	**0.146**	**0.144**	**0.142**	**0.143**
	0.6	0.482	0.489	0.516	0.493	0.219	0.204	**0.197**	**0.198**	**0.192**	**0.191**
	0.9	0.645	0.659	0.692	0.684	0.299	0.294	**0.278**	**0.281**	**0.280**	**0.282**
Mk05	**0.3**	0.542	0.528	0.524	0.510	0.233	0.130	**0.124**	**0.123**	**0.121**	**0.123**
	0.6	0.620	0.595	0.594	0.581	0.366	0.148	**0.144**	**0.142**	**0.145**	**0.144**
	0.9	0.715	0.689	0.684	0.680	0.331	0.180	**0.173**	**0.175**	**0.176**	**0.171**
Mk06	**0.3**	0.541	0.562	0.592	0.608	0.357	0.343	**0.346**	**0.341**	**0.343**	**0.343**
	0.6	1.000	1.000	1.000	1.000	0.772	0.770	**0.773**	**0.769**	**0.766**	**0.770**
	0.9	1.000	1.000	1.000	1.000	1.000	1.000	**1.000**	**1.000**	**1.000**	**1.000**
Mk07	**0.3**	0.467	0.444	0.482	0.498	0.207	0.160	**0.150**	**0.149**	**0.142**	**0.151**
	0.6	0.569	0.547	0.576	0.616	0.286	0.200	**0.193**	**0.193**	**0.193**	**0.195**
	0.9	0.686	0.653	0.680	0.722	0.393	0.219	**0.210**	**0.209**	**0.206**	**0.211**
Mk08	**0.3**	0.375	0.398	0.418	0.394	0.227	0.139	**0.125**	**0.122**	**0.124**	**0.123**
	0.6	0.438	0.469	0.480	0.465	0.298	0.165	**0.148**	**0.146**	**0.144**	**0.146**
	0.9	0.530	0.555	0.588	0.566	0.397	0.204	**0.183**	**0.180**	**0.178**	**0.179**
Mk09	**0.3**	0.449	0.442	0.458	0.452	0.291	0.217	**0.217**	**0.214**	**0.216**	**0.215**
	0.6	0.560	0.553	0.568	0.575	0.542	0.281	**0.277**	**0.271**	**0.274**	**0.278**
	0.9	0.735	0.724	0.742	0.730	0.641	0.382	**0.379**	**0.381**	**0.381**	**0.379**
Mk10	**0.3**	0.486	0.486	0.494	0.501	0.413	0.269	**0.263**	**0.262**	**0.262**	**0.259**
	0.6	0.622	0.625	0.643	0.646	0.616	0.363	**0.359**	**0.358**	**0.357**	**0.355**
	0.9	0.855	0.850	0.866	0.881	0.822	0.513	**0.499**	**0.499**	**0.499**	**0.497**

Tabla 7.14. Gap superior por el peso del subgradiente en la dirección de actualización. Mediana por problema.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
% Dif CI	37.8%	37.5%	39.5%	42.5%	20.4%	0.7%	0.1%	0.0%	0.0%	0.0%
% Dif CS	16.5%	19.3%	20.3%	13.8%	2.1%	1.0%	0.9%	0.7%	0.7%	0.6%
% Dif gap	57.7%	58.5%	59.6%	59.0%	35.3%	6.8%	2.3%	1.0%	0.9%	1.1%

Tabla 7.15. Diferencias respecto a los valores óptimos en cada problema superior por el peso del subgradiente en la dirección de actualización.

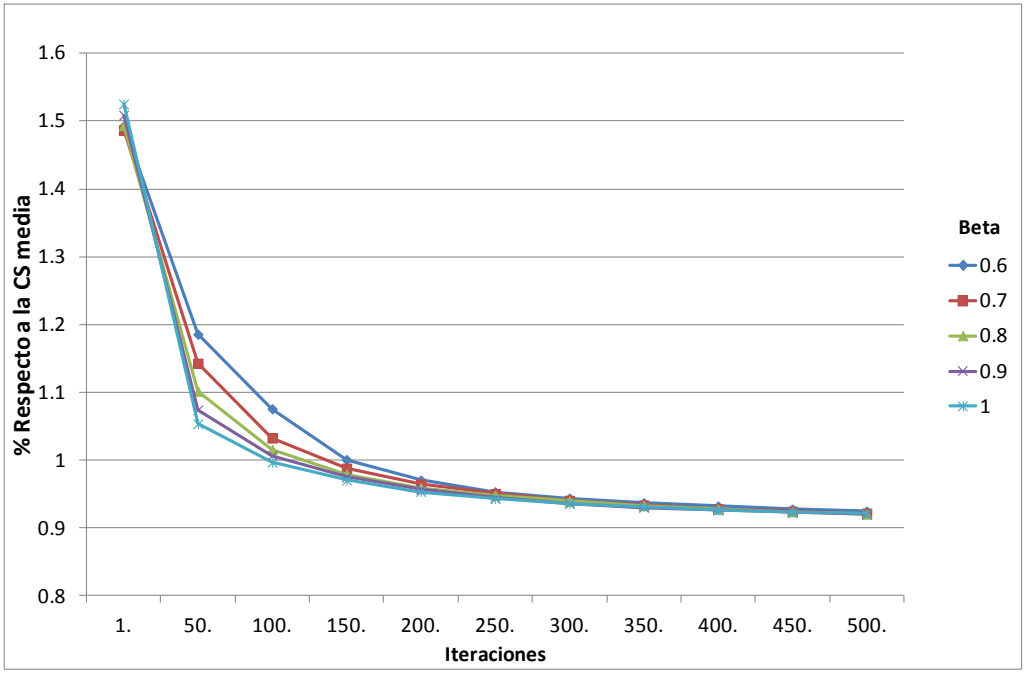

Figura 67. Porcentaje de la cota superior en cada iteración respecto a la cota superior media final por el peso del subgradiente en la dirección de actualización.

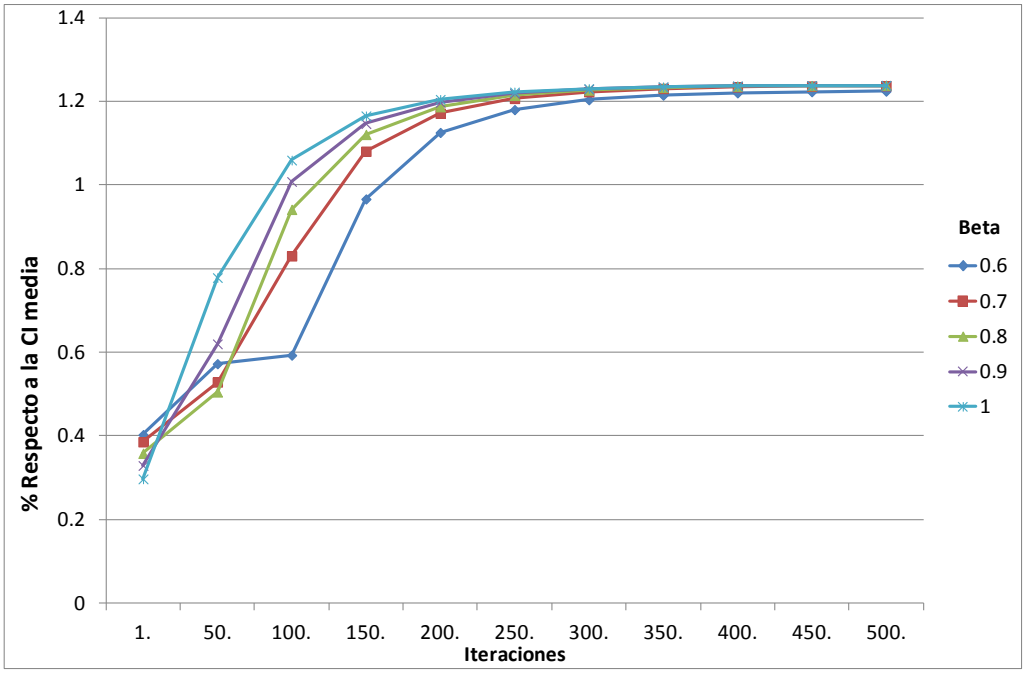

Figura 68. Porcentaje de la cota inferior en cada iteración respecto a la cota inferior media final por el peso del subgradiente en la dirección de actualización.

7.4.5. Propuesta alternativa en el cálculo del tamaño del paso en la actualización de precios

En los siguientes párrafos analizamos la propuesta de Brännlund (1995) para la actualización de precios (apartado 5.3.2). Este autor plantea un método alternativo al cálculo del paso propuesto en Fisher (1981). Bränlunnd propone actualizar tanto el tamaño del paso (α) como el peso de las iteraciones anteriores (β), calculando la dirección de actualización de precios en cada iteración mediante:

Para n=0 $d^0 = g^0$

En otro caso $d^r = \beta^r g^r + (1 - \beta^r) \cdot d^{r-1}$ (7.11)

Con:

$$\alpha^n = \beta^n = \begin{cases} \dfrac{(d^{n-1})^T d^{n-1}}{(d^{n-1})^T d^{n-1} - (g^n)^T d^{n-1}} & si\ (g^n)^T d^{n-1} < 0 \\ 1 & \text{en caso contrario} \end{cases} \quad (7.12)$$

Esta propuesta tiene la ventaja de que no es necesario ajustar la variable (γ) que permite disminuir el tamaño del paso, a medida que se van realizando iteraciones.

No obstante, la posible ventaja que supone no ajustar la variable γ se anula por la calidad de los resultados que produce. La calidad de la solución es inferior a la obtenida por las mejores soluciones del método estándar de ajuste del tamaño del paso, solo comparable a las peores soluciones obtenidas en los valores extremos de frecuencia de actualización del tamaño del paso. Este método puede servir como referencia mínima en entornos nuevos donde los parámetros aún no han sido ajustados.

Los resultados muestran una clara superioridad del método del subgradiente sobre el método de Brännlund, tanto en el cálculo de la cota superior como en el cálculo de la cota inferior (Figuras 69 y 71. Estos resultados son confirmados cuando se muestran los valores por problema tanto en la cota inferior (Figura 70), como la cota superior (Figura 72). Los valores del gap no hacen sino repetir este mismo comportamiento (Figuras 73 y 74). En la Tabla 7.16 se muestran los datos para cada problema. Se ha destacado en negrita la opción con la que se consigue el valor óptimo.

Para comprobar si existen diferencias significativas para estos valores se ha utilizado la U de Mann-Whitney para comparar medianas. Para todos los problemas el método del subgradiente básico muestra claramente un mejor comportamiento.

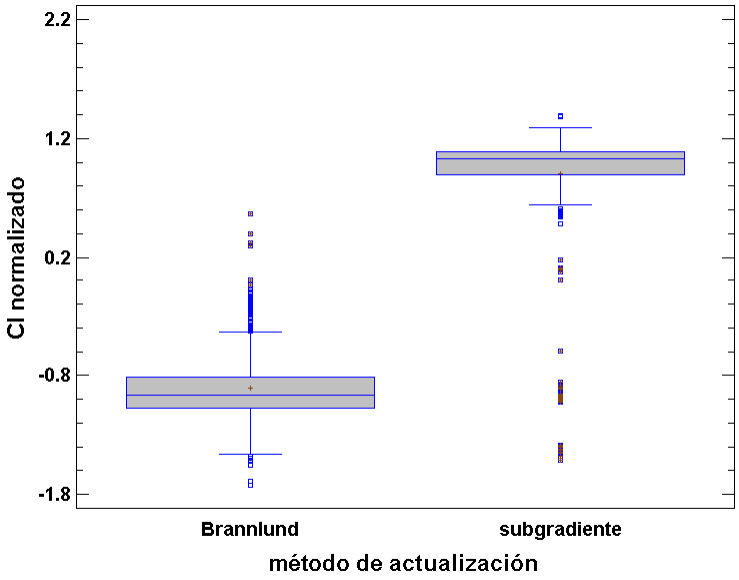

Figura 69. Cota inferior normalizada. Comparación entre el método del subgradiente estándar y el mejorado de Brännlund.

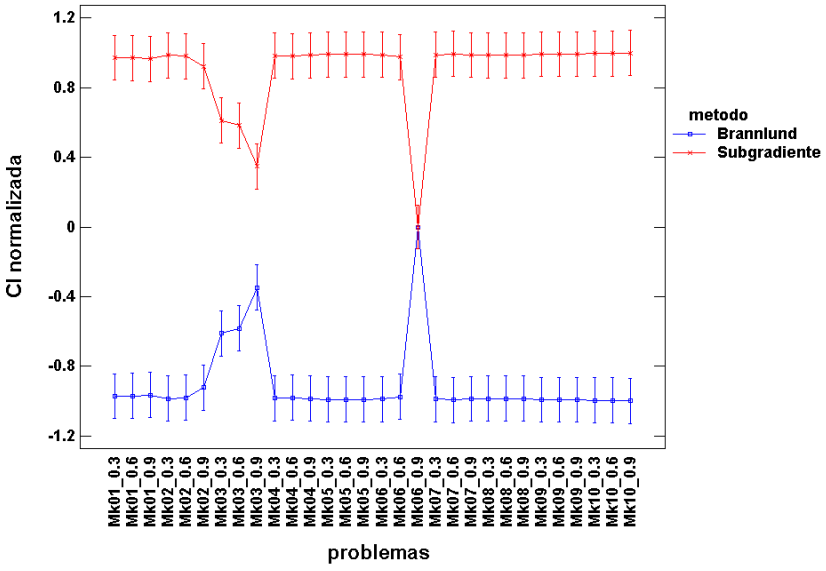

Figura 70. Cota inferior normalizada. Comparación entre el método del subgradiente estándar y el mejorado de Brännlund por problema.

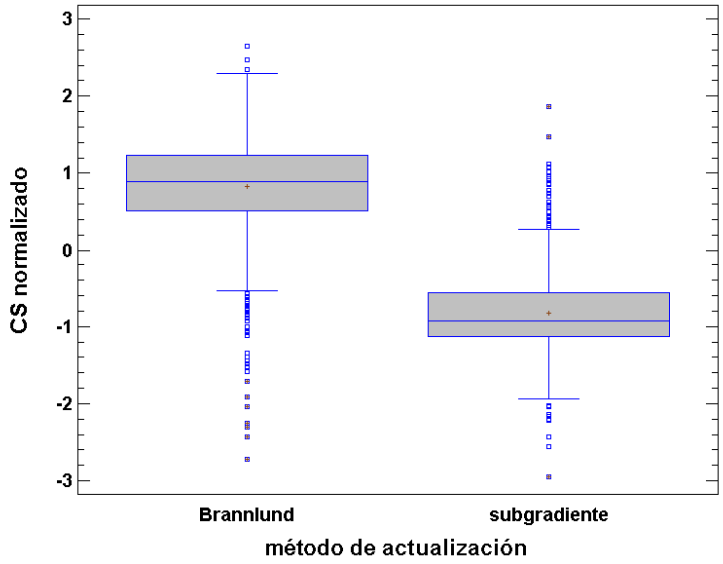

Figura 71. Cota superior normalizada. Comparación entre el método del subgradiente estándar y el mejorado de Brännlund.

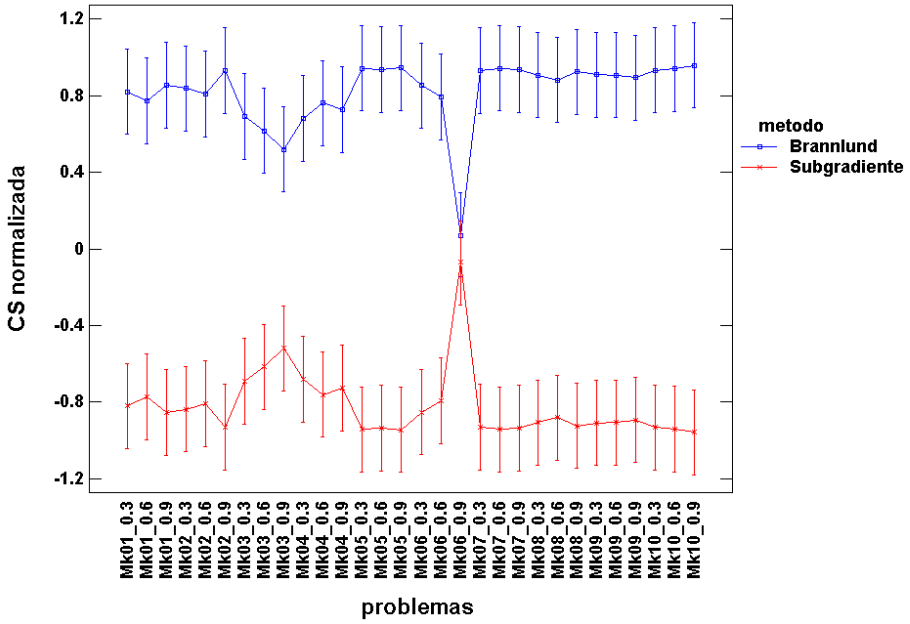

Figura 72. Cota superior normalizada. Comparación entre el método del subgradiente estándar y el mejorado de Brännlund por problema.

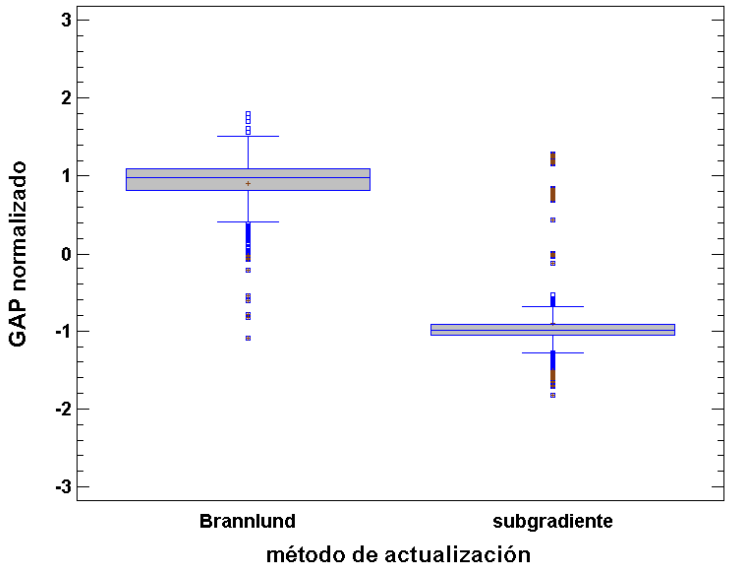

Figura 73. Gap normalizado. Comparación entre el método del subgradiente estándar y el mejorado de Brännlund.

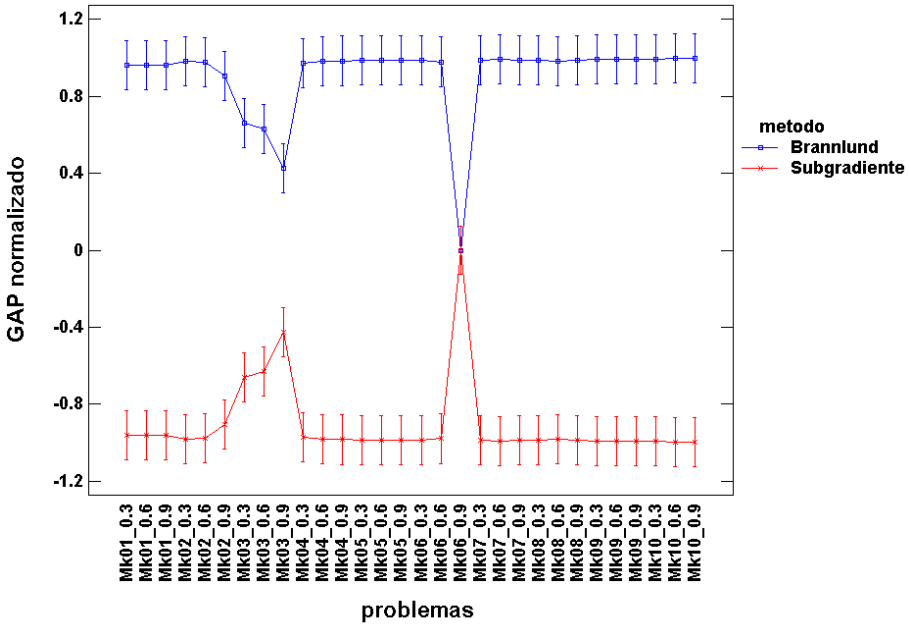

Figura 74. Gap normalizado. Comparación entre el método del subgradiente estándar y el mejorado de Brännlund por problema.

También es destacable el hecho de que las soluciones del problema obtenidas a partir de soluciones del problema relajado con menor cota inferior, tienen como

resultado soluciones del problema de peor calidad, aunque este fenómeno se amortigua en el proceso de transformación de soluciones. Así, en la cota inferior la diferencia media entre ambos métodos es de un 36,8%, mientras que en la cota superior la diferencia es de un 7,6% (Tabla 7.17).

		CI prom		CS prom		GAP prom	
código	F	Brannnlund	subgradient	Brannnlund	subgradient	Brannnlund	subgradient
Mk01	0.3	155.7	**181.6**	214	**201**	0.271	**0.114**
	0.6	92.2	**118.6**	149	**140**	0.381	**0.170**
	0.9	39.2	**61.9**	89	**78**	0.559	**0.231**
Mk02	0.3	90.7	**127.8**	169	**160.5**	0.460	**0.215**
	0.6	25.9	**62.8**	102	**91**	0.747	**0.332**
	0.9	0.0	**15.6**	44	**30**	1.000	**0.480**
Mk03	0.3	1151.2	**1334.1**	1539.5	**1509**	0.252	**0.121**
	0.6	714.6	**895.5**	1101	**1064.5**	0.349	**0.170**
	0.9	451.7	**580.4**	714	**677**	0.369	**0.181**
Mk04	0.3	344.9	**414.9**	496	**481**	0.306	**0.153**
	0.6	289.0	**289.0**	356	**356**	0.205	**0.205**
	0.9	103.5	**170.9**	246.5	**234.5**	0.580	**0.290**
Mk05	0.3	1058.0	**1297.7**	1572	**1483**	0.328	**0.125**
	0.6	849.9	**1083.6**	1355	**1261**	0.376	**0.141**
	0.9	630.3	**864.8**	1142	**1044**	0.453	**0.172**
Mk06	0.3	119.9	**247.6**	408	**378**	0.707	**0.353**
	0.6	0.0	**41.0**	197	**170.5**	1.000	**0.777**
	0.9	0.0	**0.0**	37	**36**	1.000	**1.000**
Mk07	0.3	933.0	**1224.6**	1549	**1445**	0.398	**0.156**
	0.6	647.2	**928.2**	1250	**1135**	0.482	**0.188**
	0.9	414.6	**668.6**	944.5	**842**	0.560	**0.213**
Mk08	0.3	3949.8	**4598.7**	5390	**5228**	0.266	**0.129**
	0.6	3104.0	**3761.5**	4549	**4398**	0.321	**0.152**
	0.9	2284.5	**2925.3**	3709	**3555.5**	0.383	**0.187**
Mk09	0.3	2130.7	**2926.1**	3883	**3705**	0.450	**0.220**
	0.6	1297.2	**2096.1**	3048	**2855.5**	0.575	**0.282**
	0.9	474.6	**1275.3**	2203.5	**2032.5**	0.784	**0.388**
Mk10	0.3	1301.7	**2144.8**	3146	**2891.5**	0.586	**0.260**
	0.6	526.2	**1374.8**	2370.5	**2114.5**	0.777	**0.358**
	0.9	0.0	**678.2**	1612	**1332.5**	1.000	**0.504**

Tabla 7.16. Resultados por problema. Comparación entre el método del subgradiente estándar y el mejorado de Bränlund.

	Brannnlund	subgradient
% Dif CI	36.8%	0.0%
% Dif CS	7.6%	0.0%
% Dif gap	49.9%	0.0%

Tabla 7.17. Diferencias medias respecto a los valores óptimos en cada problema. Comparación entre el método del subgradiente estándar y el mejorado de Bränlund.

En cuanto a la convergencia, en las Figuras 75 y 76 se observa un mejor comportamiento del método estándar que el del método normalizado, tanto en la cota superior como en la cota inferior.

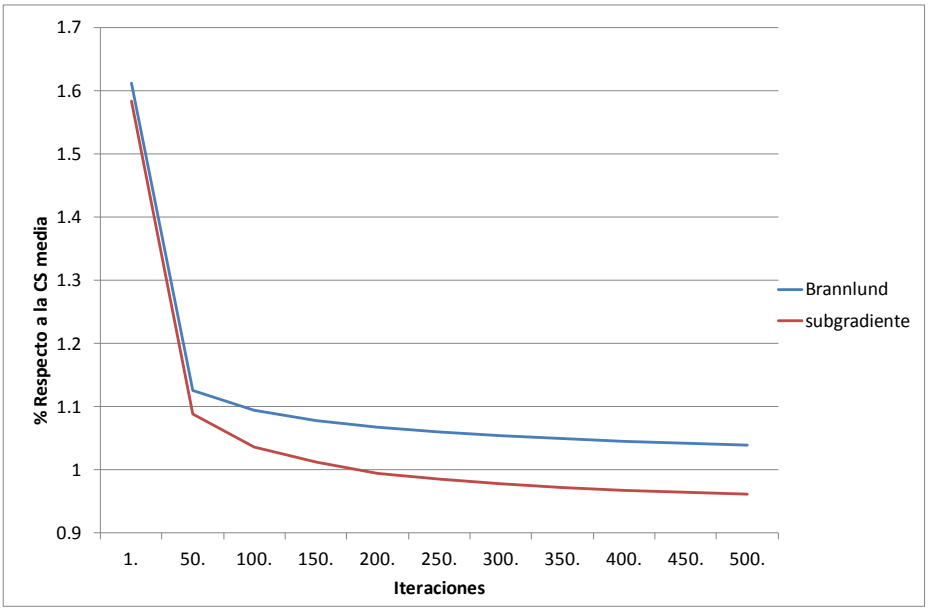

Figura 75. Porcentaje de la cota superior en cada iteración respecto a la cota superior media final. Comparación entre el método del subgradiente estándar y el mejorado de Brännlund.

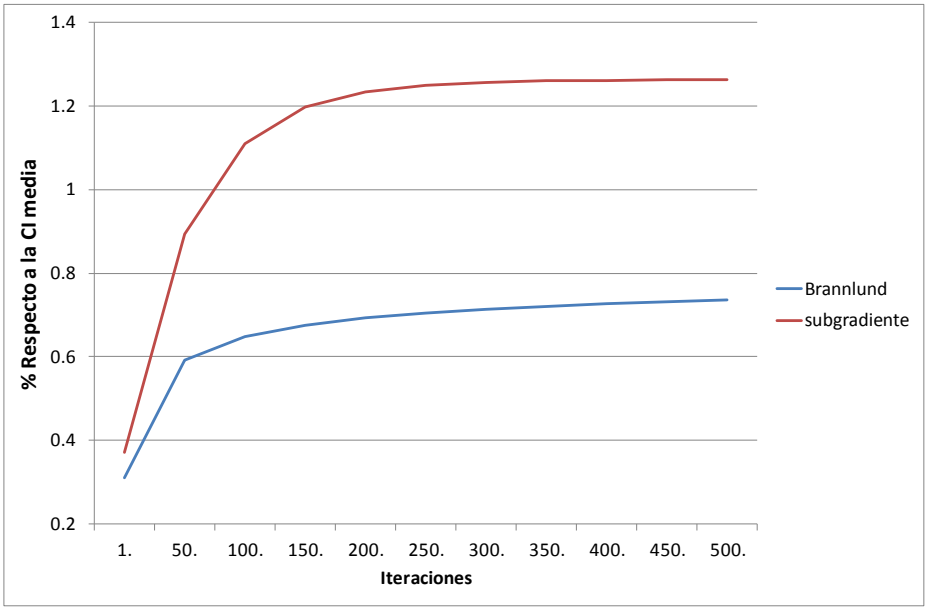

Figura 76. Porcentaje de la cota inferior en cada iteración respecto a la cota inferior media final. Comparación entre el método del subgradiente estándar y el mejorado de Brännlund.

7.4.6. Métodos que no garantizan la cota inferior

En los fundamentos teóricos de este trabajo se ha presentado el método del gradiente surrogado (apartado 5.3.3). Este método es una alternativa al método del subgradiente para resolver el dual del problema relajado, y también puede ser utilizado como método de actualización de precios en una subasta combinatoria iterativa. La ventaja de este método reside en que no es necesario calcular en cada iteración el valor óptimo del problema relajado. Aplicado al problema de programación de talleres flexibles, con este método no es necesario que en cada iteración todas las órdenes calculen el óptimo del subproblema asociado. Lo que se exige es que se mejore el valor de la función Lagrangiana respecto a la solución obtenida en la iteración anterior.

La aplicación de este método a problemas separables proporciona las variantes del método intercalado de Zhao et al (1997) y de Kaskavelis y Caramanis (1998) [apartado 6.2.2]. Estas variantes también pueden interpretarse como una subasta en la que los precios se actualizan cuando se recibe una propuesta por parte de una orden.

En el experimento se propone comparar el método básico del subgradiente intercalado propuesto por Zhao et al (1997), el método de subgradiente intercalado propuesto por Kaskavelis y Caramanis (1998), y el método del subgradiente. No se estudia el valor de la cota inferior puesto que estos métodos no garantizan su existencia, ni el valor del gap puesto que depende de la cota inferior.

Los resultados (Figuras 77 y 78, y Tabla 7.18) nos muestran que, tanto con el método del subgradiente como con el del subgradiente intercalado de Zhao et al (1997), se obtienen resultados equivalentes en cuanto a la solución del problema. En muchos de los problemas no existen diferencias significativas y ninguno de los dos métodos funciona mejor que el otro para todos los problemas. Las diferencias medias relativas obtenidas respecto al mejor valor en cada problema son cercanas a cero. Respecto al método de Kaskavelis y Caramanis (1998), los resultados obtenidos son peores que los de los otros dos métodos, dando peores resultados en cada uno de los problemas estudiados. El valor de la diferencia media relativa respecto al mejor valor es de un 9.8%, como se observa en la Tabla 7.19.

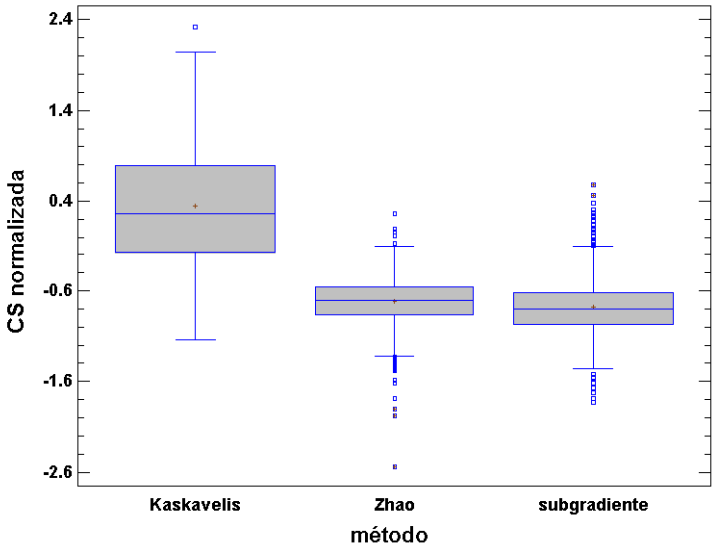

Figura 77. Cota superior normalizada. Comparación entre el método del subgradiente estándar y las versiones del método intercalado.

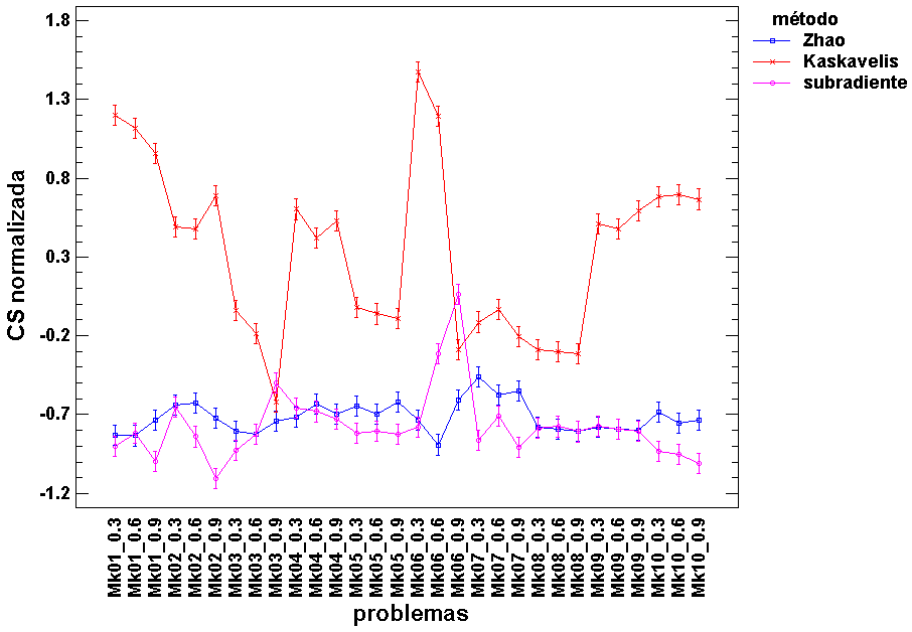

Figura 78. Cota superior normalizada. Comparación entre el método del subgradiente estándar y las versiones del método intercalado por problema.

código	F	Intercalado	Int. Kaskv.	subgradiente
Mk01	0.3	**204**	223	**204**
	0.6	**142**	159	**142**
	0.9	83	96	**80**
Mk02	0.3	**161**	168	**161**
	0.6	94	100	**92**
	0.9	32	42	**30**
Mk03	0.3	1506.5	1552.5	**1498.5**
	0.6	1066.5	1104.5	**1063**
	0.9	**676**	684	692.5
Mk04	0.3	**485**	512.5	**486**
	0.6	**361.5**	383.5	**361**
	0.9	240	263	**238**
Mk05	0.3	1498	1553	**1481**
	0.6	1279	1342.5	**1270.5**
	0.9	1065	1120.5	**1046**
Mk06	0.3	382	438.5	**380**
	0.6	**168**	216.5	182
	0.9	**17**	28	41
Mk07	0.3	1482.5	1514	**1438.5**
	0.6	1170.5	1234.5	**1160.5**
	0.9	877.5	918.5	**840.5**
Mk08	0.3	**5247.5**	5386.5	**5247.5**
	0.6	**4416.5**	4541	**4417**
	0.9	3572.5	3707	**3566**
Mk09	0.3	**3736**	3997.5	**3739.5**
	0.6	2900	3151	**2893**
	0.9	2065	2336.5	**2057**
Mk10	0.3	2947	3216.5	**2901.5**
	0.6	2176	2457.5	**2143.5**
	0.9	1412	1670	**1364**

Tabla 7.18. Cota superior. Comparación entre el método del subgradiente estándar y las versiones del método intercalado por problema. Mediana por problema.

	Intercalado	Int. Kaskv.	subgradiente
% Dif CS	1.1%	9.8%	2.3%

Tabla 7.19. Diferencias respecto a los valores óptimos en cada problema. Comparación entre el método del subgradiente estándar y las versiones del método intercalado por problema.

7.4.7. Comparación del método del subgradiente con el método walrasiano no adaptativo

En este experimento se compara el método del subgradiente con el método walrasiano no adaptativo. Como se indicó en el apartado 4.4.1, en el método walrasiano no adaptativo la función de actualización de precios $f(D_{\mathrm{hk}}^{r})$ se define como una constante (s) por el valor del exceso de demanda en un determinado *slot k* de la máquina *h*, (D_{hk}^{r}):

$$f(D_{\mathrm{hk}}^{r}) = s \cdot D_{\mathrm{hk}}^{r} \tag{7.13}$$

Este método tiene como principal ventaja su simplicidad y el hecho de que su implementación en sistemas distribuidos no necesita información global. El principal inconveniente es que el factor de cálculo del paso permanece constante, independientemente de cómo vayan evolucionando la demanda en función de los precios, y la convergencia del método puede ser muy lenta.

Se ha aplicado el método walrasiano no adaptativo a la batería de problemas utilizada en los experimentos anteriores y se ha comparado con el método del subgradiente. En el método walrasiano no adaptativo se han dado distintos valores al factor de determinación del paso *s*. Los valores son $s \in \{0.01; 0.1; 0.5; 1; 2\}$. Los resultados obtenidos (Figura 79) muestran que, en cuanto a la solución del problema, el método del subgradiente muestra un comportamiento similar al mejor encontrado mediante el método walrasiano no adaptativo. Estos se dan para los valores más bajos de s, en este caso *s*=0.01 y *s*=0.1. Los valores de la solución del problema van empeorando para valores de *s* mayores.

Estos mismos resultados se confirman cuando se analiza cada uno de los problemas (Figura 80 y Tabla 7.20). En cuanto a las diferencias medias obtenidas respecto al mejor valor en cada problema (Tabla 7.21), el mejor valor se obtiene con el método del subgradiente seguido por el valor más bajo de *s* estudiado 0.01. Como se ha mencionado antes, conforme crece el valor de *s* los valores obtenidos en la solución del problema son peores.

En cuanto a la convergencia del método, observando la evolución de las soluciones encontrada a lo largo de las iteraciones (Figura 81), se puede concluir que el método del subgradiente es el que tiene una convergencia más rápida. En cuanto al método walrasiano no adaptativo la mejor convergencia se consigue para el valor de *s*=0.1, cuya convergencia es similar a la del método del subgradiente. A pesar de que con el valor de *s*=0.01 se obtienen mejores soluciones, sin embargo su convergencia es más

lenta. Es interesante observar que, atendiendo a la convergencia del método, el patrón seguido en la solución del problema también se sigue en la cota inferior del problema (Figura 82).

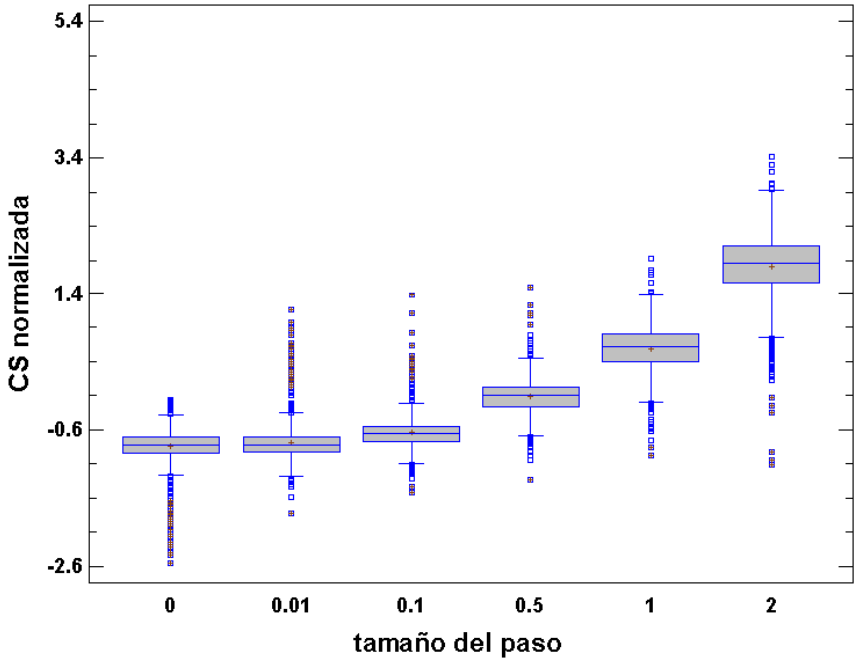

Figura 79. Cota superior normalizada. Comparación entre el método del subgradiente estándar y el método walrasiano no adaptativo.

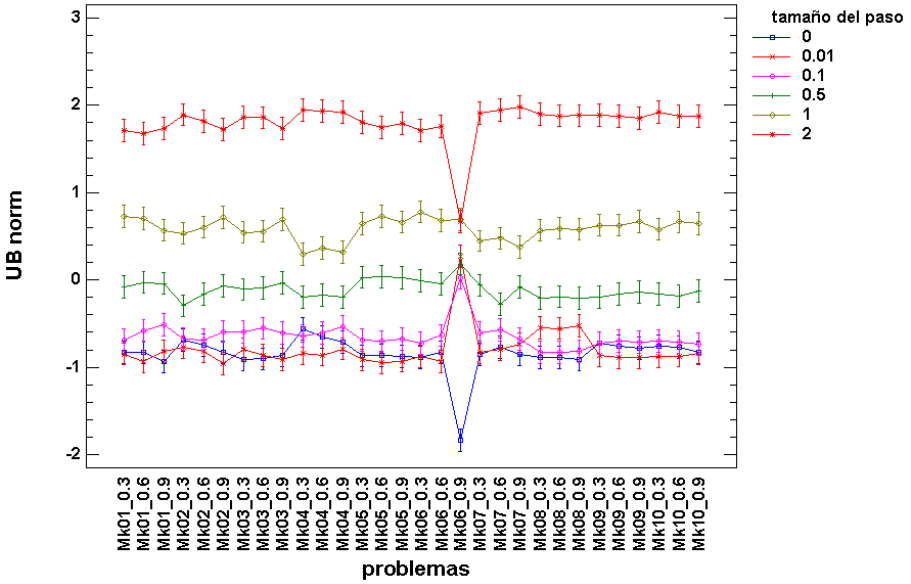

Figura 80. Cota superior normalizada. Comparación entre el método del subgradiente estándar y el método walrasiano no adaptativo por problemas.

select-problem	F	subgradiente	0.01	0.1	0.5	1	2
"Mk01.txt"	0.3	**204**	**204.5**	**206**	212.5	221	232
	0.6	**141.5**	**140.5**	144	150	157	167.5
	0.9	**80.5**	**82**	85	90	95	105
"Mk02.txt"	0.3	**162**	**159.5**	**161**	168	183	206
	0.6	**92**	**91**	**93**	102	117	139.5
	0.9	**30**	**27**	34.5	46	61	80.5
"Mk03.txt"	0.3	**1498.5**	1522	1553.5	1628.5	1730.5	1938
	0.6	**1065.5**	**1071**	1126	1199	1302.5	1514.5
	0.9	**694.5**	**687**	732.5	823.5	928.5	1080.5
"Mk04.txt"	0.3	**489**	**481**	**486**	497	510	549
	0.6	359	**354**	360.5	371	384	425.5
	0.9	**237**	**237**	242	249	262	297
"Mk05.txt"	0.3	1480	**1474.5**	1497	1566	1622	1730.5
	0.6	1272.5	**1263**	1286	1351	1413	1505.5
	0.9	1048	**1042**	1067	1127.5	1184.5	1299
"Mk06.txt"	0.3	**379**	**378.5**	386	422.5	464	511
	0.6	177.5	**174**	189	216	248	300.5
	0.9	**39.5**	108	99.5	105.5	121	122.5
"Mk07.txt"	0.3	**1437**	**1440**	1458	1529	1600	1754
	0.6	**1149**	**1147**	1160	1196	1287.5	1444.5
	0.9	**848**	858	869	936	985.5	1163
"Mk08.txt"	0.3	**5248**	5316.5	5259.5	5375.5	5529	5776.5
	0.6	**4413.5**	4469.5	4421	4544	4690	4934.5
	0.9	**3567**	3646.5	3589.5	3705	3860	4098
"Mk09.txt"	0.3	3736.5	**3695**	3733.5	3889	4123	4486
	0.6	2897.5	**2861**	2909.5	3072.5	3290	3645
	0.9	2047	**2010.5**	2070.5	2239	2484.5	2830.5
"Mk10.txt"	0.3	2916.5	**2872.5**	2934	3132.5	3398	3917
	0.6	2145	**2104.5**	2160	2347.5	2666.5	3121.5
	0.9	**1358.5**	**1354.5**	1392	1614.5	1908	2338.5

Tabla 7.20. Cota superior. Comparación entre el método del subgradiente estándar y el método walrasiano no adaptativo por problemas. Mediana por problema.

	subgradiente	0.01	0.1	0.5	1	2
% Dif CS	1.0%	2.5%	4.9%	10.8%	17.1%	25.6%

Tabla 7.21. Diferencias medias respecto a los valores óptimos en cada problema. Comparación entre el método del subgradiente estándar y el método walrasiano no adaptativo.

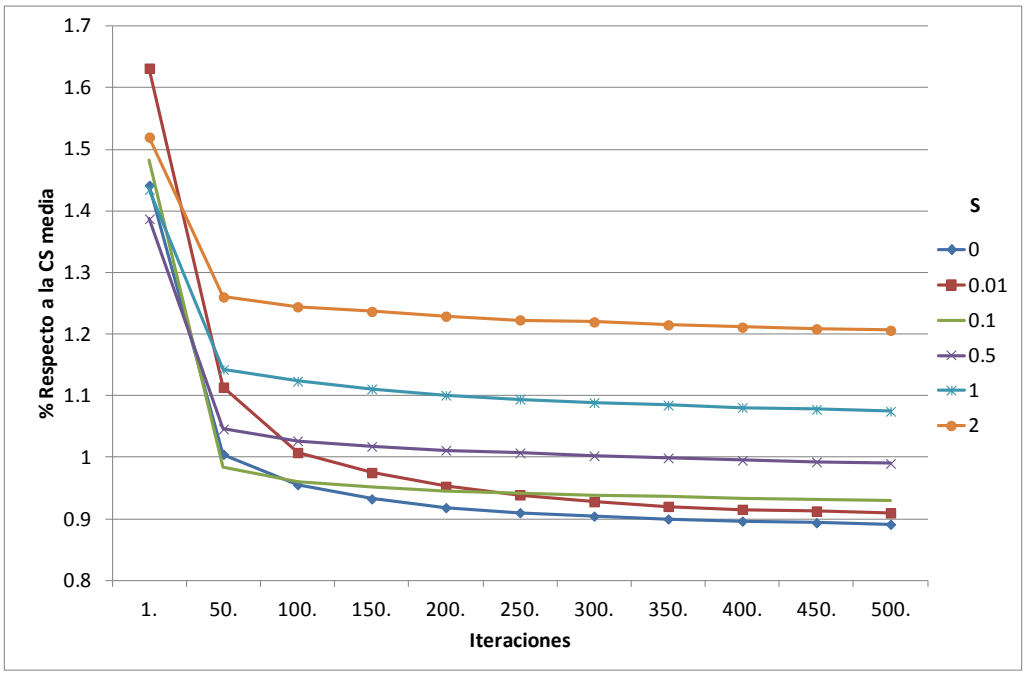

Figura 81. Porcentaje de la cota superior en cada iteración respecto a la cota superior media final. Comparación entre el método del subgradiente estándar y el método walrasiano no adaptativo.

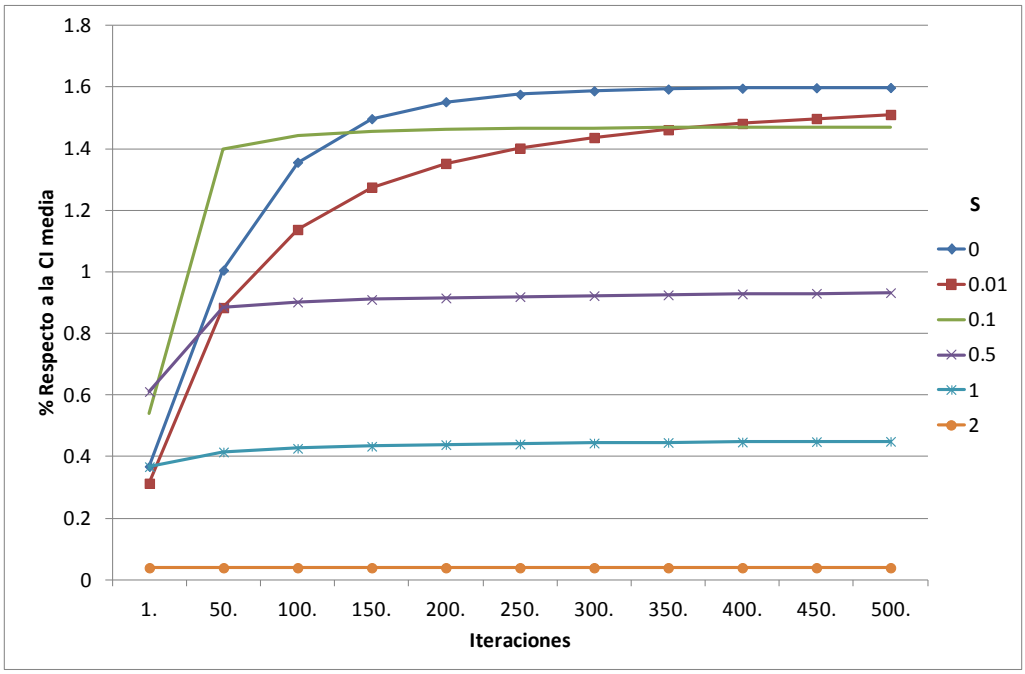

Figura 82. Porcentaje de la cota inferior en cada iteración respecto a la cota inferior media final. Comparación entre el método del subgradiente estándar y el método walrasiano no adaptativo.

7.4.8. Procedimientos asíncronos

Un aspecto que se quería estudiar en este trabajo es la dependencia de la sincronización del método. Para ello se ha construido una variante del método intercalado en la que las órdenes de trabajo envían con igual probabilidad los programas locales, actualizándose los precios de forma similar al método del gradiente intercalado. Este método se caracteriza porque permite reproducir un sistema asíncrono en el que las órdenes reciben los precios en el momento en que entregan su programa local optimizado. Este programa sirve de base para recalcular los precios a la siguiente orden, elegida al azar, que entregue un nuevo programa optimizado (método Kkv-rdm). Este mismo mecanismo, modificando la actualización del tamaño del paso, se ha utilizado con el mecanismo walrasiano no adaptativo (método Kkv-K-step). El hecho de que exista un mecanismo que tenga un buen comportamiento en modo asíncrono, supone una ventaja en su implementación en sistemas descentralizados, puesto que implica que no es necesario que exista un elemento de coordinación central que garantice el funcionamiento del sistema.

Se han comparado ambos métodos -Kkv-rdm y Kkv-K-step- con el método del subgradiente sobre la batería de problemas. Los resultados obtenidos son de interés, puesto que la versión en la que los precios se actualizan mediante el mecanismo walrasiano no adaptativo (método Kkv-K-step) tienen un comportamiento similar al del método del subgradiente, obtenido resultados equivalentes. Sin embargo, al utilizar el método de actualización de precios heredado del método del gradiente intercalado (método Kkv-rdm) los resultados son significativamente peores. Esto se muestra tanto en los resultados globales de la cota superior normalizada (Figura 83), como en los resultados de cada problema (Figura 84 y Tabla 7.22).

La Tabla 7.23 muestra las diferencias medias de cada opción respecto al mejor valor de cada problema y confirman los mismos resultados.

El buen comportamiento del procedimiento asíncrono con el mecanismo walrasiano no adaptativo supone una ventaja adicional para este sistema, ya que hace que sea un método a tener en cuenta cuando se buscan sistemas complemente descentralizados.

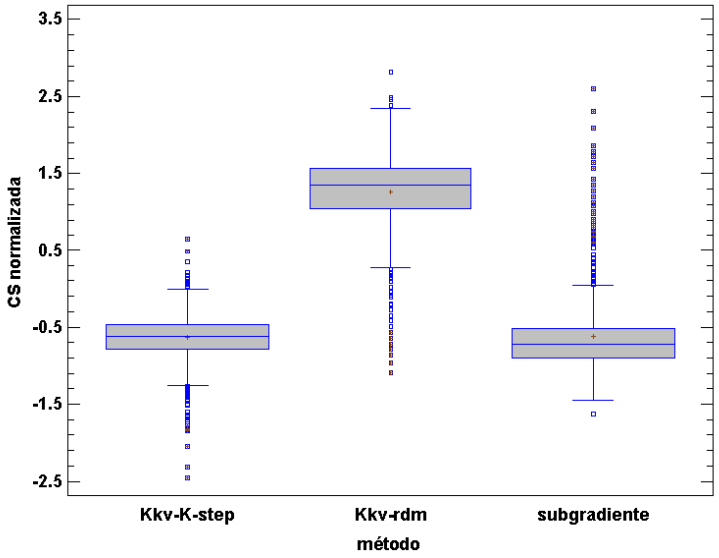

Figura 83. Cota superior normalizada. Comparación entre el método del subgradiente estándar y los métodos intercalados asíncronos.

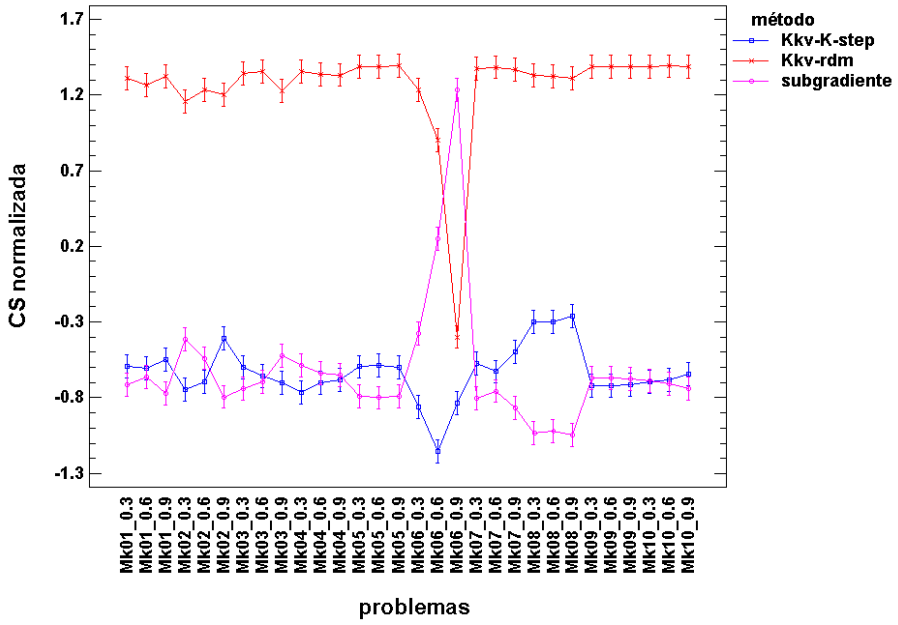

Figura 84. Cota superior normalizada. Comparación entre el método del subgradiente estándar y los métodos intercalados asíncronos por problema.

	F	subgr.	Kkv-K-step	Kkv-rdm
"Mk01.txt"	0.3	**204**	**205**	225
	0.6	**142**	**142.5**	160
	0.9	**80**	83	101
"Mk02.txt"	0.3	**161**	**159**	171
	0.6	**92**	**91.5**	104
	0.9	**30**	**32**	42
"Mk03.txt"	0.3	**1498.5**	1505	1630
	0.6	**1063**	**1066**	1195.5
	0.9	692.5	**684.5**	761.5
"Mk04.txt"	0.3	486	**482**	537
	0.6	361	**357**	414.5
	0.9	**238**	**237.5**	283.5
"Mk05.txt"	0.3	**1481**	1510	1767
	0.6	1270.5	1297.5	1554
	0.9	**1046**	1074.5	1333.5
"Mk06.txt"	0.3	380	**372**	411
	0.6	182	**160**	189.5
	0.9	41	**13**	18
"Mk07.txt"	0.3	**1438.5**	1467	1759
	0.6	**1160.5**	1177	1455
	0.9	**840.5**	888.5	1151
"Mk08.txt"	0.3	**5247.5**	5486	6024.5
	0.6	**4417**	4646	5184.5
	0.9	**3566**	3831	4333
"Mk09.txt"	0.3	3739.5	**3719.5**	4234
	0.6	2893	**2887.5**	3435
	0.9	2057	**2050.5**	2575
"Mk10.txt"	0.3	**2901.5**	2904	3419
	0.6	**2143.5**	**2149**	2637
	0.9	**1364**	1379	1857.5

Tabla 7.22. Cota superior. Comparación entre el método del subgradiente estándar y los métodos intercalados asíncronos por problema. Mediana por problema

	Intercalado	Int. Kaskv.	subgradiente
% Dif CS	0.46%	7.32%	0.00%

Tabla 7.23. Diferencias respecto a los valores óptimos en cada problema. Comparación entre el método del subgradiente estándar y los métodos intercalados asíncronos por problema.

7.5. Análisis de los métodos de construcción de programas factibles

Presentamos, en este apartado, el conjunto de experimentos que tienen como objetivo estudiar el efecto de los distintos algoritmos para la construcción de programas factibles, a partir de los problemas propuestos por las órdenes. Este apartado se centra en el procedimiento que permite convertir los programas locales propuestos por cada una de las órdenes de trabajo, que en principio no cumplirán las restricciones del problema, en un programa global que sí las cumpla. Se valoran distintas opciones utilizando métodos constructivos.

7.5.1. Comparación de los métodos de construcción de programas factibles

En este punto se estudia la influencia que tiene el método de construcción de programas factibles utilizado en la calidad de la solución obtenida. Estos métodos permiten obtener una solución que cumpla con las restricciones del problema a partir de las soluciones de los subproblemas relacionados con las órdenes de trabajo. En el apartado 6.4 se han revisado las características de estos métodos. Los métodos a estudiar son:

- El método de lanzamiento de tareas sin retraso (dptch).
- El método constructivo de Giffer-Thompon o con retraso (G-Th).
- El algoritmo voraz de construcción de programas factibles (Hoitomt et al 1993) (G).

De los resultados obtenidos se deduce que la utilización de un método u otro prácticamente no tiene influencia sobre el valor de la cota inferior (Figura 85). Los resultados por problema (Figura 86 y Tabla 7.24) muestran resultados similares para los tres métodos y la media de las diferencias relativas respecto al mejor valor es cercana a cero. Además, los datos obtenidos por problema son muy homogéneos, con un coeficiente de variación para cada problema inferior al 1%.

En cuanto la cota superior las diferencias son mucho más acusadas (Figuras 87 y 88). Los valores del gap se recogen en las Figuras 89 y 90.

El método que mejores resultados obtiene es el método de lanzamiento de tareas, seguido del método de Giffer-Thompson y del algoritmo voraz de construcción de programas factibles. Las diferencias medias de estos dos últimos métodos respecto al mejor valor obtenido son significativas y son respectivamente de 16,7% y 28,7%, respectivamente.

En los método de construcción de programas factibles existen tres fases principales. La primera es la carga o asignación de las tareas a máquinas. La segunda, es la secuenciación de las tareas en la que se determina el orden en que se van a realizar las tareas en cada máquina. Por último, la tercera es la temporización, que determina el momento de inicio y de fin planificado para cada tarea en las máquinas. Cabe indicar en este caso que los dos primeros métodos –el método de lanzamiento de tareas sin retraso y el método de Giffer- Thompson- utilizan la asignación de máquinas a tareas definida en el programa propuesto por las órdenes de trabajo, mientras que en el algoritmo voraz de Hoitomt esta asignación puede ser modificada en el proceso de construcción del programa factible.

Los resultados señalan que la asignación propuesta por la resolución del problema relajado es adecuada y tiene un mejor comportamiento que su modificación posterior.

La convergencia de la cota inferior se ve afectada por la calidad de la solución en la cota superior. Con el método del lanzamiento sin retrasos se obtienen mejores valores que con los otros dos métodos y la convergencia en la cota inferior es también mejor como se observa en la Figura 92. Sin embargo, aunque en la cota superior se obtienen distintos valores con cada método, la convergencia es equivalente, como se muestra en la Figura 91.

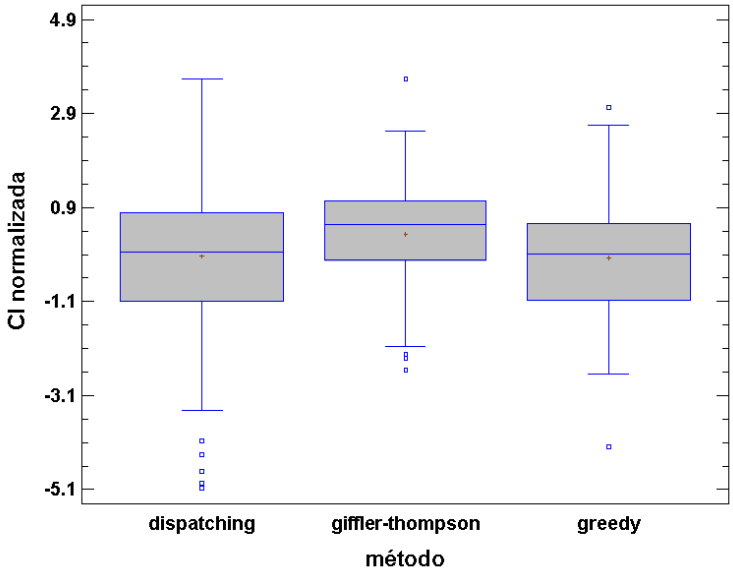

Figura 85. Cota inferior normalizada. Comparación entre los métodos de construcción de programas factibles.

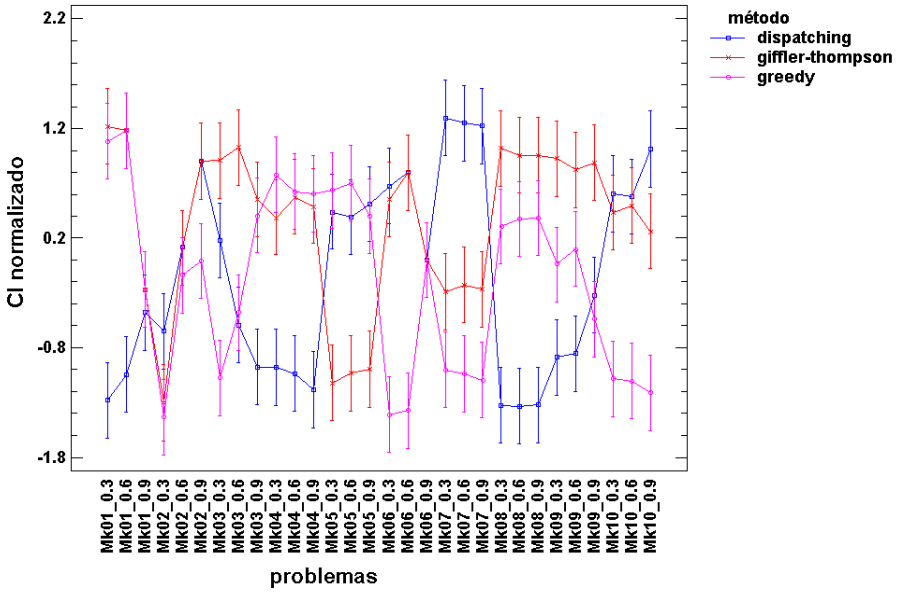

Figura 86. Cota inferior normalizada. Comparación entre los métodos de construcción de programas factibles por problema.

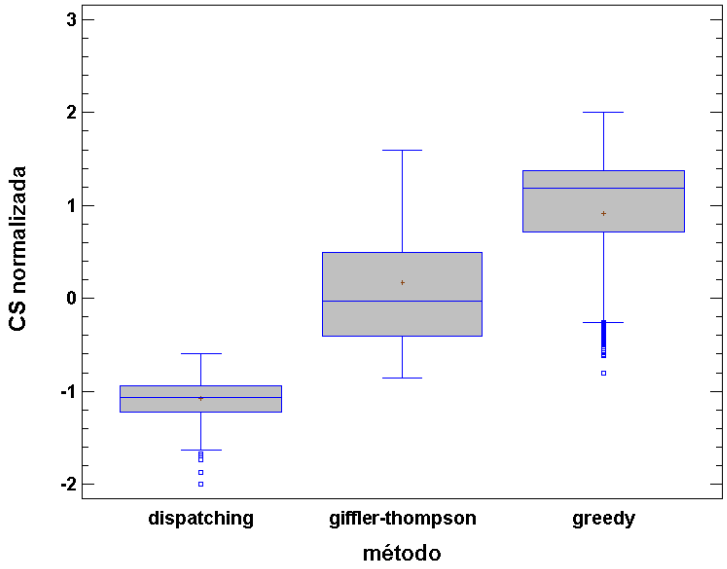

Figura 87. Cota superior normalizada. Comparación entre los métodos de construcción de programas factibles.

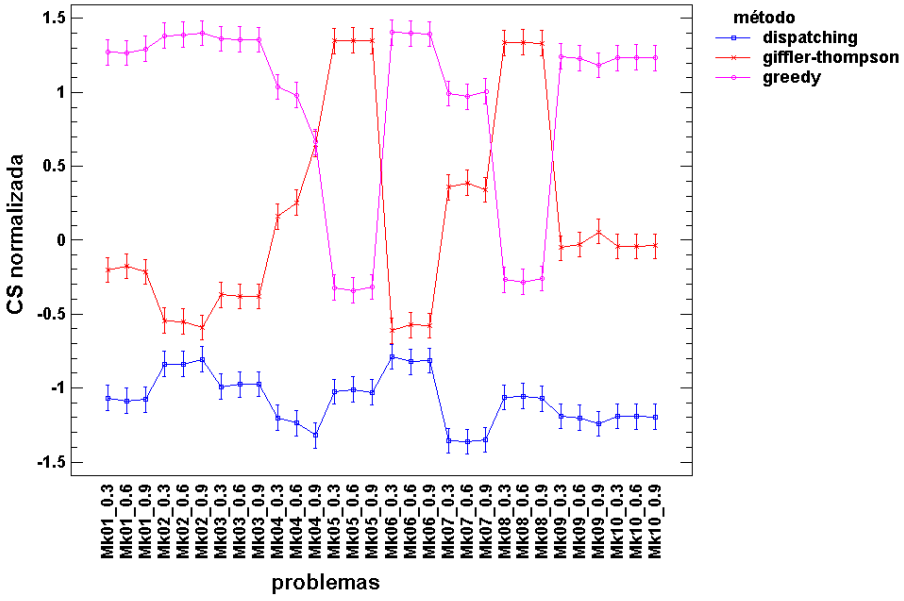

Figura 88. Cota superior normalizada. Comparación entre los métodos de construcción de programas factibles por problema.

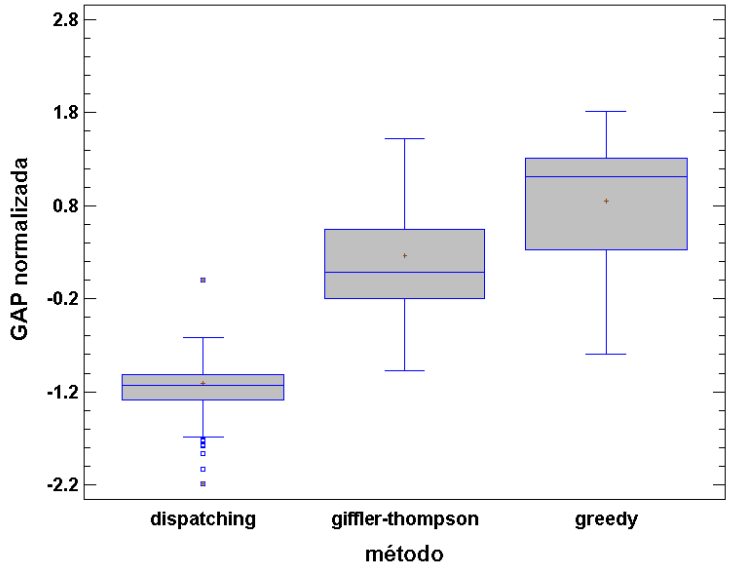

Figura 89. Gap normalizado. Comparación entre los métodos de construcción de programas factibles.

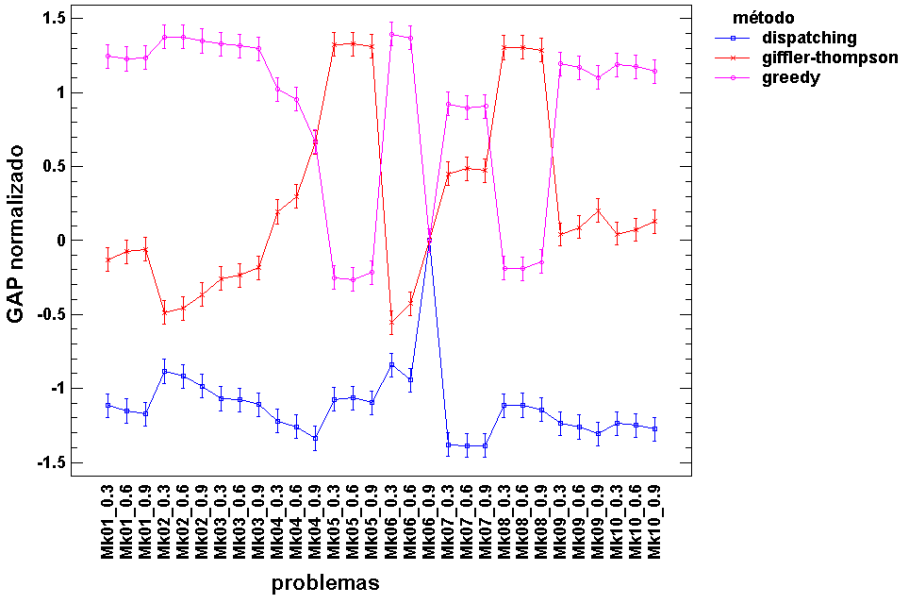

Figura 90. Gap normalizado. Comparación entre los métodos de construcción de programas factibles por problema.

		CI			CS			GAP		
código	F	DISPCH	G-TH	GR	DISPCH	G-TH	GR	DISPCH	G-TH	GR
Mk01	0.3	181.6	181.7	181.7	**204**	221	252	**0.110**	0.178	0.279
	0.6	118.6	118.7	118.7	**142**	158	186	**0.164**	0.249	0.362
	0.9	61.9	61.9	61.9	**79**	94	120	**0.217**	0.341	0.484
Mk02	0.3	127.8	127.8	127.8	**161**	169	223	**0.206**	0.244	0.427
	0.6	62.8	62.8	62.7	**92**	100	153	**0.318**	0.372	0.594
	0.9	15.6	15.6	15.6	**29**	36	100	**0.462**	0.567	0.844
Mk03	0.3	1334.1	1334.2	1334.0	**1496**	1669	2145	**0.108**	0.201	0.378
	0.6	895.6	895.8	895.6	**1063**	1222	1692	**0.157**	0.267	0.471
	0.9	581.3	582.3	582.2	**695**	851	1292	**0.166**	0.316	0.550
Mk04	0.3	415.0	415.1	415.1	**487**	518	541	**0.148**	0.199	0.233
	0.6	289.0	289.1	289.1	**359**	396	415	**0.195**	0.270	0.304
	0.9	170.9	171.0	171.0	**237**	284	284	**0.279**	0.398	0.398
Mk05	0.3	1297.7	1297.6	1297.7	**1479**	1833	1587	**0.123**	0.292	0.182
	0.6	1083.7	1083.6	1083.7	**1266**	1620	1368	**0.144**	0.331	0.208
	0.9	864.9	864.7	864.8	**1046**	1401	1153	**0.173**	0.383	0.250
Mk06	0.3	247.6	247.6	247.4	**376**	398	637	**0.341**	0.378	0.612
	0.6	41.0	41.0	40.8	**177**	204	411	**0.768**	0.799	0.901
	0.9	0.0	0.0	0.0	**39**	57	205	**1.000**	**1.000**	**1.000**
Mk07	0.3	1224.6	1224.3	1224.1	**1437**	1871	2044	**0.148**	0.346	0.401
	0.6	928.2	928.0	927.8	**1148**	1574	1725	**0.191**	0.410	0.462
	0.9	668.6	668.3	668.2	**848**	1262	1425	**0.212**	0.470	0.531
Mk08	0.3	4598.8	4601.1	4600.4	**5249**	6592	5699	**0.124**	0.302	0.193
	0.6	3761.7	3764.0	3763.4	**4412**	5759	4839	**0.147**	0.346	0.222
	0.9	2925.5	2927.6	2927.1	**3560**	4915	4030	**0.178**	0.404	0.274
Mk09	0.3	2926.2	2926.5	2926.3	**3729**	4190	4717	**0.215**	0.302	0.380
	0.6	2096.2	2096.5	2096.3	**2894**	3370	3895	**0.276**	0.378	0.462
	0.9	1275.4	1275.6	1275.4	**2047**	2563	3002	**0.377**	0.502	0.575
Mk10	0.3	2144.8	2144.8	2144.6	**2921**	3253	3642	**0.266**	0.341	0.411
	0.6	1374.8	1374.8	1374.6	**2139**	2482	2868	**0.357**	0.446	0.521
	0.9	678.3	678.1	677.9	**1358**	1696	2081	**0.501**	0.600	0.674

Tabla 7.24. Comparación entre los métodos de construcción de programas factibles. Medianas por problema.

	DISPCH	G-TH	GR
% Dif CI	0.0%	0.0%	0.0%
% Dif CS	0.0%	16.7%	28.7%
% Dif gap	0.0%	34.5%	42.4%

Tabla 7.25. Diferencias respecto a los valores óptimos en cada. Comparación entre los métodos de construcción de programas factibles.

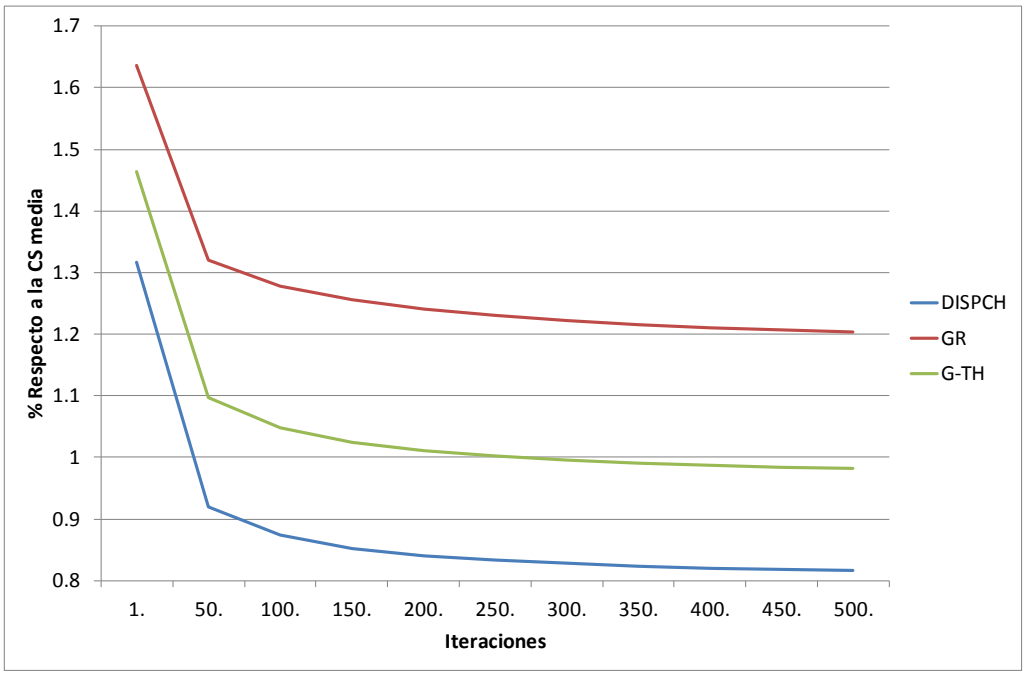

Figura 91. Porcentaje de la cota superior en cada iteración respecto a la cota superior media final. Comparación entre los métodos de construcción de programas factibles.

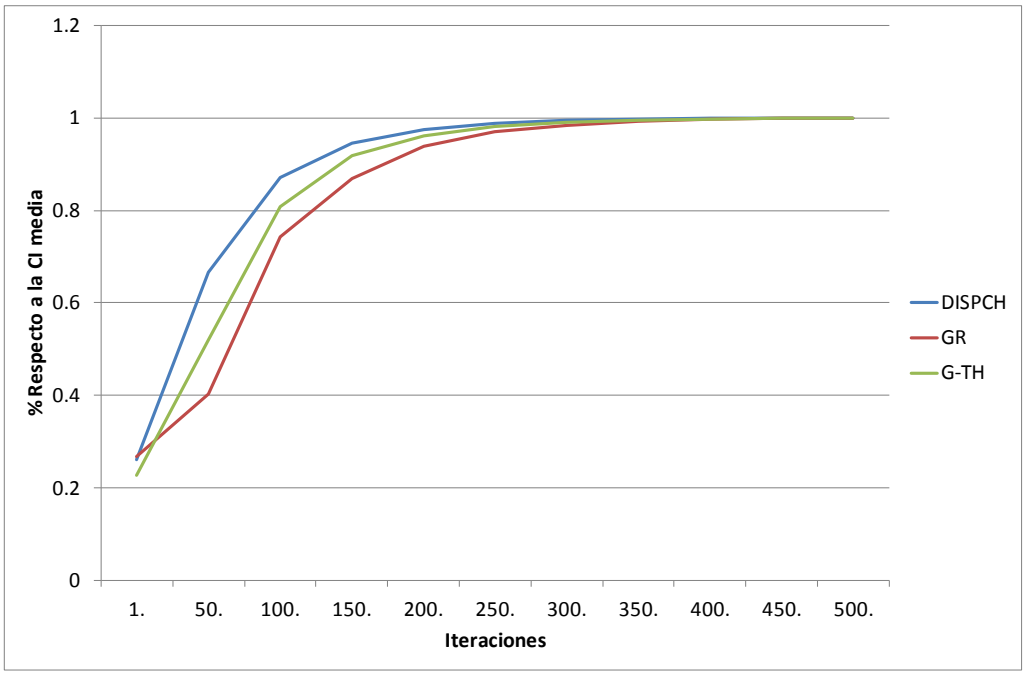

Figura 92. Porcentaje de la cota inferior en cada iteración respecto a la cota inferior media final. Comparación entre los métodos de construcción de programas factibles.

7.5.2. Reglas de prioridad primarias

En el algoritmo utilizado, la solución propuesta por las órdenes es transformada por los métodos constructivos en una solución factible. Po ello, es interesante analizar qué información de los programas propuestos se utiliza a la hora de crear la nueva solución. Los diversos trabajos que implementan esta metodología han utilizado el tiempo de comienzo de la tarea en el programa propuesto, así como el tiempo de fin de la tarea en el programa. El tiempo de la tarea del mejor programa obtenido utiliza como base el programa relajado que haya proporcionado una mejor cota inferior hasta el momento. Estos criterios se implementan sobre el método de lanzamiento de tareas sin retrasos. Asimismo, se estudiará qué efecto produce utilizar información del mejor programa propuesto obtenido hasta el momento, en lugar de obtener la información del programa actual. En las tablas de resultados se utiliza la siguiente notación:

- Tiempo de comienzo de la tarea (TSL).
- Tiempo de fin de la tarea (TFL).
- Tiempo de la tarea del mejor programa obtenido (BTSL).

Los resultados muestran que la cota inferior no presenta diferencias significativas respecto a la utilización de los distintos criterios primarios utilizados en la construcción de programas factibles. Esto se confirma para cada uno de los problemas estudiados, donde no existe ningún criterio con el que se obtengan mejores resultados. Las diferencias medias relativas respecto al mejor valor en cada problema son cercanas a cero (Figuras 93, 94, 95, 96, 97 y 98, y Tabla 7.26).

Respecto a la solución del problema, los resultados muestran diferencias significativas para los distintos criterios. Los peores resultados se obtienen cuando se utiliza la información del mejor programa propuesto obtenido hasta el momento. En este caso, siempre se obtienen resultados significativamente peores para cada uno de los problemas estudiados. Esto es debido a que se reduce el número de soluciones buscadas. Es posible ver el método de Relajación Lagrangiana como un método de búsqueda orientada por las soluciones del problema relajado. Sin embargo, una solución del problema relajado que proporcione una mejor cota inferior no garantiza la obtención de una mejor solución del problema original al ser transformado en una solución factible. Por el contrario, la utilización de los tiempos de finalización y de inicio del programa propuesto produce resultados similares en muchos de los problemas. Existe un grupo importante de problemas donde el criterio que mejores resultados proporciona es el del tiempo de finalización de la tarea. Considerando las

diferencias medias relativas respecto al mejor valor en cada problema, la utilización del tiempo de finalización de la tarea del programa propuesto produce mejores resultados con una pequeña diferencia respecto al tiempo de comienzo de la tarea (2,4%) y una diferencia importante respecto a la información del mejor programa obtenido hasta el momento (29,4%), como muestra la Tabla 7.27.

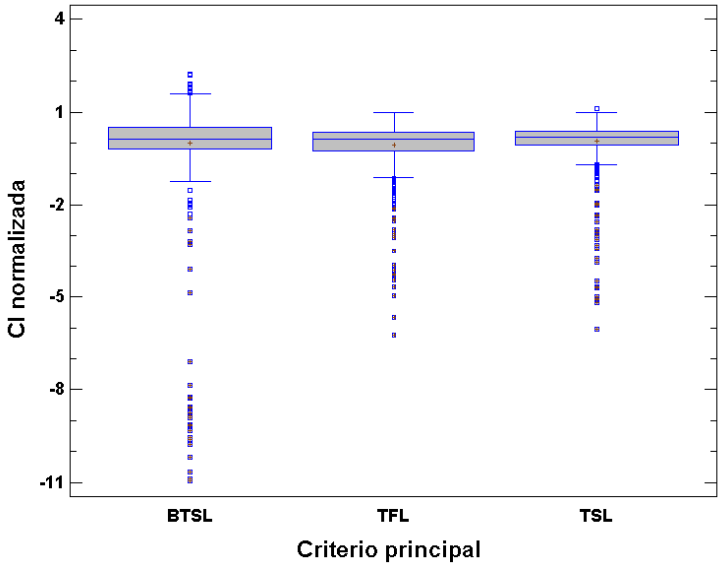

Figura 93. Cota inferior normalizada. Comparación del criterio principal utilizado en la construcción de programas factibles.

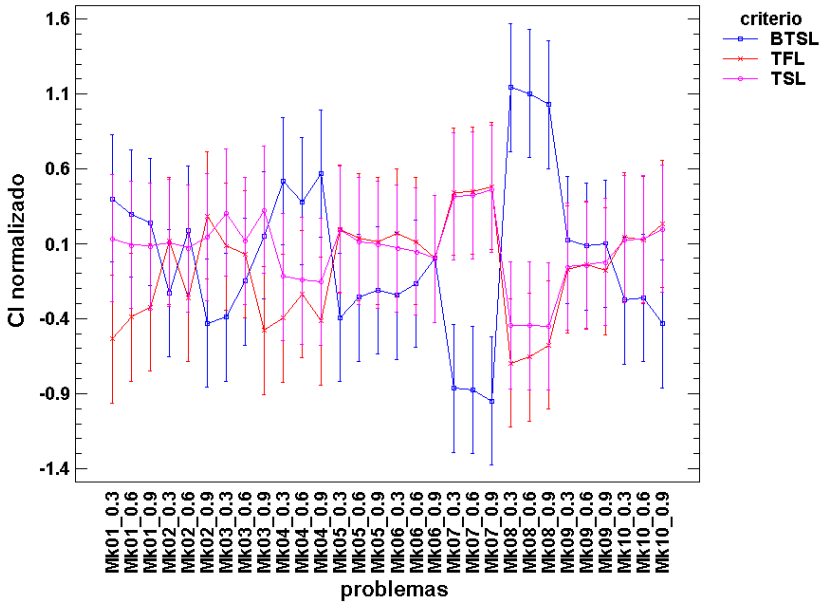

Figura 94. Cota inferior normalizada. Comparación del criterio principal utilizado en la construcción de programas factibles por problema.

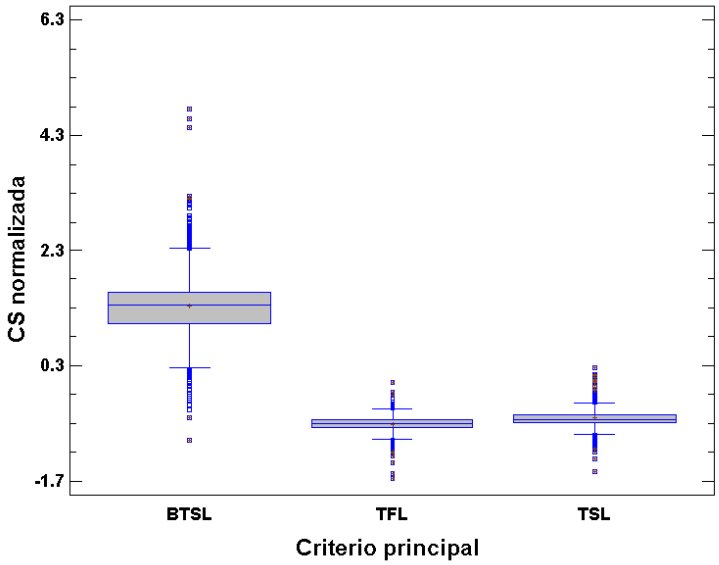

Figura 95. Cota superior normalizada. Comparación del criterio principal utilizado en la construcción de programas factibles.

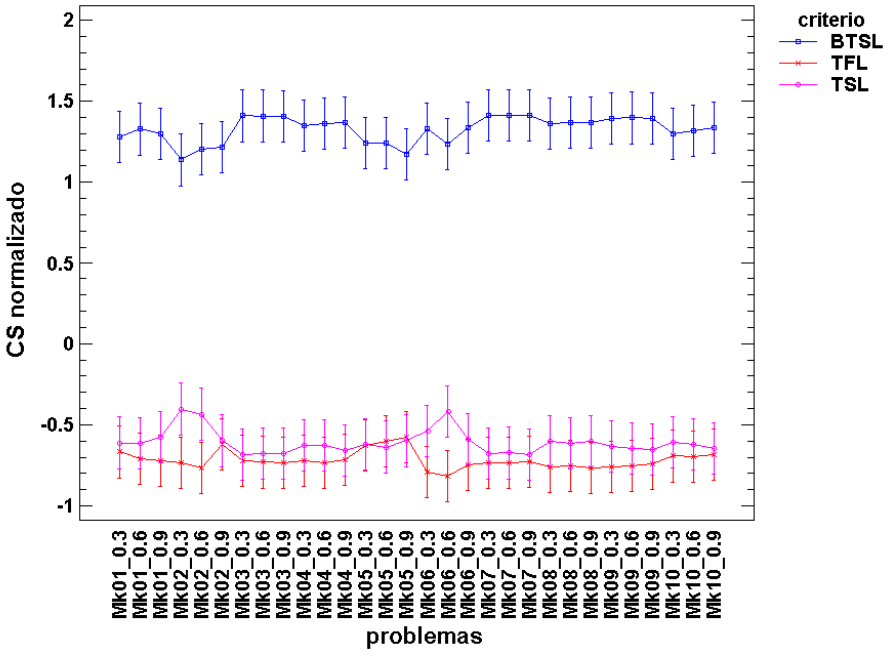

Figura 96. Cota superior normalizada. Comparación del criterio principal utilizado en la construcción de programas factibles por problema.

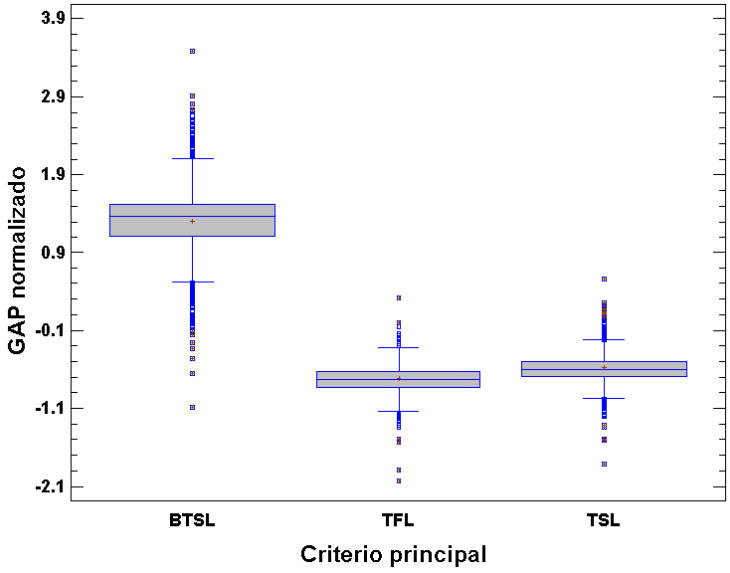

Figura 97. Gap normalizado. Comparación del criterio principal utilizado en la construcción de programas factibles.

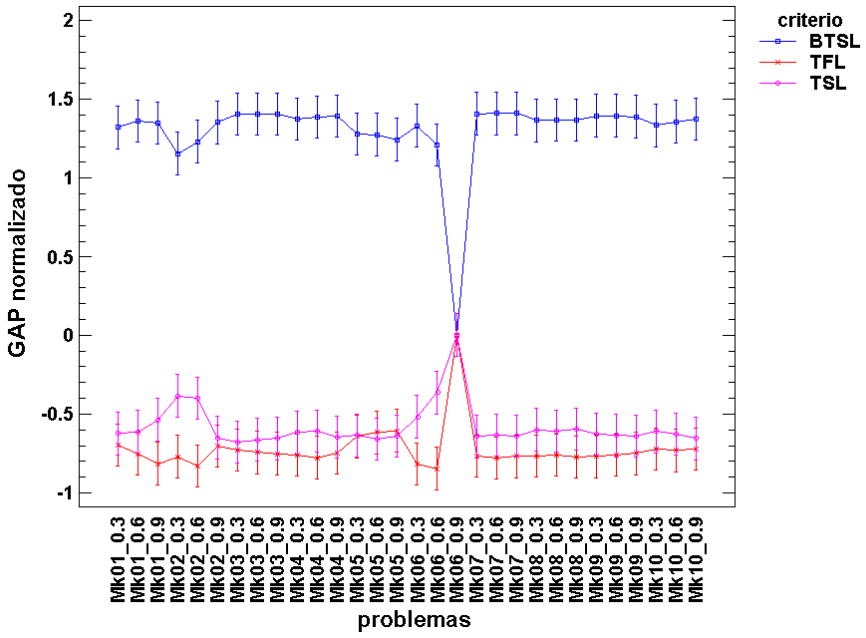

Figura 98. Gap normalizado. Comparación del criterio principal utilizado en la construcción de programas factibles por problema.

		CI prom			CS prom			Media de gap		
código	F	BTSL	TFL	TSL	BTSL	TFL	TSL	BTSL	TFL	TSL
Mk01	0.3	181.7	181.6	181.6	259	**205**	**207**	0.299	0.114	0.122
	0.6	118.7	118.6	118.6	200.5	**141.5**	**145**	0.408	0.161	0.182
	0.9	61.9	61.9	61.9	144	**80**	**86**	0.570	0.227	0.280
Mk02	0.3	127.8	127.8	127.8	181.5	**161**	**164.5**	0.296	0.206	0.223
	0.6	62.8	62.8	62.8	119.5	**92**	**96.5**	0.475	0.319	0.353
	0.9	15.6	15.6	15.6	59	**29**	**29.5**	0.736	0.462	0.471
Mk03	0.3	1334.0	1334.1	1334.2	2030.5	**1502.5**	1512.5	0.343	0.112	0.118
	0.6	895.6	895.6	895.6	1601.5	**1065.5**	1078.5	0.441	0.159	0.170
	0.9	582.0	581.4	582.0	1309	**698**	710	0.555	0.168	0.181
Mk04	0.3	415.1	415.0	415.0	660	**488**	495.5	0.371	0.150	0.162
	0.6	289.1	289.0	289.0	557	**360.5**	372	0.481	0.198	0.223
	0.9	171.1	170.9	171.0	430.5	**239**	244	0.603	0.285	0.299
Mk05	0.3	1297.6	1297.7	1297.7	1795.5	**1483**	1481	0.277	0.125	0.124
	0.6	1083.6	1083.7	1083.7	1511	**1270**	**1266**	0.283	0.147	0.144
	0.9	864.8	864.8	864.8	1269	**1050.5**	**1048**	0.318	0.177	0.175
Mk06	0.3	247.6	247.6	247.6	450	**378.5**	388.5	0.450	0.347	0.364
	0.6	41.0	41.0	41.0	224	**179.5**	189.5	0.817	0.772	0.784
	0.9	0.0	0.0	0.0	125	**36.5**	43	1.000	1.000	1.000
Mk07	0.3	1223.4	1224.6	1224.6	3009.5	**1437**	1481	0.594	0.148	0.173
	0.6	927.0	928.2	928.2	2736	**1148**	1195	0.662	0.191	0.223
	0.9	667.4	668.6	668.6	2445	**847**	881.5	0.727	0.211	0.242
Mk08	0.3	4600.1	4598.7	4598.9	5637	**5250.5**	5276	0.184	0.124	0.128
	0.6	3763.1	3761.8	3761.9	4775	**4410.5**	4439.5	0.212	0.147	0.153
	0.9	2926.6	2925.5	2925.6	3916	**3576.5**	3599.5	0.253	0.182	0.187
Mk09	0.3	2926.4	2926.2	2926.2	4232	**3725.5**	3746.5	0.308	0.215	0.219
	0.6	2096.4	2096.2	2096.2	3423	**2895**	2922	0.388	0.276	0.283
	0.9	1275.7	1275.4	1275.5	2610	**2053**	2076.5	0.511	0.379	0.386
Mk10	0.3	2144.6	2144.8	2144.8	3595	**2908**	2935	0.403	0.262	0.269
	0.6	1374.6	1374.8	1374.8	2839.5	**2134.5**	2167.5	0.517	0.356	0.366
	0.9	677.7	678.3	678.2	2261	**1359.5**	1375	0.700	0.501	0.507

Tabla 7.26. Comparación del criterio principal utilizado en la construcción de programas factibles. Medianas por problema.

	BTSL	TFL	TSL
% Dif CI	0.0%	0.0%	0.0%
% Dif CS	29.4%	0.0%	2.4%
% Dif gap	45.6%	0.1%	5.8%

Tabla 7.27. Diferencias respecto a los valores óptimos en cada problema. Comparación del criterio principal utilizado en la construcción de programas factibles.

Respecto a la convergencia, las Figuras 99 y 100 muestran la evolución de la cota superior y cuota inferior.

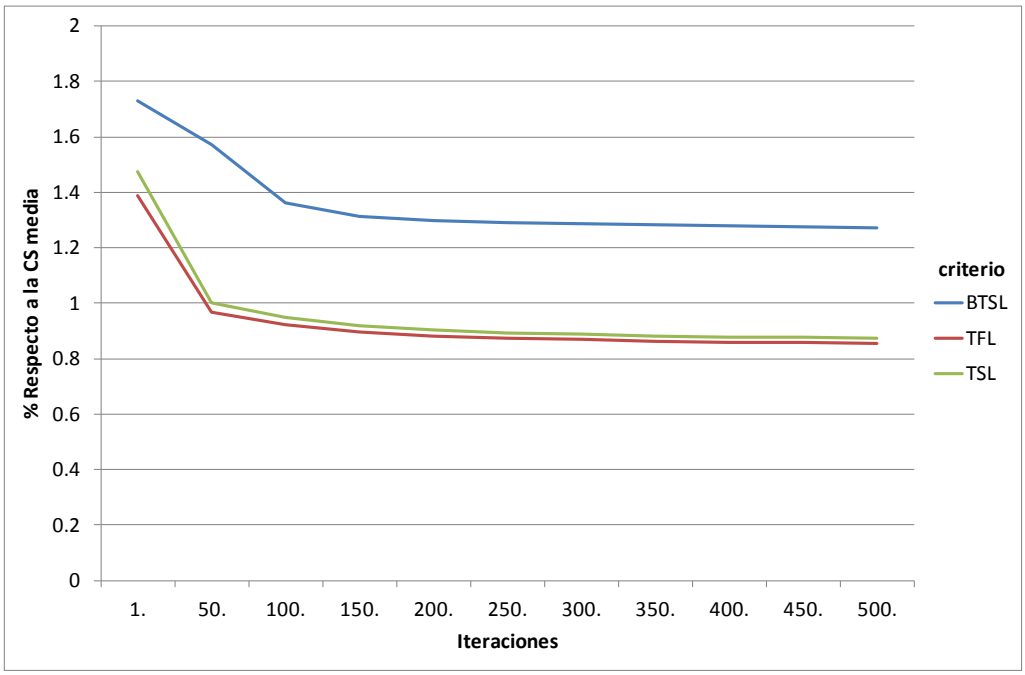

Figura 99. Porcentaje de la cota inferior en cada iteración respecto a la cota superior media final. Comparación del criterio principal utilizado en la construcción de programas factibles.

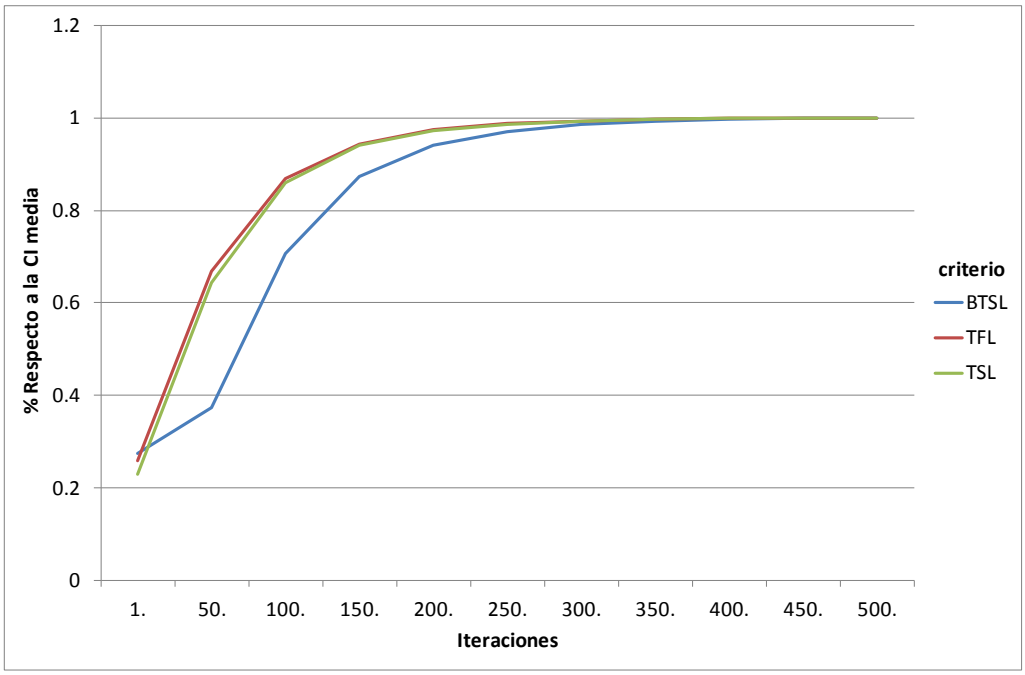

Figura 100. Porcentaje de la cota superior en cada iteración respecto a la cota superior media final. Comparación del criterio principal utilizado en la construcción de programas factibles.

7.5.3. Reglas de prioridad secundarias

Después de estudiar la influencia del criterio primario en la construcción de programas factibles, ahora analizaremos distintas alternativas para usar como criterio de decisión en caso de que el criterio principal no sea discriminatorio; es decir, cuando dos tareas tengan el mismo valor en dicho criterio. De entre los posibles criterios a estudiar se han seleccionado criterios en los que se incorpora en la decisión información del problema. Los criterios que vamos a estudiar han sido definidos en el apartado 6.4.5 y son los siguientes:

- Coste de retraso aparente (atc).
- Tiempo de finalización deseado (dft).
- Ratio crítico (rc).
- Función de penalización marginal (pf).

Los resultados muestran que ninguno de los criterios secundarios tiene mejores resultados que los demás. Respecto a la cota inferior (Figuras 101 y 102, y Tabla 7.28) tanto el criterio del coste de retraso aparente, el tiempo de finalización deseado, el ratio crítico y la penalización marginal por retraso obtienen resultados semejantes. No se han encontrado diferencias significativas en los valores obtenidos y la diferencia media relativa respecto al mejor valor en cada problema es prácticamente cero. Respecto a la cota superior (Figuras 103 y 104, y Tabla 7.28), las diferencias obtenidas en cada problema no se pueden atribuir a un criterio determinado. En las Figuras 105 y 106 se presentan los valores del gap. Las diferencias encontradas en cada problema no son significativas y la diferencia media relativa respecto al mejor valor en cada problema es muy pequeña (Tabla 7.29). En cuanto a la velocidad de convergencia, las Figuras 107 y 108 muestran que no existen diferencias para la cota superior y para la cota inferior, respectivamente.

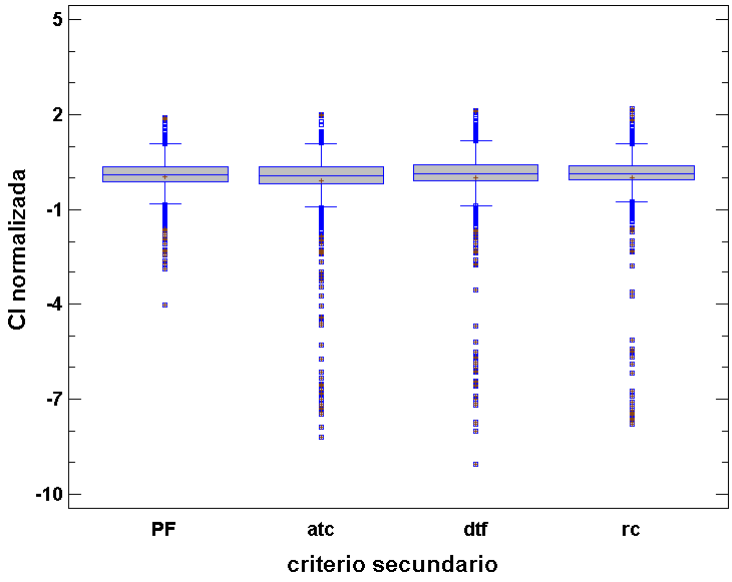

Figura 101. Cota inferior normalizada. Comparación del criterio secundario utilizado en la construcción de programas factibles.

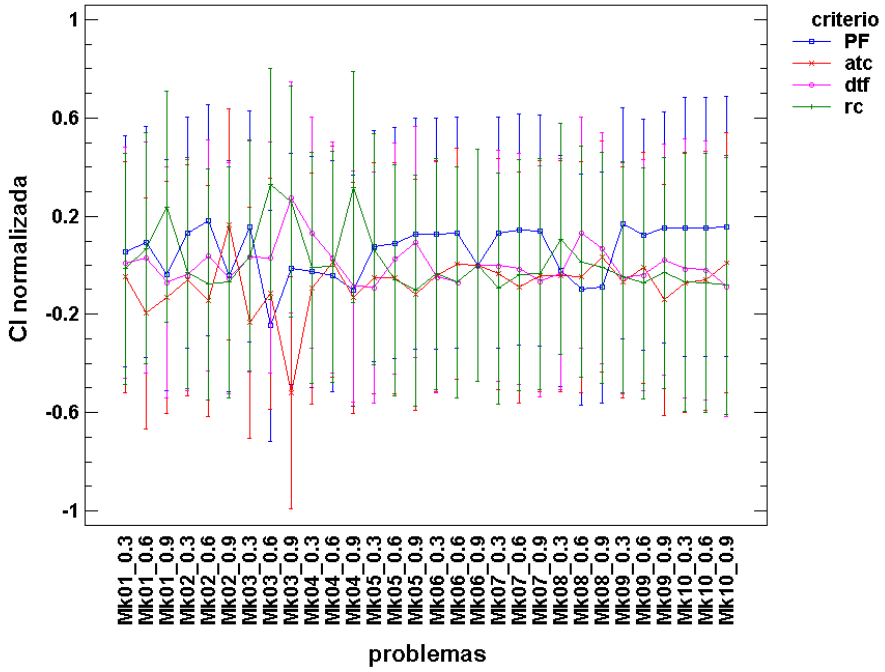

Figura 102. Cota inferior normalizada. Comparación del criterio secundario utilizado en la construcción de programas factibles por problema.

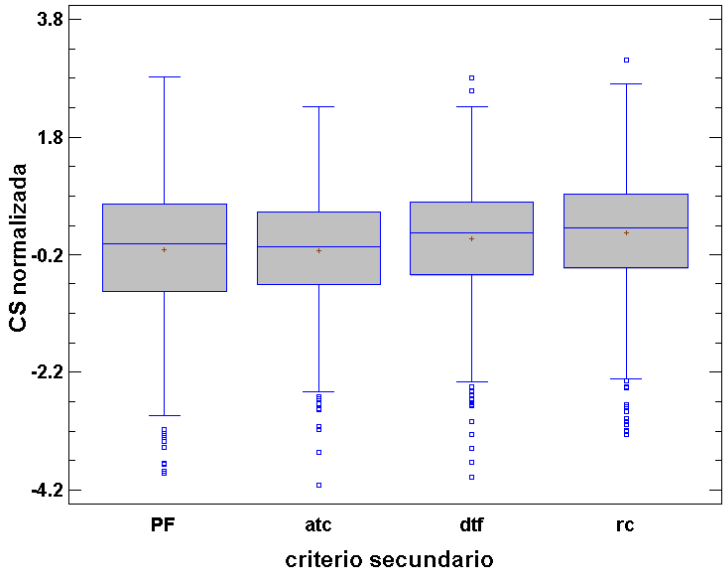

Figura 103. Cota superior normalizada. Comparación del criterio secundario utilizado en la construcción de programas factibles.

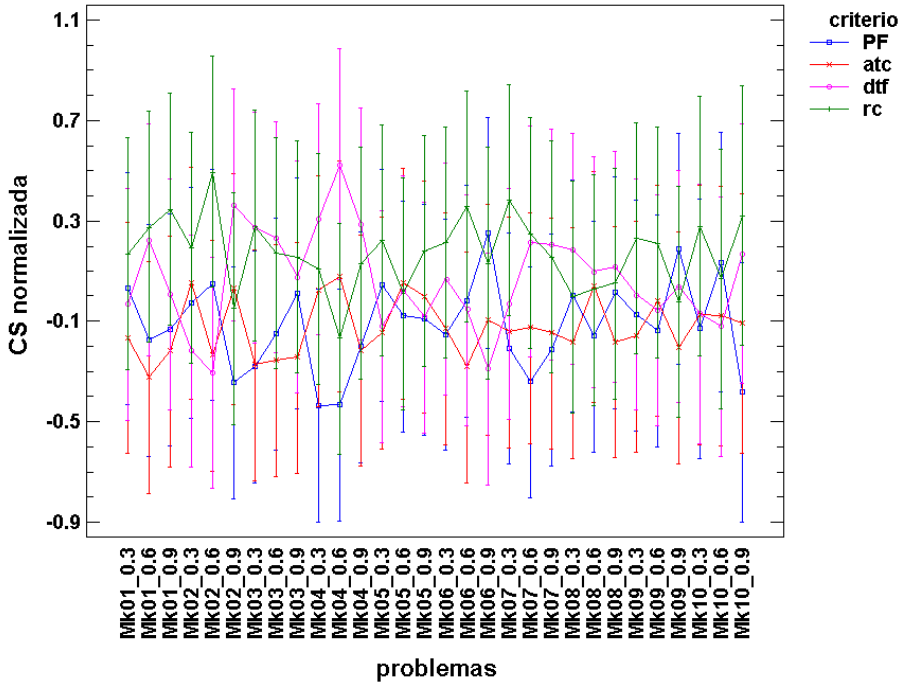

Figura 104. Cota superior normalizada. Comparación del criterio secundario utilizado en la construcción de programas factibles por problema.

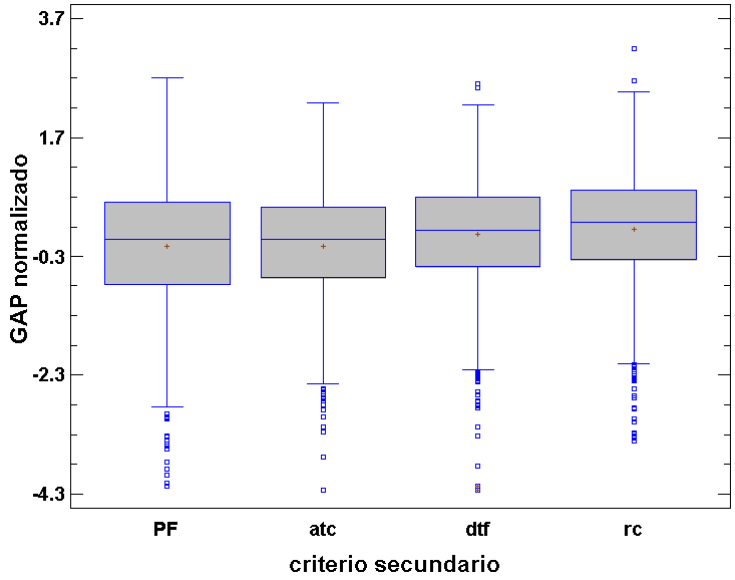

Figura 105. Gap normalizado. Comparación del criterio secundario utilizado en la construcción de programas factibles.

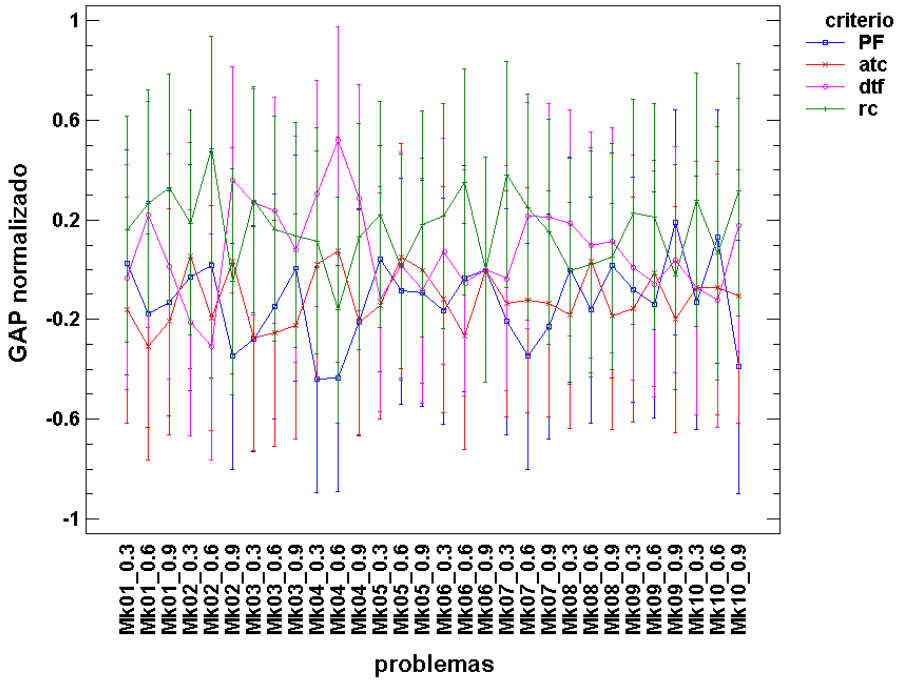

Figura 106. Gap normalizado. Comparación del criterio secundario utilizado en la construcción de programas factibles por problema.

		CI prom				CS prom			
código	F	atc	dtf	rc	PF	atc	dtf	rc	PF
Mk01	0.3	181.6	181.6	181.6	181.6	206.5	207	208	208
	0.6	118.6	118.7	118.7	118.6	142	145	144	145
	0.9	61.9	61.9	61.9	61.9	84.5	85	85	86
Mk02	0.3	127.8	127.8	127.8	127.8	165	165	165	166
	0.6	62.8	62.8	62.8	62.8	95.5	96	98	99
	0.9	15.6	15.6	15.6	15.6	30	31	29	31
Mk03	0.3	1334.2	1334.2	1334.2	1334.2	1517.5	1531	1519	1530
	0.6	895.7	895.7	895.6	895.7	1083.5	1096	1087.5	1096.5
	0.9	581.8	582.1	581.9	582.1	712.5	718.5	719	722
Mk04	0.3	415.0	415.0	415.0	415.0	502	504	500	502
	0.6	289.0	289.0	289.0	289.0	374	377	371.5	373
	0.9	170.9	170.9	170.9	171.0	245.5	249	245.5	248.5
Mk05	0.3	1297.7	1297.7	1297.7	1297.7	1488	1487.5	1485.5	1492
	0.6	1083.7	1083.7	1083.7	1083.7	1273.5	1273.5	1273	1275
	0.9	864.8	864.9	864.9	864.9	1053	1056	1054	1053
Mk06	0.3	247.6	247.6	247.6	247.6	387	388.5	386	390.5
	0.6	41.0	41.0	41.0	41.0	186	188	188	192
	0.9	0.0	0.0	0.0	0.0	46.5	43	49	47.5
Mk07	0.3	1224.6	1224.6	1224.6	1224.6	1482.5	1484.5	1485.5	1494.5
	0.6	928.2	928.2	928.2	928.2	1192.5	1202	1193	1200.5
	0.9	668.6	668.6	668.6	668.6	884.5	893	889	890
Mk08	0.3	4599.0	4598.9	4598.9	4599.1	5272.5	5286	5286	5287
	0.6	3762.0	3762.0	3762.0	3762.0	4449	4451.5	4439.5	4454.5
	0.9	2925.7	2925.7	2925.7	2925.7	3599.5	3611	3601.5	3612
Mk09	0.3	2926.2	2926.2	2926.3	2926.2	3764.5	3765	3768	3774.5
	0.6	2096.3	2096.3	2096.2	2096.2	2933	2934.5	2929.5	2941
	0.9	1275.4	1275.5	1275.5	1275.5	2089.5	2085	2099.5	2094
Mk10	0.3	2144.8	2144.9	2144.9	2144.8	2931	2937	2935	2947
	0.6	1374.8	1374.8	1374.8	1374.8	2160	2166	2176	2171
	0.9	678.3	678.2	678.3	678.3	1382	1396	1377	1399

Tabla 7.28. Comparación del criterio secundario utilizado en la construcción de programas factibles. Medianas por problema.

	atc	dtf	rc	PF
% Dif CI	0.01%	0.00%	0.00%	0.00%
% Dif CS	0.44%	0.78%	0.76%	1.38%
% Dif gap	0.41%	2.19%	1.08%	2.84%

Tabla 7.29. Comparación del criterio secundario utilizado en la construcción de programas factibles. Diferencias respecto a los valores óptimos en cada problema.

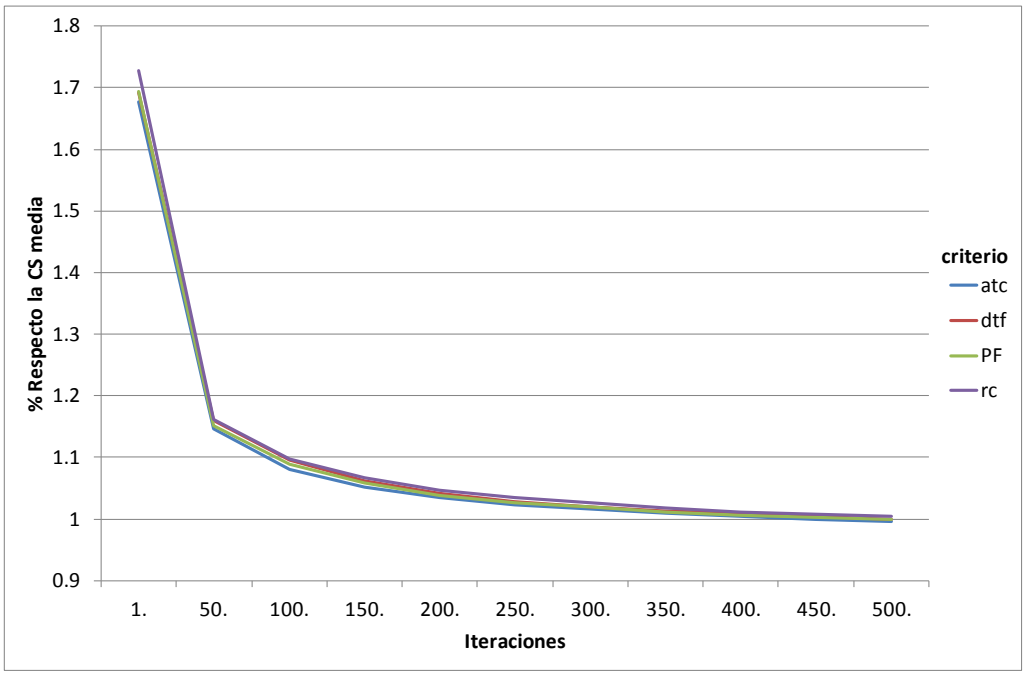

Figura 107. Porcentaje de la cota superior en cada iteración respecto a la cota superior media final. Comparación del criterio secundario utilizado en la construcción de programas factibles.

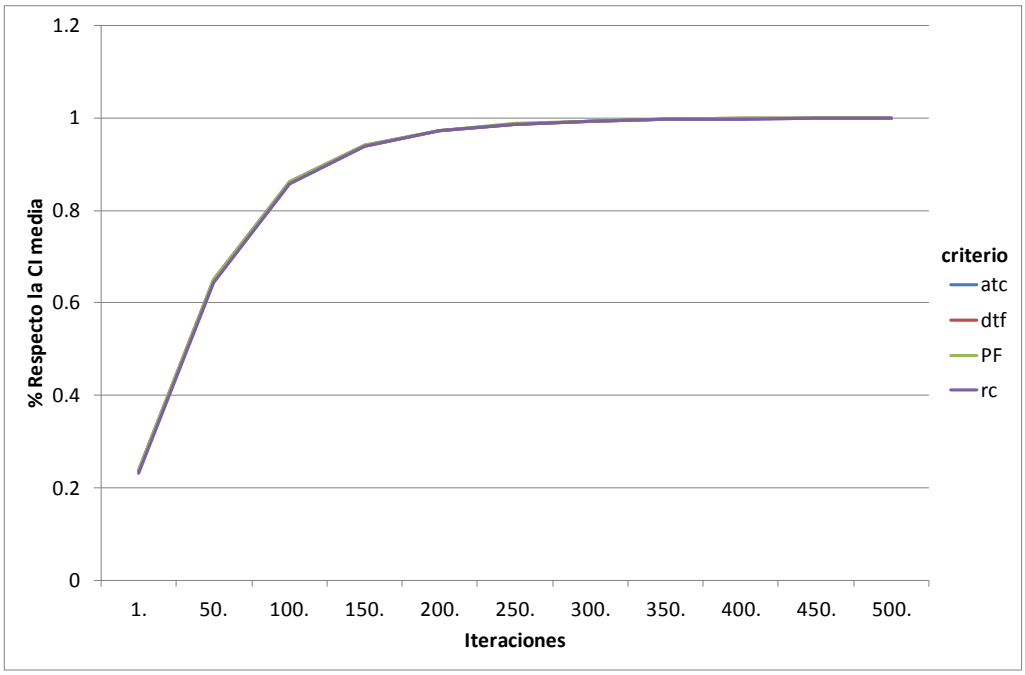

Figura 108. Porcentaje de la cota inferior en cada iteración respecto a la cota inferior media final. Comparación del criterio secundario utilizado en la construcción de programas factibles.

7.6. Conclusiones del capítulo

En este trabajo hemos estudiado cual es la mejor combinación de parámetros y de métodos a utilizar en cada uno de los pasos que forman el algoritmo de Relajación Lagrangiana, con el objetivo de que sea implementada en sistemas distribidos de programación. Para ello, se han probado distintos valores para los parámetros básicos del algoritmo de resolución, estableciendo con ello una jerarquía respecto a cuales son aquellos parámetros realmente importantes en el método –cuyo ajuste es fundamental- y cuáles tienen menor importancia.

Se han dividido los experimentos en dos bloques. En el primero se han estudiado los aspectos relacionados con la actualización de los precios de la subasta. Se han analizado los métodos relacionados con la resolución del problema dual lagrangiano asociado, revisados en la parte de metodología del presente trabajo, y se han comparado el método del subgradiente y el método de actualización walrasiano no adaptativo de actualización de precios. En el segundo bloque se han comparado los métodos, utilizados en trabajos previos, relacionados con la construcción de programas que cumplan con las restricciones del problema y de los criterios utilizados.

Una de las conclusiones básicas obtenidas es que mientras que los valores obtenidos en la cota inferior son muy homogéneos, con coeficientes de variación muy pequeños, los obtenidos como aproximación a la solución del problema o cota superior son algo más heterogéneos, con coeficientes de variación algo más altos, aunque en la mayoría de casos inferiores al 5%.

En los experimentos realizados se confirma la intuición de que los valores obtenidos en la solución del problema están relacionados con la calidad de la cota inferior del problema relajado utilizado como base. Sin embargo, esta relación se produce cuando existen grandes diferencias en la cota inferior. Cuando las diferencias son pequeñas, la calidad de la cota superior apenas se ve afectada, ya que las diferencias son amortiguadas en el proceso de construcción de programas factibles.

Otra de las conclusiones obtenidas es que también es importante la capacidad del método para generar soluciones distintas de la solución del problema relajado en el proceso de actualización de los precios. Esto se ha comprobado en el apartado 7.5.2, donde se ha estudiado la influencia de la información básica con la que se construyen los programas factibles. En este experimento se ha analizado cuál es el criterio principal en dicho proceso. Los peores resultados ocurren cuando se ha utilizado la mejor solución del problema relajado obtenida hasta el momento como base para la

generación de problemas factibles. Esto es debido a que, en este caso, se reduce el campo de búsqueda de soluciones respecto a las otras dos alternativas.

En cuanto a los parámetros relacionados con la actualización de precios, el que aparece como más relevante es el parámetro que controla el número de iteraciones sin mejora en el valor de la cota inferior. La elección de un valor u otro provoca diferencias importantes en la solución obtenida en el cálculo de la cota inferior.

En el estudio de las distintas variantes respecto al método estándar de actualización de precios, una de las que proporciona mejores resultados es la propuesta por Beasley (1993) y por Wang (2003) para el cálculo mejorado del subgradiente, que anula del subgradiente los componentes no activos en el proceso de programación. Adoptando esta mejora se obtienen mejores resultados en él cálculo de la cota inferior.

En cuanto a la utilización del método del gradiente normalizado los resultados obtenidos son algo peores que los obtenidos en el método estándar del subgradiente, aunque esta diferencia es pequeña. El método de Brännlund proporciona peores resultados que el método estándar en el cálculo de la cota interior y de la cota superior. Esta diferencia es más acusada en la cota inferior y se ve amortiguada en la cota superior. Este método puede servir de cota inferior inicial cuando no se conocen los parámetros óptimos de funcionamiento del problema.

Respecto a los métodos que no garantizan la cota inferior de la solución, con el método del gradiente intercalado de Zhao et al (1997) se han obtenido soluciones de calidad equivalente a las del método del subgradiente. Con la versión del método del gradiente intercalado de Kaskavelis y Caramanis (1998), en la que el tamaño del paso se actualiza solo cuando todas las órdenes han enviado su programa local, se han obtenido soluciones de menor calidad.

Con el método walrasiano no adaptativo de actualización de precios se obtienen resultados similares al método del subgradiente utilizando el tamaño del paso adecuado, si bien su convergencia es más lenta. En el método no adaptativo, el factor de cálculo del tamaño del paso tiene gran influencia en la calidad de la solución obtenida. Las mejores soluciones se han obtenido para los valores más pequeños del factor de cálculo del tamaño del paso. Sin embargo, para estos valores la convergencia es mucho más lenta.

Al aplicar el método walrasiano no adaptativo de actualización al método intercalado se obtienen resultados similares al método del subgradiente, aunque no se controle si todas las órdenes de trabajo han enviado su programa local en cada ciclo. El método funciona bien incluso cuando se permite que las órdenes envíen sus

programas locales con una probabilidad determinada e igual para todas. Esto es interesante si el objetivo es la descentralización total del método, aunque tiene el inconveniente de que es necesario determinar cuál es el factor de cálculo del paso adecuado en cada problema.

En cuanto a los métodos de construcción de programas factibles, el método de lanzamiento sin retrasos es el que mejor resultados aporta en la mayoría de los problemas. El método de programación con retrasos desarrollado por Giffler y Thompson (1960) y el algoritmo voraz desarrollado por Hoitomt et al (1993) obtienen peores resultados en la mayoría de problemas. Respecto a la información utilizada como criterios de selección en estos métodos, se usa un criterio principal y otro secundario, que servirá en caso de empate en el criterio principal. En cuanto al criterio principal se obtienen resultados similares utilizando el tiempo de comienzo de las tareas o el tiempo de finalización de las tareas de los programas propuestos por las órdenes de trabajo. Se obtienen peores resultados cuando se utiliza el mejor programa enviado hasta el momento en cada iteración. Respecto al criterio secundario, que se utiliza en caso de empate en el criterio principal, no existen diferencias en la utilización de uno u otro criterio.

Conclusiones y líneas futuras de trabajo

En este trabajo se ha estudiado la aplicación del método de Relajación Lagrangiana al problema de programación de talleres en un entorno multiagente. Este problema es un caso particular del problema general de programación de tareas. La utilización de este método en un entorno multiagente está directamente relacionada con la utilización de las subastas como método de coordinación para resolver el problema de asignación de tareas a máquinas y su secuenciación en un entorno productivo.

Como se mencionó en la Introducción, la realización de este trabajo implicaba el siguiente objetivo: "Estudiar la adecuación de los métodos de programación basados en subastas para ser implementados en sistemas multiagente de programación y control de la producción. Todo ello orientado al diseño de métodos que teniendo las ventajas de los procedimientos distribuidos (flexibilidad, robustez, …) presenten a la vez algunas de las bondades de los métodos centralizados (optimalidad y estabilidad)".

Para ello, se han marcado los siguientes objetivos parciales:

- Contextualizar el problema de programación de talleres flexibles definiendo el problema y revisando los principales métodos que se han aplicado a su resolución.
- Estudiar el método de coordinación basado en subastas, revisando los trabajos prácticos que lo han implementado como medio de coordinación de sistemas multiagente en entornos de producción, realizando su clasificación y

caracterización en función de distintos criterios. Estudiar la relación entre las subastas combinatorias de tipo iterativo y el método de Relajación Lagrangiana como método de resolución del problema de programación de la producción. Esta relación proporciona alternativas al método de actualización de precios. Analizar las propiedades del método de Relajación Lagrangiana que permitan determinar su capacidad de aplicación en entornos distribuidos.

- Determinar los principales parámetros de funcionamiento del método de Relajación Lagrangiana y sus valores óptimos en su aplicación al problema de programación de talleres flexibles.
- Estudiar alternativas al método del subgradiente que permitan aplicar el método de forma asíncrona y reducir el tiempo de computación en los cálculos locales.
- Estudiar las diferentes variantes del método de construcción de programas factibles, incidiendo sobre su calidad y su capacidad de distribución.
- Comparar el método de Relajación Lagrangiana con el método Walrasiano no adaptativo para la actualización de los precios. En estas comparaciones se tendrá en cuenta la calidad de la solución propuesta y la velocidad de convergencia del método.

Creemos que dichos objetivos parciales y, por tanto, el objetivo principal se han alcanzado plenamente, tal y como demuestran las conclusiones a las que hemos llegado y que plasmamos en los siguientes párrafos.

Conclusiones

- En este trabajo se ha estudiado el problema de programación de talleres flexibles. Se han revisado las principales formas de definir el problema, los principales enfoques utilizados para su resolución y se ha analizado la complejidad del problema. La formulación del problema en forma de programación entera posibilita la adopción de mecanismos como el de subasta que permiten resolver el problema. Sin embargo, el coste computacional puede ser muy elevado para horizontes de programación grandes o divisiones de tiempo pequeñas. El ajuste de estas divisiones debe representar un compromiso entre la exactitud de la solución propuesta y el coste de computación.
- Los sistemas distribuidos son una alternativa a los sistemas centralizados para la resolución de problemas complejos como el de programación de talleres. Estos

sistemas, si bien no alcanzan los niveles de calidad de solución de los sistemas centralizados, tienen un mejor comportamiento en entornos dinámicos en cuanto a su capacidad de adaptación a los cambios. Uno de los aspectos importantes, en este tipo de sistemas, es el mecanismo de coordinación que permite orientar el sistema de programación hacia un mismo objetivo. El mecanismo de coordinación basado en subastas se ha utilizado en los últimos años en distintos problemas relacionados con la programación de tareas.

- Un aspecto fundamental en la definición de una subasta es el proceso de actualización de precios. El método Walrasiano no adaptativo es uno de los más simples. En él los precios van disminuyendo o aumentando en proporción constante al exceso de oferta o de demanda, respectivamente. Este método tiene como inconveniente que el factor de cálculo del paso permanece constante independientemente de cómo vaya evolucionando la demanda en función de los precios. La determinación del valor del factor del paso óptimo depende de cada problema y la convergencia del método puede ser muy lenta.
- El método de Relajación Lagrangiana para resolución de problemas de programación entera puede interpretarse como una subasta combinatoria iterativa con fijación de precios cuando se aplica al problema de programación de talleres. La actualización de los precios de la subasta mediante el método de Relajación Lagrangiana puede ser entendido como un método walrasiano de tipo adaptativo, puesto que el factor de actualización de los precios varía en función de la relación oferta-demanda que va teniendo lugar en el proceso de fijación de los precios. De esta forma, la aplicación del método de Relajación Lagrangiana para la actualización de precios en la subasta proporciona la base matemática para estudiar las características del proceso y seleccionar los parámetros que optimicen su funcionamiento.
- El método de Relajación Lagrangiana consiste en obtener las soluciones de un problema de programación entera a partir de las soluciones de su Relajación Lagrangiana, que es más sencilla de resolver. Las soluciones de la Relajación Lagrangiana de un problema de minimización proporcionan una cota inferior de la solución.
- El método del subgradiente es un método iterativo que busca encontrar la cota más ajustada a la solución del problema original: el máximo de las cotas inferiores proporcionadas por la Relajación Lagrangiana del problema. Este método proporciona las condiciones para converger hacia la cota inferior máxima –la cota Lagrangiana-. El método de Relajación Lagrangiana busca soluciones cercanas al

óptimo del problema original generadas mediante la modificación de las soluciones de la Relajación Lagrangiana cercanas a la cota lagrangiana.

- La calidad de las soluciones obtenidas, mediante el método de Relajación Lagrangiana, está relacionada con la calidad de la cota inferior obtenida. En general, cuando la diferencia entre dos valores de la cota inferior es importante, mejores valores de la cota inferior proporcionan mejores soluciones del problema original. Pero no tiene por qué cumplirse que la solución que proporciona la cota Lagrangiana genere la solución óptima del problema original: no existe una relación directa entre las soluciones del problema dual del problema relajado y el problema original.
- El método de Relajación Lagrangiana es un método iterativo de búsqueda de soluciones. Las soluciones del problema relajado acotan la región de búsqueda de soluciones. Cuantas más soluciones diferentes se generen, más posibilidades hay de encontrar un valor óptimo.
- El proceso de transformación de las soluciones del problema relajado en soluciones del problema original tiene gran influencia en la calidad de la solución encontrada. En el problema de programación de talleres flexibles además de realizar la secuenciación y temporización por máquina de las tareas, de forma previa es necesario realizar la carga o asignación de tareas a máquinas. Los mejores resultados se han obtenido con los métodos que utilizan y no modifican la asignación resultante del problema relajado. La utilización de la información de los programas locales propuestos en cada iteración proporciona los mejores resultados. Utilizar otros criterios secundarios que los complementen no aportan diferencias significativas en la calidad de la solución del problema.
- En la resolución del problema dual Lagrangiano mediante el método del subgradiente, en el que se busca maximizar la cota inferior del problema, es necesario garantizar la optimización del problema relajado. En la aplicación del método al problema de programación de talleres, esto implica resolver el subproblema asociado a cada orden de trabajo. Existen métodos alternativos para los que no es necesario optimizar todos los subproblemas en cada iteración, como el método del subgradiente surrogado. Una versión de este método, cuando se aplica a problemas que pueden ser descompuestos como el problema de programación de talleres, es el método del gradiente intercalado en el que se optimiza de forma secuencial un solo subproblema en cada iteración. La calidad de las soluciones del problema obtenidas mediante este método es semejante al del método del subgradiente. El inconveniente es que al eliminar la condición de optimización del problema relajado ya no se garantiza que la solución obtenida del

problema relajado sea una cota inferior de la solución óptima del problema original.

- Una consecuencia del punto anterior es que el proceso de optimización de todos los subproblemas no es fundamental para obtener buenas soluciones. Esto abre la puerta a la investigación de métodos que consuman menos recursos en este paso.
- La convergencia del método de Relajación Lagrangiana es muy sensible a distintos parámetros y la calidad de los resultados obtenidos depende de cómo de ajustados estén dichos parámetros.
- La capacidad de distribución del método de Relajación Lagrangiana cuando se aplica al problema de programación de talleres depende de la función objetivo elegida. Cuando la función objetivo del problema se calcula como suma de las funciones objetivo locales de las órdenes de trabajo –como es el caso de la función objetivo estudiada, la suma ponderada de los retrasos de las órdenes de trabajo- es posible su distribución. En ese caso, al plantear el método de Relajación Lagrangiana como una subasta se permite distribuir los cálculos a realizar, como la resolución de cada subproblema de optimización local, así como la información que tiene que utilizar cada uno de los agentes participantes.
- Cuando se utiliza la subasta como mecanismo de coordinación de un sistema multiagente para la resolución del problema de programación de talleres es necesario definir el método de actualización de precios. Tanto el método walrasiano no adaptativo como el método basado en el método del subgradiente obtienen soluciones del problema de calidad similar. En el método del subgradiente es fundamental encontrar cual es el factor de cálculo del paso para obtener buenas soluciones. Por su parte, con el método del subgradiente se obtienen un mejor compromiso entre la calidad de la solución obtenida y la velocidad de convergencia del método.
- El método distribuido de Relajación Lagrangiana necesita de un elemento central, el subastador en términos de la subasta, que coordine los flujos de intercambio de información, así como la información intercambiada. Este método necesita de cierta información global para su correcto funcionamiento, aunque puede ser mínima. Es necesario un elemento central que calcule el tamaño del paso y para calcularlo se necesita recibir: (1) por parte de las órdenes de trabajo, el valor de la subfunción Lagrangiana asociada a cada una; (2) por parte de las máquinas, el valor de la suma de los multiplicadores de Lagrange –los precios- de cada una. Con eso el elemento central –subastador- puede calcular el tamaño del paso que enviará a las máquinas y les permitirá recalcular sus precios. El resto de

interacciones pueden ser realizadas directamente entre máquinas y órdenes de trabajo.

- Al utilizar el método walrasiano no adaptativo para actualizar los precios en la subasta se elimina la necesidad de información central y el método puede ser descentralizado completamente. A cambio se pierde calidad en la solución del problema si no se utiliza el factor de cálculo del paso adecuado y la convergencia es más lenta.

Líneas futuras de trabajo

- Un aspecto a estudiar es la vigencia de las conclusiones obtenidas cuando el método se aplica en entornos dinámicos, donde las condiciones del problema cambian y no son previstas con antelación.
- Otro de los aspectos a estudiar es la capacidad del método para resolver problemas reales de tamaño grande. En los experimentos realizados se ha detectado que el gap, diferencia entre la solución del problema y la cota inferior obtenida en el método, tiende a ser mayor con el aumento del tamaño de los problemas. Esto puede ser debido a una disminución de la calidad de la solución o a una pérdida de eficacia por parte del método para encontrar una cota inferior cercana al mínimo del problema original.
- La mejora de los métodos que transforman las soluciones del problema relajado en soluciones del problema original es otro de los aspectos a desarrollar. Se trata de encontrar métodos descentralizados de construcción de programas factibles que encuentren soluciones más cercanas al óptimo del problema en un tiempo razonable.
- La utilización del método del gradiente surrogado para resolver el problema dual Lagrangiano se ha mostrado como una fuente de oportunidades para encontrar nuevos métodos, relacionados principalmente con la no necesidad de optimizar cada uno de los subproblemas derivados del problema relajado. Se buscan alternativas que cumplan con las condiciones del método del gradiente surrogado, en los que un valor que no sea óptimo pero que mejore el valor de la función pueda ser obtenido de forma más sencilla y rápida y proporcione soluciones al problema original de calidad similar.
- Se ha comprobado que para el método de actualización de precios walrasiano no adaptativo es posible encontrar valores de los parámetros ligados al método con

los que se puede encontrar soluciones equivalentes a las obtenidas con el método del subgradiente. Esto abre la puerta al estudio de métodos completamente descentralizados en los que se busque una mejora de la velocidad de convergencia de este método.

- Estableciendo un paralelismo en la actualización de precios entre el método del subgradiente y el método de actualización de precios walrasiano no adaptativo, se podrían encontrar métodos derivados de este último que cumplan con las condiciones del método del gradiente surrogado, de modo que no fuera necesario realizar la optimización local por parte de cada una de las órdenes de trabajo en el problema de programación de talleres.
- También parece interesante extender el estudio realizado a otros problemas de la misma naturaleza y comparar el comportamiento del método en cada caso.

Bibliografía

Abrache J, Crainic, TG, Gendreau M, Rekik M (2007). Combinatorial auctions. *Annals of Operations Research* 153(1):131–164.

Adams J, Balas E, Zawack D (1988). The shifting bottleneck procedure for job shop scheduling. *Management Science* 34(3):391–401.

Agnetis A, Mirchandani PB, Pacciarelli D, Pacifici A (2004). Scheduling problems with two competing agents. *Operations Research* 52(2):229–242.

Agnetis A, Pacciarelli D, Pacifici A (2007). Combinatorial models for multi-agent scheduling problems. In E Levner (ed.), *Multiprocessor Scheduling: Theory and Applications*, 21-46. Vienna: I-Tech.

Aiex RM, Binato S, Resende MG (2003). Parallel GRASP with path-relinking for job shop scheduling. *Parallel Computing* 29(4):393–430.

Akers SB (1956). A graphical approach to production scheduling problems. *Operations Research* 3:429-442.

Akyol DE, Bayhan GM (2007). A review on evolution of production scheduling with neural networks. *Computers & Industrial Engineering* 53(1):95–122.

Álvarez E (2007). Multi-plant production scheduling in SMEs. *Robotics and Computer-Integrated Manufacturing* 23(6):608–613.

Anandalingam G, Day RW, Raghavan S (2005). The landscape of electronic market design. *Management Science* 51(3):316–327.

Anussornnitisarn P, Nof SY, Etzion O (2005). Decentralized control of cooperative and autonomous agents for solving the distributed resource allocation problem. *International Journal of Production Economics* 98(2):114–128.

Applegate D, Cook W (1991). A computational study of the job-shop scheduling problem. *Informs Journal on Computing* 3(2):149–156.

Araúzo JA (2007). *Control distribuido de sistemas de fabricación flexibles: Un enfoque basado en agentes*. Thesis. University of Valladolid.

Araúzo JA, Galán JM, Pajares J, López-Paredes A (2009). Multi-agent technology for scheduling and control projects in multi-project environments. An auction based approach. *Revista Iberoamericana de Inteligencia Artificial* 13(42):12–20.

Araúzo JA, Pajares J, Lopez-Paredes A (2010). Simulating the dynamic scheduling of project portfolios. *Simulation Modelling Practice and Theory* 18(10):1428–1441.

Archimede B, Coudert T (2001). Reactive scheduling using a multi-agent model: The SCEP framework. *Engineering Applications of Artificial Intelligence* 14(5):667–683.

Armentano VA, Scrich CR (2000). Tabu search for minimizing total tardiness in a job shop. *International Journal of Production Economics* 63(2):131–140.

Arora S, Barak B (2009). *Computational Complexity: A Modern Approach*. Leiden: Cambridge University Press.

Arzi Y, Iaroslavitz L (1999). Neural network-based adaptive production control system for a flexible manufacturing cell under a random environment. *IIE Transactions* 31(3):217–230.

Asano M, Ohta H (2002). A heuristic for job shop scheduling to minimize total weighted tardiness. *Computers & Industrial Engineering* 42(2-4):137–147.

Attanasio A, Ghiani G, Grandinetti L, Guerriero F (2006). Auction algorithms for decentralized parallel machine scheduling. *Parallel Computing* 32(9):701–709.

Ausubel LM, Cramton P, Milgrom P (2006). The clock-proxy auction: A practical combinatorial auction design. In P Cramton, Y. Shoham, R Steinberg (eds.), *Combinatorial Auctions*, 115–138. Cambridge: MIT Press.

Aydin ME, Öztemel E (2000). Dynamic job-shop scheduling using reinforcement learning agents. *Robotics and Autonomous Systems* 33(2-3):169–178.

Aytug H, Bhattacharyya S, Koehler GJ, Snowdon JL (1994). A review of machine learning in scheduling. *IEEE Trans. on Engineering Management* 41(2):165–171.

Babiceanu RF, Chen FF (2006). Development and applications of holonic manufacturing systems: A survey. *J of Intelligent Manufacturing* 17(1):111–131.

Babiceanu RF, Chen FF, Sturges RH (2005). Real-time holonic scheduling of material handling operations in a dynamic manufacturing environment. *Robotics and Computer-Integrated Manufacturing* 21(4-5):328–337.

Baker AD (1996). Metaphor or reality: a case study where agents BID with actual costs to schedule a factory. In SH Clearwater (ed.), *Market-Based Control: A Paradigm for Distributed Resource Allocation*, 184–223. New York:Addison-Wesley.

Baker KR (1984). Sequencing rules and due-date assignments in a job shop. *Management Science* 30(9):1093–1104.

Balas E, Vazacopoulos A (1998). Guided local search with shifting bottleneck for job shop scheduling. *Management Science* 44(2):262–275.

Balasubramanian S, Maturana FP, Norrie DH (1996). Multi-agent planning and coordination for distributed concurrent engineering. *Issues* 2(3):153–179.

Baptiste P, Flamini M, Sourd F (2008). Lagrangian bounds for just-in-time job-shop scheduling. *Computers & Operations Research* 35(3):906–915.

Baptiste P, Le Pape C, Nuijten W (2001). *Constraint-based scheduling: Applying constraint programming to scheduling problems*. Norwell: Kluwer.

Barbati M, Bruno G, Genovese A (2012). Applications of agent-based models for optimization problems: A literature review. *Expert Systems with Applications* 39(5):6020–6028.

Bean JC, Birge JR, Mittenthal J, Noon ChE (1991). Matchup scheduling with multiple resources, release dates and disruptions. *Operations Research* 39(3):470–483.

Beasley JE (1993). Lagrangean heuristics for location problems. *European Journal of Operational Research* 65(3):383–399.

Beck JC, Refalo P, Taissounieres L (2003). A hybrid approach to scheduling with earliness and tardiness costs. *Annals of Operations Research* 118:49–71.

Bellifemine F, Caire G, Poggi A, Rimassa G (2008). JADE: A software framework for developing multi-agent applications. Lessons learned. *Information and Software Technology* 50(1-2):10–21.

Bellman RE (1957). *Dynamic programming*. Princeton: Princeton University Press.

Belmonte MV, Carbone F, Fernandez A, García A, Hermoso R, et al (2005). *Diseño y aplicación de modelos multiagente para la ayuda a la decisión*. Madrid: SP-URJC.

Benders JF (1962). Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik* 4(1):238–252.

Berryman MJ, Angus SD (2010). Tutorials on agent-based modelling with NetLogo and network analysis with Pajek. In RL Dewar, F Detering (eds.), *Complex Physical, Biophysical and Econophysical Systems*, 351–375. Singapore: World Scientific Pub.

Bertsekas DP (1990). The auction algorithm for assignment and other network flow problems: a tutorial. *Interfaces* 20(4):133–149.

Bierwirth C, Kopfer H, Mattfeld DC, Rixen I (1995). Genetic algorithm based scheduling in a dynamic manufacturing environment. In M Palaniswan (ed.), *Proceedings of the Second Conference on Evolutionary Computatio*n, 439–443. New York: IEEE Press.

Blackstone JH, Phillips DT, Hogg GL (1982). A state-of-the-art survey of dispatching rules for manufacturing job shop operations. *International Journal of Production Research* 20(1):27–45.

Blazewicz J, Domschke W, Pesch E (1996). The job shop scheduling problem: conventional and new solution techniques. *European Journal of Operational Research* 93(1):1–33.

Blazewicz J, Ecker K, Pesch E, Schmidt G, Weglarz J (2007). *Handbook on scheduling: From theory to applications*. Berlin: Springer.

Blum C, Roli A (2003). Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys* 35(3):268–308.

Blum C, Sampels M (2004). An ant colony optimization algorithm for shop scheduling problems. *J of Mathematical Modelling and Algorithms* 3(3):285–308.

Bongaerts L, Monostori L, McFarlane D, Kádár B (2000). Hierarchy in distributed shop floor control. *Computers in Industry* 43(2):123–137.

Brandimarte P (1993). Routing and scheduling in a flexible job shop by tabu search. *Annals of Operations Research* 41(3):157–183.

Brännlund U (1995). A generalized subgradient method with relaxation step. *Mathematical Programming* 71(2):207–219.

Brassard G, Bratley P (1996). *Fundamentals of algorithmics*. Englewood Cliffs: Prentice Hall.

Brennan RW, Fletcher M, Norrie DH (2002). An agent-based approach to reconfiguration of real-time distributed control systems. *IEEE Transactions on Robotics and Automation* 18(4):444–451.

Brennan RW, Norrie D (2001). Evaluating the performance of reactive control architectures for manufacturing production control. *Computers in Industry* 46(3):235–245.

Brucker P (2009). www.informatik.uni-osnabrueck.de

Brucker P (2007). *Scheduling algorithms*. New York: Springer.

Brucker P, Jurisch B, Krämer A (1997). Complexity of scheduling problems with multi-purpose machines. *Annals of Operations Research* 70:57–73.

Brucker P, Jurisch, B, Sievers B (1994). A branch and bound algorithm for the job-shop scheduling problem. *Discrete Applied Mathematics* 49(1-3):107–127.

Bülbül K (2011). A hybrid shifting bottleneck-tabu search heuristic for the job shop total weighted tardiness problem. *Computers & Operations Research* 38(6):967–983.

Burke EK, Kendall G (2005). *Search methodologies: Introductory tutorials in optimization and decision support techniques*. New York: Springer.

Bussmann S, Schild K (2001). An agent-based approach to the control of flexible production systems. In *Proc of the 8th IEEE Int Conf on Emergent Technologies and Factory Automation*, Vol 2:481–488.

Camerini PM, Fratta L, Maffioli F (1975). On improving relaxation methods by modified gradient techniques. *Mathematical Programming Study* 3:26–34. Amsterdam:North Holland.

Caridi M, Cavalieri S (2004). Multi-agent systems in production planning and control: An overview. *Production Planning & Control* 15(2):106–118.

Chen D, Luh PB, Thakur LS, Moreno J (2003). Optimization-based manufacturing scheduling with multiple resources, setup requirements, and transfer lots. *IIE Transactions* 35(10):973–985.

Chen H, Chu C (2003). A lagrangian relaxation approach for supply chain planning with order/setup costs and capacity constraints. *Journal of Systems Science and Systems Engineering* 12(1):98–110.

Chen H, Chu C, Proth JM (1998). An improvement of the Lagrangean relaxation approach for job shop scheduling: A dynamic programming method. *IEEE Transactions on Robotics and Automation* 14(5):786–795 .

Chen H, Chu C, Proth JM (1995). More efficient Lagrangian relaxation approach to job-shop scheduling problems. In *Proc of the IEEE Int Conf on Robotics and Automation*, Vol 3:496–501.

Chen H, Luh P (2003). An alternative framework to Lagrangian relaxation approach for job shop scheduling. *European Journal of Operational Research* 149(3):499–512.

Chen R-M, Huang Y-M (2001). Competitive neural network to solve scheduling problems. *Neurocomputing* 37(1-4):177–196.

Cheng R, Gen M, Tsujimura Y (1999). A tutorial survey of job-shop scheduling problems using genetic algorithms, part II: hybrid genetic search strategies. *Computers & Industrial Engineering* 36(2):343–364.

Cheng R, Gen M, Tsujimura Y (1996). A tutorial survey of job-shop scheduling problems using genetic algorithms--I. representation. *Computers & Industrial Engineering* 30(4):983–997.

Cho H., Wysk RA (1993). A robust adaptive scheduler for an intelligent workstation controller. *International Journal of Production Research* 31(4):771–789.

Christo C, Cardeira C (2007). Trends in intelligent manufacturing systems. In *Proc of the IEEE Int Symposium on Industrial Electronics*, 3209–3214.

Chryssolouris G, Subramaniam V (2001). Dynamic scheduling of manufacturing job shops using genetic algorithms. *Journal of Intelligent Manufacturing* 12(3):281–293.

Church LK, Uzsoy R (1992). Analysis of periodic and event-driven rescheduling policies in dynamic shops. *International Journal of Computer Integrated Manufacturing* 5(3):153–163.

Cixing L, Yunlong Z, Chaowan Y (2005). Resource allocation based on combinatorial auction in e-manufacturing environment. In N Gu, D Wei, Z Xic et al (eds.), *Proc of the Fifth Int Conf on Computer and Information Technology*, 983–988.

Companys R (1989). *Planificación y programación de la producción*. Barcelona: Marcombo.

Cowling P, Johansson M (2002). Using real time information for effective dynamic scheduling. *European Journal of Operational Research* 139(2):230–244.

Cramton P, Shoham Y, Steinberg R (eds.) (2006). Combinatorial auctions. Boston: MIT Press.

Cuatrecasas L (2009). *Diseño avanzado de procesos y plantas de produccion flexible*, (2nd ed). Barcelona: Profit Editorial.

Cutkosky MR, Engelmore RS, Fikes RE, Genesereth MR, Gruber TR, Mark WS, Tenenbaum JM, Weber JC (1993). PACT: An experiment in integrating concurrent engineering systems. *Computer* 26(1):28–37.

Czerwinski CS, Luh PB (1994). Scheduling products with bills of materials using an improved Lagrangian relaxation technique. *IEEE Transactions on Robotics and Automation* 10(2):99–111.

Czerwinski CS, Luh PB (1992). An improved Lagrangian relaxation approach for solving job shop scheduling problems. In *Proc of the 31st IEEE Conf on Decision and Control*, Vol 1:771–776.

Dantzig GB, Wolfe P (1960). Decomposition principle for linear programs. *Operations Research* 8(1):101–111.

Davies RW (2005). *Distributed generalized vickrey auctions based on the Dantzig-Wolfe and Benders decomposition methods for linear programs*. Thesis, Bachelor of Arts, Harvard College. Cambridge, Massachusetts.

Davis R, Smith RG (1983). Negotiation as a metaphor for distributed problem solving. *Artificial intelligence* 20(1):63–109.

DeLoach S, Wood M (2001). Developing multiagent systems with agentTool. In C Castelfranchi, Y Lespérance (eds.), *Intelligent Agents VII. Agent Theories Architectures and Languages*, 30–41. Boston: Springer.

De Vries S, Vohra RV (2003). Combinatorial auctions: A survey. *Informs Journal on Computing* 15(3):284–309.

Dell'Amico M, Trubian M (1993). Applying tabu search to the job-shop scheduling problem. *Annals of Operations Research* 41(3):231–252.

Dewan P, Joshi S (2002). Auction-based distributed scheduling in a dynamic job shop environment. *International Journal of Production Research*, 40(5):1173–1191.

Dewan P, Joshi S (2001). Implementation of an auction-based distributed scheduling model for a dynamic job shop environment. *International Journal of Computer Integrated Manufacturing* 14(5):446–456.

Dewan P, Joshi S (2000). Dynamic single-machine scheduling under distributed decision-making. *International Journal of Production Research* 38(16):3759–3777.

Dorigo M, Stützle T (2004). *Ant colony optimization*. Cambridge: MIT Press.

Duffie NA (1990). Synthesis of heterarchical manufacturing systems. *Computers in Industry* 14(1-3):167–174.

Edis EB, Araz C, Ozkarahan I (2008). Lagrangian-based solution approaches for a resource-constrained parallel machine scheduling problem with machine eligibility restrictions. In NT Nguyen, L Borzemski, A Grzech, M Ali (eds.), *New Frontiers in Applied Artificial Intelligence*, 337–346. Berlin: Springerg.

El-Bouri A (2012). A cooperative dispatching approach for minimizing mean tardiness in a dynamic flowshop. *Computers & Operations Research* 39(7):1305–1314.

Essafi I, Mati Y, Dauzère-Pérès S (2008). A genetic local search algorithm for minimizing total weighted tardiness in the job-shop scheduling problem. *Computers and Operations Research* 35(8):2599–2616.

Fan M, Stallaert J, Whinston AB (2003). Decentralized mechanism design for supply chain organizations using an auction market. *Information Systems Research* 14(1):1–22.

Fang L, Luh PB, Chan H (2000). Improving the Lagrangian relaxation approach for large job-shop scheduling. In *Proc of the 39th IEEE Conf on Decision and Control*, Vol. 1:552–557.

Feng S, Li L, Cen L, Huang J (2003). Using MLP networks to design a production scheduling system. *Computers & Operations Research* 30(6):821–832.

Feng S, Stouffer KA, Jurrens KK (2005). Manufacturing planning and predictive process model integration using software agents. *Advanced Engineering Informatics* 19(2):135–142.

Ferber J (1999). *Multi-agent systems: An introduction to distributed artificial intelligence*. London: Addison Wesley.

Fisher ML (1985). An applications oriented guide to Lagrangian relaxation. *Interfaces* 15(2):10–21.

Fisher ML (1981). The Lagrangian relaxation method for solving integer programming problems. *Management Science* 27(1):1–18.

Fonseca DJ, Navaresse D (2002). Artificial neural networks for job shop simulation. *Advanced Engineering Informatics* 16(4):241–246.

Forde TK, Doyle LE (2008). A combinatorial clock auction for OFDMA-based cognitive wireless networks. In *Proc of the 3rd International Symposium on Wireless Pervasive Computing*, 329–333.

Fox MS, Barbuceanu M, Teigen R (2000). Agent-oriented supply-chain management. *International Journal of Flexible Manufacturing Systems* 12(2):165–188.

Frey D, Nimis J, Wörn H, Lockemann P (2003). Benchmarking and robust multi-agent-based production planning and control. *Engineering Applications of Artificial Intelligence* 16(4):307–320.

Garey MR, Johnson DS (1990). *Computers and intractability; A guide to the theory of NP-completeness*. New York: Freeman.

Gélinas S, Soumis F (2005). Dantzig-Wolfe decomposition for job shop scheduling. In G Desaulniers, J Desrosiers, MM Solomon (eds.), *Column Generation*, 271–302. New York: Springer.

Geoffrion AM (1974). Lagrangean relaxation for integer programming. In ML Balinski (ed.), *Approaches to Integer Programming*, 82–114. Amsterdam: North Holland.

Giffler B, Thompson GL (1960). Algorithms for solving production-scheduling problems. *Operations Research* 8(4):487–503.

Giret A, Botti V (2009. Engineering holonic manufacturing systems. *Computers in Industry* 60(6):428–440.

Glover F, Taillard E, De Werra D (1993). A user's guide to tabu search. *Annals of Operations Research* 41(1):3–28.

Gonçalves JF, Mendes JJ, Resende MG (2005). A hybrid genetic algorithm for the job shop scheduling problem. *European Journal of Operational Research* 167(1):77–95.

Gou L, Luh PB, Kyoya Y (1998). Holonic manufacturing scheduling: Architecture, cooperation mechanism, and implementation. *Computers in Industry* 37(3):213–231.

Grabowskil J, Wodecki M (2005). A very fast tabu search algorithm for job shop problem. In C Rego, B Alidaee (eds.), *Metaheuristic Optimization via Memory and Evolution*, 117–144. Boston: Kluwer.

Graham RL, Lawler EL, Lenstra JK, Rinnooy Kan AHG (1979). Optimization and approximation in deterministic sequencing and scheduling: A survey. In PL Hammer, EL Johnson, BH Korte (eds.), *Discrete Optimization II*, 287–326.Vancouver: Elsevier.

Guignard M (2003). Lagrangean relaxation. *TOP* 11(2):151–200.

Guo Y, Lim A (2006). Using a lagrangian heuristic for a combinatorial auction problem. *International Journal of Artificial Intelligence Tools* 15(3):481–489.

Hadeli, Valckenaers P, Kollingbaum M, Van Brussel H (2004). Multi-agent coordination and control using stigmergy. *Computers in Industry* 53(1):75–96.

Haupt R (1989). A survey of priority rule-based scheduling. *OR Spektrum* 11(1):3–16.

Hayes RH, Wheelwright SC (1979). Link manufacturing process and product life cycles. *Harvard Business Review* (January–February):135–142.

Held M, Wolfe P, Crowder HP (1974). Validation of subgradient optimization. *Mathematical Programming* 6(1):62–88.

Helsinger A, Wright T (2005). Cougaar: A robust configurable multi agent platform. In *Proceedings on Aerospace Conference*, 1–10.

Hiriart-Urruty J-B, Lemarechal C (1996). *Convex analysis and minimization algorithms I: Fundamentals*. Berlin: Springer.

Ho, N., Tay, J., 2005. Evolving dispatching rules for solving the flexible job-shop problem. In *IEEE Congress on Evolutionary Computation*, Vol. 3:2848– 2855.

Hoitomt DJ, Luh PB, Pattipati KR (1993). A practical approach to job-shop scheduling problems. *IEEE Transactions on Robotics and Automation* 9(1):1–13.

Hsieh F-S (2007). Combinatorial auction with minimal resource requirements. In HG Okuno, M Ali (eds.), *New Trends in Applied Artificial Intelligence*, 1072–1077. Berlin: Springer.

Hsieh F-S (2010). Combinatorial reverse auction based on revelation of Lagrangian multipliers. *Decision Support Systems* 48(2):323–330.

Iglesias C, Garijo M, González JC, Velasco JR (1998). Analysis and design of multiagent systems using MAS-CommonKADS. In M Singh, A Rao, M Wooldridge

(eds.), *Intelligent Agents IV: Agent Theories, Architectures, and Languages*, 313–327. Berlin: Springer.

Ishii N, Talavage JJ (1991). A transient-based real-time scheduling algorithm in FMS. *International Journal of Production Research* 29(12):2501–2520.

Jackson JR (1956). An extension of Johnson's result on job lot scheduling. *Naval Research Logistics Quarterly* 3(3):201–203.

Jain AS, Meeran S (1999). Deterministic job-shop scheduling: Past, present and future. *European Journal of Operational Research* 113(2):390–434.

Jayamohan M, Rajendran C (2004). Development and analysis of cost-based dispatching rules for job shop scheduling. *European Journal of Operational Research* 157(2):307–321.

Jeng MD, Chang CY (1997). Non-energy based neural networks for job-shop scheduling. *Electronics Letters* 33(5):399–400.

Jennings N (2000). On agent-based software engineering. *Artificial Intelligence* 117(2):277–296.

Jennings N, Sycara K, Wooldridge M (1998). A roadmap of agent research and development. *Autonomous Agents and Multi-Agent Systems* 1(1):7–38.

Jennings N, Wooldridge M (1998). Applications of intelligent agents. In N Jennings, M Wooldridge (eds.), *Agent Technology: Foundations, Applications and Markets*, 3–28. Berlin: Springer.

Jeong I-J, Leon VJ (2005). A single-machine distributed scheduling methodology using cooperative interaction via coupling agents. *IIE Transactions* 37(2):137–152.

Jeong I-J, Leon VJ (2002). Decision-making and cooperative interaction via coupling agents in organizationally distributed systems. *IIE Transactions* 34(9):789–802.

Jeong I-J, Yim S-B (2009). A job shop distributed scheduling based on Lagrangian relaxation to minimise total completion time. *International Journal of Production Research* 47(24):6783–6805.

Jeong K-C, Kim Y-D (1998). A real-time scheduling mechanism for a flexible manufacturing system: using simulation and dispatching rules. *International Journal of Production Research* 36(9):2609–2626.

Jiang S, Tang L (2008). Lagrangian relaxation algorithm for a single machine scheduling with release dates. In Q Zhou, J Luo (eds.), *Proc of the Int Symposium on Intelligent Information Technology Application*, Vol. 3:811–815. Los Alamitos:IEEE.

Johnson SM (1954). Optimal two- and three-stages production schedules with setup times includes. *Naval Research Logistics Quaterly* 1(4):281.

Kachitvichyanukul V, Sitthitham S (2011). A two-stage genetic algorithm for multi-objective job shop scheduling problems. *Journal of Intelligent Manufacturing* 22(3):355–365.

Kaskavelis CA, Caramanis MC (1998). Efficient Lagrangian relaxation algorithms for industry size job-shop scheduling problems. *IIE Transactions* 30(11):1085–1097.

Kaskavelis CA, Caramanis MC (1994). Application of a Lagrangian relaxation based scheduling algorithm to a semiconductor testing facility. In *Proceedings of the Fourth International Conference on Computer Integrated Manufacturing and Automation Technology*, 106–112.

Kennedy J, Eberhart R (1995). Particle swarm optimization. In *Proceedings of IEEE International Conference on Neural Networks*, Vol. 4:1942–1948.

Kim C-O, Min H-S, Yih Y (1998). Integration of inductive learning and neural networks for multi-objective FMS - scheduling. *International Journal of Production Research* 36(9):2497–2509.

Klemperer P (1999). Auction theory: A guide to the literature. *Journal of Economic Surveys* 13(3):227–286.

Koestler A (1967). *The ghost in the machine*. London: Hutchinson.

Kolonko M (1999). Some new results on simulated annealing applied to the job shop scheduling problem. *European Journal of Operational Research* 113(1):123–136.

Korf RE (1998). A complete anytime algorithm for number partitioning. *Artificial Intelligence* 106:181–203.

Kouiss K, Pierreval H, Mebarki N (1997). Using multi-agent architecture in FMS for dynamic scheduling. *Journal of Intelligent Manufacturing* 8(1):41–47.

Kreipl S (2000). A large step random walk for minimizing total weighted tardiness in a job shop. *Journal of Scheduling* 3(3):125–138.

Krishna V (2010). *Auction theory*. San Diego: Academic Press.

Kumara S, Lee Y, Chatterjee K (2002). Distributed multiproject resource control: A market-based approach. *CIRP Annals - Manufacturing Technology* 51(1):367–370.

Kutanoglu E, Wu SD (2004). Improving scheduling robustness via preprocessing and dynamic adaptation. *IIE Transactions* 36(11):1107–1124.

Kutanoglu E, Wu SD (1999). On combinatorial auction and Lagrangean relaxation for distributed resource scheduling. *IIE Transactions* 31(9):813–826.

Lawrence SR (1985). *Resource constrained project scheduling – A computational comparison of heuristic scheduling techniques*. Technical report. Pittsburgh: Carnegie Mellon University.

Lee J-H, Kim C-O (2008). Multi-agent systems applications in manufacturing systems and supply chain management: a review paper. *International Journal of Production Research* 46(1):236–265.

Lee Y-H, Kumara SR, Chatterjee K (2003). Multiagent based dynamic resource scheduling for distributed multiple projects using a market mechanism. *Journal of Intelligent Manufacturing* 14(5):471–484.

Leitão P (2009). Agent-based distributed manufacturing control: A state-of-the-art survey. *Engineering Applications of Artificial Intelligence* 22(7):979–991.

Leitão P (2004). *An agile and adaptive holonic architecture for manufacturing control*. Thesis. University of Porto.

Leitão P, Restivo FJ (2008). Implementation of a holonic control system in a flexible manufacturing system. *IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews* 38(5):699–709.

Lemaréchal C (2007). The omnipresence of Lagrange. *Annals of Operations Research* 153(1):9–27.

Leon V, Wu S, Storer R (1994ª). A game-theoretic control approach for job shops in the presence of disruptions. *International Journal of Production Research* 32(6):1451–1476.

Leon V, Wu S, Storer R (1994b). Robustness measures and robust scheduling for job shops. *IIE Transactions* 26(5):32–43.

Lian Z, Jiao B, Gu X (2006). A similar particle swarm optimization algorithm for job-shop scheduling to minimize makespan. *Applied Mathematics and Computation* 183(2):1008–1017.

Liang X, Huang M (2005). Application of tabu search-parallel genetic algorithm for job-shop scheduling. *Computer Integrated Manufacturing Systems,* 11(5):678–681.

Lim A, Tang J (2005). A Lagrangian heuristic for winner determination problem in combinatorial auctions. In S Zhang, R Jarvis (eds.), *AI 2005: Advances in Artificial Intelligence,* 736–745. Berlin: Springer.

Lin G, Solberg J (1994). An agent-based flexible routing manufacturing control simulation system. In *Proceedings of the Winter Simulation Conference*, 970–977.

Lin G, Solberg J (1992). Integrated shop floor control using autonomous agents. *IIE Transactions* 24(3):57–71.

Liu JS, Sycara KP (1998). Multi-agents co-ordination in tightly coupled task scheduling. In M Huhns, M Singh (eds.), *Readings in Agents,* 164–171. San Francisco: Morgan Kaufmann.

Liu N, Abdelrahman MA, Ramaswamy S (2007). A complete multiagent framework for robust and adaptable dynamic job shop scheduling. *IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews* 37(5):904–916.

Liu N, Abdelrahman MA, Ramaswamy S (2004). A multi-agent model for reactive job shop scheduling. In *Proceedings of the Thirty-Sixth Southeastern Symposium on System Theory*, 241–245.

Liu SQ, Ong HL, Ng KM (2005). Metaheuristics for minimizing the makespan of the dynamic shop scheduling problem. *Advances in Engineering Software* 36(3):199–205.

Lo C-C, Hsu C-C (1993). A parallel distributed processing technique for job-shop scheduling problems. In *Proceedings of International Joint Conference on Neural Networks,* Vol. 2:1602–1605.

Lorena LAN, Narciso MG (1996). Relaxation heuristics for a generalized assignment problem. *European Journal of Operational Research* 91(3):600–610.

Lou P, Liu Q, Zhou Z, Wang H, Sun X (2012). Multi-agent-based proactive–reactive scheduling for a job shop. *International Journal of Advanced Manufacturing Technology* 59(1-4):311–324.

Lu T-P, Yih Y (2001). An agent-based production control framework for multiple-line collaborative manufacturing. *International Journal of Production Research* 39(10):2155–2176.

Luck M, McBurney P, Shehory O, Willmott S (2005). *Agent technology: Computing as interaction (A roadmap for agent based computing)*. AgentLink Community.

Luh PB, Feng W (2003). From manufacturing scheduling to supply chain coordination: The control of complexity and uncertainty. *Journal of Systems Science and Systems Engineering* 12(3):279–297.

Luh PB, Gou L, Zhang Y, Nagahora T, Tsuji M, Yoneda K, Hasegawa T, Kyoya Y, Kano T (1998). Job shop scheduling with group-dependent setups, finite buffers, and long time horizon. *Annals of Operations Research* 76(0):233–259.

Luh PB, Hoitomt DJ (1993). Scheduling of manufacturing systems using the Lagrangian relaxation technique. *IEEE Transactions on Automatic Control* 38(7):1066–1079.

Luh PB, Hoitomt DJ, Max E, Pattipati KR (1990). Schedule generation and reconfiguration for parallel machines. *IEEE Transactions on Robotics and Automation* 6(6):687–696.

Luh PB, Liu F, Moser B (1999). Scheduling of design projects with uncertain number of iterations. *European Journal of Operational Research* 113(3):575–592.

Luh PB, Ni M, Chen H, Thakur LS (2003). Price-based approach for activity coordination in a supply network. *IEEE Transactions on Robotics and Automation* 19(2):335–346.

MacCarthy BL, Liu J (1993). Addressing the gap in scheduling research: a review of optimization and heuristic methods in production scheduling. *International Journal of Production Research* 31(1):59–79.

Manne AS (1960). On the job-shop scheduling problem. *Operations Research* 8(2):219–223.

Márkus A, Kis Váncza T, Monostori L (1996). A market approach to holonic manufacturing. *CIRP Annals - Manufacturing Technology* 45(1):433–436.

Masin M, Pasaogullari MO, Josh, S (2007). Dynamic scheduling of production-assembly networks in a distributed environment. *IIE Transactions* 39(4):395–409.

MASON (2012). *MASON Multiagent Simulation Toolkit* [www document]. URL http://cs.gmu.edu/~eclab/projects/mason/ (accessed 10.8.12).

Mattfeld DC, Bierwirth C (2004). An efficient genetic algorithm for job shop scheduling with tardiness objectives. *European Journal of Operational Research* 155(3):616–630.

Maturana FP, Norrie DH (1996). Multi-agent mediator architecture for distributed manufacturing. *Journal of Intelligent Manufacturing* 7(4):257–270.

McAfee RP, McMillan J (1987). Auctions and bidding. *Journal of Economic Literature* 25(2):699–738.

Mcguire JG, Kuokka DR, Weber JC, Tenenbaum JM, Gruber TR, Olsen GR (1993). SHADE: Technology for knowledge-based collaborative engineering. *Journal of Concurrent Engineering: Applications and Research* 1(3):137–146.

McMullen PR (2001). A Kohonen self-organizing map approach to addressing a multiple objective, mixed-model JIT sequencing problem. *International Journal of Production Economics* 72(1):59–71.

Mehta SV (1999). Predictable scheduling of a single machine subject to breakdowns. *International Journal of Computer Integrated Manufacturing* 12(1):15–38.

Mesghouni K, Pesin P, Trentesaux D, Hammadi S, Tahon C, Borne P (1999). Hybrid approach to decision-making for job-shop scheduling. *Production Planning & Control: The Management of Operations* 10(7):690–706.

Milgrom P (1989). Auctions and bidding: A primer. *Journal of Economic Perspectives* 3(3):3–22.

Miyashita K, Sycara K, Mizoguchi R (1996). Modeling ill-structured optimization tasks through cases. *Decision Support Systems* 17(4):345–364.

Monostori L, Váncza J, Kumara SRT (2006). Agent-based systems for manufacturing. *CIRP Annals - Manufacturing Technology* 55(2):697–720.

Muth JF, Thompson GL (1963). *Industrial scheduling*. Englewood Cliffs: Prentice Hall.

Netlogo (2012). *NetLogo Home Page* [www document]. URL http://ccl.northwestern.edu/netlogo/ (accessed 10.8.12).

Ni M, Luh PB, Moser B (2008). An optimization-based approach for design project scheduling. *IEEE Transactions on Automation Science and Engineering* 5(3):394–406.

Nishi T, Hiranaka Y, Inuiguchi M (2007). A successive Lagrangian relaxation method for solving flowshop scheduling problems with total weighted tardiness. In *IEEE International Conference on Automation Science and Engineering*, 875–880.

Nof SY, Weill R (1992). Collaborative coordination control (CCC) of distributed multimachine manufacturing. *CIRP Annals - Manufacturing Technology* 41(1):441–445.

Nowicki E, Smutnicki C (2005). An advanced tabu search algorithm for the job shop problem. *Journal of Scheduling* 8(2):145–159.

Nowicki E, Smutnicki C (1996). A fast taboo search algorithm for the job shop problem. *Management Science* 42(6):797–813.

Nwana HS (1996). Software agents: an overview. *The Knowledge Engineering Review* 11(3):205–244.

Odel J, Parunak HVD, Bauer B (2001). Extending UML for agents. In G Wagner, Y Lesperance, E Yu (eds.), *Proc of the Agent-Oriented Information Systems Workshop at the 17th National Conference on Artificial Intelligence*, 3–17.

OR-LIBRARY (nd). *Job Shop Project* [www document]. URL http://web.cecs.pdx.edu/~bart/cs510ss/project/jobshop/index.html (accessed 7.28.09).

Ouelhadj D, Petrovic S (2009). A survey of dynamic scheduling in manufacturing systems. *Journal of Scheduling* 12(4):417–431.

Papadimitriou CH, Steiglitz K (1998). *Combinatorial optimization: Algorithms and complexity*. New York: Dover Publications.

Park BJ, Choi HR, Kim HS (2003). A hybrid genetic algorithm for the job shop scheduling problems. *Computers & Industrial Engineering* 45(4):597–613.

Park H, Cutkosky MR, Concru AB, Lee S-H (1994). An agent-based approach to concurrent cable harness design. Artificial Intelligence for Engineering, Design, Analysis and Manufacturing 8(01):45–61.

Parkes DC (1999). iBundle: An efficient scending price bundle auction. In *Proceedings of the 1st ACM Conference on Electronic Commerce*, 148–157.

Parkes DC, Ungar LH (2001). An auction-based method for decentralized train scheduling. In Proceedings of the Fifth International Conference on Autonomous Agents, 43–50.

Parunak H (1996). Applications of distributed artificial intelligence in industry. *Foundations of distributed artificial intelligence* 4:139–164.

Parunak H (1987). Manufacturing experience with the contract net. In MN Huhns (ed.), *Distributed Artificial Intelligence*, 285–310. London: Pitman.

Parunak H, Baker AD, Clark SJ (2001). AARIA agent architecture: from manufacturing requirements to agent-based system design. *Integrated Computer-Aided Engineering* 8(1):45–58.

Parunak H, Sauter J, Fleischer M, Ward A (1999). The RAPPID Project: Symbiosis between industrial requirements and MAS research. *Autonomous Agents and Multi-Agent Systems* 2(2):111–140.

Pavón J, Gómez-Sanz JJ, Fuentes-Fernández R (2005). The INGENIAS Methodology and Tools. In B Henderson-Sellers, P Giorgini (eds.), *Agent-Oriented Methodologies*, 236–276. London:Idea Group Pub.

Pérez E (2000). *Análisis de algoritmos genéticos como método de resolución para problemas de programación de operaciones: Caso job shop*. Thesis. University of Valladolid.

Pezzella F, Merelli E (2000). Tabu search method guided by shifting bottleneck for the job shop scheduling problem. *European Journal of Operational Research* 120(2):297–310.

Pinedo ML (2009). *Planning and scheduling in manufacturing and services*, 2nd ed. New York: Springer.

Pinedo ML (2008). *Scheduling. Theory, Algorithms, and Systems*, 2nd ed. New York: Springer.

Pinedo ML, Singer M (1999). A shifting bottleneck heuristic for minimizing the total weighted tardiness in a job shop. *Naval Research Logistics* 46(1):1–17.

Pokahr A, Braubach L, Lamersdorf W (2005). Jadex: A BDI reasoning engine. In R Bordini, M Dastani, J Dix, A El Fallah (eds.), *Multi-Agent Programming*, 149–174. New York: Springer.

Ponnambalam SG, Jawahar N, Aravindan P (1999). A simulated annealing algorithm for job shop scheduling. *Production Planning and Control* 10(8):767–777.

Porter D, Rassenti S, Roopnarine A, Smith V (2003). Combinatorial auction design. *Proceedings of the National Academy of Sciences of the USA* 100(1):11153–11157.

Qi JG, Burns GR, Harrison DK (2000). The application of parallel multipopulation genetic algorithms to dynamic job-shop scheduling. *International Journal of Advanced Manufacturing Technology* 16(8):609–615.

Raheja AS, Subramaniam V (2002). Reactive recovery of job shop schedules – A review. *Int J of Advanced Manufacturing Technology* 19(10):756–763.

Rajabinasab A, Mansour S (2011). Dynamic flexible job shop scheduling with alternative process plans: an agent-based approach. *Int J Advanced Manufacturing Technology* 54(9-12):1091–1107.

Rajendran C, Holthaus O (1999). A comparative study of dispatching rules in dynamic flowshops and jobshops. *European Journal of Operational Research* 116(1):156–170.

Rajkumar M, Asokan P, Anilkumar N, Page T (2011). A GRASP algorithm for flexible job-shop scheduling problem with limited resource constraints. *International Journal of Production Research* 49(8):2409–2423.

Raman N, Talbot FB (1993). The job shop tardiness problem: A decomposition approach. *European Journal of Operational Research* 69(2):187–199.

Ramasesh R (1990). Dynamic job shop scheduling: a survey of simulation research. *Omega* 18(1):43–57.

Ramos C (1994). An architecture and a negotiation protocol for the dynamic scheduling of manufacturing systems. In *Proceedings of the IEEE International Conference on Robotics and Automation*, Vol. 4:3161–3166.

Rassenti S, Smith V, Bulfin R (1982). A combinatorial auction mechanism for airport time slot allocation. *The Bell Journal of Economics* 13(2):402–417.

Repast Suite (2012). *Repast Suite* [www document]. URL http://repast.sourceforge.net/ (accessed 10.8.12).

Resende MG, Ribeiro CC (2003). Greedy randomized adaptive search procedures. In F Glover, G Kochenberger (eds.), *Handbook of Metaheuristics*, 219–249. Boston: Kluwer.

Richter MK, Wong K-C (1999). Non-computability of competitive equilibrium. *Economic Theory* 14(1):1–27.

Ross R, Collier R, O'Hare GMP (2005). AF-APL–bridging principles & practice in agent oriented languages. In *Proceedings of the Second International Conference on Programming Multi-Agent Systems*, 66–88.

Rossi A, Dini G (2000). Dynamic scheduling of FMS -using a real-time genetic algorithm. *International Journal of Production Research* 38(1):1–20.

Roy B, Sussmann B (1964). *Les problemes d'ordonnancement avec contrintes disjontives*. Tehnical Report DS No. 9. SEMA. Montrouge.

Ryu K, Jung M (2003). Agent-based fractal architecture and modelling for developing distributed manufacturing systems. *International Journal of Production Research* 41(17):4233–4255.

Sadeh NM, Hildum DW, Kjenstad D (2003). Agent-based e-supply chain decision support. *J of Organizational Computing and Electronic Commerce* 13:225–241.

Sandholm T (2000). Approaches to winner determination in combinatorial auctions. *Decision Support Systems* 28(1-2):165–176.

Sanz P (2008). *Selección de socios en las empresas virtuales dinámicas*. Thesis. University of Valladolid.

Satake T, Morikawa K, Nakamura N (1994). Neural network approach for minimizing the makespan of the general job-shop. *Int J of Production Economics* 33(1-3):67–74.

Scrich CR, Armentano VA, Laguna M (2004). Tardiness minimization in a flexible job shop: A tabu search approach. *J of Intelligent Manufacturing* 15(1):103–115.

Sha DY, Hsu C-Y (2006). A hybrid particle swarm optimization for job shop scheduling problem. *Computers & Industrial Engineering* 51(4):791–808.

Shaw MJ, Park S, Raman N (1992). Intelligent scheduling with machine learning capabilities: The induction of scheduling knowledge. *IIE Transactions* 24(2):156–168.

Shaw MJ (1988). Dynamic scheduling in cellular manufacturing systems: A framework for networked decision making. *J of Manufacturing Systems* 7(2):83–94.

Shen W (2002). Distributed manufacturing scheduling using intelligent agents. *Intelligent Systems* 17(1):88–94.

Shen W, Hao Q, Yoon HJ, Norrie DH (2006). Applications of agent-based systems in intelligent manufacturing: An updated review. *Advanced Engineering Informatics* 20(4):415–431.

Shen W, Maturana F, Norrie DH (2000). MetaMorph II: an agent-based architecture for distributed intelligent design and manufacturing. *Journal of Intelligent Manufacturing* 11(3):237–251.

Shen W, Norrie DH, Barthes J-P (2001). *Multi-agent systems for concurrent intelligent design and manufacturing*. London: Taylor & Francis.

Shnits B, Rubinovitz J, Sinreich D (2004). Multicriteria dynamic scheduling methodology for controlling a flexible manufacturing system. *International Journal of Production Research* 42(17):3457–3472.

Singer M, Pinedo M (1998). A computational study of branch and bound techniques for minimizing the total weighted tardiness in job shops. *IIE Transactions* 30(2):109–118.

Siwamogsatham T, Saygin C (2004). Auction-based distributed scheduling and control scheme for flexible manufacturing systems. *International Journal of Production Research* 42(3):547–572.

Smith RG, Davis R (1981). Frameworks for cooperation in distributed problem solving. *IEEE Transactions on Systems, Man and Cybernetics* 11(1):61–70.

Smith RG (1980). The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on Computers* C-29(12):1104–1113.

Sun J, Zhang YF, Nee AYC (2001). A distributed multi-agent environment for product design and manufacturing planning. *International Journal of Production Research* 39(4):625–645.

Sun T, Luh PB, Liu M, 2006. Lagrangian relaxation for complex job shop scheduling. In *Proc of the IEEE Int Conf on Robotics and Automation*, 1432–1437.

Sun X, Noble JS, Klein CM (1999). Single-machine scheduling with sequence dependent setup to minimize total weighted squared tardiness. *IIE Transactions* 31(2):113–124.

Swaminathan JM, Smith SF, Sadeh NM (1998). Modeling supply chain dynamics: A multiagent approach. *Decision Sciences* 29(3):607–632.

Swarm (2012). *SwarmWiki* [www document]. URL http://www.swarm.org (accessed 10.8.12).

Tanev IT, Uozumi T, Morotome Y (2004). Hybrid evolutionary algorithm-based real-world flexible job shop scheduling problem: application service provider approach. *Applied Soft Computing* 5(1):87–100.

Tang L, Xuan H, Liu J (2007). Hybrid backward and forward dynamic programming based Lagrangian relaxation for single machine scheduling. *Computers & Operations Research* 34(9):2625–2636.

Tang L, Xuan H, Liu J (2006). A new Lagrangian relaxation algorithm for hybrid flowshop scheduling to minimize total weighted completion time. *Computers & Operations Research* 33(11):3344–3359.

Tang L, Zhang Y (2009). Parallel machine scheduling under the disruption of machine breakdown. *Industrial and Engineering Chemistry Research* 48(14):6660–6667.

Tempelmeier H, Derstroff M (1996). A Lagrangean-based heuristic for dynamic multilevel multiitem constrained lotsizing with setup times. *Management Science* 42(5):738–757.

Tharumarajah A (1996). Comparison of the bionic, fractal and holonic manufacturing system concepts. Int J of Computer Integrated Manufacturing 9(3):217–226.

Tiwari MK, Allada V (2004). Solving the machine-loading problem in a flexible manufacturing system using a combinatorial auction-based approach. *International Journal of Production Research* 42(9):1879–1893.

Tiwari MK, Jha SK, Anand RB (2010). Operation allocation and part type selection in E-manufacturing: An auction based heuristic supported by agent technology. *Robotics and Computer-Integrated Manufacturing* 26(4):312–324.

Tomastik RN, Luh PB (1993). The facet ascending algorithm for integer programming problems. In *Proc of the 32nd IEEE Conf on Decision and Control*, 2880–2884.

Toptal A, Sabuncuoglu I (2009). Distributed scheduling: a review of concepts and applications. *International Journal of Production Research* 48(18):5235–5262.

Tovey CA (2002). Tutorial on computational complexity. *Interfaces* 32(3):30–61.

Trentesaux D (2009). Distributed control of production systems. *Engineering Applications of Artificial Intelligence* 22(7):971–978.

Valckenaers P, Bonneville F, Van Brussel H, Bongaerts L, Wyns J (1994). Results of the holonic control system benchmark at KU Leuven. In *Proceedings of the Fourth International Conference on Computer Integrated Manufacturing and Automation Technology*, 128–133.

Valckenaers P, Van Brussel H (2005). Holonic manufacturing execution systems. CIRP Annals - Manufacturing Technology 54(1):427–432.

Vallejo D, Albusac J, Mateos JC, González-Morcillo C, Jiménez L (2010). A modern approach to multiagent development. Journal of Systems and Software 83(3):467–484.

Van Brussel H, Wyns J, Valckenaers P, Bongaerts L, Peeters P (1998). Reference architecture for holonic manufacturing systems: PROSA. *Computers in Industry* 37(3):255–274.

Vepsalainen APJ, Morton TE (1987). Priority rules for job shops with weighted tardiness costs. *Management Science* 33(8):1035–1047.

Vollmann T, Berry W, Whybark C (1997). *Manufacturing planning and control systems*. McGraw-Hill.

Vytelingum P, Rogers A, Macbeth DK, Dutta P, Stranjak A, Jennings NR (2009). A market-based approach to multi-factory scheduling. In S Das, M Ostrovsky, D Pennock, BK Szymanski (eds.), *Auctions, Market Mechanisms and Their Applications*, 74–86. Berlin: Springer.

Wallace A (2003). Sequential resource allocation utilizing agents. *International Journal of Production Research* 41(11):2481–2499.

Wang C-Q, Cao Y-F, Dai G-Z (2004). Bi-directional convergence ACO for job-shop scheduling. Computer Integrated Manufacturing Systems 10(7):820–824.

Wang J, Luh PB, Zhao X, Wang J (1997). An optimization-based algorithm for job shop scheduling. *Sadhana* 22(part 2):241–256.

Wang L, Brennan RW, Balasubramanian S, Norrie DH (2001). Realizing holonic control with function blocks. *Integrated Computer-Aided Engineering* 8(1):81–93.

Wang S-H (2003). An improved stepsize of the subgradient algorithm for solving the lagrangian relaxation problem. *Computers & Electrical Engineering* 29(1):245–249.

Wang T-Y, Wu K-B (2000). A revised simulated annealing algorithm for obtaining the minimum total tardiness in job shop scheduling problems. *International Journal of Systems Science* 31(4):537–542.

Watanabe M, Ida K, Gen M (2005). A genetic algorithm with modified crossover operator and search area adaptation for the job-shop scheduling problem. *Computers & Industrial Engineering* 48(4):743–752.

Weiss G (1999). *Multiagent systems: a modern approach to distributed artificial intelligence*. Cambridge: MIT Press.

Wellman MP (1993). A market-oriented programming environment and its application to distributed multicommodity flow problems. *Journal of Artificial Intelligence Research* 1:1–23.

Wellman MP, Walsh WE, Wurman PR, MacKie-Mason JK (2001). Auction protocols for decentralized scheduling. *Games and Economic Behavior* 35:271–303.

Winikoff M (2005). Jack™ Intelligent agents: An industrial strength platform. In R Bordini, D Mehdi, A El Fallah (eds.), *Multi-Agent Programming: Languages, Platforms and Applicatgions*, 175–193. New York: Springer.

Wolsey LA, Nemhauser GL (1999). *Integer and Combinatorial Optimization*, 1st ed. New York: Wiley.

Womack, JP, Jones DT, Roos D (2007). *The machine that changed the world*. Free Press.

Wooldridge M (2002). *An Introduction to MultiAgent Systems*. London: Wiley.

Wooldridge M, Jennings N (1995). Intelligent agents: Theory and practice. *The Knowledge Engineering Review* 10(02):115–152.

Wooldridge M, Jennings NR, Kinny D (2000). The Gaia methodology for agent-oriented analysis and design. *Autonomous Agents and Multi-Agent Systems* 3(3):285–312.

Wu B, Cheng T, Yang S, Zhang Z (2003). Price-based negotiation for task assignment in a distributed network manufacturing mode environment. *The International Journal of Advanced Manufacturing Technology* 21(2):145–156.

Wu D-W, Lu T-D, Liu X-B, Meng Y-S (2005). Parallel simulated annealing algorithm for solving job-shop scheduling problem. Computer Integrated Manufacturing Systems 11(6):847–850.

Wu S, Byeon ES, Storer RH (1999). A graph-theoretic decomposition of the job shop scheduling problem to achieve scheduling robustness. Operations Research 47(1):113–124.

Wu S, Wysk R (1989). An application of discrete-event simulation to on-line control and scheduling in flexible manufacturing. *International Journal of Production Research* 27(9):1603–1623.

Xia M, Koehler G, Whinston A (2004). Pricing combinatorial auctions. *European Journal of Operational Research* 154(1):251–270.

Xia W, Wu Z (2005). A hybrid particle swarm optimization approach for the job-shop scheduling problem. *Int J Advanced Manufacturing Technology* 29(3-4):360–366.

Xiang W, Lee HP (2008). Ant colony intelligence in multi-agent dynamic manufacturing scheduling. *Engineering Applications of Artificial Intelligence* 21(1):73–85.

Yamamoto M, Nof SY (1985). Scheduling/rescheduling in the manufacturing operating system environment. Int J of Production Research 23(4):705–722.

Yen BP-C, Wu OQ (2004). Internet scheduling environment with market-driven agents. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans* 34:281–289.

Zhang Y, Luh PB, Narimatsu K, Moriya T, Shimada T, Fang L (2001). A macro-level scheduling method using Lagrangian relaxation. *IEEE Transactions on Robotics and Automation* 17(1):70–79.

Zhang Y, Luh PB, Yoneda K, Kano T, Kyoya Y (2000). Mixed-model assembly line scheduling using the Lagrangian relaxation technique. *IIE Transactions* 32(2):125–134.

Zhao X, Luh PB, Wang J (1999). Surrogate gradient algorithm for Lagrangian relaxation. *Journal of Optimization Theory and Applications* 100(3):699–712.

Zhao X, Luh P, Wang J (1997). The surrogate gradient algorithm for Lagrangian relaxation method. In *Proceedings of the 36th IEEE Conference on Decision and Control*, 305–310.

Zhou H, Cheung W, Leung L (2009a). Minimizing weighted tardiness of job-shop scheduling using a hybrid genetic algorithm. *European Journal of Operational Research* 194(3):637–649.

Zhou R, Nee AYC, Lee HP (2009b). Performance of an ant colony optimisation algorithm in dynamic job shop scheduling problems. *International Journal of Production Research* 47(11):2903–2920.