Curso 1995/96
CIENCIAS Y TECNOLOGÍAS

DAVID ALCAIDE LÓPEZ DE PABLO

Problemas de planificación y secuenciación determinística: modelización y técnicas de resolución

Director
JOAQUÍN SICILIA RODRÍGUEZ

SERVICIO DE PUBLICACIONES
UNIVERSIDAD DE LA LAGUNA

SOPORTES AUDIOVISUALES E INFORMÁTICOS
Serie Tesis Doctorales

A mi familia

Agradecimientos

El presente trabajo ha sido realizado bajo la dirección del Profesor Dr. D. Joaquín Sicilia Rodríguez, a quien quiero expresar mi más sincero agradecimiento por su apoyo, estímulo y permanente disponibilidad y dedicación.

También quiero agradecer a los Profesores María Teresa Ramos Domínguez, Bruno Simeone, Paolo Toth, y al mismo Joaquín Sicilia Rodríguez, el hecho de que hayan promovido un programa interuniversitario de colaboración del que tanto me he beneficiado académica y personalmente.

Agradezco en modo particular al Profesor Carlos González Martín por sus acertados comentarios y sugerencias, al Profesor Paolo Toth por su cercanía y útiles orientaciones, al Profesor Daniele Vigo, amigo y colaborador en mis estancias en Bolonia, y al Profesor Jorge Riera Ledesma por su colaboración desinteresada en ciertas fases de la programación de los algoritmos.

Agradezco también a los compañeros del Departamento de Estadística, Investigación Operativa y Computación de la Universidad de La Laguna, y a los del Departamento de Electrónica, Informática y Sistemas de la Universidad de Bolonia por su apoyo y colaboración.

Gracias a todos aquellos que de alguna u otra manera me han ayudado.

David Alcaide López de Pablo

Indice

Prólogo

Prólogo

Al amparo de la Investigación Operativa, se han desarrollado diversas disciplinas orientadas a optimizar la utilización de los escasos recursos disponibles, para lograr la máxima eficacia en la distribución de los mismos.

Una de estas disciplinas, conocida como Planificación y Secuenciación (*"Scheduling and Sequencing"*) constituye un campo de trabajo y de investigación que intenta resolver cuestiones relativas a la planificación de la producción, mantenimiento y reparación de productos, control y gestión de máquinas, etc.

Los problemas de planificación ó "scheduling" siempre han existido, y su estudio científico ha experimentado un auge después de la II Guerra Mundial, coincidiendo con el desarrollo de los modelos de Investigación Operativa. Así, a partir de los años cincuenta, aparecen trabajos que resuelven problemas sencillos, hoy considerados clásicos, y que sirven de partida a modelos posteriores. Entre ellos, se pueden destacar los trabajos de Johnson (1954) y Jackson (1955), para optimizar líneas de producción; Smith (1956), para minimizar el tiempo medio de permanencia de los trabajos en un taller, (problema equivalente a minimizar el número medio de trabajos en el taller); McNaugthon (1959), para minimizar el tiempo total de proceso de trabajos interrumpibles en máquinas idénticas, Moore y Hodgson (1968), para minimizar el número de trabajos tardíos; Emmons (1969), que propone propiedades de dominancia entre soluciones para un problema de complejidad máxima como es el problema de la tardanza total; ...

En un sentido amplio, el término *Scheduling* puede entenderse como la asignación de máquinas o procesadores a lo largo del tiempo para realizar un conjunto de trabajos (Baker (1974)), o bien como resolver el problema de encontrar la asignación temporal óptima de ciertos recursos a determinadas tareas (Lawler, Lenstra, Rinnooy Kan y Shmoys (1993)).

Por tanto, en un problema de planificación siempre existirán tres componentes diferenciadas: las tareas ó actividades que se pretenden realizar, los recursos disponibles para su realización, y las finalidades u objetivos que se desean lograr y que nos permiten identificar, entre varias planificaciones posibles, aquellas que sean óptimas. Sobre este simple esquema de tres componentes resulta que muchos problemas reales en los ámbitos industrial, comercial, social, científico, e incluso de la vida cotidiana, admiten una aproximación que se ajusta a un modelo de planificación.

Cuando todos los datos del problema de planificación son conocidos a priori, el modelo se denomina determinístico. Estos modelos son estudiados por la Optimización Combinatoria. Este área de la Investigación Operativa estudia también otros modelos asociados a problemas en los que, por ejemplo, debe

determinarse una ordenación, selección ó asignación óptima en un conjunto finito de objetos. Una característica común a la mayoría de los problemas estudiados por la Optimización Combinatoria es que son relativamente "fáciles" de plantear pero difíciles de modelizar y, consecuentemente, mucho más difíciles de resolver.

Esta propiedad es especialmente frecuente en los problemas de Planificación y Secuenciación de tareas, por tanto, su resolución exige el uso de medios de computación adecuados. Los correspondientes aspectos computacionales se estudian analizando la complejidad computacional de los métodos propuestos.

De hecho, en los últimos años, se ha observado un incremento de las aplicaciones e interrelaciones entre la Planificación y Secuenciación de tareas y las Ciencias de la Computación. En este sentido, merece especial atención el análisis de la complejidad computacional de problemas combinatorios y las implicaciones resultantes para el diseño y análisis de algoritmos adecuados. Se acepta comúnmente que un problema esta bien resuelto o es fácil si se puede resolver por un algoritmo cuyo tiempo de ejecución esté acotado por una función polinomial en el tamaño del problema (Lawler (1976a)). En la mayoría de problemas de Optimización Combinatoria no se tiene conocimiento de la existencia de tal algoritmo. Entonces surge la cuestión de probar si el problema es NP-duro o puede resolverse en tiempo polinomial. (Karp (1972), Garey y Johnson (1979)). Estos conceptos complementarios son muy útiles en el análisis de problemas de planificación, donde hay cierto número de ellos que no han sido aún clasificados como pertenecientes a una o a otra categoría.

Si el problema es NP-duro, como ocurre en la mayoría de los problemas prácticos, se pueden adoptar dos aproximaciones diferentes a la solución.

La primera de ellas consiste en elegir algún método exacto para resolver el problema. Entre los métodos exactos podemos citar la programación dinámica ó las técnicas de ramificación y acotación. Estos métodos exactos requerirán con frecuencia un tiempo de búsqueda de la solución invariablemente exponencial.

La segunda opción consiste en considerar un método heurístico rápido para encontrar una solución aproximada. Posteriormente se realizaría un análisis comparativo de contraste de la calidad de la solución obtenida. Esta opción está especialmente justificada en muchos de los problemas de Planificación y Secuenciación, debido a la elevada complejidad que presentan la mayoría de ellos.

La presente memoria esta estructurada en cinco capítulos y tres apéndices finales. Los diferentes capítulos van analizando y estudiando los distintos problemas y proponiendo algoritmos para su resolución.

En el primer capítulo se realiza un planteamiento general de los problemas de planificación y se describen las características de los diferentes problemas. También se presentan conceptos generales sobre complejidad computacional y la reducibilidad computacional entre problemas de planificación unicriterio. El capítulo finaliza extendiendo el planteamiento al caso multicriterio.

El capítulo II se centra en los problemas de planificación sobre una máquina o problemas de secuenciación. Tras realizar una clasificación computacional de los diferentes problemas, se estudia una familia de problemas NP-duros y se proponen algoritmos heurísticos para su resolución. Estos algoritmos, que utilizan técnicas constructivas y técnicas de búsqueda de vecinos, convergen a soluciones relativamente buenas en tiempos aceptables.

Los problemas de la familia considerada tienen como objetivo minimizar la tardanza total. Así, se proponen sucesivamente algoritmos para el problema $1||\sum T_j$, en el que los trabajos no son interrumpibles y están disponibles desde el instante inicial; para $1|r_j|\sum T_j$, donde aparecen diferentes fechas de disponibilidad de los trabajos; para $1|r_j, pmtn|\sum T_j$, con trabajos interrumpibles; para $1|r_j, pmtn(A)|\sum T_j$, donde unos trabajos son interrumpibles y otros no; y para $1|prec|\sum T_j$, con relaciones de precedencia entre los trabajos.

Los problemas de planificación sobre varias máquinas son considerados en el capítulo III. Por su naturaleza los problemas se distinguen en función de que las máquinas sean o no capaces de realizar cualquier trabajo que se considere. Se analiza la complejidad de los diferentes problemas de planificación que surgen, y se proponen nuevos algoritmos para resolver determinados problemas.

El problema $P||C_{max}$ es formulado como un problema de programación entera 0-1. Los algoritmos propuestos utilizan técnicas de "backtracking" para obtener soluciones exactas y técnicas constructivas para obtener soluciones heurísticas.

Para el problema $P|prec|\sum T_j$ se proponen algoritmos heurísticos que se apoyan en la detección de caminos críticos del grafo de precedencias, en técnicas de búsqueda de vecinos, y en técnicas constructivas para proponer soluciones. Bajo ciertas condiciones de los datos de entrada, se puede garantizar que algunas de estas técnicas son de óptimo.

Para el problema $O||C_{max}$ se proponen heurísticas constructivas sencillas para obtener soluciones rápidamente. También se proponen rutinas simples capaces de mejorar una solución dada. Se propone un modelo de grafo disyuntivo para el problema $O||C_{max}$. A partir de dicho modelo, y diseñando las diversas componentes, se construye un algoritmo de búsqueda tabú para resolver el problema.

En algunos casos, los modelos bicriterio pueden aproximarse mejor a ciertos problemas reales que los modelos unicriterio. Así, el capítulo IV estudia y analiza problemas de planificación bicriterio. Tras plantear diversos problemas de planificación bicriterio y catalogar computacionalmente algunos de ellos se presentan diversos algoritmos para su resolución. En ocasiones, los problemas son reducidos a problemas unicriterio, mientras que en otros los algoritmos tienen una naturaleza constructiva.

En el capítulo V, se estudian diversas formulaciones de problemas de planificación mediante la Programación Matemática, realizándose una modelización y formalización de un problema concreto que se presenta en un taller de reparaciones.

Finalmente, en los tres apéndices finales se recoge la experiencia computacional desarrollada.

Capítulo I

Capítulo I: Generalidades sobre planificación.

1. Planificación y secuenciación: preliminares

Bajo el nombre genérico de *Planificación y Secuenciación* se recogen un conjunto de modelos y técnicas de Investigación Operativa que permiten resolver muchos de los problemas surgidos en diversos ámbitos. Entre ellos, podemos citar la industria, el comercio, las actividades financieras, la sanidad, sectores administrativos y de gestión tanto públicos como privados, etc. En todos ellos surgen situaciones en las que se precisa describir la manera óptima de asignar recursos escasos a diferentes actividades a lo largo de un período de tiempo. Los primeros modelos relativos a planificación y secuenciación de tareas comenzaron a desarrollarse en los años 50 y, desde entonces, han aparecido en la literatura especializada multitud de trabajos, comunicaciones y estudios presentando nuevos e interesantes resultados. Referencias básicas son, entre otras, los libros de Baker (1974) y French (1982), y los excelentes artículos de Graham y otros (1979), Lawler y otros (1982, 1993).

Este área de investigación se caracteriza por considerar un conjunto amplio y variado de problemas reales que pueden catalogarse y estudiarse en base a diferentes parámetros que configuran las distintas posibilidades que pueden presentarse en la práctica. La mayor parte de la investigación sobre estos problemas se ha centrado principalmente en los problemas de *planificación determinística* para los cuales se han desarrollado muchos modelos con sus respectivas técnicas de solución pero donde siguen apareciendo continuamente nuevos métodos y procedimientos que van mejorando las técnicas existentes. Estos modelos están basados en ciertas suposiciones que se comentan a continuación.

La primera suposición hace referencia a los recursos y a las actividades. Una *máquina* es un recurso que puede ejecutar en cada instante de tiempo una única actividad. Las actividades también se denominan *trabajos*. Se admite que, en cada instante de tiempo, cada trabajo se procesa en una única máquina. Suprimiendo esta condición, es decir, permitiendo que en un instante de tiempo un recurso sirva a varios trabajos y un trabajo use varios recursos, entraríamos en los modelos de *planificación con restricciones sobre los recursos ("resource-constrained project scheduling")*.

La segunda consideración hace referencia a la naturaleza determinística de los problemas. Toda la información que define la entrada del problema se conoce con certeza, y dado que el número de posibles planificaciones es finito, la Planificación Determinística es una parte específica de la *Optimización Combinatoria*. Por tanto, muchas de las técnicas de Optimización Combinatoria

pueden aplicarse a problemas de planificación. Una extensión natural de estos modelos consiste en asumir que ciertos datos del problema varían aleatoriamente, y, de esa forma, aparecerían los problemas de *Planificación Estocástica ("stochastic machine scheduling")*.

Ahora bien, dada la amplia variedad de problemas en *Planificación Determinística ("deterministic machine scheduling")*, hemos dirigido la atención de la presente memoria sólo a dicha área, aunque esperamos en un futuro ampliar nuestro campo de actuación a otras áreas de planificación.

La mayor parte de los modelos de planificación determinística estudiados en la literatura se restringen a la minimización de un único criterio de optimalidad. Generalmente, dicho criterio es una función regular, es decir, no decreciente en cada uno de los tiempos de completación de los trabajos, entendiéndose como tiempo de completación de un trabajo el instante en que su ejecución concluye. Esto excluye modelos en los que se consideran funciones no regulares como la prontitud (diferencia entre la fecha límite permitida para finalizar la ejecución del trabajo y el instante de completación). También es posible extender los modelos de planificación al caso en que se optimicen varios criterios simultáneamente. Estos modelos se agrupan en la *Planificación Multicriterio*, que constituye un área relativamente inexplorada.

Las aplicaciones prácticas del estudio de problemas de planificación en la vida real son numerosas ya que tareas y máquinas pueden representar una amplia variedad de casos de estructuras diversas: barcos y diques, clases y profesores, pacientes y equipo hospitalario, juicios y jueces, cenas y cocineros, programas de ordenador y procesadores, etc.

La amplia variedad de problemas existentes convierte quizás, a la Teoría de Planificación y Secuenciación, en un estudio de modelos más que en un estudio de metodologías. Así, si bien en el siguiente epígrafe se hace un análisis exhaustivo de los diferentes modelos, podemos adelantar a modo de resumen las distintas posibilidades a tener en cuenta en tal clasificación.

Los problemas de planificación y secuenciación pueden dividirse en problemas sobre una única máquina y problemas sobre varias máquinas. En los problemas sobre varias máquinas pueden distinguirse aquellos sobre máquinas no especializadas de aquellos sobre máquinas especializadas. En el primer caso todas las máquinas son capaces de ejecutar cualquier actividad, mientras que, en el segundo caso, las máquinas se especializan en ciertas actividades.

Los problemas sobre máquinas no especializadas se dividen a su vez en problemas sobre máquinas idénticas, uniformes o no relacionadas en función de las velocidades de proceso de las máquinas.

Por otro lado, los problemas sobre máquinas especializadas más generales se denominan problemas open-shop, flow-shop y job-shop. En estos problemas, cada trabajo se descompone en varias operaciones que deben acoplarse por las diferentes máquinas especializadas. La distinción entre estos problemas se hace atendiendo a las trayectorias de máquinas que deben seguir los trabajos.

Veremos, a lo largo de los siguientes epígrafes, la formulación y catalogación de los problemas de planificación unicriterio y multicriterio. En los siguientes capítulos realizaremos un estudio pormenorizado de algunos de los problemas formulados.

2. Formulación de problemas unicriterio

Los problemas de *"scheduling"* ó *planificación* pueden, en la mayoría de los casos, modelizarse de la siguiente manera: se precisan realizar n trabajos, tareas o procesos J_j $(j = 1, ..., n)$ para lo que se dispone de m máquinas o procesadores M_i $(i = 1, ..., m)$. Siempre que no haya lugar a confusión se hablará de "trabajo j" en lugar de J_j y de "máquina i" en vez de M_i . Se supone que cada máquina es incapaz de procesar varios trabajos simultáneamente y que, en un instante dado, cada trabajo puede realizarse en a lo sumo una máquina.

Obviamente, diferentes características de los trabajos y de las máquinas junto con distintos criterios de optimalidad, originan una gran variedad de problemas de planificación.

Antes de continuar adelante conviene catalogar los diferentes problemas de planificación. La clasificación de problemas de planificación puede contemplarse atendiendo a tres campos $\alpha|\beta|\gamma$. En el primer parámetro se recogen las características de las m máquinas o procesadores M_i $(i = 1, ..., m)$; en el segundo las de los n trabajos o tareas a procesar J_j $(j = 1, ..., n)$; y el último indica los criterios y el modo de optimización considerados.

Cada trabajo J_j tiene asociado los siguientes datos:

- Un número m_j de operaciones O_{1j} , O_{2j} , ..., $O_{m_j j}$ en las que puede dividirse el trabajo J_j de manera que, fijada la planificación, cada operación se procesa en una única máquina. Podemos definir $\mu_{ij} = k$ si la operación O_{ij} debe realizarse en la máquina M_k . Si el trabajo j consta de una única operación $(m_j = 1)$ podemos denotar con $\mu_j = k$ el hecho de que dicha operación deba asignarse a la máquina M_k .

- Uno ó más tiempos de procesamiento p_j ó p_{ij} *(i = 1,...,m)* necesarios para procesar el trabajo J_j en cualquier máquina *($p_{ij} = p_j \ \forall i$)* ó en la máquina M_i *(i = 1, ..., m)*.

- Una fecha de disponibilidad r_j , a partir de la cual puede comenzar a procesarse. Si $r_j = 0 \quad \forall j$, todos los trabajos están disponibles desde el mismo instante de tiempo y estaremos ante problemas de planificación *estáticos*, mientras que en caso de distintas fechas de disponibilidad, nos encontraremos ante problemas de planificación *dinámicos.*

- Una fecha de comienzo S_j , que no es constante, sino que depende de la planificación, que indica el instante de tiempo en el que comienza el procesamiento del trabajo J_j .

- Una fecha límite o de vencimiento d_j , en la que el trabajo J_j debería estar terminado para no incurrir en tardanza.

- Un peso o ponderación ω^k_j que indica la importancia relativa de J_j con respecto a los otros trabajos en el criterio k con $1 \leq k \leq K$, siendo K el número de criterios considerados.

- Una función de costo real f^k_j por cada criterio k con $1 \leq k \leq K$, donde $f^k_j(t)$ es el coste asociado, según el criterio k, al trabajo J_j cuando éste se completa en el instante de tiempo t. Dicha función dependerá, en general, de los parámetros anteriores.

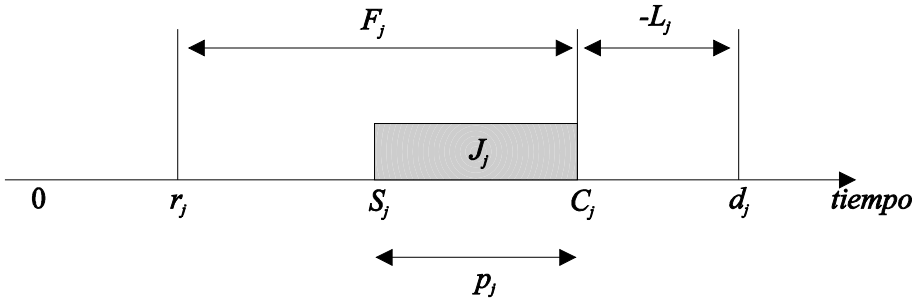


Figura 1. Datos y variables asociados a un trabajo

En el caso *determinístico* los datos m_j, r_j, p_j, p_{ij}, d_j, ω^k_j de cada trabajo son constantes conocidas mientras que en el caso *estocástico* pueden aleatorizarse dichos valores.

Comentemos ahora más específicamente los valores de los parámetros $\alpha|\beta|\gamma$ que hacen referencia a los diferentes problemas de planificación determinísticos.

2.1. Características de las máquinas

Podemos distinguir los problemas de planificación en los que se dispone de una única máquina *(α=1)* de aquellos problemas en los que podemos utilizar varias máquinas. Desde el momento en que se disponga de dos ó más máquinas cabe la posibilidad de que varias de ellas ejecuten trabajos distintos u operaciones de diferentes trabajos simultáneamente, y entonces podremos hablar de *paralelismo*. Además, será conveniente distinguir los problemas en los que las máquinas ejecutan las mismas funciones (máquinas *no especializadas*) de aquellos problemas en los que ciertas máquinas están especializadas en ciertas tareas y no pueden realizar otras (máquinas *especializadas*). Estas distinciones se sumarizan en el esquema siguiente:

NO ESPECIALIZADAS

Las máquinas ejecutan las mismas funciones.

IDENTICAS

Las máquinas tienen la misma velocidad de proceso.

UNIFORMES

Tienen distintas velocidades de proceso, pero son constantes y no dependen de los trabajos.

NO RELACIONADAS

La velocidad de proceso depende de los trabajos.

ESPECIALIZADAS

Máquinas especializadas en ciertas tareas. Van asociadas con problemas en los que los trabajos a realizar se dividen en varias operaciones o tareas.

SISTEMAS FLOW-SHOP

Cada trabajo se procesa por *todos* ó *algunos* de los procesadores siguiendo un *orden* prefijado por un patrón común.

SISTEMAS OPEN-SHOP

Cada trabajo se procesa por *todos* los procesadores y el procesamiento puede realizarse en *cualquier orden*.

SISTEMAS JOB-SHOP

El subconjunto de máquinas que procesa un trabajo y el orden de proceso son *arbitrarios* pero conocidos a priori.

Podemos distinguir las diferentes clases de máquinas atendiendo al parámetro α. El valor de dicho parámetro será *1, P, Q, R, F, O,* ó *J* según sea el problema considerado.

2.1.1. Máquinas no especializadas: $\alpha \in \{1, P, Q, R\}$

En este caso se considera, en general, que $m_j = 1 \ \forall j$, es decir, cada trabajo consta de una única operación que se puede ejecutar en cualquier máquina. Pero aunque haya una sola operación, el tiempo de procesamiento p_{ij} del trabajo J_j en la máquina M_i puede variar de una máquina a otra.

$\alpha = 1$.- Estaríamos ante el caso de una única máquina, $m = 1$, y, por tanto, $p_{1j} = p_j \ \forall j$ que denotaría el tiempo de proceso del trabajo J_j en dicha máquina.

$\alpha = P$.- Máquinas no especializadas en paralelo e idénticas, es decir, con la misma velocidad de proceso y, por tanto, $p_{ij} = p_j \ \forall i$, lo que equivale a decir que el tiempo de procesamiento del trabajo J_j , $\forall j$, no depende de la máquina que lo procese. Esto quiere decir que, fijado un trabajo a procesar, cualquier máquina es capaz de ejecutarlo y el tiempo que emplea en ello es el mismo para todas las máquinas.

$\alpha = Q$.- Máquinas no especializadas en paralelo y uniformes en el sentido de que el tiempo que tarda la máquina M_i en procesar completamente el trabajo J_j es $p_{ij} = p_j / q_i$, donde q_i es la velocidad constante de la máquina M_i que no depende de J_j. En este caso p_j denota el tiempo de procesamiento del trabajo J_j por una máquina M que se toma como referencia, q_i es el número de veces que M_i es más veloz que M de manera que p_{ij} es el tiempo de proceso del trabajo J_j en la máquina M_i. Esto quiere decir que cada máquina tiene una velocidad de proceso característica de la máquina y que, si, por ejemplo, la máquina B es 3 veces más rápida que la A y la C es 4 veces más rápida que la A; entonces, fijado un trabajo a procesar, todas las máquinas son capaces de ejecutarlo pero A requiere 3 veces más tiempo que B y 4 veces más tiempo que C. Si cambiamos de trabajo se siguen manteniendo los radios, es decir, A requerirá 3 veces más tiempo que B y 4 veces más tiempo que C para la ejecución del nuevo trabajo.

$\alpha = R$.- Máquinas no especializadas en paralelo y no relacionadas ya que la velocidad de proceso de cada máquina depende, no sólo de la máquina, sino también de los trabajos. Esto quiere decir que las velocidades de ejecución difieren de una máquina a otra y dependen también del trabajo en ejecución. Es decir, fijado un trabajo a procesar, todas las máquinas son capaces de ejecutarlo, pero si B es 3 veces más rápida que A y C es 4 veces más rápida que A para este

trabajo fijado, puede ocurrir perfectamente que para otro trabajo dichas relaciones cambien y sea, por ejemplo, B 2 veces más rápida que A, y C 3 veces más lenta que A.

2.1.2. Máquinas especializadas: $\alpha \in \{F, O, J\}$

En estos problemas, no todas las máquinas son capaces de realizar todas las tareas, sino que están especializadas en alguna de ellas. Por tanto, m_j no tiene que ser necesariamente 1, y cada trabajo J_j se divide en m_j operaciones que quizás requieran máquinas distintas. Incluso puede presentarse el caso $m_j \geq m$ con lo que habrá alguna máquina que realice varias operaciones de un mismo trabajo.

$\alpha = F$.- Se tiene un sistema flow-shop en el que cada trabajo J_j consiste en una cadena de $m_j = m$ operaciones $\{O_{1j}, O_{2j}, ..., O_{mj}\}$.$O_{ij}$ se procesa en la máquina M_i en un tiempo p_{ij}. Dicho orden es relevante y es siempre el mismo, es decir, todos los trabajos deben seguir un mismo patrón de flujo, esto es, la misma trayectoria de máquinas. No es necesario que un trabajo j pase por todas las máquinas. Si el trabajo j no se procesa por la máquina i consideraremos $p_{ij} = 0$. De esta manera englobamos también los problemas flow-shop en los que no todos los trabajos tengan que ejecutarse en todas las máquinas.

Obsérvese que un sistema flow-shop se parece a una cadena de montaje, sin embargo existen algunas diferencias: en primer lugar, en un sistema flow-shop puede existir una gran variedad de trabajos mientras que en una cadena de montaje es un producto estándar el que se procesa. En segundo lugar, los trabajos del flow shop no tienen por que ser procesados por todas las máquinas ya que un trabajo puede prescindir de una o varias máquinas. Sin embargo, todos los trabajos de una cadena de montaje han de pasar por todas las máquinas de la cadena. En tercer lugar, en el flow-shop la actuación de una máquina sobre un trabajo no depende de la máquina precedente en dicho trabajo, mientras que en una cadena de montaje si existe tal dependencia. Finalmente, en el flow-shop cada trabajo tiene su propio tiempo de procesamiento en cada máquina, mientras que en una cadena de montaje el tiempo de procesamiento de los trabajos en una máquina es el mismo para todos ellos.

Ejemplo:

Una empresa decide realizar un chequeo médico a todos sus empleados. El objetivo de la empresa es diagnosticar y tratar las posibles patologías de sus empleados de modo que todos ellos estén completamente sanos lo antes posible. El conjunto de empleados son los "trabajos" a procesar, los cuales están todos disponibles desde el instante inicial. No existen relaciones de precedencia entre

ellos puesto que el diagnóstico y tratamiento de cada empleado es independiente del de los demás. No se admiten interrupciones ni durante la fase de diagnóstico ni durante la fase de tratamiento. Existen dos "máquinas" que deben ejecutar los trabajos: la máquina de "diagnóstico" y la máquina de "tratamiento". El patrón de flujo o trayectoria de máquinas para todos los trabajos es, claramente, primero el diagnóstico y después el tratamiento. Aunque no es necesario que todos los trabajos necesiten la fase de tratamiento. El problema que tiene la empresa es un problema flow shop con $m = 2$ máquinas y n trabajos siendo n = número de empleados. Este es un problema $F2||C_{max}$ que se resuelve en tiempo $O(n \log n)$ por un algoritmo debido a Johnson (1954). Basta determinar una secuencia permutación, es decir, una ordenación de los n trabajos. La secuencia permutación óptima consiste en secuenciar primero los empleados cuyo tiempo de diagnóstico sea menor que su tiempo de tratamiento, en orden no decreciente de tiempos de diagnóstico, y luego el resto de empleados en orden no creciente de tiempos de tratamiento. Tanto el problema $Fm||C_{max}$ como el problema $Fm|pmtn|C_{max}$ con $m \geq 3$ son unarios NP-duros (Garey, Johnson y Sethi (1976), González y Sahni (1978a)).

$\alpha = O$.- Se tiene un sistema open-shop en el que cada trabajo J_j consiste en una cadena de $m_j = m$ operaciones $\{O_{1j}, O_{2j}, ..., O_{mj}\}$, donde O_{ij} se procesa en la máquina M_i en un tiempo p_{ij}. Es decir, se han numerado las operaciones del trabajo J_j de modo que la operación O_{ij} se realiza en la máquina i, $\mu_{ij} = i \ \forall j$. A diferencia con los problemas flow shop, el orden en que se realicen las operaciones es irrelevante.

Ejemplo:

Una comisión de selección de candidatos a un puesto de trabajo esta formada por 2 personas. El conjunto total de pruebas a las que deben someterse los candidatos esta particionado en dos, correspondiendo la evaluación de cada uno de los subconjuntos a cada uno de los miembros de la comisión de manera independiente. El objetivo de la comisión es realizar la selección en el menor tiempo posible. Cada miembro de la comisión necesitará un tiempo de evaluación de las pruebas de cada candidato. Dicho tiempo puede variar de unos candidatos a otros. Los candidatos pueden ser evaluados por uno u otro miembro de la comisión en orden indistinto. No existen relaciones de precedencia entre los candidatos puesto que la evaluación de cada uno de ellos es independiente de la finalización o no de las evaluaciones de los demás. La única restricción es que un determinado candidato no puede ser evaluado simultáneamente por los dos miembros de la comisión. Una vez comenzada la evaluación de un candidato por uno de los miembros de la comisión no puede interrumpirse. El problema puede entenderse como un problema open shop con $m = 2$ máquinas y n trabajos siendo n = número de candidatos existentes, es decir, es el problema $O2||C_{max}$. Este

problema puede resolverse en tiempo $O(n)$ por un algoritmo debido a González y Sahni (1976). Sin embargo, si el número de miembros de la comisión fuese $m \geq 3$, el problema $O||C_{max}$ sería NP-duro (González y Sahni (1976)).

$\alpha = J$.- Se tiene un sistema job-shop en el que cada trabajo J_j consiste en una cadena de m_j operaciones $\{O_{1j}, O_{2j}, ..., O_{m_jj}\}$. No tiene por qué ser $m_j = m$. Puede presentarse la situación en la que alguna máquina procese dos o más operaciones de un mismo trabajo. La trayectoria de máquinas de cada trabajo esta dada pero no tiene que ser la misma para todos los trabajos. Es decir, O_{ij} se procesa en la máquina μ_{ij} en un tiempo p_{ij} . Si una máquina realizase más de una operación de un trabajo dado, es lógico pensar que no ejecutaría dos operaciones seguidas de un mismo trabajo porque de lo contrario dichas operaciones podrían condensarse en una sola. Es decir, $\mu_{i-1,j} \neq \mu_{i,j} \quad \forall i = 2, ..., m_j$ y $\forall j = 1, ..., n$.

Ejemplos:

Cualquier empresa de manufacturas que no se dedique en masa a la producción de un único producto tiene problemas de planificación similares al job - shop. Cada producto tiene su propia ruta a través de las diferentes áreas de trabajo y máquinas de la factoría. En la industria de la ropa diferentes estilos tienen diferentes requerimientos de corte, cosido, planchado y empaquetado. En las plantas siderúrgicas, cada tamaño de varilla o viga de acero pasa a través de un conjunto de rodillos en su propio orden particular y bajo sus propias condiciones particulares de presión y temperatura. En la industria de impresión, el tiempo empleado en escribir un libro dependerá de su longitud, del número de ilustraciones, etc.

Nótese que los problemas flow-shop pueden considerarse como un caso particular de los problemas job-shop en los que todos los trabajos siguen el mismo patrón de flujo o trayectoria de máquinas. Así una editorial que debe planificar la producción de varios libros a través de los departamentos de escritura, impresión, encuadernado, y empaquetado sería un problema flow-shop con cuatro máquinas; cada departamento es una máquina y los trabajos, es decir, los libros, fluyen en el orden escritura, impresión, encuadernado, y empaquetado.

Cuando el parámetro α tome alguno de los valores anteriores, a excepción de $\alpha = 1$, entenderemos que el número m de máquinas es arbitrario pero fijo. Así, por ejemplo, si $\alpha = P$ estamos ante un problema en el que el número m de máquinas puede ser cualquier valor fijo que elijamos, pero si $\alpha = P2$ estamos ante un problema con dos máquinas. En este último caso, el algoritmo que lo resuelva ó la prueba de que es un problema de los llamados NP-duros serán válidos sólo para el caso de existencia de dos máquinas.

2.2. Características de los trabajos

Las condiciones impuestas por los trabajos para su ejecución se recogen en el segundo campo $\beta = \beta_1, \beta_2, \beta_3, \beta_4, \beta_5, \beta_6$ en relación a los conceptos siguientes:

2.2.1. Interrupciones: $\beta_1 \in \{pmtn, \square\}$

El símbolo "□" significa que el parámetro se omite, es decir, se deja en blanco sin especificar nada.

β_1 = *pmtn*.- Se permiten interrupciones, es decir, el procesamiento de cualquier operación puede interrumpirse y continuarse más tarde. Las interrupciones tienen sentido sólo si se puede retomar el trabajo en el estado en el que se dejo. No interesa dejar un trabajo si al retomarlo nos obliga a repetir lo que habíamos hecho.

β_1 = □.- No se permiten interrupciones.

2.2.2. Relaciones de precedencia: $\beta_2 \in \{\square, prec, tree\}$

Caracterizan las relaciones de dependencia o precedencia entre trabajos. Dos trabajos u operaciones serán dependientes si el comienzo de la ejecución de uno de ellos está condicionada a la conclusión previa del otro.

β_2 = □.- Los trabajos son independientes, es decir, no hay relaciones de precedencia entre ellos.

β_2 = *prec*.- Existe una relación de precedencia general entre los trabajos dada por un grafo G dirigido acíclico cuyos vértices representan los trabajos J_j $(j = 1, ..., n)$ y los arcos $J_j \rightarrow J_k$ indican que el comienzo de la ejecución del trabajo J_k está condicionado a la completación previa de J_j. Análogamente, si los trabajos constan de más de una operación puede plantearse la relación de precedencias entre operaciones con un grafo G dirigido acíclico cuyos vértices representen las operaciones O_{ij} $(j = 1, ..., n;\ i = 1, ..., m_j)$ y los arcos $O_{ij} \rightarrow O_{kl}$ indiquen que el comienzo de la operación O_{kl} del trabajo J_l esta condicionado a la completación previa de la operación O_{ij} del trabajo J_j.

β_2 = *tree*.- El grafo G de precedencias es un árbol. Si β_2 = *intree* ó β_2 = *outtree* dicho árbol será un árbol de ensamble (de cada vértice sale a lo sumo un arco), ó un árbol de ramificación (a cada vértice llega a lo sumo un arco), respectivamente.

2.2.3. Existencia de fechas de disponibilidad: $\beta_3 \in \{r_j, \square\}$

$\beta_3 = \square$.- Todos los trabajos están disponibles desde el instante inicial, $r_j=0 \ \forall j$, ó bien a partir de un determinado momento, $r_j = r \ \forall j$. Sin pérdida de generalidad tomaremos $r_j = 0 \ \forall j$. Estaríamos ante los problemas de planificación *estáticos.*

$\beta_3 = r_j$.- Las fechas de disponibilidad no tienen porque ser iguales. Estos serían los problemas de planificación *dinámicos.*

2.2.4. Cotas al número de operaciones: $\beta_4 \in \{m_j \leq m_{UB}, \square\}$

Este parámetro se refiere a los problemas de planificación job-shop. En estos problemas, el número m_j operaciones asociadas al trabajo J_j puede ser superior al número m de máquinas. En ciertos algoritmos que resuelvan algunos de estos problemas, ó en ciertas demostraciones de que problemas job-shop son NP-duros se puede utilizar el hecho de que el número de operaciones m_j del trabajo J_j esta acotado por una constante m_{UB}. Esta circunstancia se recoge en el parámetro β_4.

$\beta_4 = m_j \leq m_{UB}$.- Se especifica una cota superior m_{UB} de los m_j.
$\beta_4 = \square$.- Los m_j son arbitrarios, no conociéndose ninguna cota.

2.2.5. Tiempos de proceso: $\beta_5 \in \{p_{ij} = 1, \square\}$

Muchos problemas son resolubles con tiempos de proceso unitarios y no lo son con tiempos de proceso generales. Esta distinción se anota con el parámetro β_5.

$\beta_5 = p_{ij} = 1$.- Cada operación requiere un tiempo de proceso unitario.
$\beta_5 = \square$.- Los p_{ij} (ó bien p_j) son enteros arbitrarios.

2.2.6. Recursos adicionales: $\beta_6 \in \{\square, res\lambda\sigma\rho\}$

Podemos considerar que en el modelo de planificación a estudiar hay s tipos de recursos adicionales $RA_1, RA_2, ..., RA_S$ disponibles en las cantidades $ra_1, ra_2, ..., ra_S$ respectivamente. Cada trabajo J_j u operación O_{ij} tendrá asociado un vector de recursos adicionales requeridos:

$$ra(J_j) = (ra_1(J_j), ..., ra_S(J_j))$$
$$ra(O_{ij}) = (ra_1(O_{ij}), ..., ra_S(O_{ij}))$$

donde $0 \leq ra_l(J_j) \leq ra_l$, $0 \leq ra_l(O_{ij}) \leq ra_l$, denotan el número de unidades del recurso RA_l precisadas por el trabajo J_j o por la operación O_{ij}. Se supone que todos los recursos requeridos por una tarea están disponibles, bien antes de que comience su procesamiento, bien antes de que se vuelva a retomar dicha tarea (caso de planificación con interrupciones); y que, en el momento de completación ó interrupción de la misma, se liberan los recursos utilizados por ella con el fin de evitar bloqueos.

La incorporación de recursos adicionales en el modelo es especialmente adecuada en sistemas de computadores, donde los recursos adicionales pueden representar la memoria primaria, la capacidad de almacenamiento, número de canales, recursos de entrada/salida, etc. Dicha incorporación es también adecuada en aquellas situaciones en las que, la realización de las tareas en las máquinas requiera otros recursos limitados como la mano de obra, herramientas, espacio para almacenar materiales, etc.

Los recursos adicionales se incorporan al modelo gracias al parámetro $\beta_6 \in \{\square, res\lambda\sigma\rho\}$.

$\beta_6 = \square$.- No existen recursos adicionales.
Por el contrario, si $\beta_6 = res\lambda\sigma\rho$ si existen.

Téngase en cuenta que λ, σ, ρ son elementos del conjunto $\{.,k\}$ y representan, respectivamente, el número de recursos, los límites, y los requerimientos de recursos adicionales.

Si $\lambda, \sigma, \rho = .$ dichos números son arbitrarios, mientras que, si $\lambda = k_1$, $\sigma = k_2$, $\rho = k_3$, el número de recursos adicionales es k_1, hay k_2 unidades disponibles de cada recurso adicional, y se requieren, a lo sumo, k_3 unidades de cada recurso para ejecutar las tareas. Lógicamente no todos los recursos deben estar disponibles en las mismas cantidades ni todos los trabajos deben precisar el mismo número de unidades para ser procesados. Esta circunstancia puede expresarse sustituyendo los parámetros k_2, k_3 por los vectores correspondientes.

2.3. Criterios de optimalidad

El tercer y último campo expresa el número de funciones objetivo a considerar, sus características y, caso de más de un criterio, el tipo de optimización en el que se está interesado, es decir, si se buscan puntos eficientes, puntos extremos, realizar una optimización simultánea o una optimización jerárquica.

Sea γ^k uno de los criterios considerados, con $1 \leq k \leq K$, siendo K el número total de criterios que intervienen en el problema. Dicha función será, en general, una función creciente en los tiempos de completación *(función regular)*. Dependiendo de la naturaleza del problema interesará o no incorporar tiempo ocioso en la máquina. Usualmente se trabaja con funciones de la forma $\gamma^k = f_{max}$ ó $\gamma^k = \Sigma f_j$.

$$\gamma^k = \gamma^k(C_1, C_2, ..., C_n)$$

Recuérdese que, fijada la planificación, podemos calcular para cada trabajo las siguientes variables:

- tiempo de completación C_j , que indica el instante en el que el procesamiento del trabajo J_j concluye.

- la demora $L_j = C_j - d_j$, donde una demora positiva indica la tardanza en la completación del trabajo, mientras que, la conclusión del trabajo anticipadamente a su fecha límite d_j se indica por una demora negativa ó adelanto, cuyo valor absoluto es la cantidad de tiempo anticipada.

- la tardanza que viene dada por $T_j = max\ \{0, L_j\}$, e indica el retraso habido en la ejecución del trabajo J_j.

- el indicador de trabajo tardío que valdrá 1 si el trabajo no se concluye antes de su fecha límite, es decir:

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

En función de las variables anteriores se definen las funciones objetivo a minimizar. Estas pueden hacer referencia al coste máximo o al coste total.

Si hacen referencia al coste máximo serán de la forma: $f_{max} \in \{C_{max}, L_{max}\}$ con $f_{max} = max_j\{f_j(C_j)\}$, siendo $f_j(C_j) = C_j$ ó L_j.

Si hacen referencia al coste total serán:

$$\Sigma f_j \in \{\Sigma C_j, \Sigma T_j, \Sigma U_j, \Sigma \omega_j C_j, \Sigma \omega_j T_j, \Sigma \omega_j U_j\}.$$

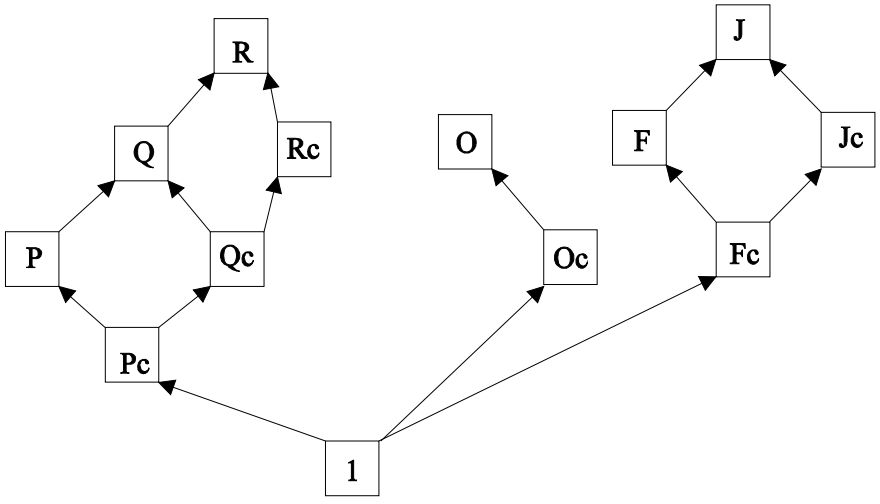
Como $\Sigma \omega_j C_j = \Sigma \omega_j L_j - \Sigma \omega_j d_j$, y el sustraendo es constante, los criterios $\Sigma \omega_j C_j$ y $\Sigma \omega_j L_j$ son equivalentes. De otra parte, cualquier planificación que minimice L_{max} minimiza T_{max} y U_{max}, pero no se da el recíproco.

Para simplificar el estudio de los problemas de planificación conviene aprovechar ciertas relaciones existentes entre ellos que expresan la dificultad o sencillez de su resolución. A esta labor se dedica el siguiente epígrafe.

3. Reducibilidad entre problemas de planificación unicriterio

La caracterización de los problemas de planificación mediante los parámetros $\alpha|\beta|\gamma$ descrita anteriormente nos permite confeccionar un esquema de cara a determinar interrelaciones entre ellos, de manera que si para un problema se encuentra un algoritmo eficiente, entonces quedarían resueltos también otros problemas estrictamente relacionados con él.

Los posibles valores de los parámetros α, β, y γ comentados anteriormente, los podemos esquematizar en los siguientes grafos:



G_0

Figura 2.a. Reducibilidad respecto a las características de las máquinas.

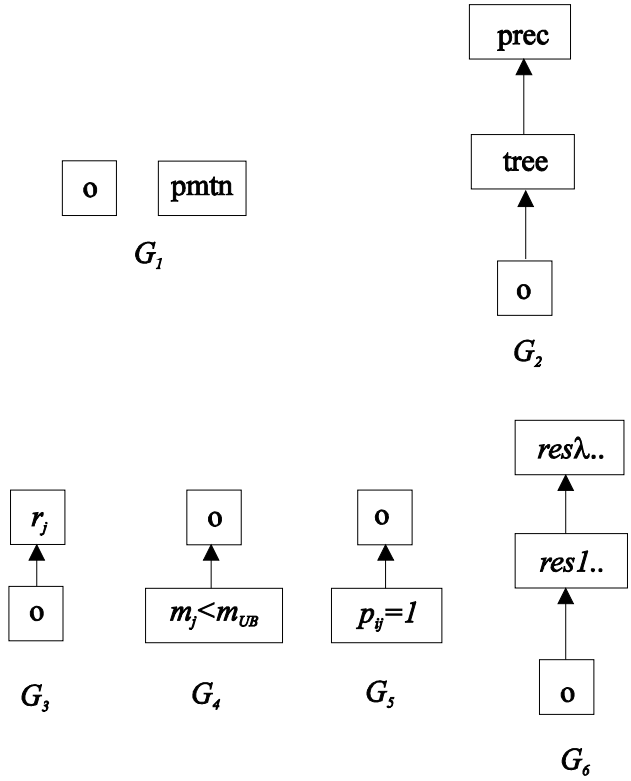


Figura 2.b. Reducibilidad respecto a las características de los trabajos.

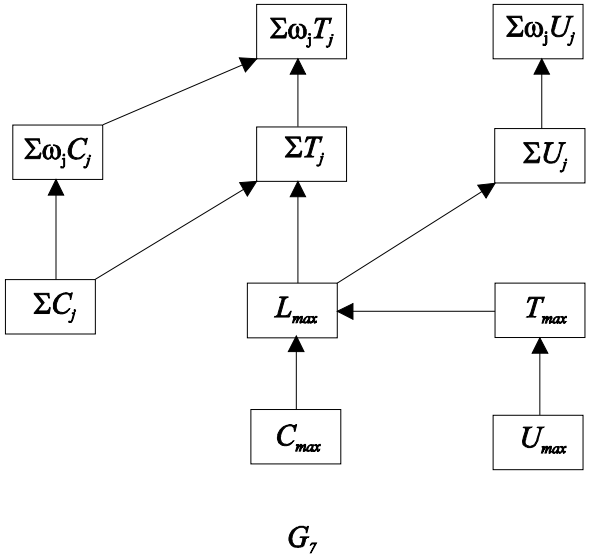


Figura 2.c. Reducibilidad respecto al criterio considerado.

El grafo G_o recoge los distintos valores del parámetro α, los grafos G_1, G_2, ..., G_6 los correspondientes a β, y el último grafo G_7 distintas funciones objetivo γ de manera que eligiendo un vértice de cada grafo se tiene determinado unívocamente un problema de planificación unicriterio.

En cada grafo, el arco (v_i, v_j) simboliza que el problema de planificación asociado al vértice v_i es más sencillo que el correspondiente al vértice v_j; es más, si el problema relativo al vértice v_j se resuelve en tiempo polinomial también se resolverá en tiempo polinomial el problema relacionado con el vértice v_i. Mientras que si éste problema fuese bastante complejo también lo sería el asociado al vértice v_j. Con ello, los grafos anteriores nos permiten visualizar una reducibilidad entre problemas de planificación unicriterio basada en la complejidad computacional de dichos problemas.

El siguiente epígrafe analiza con más detalle la complejidad computacional considerada como medida del grado de dificultad de un problema.

4. Conceptos generales sobre complejidad computacional

Como el abanico de problemas de planificación es muy amplio, para encontrar "buenos" algoritmos conviene "evaluar" de alguna manera el "grado de dificultad" de cada problema. En términos intuitivos, la "dificultad" de un problema consiste en encontrar un algoritmo "adecuado y eficiente" que lo resuelva. También intuitivamente podemos entender que un algoritmo es eficiente si su ejecución en cierta máquina o computador requiere unos recursos (tiempo, espacio o cualesquiera otros) relativamente pequeños según ciertos criterios.

Ese grado de dificultad de cada problema es precisamente la complejidad computacional del mismo. Por tanto, es conveniente estudiar la complejidad computacional de los problemas de planificación antes de afrontar la búsqueda de algoritmos eficientes que los resuelvan.

Para nuestro propósito, un *problema general* es una pregunta ó cuestión que hay que responder. El planteamiento de un problema se realiza en función de unos datos que representarán la entrada para el agente computacional. Dichos datos dependen de varios parámetros o variables libres cuyos valores se dejan sin especificar. Para que el planteamiento del problema esté completo debe formularse una sentencia o cuestión en términos de los datos de entrada. Un *problema particular* se obtiene cuando se plantea una cuestión concreta, esto es, cuando se especifican valores particulares para todos los parámetros del problema.

Un *algoritmo* para resolver el problema es un método o procedimiento que nos permite, a partir de unos datos de entrada, obtener unos datos de salida que

nos brindan la posibilidad de responder satisfactoriamente a la cuestión planteada. Para obtener los datos de salida a partir de un algoritmo A deben someterse los datos de entrada a ciertas transformaciones computadas por una cierta función f_A: la *función computada* por el algoritmo. Para realizar la computación es necesario codificar los datos de entrada con un conjunto finito de símbolos Σ denominado *alfabeto*. Dicha codificación de los datos se hace mediante un *esquema de codificación* EC, el cual consiste en una aplicación desde el conjunto de datos de entrada a un nuevo conjunto Σ^*: EC: {datos} $\rightarrow \Sigma^*$, siendo $\Sigma^* = \bigcup_{n \in N \cup \{0\}} \Sigma^n$ el *cierre de Kleene* asociado al alfabeto Σ. Una vez codificados los datos, la función computada por el algoritmo será f_A: $\Sigma^* \rightarrow \Gamma^*$, siendo Γ^* el cierre de Kleene asociado al alfabeto Γ en el que se codifican los datos de salida.

Un *problema de decisión o existencia* π es aquel problema cuya cuestión a resolver sólo tiene dos respuestas posibles: *"Si"* ó *"No"*. Es decir, dado un dato de entrada (elemento del conjunto de posibles datos de entrada D_π) el problema consiste en decidir si el mismo es un elemento del subconjunto $Y_\pi \subseteq D_\pi$ de entradas cuya respuesta es "Sí".

Si tenemos un esquema de codificación EC, podemos asociar al problema de decisión π un subconjunto de Σ^* denominado *lenguaje* $L_\pi = EC(Y_\pi) \subseteq \Sigma^*$ que será la codificación del conjunto de datos con respuesta afirmativa. La clasificación computacional del problema π se hace atendiendo a cierta clasificación del lenguaje L_π de manera unívoca, de hecho, una vez establecido el esquema de codificación EC, el problema de decidir si cierta entrada $I \in D_\pi$ está o no en Y_π se reduce al problema de decidir si su codificación x = EC(I) es miembro o no del lenguaje L_π , y por ello se dice que los problemas de decisión son problemas de reconocimiento de lenguajes.

Un *problema de búsqueda* π es aquel problema tal que para cada entrada $I \in D_\pi$ existe un subconjunto $S_\pi(I) \subseteq \Gamma^*$ de soluciones válidas para la entrada I. Dada una entrada I, solucionar el problema de búsqueda consiste en encontrar un elemento de $S_\pi(I)$, si $S_\pi(I) \neq \varnothing$, ó responder "No" si dicho subconjunto es vacío.

Un problema de decisión puede formularse como un problema de búsqueda donde $S_\pi(I)$ = {"Sí"} si $I \in Y_\pi$, ó bien $S_\pi(I) = \varnothing$ si $I \notin Y_\pi$. La clasificación computacional de un problema de búsqueda puede identificarse, salvo matices terminológicos, con la clasificación computacional del problema de decisión asociado.

Formalmente, un *problema de optimización combinatoria* π es un problema de minimización o maximización que consta de las siguientes tres partes: un conjunto D_π de entradas; para cada entrada $I \in D_\pi$, un conjunto *finito*

$S_\pi(I) \subseteq \Gamma^*$ de soluciones candidatas para la entrada I; una función m_π que asigna a cada entrada $I \in D_\pi$ y cada solución candidata $\sigma \in S_\pi(I)$ un número racional positivo $m_\pi(I, \sigma) \in Q$.

Así, una solución $\sigma^* \in S_\pi(I)$ es óptima para el problema de minimización si $m_\pi(I, \sigma^*) \leq m_\pi(I, \sigma)$ $\forall \sigma \in S_\pi(I)$. Sin pérdida de generalidad puede entenderse optimizar por minimizar.

Podemos considerar $m_\pi : D_\pi \; x \bigcup_{I \in D_\pi} S_\pi(I) \rightarrow Q$. Además, dada una entrada I $\in D_\pi$ podemos reducir el problema de optimizar $m_\pi(I, .) : S_\pi(I) \rightarrow Q$ en el problema de minimizar cierta función $f: X \subseteq N \rightarrow N$, definida sobre un conjunto X con igual cardinal que $S_\pi(I)$, ya que tanto $S_\pi(I)$ como Q son numerables.

$$\begin{array}{ccc} S_\pi(I) & \xrightarrow{m_\pi(I,.)} & Q \\ \downarrow & & \downarrow \\ X \subseteq \mathrm{N} & \xrightarrow{f} & \mathrm{N} \end{array}$$

Figura 3. Problemas equivalentes

Entonces, el problema consiste en encontrar el valor mínimo $f_o = \min_{x \in X} f(x)$ y el punto donde se alcanza ese mínimo, es decir, el $x_o \in X$ con $f(x_o) = f_o$.

El problema de encontrar el valor mínimo f_o puede entenderse como una sucesión de problemas de búsqueda y decisión de igual complejidad computacional en el sentido siguiente: dado $\alpha_1 \in N$,

¿existe un $x \in X$ tal que $f(x) \leq \alpha_1$?

Si la respuesta es afirmativa, elegir $\alpha_2 \leq \alpha_1$ y plantear la pregunta con α_2. En otro caso, elegir $\alpha_3 \geq \alpha_1$ y plantear la pregunta con α_3. El proceso continúa hasta que se halle un elemento α^* tal que la cuestión ¿existe un $x \in X$ tal que $f(x) \leq \alpha^*$? tiene respuesta afirmativa y para todo $\alpha < \alpha^*$ la respuesta es negativa. Además, dado que $f(X) \subseteq N$, y N es un conjunto bien ordenado, el proceso acaba en un número finito de pasos obteniendo $f_o = \alpha^* = \min_{x \in X} f(x)$ y el $x_o \in X$ con $f(x_o) = f_o$.

La idea para estudiar la complejidad de un problema de optimización combinatoria es muy sencilla: resolver el problema se reduce a encontrar un algoritmo adecuado que resuelva los problemas de búsqueda y decisión asociados. Ahora bien, como la complejidad de un problema de búsqueda es la complejidad del problema de decisión asociado, podemos decir que la complejidad computacional de un problema de optimización combinatoria es la complejidad computacional del problema de decisión correspondiente. Este último problema es un problema de reconocimiento de lenguajes, y clasificarlo computacionalmente es clasificar el lenguaje. A su vez, la clasificación del lenguaje estará en función

de qué máquina de Turing es capaz de reconocerlo, y de las cantidades de tiempo y espacio precisados por la máquina para ello. Esto es así porque la Tesis de Church (Church (1933 y 1936), Balcázar y otros (1988)) *"Todo algoritmo puede ser descrito por una máquina de Turing"* nos garantiza la existencia de una de estas máquinas que, al reconocer el lenguaje, nos resuelve el problema de optimización combinatoria.

Un elemento importante en las máquinas de Turing es la *función de transición* δ. Si a cada original, la función de transición asocia una *única* imagen estaríamos ante una *máquina de Turing determinística* y con *algoritmos determinísticos*. En este caso, fijada la entrada, sólo existe un único camino para ir desde el estado inicial a cualquier otro estado.

Si a cada original, la función de transición asocia *más de una* imagen estaríamos ante una *máquina de Turing no determinística* y con *algoritmos no determinísticos*. En este segundo caso, y fijada la entrada, pueden existir varios caminos para ir desde el estado inicial a cualquier otro estado.

Finalmente, si para cada original la función de transición realiza un *sorteo* entre un conjunto de posibles imágenes y le asigna *sólo una* de ellas podríamos hablar de *máquinas y algoritmos probabilísticos*.

Obviamente podemos considerar toda máquina determinística como un caso particular de las no determinísticas.

Como hemos dicho, la complejidad computacional de un problema se mide en función de los recursos consumidos por el agente computacional para resolverlo. Como agente computacional tenemos la máquina de Turing y la complejidad del problema será entonces medida como una función tal que, para cada n, nos da la cantidad de recursos requeridos por la máquina de Turing correspondiente sobre entradas de tamaño n. Entre los recursos consumidos a considerar tiene especial relevancia el tiempo y el espacio.

Si M es una máquina de Turing *determinística* y ω es una entrada puede pasar que M pare sobre ω ó que no pare. Si M para sobre ω se define el *tiempo de computación* de M sobre ω como el número de pasos de computación requeridos para que M finalice su computación para la entrada ω. Dicho tiempo queda indefinido en cualquier otro caso. La medida de complejidad o *tiempo de ejecución* será una función $T_M : N \rightarrow N$ donde $T_M(n) = \max_{|\omega|=n} \{$tiempo de computación de M sobre $\omega\}$, si M para sobre todas las entradas ω de tamaño n, y un valor indefinido en cualquier otro caso.

Si M es una máquina de Turing *no determinística* y ω es una entrada se define el *tiempo de computación* de M sobre ω como el número de pasos de

computación de la computación aceptada más corta de M sobre ω, si tal computación existe, y 1 en cualquier otro caso. La medida de complejidad o *tiempo de ejecución* será en este caso $T_M : N \rightarrow N$ donde $T_M(n) = \max_{|\omega|=n}$ {tiempo de computación de M sobre ω}

Si M es una máquina de Turing *determinística* y ω es una entrada se define el *espacio de computación* de M sobre ω como el máximo número de casillas de la cinta de la máquina de Turing analizadas durante una computación de M sobre ω, si tal máximo existe, y un valor indefinido en cualquier otro caso. La medida de complejidad o *espacio de ejecución* será una función $W_M : N \rightarrow N$ donde $W_M(n) = \max_{|\omega|=n}$ {espacio de computación de M sobre ω}, si existe tal máximo, y un valor indefinido en cualquier otro caso.

Si M es una máquina de Turing *no determinística* y ω es una entrada se llama *espacio requerido* por una computación dada al número máximo de casillas de cinta analizadas en la computación. Dicho número dependerá de la entrada y la computación. Denominaremos *espacio de computación* de M sobre ω al menor de los espacios requeridos en las computaciones aceptadas de M sobre ω, si existen, ó 1 en otro caso. Dicho número dependerá sólo de la entrada y no de ninguna computación concreta. La medida de complejidad ó *espacio de ejecución* será en este caso $W_M : N \rightarrow N$ donde $W_M(n) = \max_{|\omega|=n}$ {espacio de computación de M sobre ω}. Nótese que $W_M(n)$ no depende ni de la entrada ni de ninguna computación, únicamente del tamaño de la entrada.

Diremos que la máquina M acepta el lenguaje L en *tiempo* f(n) sí, y sólo sí, $T_M(n) \leq f(n) \ \forall n \in N$. Diremos que la máquina M acepta el lenguaje L en *espacio* f(n) sí, y sólo sí, $W_M(n) \leq f(n) \ \forall n \in N$. Por tanto, el grado de complejidad de un problema (ó la clasificación del lenguaje L correspondiente) en el sentido de *tiempo* ó *espacio* vendrá dada por la función $f: N \rightarrow N$ que mayora a T_M ó W_M.

Podemos definir las clases de complejidad o clases de lenguajes $L \subseteq \Sigma^*$ siguientes:

DTIME(f) = {$L \subseteq \Sigma^*$ tales que L es aceptado por una máquina de Turing *determinística* con $T_M(n) \leq f(n) \ \forall n \in N$}

NTIME(f) = {$L \subseteq \Sigma^*$ tales que L es aceptado por una máquina de Turing *no determinística* con $T_M(n) \leq f(n) \ \forall n \in N$}

DSPACE(f) = {$L \subseteq \Sigma^*$ tales que L es aceptado por una máquina de Turing *determinística* con $W_M(n) \leq f(n) \ \forall n \in N$}

NSPACE(f) = {$L \subseteq \Sigma^*$ tales que L es aceptado por una máquina de Turing *no determinística* con $W_M(n) \leq f(n) \ \forall n \in N$}

Estas definiciones se extienden de modo natural para familias de funciones, así por ejemplo:

$$\text{P} = \bigcup_{i\geq 0} DTIME(n^i)\text{, NP} = \bigcup_{i\geq 0} NTIME(n^i)\text{,}$$

$$\text{PSPACE} = \bigcup_{i\geq 0} DSPACE(n^i)\text{, NPSPACE} = \bigcup_{i\geq 0} NSPACE(n^i)\text{.}$$

Se tiene claramente que P $\subseteq$ NP. La cuestión ¿P = NP? permanece aún sin respuesta. Además PSPACE = NPSPACE (Aho y otros (1974)), y P $\subseteq$ NP $\subseteq$ PSPACE = NPSPACE.

La esencia de la conjetura ¿P = NP? radica en una clase de problemas denominados NP - completos, de manera que, si uno cualquiera de los problemas NP - completos fuese resoluble en tiempo polinomial, entonces sería P = NP.

Vamos a precisar la definición de la clase de lenguajes NP - completos.

Dados dos lenguajes L_1, $L_2 \subset \Sigma^*$, se dice L_1 es reducible en tiempo polinomial a L_2 ($L_1 \leq_m L_2$) sí, y sólo si, existe una función f: $\Sigma^* \rightarrow \Sigma^*$ computable en tiempo polinomial de grado m tal que $x \in L_1 \Leftrightarrow f(x) \in L_2$.

Entonces, dada una clase de lenguajes $\zeta \subseteq P(\Sigma^*)$ y un lenguaje $L \subseteq \Sigma^*$, diremos que:

1) L es ζ - duro (ζ - hard) para $\zeta \Leftrightarrow \forall L' \in \zeta \;\; L' \leq_m L$. En particular, L es NP - duro $\Leftrightarrow \forall L' \in$ NP $\;\; L' \leq_m L$, es decir, todo problema en NP se reduce polinomialmente a un problema, que llamamos NP-duro, y que no tiene porque estar necesariamente en NP.
2) L es ζ - completo (ζ - complete) para $\zeta \Leftrightarrow$ L es ζ - duro y $L \in \zeta$. En particular, L es NP - completo $\Leftrightarrow$ L es NP - duro y $L \in$ NP. La existencia de lenguajes NP - completos esta garantizada por el Teorema de Cook (Cook (1971)) que dice que el problema de satisfacibilidad es NP - completo.

Es frecuente distinguir entre los problemas *fuertemente NP-completos* (ó *unarios NP-completos*) y los problemas *NP-completos en sentido ordinario* (ó *binarios NP-completos*). Los primeros tienen asociados lenguajes que son NP-completos cuando se adopta un esquema de codificación unario (el número n se codifica con n unos), y los segundos son aquellos NP-completos cuando se adopta una codificación binaria.

Hay algunos problemas en los que, si se impone una cota superior a la magnitud de los datos o parámetros de entrada, existen algoritmos en tiempo polinomial para resolverlos. Estos algoritmos se llaman *pseudo polinomiales* porque además de ser polinomiales con respecto a los datos de entrada también lo son con respecto a la cota superior. Por tanto, no son polinomiales puros, y si se prescindiera de la cota dejarían de ser polinomiales. Esta circunstancia es muy

frecuente en problemas de scheduling donde intervienen muchos datos numéricos como parámetros de entrada (tiempos de procesamiento, fechas límite, tiempos de disponibilidad, etc.). Así, hay algoritmos considerados buenos para resolver algunos de estos problemas, que dejan de serlo cuando las magnitudes de los números de entrada son demasiado grandes.

Para terminar este capítulo introductorio, se recogen en el siguiente apartado los fundamentos necesarios para poder estudiar posteriormente problemas de planificación multicriterio.

5. Extensión al caso multicriterio

Muchos de los estudios de planificación tratan problemas en los que la bondad de la solución se mide en función de un único criterio. Sin embargo, un gran número de problemas que surgen en las situaciones prácticas tienen una naturaleza multicriterio. En estos casos tienen que considerarse todos los criterios que intervienen.

Cuando deben tomarse en cuenta diversos criterios, la selección de la solución óptima cambia según el criterio que se considere. Si los criterios no son conflictivos, entonces existen soluciones que son óptimas para todos los criterios. Estas soluciones son óptimos globales y en estas situaciones el problema es menos complicado.

Si los criterios son conflictivos, entonces no existen óptimos globales y se deben encontrar soluciones eficientes, para lo que se precisará información sobre las preferencias del decisor.

En función de la cantidad de información sobre las preferencias del decisor de la que se disponga, y de cómo se obtenga ésta, pueden considerarse diferentes métodos de Programación Multicriterio.

Los *métodos de las funciones de utilidad* aparecen cuando la información sobre la estructura de preferencias del decisor se puede expresar de forma precisa, es decir, mediante una función definida sobre un conjunto de objetivos y con rango un conjunto totalmente ordenado (por lo general R). En estos casos el problema se convierte en un problema unicriterio.

Los *métodos interactivos* surgen cuando la información que suministra el decisor sobre sus preferencias no es suficiente para formular un método perteneciente al grupo anterior, por lo que suele tener lugar un proceso iterativo en el que, en cada etapa, el analista presenta al decisor un conjunto de soluciones eficientes. El decisor realiza una valoración de las mismas facilitando así información suplementaria sobre sus preferencias. Información que el analista

incorpora al modelo para realizar la siguiente fase de cálculo. De esta manera, el decisor hace una descripción progresiva sobre sus preferencias. El proceso concluye cuando se obtenga alguna solución que satisfaga, dentro de lo posible, las aspiraciones y necesidades del decisor.

Los *métodos generadores de soluciones eficientes* son métodos que, en ausencia de cualquier información a priori sobre las preferencias del decisor se limitan a generar el conjunto de soluciones eficientes.

Existen también otras clasificaciones de los métodos de Programación Multicriterio. Así, se habla de *optimización jerárquica* cuando los criterios son ordenados de mayor a menor importancia para el decisor. Entonces, y siempre según la opinión del decisor, se establece un nivel de satisfacción mínimo para el criterio más importante, y se buscan soluciones que satisfagan dicho nivel en el criterio más importante, y que sean próximas a los niveles de satisfacción en los otros criterios. También se habla de *optimización simultánea* cuando el problema se convierte en un problema unicriterio mediante funciones de utilidad, o cuando esta conversión no es posible y se tienen que generar las soluciones eficientes sin información a priori suministrada por el decisor.

Para formular un problema multicriterio y definir los diferentes conceptos de solución conviene manejar conceptos como los de *alternativa, atributo, meta, objetivo* y *criterio.* Las definiciones de estos conceptos no están establecidas de modo único, pero, tomando como referencia a González (1986), podemos entenderlas de la siguiente manera.

Las *alternativas* son las diferentes acciones u opciones entre las que se debe obtener la solución al problema. El conjunto de alternativas también lo podemos denominar conjunto de *objetos*. Estos objetos vienen caracterizados por los *atributos*. Los *objetivos* vienen siendo valoraciones de los atributos en un espacio totalmente ordenado. Las *metas* son ciertos niveles "de conformidad" que deben alcanzar los atributos u objetivos para satisfacer ciertas necesidades impuestas a priori por el decisor. Los *criterios* son medidas o reglas que guían al decisor en el proceso de búsqueda de la solución del problema.

El conjunto de objetos $O = \{O_i / i \in I\}$ caracterizado por los atributos $X_1, ..., X_n$ se transforma en el conjunto $X = \{(X_{i1}, ..., X_{in}) / i \in I\}$ que suele ser un subconjunto de R^n. En aquellos casos en los que pueda establecerse una ordenación en X que permita resolver el problema de selección estaremos ante la *Programación Multiatributo.* Sin embargo, el carácter cualitativo de algunos atributos, el número de ellos, su dificultad de ser comparados entre sí u otros factores suelen dificultar la detección de una relación de orden en el conjunto X. En estos casos se precisan "medidas de costo", en número sensiblemente menor al número de atributos. Estas "medidas de costo", funciones u objetivos están definidos en el conjunto X y valorados en un conjunto totalmente ordenado como

R. Si $f_i : R^n \rightarrow R,\ i = 1, ..., p$ son las funciones objetivo, el problema de Programación Multiobjetivo puede plantearse como

$$min\ f(x)$$

s.a.:

$$x \in X$$

donde $f = (f_1, ..., f_p)^t : R^n \rightarrow R^p$. Es decir, la Programación Multiobjetivo convierte el problema de Programación Multiatributo sobre X en un problema de Programación Multiatributo sobre $f(X)$.

Se pueden establecer diferentes conceptos de solución. Se denomina *solución factible* a cualquier punto del conjunto X. *Soluciones ideales factibles* son aquellas que son óptimas respecto a al menos un criterio, es decir, son soluciones factibles que resuelven al menos uno de los problemas

$$min\ f_i(x)$$

s.a.:

$$x \in X$$

con x_i^* el óptimo un problema variando $i = 1, ..., p$. Si $x_1^* = ... = x_p^*$, entonces esta solución minimiza simultáneamente todos los objetivos. En este caso, dicha solución factible se denomina *solución óptima global* del problema.

Generalmente, los criterios que se consideran son contrapuestos y no existe una solución óptima global. Entonces se han de buscar otras soluciones en X cuyos niveles respecto de los objetivos sean lo mejores posibles. Los valores $f_i(x_i^*) = f_i^*$ sirven como referencia para el problema en X o en $f(X)$. La ausencia de solución óptima global obliga a interesarse por los puntos de $f(X)$ "más próximos" al *punto ideal* $(f_1^*, ..., f_p^*)$.

Una solución factible $x_E \in X$ es *eficiente* sí, y sólo sí, no existe otra solución $x \in X$ tal que $f_i(x) \leq f_i(x_E)\ \forall i = 1, ..., p$ con al menos una de las desigualdades estricta, es decir, con $f_{i_O}(x) < f_{i_O}(x_E)$ para cierto i_O. Las soluciones eficientes también se denominan *no dominadas, óptimos de Pareto, puntos admisibles, ...*

La *región eficiente,* ó conjunto de puntos eficientes, es el subconjunto de $f(X)$ que es imagen del conjunto X_E de soluciones eficientes.

$$S = \{y \in f(X) / \exists x \in X_E \subseteq X \text{ con } f(x) = y\} \subseteq f(X).$$

A cada punto de X_E le corresponderá uno de S, y a cada punto de S le correspnderá al menos uno de X_E .De ahí que las consideraciones sobre puntos eficientes pueden estar referidas indistintamente a los conjuntos S y X_E .

La elección del decisor que resuelve el problema multicriterio

$$min\ f(x)$$

s.a.:

$$x \in X$$

debe realizarse dentro de la región eficiente. Dicha elección puede realizarse de varias maneras con lo que aparecen los conceptos de solución preferida y solución satisfactoria. Se llama *solución preferida* a la solución eficiente que mejor satisface las preferencias del decisor. Y *solución satisfactoria* a la solución factible que satisface ciertas condiciones mínimas impuestas por el decisor.

La dificultad del problema multicriterio dependerá de la información que se tenga sobre las preferencias del decisor, de la naturaleza y propiedades de las funciones f_i, y de los conjuntos X_E y S adoptados en la formulación del problema y de los métodos empleados para la detección de las soluciones eficientes.

Muchos problemas de planificación multicriterio pueden resolverse desde la óptica de la optimización multicriterio, es decir, mediante la aplicación de sus métodos para detectar soluciones eficientes satisfactorias para el decisor. Los problemas de planificación multicriterio admiten una formulación específica que extiende la formulación correspondiente para problemas de planificación unicriterio.

Un problema de planificación multicriterio determinístico en su formulación más general puede establecerse extendiendo la formulación de los problemas unicriterio de manera natural, es decir, describiendo las características de las *m* máquinas que deben procesar un conjunto de *n* trabajos, los parámetros de éstos, y la descripción de los criterios que intervienen. Esto es una generalización natural de la formulación triparamétrica $\alpha|\beta|\gamma$ para problemas unicriterio descrita en epígrafes anteriores. Así, podemos representar un problema de planificación multicriterio determinístico en la forma $\alpha|\beta|(\gamma^1,...,\gamma^K)$ donde los parámetros α, β, γ^k representan las características de las *m* máquinas, de los *n* trabajos, y del criterio *k*-ésimo respectivamente con $k = 1,...,K$. Las características relevantes de máquinas, trabajos, y criterios están descritas en los epígrafes anteriores.

Hasta aquí se ha realizado el planteamiento y la formulación general de los problemas de planificación, estableciendo las características que configuran los diferentes problemas. Se han presentado la reducibilidad entre problemas y los conceptos generales sobre complejidad computacional. Pasemos a considerar en el siguiente capítulo diversos problemas de secuenciación sobre una máquina.

Capítulo II

Capítulo II: Problemas de Secuenciación sobre una máquina

1. Introducción

Este capítulo se centra en los problemas de planificación determinística sobre una máquina, aunque, al tratarse de una única máquina, el término *"problemas de secuenciación"* es quizás más adecuado para designar a estos problemas.

El capítulo comienza clasificando computacionalmente los problemas. unicriterio sobre una única máquina. Para ello se distingue si el criterio a minimizar es una función de costo expresada como un máximo, o bien, como una suma. Se presentan varios ejemplos que ilustran los métodos generales utilizados en la resolución de muchos problemas. Luego se proponen y estudian algoritmos específicos para resolver ciertos problemas. En relación con ello, en el Apéndice A, se hace un estudio computacional de algunos de los algoritmos propuestos.

2. Clasificación computacional.

Comencemos estudiando la complejidad computacional de la familia de problemas caracterizada por los parámetros $1|\beta|\gamma$. Desarrollaremos la exposición atendiendo primero al criterio de optimalidad (γ), y, posteriormente, nos fijaremos en las características de los trabajos (β).

Los problemas sobre una única máquina son los más sencillos que pueden presentarse. Muchos problemas con m máquinas los tienen como subproblemas componentes, lo que justifica su estudio. Además, algunos de estos problemas aparentemente sencillos tienen una gran complejidad, e incluso, existen problemas poco estudiados cuya complejidad está aún por descubrir.

Mencionemos que, cuando los trabajos están disponibles desde el instante inicial, esto es, si $r_j = 0 \ \forall j$ (versión estática del problema) es suficiente con buscar el óptimo entre las planificaciones sin interrupción y sin tiempo ocioso en la máquina (Conway y otros (1967)).

Comentaremos estos problemas atendiendo a la función objetivo $\gamma \in \{f_{max}, \Sigma f_j\}$, donde $f_{max} = max\ \{f_j\}$, y $f_j = f_j(C_j)$ es el coste de completar el trabajo J_j en el instante C_j.

2.1. Minimizando el máximo coste: $\gamma = f_{max}$

Recordemos que f_{max} puede ser U_{max}, T_{max}, C_{max} ó L_{max}. Ahora bien, en la versión estática C_{max} es constante puesto que se dispone de una única máquina. Además, y según los grafos de reducibilidad entre problemas de planificación unicriterio dados en el capítulo I, si resolvemos el problema L_{max} queda resuelto el problema T_{max} y, a su vez, U_{max}. Por tanto, f_{max} sólo será L_{max} en dicha versión.

$1|prec|f_{max}$ se resuelve por un algoritmo de complejidad $O(n^2)$ debido a Lawler (1973). Dicho algoritmo construye la secuencia solución eligiendo primero el último trabajo de la secuencia, después el penúltimo, y así sucesivamente hasta el primero. Para realizar esta construcción el algoritmo mantiene en cada paso una lista S de trabajos no planificados, un valor $p(S) = \sum_{j \in S} p_j$, (suma de los tiempos de procesamiento de los trabajos de S), y un subconjunto $S' \subseteq S$ de trabajos no planificados cuyos sucesores si lo están. El trabajo que se planifica en este paso es aquel trabajo $J_k \in \{J_j / j \in S\}$ tal que $f_k(p(S)) \leq f_j(p(S)) \ \forall j \in S$.

Algoritmo de Lawler

(0) $T \leftarrow \sum_{j=1}^{n} p_j$; $\Im \leftarrow \{J_1, ..., J_n\}$

(1) Determinar el conjunto U de los trabajos sin sucesores en $\Im$.

(2) Elegir de U el trabajo J_j que tiene un valor $f_j(T)$ minimal, deshacer los empates arbitrariamente; J_j se procesa desde el instante $T - p_j$ hasta el instante T.

(3) $T \leftarrow T - p_j$; $\Im \leftarrow \{J_j\}$.

(4) Si $\Im \neq \varnothing$, entonces ir al paso 1; en otro caso, parar.

El algoritmo de Lawler se ajusta fácilmente para el problema $1|\overline{d}_j, prec|f_{max}$. Si un trabajo J_k es un candidato para la última posición, entonces se tiene que comprobar si J_k no tiene sucesores y $\overline{d}_j \geq T$. Por tanto, el conjunto U contiene los trabajos que no tienen sucesores en $\Im$ y tienen una fecha límite mayor o igual a T. Alternativamente, se pueden incorporar las fechas límite redefiniendo $f_j(T) \leftarrow \infty$ para $T > \overline{d}_j$, $j = 1, ..., n$ y aplicar el algoritmo de Lawler al problema ajustado $1|prec|f_{max}$.

Las fechas límite no tienen que ser dadas explícitamente, pero pueden ser inducidas por cotas superiores dadas o por otros criterios. Por ejemplo, si g_i es una función de penalización no decreciente, para $i = 1,...,n$, entonces la restricción $g_{max} \leq G$ induce una fecha límite para cada trabajo J_i.

Ejemplo 1:

Consideremos el problema $1|prec|L_{max}$. Los datos numéricos son los siguientes: tenemos que realizar seis trabajos donde J_1 precede a J_2, que a su vez precede a J_3, y J_4 precede a J_5 y J_6.

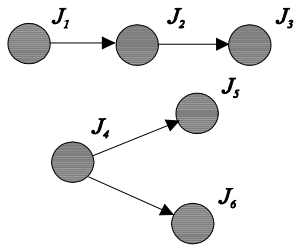


Figura 1. Gráfica de precedencias para el ejemplo 1.

Los tiempos de proceso y fechas límite son:

Trabajo	J_1	J_2	J_3	J_4	J_5	J_6
Tiempo de Proceso	2	3	4	3	2	1
Fecha Límite	3	6	9	7	11	7

El algoritmo de Lawler va determinando sucesivamente el trabajo que debe planificarse en sexta posición, quinta posición, hasta la primera posición, encontrando finalmente como secuencia óptima la secuencia: $J_1, J_2, J_4, J_6, J_3, J_5$.

También $1|pmtn,prec,r_j|f_{max}$ se resuelve por un algoritmo de complejidad $O(n^2)$ que generaliza el anterior (Baker y otros (1983)). Dicho algoritmo, primero modifica las fechas de disponibilidad. Se trabaja con una numeración de los trabajos en orden topológico, es decir, si J_j precede a $J_k \Rightarrow j < k$. Así, se toma:

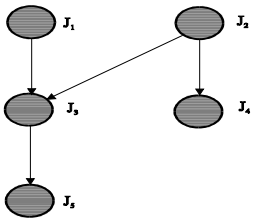
$$r_k = max\ \{r_k,\ max\ \{r_j + p_j / J_j\ \ precede\ a\ J_k\}\},\ k = 2, 3, ..., n.$$

Posteriormente se ordenan los trabajos en orden no decreciente de r_j *(ERD)* lo que permite una partición de los mismos en bloques. Cada bloque B se define como el conjunto minimal de índices de trabajos procesados sin tiempo muerto desde $r(B) = min_{j \in B}\{r_j\}$ hasta $t(B) = r(B) + p(B)$, con $p(B) = \sum_{j \in B} p_j$. Por tanto, si $j \notin B$ ocurre que $C_j \leq r(B)$, ó bien que $r_j \geq t(B)$. El algoritmo considera los bloques separadamente y, por tanto, para determinar la solución basta secuenciar los trabajos de cada bloque. Para ello, de entre los trabajos sin sucesores de un cierto bloque B_i se selecciona el trabajo J_k que origina menor costo cuando se pone en la última posición. Los restantes trabajos del bloque se replanifican en orden no decreciente de sus fechas de disponibilidad y el trabajo J_k se asigna detrás de ellos.

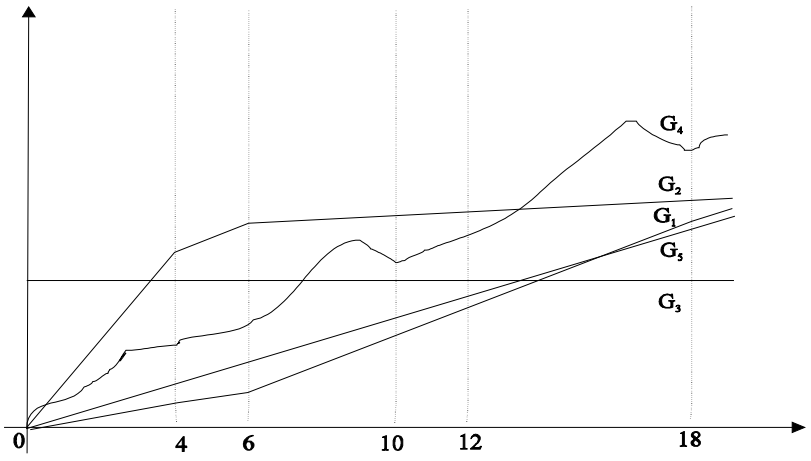
Repitiendo este procedimiento en todos los subbloques resultantes se obtiene una solución con a lo sumo *n-1* interrupciones en tiempo $O(n^2)$.

Ejemplo 2:

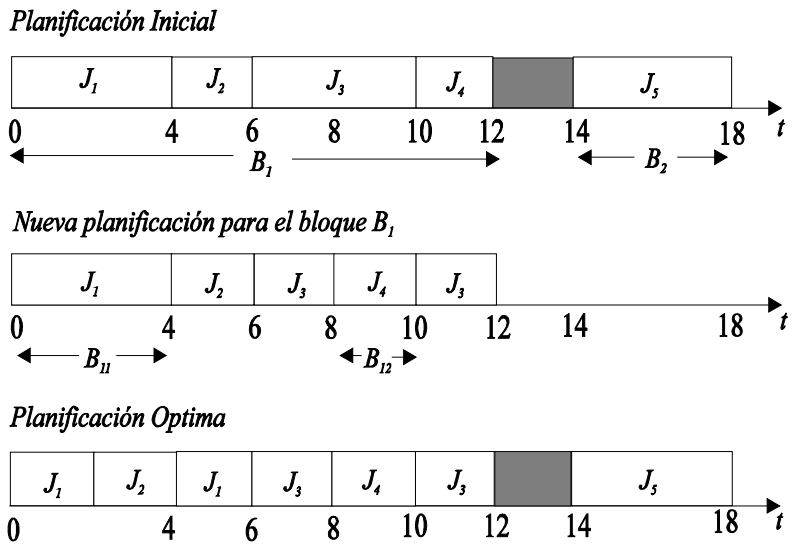
Para ilustrar el algoritmo anterior elijamos el problema $1|pmtn,prec,r_j|G_{max}$, donde $G_{max} = max_j \{g_j\}$, y las g_j son funciones de costo. Considérense 5 trabajos $\{J_1, ..., J_5\}$ cuyos tiempos de proceso y fechas de disponibilidad vienen dados por los vectores $\boldsymbol{p}$ = {4, 2, 4, 2, 4} y $\boldsymbol{r}$ = {0, 2, 0, 8, 14}, respectivamente. Las relaciones de precedencia y las funciones de costo vienen dadas por las figuras.



(a)



(b)



(c)

Figuras 2. (a) Relaciones de precedencia, (b) funciones de costo, y (c) planificaciones para el ejemplo 2.

De las relaciones de precedencias obtenemos el vector de fechas de disponibilidad modificadas $\boldsymbol{r'} = \{0, 2, 4, 8, 14\}$. Tomando el vector de fechas límites modificadas en lugar de $\boldsymbol{r}$ el algoritmo de Baker y otros (1983) para $1|pmtn,r_j|G_{max}$ determina dos bloques, $B_1 = \{J_1, J_2, J_3, J_4\}$ desde el instante de tiempo 0 al 12, y $B_2 = \{J_5\}$ desde el 14 al 18. El bloque B_2 consiste en un único trabajo y, por tanto, representa una parte óptima de la planificación. Para el bloque B_1, se determina el subconjunto de trabajos sin sucesores: $L_1 = \{J_3, J_4\}$ y se selecciona el trabajo J_3 ya que $G_3(12) < G_4(12)$. Para replanificar los trabajos del bloque B_1 mientras se procesa el trabajo J_3 (sólo si ningún otro trabajo esta disponible), obtenemos dos subbloques $B_{11} = \{J_1, J_2\}$ de 0 a 6 y $B_{12} = \{J_4\}$ de 8 a 10. El bloque B_{12} no requiere más atención. Para el bloque B_{11} encontramos $L_{11} = \{J_1, J_2\}$ y seleccionamos J_1 ya que $G_1(6) < G_2(6)$. Replanificando los trabajos de B_{11} nuevamente obtenemos la planificación óptima.

Comentemos ahora la situación del problema $1|r_j|L_{max}$. En el caso general resulta ser un problema unario NP-duro (Lenstra y otros (1977)), a pesar de que existen algoritmos polinomiales cuando los r_j son todos iguales, los d_j son todos iguales, ó los p_j son todos iguales. En efecto:

(a) Si $r_j = r \ \forall j$ se resuelve en $O(n \log n)$ por la regla de Jackson o regla EDD (Jackson (1955)), es decir, por el orden

$$d_{[1]} \leq d_{[2]} \leq \ldots \leq d_{[n]}.$$

(b) Si $d_j = d \ \forall j$ se resuelve en $O(n \log n)$ por la secuencia ERD:

$$r_{[1]} \leq r_{[2]} \leq \ldots \leq r_{[n]}.$$

(c) Si $p_j = p \ \forall j$. El problema $1|r_j,p_j = 1|L_{max}$ se resuelve extendiendo la regla de Jackson (en cualquier instante se realiza el trabajo disponible de menor fecha límite). Sin embargo $1|r_j,p_j=p|L_{max}$ requiere una aproximación más sofisticada (Simons (1978)).

Incorporando restricciones de precedencia a los tres casos anteriores se plantean problemas bien resueltos. Basta para ello actualizar convenientemente las fechas de vencimiento d_j, y las fechas de disponibilidad r_j (Lageweg y otros (1976)).

$1|prec,r_j|L_{max}$ es un problema NP-duro, puesto que $1|r_j|L_{max}$ lo es. Para $1|prec,r_j|L_{max}$ Baker y Su (1974) han propuesto un método enumerativo que genera las planificaciones activas, es decir, aquellas en las que no puede adelantarse el instante de comienzo de una operación sin que se retrase el correspondiente a otra, con el que obtienen una cota inferior. Otros autores (McMahon y Florian (1975), Lageweg y otros (1976)) también han tratado el problema y, en particular, Carlier (1980) describe un método para comparar la eficiencia de los algoritmos.

2.2. Minimizando el coste total: $\gamma = \sum f_j$

Recordemos que el coste total se define como la suma de los costes individuales de cada trabajo.

2.2.1. Suma ponderada de tiempos de completación: $1|\beta|\sum \omega_j C_j$

$1||\sum \omega_j C_j$ se resuelve en tiempo $O(n \log n)$ por la regla de Smith (1956, regla WSPT) que consiste en planificar los trabajos en orden no decreciente de sus radios p_j / ω_j, es decir:

$$\frac{p_{[1]}}{\omega_{[1]}} \leq \frac{p_{[2]}}{\omega_{[2]}} \leq \ldots \leq \frac{p_{[n]}}{\omega_{[n]}}.$$

Este resultado se puede extender a algoritmos de $O(n \log n)$ para problemas más restrictivos:

- Introduciendo relaciones de precedencia en árbol (Horn (1972), Adolphson y Hu (1973), Sidney (1975)).

- Introduciendo relaciones de precedencia en cadenas serie - paralelo (Lawler (1978)).

Sin embargo, añadir restricciones de precedencia generales origina un problema NP-duro, incluso si $p_j = 1 \ \forall j$ ó $\omega_j = 1 \ \forall j$; es decir, $1|prec|\sum \omega_j C_j$ es NP-duro (Lawler (1978), Lenstra y Rinnooy Kan (1978).

$1|r_j|\sum C_j$ es unario NP-duro (Lenstra y otros (1977)).

En cuanto a problemas con posibilidad de interrupción de los trabajos:

$1|pmtn,r_j|\sum C_j$ puede resolverse extendiendo la regla WSPT, mientras que $1|pmtn,r_j|\sum \omega_j C_j$ es unario NP-duro (Labetoulle y otros (1979)).

$1|r_j|\sum \omega_j C_j$ es NP-duro por serlo la versión no ponderada, pero puede abordarse por varios algoritmos de eliminación y de ramificación y acotación (Rinaldi y Sassano (1977), Bianco y Ricciardelli (1981), Hariri y Potts (1981)).

2.2.2. Suma ponderada de tardanzas: $1|\beta|\sum \omega_j T_j$

$1||\sum \omega_j T_j$ es unario NP-duro (Lawler (1977), Lenstra y otros (1977)), pero se han propuesto varios métodos enumerativos de solución. Dichos métodos se agilizan notablemente incorporando los criterios de eliminación desarrollados por Emmons (1969) y Shwimer (1972) que pueden extenderse a funciones de costos no decrecientes en T_j (Rinnooy Kan y otros (1975)). Cotas inferiores a la solución óptima se obtienen relajando el problema en un problema de asignación lineal (Rinnooy Kan y otros (1975)) ó en el problema de transporte (Gelders y Kleindorfer (1974,1975)) ó permitiendo que una máquina pueda procesar más de un trabajo a un tiempo (Fisher (1976)).

Si $p_j = 1 \ \forall j$, se tiene simplemente un problema de asignación lineal.

$1||\sum T_j$ es NP-duro (Du y Leung (1990)). Puede abordarse por métodos enumerativos y heurísticos. Muchos de los cuales están basados en las reglas de dominancia de Emmons (1969) que ayudan a restringir la búsqueda del óptimo. Entre estos algoritmos tenemos el algoritmo híbrido de Srinivasian (1971) que reduce la talla del problema con las propiedades de Emmons y resuelve el problema reducido por programación dinámica. Otros métodos de programación dinámica son debidos a Schrage y Baker (1978), y Lawler(1979).

Lawler (1977) dio un potente teorema de descomposición que usó para obtener un algoritmo de programación dinámica pseudopolinomial de complejidad $O(n^4 \sum p_j)$. La combinación de la programación dinámica con el teorema de

descomposición ha originado variantes que enriquecen los algoritmos anteriores (Potts y Van Wassenhove (1982 y 1987)). Lawler (1982a) usó los resultados de su algoritmo pseudopolinomial para obtener un esquema ε - aproximativo que requiere un tiempo $O(n^7/\varepsilon)$.

Otros algoritmos aproximados son los de Wilkerson e Irwin (1971) y Sicilia y Alcaide (1989). Este último de complejidad $O(n^2\Sigma p_j)$.

Introduciendo restricciones de precedencia el problema es NP-duro, incluso para $1|prec,p_j = 1|\Sigma T_j$ (Lenstra y Rinnooy Kan (1978)).

Introduciendo fechas de llegada r_j el problema $1|r_j,p_j=1|\Sigma\omega_j T_j$ es un problema de asignación lineal mientras que $1|r_j|\Sigma T_j$ es NP-duro porque $1||\Sigma T_j$ lo es.

2.2.3. Número ponderado de trabajos tardíos: $1|\beta|\Sigma\omega_j U_j$

$1||\Sigma U_j$ se resuelve en tiempo $O(n \log n)$ por el algoritmo de Moore (1968). Este algoritmo puede extenderse al caso en que ciertos trabajos tengan que estar a la hora prevista (Sidney (1973)). La generalización más amplia en la que los trabajos finalizan en, ó después de sus fechas de vencimiento d_j es NP-duro (Lawler (1982b)). Sin embargo, con pesos convenientes $(p_j < p_k \Rightarrow \omega_j \geq \omega_k)$ (Lawler 1976) o fechas de disponibilidad convenientes $(d_j < d_k \Rightarrow r_j \leq r_k)$ (Kise y otros (1978)) puede resolverse en tiempo $O(n \log n)$.

$1||\Sigma\omega_j U_j$ es binario NP-duro (Karp (1972)) pero puede resolverse por programación dinámica en tiempo $O(n\Sigma p_j)$ (Lawler y Moore (1969)).

$1|prec,p_j=1|\Sigma U_j$ es NP-duro (Garey y Johnson (1976)), incluso en el caso de que el grafo de precedencias sea una cadena (Lenstra y Rinnooy Kan (1980)).

$1|r_j|\Sigma U_j$ es unario NP-duro porque lo era $1|r_j|L_{max}$, pero con las técnicas de programación dinámica podemos resolver $1|pmtn,r_j|\Sigma U_j$ en tiempo $O(n^5)$; y $1|pmtn,r_j|\Sigma\omega_j U_j$ en tiempo $O(n^3(\Sigma\omega_j)^2)$ (Lawler (1982b).

Sahni (1976) da algoritmos aproximados para $1||\Sigma\omega_j U_j$ de complejidad $O(n^3k)$. Ibarra y Kim (1978) dan algoritmos aproximados para $1|tree|\Sigma\omega_j U_j$ de complejidad $O(kn^{k+2})$.

3. Algoritmos Propuestos

3.1. Problema 1||ΣTj

Planteamiento del problema

Se dispone de una máquina preparada para procesar n trabajos $\{J_1, ..., J_n\}$. Cada trabajo *j* lleva asignado un tiempo de procesamiento p_j conocido y una fecha límite d_j dada. Los trabajos se deben procesar sin interrupción, es decir, una vez que la máquina haya comenzado a realizar el trabajo j, debe terminarlo sin posibilidad de pausa. Dichos trabajos están todos disponibles para poder ser procesados desde el instante inicial y son independientes en el sentido de no existir relaciones de precedencia, fijadas a priori, que deban cumplir los mismos.

Recordemos que C_j representa el instante en el cual el trabajo *j* se termina de procesar, es decir, el tiempo de completación del trabajo *j*. Lógicamente este instante varía en relación con la posición que ocupa el trabajo *j* en la secuencia correspondiente de trabajos. Así, si dicho trabajo ocupase el lugar *k*, entonces $C_j = \sum_{i=1}^{k} p_{[i]}$, siendo *[i]* el trabajo que estaría en la posición *i*.

La tardanza de un trabajo *j* venía dada por la expresión $T_j = max\ \{0, L_j = C_j - d_j\}$. El problema de la tardanza total consiste en encontrar una ordenación ó secuencia $\sigma = [[1],[2],...,[n]]$ de los trabajos que minimice la tardanza total $\sum_{j=1}^{n} T_{[j]}$.

Diversos modelos relativos a la tardanza total han sido abordados entre otros autores por Potts y van Wassenhove (1987), Schrage y Baker (1978), Lawler (1977), Lenstra y Rinnooy Kan (1980), y Blazewicz (1987). Du y Leung (1990) han demostrado que el problema es NP-duro unario. Por tanto, problemas grandes no se pueden resolver óptimamente debido al crecimiento exponencial del tiempo de computación y los requerimientos de almacenamiento relativos al tamaño del problema. Métodos exactos que permiten encontrar una solución óptima para el problema pueden verse en Baker (1974), Lawler (1977) y en Potts y van Wassenhove (1982). Estos últimos autores hacen un estudio exhaustivo de varios de estos métodos. El problema es que tales procedimientos requieren un gran esfuerzo computacional tan pronto como el número de trabajos asciende a varias decenas. En tales circunstancias los métodos heurísticos pueden ofrecer soluciones aceptables, ya que la pérdida de bondad de la solución obtenida queda contrarrestada por la notable mejora del tiempo invertido en su búsqueda.

Wilkerson e Irwin (1971) presentan un método heurístico que utiliza una estrategia de intercambio de pares adyacentes. Fry y otros (1989) emplean la mejor de nueve estrategias de intercambio de pares adyacentes para obtener soluciones que mejoran significativamente la calidad de las soluciones obtenidas por Wilkerson e Irwin.

Recientemente, Holsenback y Russell (1992) han desarrollado un método heurístico que utiliza las propiedades de dominancia de Emmons (1969) y proporciona mejoras en algunos casos con respecto a Fry y otros (1989).

Describimos a continuación un método heurístico que guíe la búsqueda de una solución aproximada para el problema comentado.

Algoritmo

El algoritmo propuesto puede describirse de la siguiente manera: Una planificación factible inicial se elige como secuencia actual. Entonces, las secuencias vecinas se generan cambiando la posición de un trabajo previamente elegido. Mientras la secuencia con menor tardanza total entre las vecinas sea mejor que la secuencia actual, se cambia la secuencia actual por aquella, se generan las nuevas secuencias vecinas y se itera nuevamente. El algoritmo finaliza cuando la secuencia actual es mejor que todas sus vecinas.

Para elegir que secuencia inicial tomar conviene tener en cuenta algunas consideraciones para casos particulares del problema. En tal sentido, Baker (1974) señala que en el supuesto de que la secuencia que ordena las fechas límites de menor a mayor (secuencia *EDD*) lleve asociada a lo sumo un trabajo tardío $(T_j>0)$, entonces dicha secuencia es la óptima. También afirma que si todos los trabajos tienen la misma fecha límite, esto es, $d_j = d, \ \forall j \ 1= 1, \ ...,n$ entonces la secuencia que ordena los trabajos de acuerdo con sus tiempos de proceso en orden creciente (secuencia *SPT*) es la óptima.

Dichas consideraciones deben tenerse en cuenta a la hora de abordar la solución del problema planteado pero, como se ha comentado, en casos más generales del problema, no es posible caracterizar la secuencia óptima de forma tan sencilla.

Cuando se generan las secuencias vecinas, puede calcularse la tardanza total de ellas sin necesidad de computar todos los tiempos de completación nuevamente. Conocida la tardanza total de la secuencia actual podemos determinar la tardanza total de las nuevas secuencias usando el *Teorema 1* que enunciaremos a continuación. Usaremos para ello la siguiente notación:

- $[j]$ = trabajo localizado en la posición j
- $[j_o]$ = trabajo candidato a cambiar
- $B_{j_o\text{-}m}$ = cantidad en la que decrece el tiempo de completación del trabajo $[j_o]$ cuando se desplaza desde la posición j_o a la j_o-m.
- A^m = cantidad en la que crece la tardanza por posponer los trabajos $[j_o\text{-}m], [j_o\text{-}m+1], ..., [j_o\text{-}1]$.
- D_a^m = diferencia entre la tardanza total en la secuencia actual y la tardanza total obtenida cuando el trabajo $[j_o]$ se desplaza hacia la izquierda a la posición j_o-m (adelantar). Esta diferencia puede ser negativa.
- ΔC_m = cantidad en la que se incrementa el tiempo de completación del trabajo $[j_o]$ cuando se mueve desde la posición j_o a la j_o+m.
- Y^m = cantidad en la que se incrementa la tardanza del trabajo $[j_o]$ cuando se mueve desde la posición j_o a la j_o+m.
- Z^m = cantidad en la que decrece la tardanza por el hecho de adelantar los trabajos: $[j_o+1], ..., [j_o+m]$.
- D_b^m = diferencia entre la tardanza total en la secuencia actual y la tardanza total obtenida cuando el trabajo $[j_o]$ se desplaza a la derecha a la posición j_o+m (retrasar). Esta diferencia puede ser negativa.

Teorema 1

Sea j_o la posición del trabajo candidato $[j_o]$ en la secuencia actual. Sea T la tardanza total de dicha secuencia. Entonces se cumplen las sentencias siguientes:

a) si se desplaza el trabajo candidato a la posición j_o-m, la nueva secuencia tiene la tardanza total $T' = T - D_a^m$, donde D_a^m se determina recursivamente por las fórmulas:

$$\Delta T_{j_o-m} = \begin{cases} p_{[j_o]}, & si \quad L_{[j_o-m]} > 0 \\ 0, & si \quad L_{[j_o-m]} \le 0 \; y \; p_{[j_o]} \le \left| L_{[j_o-m]} \right| \\ L_{[j_o-m]} + p_{[j_o]}, & si \quad L_{[j_o-m]} \le 0 \; y \; p_{[j_o]} > \left| L_{[j_o-m]} \right| \end{cases}$$

$$A^0 = 0; \; A^m = A^{m-1} + \Delta T_{j_o\text{-}m};$$

$$B_{j_o} = 0, \; B_{j_o\text{-}m} = B_{j_o\text{-}m+1} + p_{[j_o\text{-}m]}$$

$$D_a^m = min \{B_{j_o\text{-}m}, T_{[j_o]}\} - A^m.$$

b) si se desplaza el trabajo candidato a la posición j_o+m, la nueva secuencia tiene la tardanza total $T' = T - D_b^m$, donde D_b^m se determina recursivamente por las fórmulas:

$$\Delta C_o = 0;\ \Delta C_m = \Delta C_{m-1} + p_{[j_o+m]}$$

$$Y^m = \begin{cases} \Delta C_m, & si \quad L_{[j_o]} > 0 \\ max\left\{0, L_{[j_o]} + \Delta C_m\right\}, & si \quad L_{[j_o]} \le 0 \end{cases}$$

$$X_{j_o+m} = \begin{cases} 0, & si \quad L_{[j_o+m]} \le 0 \\ min\left\{L_{[j_o+m]}, p_{[j_o]}\right\}, & si \quad L_{[j_o+m]} > 0 \end{cases}$$

$$Z^o = 0,\ Z^m = Z^{m-1} + X_{j_o+m};$$

$$D_b^m = Z^m - Y^m.$$

Demostración

a) Si se desplaza el trabajo candidato $[j_o]$ hacia la izquierda, sólo la tardanza $T_{[j]}$ con $j = j_o-m,\ j_o-m+1,\ ...,\ j_o$ puede cambiar (ver figura 3) de acuerdo con las fórmulas:

$$T'_{[j_o]} = \begin{cases} T_{[j_o]} - \sum_{j=j_o-m}^{j_o-1} p_{[j]}, & si \sum_{j=j_o-m}^{j_o-1} p_{[j]} \le T_{[j_o]} \\ 0, & otro\ caso \end{cases}$$

$$T'_{[j]} = max\left\{0, L_{[j]} + p_{[j_o]}\right\},\ j = j_o - m, \ldots, j_o - 1.$$

Entonces, la tardanza $T'_{[j_o]}$ se reduce en:

$$min\left\{\sum_{j=j_o-m}^{j_o-1} p_{[j]}, T_{[j_o]}\right\} = min\left\{B_{j_o-m}, T_{[j_o]}\right\}$$

cuando se mueve el trabajo $[j_o]$ a la posición j_o-m. Sin embargo, la tardanza $T_{[j_o-m+i]}$ con $i = 0,\ 1,\ ...,\ m-1$ aumenta debido al aumento de los tiempos de completación.

$$\Delta T_{j_o-m+i} = \begin{cases} p_{[j_o]}, & si \quad T_{[j_o-m+i]} > 0 \\ 0, & si \quad T_{[j_o-m+i]} = 0\ y\ p_{[j_o]} \le \left|L_{[j_o-m+i]}\right| \\ L_{[j_o-m+i]} + p_{[j_o]}, & si \quad T_{[j_o-m+i]} = 0\ y\ p_{[j_o]} > \left|L_{[j_o-m+i]}\right| \end{cases}$$

Entonces, $A^m = \sum_{i=0}^{m-1} C_{j_o-m+i}$ representa el aumento generado.

Por tanto, $D_a^m = min\left\{B_{j_o-m}, T_{[j_o]}\right\} - A^m$ es la ventaja total del cambio.

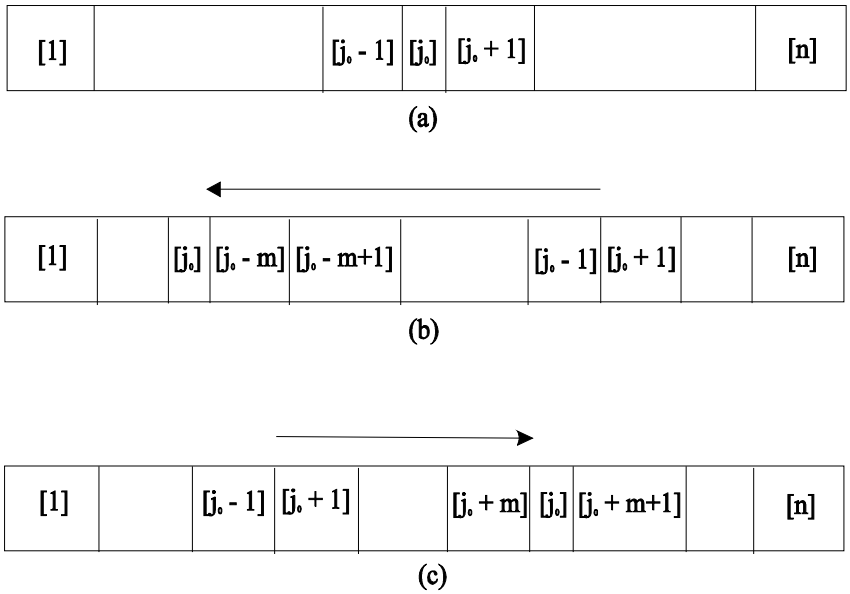


Figura 3. (a) Secuencia actual, (b) vecino: desplazamiento del trabajo candidato a la izquierda, (c) vecino: desplazamiento del trabajo candidato a la derecha.

b) La demostración en el caso de que el trabajo candidato $[j_o]$ se desplace hacia la derecha es similar. En este caso, sólo la tardanza $T_{[j]}$ *con* $j = j_o, j_o+1, ..., j_o+m$ puede cambiar (ver figura 3) de acuerdo con las expresiones:

$$T'_{[j_o]} = max\left\{T_{[j_o]},\, L_{[j_o]} + \sum_{i=1}^{m} p_{[j_o+i]}\right\} = max\left\{T_{[j_o]}, L_{[j_o]} + F_m\right\}$$

y para $j = j_o+1, ..., j_o+m$ la nueva tardanza es:

$$T'_{[j]} = \begin{cases} 0, & si\ L_{[j]} \le 0 \\ max\left\{0, L_{[j]} - p_{[j_o]}\right\}, & si\ L_{[j]} > 0 \end{cases}$$

Entonces la tardanza $T_{[j_o]}$ se incrementa en

$$max\left\{0, -T_{[j_o]} + L_{[j_o]} + \sum_{i=1}^{m} p_{[j_o+i]}\right\} = Y^m$$

mientras para cada $j = j_o+1,...,j_o+m$ la tardanza $T_{[j]}$ decrece en $min\{L_{[j]}, p_{[j_o]}\}$ si $L_{[j]} > 0$, y permanece la misma en los restantes casos.

Por tanto, la reducción obtenida es $Z^m = \sum_{j \in J'} min\ \{L_{[j]}, p_{[j_o]}\}$, siendo $J' = \{j / j_o + 1 \leq j \leq j_o + m$ y $L_{[j]} > 0\}$, y la ventaja total de ese cambio será $D_b^m = Z^m - Y^m$. □

Como la secuencia *EDD* es la que minimiza la tardanza máxima, se considera dicha secuencia como semilla inicial y si hubiese en la misma dos o más trabajos con fechas límites iguales, se establece la precedencia de estos trabajos de acuerdo con sus tiempos de procesamiento de menor a mayor.

Basado en el terorema anterior, exponemos un procedimiento heurístico para el problema de la tardanza total. El algoritmo opera de la siguiente manera: primero se elige el trabajo que va a ser candidato a cambiar de lugar (paso 2), determinándose después cuál debe ser el cambio más favorable, es decir, si debe adelantarse ó retrasarse la posición en la secuencia del trabajo elegido. La posible ganancia en la reducción de la tardanza, si adelantáramos el trabajo, viene dada en los pasos 4 y 5; mientras que si se retrasara dicho trabajo, la bondad de tal medida es cuantificada por los pasos 7 y 8 del algoritmo. La decisión de efectuar el cambio (siempre al lugar que ofrece una mayor reducción) es considerada en el paso 9. Si desistimos de efectuar el cambio, es debido a que no se reduce la tardanza total y, en consecuencia, elegiremos un nuevo trabajo que sea candidato a cambiar de lugar. El proceso finaliza cuando ningún candidato proporcione un cambio que permita reducir la tardanza total.

Algoritmo 1\|\|ΣT_j

1. Calcular la secuencia EDD, la cual es la semilla o solución inicial. Sea *[[1],[2],...,[n]]* el orden de dicha secuencia. Calcular $L_{[j]} = \sum_{i=1}^{j} p_{[i]} - d_{[j]}$ y $T_j = max\{0, L_{[j]}\}\ \forall j = 1,...,n$. Hallar $T = \sum_{j=1}^{n} T_{[j]}$. Poner $P = \varnothing$ (no se ha probado ningún trabajo).

2. Elegir el índice j_o tal que $L_{[j_o]} = max\{L_{[j]} / 1 \leq j \leq n\}$. Dicho trabajo $[j_o]$ es el candidato a cambiar de lugar.

3. Si $j_o = 1$, ir al paso 6. Si $j_o \neq 1$, pero $T_{[j_o]} = 0$, ir al paso 6. En otro caso, poner $m=1$, $G = 0$, $q = 0$, $A^0 = 0$, $B_{j_o} = 0$.

4. *(Adelantar).* Si $m=j_o$, ir al paso 6. En otro caso calcular los valores siguientes:

$$\Delta T_{j_o - m} = \begin{cases} p_{[j_o]}, & si \quad L_{[j_o - m]} > 0 \\ 0, & si \quad L_{[j_o - m]} \leq 0 \ y \ p_{[j_o]} \leq \left| L_{[j_o - m]} \right| \\ L_{[j_o - m]} + p_{[j_o]}, & si \quad L_{[j_o - m]} \leq 0 \ y \ p_{[j_o]} > \left| L_{[j_o - m]} \right| \end{cases}$$

$$A^m = A^{m-1} + \Delta T_{j_o-m};\ B_{j_o-m} = B_{j_o-m+1} + p_{[j_o-m]}$$

$$D^m = min\ \{B_{j_o-m}, T_{[j_o]}\} - A^m.$$

5. - Si $D^m = T_{[j_o]}$, entonces poner $q = j_o - m$, $G = D^m$, e ir al paso 6.
   - Si $D^m \neq T_{[j_o]}$ y $D^m > G$, entonces poner $q = j_o - m$, $G = D^m$. Hacer $m=m+1$ y volver al paso 4.
   - Si $D^m \neq T_{[j_o]}$ y $D^m \leq G$, entonces hacer $m=m+1$ y volver al paso 4.

6. Si $j_o = n$, ir al paso 9. Si $j_o \neq n$, poner $m=1, \Delta C_o=0$, y $Z^o = 0$.

7. *(Retrasar)*. Si $m = n - j_o + 1$, ir al paso 9. En otro caso, calcular los valores siguientes:

$$\Delta C_m = \Delta C_{m-1} + p_{[j_o+m]}$$

$$Y^m = \begin{cases} \Delta C_m, & si \quad L_{[j_o]} > 0 \\ max\left\{0, L_{[j_o]} + \Delta C_m\right\}, & si \quad L_{[j_o]} \leq 0 \end{cases}$$

$$X_{j_o+m} = \begin{cases} 0, & si \quad L_{[j_o+m]} \leq 0 \\ min\left\{L_{[j_o+m]}, p_{[j_o]}\right\}, & si \quad L_{[j_o+m]} > 0 \end{cases}$$

$$Z^m = Z^{m-1} + X_{j_o+m};\ D^m = Z^m - Y^m.$$

8. - Si $D^m > G$, poner $q = j_o + m$ y $G = D^m$. Hacer $m = m + 1$ y volver al paso 7.
   - Si $D^m \leq G$, hacer $m = m + 1$ y volver al paso 7.

9. *(Cambiar)*. Si $q = 0$, no hacer ningún cambio. Colocar $P = P \cup \{j_o\}$ e ir al paso 10 (búsqueda de un nuevo candidato).
   - Si $q > 0$, cambiar el trabajo $[j_o]$ al lugar q y *"rodar"* los trabajos que sean necesarios, esto es: sea $MIN = min\ \{q, j_o\}$, y $MAX = max\ \{q, j_o\}$.
   - si $MIN = q$, entonces $s(q) = [j_o]$, $s(q+1) = [q]$,..., $s(q+k) = [q+k-1]$,..., $s(j_o) = [j_o-1]$.
   - si $MIN = j_o$, entonces $s(j_o) = [j_o+1]$, ..., $s(j_o+k) = [j_o+k+1]$, ..., $s(q-1) = [q]$, $s(q) = [j_o]$.

   Antes de salir de este paso, recolocar los trabajos de manera que el trabajo de la posición i de la secuencia sea $[i] = s(i),\ \forall i \in [MIN, MAX]$.

   Volver a calcular $L_{[j]}$ y $T_{[j]}$ para $j \in [MIN, MAX]$ según las fórmulas del paso 1. Poner $T = T - G$, $P = \varnothing$ e ir al paso 2.

10. - Si *card(P) = n*, parar. La secuencia *[[1],[2],...,[n]]* nos da la solución, siendo *T* el valor de la tardanza total para dicha secuencia.
    - Si *card(P)* $\neq n$, entonces calcular el j_1 tal que $L_{[j_1]} = max\ \{L_{[k]} / k \notin P\}$. Hacer $j_o = j_1$ y volver al paso 3.

En el siguiente apartado se analiza la complejidad computacional de este algoritmo.

Complejidad del algoritmo

El algoritmo tiene una complejidad $O(n^2 \sum_{j=1}^{n} p_j)$, desglosada de la forma siguiente. El paso 1 requiere *O(n log n)* operaciones, mientras que los pasos del 2 al 10 requieren *O(n)* por cada iteración. El número de iteraciones para estos pasos es a lo sumo de $\sum_j T_j(EDD) - T_{max}(EDD)$, siendo $T_j(EDD)$ la tardanza del trabajo *j* correspondiente a la secuencia *EDD* y $T_{max}(EDD)$ la tardanza máxima para la misma secuencia. La cuestión será determinar qué valor puede tomar dicha expresión. La variabilidad de la misma es grande pero siempre estará acotada por $n\sum_{j=1}^{n} p_j$. Por tanto, la recursividad de los pasos segundo al décimo nos lleva a una complejidad $0(n^2 \sum_{j=1}^{n} p_j)$ que supera ampliamente la del paso 1 y, en consecuencia, dicha expresión será la complejidad del algoritmo.

Debemos señalar que el valor $\sum_j T_j(EDD) - T_{max}(EDD)$ se calcula con sólo realizar el paso 1 y en tal instante estaríamos en condiciones de predecir el comportamiento de la heurística con respecto a la velocidad de convergencia del algoritmo, sin necesidad de ejecutar los siguientes pasos ni esperar el final del mismo.

Comentemos por último que el algoritmo se ha programado utilizando el lenguaje de programación C. También se han codificado otros algoritmos propuestos por otros autores. En el Apéndice A se realiza un estudio comparativo de dichos algoritmos.

3.2. Problema 1|rj|∑Tj

Planteamiento del problema

Se considera un conjunto de n trabajos a realizar y una sola máquina, la cual puede procesar todos los trabajos pero con la imposibilidad de realizar varios de ellos simultáneamente. Cada trabajo lleva asignado unos tiempos que lo caracterizan y en base a ellos se pretende obtener secuencias de trabajos que sean óptimas, o en su defecto aproximadas a éstas, en relación con la tardanza total como medida de efectividad.

En el presente epígrafe se estudia el modelo de minimización de la tardanza total incluyendo tiempos de disponibilidad, y se considera la posible presencia de interrupciones en el procesamiento de los trabajos. Lawler, Lenstra y Rinnooy Kan (1982) comentan que la inclusión de tiempos de disponibilidad para los trabajos, problema $1|r_j|\sum T_j$, es aún más complejo que $1||\sum T_j$ y, por tanto, $1|r_j|\sum T_j$ también es un problema NP-duro unario.

El problema consiste en procesar n trabajos independientes sobre una máquina. Cada trabajo j queda caracterizado por su tiempo de procesamiento p_j, su fecha límite d_j en la que debería estar terminado, y su fecha de disponibilidad r_j que fija el instante a partir del cual puede comenzar a procesarse. Supuesta una ordenación de los trabajos, podemos calcular para cada trabajo j el tiempo de completación C_j y la correspondiente tardanza $T_j = max\ \{0,\ C_j - d_j\}$. El objetivo será determinar una secuenciación de trabajos que minimice la tardanza total $\sum T_j$.

Algoritmo

A continuación se propone un algoritmo heurístico para el problema $1|r_j|\sum T_j$. Se entenderá cómo algoritmo $1||\sum T_j$ el algoritmo propuesto en el epígrafe anterior para el problema $1||\sum T_j$.

El algoritmo, que denotaremos por algoritmo $1|r_j|\sum T_j$, va construyendo la secuencia solución desde el instante inicial hasta el final. En todo momento se conoce el conjunto S de trabajos no planificados susceptibles de ser secuenciados a partir de ese instante. El orden de planificación de los trabajos de S viene dado por el algoritmo $1||\sum Tj$. Cuando aparece un nuevo instante r' a partir del cual pueden planificarse nuevos trabajos, se estudia la *"holgura"* $h = r' - C$ siendo C el instante de completación del último trabajo planificado. Si hay trabajos de S planificables en el intervalo de amplitud h, $[C,r']$, se planifican aprovechando al máximo la holgura h sin sobrepasar el umbral r' (pasos 4 a 6). Si ya no quedan trabajos en el actual S con tiempos de proceso inferiores a h intervienen en consideración los trabajos planificables a partir de r' construyéndose un nuevo S (paso 8). En el paso 9 se aplica el algoritmo $1||\sum T_j$ y se vuelva a calcular la tardanza $T2$ dada por él, en el caso de que el primer trabajo $b(1)$ pueda planificarse a partir de $r' - h$. En el paso 10 se evalúa la tardanza $T1$ cuando $b(1)$ no es

planificable a partir de ese instante. En el paso 11 se comparan dichas tardanzas. El paso doce es la parada.

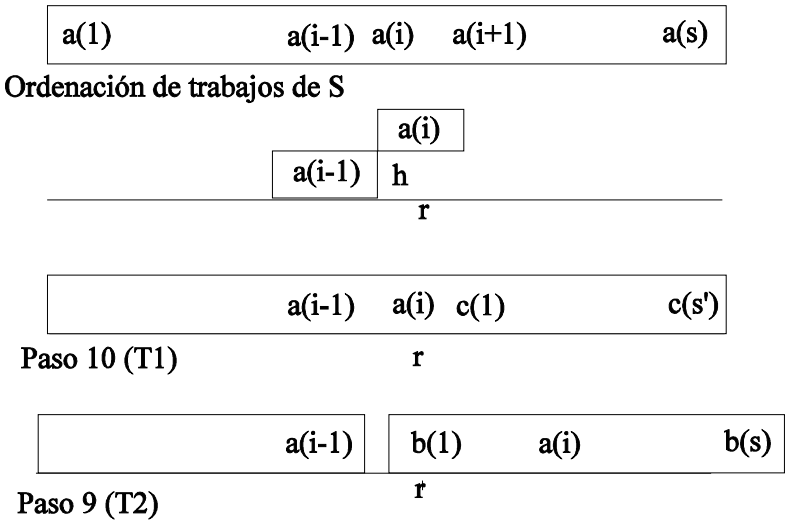


Figura 4. Idea del algoritmo $1|r_j|\sum T_j$

Algoritmo $1|r_j|\sum T_j$

1. Ordenar los trabajos en orden creciente de fechas de disponibilidad r_j. Sea $l(1),...,l(n)$ dicho orden. Sea R la lista ordenada de trabajos secuenciados. Inicialmente la lista esta vacía. Poner $k = 0$ (número de trabajos completados), y $J = \{1,...,n\}$.

2. Determinar $S = \{j \in J / r_j = r_{l(1)}\}$. Sea $s = card(S)$. Poner $C = 0$.

3. Obtener la ordenación heurística $[a(1),...,a(s)]$ de los trabajos del conjunto S dada por el algoritmo $1||\sum T_j$. Poner $i = 1$.

4. - Si $i = s+1$ ó $k+s = n$, ir al paso 6.
   - En otro caso, poner $C = C + p_{a(i)}$.
     - Si $C \leq r_{l(k+s+1)}$ ir al paso 5
     - Si $C > r_{l(k+s+1)}$ calcular $h = r_{l(k+s+1)} - C + p_{a(i)}$, $C = C - p_{a(i)}$ e ir al paso 6.

5. Poner $i = i+1$ y volver al paso 4.

6. - Si $i = s+1$ ó $k+s = n$, añadir $a(1), a(2), ...,a(s)$ en este orden a la lista R y quitarlos de J. Poner $k = k+s$.
     - Si $J = \varnothing$ ir al paso 12

- Si $J \neq \varnothing$ calcular $S = \{j \in J / r_j = r_{l(k+1)}\}$, $s = card(S)$, tomar $d'_j = d_j - r_{l(k+1)}$ $\forall j \in S$, y retornar al paso 3.

- En otro caso, añadir *a(1), a(2), ..., a(i-1)* a la lista *R* en ese orden. Poner $j_o = i$, $l = 0$.

7. - si $j_o = s$ ir al paso 8.
   - si $j_o \neq s$ plantear la siguiente pregunta:
   ¿existe algún trabajo con índice $j \in \{j_o+l,...,s\}$ *tal que* $p_{a(j)} \leq h$ *y* $C \geq r_{a(j)}$ *?*
   - SI, entonces, sea $a(j_1)$ dicho trabajo con j_1 el menor índice. Poner $j_o = j_1$. Colocar $a(j_o)$ en *R*. Hacer $h = h - p_{a(j_o)}$, $l = l + 1$, $C = C + p_{a(j_o)}$, y volver al comienzo del paso 7.
   - NO ó el proceso anterior termina, entonces, ir al paso 8.

8. Quitar de *J* los trabajos que están en *R*.
   - Si $J = \varnothing$ ir al paso 12
   - Si $J \neq \varnothing$ sea *Q* el conjunto de trabajos de *S* que no están en *R*. Calcular $S = \{j \in J / r_j \leq r_{l(k+s+1)}\}$, Llamar $s = card(S)$, $r = r_{l(k+s+1)}$, $k = k + i - 1 + l$.
     - si $s = 1$, poner $i = 1$ e ir al paso 4.
     - si $s \neq 1$, ir al paso 9.

9. Poner $d'_j = d_j - r$ $\forall j \in S$ y obtener la ordenación heurística *b(1),...,b(s)* de los trabajos del conjunto *S* dada por el algoritmo $1||\sum Tj$. Sea *T2* la tardanza de dicha ordenación.
   - si $b(1) \notin Q$ ir al paso 10.
   - si $b(1) \in Q$, poner $h_{b(0)} = h$ *y* $C'_{b(0)} = C$. Calcular recursivamente para $j = 1,...,s$:

$$h_{b(j)} = min \{h_{b(j-1)}, C'_{b(j-1)} + h_{b(j-1)} - r_{b(j)}\}$$

$$C'_{b(j)} = C + h + \sum_{q=1}^{j} p_{b(q)} - h_{b(j)}$$

   Actualizar $T_{b(j)} = max \{0, L_{b(j)} = C'_{b(j)} - d_{b(j)}\}$ $\forall j \in \{1,...,s\}$ y tomar $T2 = \sum_{j=1}^{s} T_{b(j)}$.

10. Tomar $S' = S - \{a(i)\}$, $s' = s - 1$, calcular $C_{a(i)} = C + p_{a(i)}$, $d'_j = d_j - C_{a(i)}$ $\forall j \in S'$ y obtener la ordenación heurística *c(1), c(2), ..., c(s')* de los trabajos de S' dada por el algoritmo $1||\sum Tj$. Sea *T* la tardanza dada por el algoritmo. Calcular $T_{a(i)} = max \{0, C_{a(i)} - d_{a(i)}\}$ y poner $T1 = T + T_{a(i)}$ (tardanza asociada a la ordenación *a(i), c(1), ..., c(s'))*.

11. - Si $T1 < T2$, entonces poner $a(1) = a(i)$, $a(2) = c(1)$, ..., $a(s) = c(s')$, $i = 1$, e ir al paso 4.

- Si $T1 \geq T2$, entonces poner $a(1) = b(1), a(2) = b(2), ..., a(s) = b(s), i = 1$, e ir al paso 4.

12. *Parar*. La lista R da el orden en que se deben procesar los trabajos. La propia secuencia nos determina los tiempos ociosos sin más que comparar los tiempos de completación con los r_j.

A continuación se estudia la complejidad teórica de este algoritmo.

Complejidad del Algoritmo

La complejidad del algoritmo $1|r_j|\sum T_j$ viene marcada por la complejidad de los pasos 1, 3, 9 y 10. El paso 1 es una ordenación y, por tanto, tiene complejidad $O(n \log n)$. En los pasos 3, 9 y 10 hay sendas aplicaciones del algoritmo $1||\sum Tj$. En el paso 9 quizás sea necesario además un nuevo cálculo de la tardanza $T2$ con a lo sumo $O(n)$ operaciones. Por tanto, la complejidad esta acotada superiormente por el número de veces que se aplica el algoritmo $1||\sum Tj$.

Cada aplicación del algoritmo $1||\sum Tj$ es siempre a trabajos de un determinado $S \subset \{1,...,n\} - R$ siendo R el conjunto de trabajos ya planificados. El contenido es estricto, es decir, el subconjunto es propio, salvo para el último S, ya que en caso contrario todos los r_j serían iguales y estaríamos ante el problema $1||\sum Tj$.

El número de S distintos esta acotado superiormente por el número de valores distintos de los r_j, es decir, n. A cada S se le aplica el algoritmo $1||\sum T_j$ una única vez. Si

$$s_{max} = \max_{\substack{S \subset \{1,...,n\} \\ S \text{ considerado}}} \{card(S)\}$$

se tiene que la complejidad de $1|r_j|\sum T_j$ es del orden de

$$\text{(número de S distintos)}\ s_{max}^2 \sum_{j=1}^{n} p_j$$

que en cualquier caso esta acotada por $O(n^3 \sum p_j)$.

3.3. Problema 1|rj,pmtn|∑Tj

Planteamiento del problema

Como en el caso anterior, el problema consiste en procesar n trabajos independientes sobre una máquina. En este caso, todos los trabajos pueden ser interrumpibles, es decir, la ejecución de cualquier trabajo puede pararse en un instante dado y retomarse posteriormente a partir de ese instante sin necesidad de volver a procesar lo que se haya hecho hasta el instante de interrupción. Cada trabajo j queda caracterizado por su tiempo de procesamiento p_j, su fecha límite d_j en la que debería estar terminado, y su fecha de disponibilidad r_j que fija el instante a partir del cual puede comenzar a procesarse. Supuesta una ordenación de los trabajos, podemos calcular para cada trabajo j el tiempo de completación C_j y la correspondiente tardanza $T_j = max\ \{0,\ C_j - d_j\}$. El objetivo será determinar una secuenciación de trabajos que minimice la tardanza total $\sum Tj$.

Algoritmo

El algoritmo que a continuación se propone mantiene la notación e ideas del algoritmo dado para el problema anterior, e incorpora la característica de interrumpibilidad de los trabajos.

Algoritmo 1\|rj,pmtn\|∑Tj

1. Ordenar los trabajos en orden creciente de fechas de disponibilidad rj. Sea $l(1),...,l(n)$ dicho orden. Sea R la lista ordenada de trabajos secuenciados. Inicialmente la lista esta vacía. Poner $k = 0$ (número de trabajos completados), y $J = \{1,...,n\}$.

2. Determinar $S = \{j \in J / r_j = r_{l(1)}\}$. Sea $s = card(S)$. Poner $C = 0$.

3. Obtener la ordenación heurística $[a(1),...,a(s)]$ de los trabajos del conjunto S dada por el algoritmo $1||\sum Tj$. Poner $i = 1$.

4. - Si $i = s+1$ ó $k+s = n$, ir al paso 6.
   - En otro caso, poner $C = C + p_{a(i)}$.
     - Si $C \leq r_{l(k+s+1)}$ ir al paso 5
     - Si $C > r_{l(k+s+1)}$ poner $p_{a(i)} = C - r_{l(k+s+1)}$, $C = r_{l(k+s+1)}$ e ir al paso 6.

5. Poner $i = i+1$ y volver al paso 4.

6. - Si $i = s+1$ ó $k+s = n$, añadir $a(1), a(2), ...,a(s)$ en este orden a la lista R y quitarlos de J. Poner $k = k+s$.
     - Si $J = \varnothing$ ir al paso 8

- Si $J \neq \varnothing$ calcular $S = \{j \in J / r_j = r_{l(k+1)}\}$, $s = card(S)$, tomar $d'_j = d_j - r_{l(k+1)}$ $\forall j \in S$, y retornar al paso 3.
- En otro caso, ir al paso 7.

7. Añadir *a(1), a(2), ..., a(i)* a la lista *R* en ese orden, quitando de *J* los trabajos *a(1), a(2), ..., a(i-1)*.
   - Si $J = \{a(i)\}$ ir al paso 8.
   - Si $J \neq \{a(i)\}$ calcular $S = \{j \in J / r_j \leq r_{l(k+s+1)}\}$, $s = card(S)$, poner $k = k + i - 1$, tomar $d'_j = d_j - r_{l(k+s+1)}$ $\forall j \in S$, y retornar al paso 3.

8. *Parar*. La lista *R* da el orden en que se deben procesar los trabajos, siendo $r_{l(1)}, r_{l(2)}, ..., r_{l(n)}$ los puntos de posible interrupción de los mismos.

Analicemos seguidamente la complejidad teórica de este algoritmo.

Complejidad del algoritmo

La complejidad del algoritmo propuesto para el problema $1|rj,pmtn|\sum Tj$ es $O(n^3 \sum p_j)$ puesto que utiliza a lo sumo *n* veces el algoritmo dado para el problema $1||\sum Tj$ en este mismo capítulo con complejidad $O(n^2 \sum p_j)$.

3.4. Problema 1|rj,pmtn(A)|∑Tj

Planteamiento del problema

Como en los casos anteriores, el problema consiste en procesar n trabajos independientes sobre una máquina. En este caso, sólo los trabajos de cierto subconjunto $A \subseteq J$ pueden ser interrumpibles, es decir, la ejecución de cualquier trabajo de *A* puede pararse en un instante dado y retomarse posteriormente a partir de ese instante sin necesidad de volver a procesar lo que se haya hecho hasta el instante de interrupción. El subconjunto *A* es conocido a priori. Cada trabajo *j* queda caracterizado por su tiempo de procesamiento p_j, su fecha límite d_j en la que debería estar terminado, y su fecha de disponibilidad r_j que fija el instante a partir del cual puede comenzar a procesarse. Supuesta una ordenación de los trabajos, podemos calcular para cada trabajo *j* el tiempo de completación C_j y la correspondiente tardanza $T_j = max\{0, C_j - d_j\}$. El objetivo será determinar una secuenciación de trabajos que minimice la tardanza total $\sum Tj$.

Algoritmo

En el algoritmo que se expone a continuación, se combinan los algoritmos propuestos para los problemas $1|r_j|\sum T_j$ y $1|r_j,pmtn|\sum T_j$ en función de que el trabajo en consideración *a(i)* sea o no interrumpible.

Algoritmo $1|r_j,pmtn(A)|\sum T_j$

1. Ordenar los trabajos en orden creciente de fechas de disponibilidad r_j. Sea $l(1),...,l(n)$ dicho orden. Sea R la lista ordenada de trabajos secuenciados. Inicialmente la lista esta vacía. Poner $k = 0$ (número de trabajos completados), y $J = \{1,...,n\}$.

2. Determinar $S = \{j \in J / rj = r_{l(1)}\}$. Sea $s = card(S)$. Poner $C = 0$.

3. Obtener la ordenación heurística $[a(1),...,a(s)]$ de los trabajos del conjunto S dada por el algoritmo $1||\sum Tj$. Poner $i = 1$.

4. - Si $i = s+1$ ó $k+s = n$, ir al paso 6.

   - En otro caso, poner $C = C + p_{a(i)}$.
     - Si $C \leq r_{l(k+s+1)}$ ir al paso 5
     - Si $C > r_{l(k+s+1)}$ y $a(i) \in A$, poner $p_{a(i)} = C - r_{l(k+s+1)}$, $C = r_{l(k+s+1)}$ e ir al paso 6.
     - Si $C > r_{l(k+s+1)}$ y $a(i) \notin A$, calcular $C = C - p_{a(i)}$ y $h = r_{l(k+s+1)} - C$ e ir al paso 6.

5. Poner $i = i+1$ y volver al paso 4.

6. - Si $i = s+1$ ó $k+s = n$, añadir $a(1), a(2), ...,a(s)$ en este orden a la lista R y quitarlos de J. Poner $k = k+s$.
     - Si $J = \varnothing$ ir al paso 12
     - Si $J \neq \varnothing$ calcular $S = \{j \in J / r_j = r_{l(k+1)}\}$, $s = card(S)$, tomar $d'_j = d_j - r_{l(k+1)}$ $\forall j \in S$, y retornar al paso 3 (tomando como datos de entrada los nuevos p_j y los d'_j).

   - En otro caso, añadir $a(1), a(2), ..., a(i-1)$ en este orden a la lista R y quitarlos de J.

7. - Si $a(i) \in A$, entonces añadir $a(i)$ a la lista R.
     - Si $J = \{a(i)\}$ ir al paso 12.

- Si $J \neq \{a(i)\}$ calcular $S = \{j \in J / r_j \leq r_{l(k+s+1)}\}$, $s = card(S)$, poner $k=k+i-1$, tomar $d'_j = d_j - r_{l(k+s+1)}$ $\forall j \in S$, y retornar al paso 3.

- Si $a(i) \notin A$, y $J = \varnothing$ ir al paso 12.

- Si $a(i) \notin A$, y $J \neq \varnothing$, entonces sea Q el conjunto de trabajos de S que no están en R. Calcular $S = \{j \in J / r_j \leq r_{l(k+s+1)}\}$, $s = card(S)$. Llamar $r = r_{l(k+s+1)}$, $k=k+i-1$.
  - si $s = 1$, poner $i = 1$ e ir al paso 4.
  - si $s \neq 1$, ir al paso 8.

8. Tomar $S' = S - \{a(i)\}$, $s' = s - 1$. Obtener la ordenación heurística $[c(1), ..., c(s')]$ de los trabajos de S' dada por el algoritmo $1||\sum Tj$. Sea $T1$ la tardanza asociada a la ordenación $[a(i),c(1),c(2),...,c(s')]$. (Al algoritmo $1||\sum Tj$ se le envía $d'_j = d_j - C_{a(i)}$ donde $C_{a(i)} = C + p_{a(i)}$, y nos proporciona una tardanza a la que sumamos $T_{a(i)} = max\{0, C_{a(i)} - d_{a(i)}\}$ para obtener $T1$).

9. Obtener la ordenación heurística $[b(1),...,b(s)]$ de los trabajos del conjunto S dada por el algoritmo $1||\sum Tj$. Sea $T2$ la tardanza total de dicha ordenación. (Nótese que debe tomarse $d'_j = d_j - r$ $\forall j$ antes de llamar al algoritmo $1||\sum Tj$).

   - si $b(1) \notin Q$, poner $j_o = i$ e ir al paso 10.

   - si $b(1) \in Q$, poner $h_{b(0)} = h$ y $C'_{b(0)} = C$. Calcular recursivamente, para $j = 1,2,...,s$, los valores:

$$h_{b(j)} = min\{h_{b(j-1)}, C'_{b(j-1)} + h_{b(j-1)} - r_{b(j)}\}$$

$$C'_{b(q)} = C + h + \sum_{q=1}^{j} p_{b(q)} - h_{b(j)}.$$

   Actualizar $T_{b(j)} = max\{0, L_{b(j)} = C'_{b(j)} - d_{b(j)}\}$ $\forall j \in \{1,2,...,s\}$ y tomar $T2 = \sum_{j=1}^{s} T_{b(j)}$. Ir al paso 11.

10. - Si $j_o = s$, ir al paso 11.

    - Si $j_o \neq s$, plantear la siguiente pregunta:
    ¿existe algún trabajo con índice $j \in \{j_o + 1,...,s\}$ tal que $C \geq r$ y que cumpla alguna de las condiciones siguientes: $b(j) \in A$ ó $p_{b(j)} \leq h$?

      - SI, sea entonces $b(j_1)$ dicho trabajo con j_1 el menor índice. Poner $j_o = j_1$. Colocar $b(j_o)$ en R.
        - si $p_{b(j_o)} \leq h$, poner $k=k+1$, hacer $h = h - p_{b(j_o)}$, $C = C + p_{b(j_o)}$ y volver al comienzo del paso 10.

- si $p_{b(j_o)} > h$, hacer $C = C + h$, $p_{b(j_o)} = p_{b(j_o)} - h$, $h = 0$, y, en base a la nueva ordenación de los trabajos *b(j)*, calcular la nueva tardanza y ponerla en *T2*. Ir al paso 11.

- NO, ó el proceso anterior termina, en este caso, y en base a la nueva ordenación de los trabajos *b(j)*, calcular la nueva tardanza y ponerla en *T2*. Ir al paso 11.

11. - Si $T1 < T2$, entonces poner $a(1) = a(i), a(2) = c(1), ..., a(s) = c(s'); i = 1$, e ir al paso 4.

- Si $T1 \geq T2$, entonces poner $a(1) = b(1), a(2) = b(2), ..., a(s) = b(s); i = 1$, e ir al paso 4.

12. *Parar*. La lista *R* da el orden en que se deben procesar los trabajos. La propia secuencia nos determina los tiempos muertos sin más que comparar los tiempos de completación con los r_j .

A continuación se presenta la complejidad teórica de este algoritmo.

Complejidad del algoritmo

La complejidad del algoritmo propuesto para el problema $1|rj,pmtn(A)|\sum Tj$ es $O(n^3 \sum pj)$ puesto que utiliza a lo sumo n veces el algoritmo dado para el problema $1||\sum Tj$ en este mismo capítulo con complejidad $O(n^2 \sum pj)$.

3.5. Problema 1|prec|∑Tj

Planteamiento del problema

Sean *n* trabajos que se desean procesar sobre una máquina, los cuales cumplen las siguientes características: todos están disponibles para ser procesados en cualquier momento y una vez iniciada la realización de un trabajo, éste no puede interrumpirse para comenzar ningún otro. La secuenciación de los trabajos debe adecuarse a unas relaciones de precedencia entre trabajos fijada a priori. Dicha relación es recogida en un grafo acíclico *G* cuyos nodos sean los trabajos numerados de tal forma que si *(i,j)* es una arista del grafo, se entenderá que *i* debe ir siempre antes que *j* en cualquier planificación factible.

Denotaremos por $\Gamma(j)$ al conjunto de trabajos que deben realizarse posteriormente a la completación del trabajo *j*, esto es, el conjunto de vértices

siguientes a j en el grafo *G*. Dar una estructura de precedencias *R* sobre los trabajos, equivaldrá a especificar los correspondientes conjuntos $\Gamma(j)\ \forall j$.

Una secuencia *S* se dice que es compatible con la estructura de precedencia *R*, si no vulnera ninguna de las restricciones de precedencia fijadas en *R*, es decir, no podrán existir dos trabajos con la ordenación *"i anterior a j"* en la secuencia *S*, si $i \in \Gamma(j)$.

Cada trabajo *j* lleva asignado un tiempo de procesamiento p_j conocido y una fecha límite d_j dada. Si no se ha completado el procesamiento de un trabajo antes de su fecha límite, se incurre en una tardanza la cuál es medida por el número de unidades de tiempo en que se ha superado dicha fecha.

El tiempo de completación del trabajo *j,* esto es, C_j varía en relación con la posición que ocupa el trabajo *j* en la secuencia correspondiente de trabajos. Así, si dicho trabajo ocupase el lugar *k*, entonces $C_j = \sum_{i=1}^{k} p_{[i]}$, siendo *[i]* el trabajo que estaría en la posición *i*. Recordemos que la tardanza de un trabajo *j* viene dada por la expresión $T_j = max\{0, L_j = C_j - d_j\}$. El problema consiste en encontrar una ordenación o secuencia *[[1],...,[n]]* de trabajos, compatible con la estructura de precedencias, que minimice la tardanza total $\sum_{j=1}^{n} T_{[j]}$.

Métodos exactos que permiten encontrar una solución óptima para el problema pueden verse en Baker (1974), Baker y Martin (1974), Lawler (1977) y Potts y Van Wassenhove (1987). El problema es que tales procedimientos requieren un gran esfuerzo computacional cuando el número de trabajos es de varias decenas. De hecho, Graham y otros (1979) catalogan la complejidad del problema $1|prec|\sum Tj$ como superior a la del problema $1||\sum Tj$ en el que no se consideran relaciones de precedencia entre los trabajos. Puesto que Du y Leung (1990) probaron que $1||\sum Tj$ es NP-duro mediante una reducción polinomial al problema de la partición, podemos afirmar que $1|prec|\sum Tj$ es un también un problema NP-duro. En tales circunstancias los métodos heurísticos pueden ofrecer soluciones aceptables ya que la pérdida de bondad de la solución obtenida queda contrarrestada por la notable mejora del tiempo invertido en su búsqueda.

Se describe a continuación dos métodos heurísticos que guíen la búsqueda de una solución aproximada para el problema $1|prec|\sum Tj$.

Algoritmo A

Antes de describir paso a paso el algoritmo propuesto, comentemos brevemente las ideas generales del mismo. La estructura de precedencias entre los trabajos permite clasificar los mismos en *k* niveles de forma que, la ejecución de

cualquier trabajo del nivel i-ésimo, requiere haber realizado previamente al menos un trabajo del nivel *i-1*.

Dentro de cada nivel se considera la correspondiente secuencia *EDD* (ordenación de menor a mayor de los diversos trabajos de acuerdo con las fechas límites) y posteriormente se agrupan las diferentes ordenaciones en un sólo vector, poniendo en primer lugar las de menor nivel. Dicho vector será nuestra secuencia semilla.

Si en dicha secuencia hubiesen dos o más trabajos de un mismo nivel con fechas límites iguales, se ordenan estos trabajos de acuerdo con sus tiempos de proceso de menor a mayor. El organigrama del algoritmo puede verse en la figura

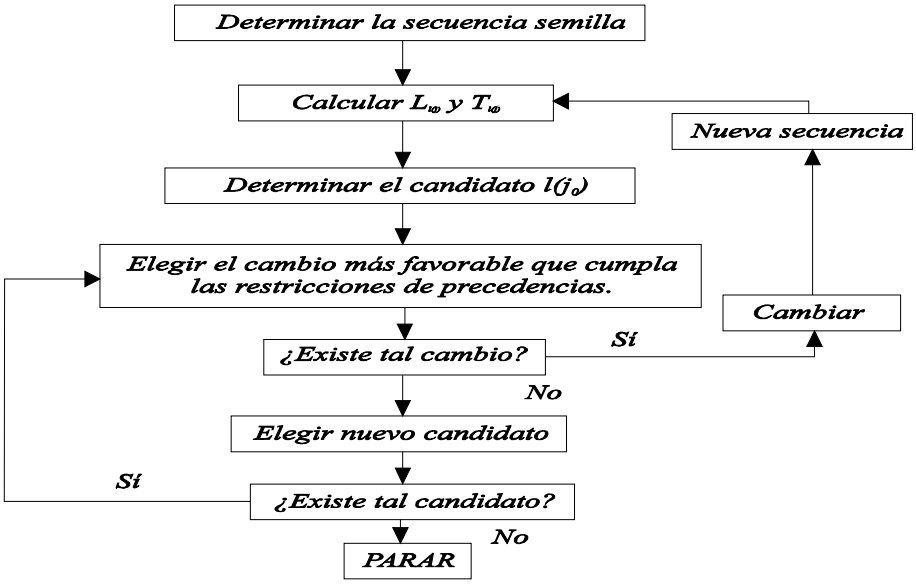


Figura 5. Organigrama del algoritmo A.

Tras determinar la secuencia inicial, se calculan las tardanzas de los trabajos y se selecciona el trabajo candidato a cambiar de lugar (paso 3). Posteriormente se decide cuál debe ser el cambio más favorable, es decir, si el trabajo elegido debe adelantarse ó retrasarse, teniendo siempre presente que el cambio seleccionado debe ser compatible con la estructura de precedencias establecida inicialmente.

La posible ganancia en la reducción de la tardanza, si adelantáramos el trabajo, viene dada en los pasos 5 y 6; mientras que si se retrasara, la bondad de tal medida es cuantificada por los pasos 8 y 9 del algoritmo.

La decisión de efectuar el cambio, siempre al lugar que ofrece una mayor reducción, se toma en el paso 10. Si desistimos de hacerlo, será debido a que no decrece la tardanza total y, en consecuencia, elegiremos un nuevo trabajo

candidato a cambiar de lugar. El proceso finaliza cuando ningún trabajo proporcione un cambio que permita reducir la tardanza total.

Algoritmo 1|prec|$\sum T_j$ modo A

1. Establecer la ordenación de trabajos por niveles de acuerdo con el grafo acíclico que fija las relaciones de precedencia. Sea n_i el número de trabajos del nivel *i*-ésimo y consideremos que *k* representa el número de niveles que hay en el grafo acíclico.

2. Ordenar los trabajos de cada nivel en orden *EDD*, y tomar como secuencia semilla a la ordenación resultante que denotaremos por:
   $[[1],[2],...,[n_1];[n_1+1],...,[n_1+n_2];[n_1+n_2+1],...,[n_1+n_2+n_3];...;[n-n_k+1],...,[n]$
   Calcular $L_{[j]} = \sum_{i=1}^{j} p_{[i]} - d_{[j]}$ y $T_j = max\ \{0,\ L_{[j]}\}\ \forall j = 1,...,n$. Hallar $T = \sum_{j=1}^{n} T_{[j]}$. Poner $P = \varnothing$ (representa los trabajos que han sido probados).

3. Elegir el índice j_o tal que $L_{[j_o]} = max\ \{L_{[j]} / 1 \leq j \leq n\}$. Dicho trabajo $[j_o]$ es el candidato a cambiar de lugar.

4. Si $j_o = 1$, ir al paso 7. Si $j_o \neq 1$, pero $T_{[j_o]} = 0$, ir al paso 7. En otro caso, poner $m=1,\ G=0,\ q=0,\ A^0=0,\ B_{j_o}=0$. Calcular m_1 el índice mínimo para el cual $[j_o] \in \Gamma(j_o\text{-}m_1)$; si no existe dicho valor, colocar $m_1 = j_o$.

5. *(Adelantar)*. Si $m = j_o$, ó $m = m_1$ ir al paso 7. En otro caso calcular los valores $\Delta T_{j_o\text{-}m}$, A^m, $B_{j_o\text{-}m}$, D^m de igual forma que en el paso 4 del algoritmo *1||*$\sum T_j$:

$$\Delta T_{j_o-m} = \begin{cases} p_{[j_o]}, & si \quad L_{[j_o-m]} > 0 \\ 0, & si \quad L_{[j_o-m]} \leq 0 \ y \ p_{[j_o]} \leq \left| L_{[j_o-m]} \right| \\ L_{[j_o-m]} + p_{[j_o]}, & si \quad L_{[j_o-m]} \leq 0 \ y \ p_{[j_o]} > \left| L_{[j_o-m]} \right| \end{cases}$$

$$A^m = A^{m-1} + \Delta T_{j_o\text{-}m} \ ; \ B_{j_o\text{-}m} = B_{j_o\text{-}m+1} + p_{[j_o\text{-}m]}$$

$$D^m = min\ \{B_{j_o\text{-}m}\ , T_{[j_o]}\} - A^m.$$

6. - Si $D^m = T_{[j_o]}$, entonces poner $q = j_o - m,\ G = D^m$, e ir al paso 7.
   - Si $D^m \neq T_{[j_o]}$ y $D^m > G$, entonces poner $q = j_o - m,\ G = D^m$. Hacer $m = m + 1$ y volver al paso 5.
   - Si $D^m \neq T_{[j_o]}$ y $D^m \leq G$, entonces hacer $m = m + 1$ y volver al paso 5.

7. Si $j_o = n$, ir al paso 10. Si $j_o \neq n$, poner $m = 1$, $\Delta C_o = 0$, y $Z^o = 0$. Sea m_2 el índice mínimo para el cual $[j_o + m_2] \in \Gamma([j_o])$. Si no existe tal valor, poner $m_2 = n - j_o + 1$.

8. *(Retrasar)*. Si $m = n - j_o + 1$, ó $m = m_2$, ir al paso 10. En otro caso, calcular los valores ΔC_m , Y^m , X_{j_o+m} , Z^m y D^m usando las fórmulas dadas en el paso 7 del algoritmo $1||\Sigma T_j$:

$$\Delta C_m = \Delta C_{m-1} + p_{[j_o+m]}$$

$$Y^m = \begin{cases} \Delta C_m, & si \quad L_{[j_o]} > 0 \\ max\left\{0, L_{[j_o]} + \Delta C_m\right\}, & si \quad L_{[j_o]} \leq 0 \end{cases}$$

$$X_{j_o+m} = \begin{cases} 0, & si \quad L_{[j_o+m]} \leq 0 \\ min\left\{L_{[j_o+m]}, p_{[j_o]}\right\}, & si \quad L_{[j_o+m]} > 0 \end{cases}$$

$$Z^m = Z^{m-1} + X_{j_o+m}; \; D^m = Z^m - Y^m.$$

9. - Si $D^m > G$, poner $q = j_o + m$ y $G = D^m$. Hacer $m = m + 1$ y volver al paso 8.
   - Si $D^m \leq G$, hacer $m = m + 1$ y volver al paso 8.

10. *(Cambiar)*. Si $q = 0$, no hacer ningún cambio. Colocar $P = P \cup \{j_o\}$ e ir al paso 11 (búsqueda de un nuevo candidato).
    - Si $q > 0$, cambiar el trabajo $[j_o]$ al lugar q y *"rodar"* los trabajos que sean necesarios, esto es: sea $MIN = min\ \{q, j_o\}$, y $MAX = max\ \{q, j_o\}$.
    - si $MIN = q$, entonces $s(q) = [j_o]$, $s(q+1) = [q]$, ..., $s(q+k) = [q+k-1]$, ..., $s(j_o) = [j_o-1]$.
    - si $MIN = j_o$, entonces $s(j_o) = [j_o+1]$, ..., $s(j_o+k) = [j_o+k+1]$, ..., $s(q-1) = [q]$, $s(q) = [j_o]$.
    Antes de salir de este paso, recolocar los trabajos en modo que el trabajo de la posición i de la secuencia sea $[i] = s(i), \; \forall i \in [MIN, MAX]$.
    Volver a calcular $L_{[j]}$ y $T_{[j]}$ para $j \in [MIN, MAX]$ según las fórmulas del paso 2. Poner $T = T - G$, $P = \varnothing$ e ir al paso 3.

11. - Si $card(P) = n$, *Parar*. La secuencia $[[1],[2],...,[n]]$ nos da la solución, siendo T el valor de la tardanza total para dicha secuencia.
    - Si $card(P) \neq n$, entonces calcular el j_1 tal que $L_{[j_1]} = max\ \{L_{[k]} / k \notin P\}$. Hacer $j_o = j_1$ y volver al paso 4.

Estudiemos ahora la complejidad de este algoritmo.

Complejidad del algoritmo

El algoritmo tiene una complejidad $O(n^2 \sum_{j=1}^{n} p_j)$. $O(n^2)$ operaciones necesita el paso 1, y $O(n \log n)$ operaciones requiere el paso 2. Los pasos del 3 al 11 requieren $O(n)$ operaciones por cada iteración. El número de iteraciones para estos pasos es a lo sumo de $NI = \sum_j T_j(Semilla) - T_{max}(EDD)$, siendo $T_j(Semilla)$ la tardanza del trabajo j correspondiente a la secuencia inicial y $T_{max}(EDD)$ la tardanza máxima para la secuencia *EDD*. Habrá que determinar el valor que puede tomar dicha expresión, aunque siempre estará acotada por $n\sum_{j=1}^{n} p_j$. Por tanto, la repetición de los pasos tercero al undécimo tiene una complejidad $O(n^2 \sum_{j=1}^{n} p_j)$ Dicha expresión será la complejidad del algoritmo.

El valor *NI* se calcula con sólo realizar los pasos 1 y 2. Con ese dato podemos determinar la rapidez de la heurística, sin tener necesidad de realizar los siguientes pasos.

Veamos ahora un ejemplo de ejecución de este algoritmo.

Ejemplo 3

Consideremos el problema de secuenciar ocho trabajos, cuyos tiempos de proceso y fechas límites se recogen en la siguiente tabla:

j	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***
p_j	*121*	*147*	*102*	*79*	*130*	*83*	*96*	*88*
d_j	*260*	*269*	*400*	*266*	*337*	*336*	*683*	*719*

Las relaciones de precedencia vienen dadas por el grafo acíclico:

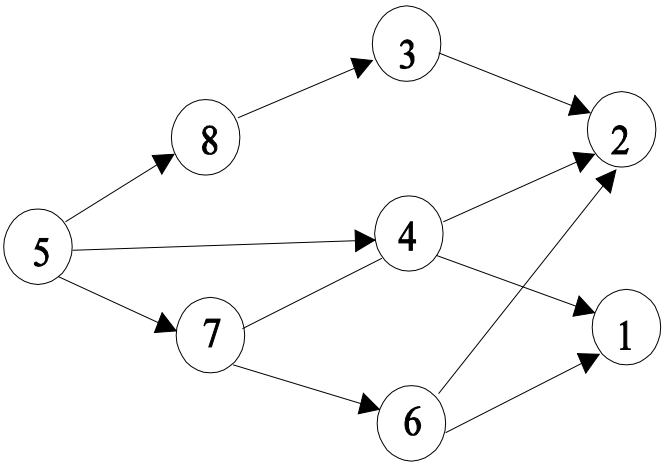


Figura 6. Grafo de precedencias para el ejemplo 3.

Siguiendo el algoritmo propuesto, la secuencia semilla será:

5	7	8	4	6	3	1	2

a la que corresponden los L_j:

{-207, -457, -405, 127, 140, 178, 439, 577},

y una tardanza de 1461.

Inicialmente, el candidato a cambiar es el trabajo 2 pero el mismo no ofrece cambio favorable; idéntica situación se presenta para los trabajos 1 y 3. La elección del trabajo 6 nos proporciona una reducción de la tardanza en 57 unidades si instalamos dicho trabajo en la posición 3 de la secuencia, obteniendo así la nueva secuencia

5	7	6	8	4	3	1	2

De la misma forma se descartan los candidatos 2 y 1, y se elige como nuevo candidato el trabajo número 4. Dicho trabajo se coloca en la tercera posición por ser el lugar que otorga mayor descenso en la tardanza total, quedando ésta en 1285. La nueva secuencia será

5	7	4	6	8	3	1	2

Repitiendo el proceso, el algoritmo finaliza con la secuencia

5	7	4	6	1	8	3	2

con tardanza 1216.

Nótese que a lo largo del desarrollo del ejemplo, las diversas soluciones parciales que se van obteniendo verifican las restricciones de precedencia establecidas en el grafo de partida.

Algoritmo B

Se propone una segunda heurística de búsqueda local para obtener solución al problema planteado. Como en el algoritmo anterior se clasifican los trabajos en varios niveles. Los trabajos de nivel 0 son los trabajos sin predecesores en el grafo de precedencias. Un trabajo *j* pertenece al nivel *k* si el camino con máximo número de arcos entre todos los caminos desde los trabajos de nivel 0 al trabajo *j* tiene exactamente *k* arcos.

Como la tardanza total es una medida de actuación regular, en el problema de una máquina existe una secuencia óptima en la que no hay tiempo ocioso de la máquina, y no puede obtenerse ninguna mejora permitiendo interrupciones. Por tanto, la búsqueda se reduce a planificaciones permutación.

Puede decirse que una planificación permutación es compatible si el orden de los trabajos no altera las restricciones de precedencia. Se considera un *movimiento* a cualquier cambio del orden de los trabajos en una secuencia permutación. Si se aplica un movimiento a una secuencia compatible y se obtiene otra secuencia compatible, el movimiento se denomina un *movimiento factible ó compatible.*

El tipo de movimiento que considera este algoritmo es el siguiente: primero, se extrae el trabajo con máxima demora en la secuencia actual y se inserta en otra posición en la secuencia. Un movimiento de esta clase proporciona una variación de la tardanza total asociada a la secuencia. Esta variación se computa por el Teorema dado en la sección 3.1 de este mismo capítulo.

La diferencia esencial entre el algoritmo que a continuación se propone y el algoritmo anterior radica en la planificación semilla y en el modo de ordenar los trabajos en cada nivel al construir las sucesivas planificaciones permutación. Como secuencia semilla se propone elegir una de las siguientes opciones dadas en Alcaide, Sicilia y Ramos (1992):

LEDD ("Leveled Earliest Due Date") que consiste en ordenar los trabajos por niveles, y, dentro de cada nivel, en orden creciente de fechas límite.

PNBR ("Precedence Net Benefit of Relocation") que consiste en, partiendo de una secuencia *LEDD*, aplicar la heurística *NBR* propuesta por Holsenback y Russell (1992) para el problema $1||\sum Tj$, pero con una pequeña modificación para contemplar la estructura de precedencias existente entre los trabajos: el conjunto de trabajos candidatos a replanificar en su algoritmo queda restringido a aquellos trabajos cuya replanificación es un movimiento factible.

La razón de considerar estas opciones se justifica por el hecho de que la heurística *NBR* se comporta bien en el problema sin restricciones de precedencia. La opción *LEDD* se toma en cuenta porque puede ser útil en varios casos. Por ejemplo, si el número de trabajos por nivel es suficientemente pequeño en relación con el número de niveles del grafo, la secuencia *LEDD* proporciona una solución próxima a la secuencia *PNBR* en un tiempo de ejecución menor.

Algoritmo $1|prec|\sum T_j$ modo B

1. Elegir una secuencia semilla compatible como secuencia actual.

2. Encontrar el trabajo con máxima demora en la secuencia actual. Este es el trabajo candidato para generar un movimiento factible. Marcar el trabajo candidato.

3. Determinar la mejor posición compatible para el trabajo candidato de acuerdo con la heurística dada en Sicilia y Alcaide (1989) y construir la nueva secuencia.
   - si la nueva secuencia es mejor que la secuencia actual, actualizar la secuencia actual, eliminar las marcas y volver al paso 2.
   - en otro caso, continuar en el paso 4.

4. Encontrar el trabajo no marcado con máxima demora en la secuencia actual.
   - si tal trabajo no existe, porque todos los trabajos están marcados, parar. La secuencia actual es la salida del algoritmo.
   - en otro caso, tomar dicho trabajo como trabajo candidato, marcar el trabajo y retornar al paso 3.

   Seguidamente se estudia la complejidad de este algoritmo.

Complejidad del algoritmo

La complejidad del algoritmo es $O(n^2 \sum_{j=1}^{n} p_j)$. El paso 1 puede computarse en orden de tiempo de a lo sumo $O(n^2)$, independientemente de que la secuencia inicial sea *LEDD* ó *PNBR*. Los pasos del 2 al 4 requieren un tiempo del orden $0(n)$ por cada iteración. El número de iteraciones para estos pasos es a lo sumo de $\sum_j T_j(Semilla)$, siendo $T_j(Semilla)$ la tardanza del trabajo j correspondiente a la secuencia inicial. Esta cota superior es menor que $n\sum_{j=1}^{n} p_j$. Por tanto, la complejidad de los pasos segundo al cuarto esta acotada por $0(n^2 \sum_{j=1}^{n} p_j)$. Esta cantidad es mayor que la complejidad del paso 1, por lo que dicho orden será la complejidad del algoritmo.

Es importante notar que el valor $\sum_j T_j(Semilla)$ se calcula con sólo realizar el paso 1. Justo después del paso 1 puede predecirse el comportamiento de la heurística con respecto a la velocidad de convergencia del algoritmo, sin necesidad de ejecutar los siguientes pasos.

Señalemos que la heurística propuesta es aplicable también al problema particular de la tardanza total con restricciones de precedencia y tiempos de proceso unitarios, (problema $1|prec,p_j=1|\sum Tj$), en cuyo caso la complejidad sería de $O(n^3)$.

Para finalizar el presente capítulo recordemos que se han planteado varios problemas de planificación sobre una máquina. La atención se ha centrado en ciertos problemas NP-duros. Se han propuesto diversos algoritmos heurísticos para su resolución. En relación con ellos, en el Apéndice A se realiza un estudio computacional dirigido a comparar el comportamiento de la heurística propuesta para $1||\sum Tj$ con las presentadas por otros autores.

Por último, tal y como se comentó al principio de la memoria, digamos que un grupo importante de problemas de planificación se plantean bajo la consideración de poder disponer de varias máquinas o procesadores para la realización de los trabajos. El próximo capítulo analiza y estudia estos problemas.

Capítulo III

Capítulo III: Problemas de Planificación sobre varias máquinas

1. Introducción

En este capítulo se analizan los problemas de planificación unicriterio sobre varias máquinas. El capítulo comienza sumarizando la clasificación computacional de estos problemas. Se realiza dicha clasificación distinguiendo si las máquinas son o no especializadas en la ejecución de las diversas actividades. Recordemos que si las máquinas son *no especializadas* entonces cualquier máquina es capaz de ejecutar cualquier tarea. Mientras que si las máquinas son *especializadas* ciertos trabajos u operaciones de los mismos deben ser ejecutados por ciertas máquinas y no por otras.

Al clasificar computacionalmente los problemas sobre máquinas no especializadas se diferencia en función de que los trabajos sean interrumpibles o no. Sin embargo, la clasificación de problemas sobre máquinas especializadas se hace atendiendo a las condiciones a respetar por las *"trayectorias"*. Entendiéndose por *"trayectoria"* la sucesión de máquinas que deben seguir las operaciones correspondientes a cada trabajo.

En el tercer epígrafe del capítulo se proponen nuevos algoritmos que ofrecen alternativas para resolver algunos de estos problemas. Un estudio computacional de algunos de estos algoritmos puede verse en el Apéndice B.

2. Clasificación computacional

2.1. Máquinas no especializadas

Supongamos que se desean procesar n trabajos y para ello disponemos de m máquinas donde cualquier máquina es capaz de realizar cualquier trabajo, aunque quizás no con la misma velocidad; por lo que distinguiremos entre máquinas idénticas *($\alpha = P$)*, uniformes *($\alpha = Q$)*, y no relacionadas *($\alpha = R$)*.

Mientras los problemas de planificación en máquinas no especializadas *sin interrupciones* suelen ser difíciles: $P2||C_{max}$, $P2||\sum \omega_j C_j$ son binarios NP-duros (Bruno y otros (1974), Lenstra y otros (1977)). Los problemas de planificación en máquinas no especializadas *con interrupciones* acostumbran a ser más sencillos:

$P|pmtn|C_{max}$ tiene complejidad $O(n)$ (McNaughton (1959)). Sin embargo, también hay problemas complicados en los que los trabajos pueden interrumpirse: $P2|pmtn|\sum \omega_j C_j$ es NP-duro (Lawler y otros (1982)). $P|pmtn|\sum T_j$ es NP-duro porque $1|pmtn|\sum T_j$ es NP-duro en sentido ordinario y, además, $1|pmtn|\sum \omega_j T_j$ lo es en el sentido fuerte. La complejidad de $P|pmtn|\sum T_j$ respecto a una codificación unaria es una cuestión abierta. (Lawler y otros (1993)).

Es natural, por tanto, estudiar por un lado los problemas de planificación sin interrupciones y por otro los problemas con interrupciones:

2.1.1. Planificación sin interrupciones

Comenzaremos primero comentando los problemas de planificación con $p_j=1 \ \forall j$ y luego los problemas con p_j generales.

2.1.1.1. Tiempos de proceso unitarios

$Q|p_j=1|\sum f_j$; y $Q|p_j=1|f_{max}$. son problemas en los que cualquier máquina puede realizar cualquier tarea, pero las velocidades de proceso son constantes características de cada máquina.

Tanto si la función objetivo es el coste total como si es el coste máximo, estos problemas se plantean como problemas de flujo en redes con la siguiente interpretación (Lawler y otros (1982), Blazewicz y otros (1993)):

Se dispone de n fuentes j $(j = 1, ..., n)$ y mn sumideros (i,k) $(i=1, ..., m; k=1, ..., n)$. El costo del arco $(j, (i,k))$ es $c_{ijk} = f_j(k/q_i)$, siendo q_i la velocidad de la máquina M_i. El flujo de dicho arco es x_{ijk} donde

$$x_{ijk} = \begin{cases} 1, & si\ J_j\ se\ ejecuta\ en\ M_i\ en\ el\ puesto\ k \\ 0, & en\ otro\ caso \end{cases}$$

Nótese que, si M_i es q_i veces más rápida que la máquina M que se toma de referencia y para la que todos los trabajos tienen tiempos de proceso unitario, entonces k/q_i es el instante de completación del trabajo J_j cuando éste es el k-ésimo trabajo que se ejecuta en la máquina M_i, y de ahí la expresión de los c_{ijk}. El problema consiste en:

$$min \sum_i \sum_j \sum_k c_{ijk} x_{ijk}$$

si la función objetivo es una suma; ó en:

$$\min \max_{i,j,k} \left\{ c_{ijk} x_{ijk} \right\}$$

si la función objetivo es un máximo. En cualquier caso las restricciones son:

$$\sum_i \sum_k x_{ijk} = 1, \quad \forall j$$

$$\sum_j x_{ijk} \leq 1, \quad \forall i, \forall k$$

$$x_{ijk} \geq 0, \quad \forall i, \forall j, \forall k$$

El tiempo que se requiere para preparar los datos para este problema de transporte es $O(mn^2)$. Posteriormente el problema puede resolverse en tiempo $O(n^3)$. Si $m \leq n$, puede suponerse que $O(n^3)$ es la complejidad de estos problemas. (Lawler y otros (1982), Lawler y otros (1993)).

Consideremos ahora máquinas idénticas. Así, el problema $P|prec,p_j=1|C_{max}$ es NP-duro (Ullman (1975)). Lenstra y Rinnooy Kan (1978) muestran que incluso el problema de decidir si existe una planificación factible de longitud a lo sumo *3* es NP-completo. Es una cuestión abierta para cualquier valor constante del número de máquinas $m \geq 3$, es decir, $Pc|prec,p_j=1|C_{max}$ con $c \geq 3$ son problemas abiertos. El problema está bien resuelto, sin embargo, si la relación de precedencia es de tipo árbol o si $m = 2$. (Lawler y otros (1993)).

$P|tree=intree,p_j=1|C_{max}$ puede resolverse en tiempo $O(n)$ por el algoritmo de Hu (1961) (ver también Hsu (1966), Baker (1974), Sethi (1976)).

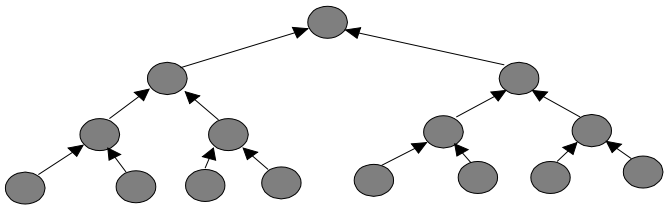


Figura 1. Un árbol de ensamble

En este algoritmo se define el nivel de un trabajo como el número de trabajos del único camino existente desde ese trabajo a la raíz del árbol de precedencias, que es un árbol de ensamble como el de la figura 1. El algoritmo de Hu se desarrolla en dos etapas: etapa de etiquetado y etapa de planificación o scheduling. En la etapa de etiquetado se etiqueta cada trabajo con el nivel que ocupa en el árbol. Las etiquetas establecen una relación de prioridad entre los trabajos, teniendo más prioridad los trabajos con mayor etiqueta. Esto permite establecer una lista de los trabajos en orden creciente de sus prioridades. La etapa de planificación consiste en asignar el primer trabajo de la lista a la primera máquina que quede libre hasta que la lista quede vacía.

El algoritmo de Hu puede adaptarse para resolver el problema $P|intree,p_j=1|L_{max}$, sin embargo, $P|outtree,p_j=1|L_{max}$ es NP-duro (Brucker y otros (1977)).

Existen otros algoritmos y pruebas de NP-dureza para problemas $P|prec,p_j=1|C_{max}$ en los que la estructura de precedencias no consiste en "intrees" ni en "outtrees" sino, por ejemplo, en combinaciones de ambos tipos de precedencias ("opposing forests"), grafos por niveles, y otros tipos de grafos. (Dolev (1981), Garey y otros (1981), Warmuth (1980)).

$P2|prec,p_j=1|C_{max}$ puede resolverse eficientemente por varios algoritmos polinomiales como, por ejemplo, los de Fujii y otros (1969, 1971) de $O(n^3)$, Coffman y Graham (1972) de $O(n^2)$, Gabow (1980) de $O(n^2)$.

Finalmente, podemos mencionar que tanto $P2|prec,p_j\in\{1,2\}|C_{max}$ como $P2|prec,p_j\in\{1,2\}|\sum C_j$ son NP-duros (Ullman (1975), Lenstra y Rinnooy Kan (1978)). Para el problema $P2|prec,p_j\in\{1,k\}|C_{max}$ existe un algoritmo aproximado debido a Goyal (1977) que es una versión generalizada del algoritmo de Coffman y Graham (1972).

2.1.1.2. Tiempos de proceso generales

Máquinas no especializadas idénticas

$1||\sum \omega_j C_j$ está bien resuelto en $O(n \log n)$ por la secuencia *SPT*.

$P||\sum C_j$ también se resuelve en $O(n \log n)$ por la generalización de dicha secuencia (Conway y otros (1967)) para lo que se supone que n es múltiplo de m. Esto no supone pérdida de generalidad ya que si no lo es pueden añadirse trabajos con $p_j = 0$.

$P||\sum \omega_j C_j$: Aunque la secuencia *WSPT*

$$\frac{p_1}{\omega_1} \leq \frac{p_2}{\omega_2} \leq \ldots \leq \frac{p_n}{\omega_n}$$

es óptima para $1||\sum \omega_j C_j$, no lo es para $P||\sum \omega_j C_j$; pero, según nos indican Eastman, Even e Isaacs (1964) nos permite obtener una cota inferior a la solución óptima, esto es:

$$\sum_{j=1}^{n} \omega_{[j]} C^*_{[j]} \geq \frac{m+n}{m(n+1)} \sum_{j=1}^{n} \sum_{k=1}^{j} \omega_{[j]} p_{[k]}$$

donde $J_{[1]}, J_{[2]}, \ldots, J_{[n]}$ es la secuencia *WSPT* y $\sum_{j=1}^{n} \omega_{[j]} C^*_{[j]}$ es el valor óptimo de la función objetivo. Utilizando dicha cota Elmaghraby y Park (1974) y Barnes y Brennan (1977) han desarrollado algoritmos de ramificación y acotación. Por su parte, Sahni (1976) ha construido algoritmos aproximados A_k de complejidad $O(n(n^2k)^{m-1})$ siendo el radio de aproximación:

$$\frac{\sum \omega_j C_j(A_k)}{\sum \omega_j C^*_j} \leq 1 + \frac{1}{k}$$

e incluso si $m=2$, A_k puede ejecutarse en tiempo $O(n^2k)$.

$P2||\sum \omega_j C_j$ es binario NP-duro (Lenstra y otros (1977)).

$P||C_{max}$ es el problema de asignar n tareas a m procesadores de modo que cada trabajo se asigna a exactamente un procesador y, en un instante de tiempo dado, cada procesador sólo puede ejecutar un único trabajo. El objetivo es minimizar el máximo tiempo de completación (*"makespan"*). Este problema es NP-duro en sentido fuerte (Garey y Johnson (1979)). Existen algoritmos aproximados para resolverlo como los de Lawler y otros (1993), y Cheng y Sin (1990). Se han desarrollado relativamente pocos algoritmos exactos para este problema. Para pequeños valores de m y U, siendo U una cota superior de la solución del valor C^*_{max} óptimo, puede resolverse óptimamente por programación dinámica como se describe en Blazewicz (1987).

El problema es un problema relativamente próximo al problema *"bin packing" ("bin packing problem" (BPP)).* Dados n items de tamaños $p_1,\ldots,p_n$ y un número ilimitado de cubos ("bins") de capacidad C, el problema *BPP* consiste en asignar cada item a un bin sin exceder la capacidad de manera que el número de cubos usados se minimice. El problema *BPP* es NP-duro en sentido fuerte. Esta analogía les ha valido a Coffman, Garey y Johnson (1978) para obtener un algoritmo aproximado *Multifit* para $P||C_{max}$, que encuentra mediante búsqueda

binaria el menor valor de C tal que la solución encontrada para *BPP* por el algoritmo aproximado *FFD* (*"First-Fit Decreasing"*) tiene un valor que no es mayor que m. Estas ideas pueden usarse para resolver $P||C_{max}$ exactamente. Dada una cota inferior L y una cota superior U de C^*_{max}, se determina, mediante búsqueda binaria, el menor valor de C $(L \leq C \leq U)$ tal que la solución exacta del problema *BPP* no use más de m cubos.

Martello y Toth (1990) proponen un algoritmo de ramificación y acotación para el problema *BPP*. Por su parte, Dell'Amico y Martello (1994), utilizan estos resultados para construir un algoritmo de ramificación y acotación para el problema $P||C_{max}$.

Máquinas no especializadas uniformes

$Q||\Sigma C_j$: El algoritmo de orden $O(n \log_2 n)$ para $P||\Sigma C_j$ puede generalizarse para resolver $Q||\Sigma C_j$ (Conway y otros (1967)). También existe un procedimiento de $O(n \log_2 n)$ para resolver este problema debido a Horowitz y Sahni (1976).

Máquinas no especializadas y no relacionadas

$R||\Sigma C_j$: Este problema esta bien resuelto ya que se formula como un problema de transporte de tamaño m x n y complejidad $O(n^3)$ (Horn (1973), Bruno y otros (1974).

La formulación del problema requiere las variables de decisión:

$$x_{ijk} = \begin{cases} 1, & si\ J_j\ es\ el\ k-esimo\ ultimo\ trabajo\ procesdo\ en\ M_i \\ 0, & otro\ caso \end{cases}$$

$$\forall i = 1, ..., m; \quad \forall j = 1, ..., n; \quad \forall k = 1, ..., n$$

y se trata de resolver el problema:

$$min \sum_{i=1}^{m} \sum_{j=1}^{n} \sum_{k=1}^{n} kp_{ij} x_{ijk}$$

s.a.:

$$\sum_{i=1}^{m} \sum_{k=1}^{n} x_{ijk} = 1, \quad \forall j$$

$$\sum_{j=1}^{n} x_{ijk} \leq 1, \quad \forall i, \forall k$$

$$x_{ijk} \in \{0,1\} \quad \forall i, \forall j, \forall k$$

Las restricciones aseguran que cada trabajo se planifica exactamente una vez y que cada posición en la máquina se ocupa por a lo sumo un trabajo. Se trata de un problema de emparejamiento bipartito ponderado, por lo que las restricciones enteras pueden sustituirse por restricciones de no negatividad sin alterar el conjunto factible, es decir, por:

$$x_{ijk} \geq 0, \quad \forall i, \forall j, \forall k$$

2.1.2. Planificación con interrupciones

2.1.2.1. Minimizando el coste total

Máquinas no especializadas idénticas

$P|pmtn|\sum C_j$: El teorema de McNaughton (1959) garantiza que no hay planificación para $P|pmtn|\sum \omega_j C_j$ con un número finito de interrupciones que mejore la planificación sin interrupciones óptima. Por tanto, los procedimientos de optimización para $P||\sum C_j$ son aplicables a $P|pmtn|\sum C_j$ que resulta ser un problema de complejidad $O(n \log_2 n)$. El teorema de Mc Naughton y el hecho de que $P2||\sum \omega_j C_j$ sea NP-duro nos prueban que $P2|pmtn|\sum \omega_j C_j$ también es NP-duro.

$P|pmtn|\sum U_j$ es binario NP-duro (Lawler (1981)), con lo que $P|pmtn|\sum \omega_j U_j$ también lo es.

Máquinas no especializadas uniformes

$Q|pmtn|\sum C_j$: Aunque no es aplicable el citado teorema de McNaughton al problema sobre máquinas uniformes $Q|pmtn|\sum C_j$, este problema es resoluble también por un algoritmo polinomial.

Incluso se puede afirmar que existe una planificación con interrupciones óptima en la que $C_j \preceq C_k$ si $p_j < p_k$ (Lawer y Labetoulle (1978)). Tal planificación se puede deducir de la siguiente manera: se ordenan primero los trabajos en el

orden *SPT*. A partir de este orden se obtiene una planificación óptima, interrumpiendo los sucesivos trabajos en el tiempo permisible, sobre las *m* máquinas para minimizar su tiempo de completación. González (1977) propone un procedimiento de complejidad *O(n log n + m n)*, obteniendo una planificación óptima con a lo sumo *(m-1)(n - (1/2)m)* interrupciones.

Además, el problema $Q|pmtn,d_j=d|\Sigma C_j$ puede resolverse por una extensión del procedimiento anterior (González (1977)).

Puesto que, como sabemos, $1|pmtn|\Sigma\omega_j U_j$ es binario NP-duro, resulta que $Q|pmtn|\Sigma\omega_j U_j$ es también NP-duro.

$Q2|pmtn|\Sigma\omega_j U_j$ puede resolverse por un algoritmo pseudopolinomial de complejidad $O(n^2(\Sigma\omega_j)^2)$ (Lawler (1981).

$Qm|pmtn|\Sigma\omega_j U_j$ con $m \geq 3$, es pseudopolinomialmente resoluble en tiempo $O(n^{3m-5}(\Sigma\omega_j)^2)$. (Lawler (1981)).

$Qm|pmtn|\Sigma U_j$ es resoluble en tiempo polinomial como consecuencia inmediata de los resultados anteriores.

Máquinas no especializadas y no relacionadas

$R|pmtn|\Sigma C_j$: Muy poco se conoce a cerca del problema $R|pmtn|\Sigma C_j$. Esta es una de las cuestiones más controvertidas en el área de la planificación con interrupciones. De hecho es un problema todavía abierto. (Blazewicz y otros (1993)).

2.1.2.2. Minimizando el costo máximo

Máquinas no especializadas idénticas

$P|pmtn|C_{max}$ es polinomialmente resoluble porque una cota inferior a la solución óptima es:

$$max\left\{max_j\{p_j\}, \frac{1}{m}\sum_j p_j\right\}$$

y es posible encontrar una planificación que alcance esa cota, es decir, una planificación óptima, en tiempo *O(n)* (McNaughton (1959)). El procedimiento consiste en ir llenando las máquinas sucesivamente planificando en ellas los trabajos en cualquier orden e interrumpiendo aquel trabajo que rebase la cota anterior. De esta manera el número máximo de interrupciones que puede tener la planificación es *m-1*. Es posible diseñar una clase de problemas para los que el número de interrupciones sea minimal, pero, en general, minimizar el número de interrupciones es un problema NP-duro.

Mientras que $P|pmtn,prec,p_j=1|C_{max}$ es NP-duro (Ullman (1976)), los problemas $P|pmtn,intree|C_{max}$ y $P2|pmtn,prec|C_{max}$ pueden resolverse en tiempo polinomial por el algoritmo de Muntz y Coffman (Muntz y Coffman (1969, 1970); Baker (1974)).

El algoritmo de Muntz y Coffman para resolver $P2|pmtn,prec|C_{max}$ requiere que los trabajos se particionen en varios subconjuntos S_k, de manera que todos los trabajos de un subconjunto particular sean mutuamente independientes, es decir, si J_i precede a J_j, entonces *i, j* serán miembros de distintos subconjuntos. Una vez asignados los trabajos a los subconjuntos las relaciones de precedencia existirán sólo entre pares de subconjuntos con lo que en cada subconjunto los trabajos son independientes y puede aplicarse el algoritmo de McNaughton (1959). Posteriormente se secuencian los conjuntos de acuerdo con las relaciones de precedencia. Un método para determinar los subconjuntos se describe en dos pasos: el primero de etiquetaje de los trabajos y el segundo de asignación de los trabajos a los subconjuntos. (Muntz y Coffman (1969, 1970); Baker (1974)).

El algoritmo de Muntz y Coffman para resolver $P|pmtn,intree|C_{max}$ se ayuda del concepto de "processor sharing" (procesador compartido) que permite que a un trabajo dado se le asigne una capacidad de recurso $\rho \leq 1$. De esta manera se modeliza el hecho de que se permita que el trabajo se procese por sólo *"parte de una máquina"*. Si se le asigna al trabajo J_j la capacidad de recurso ρ $(0 < \rho \leq 1)$, el tiempo de ejecución del trabajo en la máquina sería p_j/ρ. Si $\rho=1$ la máquina se dedica solamente a ejecutar el trabajo. Es decir, ρ indica el radio en el cual un trabajo es procesado por una máquina en particular.

Permitiendo "processor sharing" temporalmente, el algoritmo de Muntz y Coffman para $P|pmtn,intree|C_{max}$ queda como un algoritmo en 2 etapas. La primera etapa es una etapa de etiquetado que generaliza la del algoritmo de Hu (ver Baker (1974)) ya que a cada trabajo J_j se le etiqueta con la suma de los tiempos de procesamiento de los trabajos del único camino desde J_j a la raíz. Esta etapa permite ordenar los trabajos en orden creciente de etiquetas. Considerando esa ordenación y actualizando convenientemente los radios de procesamiento se determina una planificación óptima en la segunda etapa o etapa de planificación.

Lam y Sethi (1977) analizan la efectividad del algoritmo de Muntz y Coffman *(MC)* para $P|pmtn,prec|C_{max}$ y prueban que

$$\frac{C_{max}(MC)}{C^*_{max}} \leq 2 - \frac{2}{m}$$

para $m \geq 2$.

González y Johnson (1980) han desarrollado un algoritmo totalmente diferente al de Muntz y Coffman para el problema $P|pmtn,tree|C_{max}$. El algoritmo de González y Johnson tiene complejidad $O(n \log_2 m)$ y obtiene planificaciones óptimas con a lo sumo $n-2$ interrupciones. El algoritmo comienza por las raíces en lugar de por las hojas del árbol y determina las prioridades entre trabajos considerando el tiempo total de procesamiento en subárboles en vez de atender a los caminos críticos.

$P|pmtn|L_{max}$ y $P|pmtn,r_j|C_{max}$ se resuelven en tiempo $O(n^2)$ por un algoritmo debido a Horn (1974). La complejidad de estos problemas puede reducirse a $O(mn)$ (González y Johnson (1980)). Aún más, la existencia de una planificación factible con interrupciones y sujeta a las fechas de disponibilidad y fechas límite puede comprobarse en tiempo $O(n^3)$ utilizando un modelo de flujo en redes (Horn (1974)).

En el caso de restricciones de precedencia, las versiones con interrupciones de los algoritmos polinomiales de Brucker, Garey y Johnson (1977) para resolver los problemas $P|intree,p_j=1|L_{max}$, $P2|prec,p_j=1|L_{max}$, $P2|prec,r_j,p_j=1|L_{max}$ permiten resolver polinomialmente los correspondientes problemas con interrupciones, es decir, $P|pmtn,intree,p_j=1|L_{max}$, $P2|pmtn,prec,p_j=1|L_{max}$, $P2|pmtn,prec,r_j,p_j=1|L_{max}$ son polinomialmente resolubles (Lawler (1980)).

Máquinas no especializadas uniformes

$Q|pmtn,prec|C_{max}$: Horvath, Lam y Sethi (1977) adaptan los algoritmos de Muntz y Coffman para resolver $Q|pmtn|C_{max}$ y $Q2|pmtn,prec|C_{max}$ en tiempo $O(mn^2)$. Las planificaciones obtenidas tienen a lo sumo $(m-1)n^2$ interrupciones.

González y Sahni (1978b) resuelven $Q|pmtn|C_{max}$ por un algoritmo más eficiente de complejidad $O(n)$. La planificación óptima encontrada no tiene más de $2(m-1)$ interrupciones y el valor óptimo de C_{max} es

$$C^*_{max} = max\left\{ max_{1 \leq k \leq m-1}\left\{ \frac{\sum_{j=1}^{k} p_j}{\sum_{i=1}^{k} q_i} \right\}, \frac{\sum_{j=1}^{n} p_j}{\sum_{i=1}^{m} q_i} \right\}$$

donde los trabajos y máquinas están numerados de manera que

$$p_1 \geq p_2 \geq \ldots \geq p_n \; y \;\; q_1 \geq q_2 \geq \ldots \geq q_m$$

siendo q_i la velocidad de la máquina M_i y $p_{ij} = p_j / q_i$ el tiempo de proceso del trabajo j en la máquina i.

El algoritmo de González y Johnson para $P|pmtn,tree|C_{max}$ puede adaptarse al caso $Q2|pmtn,tree|C_{max}$.

Jaffe (1980) ha estudiado el problema $Q|pmtn,prec|C_{max}$ y la efectividad de las planificaciones, sin tiempo ocioso no forzado, en las que en cualquier instante los trabajos pasen a ser procesados por las máquinas más rápidas que queden disponibles. Ha observado que

$$\frac{C_{max}}{C^*_{max}} \leq \sqrt{m} + \frac{1}{2}$$

siendo C_{max} el tiempo máximo de completación de los trabajos en dichas planificaciones.

$Q|pmtn,r_j|C_{max}$ y, por simetría, $Q|pmtn|L_{max}$, se resuelven en tiempo $O(n \log_2 n + mn)$ obteniendo planificaciones óptimas cuyo número de interrupciones es $O(mn)$ (Sahni y Cho (1979), Labetoulle y otros (1979).

$Q2|pmtn,r_j|L_{max}$ se resuelve en tiempo polinomial (Labetoulle y otros (1979)).

$Q|pmtn,r_j|L_{max}$ fue resuelto en tiempo polinomial por Martel (1981). El método de resolución es, en realidad, un caso especial de un algoritmo más general para computar flujos en redes poli-matroidales (Lawler y Martel (1980)).

Máquinas no especializadas y no relacionadas

Consideremos ahora aquellos problemas con máquinas no especializadas y no relacionadas donde la función objetivo es un costo máximo. No existen algoritmos polinomiales para estos problemas en ausencia de interrupciones (Blazewicz y otros (1993)).

$R|pmtn|C_{max}$ puede resolverse por un método de dos fases. La primera fase consiste en resolver un problema de programación lineal formulado independientemente por Blazewicz y otros (1976, 1977), y por Lawler y

Labetoulle (1978). La segunda fase usa la solución de este problema de programación lineal y produce una planificación óptima con interrupciones.

Si $x_{ij} \in \{0, 1\}$ denota la parte del trabajo J_j que se procesa en la máquina M_j. La formulación del problema de programación lineal es como sigue:

$$min\ C_{max}$$

s.a.:

$$C_{max} - \sum_{j=1}^{n} p_{ij} x_{ij} \geq 0, \quad i = 1,\ldots,m$$

$$C_{max} - \sum_{i=1}^{m} p_{ij} x_{ij} \geq 0, \quad j = 1,\ldots,n$$

$$\sum_{i=1}^{m} x_{ij} = 1, \quad j = 1,\ldots,n$$

Una vez resuelto este problema, se tiene $C_{max} = C^*_{max}$ y los valores óptimos x^*_{ij}. Sin embargo, no conocemos como planificar las partes de los trabajos, es decir, como asignarlas a las máquinas a lo largo del tiempo. Una planificación puede construirse de la siguiente manera. Sea $T = [t^*_{ij}]$ una matriz de $m \times n$ definida por $t^*_{ij} = p_{ij}\, x^*_{ij}$, $i=1,...,m$, $j=1,...,n$. Los elementos de T reflejan valores óptimos de tiempos de proceso de trabajos particulares sobre las máquinas. La j-ésima columna de T corresponde al trabajo J_j que se llamará *crítico* si $\sum_{i=1}^{m} t^*_{ij} = C^*_{max}$. Sea Y la matriz $m \times m$ diagonal cuyo elemento y_{kk} es el tiempo total ocioso de la máquina M_k, es decir, $y_{kk} = C^*_{max} - \sum_{j=1}^{n} t^*_{kj}$. Las columnas de Y corresponden con trabajos artificiales. Sea $V = [T, Y]$ una matriz $m \times (n+m)$. Puede definirse un conjunto U que contenga m elementos positivos de la matriz V teniendo exactamente un elemento de cada columna crítica y a lo sumo un elemento de las otras columnas, y con exactamente un elemento de cada fila. U corresponde a un conjunto de trabajos que pueden procesarse en paralelo en la planificación óptima. Por tanto, puede usarse para construir una planificación parcial de longitud $\delta > 0$. Una planificación óptima se establece entonces como unión de planificaciones parciales. El siguiente algoritmo construye una planificación óptima correspondiente a la solución del problema *LP* para $R|pmtn|C_{max}$.

Algoritmo (Lawler y Labetoulle (1978))

begin
  $C := C^*_{max}$;

```
while C>0 do
begin
   construir el conjunto U; /* por tanto se elige un subconjunto de trabajos
    para ser procesados en una planificación parcial */
   v_min := min_{v_ij ∈ U} {v_ij} ;
   v_max := max_{j ∈ {j'/v_ij ∉ U para i=1,...,m}} { Σ_i v_ij } ;
   if C - v_min ≥ v_max then
      δ := v_min
   else
      δ := C - v_max ; /* la longitud de la planificación parcial es δ */
   C := C - δ;
   for cada v_ij ∈ U do
      v_ij := v_ij - δ;
      /* la matriz V cambia pero por la definición de δ sus elementos
      no pueden ser negativos */
end;
end;
```

Para un m fijo es posible la resolución del problema en tiempo lineal: $R2|pmtn|C_{max}$ puede resolverse en tiempo $O(n)$. (González y otros (1990).

$R|pmtn,r_j|L_{max}$ admite una formulación similar como problema de programación lineal. (Lawler y Labetoulle (1978)).

2.2. Máquinas especializadas

Estudiaremos ahora la complejidad de los problemas de planificación en los que cada trabajo consiste en un conjunto de tareas y para realizar la planificación se dispone de varias máquinas especializadas.

Recordemos que en los problemas open shop *(O)* el orden en el que los trabajos pasan por las máquinas es irrelevante pero todo trabajo tiene asignado un tiempo de proceso en todas las máquinas. Sin embargo, en los problemas flow shop *(F)*, todos los trabajos deben pasar por todas las máquinas y en el mismo orden aunque el tiempo de proceso de ciertos trabajos en ciertas máquinas puede ser cero. En los problemas job shop *(J)* el subconjunto de máquinas que procesa un trabajo y el orden de proceso pueden diferir de unos trabajos a otros, pero serán siempre conocidos a priori.

2.2.1. Planificación Open shop

Comentaremos primero los problemas open shop en los que no se permiten interrupciones y luego aquellos en los que es posible interrumpir la ejecución de un trabajo para luego continuar su procesamiento a partir del punto donde se dejo.

$O2||C_{max}$ se resuelve en tiempo $O(n)$ por el algoritmo de González y Sahni (1976). Dicho algoritmo puede describirse de la forma siguiente:

Denotemos por $a_j = p_{1j}$ y por $b_j = p_{2j}$ los tiempos de proceso en la primera y segunda máquinas. Sean los dos subconjuntos de trabajos $A=\{J_j / a_j \geq b_j\}$ y $B=\{J_j / a_j < b_j\}$. Elijamos dos trabajos distintos J_r y J_l tales que $a_r \geq max_{J_j \in A}\{b_j\}$ y $b_l \geq max_{J_j \in B}\{a_j\}$. Construyamos los conjuntos $A'=A - \{J_r, J_l\}$ y $B'=B - \{J_r, J_l\}$. Una planificación factible para $A' \cup \{J_r\}$ y $B' \cup \{J_l\}$ sería la de la figura 2 (a), donde los trabajos de A' y B' se ordenan arbitrariamente. Siendo $P_1=\sum_j a_j$ y $P_2=\sum_j b_j$ los tiempos de completación de las máquinas, podemos suponer que $P_1-a_l \geq P_2-b_r$ (el caso contrario sería simétrico) y entonces podemos arreglar la planificación anterior para obtener otra mejorada (figura 2 (b)). Ahora no hay tiempo ocioso en ninguna máquina. En la figura 2 (c) obtenemos una mejora con $C_{max}=max\{P_1 , P_2\}$ para el caso $a_r \leq P_2 - b_r$; y en la figura 2 (d) la mejoría para el caso contrario con $C_{max} = max \{P_1, a_r + b_r\}$.

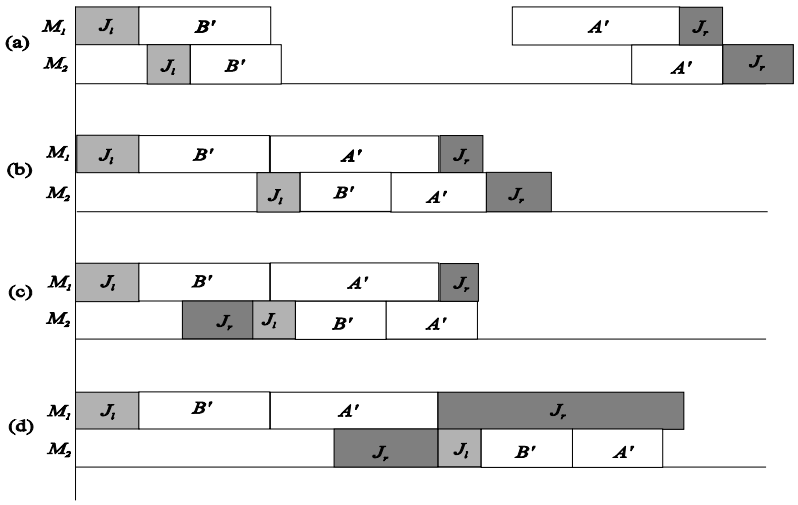


Figura 2. Ilustración del algoritmo para $O2||C_{max}$.

Como las dos tareas de un mismo trabajo no se pueden ejecutar simultáneamente, para cualquier planificación factible se tiene la cota

$$C_{max} \geq max\ \{P_1, P_2, max_j\ \{a_j + b_j\}\}$$

y como con el algoritmo anterior se alcanza una cota inferior, dicho algoritmo es óptimo.

Ejemplo 1:

Consideremos un ejemplo numérico. Los ocho trabajos a realizar y sus tiempos de proceso son los que se indican.

Trabajo	J_1	J_2	J_3	J_4	J_5	J_6	J_7	J_8
$a_j=p_{1j}$	6	7	5	3	14	8	2	9
$b_j=p_{2j}$	4	5	3	7	6	10	4	8

Los conjuntos citados son $A = \{J_1, J_2, J_3, J_5, J_8\}$ y $B = \{J_4, J_6, J_7\}$. Puede tomarse $a_r = 8$ con $r = 6$, y $b_l = 8$ con $l = 8$; de donde $A' = \{J_1, J_2, J_3, J_5\}$ y $B' = \{J_4, J_7\}$. Como $P_1 - a_l = 45 \geq P_2 - b_r = 40$ partimos de una planificación factible similar a la (b) de la figura 2 (véase (b) en la figura 3). Como $a_r = 8 \leq 40 = P_2 - b_r$ la solución óptima es de la forma (c) (véase la figura 3 (c)):

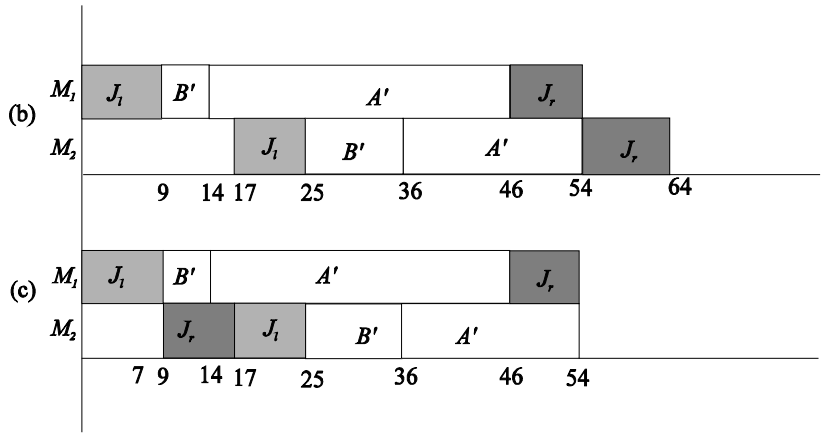


Figura 3. Ilustración del ejemplo 1.

Ishii y Nishida (1986) estudian una variante del problema $O2||C_{max}$ en la que la velocidad de las máquinas es controlable y proponen un algoritmo de complejidad $O(n \log n)$ que encuentra la velocidad óptima de cada máquina y la planificación óptima.

$O3||C_{max}$ es binario NP-duro (González y Sahni (1976)).

En general, para $m > 2$, los problemas open shop de minimizar el tiempo total de completación de los trabajos ("makespan") sin interrupciones con m máquinas son NP-completos (González y Sahni (1976, 1978a)).

$O2|r_j|C_{max}$; $O2|tree|C_{max}$; $O||C_{max}$ son unarios NP-duros (Lawler y otros (1981)).

Comentemos ahora los problemas open shop con interrupciones.

$O2|pmtn|C_{max}$ se resuelve en $O(n)$ por el algoritmo comentado anteriormente para $O2||C_{max}$ ya que la posibilidad de interrupciones no proporciona ventaja alguna.

$O|pmtn|C_{max}$ se resuelve en tiempo polinomial (González y Sahni (1976), Lawler y Labetoulle (1978), González (1979)).

Si consideramos la existencia de diferentes fechas de disponibilidad los problemas correspondientes también se resuelven polinomialmente.

$O2|pmtn,r_j|C_{max}$ es de complejidad $O(n)$ (Lawler y otros (1981)).

$O|pmtn,r_j|L_{max}$ también está resuelto en tiempo polinomial (Cho y Sahni (1978)).

2.2.2. Planificación flow shop

$F2||C_{max}$ es polinomial (Johnson (1954)). Dicho autor demostró que existe una planificación óptima en la que J_j precede a J_k si $min\ \{p_{1j}, p_{2k}\} \leq min\ \{p_{2j}, p_{1k}\}$ con lo que el problema puede resolverse en $O(n \log n)$ sin más que ordenar primero los trabajos con $p_{1j} \leq p_{2j}$ en orden creciente de p_{1j} y los restantes trabajos en orden decreciente de p_{2j} .

$F3||C_{max}$; $F2|r_j|C_{max}$; $F2|tree|C_{max}$ son unarios NP-duros (Garey y otros (1976), Lenstra y otros (1977)).

Para el problema $Fm||C_{max}$, Szwarc (1977) ha estudiado una generalización de la regla de Johnson en el caso $m = 3$ y la optimalidad de dicha generalización. Condiciones de dominancia para este problema basadas en la regla de Johnson se han propuesto por Gupta y Reddi (1978), Szwarc (1978).

$F2|tree|C_{max}$ es unario NP-duro, sin embargo, si relajamos el concepto de trabajos precedentes diciendo que $J_j \rightarrow' J_k \Leftrightarrow O_{ij}$ debe preceder a O_{ik} para $i=1,2$; entonces $F2|tree'|C_{max}$ tiene complejidad $O(n \log n)$ (Sidney (1979)). Sin embargo, $F2|prec'|C_{max}$ es unario NP-duro (Monma (1980)).

Szwarc (1981) ofrece una generalización natural del algoritmo de Johnson para dos máquinas al caso de m máquinas contemplando la existencia de relaciones de precedencia entre trabajos, y proporciona las condiciones suficientes para la optimalidad de la planificación resultante.

Consideremos ahora la posibilidad de interrumpir trabajos:

$F2|pmtn|C_{max}$ se resuelve polinomialmente por el algoritmo de Johnson para $F2||C_{max}$ porque las interrupciones de M_1 y M_2 pueden suprimirse sin incrementar C_{max}.

$F3|pmtn|C_{max}$ y $F2|pmtn,r_j|C_{max}$ son unarios NP-duros (González y Sahni (1978a), Cho y Sahni (1978)).

En general, los problemas de minimizar el makespan, o tiempo de completación de los trabajos, para un número m de máquinas superior a dos son NP-completos (González y Sahni (1978a)). En este sentido, Lageweg y otros (1978) determinan un esquema de clasificación de las cotas inferiores que genera nuevas cotas más próximas al óptimo.

Por su parte, Gupta (1986), considera el problema flow shop con m máquinas estático $(r_j = r \quad \forall j)$ y minimiza el makespan permitiendo la existencia de tiempos setup de preparación de las máquinas. Demuestra que este problema es NP-completo, y propone algoritmos de resolución estudiando su efectividad.

2.2.3. Planificación job shop

Los problemas job shop de minimizar el makespan con dos máquinas y con interrupciones son NP-completos (González y Sahni (1978a)). Sin embargo, en ausencia de interrupciones algunos problemas son polinomiales.

$J2|m_j \leq 2|C_{max}$ se resuelve en $O(n \log n)$ por una extensión del algoritmo de Johnson para $F2||C_{max}$ (Jackson (1956)). Para ello sea J_i el conjunto de trabajos con todas sus operaciones en la máquina i $(i = 1, 2)$ y J_{hi} el conjunto de trabajos que van desde la máquina h a la máquina i $(hi = 12, 21)$. Ordenando los dos últimos conjuntos con el algoritmo de Johnson, obtenemos una planificación óptima ejecutando los trabajos de M_1 en el orden J_{12}, J_1, J_{21} y los de M_2 en el orden J_{21}, J_2, J_{12} .

$J2|p_{ij}=1|C_{max}$ tiene complejidad $O(n \log n)$ (Hefetz y Adiri (1979)).

$J2|m_j \leq 3|C_{max}$; $J3|m_j \leq 2|C_{max}$ son binarios NP-duros (Lenstra y otros (1977), González y Sahni (1978a)).

$J2|p_{ij} \in \{1,2\}|C_{max}$; $J3|p_{ij}=1|C_{max}$ son unarios NP-duros (Lenstra y Rinnooy Kan (1979)) y aún permitiendo interrupciones lo siguen siendo.

$J||C_{max}$: El problema job shop, en general, es extremadamente duro de resolver, de hecho, Muth y Thompson plantearon en 1963 un problema con 10 trabajos y 10 máquinas (Muth y Thompson (1963)) para cuya resolución se tuvo que esperar hasta 1989 (Carlier y Pinson (1989)).

No obstante, Balas (1985), ha formulado el problema como un problema de dar orientación a ciertos arcos no dirigidos de un grafo G de manera que se minimice la longitud del camino más largo de G. Dicho grafo G se caracteriza por la terna $G = (N^o, A^o, E)$ donde N es el conjunto de operaciones en las que se dividen los n trabajos, A es el conjunto de arcos orientados que indican las precedencias entre dichas operaciones, E es el conjunto de arcos no orientados. Existe el par $\{i, j\} \in E$ si las operaciones i y j deben hacerse en la misma máquina. $N^o = N \cup \{0\}$ y $A^o = A \cup \{ (0, j) / \text{a la operación } j \text{ no le precede ninguna otra}\}$ son los conjuntos ampliados de nodos y arcos dirigidos.

El problema de seleccionar la orientación óptima para los arcos de E puede tratarse como un problema de programación disyuntiva sin recurrir a variables enteras. Balas, en su artículo, da una caracterización parcial del poliedro asociado a ese problema ya que desarrolla un método para determinar facetas de dicho poliedro y decidir qué facetas son adyacentes. Finalmente da un procedimiento para encontrar una desigualdad que separa una solución dada de una región determinada por un conjunto de restricciones.

3. Algoritmos Propuestos

3.1. Máquinas no especializadas

3.1.1. Problema P||Cmax.

Planteamiento del problema

Se considera un problema de planificación determinística que ha motivado el interés de muchos investigadores: la planificación de *n* trabajos independientes sobre dos ó más máquinas o procesadores idénticos con el objetivo de minimizar el tiempo global de completación de todos los trabajos.

Formalmente, dados un conjunto de *n* trabajos y un conjunto de *n* enteros positivos $\{p_1,...,p_n\}$, donde p_j es el tiempo de proceso del trabajo *j*, se trata de planificar la ejecución de los trabajos por los procesadores de modo que el instante de completación de todos los trabajos se presente lo antes posible. La planificación a determinar debe contemplar las hipótesis de que cada trabajo puede ser procesado en un instante dado por a lo sumo un procesador, y que todos los trabajos están disponibles para ser procesados desde el instante inicial.

Los trabajos pueden planificarse sobre las máquinas de manera continua o interrumpida, es decir, puede o no permitirse la interrupción temporal de los trabajos. Cuando es factible dicha interrupción, la ejecución de un trabajo puede abandonarse temporalmente, y completar su procesamiento posteriormente, quizás en otra máquina diferente. En este caso, el problema $P|pmtn|C_{max}$ (y, por tanto, $P2|pmtn|C_{max}$) puede resolverse de manera muy eficiente en tiempo polinomial. Es fácil ver que la solución óptima es el máximo entre dos valores: el tiempo máximo de proceso de las tareas, y el promedio sobre el número de máquinas del tiempo total de proceso de todos los trabajos (McNaughton (1959)).

En el caso de que los trabajos no se puedan interrumpir, el problema de minimizar el tiempo global de completación de los trabajos es más difícil. De hecho, Garey y Johnson (1979) probaron que el problema $P||C_{max}$ es NP-duro en sentido fuerte. Incluso $P2||C_{max}$ es también NP-duro (Blazewicz (1987)). Referencias básicas para este problema son las siguientes: Baker (1974), French (1982), Graves y otros (1993), Lawler y otros (1982). Debido a la intratabilidad del problema, el esfuerzo de muchos investigadores ha sido dirigido hacia el

diseño y análisis de algoritmos aproximados (Hochbaum y Shmoys (1987)). No obstante, y como ya se indicó en el segundo epígrafe de este capítulo, para pequeños valores de m y U, siendo m el número de máquinas y U una cota superior de la solución óptima, el problema puede resolverse óptimamente mediante programación dinámica como se describe en Blazewicz (1987). El problema también tiene analogía con el problema "bin packing". Esto ha permitido a Dell'Amico y Martello (1994) construir un algoritmo de ramificación y acotación para el problema $P||C_{max}$.

A continuación consideramos el problema en el que los trabajos no puedan interrumpirse. Se propone un algoritmo exacto para $P2||C_{max}$ y una derivación heurística del mismo. También se desarrollan heurísticas para el problema $P||C_{max}$.

Algoritmo P2||Cmax

El método que se presenta es un método backtracking para obtener la planificación que minimiza el instante final de completación de todos los trabajos *("makespan")*. El problema puede formularse como un problema de programación entera 0-1 de la siguiente manera:

$$min \left| \sum_{j=1}^{n} \xi_j p_j - M^* \right|$$

s.a.:

$$\xi_j = 0 \quad o \quad \xi_j = 1$$

donde ξ_j es *1* sí, y sólo si, el trabajo j se asigna a la primera máquina y *0* si se asigna a la segunda; y la constante M^* es

$$M^* = max\left\{ max\{p_j\}, \sum_{j=1}^{n} \frac{p_j}{2} \right\}$$

Algoritmo

1. *(Inicialización)*.
   - Sea $J = \{1,...,n\}$ la lista de trabajos en orden no creciente de tiempos de proceso *(orden LPT)*, es decir, $p_1 \geq p_2 \geq ... \geq p_n$.
   - Poner $k=1$; $S=\{1\}$; $S^*=\{1\}$; $rama=0$; $primero = \{2,3,...,n\}$; $último = \{n,n,...,n\}$ con $|último|=n-1$; $c=p_1$; $\theta=p_1-M^*$; $c^*=p_1$; $\theta^*=|p_1-M^*|$.
   - Si $\theta^*=0$, ó $\theta > 0$, entonces parar, S^* es la solución óptima.

- En otro caso, continuar en el paso 2.

2. *(Test de avance-retroceso).*
   - Si $primero[k] \leq último[k]$ ir al paso 3.
   - En otro caso:
     - Si $k \neq 1$ ir al paso 4.
     - Si $k = 1$ parar, S^* es la solución óptima.

3. *(Avance).*
   - Actualizar $S = S \cup \{primero[k]\}$; $c = c + p_{primero[k]}$; $\theta = c - M^*$.
   - Preparar para backtracking: $primero[k] = primero[k] + 1$.
   - Si $|\theta| < \theta^*$, entonces $\theta^* = |\theta|$; $S^* = S$; $c^* = c$.
   - Si $\theta = 0$, parar, S^* es la solución óptima.
   - Si $\theta < 0$, entonces poner $primero[k+1]=S[k+1]+1$; $k=k+1$; e ir al paso 2.
   - Si $\theta > 0$, entonces poner $k = k + 1$; $rama = rama + 1$; e ir al paso 4.

4. *(Retroceso).*
   - Si $\theta > 0$, entonces poner $c=c - p_{S[k]}$; $S=S - \{S[k]\}$; $k=k - 1$; e ir al paso 2.
   - En otro caso, un trabajo corto ha sido introducido sin alcanzar M^*. Sea $S=\{i_1, i_2, ..., i_{q-1}, i_q, i_{q+1}, ..., i_k\}$, donde q es el menor índice tal que $p_{i_l}=p_{n-(k-l)}$ para $l = k, k-1, ..., q+2, q+1$. Entonces se continúa de esta manera:
     - Si $q \neq 1$, actualizar $S=\{i_1, i_2, ..., i_{q-1}\}$; $c = \sum_{l=1}^{q-1} p_{i_l}$; $\theta=c - M^*$; $rama=rama + 1$; para cada $l = k, k-1, ..., q+1, q$, hacer $último[l]=último[l] - (k - l)$; $k = q-1$, e ir al paso 2.
     - Si $q=1$, poner $rama=rama + 1$, y parar. S^* es la solución óptima.

Ciertamente, el tiempo de ejecución del algoritmo no es polinomial puesto que se visitan un gran número de ramas durante la búsqueda. Recordemos que el problema es NP-duro. Partiendo del algoritmo anterior, es posible construir un método heurístico que no explore un gran número de ramas del árbol de búsqueda. Para ello, podemos cambiar el paso 4 del algoritmo por el paso señalado a continuación para obtener un método aproximado.

4'. *(Retroceso).*
   - Si $\theta > 0$, entonces poner $c=c - p_{S[k]}$; $S=S - \{S[k]\}$; $k=k - 1$; e ir al paso 2.
   - En otro caso, un trabajo corto ha sido introducido sin alcanzar M^*. Sea $S=\{i_1, i_2, ..., i_{q-1}, i_q, i_{q+1}, ..., i_k\}$, donde q es el menor índice tal que $p_{i_l} = p_{n-(k-l)}$ para $l = k, k-1, ..., q+2, q+1$. Posteriormente se continúa de esta manera:
     - Si $q>2$, actualizar $S=\{i_1, i_2, ..., i_{q-2}\}$; $c = \sum_{l=1}^{q-2} p_{i_l}$; $\theta=c-M^*$; $rama=rama+1$; para cada $l = k, k-1, ..., q+1, q$, hacer $último[l]=último[l]-(k - l)$; $k = q-2$, e ir al paso 2.
     - Si $q \leq 2$, poner $rama = rama + 1$, y parar. S^* es una solución heurística.

A continuación se estudia la complejidad computacional teórica de este algoritmo.

Complejidad del Algoritmo

La complejidad del paso 1 es *O(n log n)* y los otros pasos tienen una complejidad de *O(nR)*, donde *R* es el número de ramas exploradas (es decir, el valor final de la variable *rama*). Por tanto, la complejidad del algoritmo es el máximo entre ambas cantidades. Generalmente *R* es mayor que *log n*, y *O(nR)* sería la complejidad total del algoritmo. *R* es acotado por $2^{n-2} - 2^{n-2-[n/2]}$. Sin embargo, en la práctica el valor *R* es menor que esta cota. De hecho, en los ejemplos computados se obtienen pequeños valores de *R* en relación con el tamaño de la entrada al problema. Estos valores son aún menores en el algoritmo heurístico.

Algoritmos P||Cmax

Si se dispone de más de dos máquinas, y el número de trabajos es superior al número de máquinas, el problema es aún más difícil. Con dos máquinas, basta con conocer la planificación en una de ellas para tener también determinada la planificación en la otra. Con más de dos máquinas la situación cambia. Si el número de máquinas es mayor o igual que el número de trabajos, no hay problema porque se asignan los trabajos a las máquinas mediante una correspondencia uno a uno. El problema aparece cuando el número de máquinas es menor que el número de trabajos.

Una formulación del problema mediante un modelo de Programación Entera 0-1 puede construirse fácilmente. Sea ξ_{jk} una variable indicador definida como sigue: $\xi_{jk} = 1$ si el trabajo *j* se asigna a la máquina *k*, y $\xi_{jk} = 0$ en cualquier otro caso. Entonces, una formulación completa del problema como problema de Programación Entera es:

$$min \quad max_{1 \le k \le m} \left\{ \sum_{j=1}^{n} p_j \xi_{jk} \right\}$$

s.a.:

$$\sum_{k=1}^{m} \xi_{jk} = 1, \quad \forall j = 1, \ldots, n$$

$$\xi_{jk} = 0 \quad o \quad \xi_{jk} = 1$$

Esta formulación contiene *n* restricciones y *nm* variables, pero el objetivo es una función no lineal. Por tanto, resolver este problema es difícil y las aproximaciones estándar de la Programación Entera requieren un gran esfuerzo computacional. Esta es la razón de investigar procedimientos heurísticos para encontrar soluciones aproximadas a el problema. Propondremos a continuación tres procedimientos heurísticos. En ellos consideraremos la constante

$$M^* = max\left\{ max\{p_j\}, \sum_{j=1}^{n} \frac{p_j}{m} \right\}.$$

Procedimiento 1:

1. Aplicar el algoritmo $P2||C_{max}$ para obtener el conjunto de trabajos que debe procesar la primera máquina.

2. Repetir sucesivamente con las otras máquinas y trabajos no planificados hasta que no queden trabajos por asignar.

En este procedimiento dos planificaciones posibles aparecen para la misma máquina, una finalizando antes ó en el valor M^* y otra después de esta constante. La decisión sobre que planificación elegir para la máquina se toma de acuerdo a la siguiente idea.

Para cada máquina i computamos el resto $R^i = \sum_{k=0}^{i} R_k$, con $R_o = 0$; $R_k = \sum_{j \in S_k} p_j - M^*$, y S_k la planificación en la máquina k. Si $R^i \geq 0$ y aparece uno de esos "empates" cuando se busca la planificación para la máquina $i+1$, se prefiere planificar por debajo de la constante M^*. En otro caso, si $R^i < 0$ y encontramos un "empate", se decide planificar sobrepasando el valor M^*.

Complejidad del procedimiento 1

Para evaluar la complejidad de este procedimiento, nótese que utiliza *m* veces el algoritmo propuesto para $P2||C_{max}$. Por tanto, su complejidad será del orden $O(mnR)$.

Procedimiento 2:

1. Ordenar los trabajos en orden inverso al *SPT*, es decir, en orden *LPT*. Esto es, de modo que $p_1 \geq p_2 \geq ... \geq p_n$. Considerar la lista $U = \{1,...,n\}$ de trabajos no planificados.

2. Elegir el primer trabajo de la lista U, extraerlo de U, y asignarlo a la máquina 1.

3. Sea j el primer trabajo de la lista U. Asignar el trabajo j a la máquina i tal que

$$C_{max}(i) = max \{C_{max}(k) / 1 \leq k \leq m \ \ y \ \ C_{max}(k) + p_j \leq M^*\}.$$

Actualizar $C_{max}(i) = C_{max}(i) + p_j$ y extraer j de U.
Si tal máquina no existe, asignar el trabajo j a la máquina i con

$$C_{max}(i) = min \{C_{max}(k) / 1 \leq k \leq m \ \ y \ \ C_{max}(k) + p_j > M^*\}.$$

Actualizar $C_{max}(i) = C_{max}(i) + p_j$ y extraer j de U.

4. Si $U = \varnothing$, parar. En otro caso continuar en el paso 3.

Procedimiento 3:

1. Ordenar los trabajos en orden inverso al *SPT*, es decir, en orden *LPT*. Esto es, de modo que $p_1 \geq p_2 \geq ... \geq p_n$. Considerar la lista $U = \{1,...,n\}$ de trabajos no planificados.

2. Elegir el primer trabajo de la lista U, extraerlo de U, y asignarlo a la máquina 1.

3. Sea j el primer trabajo en la lista U. Para elegir la máquina en la cual asignar el trabajo j considerar el conjunto $S_j = \{r / C_{max}(r) + p_j \leq M^*\}$ y la cantidad $C_j = max_{k \in S_j} \{C_{max}(k)\}$.

- Si $S_j \neq \varnothing$, entonces asignar el trabajo j a la máquina i tal que

$$\left| C^j - (C_{max}(i) + p_j) \right| = min_{r \in S_j} \left\{ \left| C^j - (C_{max}(r) + p_j) \right| \right\}$$

Actualizar $C_{max}(i) = C_{max}(i) + p_j$ y extraer j de U.

- En otro caso, asignar el trabajo j a la máquina i tal que $C_{max}(i) = min_r \{C_{max}(r)\}$. Actualizar $C_{max}(i) = C_{max}(i) + p_j$ y extraer j de U.

4. Si $U = \varnothing$, parar. En otro caso continuar en el paso 3.

Complejidad de los procedimientos 2 y 3

Los procedimientos 2 y 3 tienen complejidad de orden $O(n \log n)$ en su paso 1. Los otros pasos de ambos algoritmos requieren un tiempo de $O(nm)$. Por tanto, la complejidad de ambos algoritmos es de $O(n \log n + nm)$.

3.1.2. Problema P|prec|∑Tj.

Planteamiento del problema

El problema que nos ocupa puede plantearse en los siguientes términos: se dispone de m máquinas idénticas con igual capacidad y velocidad de procesamiento, y n trabajos que deben ser procesados con dichas máquinas. Cada trabajo se asigna a una sola máquina y, en un instante de tiempo dado, cada máquina puede ejecutar un único trabajo. Una vez asignado un trabajo a una máquina, éste se ejecuta sin posibilidad de interrupción. Los trabajos vienen caracterizados por los tiempos de procesamiento p_j y las fechas límite d_j $(j=1,...,n)$ que representan los tiempos necesarios para ejecutarlos y los instantes de tiempo en los que debería completarse la ejecución de los mismos. Si algún trabajo j sobrepasa esa fecha límite se incurre en una demora L_j que será la diferencia entre la fecha de completación del trabajo C_j y la fecha límite d_j. El valor $T_j=max\{0, L_j\}$ será la tardanza asociada al trabajo j.

El objetivo deseado es buscar una planificación de los trabajos sobre las máquinas de manera que se minimice la tardanza total $\sum Tj$ de los trabajos. Plantearemos el modelo de planificación teniendo en cuenta la existencia de relaciones de precedencia entre los trabajos. También supondremos que todos los trabajos están disponibles para ser procesados desde el instante inicial, es decir, $r_j=0 \ \forall j$. El problema que se plantea se denota en la literatura como problema $P|prec|\sum Tj$. Este problema está en el grupo de los llamados NP-duros por su alto grado de dificultad, puesto que Graham y otros (1979) garantizan que este problema es más complejo que los problemas $P|prec,p_j=1|C_{max}$ y $1|prec,p_j=1|\sum Tj$; y, un año antes, Lenstra y Rinnooy Kan (1978) probaron que estos dos últimos problemas son NP-duros. La NP-dureza del problema invita a estudiarlo desde un punto de vista heurístico.

Algoritmo

Para modelizar el problema y construir una heurística que proporcione una solución del mismo consideremos que las relaciones de precedencia entre los trabajos vienen caracterizadas mediante un grafo dirigido $G = (X,A)$, de forma que el conjunto de vértices X sea el conjunto total de trabajos y cada arco $(i,j) \in A$

represente que el trabajo *j* requiere, para ser procesado, que haya finalizado previamente el procesamiento del trabajo *i*.

Podemos denotar una planificación *s* mediante un vector $s = (s^1,s^2,...,s^m)$ donde $s^i = (J_{i,1}, J_{i,2}, ..., J_{i,n_i})$ es la secuencia de trabajos asignada a la máquina *i* $(i = 1,...,m)$. De acuerdo con ello, toda planificación $s = (s^1,s^2,...,s^m)$ sobre las *m* máquinas será *factible* si es compatible con el grafo $G = (X,A)$, es decir, no vulnera ninguna de las restricciones de precedencia fijadas por el grafo *G*.

Comenzaremos asignando longitudes a las aristas del grafo de precedencias $G = (X,A)$. La longitud de la arista $(i,j) \in A$ es el tiempo de procesamiento p_i del trabajo J_i. Se añaden dos vértices ficticios al grafo, llamados vértice inicio *I* y vértice final *F*. Se construyen aristas de longitud 0 desde el vértice *I* a todos los trabajos de nivel 1 (aquellos que no tienen predecesores), y aristas (i,F) desde aquellos trabajos J_i que no tienen trabajos siguientes, al vértice final *F*. Se asigna la longitud p_i a cada arista (i,F).

A continuación se aplica un algoritmo de caminos máximos (por ejemplo el de Dijkstra (1959), o la técnica utilizada en el método PERT de planificación de proyectos para determinar los tiempos "early" y "last" de cada vértice del grafo *PERT*) para etiquetar cada trabajo J_j con la marca e_j = *longitud del camino máximo desde I a* J_j (tiempo *"early"* del vértice J_j).

Denotaremos por n_l el número de trabajos de nivel *l* y por $n^* = max_l \{n_l\}$. Si el número de máquinas *m* es mayor o igual que n^*, la planificación óptima para el problema planteado se obtiene fácilmente por el procedimiento siguiente cuya complejidad es $O(mn^2)$ (si utilizamos el algoritmo de Dijkstra para detectar los tiempos "early") ó bien $O(mn \log n)$ (si utilizamos la técnica asociada al método *PERT*):

Algoritmo (óptimo para $m \geq n^*$***):***

0. Inicialmente todas las máquinas están disponibles. Una vez que se asignen los trabajos de un camino crítico a una máquina, ésta deja de estar disponible y, por tanto, no se le asignarán más trabajos.

1. Se determina el camino crítico de *I* a *F* (camino de longitud máxima) y se asignan a la máquina 1 los trabajos correspondientes a vértices de dicho camino, manteniendo la ordenación.

2. Quitar del grafo de precedencias los vértices *I* y *F*, junto con los vértices correspondientes a trabajos asignados y sus respectivos arcos incidentes. Así obtendremos un grafo parcial. Ordenar el nuevo grafo por niveles. Volver a

construir los vértices *I* y *F* y los arcos asociados *(I,i)* y *(j,F)*, estableciendo las longitudes correspondientes como ha sido mencionado anteriormente.

3. Sobre el nuevo grafo, obtener el camino crítico de *I* a *F* y asignar ordenadamente la secuencia de trabajos a la primera máquina disponible, pero teniendo en cuenta que el inicio de la secuenciación debe ser desde el valor e_i, siendo *i* el primer trabajo de este camino crítico.

4. Repetir sucesivamente los pasos 2 y 3, con las restantes máquinas, hasta que se planifiquen todos los trabajos.

Analicemos ahora el otro caso. Si el número de máquinas es inferior al número máximo de trabajos por nivel del grafo inicial *G*, entonces la solución óptima no es sencilla de encontrar, de hecho, como se mencionó anteriormente, el problema es NP-duro.

Propondremos a continuación un procedimiento que ofrece una solución próxima a la óptima. Comenzaremos con una solución sencilla y fácil de generar. A partir de ella, modificaremos la asignación y colocación de los trabajos de forma que se reduzca en cada paso la tardanza total $\sum Tj$. Partiremos de la lista que ordena los trabajos en orden creciente de niveles y, dentro de cada nivel, en orden creciente de fechas límite *(orden EDD)*. Esta última lista será la que utilizaremos para asignar los trabajos a las máquinas y, de esta manera, obtener una planificación semilla en tiempo $O(n^2)$, que posteriormente será mejorada por el algoritmo $P|prec|\sum Tj$ de complejidad $O(n^2 \sum p_j)$.

Algoritmo semilla:

1. Establecer la ordenación de trabajos por niveles de acuerdo con el grafo que fija las relaciones de precedencia. Ordenar los trabajos de cada nivel en orden *EDD*. Tendremos una lista *U* de todos los trabajos. Poner $C_{max}(i) = 0, \ \forall i = 1,...,m.$ ($C_{max}(i)$ representa el instante en el que la máquina *i* finaliza la ejecución del último trabajo asignado a ella) y $C_j = 0, \ \forall j = 1,...,n.$ (C_j representa el instante en el que finaliza la ejecución del trabajo *j*).

2. Coger el primer trabajo *j* de la lista *U*.
   - Si existe alguna máquina *i* con $C_{max}(i) = e_j$, asignar el trabajo *j* a dicha máquina e ir al paso 4.

   - En otro caso, ir al paso 3.

3. - Si existen máquinas con $C_{max}(i) < e_j$, asignar el trabajo *j* a la máquina i^* tal que

$$C_{max}(i^*) = max \{C_{max}(i) / C_{max}(i) < e_j\}.$$

- Si no existen tales máquinas, asignar el trabajo j a la máquina i^* tal que

$$C_{max}(i^*) = min \{C_{max}(i) / C_{max}(i) > e_j\}.$$

4. *(Actualización)*. Hacer $U = U - \{j\}$. Si el trabajo j se asigna a la máquina i, y si denotamos con $\Gamma(j) = \{k / (j,k) \in A\}$, hacer lo siguiente:

$$C_{max}(i) = max \{C_{max}(i), e_j\} + p_j;$$

$$C_j = C_{max}(i);$$

$$e_k = max \{e_k, C_j\} \quad \forall k \in \Gamma(j).$$

- Si $U = \varnothing$, parar. La planificación que tenemos es la salida del algoritmo.
- Si $U \neq \varnothing$, volver al paso 2.

Algoritmo P\|prec\|ΣTj:

0. Aplicar el algoritmo semilla para obtener una planificación inicial s.

1. Seleccionar el primer trabajo tardío $(L_j > 0)$ de la planificación inicial. (Si no existiese ningún trabajo tardío s sería óptima y pararíamos).

2. Una vez seleccionado el trabajo j, quitar de la planificación s todos aquellos trabajos cuya fecha de comienzo sea mayor o igual que e_j. Sea R el conjunto de trabajos quitados. Poner $C_k = 0, \ \forall k \in R$.

3. Actualizar las cantidades $C_{max}(i)$ para cada máquina $i = 1,...,m$ poniendo $C_{max}(i) = C_{l_i}$ siendo l_i el último trabajo planificado en la máquina i.
   Para cada trabajo j de R calcular su e_j = tiempo "early" del trabajo J_j , esto es, la longitud del camino máximo desde I hasta j en el grafo inicial G; y luego actualizar estos e_j colocando

$$e_j = max [e_j , max_k \{C_k / (k,j) \in A\}].$$

4. Ordenar los trabajos de R en orden creciente de niveles y, dentro de cada nivel, en orden *EDD*. Llamar U a la lista obtenida.

5. Poner $U = R$ y repetir los pasos 2, 3, y 4 del algoritmo semilla hasta que la lista U esté vacía y se obtenga otra planificación s'.

6. - Si la nueva planificación s' tiene mayor o igual $\sum Tj$ que la planificación actual s, quitar los trabajos recientemente asignados y volver a considerar el mismo conjunto R. Poner $C_k = 0 \quad \forall k \in R$. Actualizar los $C_{max}(i)$ y e_j como en el paso 3. Ir al paso 7.

   - En otro caso, tomar por planificación actual la nueva planificación $(s = s')$ y volver al paso 1.

7. Aplicar el algoritmo para el problema $1|prec|\sum Tj$ dado en el capítulo II y obtener una secuencia de los trabajos del conjunto R. A partir de ella, y tomando $U = R$, ir asignando ordenadamente los trabajos a las máquinas de acuerdo con su e_j, como en los pasos 2, 3, y 4 del Algoritmo semilla, hasta obtener otra nueva planificación s'.

8. - Si la nueva planificación tiene mayor o igual $\sum Tj$ que la planificación actual, parar.

   - En otro caso, tomar como planificación actual la nueva planificación y volver al paso 1.

Complejidad del algoritmo

Como ya ha sido comentado, el algoritmo $P|prec|\sum Tj$ tiene una complejidad computacional teórica de $O(n^2 \sum p_j)$. En el apéndice B se hace un estudio computacional de este algoritmo.

3.2. Máquinas especializadas

3.2.1. Problema O||Cmax

Planteamiento del problema

El problema de minimizar el instante final de completación de todos los trabajos en un open-shop, denotado por $O||C_{max}$ de acuerdo con la clasificación propuesta por Graham y otros (1979) y Lawler y otros (1993), puede describirse de la siguiente manera: disponemos de m máquinas $M_1, ..., M_m$ para la realización de n trabajos $J_1, ..., J_n$. Cada trabajo J_j se descompone en m operaciones O_{ij} con i

= 1, ..., m. La operación O_{ij} se ejecuta en la máquina M_i en un tiempo p_{ij}. Para cualquier trabajo dado, el orden de proceso de sus operaciones no es relevante. No se permiten interrupciones de los trabajos, esto es, no es posible parar temporalmente la ejecución de una operación para luego retomarla en el punto donde se detuvo su ejecución. La única restricción adicional es que no se permiten solapamientos, es decir, un mismo trabajo no puede ejecutarse por más de una máquina simultáneamente.

La función objetivo a minimizar en el problema $O||C_{max}$ es el tiempo total necesario para completar todos los trabajos. Sea C_j el instante de completación del trabajo *j;* el instante de completación de todos los trabajos será $C_{max} = max_{1 \leq j \leq n} \{C_j\}$. González y Sahni (1976) probaron que el problema sin interrupciones $O||C_{max}$ con un valor arbitrario $m > 2$ es NP-duro en sentido fuerte, mientras que el problema con $m = 2$, también denotado por $O2||C_{max}$, puede resolverse óptimamente en tiempo lineal. Variantes de este problema han sido estudiadas por otros autores.

González y Sahni (1976) presentan un algoritmo polinomial que proporciona una solución óptima para el problema con interrupciones y un número arbitrario de máquinas, problema $O|pmtn|C_{max}$.

Ishii y Nishida (1986) han desarrollado un algoritmo de complejidad *O(n log n)* para la variante del problema $O2||C_{max}$ donde la velocidad de las máquinas es controlable. Su algoritmo encuentra la velocidad óptima para cada máquina y la planificación óptima.

El problema $O2|rd|C_{max}$ donde una restricción de dependencia de ruta se establece, es decir, el tiempo de procesamiento de un trabajo depende de la ruta o trayectoria en la que el trabajo se procese a través de las máquinas, ha sido mostrado como binario NP-duro por Adiri y Amit (1983).

El caso especial del problema $O3||C_{max}$ en el cual $max_j \{p_{hj}\} \leq min_j \{p_{ij}\}$ para cierto par de máquinas M_h , M_i con $h \neq i$, es decir, una máquina (la *i*) domina a la otra (problema $O3|dm|C_{max}$), puede resolverse en tiempo polinomial (Adiri y Hefetz (1980), Adiri y Aizikowitz (1989)). La principal idea del algoritmo de *O(n)* propuesto por Adiri y Aizikowitz (1989) es no considerar la máquina dominada y aplicar el algoritmo de González y Sahni (1976) a las otras dos máquinas, para después secuenciar las operaciones de la máquina dominada sin conflictos.

Liu y Bulfin (1987) estudiaron problemas open-shop ordenados con trabajos y máquinas ordenadas. Un open shop tiene trabajos ordenados si $p_{i,j} \leq p_{i,j+1}$ para $i = 1, ..., m$ y $j = 1, ..., n-1$. Se dice también que tiene máquinas ordenadas si $p_{i,j} \leq p_{i+1,j}$ para $i = 1, ..., m-1$ y $j = 1, ..., n$. Un open shop se dice ordenado *(ord)* si tiene tanto máquinas ordenadas como trabajos ordenados. Estos autores han probado que $O3|ord|C_{max}$ es binario NP-duro.

En los epígrafes siguientes se proponen algoritmos heurísticos para $O||C_{max}$ y se desarrolla un algoritmo de búsqueda tabú para este problema. Un estudio computacional de estos algoritmos se realiza en el Apéndice B.

Algoritmos heurísticos para O||Cmax

Se describen a continuación tres algoritmos determinísticos sencillos basados en *"list-scheduling"* ó planificación de listas. Estos algoritmos consideran sólo *"planificaciones activas" ó "planificaciones plenas"*, es decir, planificaciones en las que, en cada instante de tiempo, al menos una máquina está procesando algún trabajo. Este tipo de planificaciones son un conjunto dominante dentro del conjunto de planificaciones solución (Baker, 1974).

Los algoritmos heurísticos que se proponen, y cuyo análisis del caso peor se discute (determinando para cada algoritmo el radio del caso peor) pueden usarse para obtener soluciones aproximadas al problema $O||C_{max}$. .

Antes de comenzar la descripción de los algoritmos heurísticos se introduce una cota inferior inmediata debida a González y Sahni (1976) que será útil en lo sucesivo. Una cota inferior al tiempo de completación de todos los trabajos o "makespan" es:

$$LB = max\left\{max_i\left\{\sum_{j=1}^{n} p_{i,j}\right\}, max_j\left\{\sum_{i=1}^{m} p_{i,j}\right\}\right\}$$

Esta cota se justifica por el hecho de que una máquina no puede procesar más de dos operaciones simultáneamente, y, por eso, al menos un intervalo de tiempo de amplitud $\sum_{j=1}^{n} p_{i,j}$ se necesita para que la máquina M_i procese todas sus operaciones; de ahí el máximo de estas sumas es una cota inferior. Por otro lado, como dos operaciones de un mismo trabajo no pueden procesarse simultáneamente, al menos un intervalo de tiempo de amplitud $\sum_{i=1}^{m} p_{i,j}$ se requiere para completar la ejecución del trabajo J_j ; por tanto, el máximo de estas sumas es también una cota inferior. En consecuencia, LB es una cota inferior de la función objetivo del problema $O||C_{max}$.

Las tres heurísticas son algoritmos de planificación de listas. Cada una de ellas esta caracterizada por utilizar un criterio diferente para determinar el orden en el que los trabajos se consideran. Las tres usan el mismo planificador para construir la solución factible. Cuando el orden es conocido, los trabajos se planifican en cada máquina de acuerdo con el orden prefijado. Para ello, el planificador construye la solución factible seleccionando la primera máquina

disponible que tiene alguna operación aún no planificada. Los empates se rompen arbitrariamente. La próxima operación no planificada, de acuerdo con el orden de asignación para esa máquina, se planifica en ella *"tan pronto como sea posible"*. Esto significa que la operación seleccionada se planifica en la máquina comenzando en el instante más temprano de modo que no haya solapamientos con ninguna otra operación ya planificada del mismo trabajo.

Heurística LPT

En esta primera heurística, la solución factible se obtiene planificando las operaciones sobre cada máquina de acuerdo con la bien conocida regla *LPT*, es decir, en cada máquina las operaciones se consideran en orden no creciente de los tiempos de proceso. Veamos un ejemplo:

Ejemplo 2:

Considérese el problema open shop con tres máquinas y cuatro trabajos y los siguientes tiempos de proceso de las operaciones:

trabajo ***máquina***	***1***	***2***	***3***	***4***	$\sum_j p_{ij}$
1	*10*	*8*	*5*	*3*	*26*
2	*6*	*7*	*9*	*9*	*31*
3	*7*	*8*	*3*	*3*	*21*
$\sum_i p_{ij}$	*23*	*23*	*17*	*15*	*LB = 31*

Los órdenes *LPT* de las máquinas son:

máquina	***orden LPT***
1	*1-2-3-4*
2	*3-4-2-1*
3	*2-1-3-4*

y la asignación de las operaciones a las máquinas con la estrategia "tan pronto como sea posible" proporciona la siguiente planificación factible:

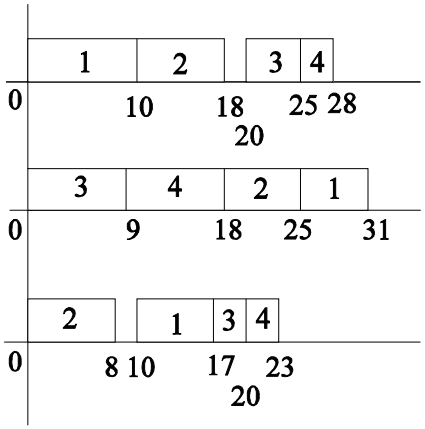


Figura 4. Ilustración del ejemplo 2.

cuyo instante de completación de todos los trabajos es 31, es decir, es una planificación óptima.

Heurística SPT

En esta heurística, la solución factible se obtiene planificando las operaciones en cada máquina de acuerdo con la bien conocida regla *SPT*, es decir, en cada máquina las operaciones se consideran en orden no decreciente de los tiempos de proceso. Veamos un ejemplo.

Ejemplo 3:

Considérese el mismo problema que en el ejemplo de la heurística anterior. El orden SPT en las máquinas es:

máquina	***orden SPT***
1	*4-3-2-1*
2	*1-2-3-4*
3	*3-4-1-2*

y la solución factible es:

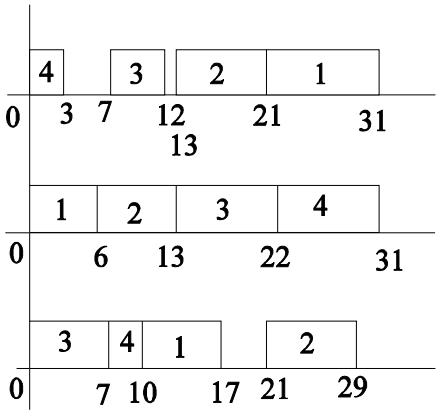


Figura 5. Ilustración del ejemplo 3.

con un valor del makespan de 31, es decir, óptimo.

Heurística LIS

La última heurística basada en planificación de listas propuesta es la heurística *LIS*. Este algoritmo heurístico actúa de la siguiente manera. Las máquinas se emparejan en pares disjuntos. Luego, a cada par, se le aplica el algoritmo de complejidad *O(n)* debido a González y Sahni (1976) para resolver el correspondiente problema $O2||C_{max}$. De esta manera, se obtiene una permutación de las operaciones para cada máquina. Así, cada máquina tiene un orden de asignación de las operaciones. Los pares de máquinas se escogen arbitrariamente. Si el número de máquinas es impar, hay una máquina que no forma parte de ningún par. Para esta máquina se establece un orden de asignación arbitrario.

Dos rutinas sencillas de mejora

Si *s* es una solución para la cual no existe tiempo ocioso ni en la máquina *i* que define el tiempo total de completación de todos los trabajos ni en el trabajo *j* que define dicho instante, entonces *s* alcanza la cota inferior y estamos ante una solución óptima. En otro caso, este par (máquina *i* , trabajo *j*) define una operación $x = O_{ij}$.

La primera rutina de mejora intenta planificar *x* antes sin cambiar los tiempos de completación de las otras operaciones y satisfaciendo las restricciones de no solapamientos.

La segunda rutina de mejora reconstruye toda la planificación. Para ello, identifica el instante temporal de comienzo *"cio"* del primer, segundo y sucesivos intervalos de tiempo ocioso para la máquina que define el tiempo de completación de todos los trabajos, reconstruyendo todas las posibles planificaciones con el planificador de listas partiendo de la planificación parcial s_p , y tomando la mejor solución construida como salida de la rutina de mejora. La planificación parcial s_p es la planificación *s* prescindiendo de todas las operaciones cuyos tiempos de completación exceden de *"cio"*, y la operación *x* que define el tiempo total de completación de los trabajos de *s* planificada en el intervalo $[cio, cio+p_x]$.

Análisis del caso peor para los algoritmos heurísticos propuestos

Dada una entrada *I* de un problema de optimización, sea $v^*(I)$ el valor de la solución óptima del problema, y sea $v^h(I)$ el valor de la solución obtenida usando el algoritmo heurístico. El radio de ejecución del caso peor de un algoritmo heurístico *h* es el menor valor *k* tal que

$$\frac{v^h(I)}{v^*(I)} \leq k \quad para\ cada\ I.$$

A continuación se obtiene el radio de ejecución para el caso peor de cualquier heurística que produzca una planificación activa o plena para $O||C_{max}$.

Proposición 1:

Sea s^* una solución óptima del problema $O||C_{max}$ y *s* cualquier planificación activa para este problema. Entonces

$$\frac{v(s)}{v(s^*)} \leq m.$$

Demostración:

Sea $P = \sum_{i=1}^{m} \sum_{j=1}^{n} p_{ij}$, entonces, para cualquier planificación activa *s,* $v(s) \leq P$. También tenemos que para cualquier planificación óptima s^* se cumple $v(s^*) \geq P/m$; entonces, $\frac{v(s)}{v(s^*)} \leq m.$ □

Como puede verse en el siguiente resultado, la cota *m* del caso peor que se acaba de obtener para planificaciones activas puede reducirse a $\left\lceil \frac{m}{2} \right\rceil$, es decir, al menor entero que es mayor o igual a *m/2*, para el algoritmo *LIS.*

Proposición 2:

Sea s^* una solución óptima de cualquier problema $O||C_{max}$ y s_{LIS} la solución obtenida mediante el algoritmo *LIS*. Entonces

$$\frac{v(s^{LIS})}{v(s^*)} \leq \left\lceil \frac{m}{2} \right\rceil.$$

Demostración:

Para cada par de máquinas *i,* $1 \leq i \leq \left\lceil \frac{m}{2} \right\rceil$, sea s^i la planificación obtenida con el algoritmo de *O(n)* de González y Sahni (1976) para $O2||C_{max}$. Cada una de tales planificaciones es óptima para el correspondiente par de máquinas. Por lo tanto, $v(s^*) \geq max_i \{v(s^i)\}$.

En el caso peor la planificación obtenida con el algoritmo *LIS* tiene un valor de la función objetivo igual a $v(s_{LIS}) \leq \sum_i v(s^i) \leq \left\lceil \frac{m}{2} \right\rceil max_i \{v(s^i)\}$. En consecuencia, $\frac{v(s_{LIS})}{v(s^*)} \leq \left\lceil \frac{m}{2} \right\rceil$ □.

Veamos ahora una heurística tabú para el problema $O||C_{max}$. que utiliza las soluciones de las heurísticas anteriores como soluciones iniciales.

Una heurística tabú para O||Cmax

Antes de establecer un algoritmo de búsqueda tabú para el problema que nos ocupa se comentan las líneas generales de la metodología tabú.

La metodología de búsqueda tabú

Dado un problema de optimización *P*, sea *S* el conjunto de soluciones factibles para *P* y sea *c* una función de costo $c: S \rightarrow \Re$. Para diseñar un algoritmo de *búsqueda local* para *P* necesitamos comenzar con una solución factible $s \in S$, y definir una estructura de vecinos $N: S \rightarrow 2^S$. La búsqueda comienza con una solución inicial *s* y continúa a uno de sus vecinos *s'*, seguidamente genera el conjunto de vecinos *N(s')*, y repite nuevamente hasta que se satisfaga un criterio de parada. En este proceso el algoritmo almacena la mejor solución encontrada s^*.

La *búsqueda tabú* es un método de optimización basado en técnicas de búsqueda local. La búsqueda evoluciona desde una solución factible al mejor de sus vecinos no prohibidos. Esta estrategia permite evitar ciclos, y guía la búsqueda a regiones inexploradas. Sin embargo, almacenar soluciones completas en una lista de soluciones prohibidas y comprobar si una solución candidata pertenece o no a la lista resulta generalmente excesivo e inaceptable en términos de requerimientos de memoria y tiempo de computación. Por ello, normalmente en la lista tabú se almacena solamente el movimiento opuesto al movimiento aplicado durante la búsqueda para transformar una solución en una nueva (es decir, el movimiento que hace regresar desde la solución nueva a la antigua). Una solución *s'* se considera prohibida si la solución actual *s* puede transformarse en la solución *s'* aplicando uno de los movimientos en la lista tabú. Además de la característica de ser o no tabú, un criterio de aspiración se le asocia a cada movimiento. Si el movimiento tabú actual satisface el criterio de aspiración asociado se considera un movimiento admisible.

Una descripción general de un algoritmo de búsqueda tabú puede presentarse de la siguiente manera:

```
Procedure TS
  begin
    encontrar una solución factible inicial s;
    c*:=c(s);  s*:=s; lista_tabu:=∅;
    repeat
      Sea A(s) el conjunto de candidatos, A(s):={s'∈N(s) tales que el
      movimiento de s a s' no es tabú, ó satisface un criterio de aspiración};
      Elegir s̄ ∈ A(s) tal que s̄ tiene la menor estimación de la función de
      costo c;
      Poner el movimiento que lleva de s̄ a s en la lista tabú;
      s:= s̄ ;
      if c(s) < c* then
        begin
        c*:=c(s);  s*:=s;
        end
    until criterio de parada = VERDAD;
    return s* y c*;
```

end

Para explicar el procedimiento que se propone para generar soluciones factibles iniciales y las estructuras de vecinos adoptadas, se introduce primero un modelo de grafo para el problema $O||C_{max}$.

Un modelo de grafo disyuntivo para el problema O||Cmax

El problema $O||C_{max}$ puede describirse mediante un modelo de grafo disyuntivo. La teoría de modelos de grafos disyuntivos se debe a Roy y Sussmann (1964). Estos modelos han sido aplicados por Laarhoven, Aarts y Lenstra (1988) y Dell'Amico y Trubian (1993) para el problema $J||C_{max}$. Ellos consideran un grafo disyuntivo $G = (V, A, E)$ donde V es el conjunto de nodos, A es el conjunto de arcos dirigidos conjuntivos, y E es el conjunto de arcos no dirigidos disyuntivos (lados), definidos como sigue:

$V = \Omega \cup \{0\} \cup \{N+1\}$, donde Ω es el conjunto de todas las operaciones a realizar, $N=|\Omega|$, y $\{0\}$ y $\{N+1\}$ son nodos especiales que identifican los instantes de comienzo y finalización de todas las tareas del job shop.

$A = \{(i,j)$: la operación i es un predecesor inmediato de la operación j en la cadena del trabajo J_i $(=J_j)\} \cup \{(0,j): j \in \Omega\} \cup \{(i,N+1): i \in \Omega\}$.

$E = \{(i,j): i,j \in \Omega$ y las operaciones i y j no deben hacerse en la misma máquina$\}$.

Este modelo puede extenderse al problema $O||C_{max}$. Sin embargo, nótese que, en problemas job shop, el orden en que se realizan las operaciones de cada trabajo es conocido. Por tanto, existe un conjunto de arcos A en el modelo y el problema consiste en decidir una orientación de los lados de E para obtener un grafo acíclico con camino máximo *(camino crítico)* de 0 a $N+1$ mínimo. Esta longitud es el valor de la función objetivo para la planificación asociada.

Mientras tanto, en problemas open shop, no existe ningún orden fijado para el procesamiento de las operaciones de un trabajo, ni tampoco para las operaciones de una máquina. No existe ningún conjunto fijo A de arcos orientados. Además, se consideran dos conjuntos de lados E_J, E_M donde:

$E_J = \{(i,j)$ tales que i,j son operaciones de un mismo trabajo$\} \cup \{(0,j) : j \in \Omega\} \cup \{(i,N+1) : i \in \Omega\}$.

$E_M = \{(i,j)$ tales que i,j son operaciones de una misma máquina$\}$.

A cada vértice $i \in \Omega$ se le asocia un peso p_i. Los vértices *0* y *N+1* tienen peso *0*. El instante de comienzo y el instante de completación de los vértices *0* y *N+1* representan, respectivamente, el instante de comienzo y finalización de todas las tareas que componen el open shop. Se puede comprobar fácilmente que cualquier orientación de los lados que no crea ciclos corresponde a una planificación factible de las operaciones en las máquinas.

Puesto que longitud de un camino se define como la suma de los pesos de los vértices en el camino, el problema del open shop puede resolverse encontrando una orientación acíclica de *G* tal que la longitud del camino más largo de *0* a *N+1 (camino crítico)* sea minimizada.

El algoritmo de búsqueda tabú para el problema $O||C_{max}$ que se propone necesita una solución inicial, una estructura de vecinos, una lista tabú, un criterio de aspiración, y un criterio de parada. La experiencia computacional efectuada nos indica que en ocasiones cierta aleatorización puede ser ventajosa.

En los próximos apartados se describen como se seleccionan todas estas componentes en el algoritmo tabú propuesto.

Una solución inicial

En apartados anteriores se han descrito tres algoritmos determinísticos de planificación de listas (*LPT, SPT,* y *LIS)* para construir soluciones aproximadas al problema $O||C_{max}$. También se han establecido dos rutinas sencillas de mejora. En las primeras versiones de la heurística tabú la solución inicial que se tomaba era la mejor entre las tres soluciones encontradas por los algoritmos de planificación de listas más las rutinas de mejora. Sin embargo, se ha observado que, en ocasiones, comenzar con una solución inicial peor lleva a soluciones mejores. Por tanto se proponen dos algoritmos tabú con múltiples soluciones iniciales. El conjunto de soluciones iniciales sucesivas tiene nueve elementos caracterizados por las tres soluciones heurísticas *LPT, LIS,* y *SPT,* y los valores *250 n, 500 n* y *1000 n* del parámetro *VUELVE* que controla las vueltas atrás en la búsqueda para volver a empezar con el mejor grafo encontrado hasta ese momento. Este parámetro se describe más adelante.

Estructuras de vecinos

Se consideran dos tipos de transiciones que definen una estructura de vecinos de la solución actual *s*. El primer tipo de transición puede obtenerse invirtiendo el orden de proceso de dos operaciones consecutivas de un mismo trabajo en diferentes máquinas, ó de dos trabajos consecutivos en la misma

máquina. Mientras, el segundo tipo de transición puede obtenerse desplazando el procesamiento de una operación entre otras dos operaciones del mismo trabajo en diferentes máquinas, ó de diferentes trabajos en la misma máquina.

Más aún, se puede restringir nuestra atención a considerar transiciones que involucran operaciones de un camino crítico, es decir, aquellas que pertenecen a la secuencia de operaciones que determinan el valor de la función objetivo de la solución definida por el grafo actual. Se definen entonces, los siguientes conjuntos de vecinos de una solución factible *s*.

N1(s) obtenido al invertir el orden de proceso de dos operaciones críticas consecutivas del mismo trabajo, ó, alternativamente, de dos operaciones críticas consecutivas de diferentes trabajos en la misma máquina.

N2(s) obtenido al desplazar el procesamiento de una operación crítica entre otras dos operaciones críticas de un mismo trabajo en diferentes máquinas, ó, alternativamente, entre otras dos operaciones críticas de diferentes trabajos en la misma máquina.

Dada la solución actual *s*, para cada nueva solución posible $s' \in N(s)$ se calcula el valor exacto del camino de longitud máxima en *s'*. Esto se hace de la siguiente manera. Para cada operación *k*, sean r_k, t_k, r'_k, y t'_k el menor instante de comienzo posible de la operación *k* y la longitud máxima de los caminos de *k* a *N+1* en las planificaciones *s* y *s'*, respectivamente. Más aún, sea $Q = \{Q_1, ..., Q_q\}$ el conjunto de operaciones a permutarse para obtener la solución *s'* a partir de la *s*, es decir, en s' se tiene la cadena $Q_1 \rightarrow Q_2 \rightarrow ... \rightarrow Q_q$ y esta permutación es la única diferencia entre *s* y *s'*. Las operaciones del conjunto *Q* son operaciones del mismo trabajo o de la misma máquina. Se calcula el valor exacto de la longitud del camino crítico en *s'* extendiendo un procedimiento propuesto por Dell'Amico y Trubian (1993) para el problema job shop. Nótese que, en los problemas job shop, el orden de las operaciones de cada trabajo esta fijado, y se tiene que decidir solamente el orden de las operaciones en cada máquina. Mientras, en problemas open shop, no existe ningún orden fijado, y, entonces, se tienen que decidir ambos, el orden de operaciones en cada trabajo y en cada máquina. Por tanto, en el procedimiento propuesto por Dell'Amico y Trubian el conjunto *Q* se compone de diferentes trabajos de una misma máquina. Ahora, a diferencia de lo anterior, en problemas open shop, *Q* es una cadena de trabajos con una máquina común ó una cadena de máquinas con un trabajo común. Por tanto,

$$r'_k = max\{r_i + p_i, r'_{k-1} + p_{k-1}\}$$

$$t'_k = max\{t_j + p_j, t'_{k+1} + p_{k+1}\}$$

donde *i,j* el predecesor y sucesor de *k* en *s* y *s'* fuera de *Q*; y *k-1, k+1* los correspondientes en *s'* dentro de *Q*.

Exploración eficiente de vecinos

La estructura de vecinos *N1* considera arcos críticos. La convergencia de esta estructura a la solución óptima esta garantizada por consideraciones análogas a las ya establecidas para el problema job shop. La exploración de estos vecinos puede realizarse más rápidamente considerando la idea de *bloque.* Esta idea nos permite establecer una *regla de dominancia* en la exploración de los vecinos.

Si *s* es una solución factible para el problema open shop, y si $\pi_1, ..., \pi_r$ es una secuencia de operaciones que constituyen un camino crítico en *s* de acuerdo con el modelo de grafo disyuntivo adoptado, el valor de la función objetivo será $z(s) = C_{\pi r}$, es decir, el tiempo de completación de la operación π_r. La cadena $\pi_1, ..., \pi_r$ puede cubrirse por bloques. Se define un bloque como el conjunto

$$B_{hk} = \{\pi_h, \pi_{h+1},..., \pi_k\}$$

de operaciones críticas consecutivas en un mismo trabajo o en una misma máquina sin tiempo ocioso entre ellas, es decir, $C_{\pi j} + p_{\pi j+1} = C_{\pi j+1}$ $(j = h, ..., k-1)$.

El camino crítico $\pi_1, ..., \pi_r$ se recubre por los bloques $B_{1h1}, B_{h1h2}, ..., B_{htr}$, donde $1 \leq h_1 \leq h_2 \leq ... \leq h_t \leq r$. Bloques consecutivos B_{hk}, B_{kl} verifican que, o bien el conjunto de operaciones de B_{hk} están en la misma máquina y el conjunto de operaciones de B_{kl} están en el mismo trabajo, o bien B_{hk} es un bloque de operaciones con un trabajo común y B_{kl} con una máquina común. La operación común π_k pertenece tanto al trabajo como a la máquina que entran en consideración.

Dado el bloque B_{hk}, invertir un arco interno (π_x, π_y) del bloque no hace cambiar el tiempo de completación de las otras operaciones del bloque, en particular, el tiempo de completación de la última operación del bloque no cambia. Esta propiedad nos proporciona una regla de dominancia para explorar los vecinos.

Regla de Dominancia 1:

Invertir un arco interno de un bloque B_{hk} de operaciones críticas no proporciona una reducción del tiempo de completación de la solución actual.

Por tanto, estos arcos no se consideran.

Lista y estrategias tabú

Las principales componentes de un algoritmo de búsqueda tabú son las *estructuras de memoria* para identificar la evolución de la búsqueda, y las *estrategias* , para emplear la información memorizada del mejor modo posible. En los próximos párrafos se explica que información se ha memorizado y que estrategias se han seguido para diseñar un algoritmo tabú para el problema $O||C_{max}$.

La estructura de memoria fundamental es la lista tabú. A cada movimiento que transforma la solución s en la solución $s' \in N(s)$ se le asocia un conjunto de *atributos*. Se memorizan en la lista tabú los atributos de los movimientos realizados y, en cada iteración, se selecciona el mejor movimiento del conjunto de candidatos cuyos atributos no pertenecen a la lista tabú, ó si pertenecen a dicha lista, pero el movimiento satisface un criterio de aspiración. La lista tabú tiene una dimensión finita y se gestiona con una estrategia *FIFO*.

Cuando se considera la estructura de vecinos *N1(s)*, un movimento consiste en invertir un arco que involucra a dos operaciones críticas que no son internas a ningún bloque del correspondiente recubrimiento de bloques del camino crítico, y el arco inverso se memoriza como tabú. Con la estructura de vecinos *N2(s)*, una operación se inserta entre otras dos, y la inserción opuesta se memoriza como prohibida.

En el apéndice B se recoge la experiencia computacional realizada para la búsqueda tabú. En la implementación realizada, la longitud l de la lista tabú se establece inicialmente a *2* y varía dinámicamente en el intervalo *[3, 3n/4]*. Cada *25 n* movimientos la longitud se controla de acuerdo con la siguiente regla: si z^* se ha actualizado durante los últimos *500* movimientos (donde z^* es el valor de la función objetivo de la mejor solución encontrada), entonces $l = min \{3n/4 \ , \ l+2\}$; en otro caso $l = max \{3, l-2\}$.

Dada la solución actual s, para cada solución $s' \in N(s)$ el valor de la función objetivo se evalúa de acuerdo con las técnicas descritas en apartados anteriores. Si $\bar{s}$ denota la solución candidata con menor valor $z(\bar{s})$ se tiene lo siguiente. Si $z(\bar{s}) \geq z^*$ entonces se selecciona la solución no tabú con menor valor de la función objetivo para la próxima iteración. En otro caso, un criterio de aspiración se aplica y se selecciona $\bar{s}$ aunque sea tabú.

La experiencia computacional realizada muestra que la implementación de componentes aleatorias es útil. Hay dos perturbaciones aleatorias cuando se trabaja con la estructura de vecinos *N1*. Ambas van relacionadas con el parámetro

$\gamma_{max}=20n$. Inicialmente, estas dos perturbaciones están desactivadas. Después de γ_{max} iteraciones, la perturbación *PT1* se activa. La perturbación *PT2* se activa después de $\gamma_{max} + n$ iteraciones. A partir de ahí, ambas perturbaciones se activan y desactivan dinámicamente.

Cuando *PT1* está activa, *N1* selecciona el *k*-ésimo arco del actual camino crítico independientemente de cual sea su estimación para la función de costo ó su estado tabú, y *PT1* se desactiva. El valor *k* se toma de la *U[1,n]*. Después de γ_{max} iteraciones desde la última desconexión de *PT1*, ó desde la última actualización de z^*, ó desde la última aplicación de *PT2,* la perturbación *PT1* se conecta de nuevo y *k* se vuelve a aleatorizar.

Cuando *PT2* está activa, *N1* elige como mejor arco candidato al penúltimo de los arcos que han sido previamente considerados como mejor arco candidato, y realiza el movimiento con ese arco. Después de $\gamma_{max} + n$ iteraciones desde la última actuación de *PT2*, ó desde la última actualización de z^*, ó desde la última aplicación de *PT1,* la perturbación *PT2* se conecta de nuevo.

Cada *VUELVE* movimientos sin ninguna actualización de z^*, el algoritmo realiza una vuelta atrás retomando la mejor solución encontrada hasta ese momento y recomenzando la búsqueda a partir de su grafo asociado con un arco seleccionado aleatoriamente. Este tipo de *"volver a empezar"* desde la mejor solución encontrada puede originar ciclos. Para evitarlos, el algoritmo mantiene una lista con los arcos seleccionados en estos "volver a empezar". Cuando z^* se actualiza, la lista se vacía. Cada vez que se hace un "volver a empezar" se aplica la primera de las rutinas de mejora comentadas anteriormente.

VUELVE se fija inicialmente a *250 n*. Si, comenzando con las soluciones iniciales *LPT, LIS, SPT* la búsqueda finaliza sin alcanzar la cota inferior, entonces *VUELVE* se pone sucesivamente a *500 n* y *1000 n*. Se llega al final de cada simple ejecución de la búsqueda cuando se alcanza la cota inferior, ó cuando se excede un límite temporal. Este límite temporal se fija a *600* segundos para problemas de gran tamaño.

Finalmente, recordemos que en el Apéndice B se recoge la experiencia computacional realizada de los algoritmos propuestos para resolver el problema $O||C_{max}$.

Capítulo IV

Capítulo IV: Planificación bicriterio

1. Introducción

Hemos visto en capítulos anteriores como los modelos de planificación y secuenciación permiten representar y estudiar muchos problemas que surgen en la vida real. Ahora bien, cuánto mayor sea la proximidad del modelo al problema, más precisa será la información suministrada al decisor con objeto de ayudarle en su decisión.

En este sentido, en muchas ocasiones, los modelos bicriterio pueden adaptarse mejor a los problemas reales que los modelos unicriterio y, por tanto, deben ser considerados.

Hasta ahora se han analizado un amplio número de problemas de planificación unicriterio catalogados como difíciles ó NP-duros. Cuando varios criterios entran en consideración la complejidad de estos problemas se ve generalmente aumentada. Esto es especialmente notorio si los criterios son conflictivos entre sí, es decir, los óptimos en los diversos criterios difieren bastante. En estas situaciones, y particularmente cuando se consideran funciones objetivo lineales, es necesario determinar los puntos eficientes u óptimos de Pareto. La detección de estos puntos es, en muchos casos, un problema NP-duro o una cuestión abierta, aunque, en otras ocasiones pueden detectarse polinomialmente.

2. Planteamiento de diversos problemas y complejidad computacional

El estudio de modelos de planificación bicriterio es aún un tema incipiente dentro de la Planificación y Secuenciación de Tareas. Algunas cosas se han hecho ya, pero fundamentalmente en problemas bicriterio sobre una máquina.

Como se ha comentado en el capítulo I, muchos problemas de planificación bicriterio (y, en general, multicriterio) pueden resolverse mediante métodos de Programación Bicriterio (Multicriterio), es decir, mediante la aplicación de sus métodos para detectar soluciones eficientes satisfactorias para el decisor.

En la bibliografía actual existen diversas clasificaciones de los métodos de Programación Multicriterio y, en particular, de la Programación Bicriterio. Así, se tienen los *métodos de las funciones de utilidad* que aparecen cuando la información sobre la estructura de preferencias del decisor es tal que puede expresarse en forma precisa mediante una única función convirtiéndose el problema en un problema unicriterio.

Los *métodos interactivos* surgen cuando la información que se tiene sobre las preferencias del decisor no es suficiente para emplear un método de funciones de utilidad, y entonces tiene lugar un proceso iterativo decisor-analista por el que el decisor va suministrando progresivamente información al analista. Este último utiliza la información para mejorar el modelo y presentarle al decisor sucesivos conjuntos de soluciones eficientes. El proceso finaliza cuando se encuentra alguna solución que satisfaga, dentro de lo posible, las necesidades del decisor.

Los *métodos generadores de soluciones eficientes* son aplicados para generar soluciones eficientes en ausencia de cualquier información sobre las preferencias del decisor.

Otras clasificaciones hablan de optimización jerárquica y optimización simultánea. La *optimización jerárquica* consiste en especificar niveles de satisfacción para los criterios. Se seleccionan entonces soluciones que satisfagan dicho nivel en el criterio más importante y que sea próxima a los niveles de satisfacción en los otros criterios.

Se habla de *optimización simultánea* cuando el problema se convierte en un problema unicriterio mediante funciones de utilidad lineales o generales. También cuando está conversión no es posible y se tienen que generar las soluciones eficientes de modo independiente porque no se dispone de información a priori de las preferencias del decisor.

Los problemas de planificación bicriterio admiten una formulación específica. Un problema de planificación bicriterio determinístico en su formulación más general puede establecerse describiendo las características de las *m* máquinas que deben procesar un conjunto de *n* trabajos, los parámetros de éstos, y la descripción de los criterios que intervienen. Es decir, siguiendo una generalización natural de la formulación triparamétrica $\alpha|\beta|\gamma$ seguida por Graham y otros (1979) para problemas unicriterio, representamos un problema de planificación bicriterio determinístico en la forma $\alpha|\beta|(\gamma^1, \gamma^2)$ donde los parámetros α, β, γ^k representan las características de las m máquinas, de los n trabajos, y del criterio *k*-ésimo respectivamente con *k = 1,2*. Estas características vienen parametrizadas de la misma forma descrita en capítulos anteriores para el caso unicriterio.

Sea Ω el conjunto de planificaciones factibles para el problema de planificación bicriterio determinístico $\alpha|\beta|(\gamma^1,\gamma^2)$ en cuestión, y sea σ un elemento de Ω. En la planificación σ el tiempo de completación del trabajo J_j es $C_j(\sigma)$ con $j = 1,...,n$. Los criterios de actuación suelen ser funciones vectoriales en los tiempos de completación, $f^k:\Omega\to\Re$, con $f^k(\sigma)=f^k(C_1(\sigma),...,C_n(\sigma))$, $k = 1,2$. Dichas funciones se denominan regulares si son crecientes en los tiempos de completación. En general, los criterios considerados pueden ser funciones regulares o no.

Algunos conceptos a los que haremos referencia son los siguientes:

Definición 1.

Una solución factible $\sigma \in \Omega$ es eficiente u óptimo de Pareto respecto a los dos criterios de actuación f^1, f^2 si no existe planificación factible $\pi \in \Omega$ con $f^i(\pi) \le f^i(\sigma), \forall i = 1,2,$ y al menos una de las desigualdades estricta.

Definición 2.

La región eficiente, o conjunto de puntos eficientes, es el subconjunto de $f(\Omega)$, con $f = (f^1, f^2)$ que es imagen del conjunto Ω_E de planificaciones eficientes.

$$S = \{ y=(a_1,a_2) \in f(\Omega) \text{ tales que } \exists\, \sigma \in \Omega_E \text{ con } f(\sigma) = (f^1(\sigma), f^2(\sigma)) = (a_1,a_2) = y \}$$

La frontera eficiente es la frontera de S.

Definición 3.

Una planificación factible $\sigma \in \Omega$ es extrema con respecto a f^1, f^2 si corresponde a un vértice de la frontera eficiente.

Definición 4.

Dos planificaciones $\sigma, \pi \in \Omega$ son *eficientemente equivalentes* , $\sigma \sim_E \pi$, si ambas son eficientes y los puntos $(f^1(\sigma), f^2(\sigma))$ y $(f^1(\pi), f^2(\pi))$ coinciden.

Obviamente, " $\sim_E$ " es una relación de equivalencia en el conjunto de planificaciones eficientes Ω_E , con $\Omega_E \subseteq \Omega$.

Definición 5.

Dos planificaciones $\sigma, \pi \in \Omega$ son *extremamente equivalentes* , $\sigma \sim_{EX} \pi$, si ambas son eficientes extremas y los puntos $(f^1(\sigma), f^2(\sigma))$ y $(f^1(\pi), f^2(\pi))$ coinciden.

Obviamente, " $\sim_{EX}$ " es una relación de equivalencia en el conjunto de planificaciones eficientes extremas Ω_{EX}, con $\Omega_{EX} \subseteq \Omega_E \subseteq \Omega$.

Obviamente, y de modo natural, las definiciones anteriores pueden extenderse a K criterios. En el caso que nos ocupa, caso bicriterio, tomamos el problema $\alpha|\beta|(\gamma^1, \gamma^2)$ donde las medidas de actuación f^1, f^2 asociadas a los criterios las denotaremos por f y g respectivamente. Entenderemos por optimizar la f el hecho de resolver el problema unicriterio $\alpha|\beta|\gamma^1$, y, por optimizar la g el hecho de resolver el problema unicriterio $\alpha|\beta|\gamma^2$. Sin pérdida de generalidad podemos suponer que la optimización es una minimización.

Comenzaremos haciendo referencia al caso de independencia entre los criterios, es decir, la información disponible sobre las preferencias del decisor no nos permite componer los criterios en una única función. Posteriormente veremos problemas en los que dicha composición es posible.

2.1. Minimización simultánea: independencia

2.1.1. Criterios (U_{max}, T_{max})

Cuando se consideran estos criterios simultáneamente el problema se reduce al problema unicriterio con el único criterio T_{max} , es decir, basta con encontrar la planificación que minimiza T_{max}. En este caso cualquier óptimo de Pareto es óptimo del problema.

El razonamiento se ilustra en la figura 1 donde se aprecian los posibles pares de valores para los objetivos.

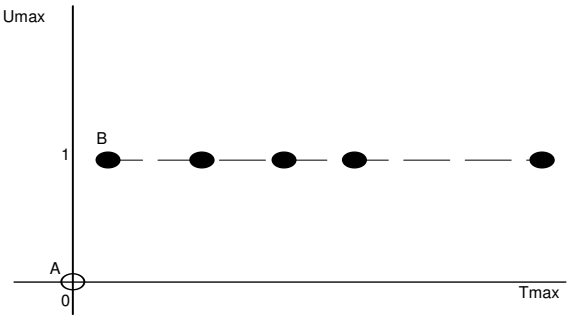


Figura 1. Posibles valores para (Umax, Tmax).
Si A es factible, es óptimo. Si no, B es óptimo.

Esto es debido a que, para cualquier planificación σ se verifican de manera obvia las relaciones:

$$Tmax(\sigma) > 0 \Leftrightarrow Umax(\sigma) = 1$$

$$Tmax(\sigma) = 0 \Leftrightarrow Umax(\sigma) = 0.$$

2.1.2. Criterios (L_{max} , T_{max})

En este caso el problema se reduce al problema unicriterio con el único criterio L_{max}, es decir, basta con encontrar la planificación que minimiza L_{max}. En este caso, al igual que en el caso anterior, cualquier óptimo de Pareto es óptimo del problema. Esto es así en virtud de las relaciones

$$Lmax(\sigma) \geq 0 \Leftrightarrow Tmax(\sigma) = Lmax(\sigma)$$

$$Lmax(\sigma) \leq 0 \Leftrightarrow Tmax(\sigma) = 0.$$

Los posibles pares de valores para los criterios se muestran en la figura 2:

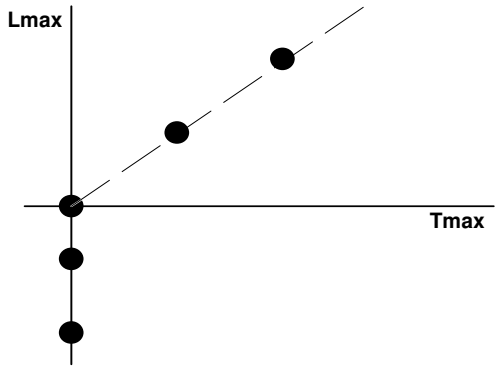


Figura 2. Posibles valores para los criterios (Lmax, Tmax)

2.1.3. Criterios (U_{max}, L_{max})

Como en los casos anteriores, en este caso el problema se reduce a un problema unicriterio. Es suficiente con minimizar L_{max}. Esto es así, como consecuencia directa de las relaciones ya expresadas entre los criterios. También aquí se verifica que cualquier óptimo de Pareto es óptimo del problema.

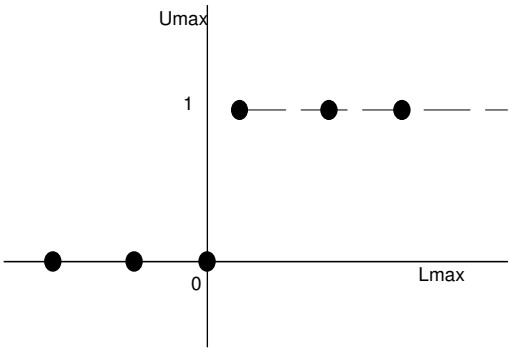


Figura 3. Posibles valores de los objetivos (Umax. Lmax)

Parece observarse cierta regularidad al combinar criterios conectados por un camino en el grafo de reducibilidad establecido para problemas unicriterio en los capítulos anteriores. No obstante, esta regularidad debe comprobarse en cada caso antes de generalizarse.

2.1.4. Criterios (C_{max}, U_{max})

En este caso, la resolución de un problema unicriterio da bastante información sobre el problema bicriterio.

Nótese que si consideramos una única máquina y fechas de disponibilidad r_j iguales a cero, el valor de C_{max} es constante, y, por tanto, basta con resolver el problema unicriterio en U_{max}. Este problema con una sola máquina puede resolverse polinomialmente con cualquier algoritmo que minimice el número de trabajos tardíos, como el de Moore y Hodgson (1968).

Si se cuenta en el modelo con la presencia de varias máquinas. Sea σ^* una planificación óptima para el problema que tiene por único criterio a C_{max}.

- Si $U_{max}\ (\sigma^*) = 0$, entonces σ^* es también óptima para el problema bicriterio.

- Si $U_{max}\ (\sigma^*) = 1$, entonces debe calcularse σ_0 la solución óptima del problema

$$min\ C_{max}(\sigma)$$

s.a.:

$$U_{max}(\sigma)$$

La gráfica de las soluciones factibles para este problema es la de la figura siguiente:

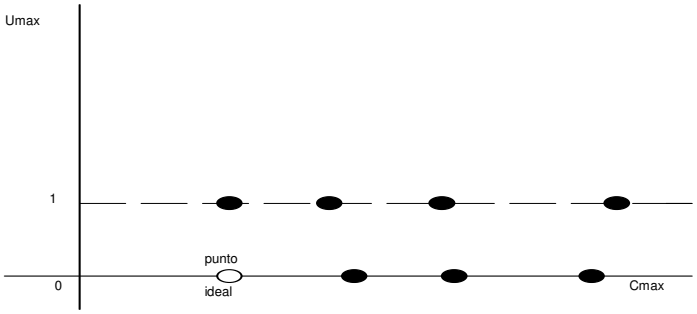


Figura 4. Posibles valores para los criterios (Cmax, Umax).

2.1.5. Criterios (C_{max}, L_{max})

En los problemas en los que interviene una sola máquina, y, suponiendo que los trabajos están todos disponibles desde el instante inicial, es decir, las fechas de disponibilidad r_j, son todas iguales a cero, el valor C_{max} es constante, y, por tanto, el problema se reduce al problema unicriterio con el criterio L_{max}, criterio que es optimizado en tiempo polinomial por la secuencia *EDD*, aquella que consiste en ordenar los trabajos en orden creciente de sus fechas límite.

Si intervienen dos o más máquinas, obtenemos problemas más complejos y nos vemos en la necesidad de determinar los óptimos de Pareto.

La gráfica bidimensional de posibles valores para estos criterios sería la de la figura 5:

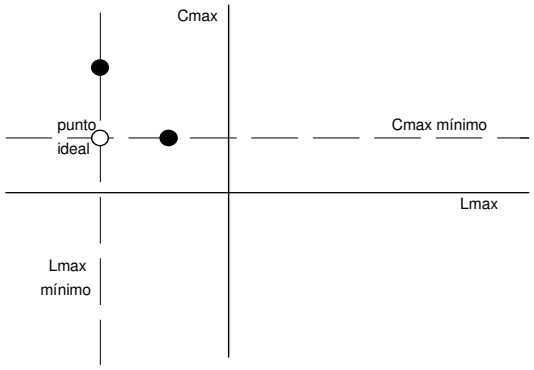


Figura 5. Posibles valores en los criterios (Cmax,Lmax)

2.1.6. Criterios (Cmax, Tmax)

En este caso son también aplicables las consideraciones hechas para el caso anterior, incluso la secuencia *EDD* también minimiza T_{max} cuando se considera una única máquina. La gráfica de posibles valores para los criterios tiene un aspecto similar al anterior, salvo que T_{max} no puede tomar valores negativos.

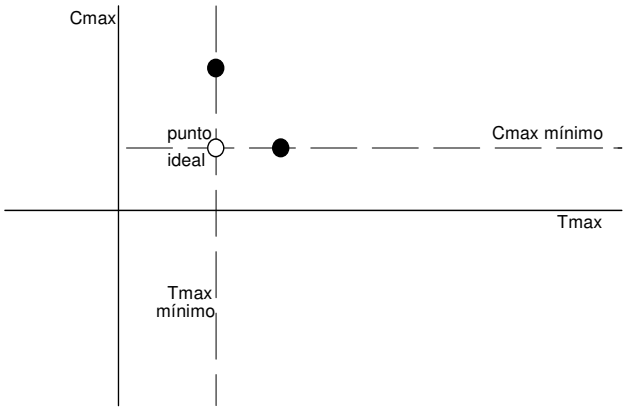


Figura 6. Posibles valores en los criterios (Cmax,Tmax)

Veamos ahora el caso en el que existe una relación funcional entre los criterios, es decir, los criterios se combinan en una única función objetivo que se pretende minimizar.

2.2. Minimización simultánea: función objetivo compuesta

2.2.1. Criterios (P_{max}, L_{max})

Hoogeveen (1992a) considera el problema bicriterio sobre una máquina donde los criterios que intervienen son la puntualidad y la demora máximas. El conjunto $J = \{ J_1, ..., J_n\}$ esta formado por los trabajos a secuenciar, donde cada trabajo J_j tiene asociados los datos p_j, que representa su tiempo de completación, d_j su fecha límite de completación, y s_j el instante de comienzo deseado. Se consideran las variables S_j, instante real de comienzo de la ejecución, $P_j = s_j - S_j$ puntualidad en el comienzo de la ejecución y $L_j = C_j - d_j$ la demora en la completación de la actividad. Los criterios elegidos son entonces

$$P_{max} = max_{j=1,...,n}\left\{P_j\right\}$$

$$L_{max} = max_{j=1,...,n}\left\{L_j\right\}$$

Nótese que L_{max} es una medida de actuación regular, es decir, creciente en los tiempos de completación, mientras que P_{max} no lo es. De hecho, cuando $s_j = d_j - p_j \ \forall j$ la puntualidad máxima P_{max} y la anticipación máxima E_{max} coinciden. (Recuérdese que $E_j = max\left\{0, d_j - C_j\right\}$). Hoogeveen muestra que el problema de

minimización simultánea con una función objetivo compuesta general $1|nmit|F(P_{max},L_{max})$ es polinomial cuando $s_j \in [d_j - p_j , d_j] \ \forall j$, donde el parámetro *nmit* denota el hecho de que no se admite tiempo ocioso en la máquina.

2.2.2. Criterios (ΣC_j, f_{max}) con f_{max} regular, y criterios (ΣC_j, E_{max})

Hoogeveen y van de Velde (1992a) consideran los problemas bicriterio sobre una máquina donde el criterio que acompaña a la suma total de los tiempos de completación de todos los trabajos es una función regular $f_{max} = max_{j=1,..,n}\{f_j(C_j)\}$ con $f_j(C_j)$ creciente $\forall j$; o bien, la función no regular de la anticipación máxima E_{max} .

El problema $1||F(\Sigma C_j,E_{max})$ es polinomialmente resoluble con una complejidad $O(n^3 \ min \ \{n, \ log \ n \ + \ log \ p_{max}\})$ con $p_{max} = max_{1\le j\le n}\{p_j\}$. En el caso particular $f_{max} = L_{max}$, la complejidad se reduce a $O(n^3)$. El número de óptimos de Pareto para $(\Sigma C_j,f_{max})$ esta acotado por $\frac{n(n-1)}{2}+1$ y esta cota es asintótica, es decir, es posible encontrar una sucesión de entradas al problema cuya correspondiente sucesión de cardinales del conjunto de óptimos de Pareto converja a esa cota.

Sin embargo, el problema $1|pmtn,nmit|F(\Sigma C_j,E_{max})$ es pseudopolinomialmente resoluble en tiempo $O(n\Sigma p_j)$, y es NP-duro en sentido ordinario, pero no en sentido fuerte. Su relajación $1|pmtn|F(\Sigma C_j,E_{max})$ si es un problema NP-duro en sentido fuerte.

Cuando se permiten interrupciones, el número de planificaciones extremas con respecto a E_{max} y ΣC_j esta acotado por $\frac{n(n-1)}{2}+1$.

Casos particulares polinomialmente resolubles del problema son:

- $1|pmtn,nmit|\alpha_1\Sigma C_j+\alpha_2 E_{max}$ en tiempo $O(n^4)$ si $\alpha_1 \ge \alpha_2$.
- $1||\alpha_1\Sigma C_j+\alpha_2 E_{max}$ en tiempo $O(n^4)$.
- $1|pmtn|\alpha_1\Sigma C_j+\alpha_2 E_{max}$ también en tiempo $O(n^4)$.

2.2.3. Problemas justo a tiempo

Un problema que surge directamente del ámbito industrial es el problema *"just-in-time manufacturing"*. La metodología *"justo a tiempo"* consiste en realizar una planificación del proceso de fabricación de modo que, la completación de la fabricación de los productos tenga lugar en instantes tales que: por un lado, dichos instantes sean anteriores a las fechas límite de completación de la fabricación de los productos; y, por otro, no sean demasiado anteriores. Se trata pues de minimizar dos tipos de costos inherentes al problema: el *costo de penalización* por no completar la producción antes de las fechas límite, y el *costo de almacenamiento* de aquellos productos finalizados antes de sus fechas límite. Este problema puede modelizarse como un problema bicriterio sobre una única máquina donde, desde el punto de vista de la optimización simultánea, la función a minimizar es

$$f(\sigma) = \Sigma(\alpha_j E_j + \beta_j T_j)$$

donde $\Sigma\alpha_j E_j$ es el costo de almacenamiento de los productos cuando su fabricación se completa *antes de* sus fechas límite, y $\Sigma\beta_j T_j$ la penalización en la que se incurre cuando la fabricación de los productos se finaliza *después de* su fecha límite.

3. Algoritmos y Métodos de Resolución

3.1. Obtención de puntos eficientes para problemas bicriterio

Se proponen a continuación sendos algoritmos para calcular los puntos eficientes y los puntos eficientes extremos que están sobre la frontera de la envoltura convexa de la región factible. Ambos algoritmos difieren principalmente en el paso 3, pues el segundo algoritmo necesita calcular otra pendiente para seguir el procedimiento de búsqueda. En la descripción de los algoritmos se utiliza la notación establecida al principio del epígrafe 2 de este mismo capítulo.

Sea Ω el conjunto de planificaciones σ posibles, sea H la envoltura convexa del conjunto de puntos $(f(\sigma),g(\sigma))$ donde $\sigma \in \Omega$, y sea $Fr(H)$ la frontera de H.

Algoritmo para obtener los puntos eficientes o de Pareto en Fr(H).
(Algoritmo E)

Paso 1:

- Sea $\sigma^1 \in \Omega$ la planificación que optimiza la *f*, y sea $\sigma^2 \in \Omega$ la planificación que optimiza la *g*. Si hay varias σ^1 elegir una con valor $g(\sigma^1)$ menor. Si hay varias σ^2 elegir una con valor $f(\sigma^2)$ menor.
- Sea $\boldsymbol{E}$ el conjunto de planificaciones eficientes en *Fr(H)*. Inicialmente $\boldsymbol{E}$ es vacío.
- Almacenar σ^1 y σ^2 en $\boldsymbol{E}$.
- Sean los puntos $(f(\sigma^1), g(\sigma^1))$, y $(f(\sigma^2), g(\sigma^2))$ los valores correspondientes a las dos planificaciones.
- Poner $A = \sigma^1$, $B = \sigma^2$, e $i = 0$.

Paso 2:

- Calcular la pendiente *m* y el parámetro *b* de la forma siguiente:

$$m = \frac{g(B) - g(A)}{f(B) - f(A)}, \quad b = \frac{m}{m-1}.$$

- Calcular la planificación $\sigma \in \Omega$ que sea mínima respecto al valor *b,* es decir, aquella que minimiza

$$bf(\sigma) + (1-b)g(\sigma).$$

Si hay más de una planificación mínima con respecto a *b,* elegir aquella distinta de *A* y con valor $f(\sigma)$ más pequeño.

- Sea el punto $(f(\sigma), g(\sigma))$.

- Si la planificación σ coincide con *B*, ir al paso 4.
- En otro caso, ir al paso 3.

Paso 3:

- Almacenar σ en $\boldsymbol{E}$. Colocar $i = i + 1$, $C(i) = A$, $D(i) = \sigma$.
- Poner $A = \sigma$ y volver al paso 2.

Paso 4:

- Si $i \neq 0$, poner $A = C(i)$, $B = D(i)$, $i = i - 1$ y volver al paso 2.
- Si $i = 0$, *Parar.* $\boldsymbol{E}$ da el conjunto de planificaciones eficientes en *Fr(H)*.

Algoritmo para obtener los puntos eficientes extremos en Fr(H). (Algoritmo EX)

Paso 1:

- Sea $\sigma^1 \in \Omega$ la planificación que optimiza la *f*, y sea $\sigma^2 \in \Omega$ la planificación que optimiza la *g*. Si hay varias σ^1 elegir una con valor $g(\sigma^1)$ menor. Si hay varias σ^2 elegir una con valor $f(\sigma^2)$ menor.

- Sea ***EX*** el conjunto de planificaciones eficientes extremas en *Fr(H)*. Inicialmente ***EX*** es vacío.
- Almacenar σ^1 y σ^2 en ***EX***.
- Sean los puntos $(f(\sigma^1), g(\sigma^1))$, y $(f(\sigma^2), g(\sigma^2))$ los valores correspondientes a las dos planificaciones.
- Poner $A = \sigma^1$, $B = \sigma^2$, e $i = 0$.

Paso 2:
- Calcular la pendiente m_1 y el parámetro b de la forma siguiente:

$$m_1 = \frac{g(B) - g(A)}{f(B) - f(A)}, \quad b = \frac{m_1}{m_1 - 1}.$$

- Calcular la planificación $\sigma \in \Omega$ que sea mínima respecto al valor b, es decir, aquella que minimiza $bf(\sigma) + (1-b)g(\sigma)$. Si hay más de una planificación mínima con respecto a b, elegir aquella con valor $f(\sigma)$ más pequeño.
- Sea el punto $(f(\sigma), g(\sigma))$.

- Si la planificación σ coincide con A ó con B, ir al paso 4.
- En otro caso, ir al paso 3.

Paso 3:
- Calcular la pendiente m_2 entre los puntos de las planificaciones B y σ, es decir:

$$m_2 = \frac{g(B) - g(\sigma)}{f(B) - f(\sigma)},$$

- Si $m_2 \neq m_1$, poner σ en ***EX***. Colocar $i = i + 1$, $C(i) = A$, $D(i) = \sigma$. Poner $A = \sigma$, y volver al paso 2.
- Si $m_2 = m_1$ ir al paso 4.

Paso 4:
- Si $i \neq 0$, poner $A = C(i)$, $B = D(i)$, $i = i - 1$ e ir al paso 2.
- Si $i = 0$, *Parar*. ***EX*** da el conjunto de puntos eficientes extremos en *Fr(H)*.

Estudiemos ahora la convergencia de estos algoritmos.

Convergencia de los algoritmos

Los teoremas siguientes garantizan la convergencia de estos algoritmos.

Teorema 1.

El algoritmo *E* converge, es decir, genera todos los puntos eficientes respecto a (*f, g*) que estén en *Fr(H)*.

Demostración:

Tendremos que demostrar:

i) *A* es eficiente, y, por simetría, *B* también lo es.
ii) dado el par de eficientes *A, B,* la planificación σ_b que minimiza $bf(\sigma)+(1-b)g(\sigma)$ con $f(\sigma)$ más pequeño, y distinta de *A*, también lo es.
iii) todos los pares posiblemente generadores de planificaciones σ_b eficientes son implícitamente explorados.

i) y iii) son evidentes por construcción. Para ii), sea $m=\frac{g(B)-g(A)}{f(B)-f(A)}$ la pendiente de la recta que une los puntos asociados a las planificaciones *A* y *B*. Sea $b=\frac{m}{m-1}$, y sea σ_b la correspondiente planificación encontrada en el paso 2. Por la elección de *A* y *B* se tiene que $f(A) < f(\sigma_b) \leq f(B)$, y, $g(B) \leq g(\sigma_b) < g(A)$. Por tanto, $-\infty < m < 0$, $1 > b > 0$, y el punto $(f(\sigma_b), g(\sigma_b))$ está en el triángulo definido por los puntos asociados a las planificaciones *A* y *B*, puntos *(f(A), g(A))* y *(f(B), g(B))*, y el punto *(f(A), g(B))*. Sea $h(x,y) = bx + (1-b)y$, como $1>b>0$, entonces $h(x,y)$ es creciente en ambos argumentos. Consecuentemente, la minimalidad de σ_b garantiza que su punto asociado sea eficiente. □

Teorema 2.

El algoritmo *EX* converge, es decir, genera todos los puntos eficientes extremos respecto a (*f, g*) que estén en *Fr(H)*.

Demostración:

Tendremos que demostrar:

i) *A* es eficiente extrema, y, por simetría, *B* también lo es.
ii) para cualquier par de eficientes extremas *A, B*, el algoritmo *EX* encuentra todos los puntos eficientes extremos que existen en el triángulo definido por los puntos asociados a *A* y *B*, puntos *(f(A), g(A))* y *(f(B), g(B))*, y el punto *(f(A), g(B))*.

i) es evidente por construcción. Para ii), sean *A, B* dos planificaciones eficientes extremas, y sea σ_b la planificación asociada de acuerdo con el paso 2. Por el *Teorema 1,* σ_b es eficiente. Sea m_2 la pendiente de la recta que une los puntos asociados a σ_b y *B*. Si $m_2 = m_1$ los puntos asociados a *A,* σ_b , y *B* están alineados quedando σ_b en medio y, por tanto, σ_b no es extrema. Si $m_2 \neq m_1$, entonces σ_b es extrema pues, si no lo fuese, su punto asociado sería combinación lineal convexa de dos puntos asociados a sendas planificaciones eficientes extremas σ_{b1} , σ_{b2} . Al menos uno de estos puntos estaría en el mismo triángulo que el punto asociado a σ_b, lo que contradiría la minimalidad de σ_b con respecto a $bf(\sigma)+(1-b)g(\sigma)$ y, en segundo término, su minimalidad con respecto a $f(\sigma)$.

Nótese que las relaciones "$\sim_E$", en el conjunto de planificaciones eficientes, $\Omega_E \subseteq \Omega$ y "$\sim_{EX}$" en el conjunto de planificaciones extremas $\Omega_{EX} \subseteq \Omega_E \subseteq \Omega$, son relaciones de equivalencia en sus respectivos conjuntos. Todas las planificaciones de la misma clase de equivalencia tienen asociado el mismo punto eficiente (eficiente extremo) $(f(\sigma),g(\sigma))$. En este sentido, los algoritmos *E*, y *EX*, generan para cada clase un único punto eficiente, y eficiente extremo, es decir, un único representante de cada clase de equivalencia.

Complejidad Computacional

La complejidad de los algoritmos propuestos depende del número *N* de puntos eficientes en *Fr(H)*, y del número *M* de puntos eficientes extremos en *Fr(H)* respecto al problema bicriterio (*f, g*), y también, de la complejidad *O(f), O(g), O(bf + (1-b)g)* relativas a los problemas unicriterio asociados a *f, g,* y *b f* + (1 - *b*) *g* respectivamente.

Características computacionales del Algoritmo E

a) resuelve el problema *f* una sola vez, y el problema *g* una sola vez.
b) resuelve el problema *bf* + (1-*b*)*g* tantas veces como puntos genera. El número de coincidencias $\sigma = B$ es de orden *N*.
c) el número de cálculos de *m* y *b* esta acotado por *2N*.

Por tanto, la complejidad del algoritmo *E* esta acotada superiormente por

$$O(f) + O(g) + N\,O(bf + (1-b)g) + 2N$$

Así, si los problemas unicriterio asociados son polinomiales, y N está acotado polinomialmente, entonces el algoritmo E es polinomial.

Características computacionales del Algoritmo EX

a) resuelve el problema f una sola vez, y el problema g una sola vez.
b) resuelve el problema $bf + (1-b)g$ tantas veces como puntos genera. El número de coincidencias $\sigma = A$, ó $\sigma = B$, ó σ distinta de A y B pero $m_2 = m_1$, es de orden M.
c) el número de cálculos de m_1 , b, y m_2 está acotado por $3M$.

Por tanto, la complejidad del algoritmo EX esta acotada superiormente por

$$O(f) + O(g) + M\,O(bf + (1-b)g) + 3M$$

Nótese que generalmente M es menor que N con lo que el algoritmo EX será usualmente más rápido que el algoritmo E.

Así, si los problemas unicriterio asociados son polinomiales, y M está acotado polinomialmente, entonces el algoritmo EX es polinomial.

Los algoritmos propuestos consideran el problema de caracterizar un amplio número de puntos eficientes y puntos eficientes extremos asociados a cualquier problema de planificación bicriterio. Dichos algoritmos determinan las planificaciones eficientes y las planificaciones eficientes extremas que estén sobre la frontera de la envoltura convexa de la región factible; y, no aportan mayor complejidad que la necesaria para resolver los problemas unicriterio asociados.

Como se ha indicado más arriba, puede acontecer que varias planificaciones eficientes tengan asociado el mismo punto eficiente, y que varias planificaciones eficientes extremas tengan asociado el mismo punto eficiente extremo. En estos casos, los algoritmos propuestos generan una única planificación asociada a cada punto generado, es decir, un único representante de cada clase de equivalencia.

Si se desean obtener todas las planificaciones de una determinada clase de equivalencia, habría que considerar el problema bicriterio concreto sobre el que se estén aplicando los algoritmos, y, tomando como referencia los valores de los objetivos y el representante de la clase, detectar todos los demás miembros de dicha clase. En el apéndice C se recoge una experiencia computacional realizada para estudiar estos algoritmos.

Estudiemos ahora los criterios *(P_{max},L_{max})*.

3.2. Problemas Bicriterio con los criterios (P_{max},L_{max})

Supóngase que se tienen que planificar *n* trabajos independientes en una única máquina. Dicha máquina no puede procesar simultáneamente más de un trabajo. Los trabajos están disponibles para su procesamiento desde el instante inicial. No se permiten interrupciones de los trabajos. El procesamiento del trabajo J_i, *(i=1,...,n)* requiere un tiempo p_i, medido sin interrupciones, e idealmente debe comenzar en el instante s_i (instante deseado o preferible de comienzo), y también debe completarse idealmente antes de una fecha límite dada d_i. Dada una planificación σ de los trabajos se pueden computar los valores para σ de las variables asociadas a cada trabajo J_i; es decir, S_i el instante de comienzo, $C_i = S_i + p_i$ el tiempo de completación sin solapamientos, $P_i = s_i - S_i$ su puntualidad y $L_i = C_i - d_i$ su demora. También pueden evaluarse en referencia a σ los valores $P_{max} = max_{1\le i\le n}\{P_i\}$, y $L_{max} = max_{1\le i\le n}\{L_i\}$

Nótese que si $s_i = d_i - p_i, \forall i$ entonces la puntualidad de cada trabajo coincide con su anticipación (o prontitud en la completación) ya que

$$P_i = s_i - S_i = d_i - p_i - (C_i - p_i) = d_i - C_i = max\{0, d_i - C_i\} = E_i$$

y, por tanto, los criterios P_{max} y E_{max} son equivalentes en ese caso.

Los problemas considerados respecto a los criterios P_{max} y L_{max} son *1||F(P_{max}, L_{max})* y *1|nmit|F(P_{max}, L_{max})* siendo *F* una función compuesta general. Aunque el primero de estos problemas ya fue estudiado por Smith (1956), sólo unos pocos problemas bicriterio han sido estudiados desde entonces. Muchos de ellos corresponden a minimización jerárquica como los estudiados por el propio Smith (1956) y por Shanthikumar (1983). Otros realizan una minimización simultánea agregando los criterios en una única función objetivo compuesta como en Nelson, Sarin y Daniels (1986). Hay muchas contribuciones utilizando técnicas de ramificación y acotación y algunas que no hacen uso primordial de estas técnicas como las debidas a Garey, Tarjan y Wilfong (1988), que dan un algoritmo de *O(n (log* $\sum p_i$*))* para el problema $1||max\{E_{max}, L_{max}\}$; y, Hoogeveen y Van de Velde (1992a), que aportan un algoritmo de $O(n^3 min\{n, \log \sum p_i\})$ para el problema $1||F(f_{max}, \sum C_i)$, y un algoritmo de $O(n^4)$ para el problema $1||\alpha E_{max} + \sum C_i$ con $\alpha \le 1$.

3.2.1. Problemas unicriterio relacionados

Recordemos que los problemas $1||P_{max}$ y $1/\!/L_{max}$ son resolubles en tiempo polinomial. El problema de minimizar la puntualidad máxima en ausencia de tiempo ocioso de la máquina se optimiza mediante la regla *MTST*, y el de minimizar la demora máxima mediante la regla *EDD*.

La regla *MTST* consiste en secuenciar los trabajos en orden no decreciente de sus instantes de comienzo ideales s_i. La regla *EDD*, Jackson (1955), consiste en secuenciar los trabajos en orden no decreciente de sus fechas límite de entrega d_i. Cada una de estas secuencias puede obtenerse en tiempo $O(n \log n)$

La regla *MTST* puede entenderse como una generalización de la regla *EDD*. Obviamente, y dado que *MTST* minimiza P_{max} y *EDD* minimiza L_{max}, si ambas secuencias coinciden, la secuencia resultante será óptima tanto para $1||F(P_{max}, L_{max})$ como para $1|nmit|F(P_{max}, L_{max})$, ya que en este último caso la incorporación de tiempo ocioso en la máquina no conlleva una disminución del valor de la función objetivo. Desafortunadamente ambas secuencias generalmente no coinciden por lo que se hace necesario calcular los óptimos de Pareto para (P_{max}, L_{max})

Sabiendo que si $F(f^1(\sigma),...,f^K(\sigma))$ es una función objetivo compuesta de K criterios, y si F es no decreciente en todos sus argumentos, entonces existe un óptimo de Pareto de $(f^1,...,f^K)$ que minimiza la función F; será evidente deducir que si el número de puntos óptimos de Pareto esta acotado por un polinomio en n, por ejemplo n, y, cada óptimo de Pareto puede detectarse en tiempo polinomial, entonces, existirá un algoritmo polinomial para minimizar la función F. Esto es también cierto para el caso particular $K = 2$.

Comencemos analizando el problema $1|nmit|F(P_{max}, L_{max})$. Para hallar los puntos óptimos de Pareto de (P_{max}, L_{max}) bajo la hipótesis de no permitir tiempo ocioso en la máquina podemos seguir la siguiente estrategia:

i) Dado un valor P de P_{max} que corresponde a un posible óptimo de Pareto (P,L) para (P_{max}, L_{max}) resolvemos $1|P_{max} \le P$, nmit$|L_{max}$ y obtenemos L.

ii) Determinamos el próximo valor P_{max} que corresponde a un posible óptimo de Pareto.

La estrategia anterior presenta algunas dificultades:

1. ¿Cómo seleccionar el valor inicial P de P_{max}?

2. ¿Cómo resolver el problema $1|P_{max} \leq P$, nmit$|L_{max}$?

3. ¿Cómo determinar el próximo valor P_{max} que corresponde a un posible óptimo de Pareto de tal modo que el número total de puntos generados no sea demasiado grande?

La primera dificultad puede subsanarse eligiendo como P_{max} inicial el que reporta la regla *MTST*, ya que no puede haber un óptimo de Pareto con un valor P_{max} inferior a P_{max}*(MTST)*

La segunda dificultad concierne a la resolución del problema $1|P_{max} \leq P$, $nmit|L_{max}$. La restricción $P_{max} \leq P$ induce para cada trabajo J_j una fecha de disponibilidad $r_j = s_j - P$, ya que $s_j - S_j \leq P, \forall j$; y, por tanto, el problema es idéntico a $1|r_j = s_j - P, nmit|L_{max}$. Lenstra, Rinnooy Kan y Brucker (1977) prueban que el problema con r_j generales $1|r_j, nmit|L_{max}$ es fuertemente NP-duro. No obstante, para el problema $1|r_j = s_j - P, nmit|L_{max}$, donde los instantes ideales de comienzo verifican $s_j \in \{d_j - p_j, d_j\}$, Hoogeveen (1992a) encuentra un algoritmo de $O(n \log n)$ que lo resuelve.

En cuanto a la tercera dificultad, es claro que incrementando el valor de P en una unidad en cada iteración se obtendrían todos los óptimos de Pareto, pero esta técnica no nos llevaría a un algoritmo polinomial.

Deben entonces subsanarse las dificultades de modo que la estrategia esbozada pueda desarrollarse completamente en tiempo polinomial.

Subsanando las dificultades 2) y 3) puede derivarse un algoritmo polinomial para el problema $1|nmit|F(P_{max}, L_{max})$ siguiendo la estrategia planteada. Dicha estrategia permite concluir que, si es admisible la incorporación de tiempo ocioso en la máquina, el problema $1||F(P_{max}, L_{max})$ resulta, a diferencia del anterior, NP-duro en sentido fuerte.

Una subclase de $1|r_j, nmit|L_{max}$ *resoluble en tiempo polinomial*

Para enfrentar la dificultad 2), bastaría con resolver polinomialmente el problema $1|P_{max} < P, nmit|L_{max}$. Para ello podemos considerar una subclase de problemas de $1|r_j, nmit|L_{max}$ que puede resolverse polinomialmente de modo que nuestro problema sea uno de esa clase.

Lenstra, Rinnooy Kan y Brucker (1977) han probado que, para el caso general, el problema $1|r_j, nmit|L_{max}$ es fuertemente NP-duro. Sin embargo, Lawler, Lenstra, Rinnooy Kan y Shmoys (1993) reflejan que los casos particulares a)

$r_j = r, \forall j$; b) $d_j = d, \forall j$; c) $p_j = p, \forall j$; y d) se permiten interrupciones de los trabajos; son problemas polinomialmente resolubles.

Hoogeveen (1992a) toma la subclase de problemas de $1|r_j, nmit|L_{max}$ en la que intervienen diferentes fechas de disponibilidad dependientes de la posición relativa en la secuencia pero independientes de los trabajos. Estos problemas pueden denotarse por $1|r_{[k]} = s_j - K_k, nmit|L_{max}$ donde [k] indica la posición k-ésima, $K = (K_1, \ldots, K_n) \in \Re^n$ es un vector de $\Re^n$ con $K_i \leq K_{i+1}, \forall i = 1, \ldots, n-1$. El trabajo J_j puede comenzar en la posición k-ésima en σ si $s_j - K_k \leq C_{[k-1]}(\sigma)$. El problema $1|P_{max} < P, nmit|L_{max}$ es un problema de esta clase. La resolución de cualquier problema de dicha clase se lleva a cabo por una extensión de la regla de Jackson o regla *EDD*. La extensión de la regla *EDD* a la que se hace referencia viene dada por el Algoritmo *A*. Este algoritmo consigue que los empates de la regla *EDD* se decidan de acuerdo con el orden no decreciente de instantes ideales de comienzo s_j

Algoritmo A (extensión de la regla EDD)

(0). Inicialización: $T \leftarrow 0; k \leftarrow 1; U \leftarrow \{J_1, \ldots, J_n\}; V \leftarrow \varnothing$. (*T* denota el instante de comienzo del trabajo en la *k*-ésima posición).

(1) Para cada trabajo $J_j \in U$ si $s_j - K_k \leq T$, entonces $V \leftarrow V \cup \{J_j\}$ y $U \leftarrow U - \{J_j\}$. (*U* denota el conjunto de trabajos no planificados que no pueden comenzar a procesarse en el instante *T, V* denota el conjunto de trabajos no planificados que pueden comenzar a procesarse en el instante *T*.)

(2) Si *V* es vacío, entonces parar. En otro caso, determinar el trabajo con la menor fecha límite en el conjunto *V*. Si hay empates, elegir el trabajo con menor fecha de completación ideal. Si aún hay empates, elegir el trabajo con menor índice. Supóngase que J_i es elegido. Asignarlo a la *k*-ésima posición.

(3) $T \leftarrow T + p_j; k \leftarrow k + 1; V \leftarrow V - \{J_j\}$

(4) Si permanecen trabajos sin asignar, ir al paso 1.

El Algoritmo *A* es polinomial. Además resuelve óptimamente el problema, con fechas de disponibilidad dependientes de la posición, $1|r_{[k]} = s_j - K_k, nmit|L_{max}$. Para comprobar esto último nos podemos basar en los siguientes resultados:

a) Dada σ una planificación arbitraria y dados J_i y J_i dos trabajos, donde J_i esta planificado antes que J_j en σ. Si J_j no puede comenzar a procesarse en el instante $C_j(\sigma)$, ó, si $L_i(\sigma) > L_j(\sigma)$, entonces, ocurre que $s_j > s_i$ y $d_i > d_j$.

b) *(Regla de dominancia)* Dados J_i y J_j dos trabajos arbitrarios con $s_i \leq s_j$ y $d_i \leq d_j$ y al menos una de las desigualdades estricta, entonces, existe una planificación óptima para $1|r_{[k]} = s_j - K_k, nmit|L_{max}$, en la que J_i precede a J_j

Veamos a continuación un ejemplo numérico de este algoritmo.

Ejemplo del Algoritmo A

Considérese la tabla de datos referentes a los tiempos de proceso, fechas límites, e instantes ideales de comienzo de ocho trabajos, así como el vector de valores K_k , siguientes:

j	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***
p_j	*121*	*79*	*147*	*83*	*130*	*102*	*96*	*88*
d_j	*260*	*266*	*269*	*336*	*337*	*400*	*683*	*719*
s_j	*0*	*150*	*0*	*100*	*100*	*0*	*400*	*600*

$\mathbf{K_k}$	0	25	50	75	100	150	150	200

La ejecución del algoritmo sería:

(0) *T=0; k=1; U={1,...,8}; V=∅;*K_k*=0.*
(1) *V={1,3,6}; U={2,4,5,7,8}.*
(2) Se elige el trabajo de índice 1. σ *= {1}.*
(3) *T=121; k=2; V={3,6};* K_k*=25.*

(1) *V={3,6,4,5};U={2,7,8}.*
(2) Se elige el trabajo de índice 3. σ *= {1-3}.*
(3) *T=268; k=3; V={6,4,5};* K_k*=50.*

(1) *V={6,4,5,2};U={7,8}.*
(2) Se elige el trabajo de índice 2. σ *= {1-3-2}.*
(3) *T=347; k=4; V={6,4,5};* K_k*=75.*

(1) *V={6,4,5,7};U={8}.*
(2) Se elige el trabajo de índice 4. σ *= {1-3-2-4}.*
(3) *T=430; k=5; V={6,5,7};* K_k*=100.*

(1) *V={6,5,7};U={8}.*
(2) Se elige el trabajo de índice 5. σ = *{1-3-2-4-5}.*
(3) *T=560; k=6; V={6,7};* K_k*=150.*

(1) *V={6,7,8};U=∅.*
(2) Se elige el trabajo de índice 6. σ = *{1-3-2-4-5-6}.*
(3) *T=662; k=7; V={7,8};* K_k*=150.*

(1) *V={7,8};U=∅.*
(2) Se elige el trabajo de índice 7. σ = *{1-3-2-4-5-6-7}.*
(3) *T=758; k=8; V={8};* K_k*=200.*

(1) *V={8};U=∅.*
(2) Se elige el trabajo de índice 8. σ = *{1-3-2-4-5-6-7-8}.*
(3) *T=846; k=9; V=∅.*

(4) *Parar.* La planificación propuesta es σ = *{1-3-2-4-5-6-7-8}.*

El Algoritmo *A* también resuelve óptimamente, por ser miembros de la subclase anterior, los problemas $1/P_{max} < P, nmit | L_{max}$ y $1/L_{max} < L, nmit | P_{max}$. Para cerciorarse de ello basta con considerar que $P_j = s_j - S_j$, y que $L_j = C_j - d_j = S_j + p_j - d_j = S_j - (d_j - p_j)$ y elegir los K_k convenientemente.

Veamos que ocurre cuando no se permite tiempo ocioso en la máquina.

3.2.2. Puntos óptimos de Pareto cuando no se permite tiempo ocioso en la máquina.

Para afrontar la dificultad 3), se tiene que encontrar una cota polinomial en n al número de puntos óptimos de Pareto, dicha cota puede ser precisamente *n*. También es necesario indicar como determinar el nuevo valor P_{max} y establecer un algoritmo para encontrar todos los puntos óptimos de Pareto.

El algoritmo que presenta Hoogeveen (1992a) permite determinar todos los valores *P* de P_{max} que pueden corresponder a un punto óptimo de Pareto. Conocido *P*, el correspondiente valor L_{max} puede determinarse en tiempo $O(n \log n)$ resolviendo el correspondiente problema $1/P_{max} < P, nmit / L_{max}$ por el Algoritmo *A*. Acotando el número de óptimos de Pareto para (P_{max}, L_{max}) por n se deduce inmediatamente un algoritmo polinomial para encontrarlos todos.

Nótese que si P es una cota superior arbitraria de P_{max} se puede hacer decrecer L_{max} sin más que intercambiar dos trabajos que no están planificados en orden *EDD*. Esta idea induce a particionar la secuencia σ en bloques de tal manera que los intercambios necesarios para decrecer L_{max} sólo tienen lugar en el interior de cada bloque.

Algoritmo de partición (de una planificación σ en bloques)

(1) Seleccionar el trabajo J_i para ser primero en el bloque. Comparar la fecha límite de J_i con la fecha límite de sus sucesores en la secuencia hasta encontrar un trabajo que no tenga fecha límite menor. Sea J_k tal trabajo. El bloque contiene J_i y todos los trabajos siguientes en la secuencia hasta J_k.

(2) Proceder con el siguiente trabajo hasta que la planificación este totalmente particionada en bloques.

Dada σ una planificación óptima para $1/P_{max} < P, nmit/L_{max}$ obtenida por el algoritmo *A*, donde *P* es una cota superior arbitraria de P_{max}, con $P > P(MTST)$, y siendo $P(MTST) = P_{max}(MTST)$. Particionando σ en bloques de acuerdo con el algoritmo de partición se tiene que cualquier bloque *B* satisface las siguientes propiedades:

(1) si el trabajo J_i es el primer trabajo en *B*, entonces todos los trabajos J_i en σ con menor fecha límite planificados después de J_i también pertenecen al bloque *B*.

(2) los trabajos en *B* se planifican en el siguiente orden: el trabajo con mayor fecha límite se planifica primero, los otros trabajos se planifican en orden *EDD*.

Además se sabe que:

(a) Sean P_1 y P_2 dos valores arbitrarios de una cota superior de P_{max}, con $P_1 \leq P_2$, y si σ_1 y σ_2 son las planificaciones óptimas obtenidas aplicando el algoritmo *A* a los problemas $1/P_{max} < P_1, nmit/L_{max}$, y $1/P_{max} < P_2, nmit/L_{max}$, respectivamente. Supongamos que se particiona σ_1 en bloques de acuerdo con el algoritmo de la partición y *B* es un bloque arbitrario de σ_1, con T_1 y T_2 los instantes de comienzo y completación del bloque *B* en σ_1, respectivamente. Entonces los trabajos pertenecientes a *B* se procesan en σ_2 durante el intervalo $[T_1, T_2]$.

(b) Sea σ una planificación óptima para $1/P_{max} < P, nmit/L_{max}$, obtenida por el algoritmo *A*, donde *P* es una cota superior arbitraria de P_{max}, y, sea *B* un bloque que contiene un trabajo J_i con $L_i(\sigma) = L_{max}(\sigma)$. En aras de disminuir $L_{max}(\sigma)$, es necesario incrementar *P* de modo que otro trabajo en el bloque *B* pueda planificarse primero.

(c) Sean σ_1 y σ_2 dos planificaciones obtenidas por el algoritmo *A*, a las que corresponden, respectivamente, los puntos óptimos de Pareto (P_1, L_1) y (P_2, L_2). Supóngase $P_1 < P_2$. Particionamos σ_1 y σ_2 en bloques aplicando el algoritmo de la partición. Entonces, el número de bloques en los que σ_1 ha sido particionada es inferior al número de bloques en los que ha sido particionada σ_2.

Consecuentemente el número de óptimos de Pareto esta acotado por *n* y esta cota es asintótica.

De los resultados anteriores se deduce inmediatamente que un nuevo punto óptimo de Pareto puede obtenerse incrementando el valor *P* de la cota superior de P_{max} tal que para cada bloque *B* que contiene un trabajo J_i con $L_i = L_{max}$ otro trabajo puede ser planificado en la primera posición en *B*. Esta observación es la base de un algoritmo que, dada una planificación óptima σ para $1/P_{max} < P, nmit/L_{max}$, determina el próximo valor P_{max} correspondiente a un punto óptimo de Pareto.

Algoritmo P

(0) Particionar σ en bloques de acuerdo con el algoritmo de la partición.

(1) Determinar para cada bloque *B* que contiene un trabajo J_i con $L_i(\sigma) = L_{max}(\sigma)$ el valor de una cota superior *P* tal que otro trabajo en *B* pueda ser planificado en la primera posición. Si J_i es el único trabajo en *B*, entonces L_{max} no puede decrecer, por tanto, parar.

(2) Elegir el máximo de los valores encontrados en el paso 1. Este máximo es el nuevo valor de *P*.

Una implementación correcta de todas las propiedades deducidas anteriormente nos llevan a un algoritmo que determina todas las planificaciones óptimas de Pareto en tiempo $O(n^2 \log n)$. Se ganaría en tiempo si se determinan los puntos óptimos de Pareto en lugar de las planificaciones. Nótese que después de seleccionar un punto óptimo de Pareto, la correspondiente planificación se deduce fácilmente.

El algoritmo que se establece depende fuertemente de las propiedades de los bloques. Supóngase que los trabajos están numerados en orden creciente de fechas límite deshaciendo los empates en el orden creciente de instantes de comienzo ideales. Considérese un bloque arbitrario *B*, y supóngase que contiene los trabajos $\{J_i,...,J_l\}$. Entonces el tiempo de completación de cada uno de los trabajos en $\{J_i,...,J_{l-1}\}$ es igual al tiempo de completación en la secuencia *EDD* más el valor p_l. Por tanto, la demora máxima relativa al bloque *B* se alcanza en el trabajo de $\{J_i,...,J_{l-1}\}$ que tiene la demora máxima en la planificación *EDD*. Más aún, el trabajo que ocupa la primera posición en *B* cuando la cota superior de P_{max} se incrementa mínimamente es el trabajo de $\{J_i,...,J_{l-1}\}$ que tiene el menor instante ideal de comienzo; y la cantidad mínima necesaria para incrementar *P* es $s_j - C_{i-1}(EDD) - P$.

Las observaciones anteriores muestran que, una vez almacenados los necesarios órdenes por bloque, se pueden determinar el valor L_{max} de las planificaciones de *B* y el valor de la cota superior que se necesita para alterar la secuencia en *B* en tiempo constante. Para ello se almacenan los valores L_{max} y el valor de la cota superior asociados a cada bloque *B* en un árbol ordenado. De esta manera puede determinarse el nuevo valor P_{max}, y su correspondiente L_{max} en tiempo $O(n\log n)$. Como establecer los órdenes en cada bloque lleva un tiempo $O(n)$, el algoritmo *B* se ejecuta en tiempo $O(n^2)$.

Algoritmo B

(0) Resolver el problema $1|nmit|P_{max}$, obteniendo el valor $P(MTST)$. Resolver el problema $1/P_{max} < P(MTST), nmit/L_{max}$ obteniendo la secuencia σ. Almacenar $(P_{max}(\sigma), L_{max}(\sigma))$. Particionar σ en bloques aplicando el algoritmo de la partición.

(1) Determinar el orden *MTST*; el orden $L_j(EDD)$ (en el que los trabajos se ordenan de acuerdo a valores no decrecientes de sus demoras en la secuencia *EDD*); y los valores $C_j(EDD), \forall j = 1,...,n$. Particionar las ordenaciones *MTST* y $L_j(EDD)$ de acuerdo a la partición de σ. Determinar para cada bloque *B* su demora máxima relativa y su nuevo valor de la cota superior. Almacenar esos valores en un árbol ordenado.

(2) Determinar el elemento mínimo en el árbol ordenado que contiene los nuevos valores de las cotas superiores relativas a los bloques; sea éste P.
- Si $P = \infty$, ir al paso 5; todos los puntos interesantes ya han sido descubiertos.

- En otro caso, supóngase que P se origina desde el trabajo J_k; y supongamos que este trabajo esta en el bloque B que contiene los trabajos $\{J_i,...,J_l\}$

(3) Partir B en dos partes: la primera parte contiene los trabajos $J_i,...,J_k$, mientras que la segunda parte contiene los trabajos $J_{k+1},...,J_l$. Determinar para cada parte el valor de la demora máxima relativa al bloque, almacenar estos valores en el árbol ordenado y borrar el valor antiguo de la demora máxima relativa a B del árbol. Determinar para la segunda parte el nuevo valor de la cota superior correspondiente al bloque; sea este valor P_1. Si $P_1 < P$ la segunda parte debe partirse nuevamente. Esto puede hacerse de la misma manera que se ha descrito. El proceso se repite hasta que el valor corriente P_1 sea mayor que P. Si $P_1 > P$, entonces hemos obtenido un punto interesante con valor P_{max} igual a P, y valor L_{max} igual al elemento maximal en el árbol ordenado que contiene los valores de demora máxima relativos a los bloques.

(4) Almacenar este punto, e ir al paso 2.

(5) Si se quiere optimizar una función compuesta F en los criterios, evaluar para cada punto interesante la función F, y elegir el de valor mínimo. Supóngase que el mínimo se alcanza en el punto (P,L). La planificación óptima correspondiente se determina entonces resolviendo el problema $1/P_{max} < P,nmit|L_{max}$ mediante la extensión de la regla de Jackson (algoritmo A).

Veamos ahora que ocurre cuando sí se permite tiempo ocioso.

3.2.3. Planificaciones óptimas de Pareto cuando se permite tiempo ocioso en la máquina.

La estrategia planteada anteriormente, que permite resolver óptima y polinomialmente el problema de hallar los óptimos de Pareto de (P_{max}, L_{max}) cuando no se admite tiempo ocioso en la máquina, y, cuando se satisface la condición $s_j \in \left[d_j - p_j, d_j\right] \forall j$; puede usarse para garantizar que, en presencia de tiempo ocioso en la máquina, el problema $1||F(P_{max}, L_{max})$ es fuertemente NP-duro.

El problema $1/P_{max} < P/L_{max}$ es resoluble en tiempo polinomial, con lo cual se subsanaría la dificultad 2) en la aplicación de la estrategia. Además, la curva

que conecta todos los puntos (P, L), donde L es una salida del problema $1/P_{max} < P/L_{max}$, es una función lineal a trozos con gradiente alternativamente -1 y 0, y que puede calcularse en tiempo $O(n^2 \log n)$. Pero, a pesar de todo ello, el problema $1||F(P_{max}, L_{max})$ sigue siendo NP-duro en sentido fuerte. Este hecho contrasta con la complejidad polinomial del problema $1|nmit|F(P_{max}, L_{max})$.

Para estudiar el problema $1/P_{max} < P/L_{max}$ se puede razonar de la siguiente manera. Como L_{max} es una medida regular, basta buscar la solución óptima entre las planificaciones activas. Estas planificaciones tienen la característica que no se puede anticipar el procesamiento de ningún trabajo sin que ello conlleve el incremento del tiempo de completación de algún otro. Teniendo en cuenta el análisis realizado para la versión del problema en la que no se permite tiempo ocioso en la máquina, y dado que el problema $1/P_{max} < P/L_{max}$ no es fácilmente abordable en si mismo, se vuelve a formular en modo que se aplique el estudio hecho para la versión que no admite tiempos ociosos en la máquina.

La posibilidad de insertar tiempo ocioso en la máquina tiene una importante consecuencia: si se tiene una secuencia parcial sin tiempo ocioso en la que los primeros *k-1* trabajos están fijados, en vez de planificar el trabajo disponible con la menor fecha límite en la posición *k*, puede ser ventajoso esperar hasta que otro trabajo con menor fecha límite esté disponible. Esto parece similar a incrementar la cota superior *P* como en la sección anterior, pero difiere ya que insertar tiempo ocioso afecta a los tiempos de completación de los trabajos que se planifiquen a partir de esa posición. Esta circunstancia induce a introducir en el modelo fechas de disponibilidad dependientes de la posición.

Así, se vuelve a formular el problema $1/P_{max} < P/L_{max}$ en modo que aparezcan fechas de disponibilidad dependientes de la posición y sea equivalente a un problema en el que se incorpore la restricción de no permisibilidad de tiempo ocioso en la máquina.

Para ello supóngase que σ es una secuencia óptima para $1/P_{max} < P/L_{max}$. Sea $\overline{\sigma}$ la secuencia correspondiente eliminando los tiempos ociosos. Tomemos $P_{[i]}\left(\overline{\sigma}\right)$ la puntualidad del trabajo en la posición i en $\overline{\sigma}$. Sean $K_i = max_{1 \le k \le i} P_{[k]}, i = 1, \ldots, n$. La cantidad total de tiempo ocioso que puede insertarse en $\overline{\sigma}$ antes que $J_{[i]}$ se realice para originar una planificación factible con respecto a la restricción $P_{max} \le P$ es $max\{K_i - P, 0\}$.

Luego, el problema $1/P_{max} < P/L_{max}$ puede formularse como $1/P_{[i]} \le K_i, nmit/max_{1 \le i \le n}\left\{L_{[i]} + max\{K_i - P, 0\}\right\}$, donde el conjunto de restricciones $P_{[i]} \le K_i$ induce el conjunto de fechas de disponibilidad posicionales

$$r_{[i]} = s_j - K_i; i = 1,\ldots,n; j = 1,\ldots,n\,.$$

Para resolver este último problema puede usarse una aproximación paso a paso: dado un vector no decreciente de cotas superiores K^j que posiblemente corresponda a una solución óptima del problema anterior, determinamos K^{j+1} incrementando al menos una componente de K^j en la cual $L_{[i]}$ decrece, siendo $J_{[i]}$ el trabajo que define el $max_{1\le k\le n}\left\{L_{[k]} + max\{K_k - P, 0\}\right\}$. Nótese que, dado un vector $K^j = (K_1^j,\ldots,K_n^j)$ de cotas superiores con $K_i^j \le K_{i+1}^j, \forall i = 1,\ldots,n-1$, el conjunto óptimo de valores $L_{[i]}$ se determina resolviendo los problemas $1|P_{[i]} \le K_i^j, nmit\,|L_{max}$ con el algoritmo *A*. Nótese que las componentes de K^j son no decrecientes, y también que las componentes de K^{j+1} no son inferiores a las componentes correspondientes de K^j.

Sea $\left\{K^1,\ldots,K^m\right\}$ el conjunto de vectores que se obtienen con la aproximación anterior. Definimos σ_j la secuencia obtenida al resolver el problema $1|P_{[i]} \le K_i^j, nmit\,|L_{max}$ con el algoritmo *A*; definimos $\sigma_j(P)$ como la planificación activa que se obtiene cuando se convierte σ_j en factible con respecto a la restricción $P_{max} \le P$. Claramente los vectores $\left\{K^1,\ldots,K^m\right\}$ tienen que ser minimales, esto es, si una de las componentes de $K^j, j = 1,\ldots,m$ decrece, entonces σ_j pasa a ser no factible con respecto a las restricciones $P_{[i]} \le K_i^j$.

Para aplicar el algoritmo paso a paso anterior se deben determinar el conjunto de vectores de cotas superiores $\left\{K^1,\ldots,K^m\right\}$ que guíen a una posible secuencia óptima $\sigma_j(P)$ para el problema $1/P_{max} < P/L_{max}$. Estos vectores se detectan de una manera similar a como se detecta el conjunto de posibles valores P_{max} correspondientes a óptimos de Pareto para la versión *"nmit"* del problema, es decir, sin admitir tiempo ocioso en la máquina.

Considérese un vector K del conjunto $\left\{K^1,\ldots,K^m\right\}$. Sean *k* y *l* tales que $K_{k-1} < K_k = \ldots = K_l < K_{l+1}$. El conjunto de posiciones *{k,...,l}* se llama un *grupo de posiciones*, el conjunto de trabajos $\left\{J_{[k]},\ldots,J_{[l]}\right\}$ se llama un *grupo de trabajos*. Nótese que la correspondiente secuencia $\sigma(P)$ no contiene tiempo ocioso en un grupo de trabajos. La demora máxima de un grupo *G* será

$$L(G)_{max} = max\left\{L_{[i]}(\sigma) / J_{[i]} \in G\right\},$$

y $K(G)$ como el valor K común para el grupo de posiciones correspondiente a G.

Sea σ_j la secuencia obtenida resolviendo $1|P_{[i]} \le K_i^j, nmit|L_{max}$ por el algoritmo A, donde $K^j \in \{K^1,...,K^m\}$. Particionamos σ_j en grupos. Sean G_1 y G_2 dos grupos arbitrarios de trabajos, donde G_1 se completa antes que G_2. Si $J_{[i]}$ y $J_{[j]}$ son dos trabajos arbitrarios en G_1 y G_2, respectivamente, entonces $d_{[i]} < d_{[j]}$.

Sean K^j y K^k dos vectores arbitrarios del conjunto $\{K^1,...,K^m\}$, y denotemos por K^k el mayor de los dos. Sean σ_j y σ_k secuencias óptimas obtenidas aplicando el algoritmo A a $1|P_{[i]} \le K_i^j, nmit|L_{max}$ y a $1|P_{[i]} \le K_i^k, nmit|L_{max}$, respectivamente. Particionamos σ_j y σ_k en grupos, y sean G_1 y G_2 dos grupos arbitrarios de trabajos en σ_j, donde G_1 se completa antes que G_2 en σ_j. Si $J_{[a]}$ y $J_{[b]}$ son dos trabajos arbitrarios en G_1 y G_2, respectivamente, entonces, $J_{[a]}$ precede a $J_{[b]}$ en σ_k.

De los resultados anteriores se deduce que si comenzamos con un vector K de cotas superiores de $P_{[i]}$, entonces el único camino para decrecer $L(G)_{max}$ es incrementar el valor de las cotas superiores para todo el grupo G o para parte del grupo. Esta observación es la base del siguiente algoritmo.

Algoritmo K

(0) Sea K un vector de cotas superiores de $P_{[i]}$. Sea σ la secuencia obtenida aplicando el algoritmo A a $1|P_{[i]} \le K_i, nmit|L_{max}$.

(1) Sea G el primer grupo en la secuencia que alcanza el valor $max\{L(G)_{max} + K(G)\}$. Particionemos este grupo en bloques por el algoritmo de la partición.

(2) Determinar el conjunto de bloques B en G que contienen un trabajo J_i con $L_i(\sigma) = L(G)_{max}$.

(3) Determinar para cada bloque $B \in$ B cuánto puede incrementarse la cota superior $K(G)$ para dejar que otro trabajo en B sea planificado en la primera posición en B. Denotemos este valor por $K(B)$. Si B consiste en un único trabajo, entonces $K(B) = \infty$ y, por tanto, $L(G)_{max}$ no puede decrecer; parar.

(4) El nuevo vector de cotas superiores K puede computarse a partir del antiguo como sigue: déjese al primer bloque de B comenzar en la posición $k + 1$.

Los primeros *k* elementos permanecen iguales. Ahora se consideran las posiciones que permanecen en *G*, supóngase que son las posiciones *k + 1, ...,l.* El nuevo valor K_{i+1} para la cota superior es $K_{i+1} = max\{K_i, K(B)\}$, donde B es el bloque que contiene la posición *i+1*. Los elementos del nuevo vector de cotas superiores K correspondientes a posiciones después de *G* son iguales al máximo entre K_l y sus antiguos valores.

Dado que cualquier grupo de trabajos *G* tiene el mismo valor *K* para todos los trabajos J_i del grupo, el algoritmo es correcto y su complejidad es de $O(n)$.

Tenemos ya los elementos necesarios para formular un algoritmo para determinar todos los vectores $\{K^1,...,K^m\}$ y las correspondientes planificaciones $\{\sigma_1,...,\sigma_m\}$. Denotemos por K^{MTST} el vector con

$$K_i^{MTST} = max\{P_{[j]}(MTST) / j = 1,...,i\}; \forall i = 1,...,n.$$

Algoritmo C

(0) Determinar el vector K^{MTST}; $l \leftarrow 1; K^l \leftarrow MTST$.

(1) Resolver $1|P_{[i]} \le K_i^j, nmit|L_{max}$ aplicando el algoritmo *A*, lo que da la secuencia σ_j.

(2) $l \leftarrow l+1$. Computar el nuevo vector K^l aplicando el Algoritmo K. Si $K < \infty$, ir al paso 1. En otro caso ir al paso 3.

(3) Todos los vectores $K \in \{K^1,...,K^m\}$ han sido determinados. Parar.

Ahora conviene visualizar el conjunto de vectores $\{K^1,...,K^m\}$ y eliminar todos aquellos que guían a secuencias dominadas. Una secuencia σ_{j+1} es dominada por una secuencia σ_j si $L_{max}(\sigma_j) \le L_{max}(\sigma_{j+1})$. A lo sumo *n* vectores de cotas superiores son determinados por el algoritmo *C*.

Si K^j y K^k son dos vectores arbitrarios del conjunto $\{K^1,...,K^m\}$. Sea K^k el mayor de los dos. Sean σ_j y σ_k secuencias óptimas obtenidas aplicando el algoritmo *A* a $1|P_{[i]} \le K_i^j, nmit|L_{max}$ y a $1|P_{[i]} \le K_i^k, nmit|L_{max}$, respectivamente. Particionamos σ_j y σ_k en bloques de acuerdo con el algoritmo de la partición.

Entonces, σ_k se particiona en más bloques que σ_j. Por tanto, el algoritmo *C* calcula a lo sumo *n* planificaciones, y, de aquí, su complejidad es $O(n^2 \log n)$.

Una consecuencia directa de estos razonamientos es que el problema $1/P_{max} < P/L_{max}$ es resoluble en tiempo $O(n^2 \log n)$; y también que el problema $1|r_j|L_{max}$ se puede resolver en tiempo $O(n^2 \log n)$ cuando $r_j \in \left[d_j - p_j - C, d_j - C\right]; \forall j = 1,\ldots,n$ para cierta constante C.

Sin embargo, el problema $1||F(P_{max}, L_{max})$ es fuertemente NP-duro ya que es polinomialmente reducible a un problema de circuito hamiltoniano.

Se estudian a continuación problemas bicriterio donde uno de los criterios son la suma de los tiempos de completación, y el criterio que le acompaña es una función de máximo regular, o bien la anticipación máxima.

3.3. Problemas bicriterio (ΣC_j,f_{max}), con f_{max} regular, y (ΣC_j,E_{max})

Se estudian a continuación los problemas bicriterio *(ΣC_j,f_{max})*, con f_{max} regular, y *(ΣC_j,E_{max})* desde el punto de vista de la minimización simultánea, considerando independencia de los criterios y relación entre ellos mediante una función objetivo compuesta.

3.3.1. Problemas unicriterio relacionados

Los problemas unicriterio relacionados consideran los criterios $\sum C_j$, L_{max}, E_{max}, f_{max}. Recordemos que estos problemas se resuelven polinomialmente mediante ordenaciones adecuadas de la secuencia de trabajos, tal como se indicó en el capítulo II. Es decir, $1||\sum C_j$ se resuelve con la regla *SPT* (Smith (1956)); $1||L_{max}$ con la regla *EDD* (Jackson (1955)); y, $1|nmit|E_{max}$ con la regla *MST*. Nótese que en el último caso se impone la restricción *"nmit",* de no permisibilidad de tiempo ocioso en la máquina, para evitar soluciones no acotadas.

Además, en virtud de un resultado debido a Lawler (1973), el problema $1||f_{max}$ se resuelve también polinomialmente, de $O(n^2)$, construyendo la secuencia desde la última posición a la primera.

3.3.2. Minimizando el tiempo total de completación y el costo máximo.

El problema $1||F(\sum C_j, f_{max})$ ha sido estudiado por Van Wassenhove y Gelders (1980), los cuales propusieron un algoritmo que creyeron pseudopolinomial. Posteriormente, Hoogeveen y van de Velde (1992b) comprobaron que el algoritmo era realmente polinomial, es decir, $1||F(\sum C_j, f_{max})$ es de $O(n^3 min\{n, \log n + \log p_{max}\})$ con $p_{max} = max_{1\le j\le n}\{p_j\}$, y que $1||F(\sum C_j, L_{max})$ es de $O(n^3)$. Este hecho convierte a los algoritmos de ramificación y acotación de Sen y Gupta (1983) y Nelson y otros (1986) en obsoletos.

El problema $1||f_{max}$ fue resuelto por Lawler (1973) por un algoritmo de orden $O(n^2)$. Emmons (1975) ha estudiado una extensión donde considera el problema jerárquico de minimizar $\sum C_j$ sujeto a la restricción de que f_{max} es minimal; y, por tanto, analiza el problema $1|f_{max} \le f^*/\sum C_j$ donde f^* denota la salida por el algoritmo de Lawler para $1||f_{max}$. Emmons resuelve el problema de minimizar el tiempo total de completación sujeto a costo máximo minimal en tiempo $O(n^2)$.

Obsérvese, sin embargo, que una cota superior de $f_j(C_j)$ induce una fecha límite $\overline{d}_j$ sobre el tiempo de completación de J_j. Cada fecha límite puede determinarse en tiempo $O(n \log \sum p_j)$ por una búsqueda binaria sobre los $O(\sum p_j)$ posibles instantes de completación. Además $\overline{d}_j$ se computa en tiempo constante si f_j tiene inversa. Por tanto, una vez que las fechas límite han sido computadas, el problema en un segundo paso consistirá en minimizar el tiempo total de completación sujeto a esas fechas límite, es decir, $1|\overline{d}_j/\sum C_j$, que requiere sólo tiempo $O(n \log n)$. Smith (1956) establece un algoritmo para $1|f_{max} \le f|\sum C_j$, donde f es una cota superior del costo de la planificación. Dicho algoritmo es el siguiente:

Algoritmo 3A

Paso 1: Calcular para cada trabajo J_j la fecha límite $\overline{d}_j$ inducida por $f_j(C_j) \le f$.
Paso 2: $T \leftarrow \sum p_j$.

Paso 3: Determinar $U \leftarrow \left\{ J_j \in J / \overline{d}_j \geq T \right\}$ como el conjunto de trabajos que se permiten completar en el instante T.

Paso 4: Determinar J_j tal que $p_j = max\left\{ p_i / J_i \in U \right\}$; en caso de empates, J_j se elige como el trabajo con menor costo cuando se completa en el instante T.

Paso 5: $J \leftarrow J - \left\{ J_j \right\}$; $T \leftarrow T - p_j$.

Paso 6: Si $T > 0$, ir al paso 3. En otro caso parar.

El algoritmo *3A* determina los puntos óptimos de Pareto con respecto a $\sum C_j$ y a f_{max}; además, nótese que el costo máximo de cada planificación óptima de Pareto varía desde f^* a $f_{max}(SPT)$.

Un algoritmo, similar al de Van Wassenhove y Gelders (1980) para $1||F(\sum C_j, L_{max})$, explota la propiedad anterior para encontrar el conjunto de óptimos de Pareto.

Algoritmo 3B

Paso 1: Calcular f^* y $f_{max}(SPT)$, sea $k \leftarrow 1$.

Paso 2: Resolver el problema $1|f_{max} \leq f(SPT)|\sum C_j$; esto produce la primera planificación óptima de Pareto, $\sigma^{(1)}$, y el primer punto óptimo de Pareto: $(\sum C_j(\sigma^{(1)}), f_{max}(\sigma^{(1)}))$.

Paso 3: $k \leftarrow k+1$. Resolver $1|f_{max} \leq f_{max}^{(k-1)}|\sum C_j$; esto produce la *k*-ésima secuencia óptima de Pareto, $\sigma^{(k)}$, y el *k*-ésimo punto óptimo de Pareto: $(\sum C_j(\sigma^{(k)}), f_{max}(\sigma^{(k)}))$.

Paso 4: Si $f_{max}(\sigma^{(k)}) > f^*$, ir al paso 3. En otro caso, o si ya han sido generados todos los óptimos de Pareto, parar.

Un elemento crucial es el número de puntos óptimos de Pareto generados por el algoritmo *3B*. Hoogeveen y Van de Velde (1992a) prueban que hay $O(n^2)$ planificaciones óptimas de Pareto, lo cual permite establecer la naturaleza polinomial del algoritmo.

Dada la secuencia σ se pueden definir los indicadores:

$$\delta_{ij}(\sigma) = \begin{cases} 1, & si \quad S_i(\sigma) < S_j(\sigma) \ y \ p_i > p_j \\ 0, & otro\ caso \end{cases}$$

$$\Delta(\sigma) = \sum_{i,j} \delta_{ij}(\sigma)$$

donde $\delta_{ij} = 1$ indica que los trabajos J_i y J_j están planificados en orden diverso al *SPT* en σ, es decir, implica que el intercambio de los trabajos J_i y J_j es un intercambio positivo porque hace decrecer el tiempo total de completación. Nótese también que $\Delta(SPT) = 0$, y que

$$\Delta(\sigma) \leq \frac{n(n-1)}{2} \quad \forall \sigma \in \Omega$$

donde Ω es el conjunto de todas las planificaciones factibles. El intercambio de los trabajos J_i y J_j en la secuencia σ sería *neutral con respecto a* σ si los tiempos de proceso p_i y p_j fuesen iguales.

Si la planificación π se obtiene desde la σ a través de un intercambio positivo, entonces $\Delta(\pi) < \Delta(\sigma)$. Además, si σ y π son dos planificaciones óptimas de Pareto con $\sum C_j(\sigma) < \sum C_j(\pi)$, entonces $\Delta(\sigma) < \Delta(\pi)$.

De lo anterior resulta que el número de planificaciones óptimas de Pareto está acotado por

$$\frac{n(n-1)}{2} + 1,$$

y dicha cota es asintótica. De ahí que el problema $1||F(\sum C_j, f_{max})$ sea resoluble en tiempo $O(n^3 min\{n, \log n + \log p_{max}\})$.; y el problema $1||F(\sum C_j, L_{max})$ en tiempo $O(n^3)$.

3.3.3. Minimizando el tiempo total de completación y la anticipación máxima.

Se consideran los criterios $\sum C_j$ y E_{max} simultáneamente. Como no existe tiempo ocioso en la máquina, los trabajos se ejecutan en el intervalo $\left[0, \sum p_j\right]$. Al no existir tiempo ocioso, es evidente que en cada planificación óptima de Pareto σ se tiene $E^* \leq E_{max}(\sigma) \leq E_{max}(SPT)$ donde E^* es la salida óptima de $1|nmit|E_{max}$; y $\sum C_j^* \leq \sum C_j(\sigma) \leq \sum C_j(MTST)$

.

Cualquier cota superior E de E_{max} induce, para cada trabajo J_j, una fecha de disponibilidad $r_j = max\{0, d_j - p_j - E\}$. El valor asociado de $\sum C_j$ puede computarse resolviendo $1|r_j,nmit|\sum C_j$. Sin embargo, Lenstra, Rinnooy Kan y Brucker (1977) muestran que este problema es NP-duro en sentido fuerte (ver también Garey y Johnson, 1979).

Una idea para abordar el problema sería relajar el mismo permitiendo interrupciones de los trabajos. El problema relajado, es decir, $1|pmtn,r_j|\sum C_j$ se resuelve en tiempo $O(n \log n)$ por el algoritmo de Baker (Baker, 1974): *"asignar siempre la máquina al trabajo disponible con tiempo de proceso restante mínimo."* La secuencia generada por este algoritmo no contiene tiempo ocioso y da un valor $E \geq E^*$.

La introducción de interrupciones tiene un efecto directo importante a la hora de encontrar todos los óptimos de Pareto. Si se permiten interrupciones, cualquier valor de E_{max} en el rango $\left[E^*, E_{max}(SPT)\right]$ puede obtenerse y, por consiguiente, corresponde a un punto óptimo de Pareto. Además, como $E_{max}(SPT) - E^* \leq \sum p_j$, el número de planificaciones óptimas de Pareto está pseudopolinomialmente acotado. El algoritmo siguiente genera estas $O(\sum p_j)$ planificaciones:

Algoritmo 3C

Paso 1: Sea $E^{(1)} \leftarrow E_{max}(SPT)$ y $k \leftarrow 1$.

Paso 2: Resolver $1|pmtn,\ r_j = d_j - p_j - E^{(k)}/\sum C_j$ obteniendo la *k*-ésima planificación óptima de Pareto denotada por $\sigma^{(k)}$.

Paso 3: $k \leftarrow k+1$, $E^{(k)} \leftarrow E^{(k-1)} - 1$; si $E^{(k)} \geq E^*_{max}$, entonces ir al paso 2. En otro caso, parar.

Consecuentemente, el problema $1|pmtn,nmit|\ F(\sum C_j, E_{max})$ es resoluble en tiempo $O(\sum p_j)$. Sobre la complejidad de $1|pmtn,nmit|\ F(\sum C_j, E_{max})$ nótese lo siguiente: se puede obtener una serie de 2^n puntos óptimos de Pareto consecutivos multiplicando los tiempos de proceso por 2^n. Como el problema de minimizar una función arbitraria *F(x,y)* no decreciente en ambos argumentos sobre 2^n valores consecutivos es NP-duro en el sentido fuerte, se tiene que $1|pmtn,nmit|\ F(\sum C_j, E_{max})$ es NP-duro en sentido ordinario, (pero no en sentido fuerte como cuando los tiempos de proceso son exponenciales).

Del razonamiento anterior se sigue inmediatamente que $1|pmtn|\ F(\sum C_j, E_{max})$ es fuertemente NP-duro.

Finalmente se considera el caso particular en el que la función objetivo compuesta es lineal, es decir, de la forma $\alpha_1 \sum C_j + \alpha_2 E_{max}$. Para resolver la variante lineal, sólo se tienen que determinar los puntos extremos. Inicialmente puede suponerse que no se permite tiempo ocioso en la máquina, por tanto sólo se consideran valores E_{max} en el intervalo $\left[E^*, E_{max}(SPT)\right]$. Denotando por $\sigma(E)$ la secuencia obtenida por el algoritmo de Baker para $1|pmtn, E_{max} \le E\ |\ \sum C_j$; $\sigma(E)$ corresponde a $(E, \sum C_j(\sigma(E)))$. Se dice que ha ocurrido un *intercambio completo* en $\sigma(E)$ si existen dos trabajos J_i y J_j de modo que J_i comienza antes que J_j en $\sigma(E-1)$, pero después de J_j en $\sigma(E)$. Pues bien, una cota superior E de E_{max} corresponderá a un punto extremo de $(\sum C_j, E_{max})$ solamente si un intercambio completo ha ocurrido en $\sigma(E)$.

Evidentemente, para determinar el conjunto de puntos extremos deben detectarse los valores candidatos E para los que debe tener lugar un intercambio completo en $\sigma(E)$. Dado un par de trabajos J_i y J_j con $p_i > p_j$ y J_i comenzando antes de J_j en $\sigma(E)$, el incremento necesario para permitir un intercambio completo entre J_i y J_j es igual a la diferencia entre la fecha de disponibilidad de J_j, deducida de $E_{max} \le E$, y el tiempo de comienzo de J_i en $\sigma(E)$. Sin embargo, si J_i se procesa entre los instantes de comienzo y completación de un trabajo interrumpible J_k, entonces un incremento de E conllevaría primero a un desplazamiento de J_i y J_j a la izquierda; y el intercambio completo de J_i y J_j puede no ocurrir antes de que un intercambio completo tenga lugar entre algún J_k y ambos J_i y J_j.

Las observaciones anteriores se usan en el algoritmo *3D* que, dado un valor de la cota superior E y la correspondiente planificación $\sigma(E)$, calcula el menor valor $\overline{E} > E$ que posiblemente corresponda a un punto extremo. La variable $a_j, \forall j = 1,\ldots,n$ representa el incremento necesario de E para realizar un intercambio completo del trabajo J_j.

Algoritmo 3D

Paso 1: Sea $T \leftarrow 0$ y $a_j \leftarrow \infty, \forall j = 1,\ldots,n$.

Paso 2: Sea J_j el trabajo que comienza en el instante *T*. Considérese los siguientes dos casos:

(a) J_j es un trabajo que puede interrumpirse. Entonces a_j es la longitud de esta porción de J_j. Sea J_l el primer trabajo que comienza después del instante $C_j(\sigma(E))$ con $p_l \geq a_j$. Colocar $T \leftarrow S_l(\sigma(E))$.

(b) J_j no es un trabajo interrumpible. Entonces

$$a_j \leftarrow min\left\{d_i - p_i - E - S_j(\sigma(E)) / J_j \in J\right\}$$

donde J denota el conjunto de trabajos para los cuales $d_i - p_i - E > S_j(\sigma(E))$ y $p_j > p_i$. Colocar $T \leftarrow C_j(\sigma(E))$.

Paso 3: Si $T < \sum p_j$, entonces ir al paso 2. En otro caso, ir al paso 4.

Paso 4: Poner $\overline{E} \leftarrow min_j\left\{a_j\right\} + E$. Parar.

Este algoritmo *3D* cumple que todos los valores E que puedan corresponder a un punto extremo $(E, \sum C_j(\sigma(E)))$ son generados por él.

Hoogeveen y van de Velde (1992a) probaron que el número de valores E de E_{max} generados por el algoritmo *3D* esta polinomialmente acotado y establecieron que $1|pmtn,nmit|\ \alpha_1 \sum C_j + \alpha_2 E_{max}$ puede resolverse en tiempo $O(n^4)$.

El argumento en que se basan dichos autores para garantizar la polinomialidad del algoritmo es el siguiente: dada una planificación σ se definen las funciones indicadoras:

$$\delta_{ij}(\sigma) = \begin{cases} 1, & si\ C_i(\sigma) \leq S_j(\sigma)\ y\ p_i > p_j \\ 0, & otro\ caso \end{cases}$$

$$\Delta_j(\sigma) = card\left(\mathrm{I}_j\right) + \sum_{i=1}^{n} \delta_{ij}(\sigma)$$

con I_j el conjunto de interrupciones de J_j y

$$\Delta(\sigma) = \sum_{i=1}^{n} \sum_{j=1}^{n} \delta_{ij}(\sigma)$$

Entonces se tienen los siguientes resultados:

a) Si E_1 y E_2 dos valores E_{max} generados por el algoritmo *3D*, con $E_1 > E_2$, resulta que $\Delta(\sigma(E_1)) < \Delta(\sigma(E_2))$.

b) Si se permiten interrupciones, entonces el número de planificaciones extremas con respecto a E_{max} y $\sum C_j$ esta acotado por $\frac{n(n-1)}{2}+1$

Aunque, es relativamente fácil construir un ejemplo tal que el algoritmo *3D* genere $\frac{n(n-1)}{2}+1$ valores E_{max} diferentes, sigue siendo una cuestión abierta ver si esta cota del número de puntos extremos es asintótica o no.

A partir de las conclusiones obtenidas para el problema con interrupciones $1|pmtn,nmit|\alpha_1\sum C_j+\alpha_2 E_{max}$ se deducen resultados para la versión del problema sin interrupciones ya que:

a) Si $\alpha_1=\alpha_2$, entonces existe una planificación sin interrupciones que es óptima para $1|pmtn,nmit|\alpha_1\sum C_j+\alpha_2 E_{max}$.
b) Si $\alpha_1>\alpha_2$, entonces cualquier planificación óptima para $1|\text{pmtn,nmit}|\alpha_1\sum C_j+\alpha_2 E_{max}$ carece de interrupciones.
c) Si $\alpha_1\geq\alpha_2$, entonces $1||\alpha_1\sum C_j+\alpha_2 E_{max}$ es resoluble en tiempo $O(n^4)$.

Nótese que si $\alpha_1<\alpha_2$, insertar tiempo ocioso puede decrecer el valor de la función objetivo compuesta. Así, supongamos las planificaciones $\sigma(E)$ y $\sigma(E+1)$, con $E<E^*$. El tiempo ocioso insertado entre los trabajos tiene la misma conducta que un trabajo interrumpible que se complete el último: si E se incrementa en una unidad entonces todos los trabajos que tienen tiempo ocioso entre su instante de comienzo y el instante 0 son desplazados una unidad hacia la izquierda. Por tanto, dado el valor $\overline{E}$ de E_{max} del primer punto extremo se puede determinar el conjunto de puntos extremos añadiendo un trabajo extra J_0 a la entrada con $p_0=d_0=E^*-\overline{E}+p_{max}+1$. El valor E depende de el radio $\frac{\alpha_2}{\alpha_1}$. Si $q>n$, entonces la inserción de tiempo ocioso siempre decrece el valor de la función objetivo y la solución óptima es no acotada. Si $q\leq n$, entonces la inserción de tiempo ocioso decrece el valor de la función objetivo hasta que no haya más que $\lceil q-1\rceil$ trabajos que tengan tiempo ocioso entre su instante de comienzo y el instante 0. El valor correspondiente a la cota superior E_{max} se determina fácilmente.

Como el número de puntos extremos es a lo sumo $\frac{n(n+1)}{2}+1$, y cada valor E_{max} corresponde a un punto extremo determinado por una aplicación

iterativa del algoritmo *3D*, el problema 1|pmtn|$\alpha_1 \sum C_j + \alpha_2 E_{max}$ es resoluble en tiempo $O(n^4)$.

A continuación se estudian problemas bicriterio en los que las funciones que intervienen son funciones de costo máximo regulares.

3.4. Problemas bicriterio con funciones de costo máximo regulares

3.4.1. Problemas unicriterio relacionados

Como se ha señalado en capítulos anteriores, Lawler (1973) presentó un algoritmo de $O(n^2)$ que resuelve óptimamente el problema unicriterio $1|prec|f_{max}$. Además, dicho algoritmo es aplicable fácilmente al problema $1|\overline{d}_j, prec|f_{max}$ ya que este problema puede plantearse como un problema $1|prec|f_{max}$, con lo que bastaría aplicar el algoritmo de Lawler al problema ajustado $1|prec|f_{max}$. Nótese, que las fechas límite no tienen que ser dadas explícitamente, pero pueden ser inducidas por cotas superiores dadas o por otros criterios. Por ejemplo, si g_i es una función de penalización no decreciente, para $i = 1, ..., n$, entonces la restricción $g_{max} \leq G$ induce una fecha límite para cada trabajo J_i.

3.4.2. Puntos óptimos de Pareto para (f_{max},g_{max}) con f_{max} y g_{max} regulares.

Para detectar los óptimos de Pareto de este problema puede aplicarse la estrategia comentada para el problema bicriterio (P_{max},L_{max}). Primero se resuelve el problema $1||f_{max}$ y se obtiene un primer valor F que corresponde a un candidato a óptimo de Pareto (F,G). Luego se determina el correspondiente valor de G resolviendo el problema $1/f_{max} \leq F /g_{max}$ por el algoritmo de Lawler. Finalmente se determina un nuevo valor F que corresponde a un posible óptimo de Pareto (F,G), y se repite el proceso hasta que todos los óptimos de Pareto hayan sido generados.

Al igual que ocurría para los criterios (P_{max},L_{max}), surgen dificultades para aplicar esta estrategia:

1.- ¿cómo determinar el nuevo valor de F?

2.- ¿cuántos de estos valores deben computarse antes de determinar todos los óptimos de Pareto?

La primera dificultad puede afrontarse de la siguiente manera: sea σ una planificación obtenida resolviendo $1/f_{max} \leq F/g_{max}$ por el algoritmo de Lawler, y J_j es un trabajo para el que se alcanza el valor $g_{max}(\sigma)$, es decir, $g_j(C_j(\sigma)) = max_{1 \leq i \leq n} g_i(C_i(\sigma))$. Como g_j es no decreciente, un punto óptimo de Pareto con menor valor g_{max} puede obtenerse sólo si el instante de completación de J_j decrece. Por tanto, cualquier trabajo J_i que esté antes que J_j en σ y tenga $g_i(C_j(\sigma)) < g_{max}$ tiene que completarse no antes del instante $C_j(\sigma)$. Esta observación es la base de un algoritmo debido a Hoogeveen (1992b) para determinar el incremento de *F* necesario para alcanzar un nuevo candidato a ser punto óptimo de Pareto. El procedimiento se describe a continuación.

Algoritmo Nueva Cota Superior (NUB)

(1) Dada una planificación σ obtenida por el algoritmo de Lawler, determinar el conjunto $\Im$ de trabajos que alcanzan $g_{max}(\sigma)$.

(2) Determinar para cada trabajo $J_j \in \Im$ el conjunto U_j de trabajos J_i que están planificados en σ y que tienen $g_i(C_j(\sigma)) < g_{max}(\sigma)$. Si $U_j = \varnothing$ para cierto $J_j \in \Im$, entonces $g_{max}(\sigma)$ no puede decrecer; parar. En otro caso, para cada trabajo $J_j \in \Im$, definir $F_j = min\{f_i(C_j(\sigma)) / J_i \in U_j\}$. La nueva cota superior *F* de f_{max} es el máximo de los valores F_j.

Este algoritmo se basa en lo siguiente:

Sea $(\hat{F}, \hat{G})$ un punto óptimo de Pareto con respecto a (f_{max}, g_{max}), y sea σ la correspondiente secuencia. Si *F* es una nueva cota para f_{max} obtenida por el algoritmo *NUB*, dado σ, entonces no hay punto óptimo de Pareto correspondiente al valor $\tilde{F}$ con $F > \tilde{F} > \hat{F}$.

El algoritmo *NUB* determina a lo sumo $\frac{n(n-1)}{2}$ valores *F*. Así, el número de puntos óptimos de Pareto con respecto a (f_{max}, g_{max}) es a lo sumo igual a $\frac{n(n-1)}{2} + 1$. Además, dicha cota es asintótica.

Los resultados anteriores garantizan que el algoritmo siguiente determina, en tiempo polinomial, todos los óptimos de Pareto y el valor de la solución óptima para el problema $1||F(f_{max}, g_{max})$.

Algoritmo 4A

(0) Determinar f^* y g^* resolviendo $1||f_{max}$ y $1||g_{max}$, respectivamente. Poner $F \leftarrow f^*$.

(1) Resolver $1|f_{max} \leq F|g_{max}$. Sea G la salida. Añadir (F, G) al conjunto de puntos óptimos de Pareto (a menos que esté dominado por el punto óptimo de Pareto previamente obtenido). Si $G = g^*$, entonces ir al paso 3.

(2) Determinar el nuevo valor F aplicando el algoritmo *NUB* a la planificación obtenida en el paso previo. Ir al paso 1.

(3) El conjunto de puntos óptimos de Pareto ha sido obtenido. El problema $1||F(f_{max}, g_{max})$ esta resuelto computando el valor de la función objetivo para cada punto del conjunto de óptimos de Pareto y eligiendo el óptimo.

El tiempo de ejecución del algoritmo *4A* es $O(n^4)$ que es el tiempo de resolver $O(n^2)$ entradas del problema $1|f_{max} \leq F|g_{max}$.

Como cierre de este capítulo, resumamos lo visto en él. Se han estudiado diversos problemas de planificación bicriterio, catalogando computacionalmente algunos de ellos y se han incorporado diversos algoritmos para su resolución. En este sentido conviene notar que la planificación bicriterio es un tema abierto en el cual se ha trabajado relativamente poco. En relación al apartado de minimización simultánea, es decir, cuando se pretenden optimizar los criterios sin unificarlos en una única función, se proponen algoritmos para calcular los puntos eficientes y los puntos eficientes extremos que están en la frontera de la envoltura convexa del conjunto de puntos eficientes para cualquier problema bicriterio. En el Apéndice C se ilustra una experiencia computacional para valores concretos de los dos criterios.

Veamos en el próximo capítulo como se puede formular mediante la Programación Matemática algunos de los modelos de planificación vistos en los

capítulos anteriores y otros nuevos, que incluyen diversas consideraciones no recogidas hasta ahora.

Capítulo V

Capítulo V: La programación matemática en los problemas de planificación

1. Introducción

Los modelos vistos en los capítulos anteriores cubren un amplio grupo de problemas de planificación, pero estos modelos no son suficientes para resolver todos los posibles problemas que aparecen en las aplicaciones prácticas.

En este último caso, la alternativa puede ser modelizar los problemas y realizar una formulación matemática adecuada, para, posteriormente, afrontar su resolución mediante la programación matemática.

Siguiendo esta línea se comenta a continuación algunas formulaciones planteadas por diversos autores para abordar problemas de planificación.

2. Algunos modelos propuestos en la literatura

2.1. Modelo de Balas (1985)

Es un modelo para problemas job shop. Como se comentó en el primer capítulo, en un problema job shop se tienen que realizar n trabajos para lo cual se dispone de m máquinas. Cada trabajo J_j consiste en una cadena de m_j operaciones $\{O_{1j}, O_{2j}, ..., O_{mjj}\}$. No tiene porque ser $m_j = m$. Puede ocurrir que alguna máquina procese dos o más operaciones de un mismo trabajo. La trayectoria de máquinas de cada trabajo esta dada, pero no tiene que ser la misma para todos los trabajos. Es decir, O_{ij} se procesa en la máquina μ_{ij} en un tiempo p_{ij}. Si una máquina realizase más de una operación de un trabajo dado, es lógico pensar que no procesaría dos operaciones seguidas de un mismo trabajo, porque de lo contrario dichas operaciones podrían agruparse en una sola. Es decir, $\mu_{i-1j} \neq \mu_{ij}$ $\forall i=2,...,m_j$, $\forall j=1,...,n$. Puede entenderse que existe una partición del conjunto de operaciones o tareas en m conjuntos $T_1, T_2, ..., T_m$ de tal forma que las tareas de T_i deben ser procesadas por la máquina M_i. Se pueden denotar las operaciones o tareas como $\{ta_1,...,ta_N\}$ donde N es el número total de operaciones en que se dividen los trabajos. Si $ta_j \in T_i$, se denota por t_{ij} la suma de los tiempos de preparación y procesamiento que M_i invierte en la realización completa de ta_j.

Las variables de decisión necesarias para este modelo serán

$$x_{jt} = \begin{cases} 1, & si\ ta_j\ comienza\ a\ procesarse\ en\ el\ instante\ t \\ 0, & otro\ caso \end{cases}$$

Y, en el caso de que el objetivo a minimizar fuese el tiempo total de completación de todos los trabajos o makespan, la formulación matemática del problema sería:

$$min_{i\in\{1,\ldots,m\}} max_{ta_j\in T_i} \left\{ t\ x_{jt} + t_{ij} \right\}$$

s.a.:

$$tx_{jt} - sx_{ps} \geq t_{ij} \quad \text{si } ta_j \in T_i, \text{ } ta_j, ta_p \text{ pertenecen al mismo trabajo y } ta_j \text{ precede a } ta_p \qquad (1)$$

$$tx_{jt} - sx_{ps} \geq t_{ij} \vee sx_{ks} - tx_{jt} \geq t_{ik} \quad \forall ta_j, ta_k \in T_i \text{ tales que } ta_j \text{ y } ta_k \text{ no pertenecen al mismo trabajo} \qquad (2)$$

$$x_{jt} \in \{0, 1\} \qquad (3)$$

A las desigualdades (1) se les conoce como restricciones conjuntivas, y a las desigualdades (2) como restricciones disyuntivas.

El problema se puede también formularse mediante un grafo disyuntivo (véase Laarhoven, Aarts, Lenstra (1988), Dell'Amico y Trubian (1993)). Si $TA=\{ta_1,\ldots,ta_N\}$ es el conjunto de operaciones o tareas, consideramos el grafo $G=(V, A, E)$ donde

$V = TA \cup \{ta_0\} \cup \{ta_{N+1}\}$ siendo ta_0, ta_{N+1} vértices ficticios que representan el inicio y el final de la realización de todos los trabajos.

$A = \{(ta_i, ta_j)$ tales que la operación ta_i es un predecesor inmediato de la operación ta_j en la cadena de operaciones del trabajo del que ambas forman parte$\} \cup \{(ta_0,ta_j), ta_j \in TA\} \cup \{(ta_i, ta_{N+1}) / ta_i \in TA\}$.

$E = \{(ta_i, ta_j) / ta_i, ta_j \in TA$ son operaciones que deben realizarse por la misma máquina$\}$.

El conjunto A es un conjunto propiamente de arcos porque la orientación viene fijada por las trayectorias de máquinas que deben seguir los diferentes trabajos. El

conjunto E es un conjunto de aristas. Una selección S consiste en ordenar todos los arcos disyuntivos (elementos de E) de modo que, si $(ta_i, ta_j) \in S$, $(ta_j, ta_i) \notin S$. A los arcos (ta_0, ta_j), (ta_i, ta_{N+1}) $\forall i$, $\forall j$ se les asigna una longitud 0. A cualquier otro arco (ta_k, ta_l) se le asigna una longitud t_{ik} si $ta_k \in T_i$, donde t_{ik} es la suma de los tiempos de preparación y proceso de ta_k en M_i..

Fijada la selección S se tiene un grafo dirigido acíclico $G(S)$. Llamando $\mu(S)$ a la longitud del camino crítico en $G(S)$ el problema consiste en encontrar la selección S^* con

$$\mu(S^*) = min_{S \in E} \mu(S)$$

Una extensión de este modelo al problema open shop se ha presentado en el capítulo III.

2.2. Modelo de Dietrich y Escudero (1990)

Este es un modelo que permite resolver problemas que se presentan al planificar tareas con un horizonte temporal acotado y tiempos de preparación *("set up")*. Se tienen que realizar n trabajos $\{J_1,...,J_n\}$. Cada uno de los cuales consta de una única operación o tarea $J_j = \{O_j\}$, y $N = n$. Las tareas se agrupan en clases o tipos $I_1, I_2,...,I_s,...,I_k$ de tal forma que $\bigcup_{s=1}^{k} I_s = \{O_1,...,O_N\} = TA$, siendo I_s las operaciones del tipo s-ésimo.

Si T_i son las operaciones que pueden realizarse en la máquina M_i, entonces I_{is} serían las operaciones de tipo s que se deben realizar en la máquina M_i. De donde $\bigcup_{s=1}^{k} I_{is} = T_i$.

Cada máquina M_i dispone de un tiempo de procesamiento U_i. Si se denota por

T_{ijs} = tiempo que la máquina M_i emplea en procesar la operación O_j del tipo I_s supuesto que $O_j \in I_{is}$.

S_{is} = tiempo de preparación de M_i para procesar el primer objeto de la clase I_s.

A_{is} = costo de preparación de M_i para procesar la clase I_s.

B_{ijs} = costo de proceso por M_i de la operación O_j del tipo I_s supuesto $O_j \in I_{is}$.

el problema consiste en asignar los trabajos a las máquinas de modo que el tiempo de procesamiento empleado por cada máquina no exceda de su tiempo disponible y se minimice el coste total.

Las variables de decisión tomadas por Dietrich y Escudero para conformar este modelo son:

$$x_{ijs} = \begin{cases} 1, & si\ la\ tarea\ ta_j\ de\ clase\ I_s\ se\ procesa\ por\ M_i \\ 0, & otro\ caso \end{cases}$$

$$z_{is} = \begin{cases} 1, & si\ existen\ tareas\ de\ I_s\ procesadas\ por\ M_i \\ 0, & otro\ caso \end{cases}$$

y la formulación del problema sería:

$$min \sum_{i=1}^{m} \sum_{s=1}^{k} \left(A_{is} z_{is} + \sum_{j \in I_{is}} B_{ijs} x_{ijs} \right)$$

s.a.:

$$x_{ijs} \leq z_{is} \quad \forall j \in I_{is},\ s \in \{1,..,k\} \quad \forall i,\ 1 \leq i \leq m \qquad (1)$$

$$\sum_{i=1/\, j \in I_s}^{m} x_{ijs} = 1 \quad \forall j \in I_s \quad \forall s \qquad (2)$$

$$\sum_{s=1}^{k} \left(S_{is} z_{is} + \sum_{j \in I_{is}} T_{ijs} x_{ijs} \right) \leq U_i \qquad (3)$$

$$x_{ijs},\ z_{is} \in \{0, 1\} \qquad (4)$$

donde (2) indica que cada tarea la procesa una única máquina; y (3) que la cantidad de tiempo que cada máquina procesa no excede del tiempo de disponibilidad de la máquina.

2.3. Modelo de Escudero y Pérez (1990)

Se trata de modelizar los problemas que se presentan cuando se deben seleccionar los trabajos a realizar y planificarlos en modo que el aprovechamiento de los recursos se optimice. Es decir, es una planificación y selección de trabajos o proyectos con recursos limitados.

Sea $J = \{J_0, J_1, ..., J_n\}$ una familia de trabajos o proyectos a realizar. Sea $TA = \{ta_1, ...,ta_N\}$ las tareas a realizar si decidiésemos efectuar todos los trabajos de J. Los elementos de J se interpretan como conjuntos constituyentes de una partición de TA de modo que $ta_j \in J_g$, con $g \neq 0$, sí, y sólo sí, la tarea ta_j forma parte del trabajo J_g. Y J_0 representa las tareas comunes a todos los trabajos. Así mismo, si T_i denota las operaciones o tareas que puede procesar la máquina M_i, entonces $T_1, ..., T_m$ configuran un recubrimiento de TA. En este modelo cada máquina necesita un tiempo de preparación para comenzar a procesar las operaciones, pero existen ciertas clases I_s de tareas tales que, una vez preparada la máquina para procesar una operación de la clase, el resto de operaciones puede realizarse sin consumir tiempo de set up. Estas clases $I_1, ..., I_k$. constituyen una partición de TA. Si, además, denotamos por I_{is} a las tareas de la clase I_s que se pueden procesar por M_i resulta

$$\bigcup_{s=1}^{k} I_{is} = T_i \text{ y } \bigcup_{i=1}^{m} I_{is} = I_s.$$

Como los recursos son limitados se tiene también una partición del conjunto $J'=\{J_1,...,J_n\}$ de trabajos o proyectos formando una familia $C=\{C_1,...,C_p\}$ de modo que el problema consiste en decidir un único trabajo a realizar de cada una de estas p clases, y después planificar los trabajos elegidos para que se optimice cierta función objetivo. Cada máquina M_i está disponible una cantidad de tiempo limitada U_i.

Se supone que en cada instante de tiempo una máquina sólo puede realizar una tarea, y que el conjunto de operaciones esta dotado de un orden parcial. Dicho orden parcial indica un orden de preferencia en el procesamiento de las tareas. Se considera la hipótesis de que las tareas se procesan consecutivamente y que cada tarea ta_j lleva asociado unos valores r_j, d_j, y t_{ij} que se corresponden, respectivamente, con el tiempo de preparación empleado, su fecha límite, y el tiempo de procesamiento.

Para realizar las tareas se precisan consumir recursos de tres tipos R^1, R^2, R^3, donde R^1 representa los recursos consumidos por los trabajos que se deciden procesar (por ejemplo, recursos de inversión de capital, etc.); mientras que R^2 denota los recursos consumidos en la fabricación de ítems necesarios para el procesamiento de las operaciones (por ejemplo, recursos de materias primas). Su

disponibilidad varía durante el período de planificación; y R^3 son los recursos consumidos en la realización de las operaciones, cuya disponibilidad permanece constante durante todo el horizonte de planificación (por ejemplo, los utensilios para la fabricación).

El problema consiste en determinar los tiempos de inicio del procesamiento de cada tarea y decidir que operaciones debe realizar cada máquina para que, cumpliendo las restricciones impuestas, se minimice el costo total.

Dicho problema puede formularse de la siguiente manera. Sea $G= (V,A)$ un grafo que modeliza las preferencias entre las tareas; siendo $V = \{j / j=1,..,N\}$ los N índices de las tareas y $A = \{(j,l)$ tales que la tarea ta_j precede inmediatamente a la tarea $ta_l\}$. Se tienen los valores:

c_g = costo asociado de elegir el trabajo J_g para su procesamiento.
c_{jt} = costo de comenzar a procesar ta_j en el instante de tiempo t.
TI_j = periodo de tiempo en el que se puede comenzar a procesar ta_j.

Las variables de decisión serán:

$$x_{ijst} = \begin{cases} 1, & si\ M_i\ comienza\ la\ tarea\ ta_j \in I_s\ en\ el\ instante\ t \\ 0, & otro\ caso \end{cases}$$

$$y_g = \begin{cases} 1, & si\ se\ decide\ procesar\ el\ trabajo\ J_g \\ 0, & otro\ caso \end{cases}$$

$$x_{jt} = \begin{cases} 1, & si\ ta_j\ comienza\ a\ procesarse\ en\ el\ instante\ t \\ 0, & otro\ caso \end{cases}$$

Una formulación matemática del modelo se expresa a continuación:

$$min \quad \sum_{j=1}^{N} \sum_{t \in TI_j} c_{jt} x_{jt} \quad + \quad \sum_{g=1}^{n} c_g y_g$$

sujeto a:

$$\sum_{t \in TI_j} x_{jt} = 1 \qquad \forall j \in J_0 \qquad (1)$$

$$\sum_{t \in TI_j} x_{jtst} = y_g \qquad \forall ta_j \in J_g \ \text{ con } g \neq 0 \qquad (2)$$

$$\sum_{g \in C_k} y_g = 1 \qquad \forall k \qquad (3)$$

$$\sum_{t\in TI_j} t x_{jt} + t_j x_{jt} + q_{jj_1} x_{jt} \quad \leq \quad \sum_{t_1\in TI_{j_1}} t_1 x_{j_1 t_1} \qquad \forall (j,j_1) \in A \qquad (4)$$

siendo q_{jj1} e tiempo que debe transcurrir desde que se acaba de procesar ta_j hasta que se comienza a procesar ta_{j1}.

$$\sum_{j\in T_i} \sum_{t\in T_1(h)} x_{jt} \leq 1, \qquad \forall h\in H, \ \forall i \qquad (5)$$

siendo H el horizonte de planificación y $T_1(h) = \{t\in TI_j / h - t_{ij} - pausa_i + 1 \leq t \leq h\}$, donde $pausa_i$ denota el tiempo que debe transcurrir entre la asignación consecutiva de dos tareas a M_i.

$$\sum_{i=1}^{m} \sum_{j\in TA_j} \sum_{t\in T_{1i}(h)} a^1_{jrk} w_{ijt} \quad \leq \quad b^1_{rh} \qquad \forall h\in H \qquad (6)$$

siendo $T_{1i}(h) = \{t\in TI_j / h - t_{ij} + 1 \leq t \leq h\}$; a^1_{jrk} la cantidad de recurso r requerida por j durante el k-ésimo periodo de asignación; $w_{ijt} = \sum_{s \ con \ j\in I_s} x_{ijst}$; y b^1_{rh} la cantidad de recurso r disponible para $r \in R^1$.

$$\sum_{g=0}^{n} a^2_{gr} y_g \leq b^2_r, \qquad \forall r\in R^2 \qquad (7)$$

siendo a^2_{gr} la cantidad de recurso r requerida por las operaciones que constituyen el trabajo J_g y b^2_r denota la cantidad disponible de recurso $r \in R^2$; y

$$\sum_{j\in TA} \sum_{t\in TI_j} a^1_{jtr} w_{jt} \quad \leq \quad b^3_r \qquad \forall r\in R^3 \qquad (8)$$

siendo a^3_{jtr} la cantidad de recurso r requerida por la operación ta_j supuesto que su asignación comienza en el periodo t; y b^3_r indica la cantidad disponible de $r \in R^3$.

Las restricciones (1) indican que todas las tareas de J_0 deben realizarse; las restricciones (2) garantizan que, una vez seleccionado un trabajo, todas sus operaciones se procesan. (3) obliga a que de cada una de las p clases C_k se elija un único trabajo. (4) establece que se respeten las precedencias; y (5) garantiza que en cada máquina M_i, y en cada instante de tiempo t no se comienzan a procesar más de una tarea. Las tres últimas restricciones indican que el consumo del recurso r no puede superar a la cantidad disponible.

2.4. Modelo de Fischetti, Martello y Toth (1992)

Se trata de un modelo para resolver el problema de planificar n trabajos $\{J_1,..., J_n\}$ en máquinas no especializadas idénticas de manera que el número de

máquinas requeridas sea mínimo. Cada trabajo J_j tiene una fecha de disponibilidad r_j y una fecha límite d_j. Este tipo de problemas se denominan *FJS* ("Fixed Job Schedule"). Fischetti, Martello y Toth (1992) consideran dos generalizaciones de este problema. Si $J_{f(i)}$ y $J_{l(i)}$ son respectivamente el primer y último trabajo asignados a la máquina M_i, se impone una cota superior s ("tiempo de extensión") sobre la diferencia $d_{l(i)} - r_{f(i)}$ para cada máquina M_i. Este problema se denomina *FJSS* ("Fixed Job Schedule con restricciones Spread-time"). El tiempo de proceso del trabajo J_j se define como $d_j - r_j$. Imponiendo una nueva cota superior w ("tiempo de trabajo") sobre la suma de los tiempos de proceso de los trabajos asignados a cada máquina, se produce el problema *FJSW* ("Fixed Job Schedule con restricciones Working-time"). *FJS, FJSS, FJSW* son casos particulares de un problema más general que es el *BDSP* ("Bus Driver Scheduling Problem").

FJS puede resolverse en tiempo polinomial *O(n log n)* por los algoritmos dados por Gupta, Lee y Leung (1979) y Nakajima, Hakimi y Lenstra (1982). *FJSS* y *FJSW* son NP-duros. Fischetti, Martello y Toth (1987,1989) presentan algoritmos eficientes de ramificación y acotación, y algoritmos polinomiales para las relajaciones de estos problemas en las que se permiten interrupciones.

El modelo que Fischetti, Martello y Toth (1992) proponen para el *BDSP* lo describen de la siguiente manera: dada la planificación diaria de la compañía de transportes, se definen n partes del trabajo J_j que requieren una conducción sin interrupciones desde el instante inicial r_j al instante final d_j $(j=1,...,n)$. Se trata de encontrar el mínimo número de conductores necesarios para realizar todas las partes de los trabajos satisfaciendo las restricciones impuestas por la unión contractual y las regulaciones de la compañía. Estas restricciones pueden variar según los diferentes contextos prácticos (Martello y Toth (1986)), pero siempre están presentes dos de ellas: para cada conductor M_i existe un límite w sobre el tiempo total de conducción y un límite temporal s sobre la duración total del servicio.

En el modelo, los autores suponen, sin pérdida de generalidad, que:

(a) todos los datos son enteros positivos

(b) $0 < d_j - r_j \leq w \leq s$

(c) cada conductor realiza una parte del trabajo J_j en el intervalo de tiempo $(r_j, d_j]$

(d) $r_1 \leq r_2 \leq ... \leq r_n$

Dada una cota superior $m \leq n$ al número de conductores se introducen las variables binarias

$$y_i = \begin{cases} 1, & \textit{si interviene el conductor } M_i \\ 0, & \textit{otro caso} \end{cases}$$

$$x_{ij} = \begin{cases} 1, & si\ el\ conductor\ M_i\ realiza\ parte\ de\ J_j \\ 0, & otro\ caso \end{cases}$$

de donde el problema *BDSP* puede formalizarse en la forma:

$$min \quad \sum_{i=1}^{m} y_i \qquad (1)$$

$$x_{ij} \le y_i \qquad \forall i=1,...,m; \quad \forall j=1,...,n \qquad (2)$$

$$\sum_{i=1}^{m} x_{ij} = 1 \qquad \forall j=1,...,n \qquad (3)$$

$$x_{ij} + x_{ik} \le 1 \quad \forall i=1,...,m; \ \forall j=1,...,n-1; \ \forall k \in \{l>j: r_j < d_j\} \qquad (4)$$

$$x_{ij} \in \{0,1\} \quad \forall i=1,...,m; \ \forall j=1,...,n \qquad (5)$$

$$y_i \in \{0,1\} \quad \forall i=1,...,m \qquad (6)$$

$$x_{ij} + x_{ik} \le 1 \quad \forall i=1,...,m; \ \forall j=1,...,n-1; \ \forall k \in \{l: d_l > r_j + s\} \qquad (7)$$

$$\sum_{j=1}^{n} (d_j - r_j) x_{ij} \le i \qquad \forall i=1,...,m \qquad (8)$$

$$F(x,y) = 0 \qquad (9)$$

Las restricciones (2) imponen que y_i valga 1 en el caso de que se utilice el conductor M_i, (3) que cada parte de un trabajo se asigna a un único conductor, y (4) que ningún conductor realiza partes de trabajos que se solapen en el tiempo. Por tanto, (1)-(6) da el problema *FJS* (las partes de los trabajos corresponden a las tareas y los conductores a los procesadores o máquinas). Las restricciones (7) imponen la cota superior al tiempo de extensión *("spread time bound");* por tanto (1)-(7) da el problema *FJSS*. Las restricciones (8) imponen la cota superior al tiempo de trabajo *("working time bound")*, por tanto, (1)-(6) y (8) dan *FJSW*. Finalmente, (9) expresa las restricciones establecidas por la unión contractual y las regulaciones de la compañía para el problema *BDSP*.

2.5. Modelo de Daniels y Mazzola (1994)

Estos autores consideran el problema flow shop estático consistente en un conjunto de n trabajos independientes $\{J_1,...,J_n\}$ y disponibles desde el instante

inicial $r_j = 0 \ \forall j$. Suponen que cada trabajo tiene exactamente una única operación en cada una de las m máquinas $\{1,..,m\}$. Las máquinas están siempre en disposición de realizar trabajos. Al ser un problema flow shop, la trayectoria de máquinas que siguen los trabajos es la misma para todos ellos. Se supone, sin pérdida de generalidad, que las máquinas han sido numeradas de modo que el orden creciente es la trayectoria común que deben seguir los trabajos.

Los autores consideran una extensión del modelo en la que suponen la existencia de un recurso disponible en cantidad limitada R, y que se utiliza en el proceso de producción. La cantidad existente de este recurso no cambia durante el proceso de producción. Modelizan que cada operación (i,j) puede procesarse en K_{ij} modos posibles, siendo p_{ijk} el tiempo de proceso de la operación (i,j) cuando ésta se procesa en el modo k. El valor r_{ijk} representa las unidades del recurso adicional requeridas para el procesamiento de (i,j) en el modo k. Sin pérdida de generalidad, puede suponerse que p_{ijk}, r_{ijk} son enteros y decrecientes en el índice k.

Si C_{ij} representa el instante de completación de la operación (i,j); $C_{max}=max_j \{C_{mj}\}$ es el instante final de completación de todos los trabajos; y $T = \sum_{i=1}^{m} \sum_{j=1}^{n} p_{ij1}$ es una cota superior de C_{max}, siendo M una constante suficientemente grande, los autores consideran las variables de decisión:

$$y_{jh} = \begin{cases} 1, & \textit{si el trabajo } j \textit{ precede al } h \\ 0, & \textit{otro caso} \end{cases}$$

$$x_{ijkt} = \begin{cases} 1, & \textit{si } J_j \textit{ se completa en la maquina } i \textit{ trabajando} \\ & \textit{esta en el modo } k \textit{ en el instante } t \\ 0, & \textit{otro caso} \end{cases}$$

y el problema consiste en:

$$min \ C_{max}$$

sujeto a:

$$C_{max} \geq C_{mj} \qquad \forall j \qquad (1)$$

$$p_{ij} = \sum_{k=1}^{K_{ij}} p_{ijk} \sum_{t=1}^{T} x_{ijkt} \qquad \forall i \ , \ \forall j \qquad (2)$$

$$C_{ih} - C_{ij} + M(1 - y_{jh}) \geq p_{ih} \qquad \forall j, \ h \neq j \qquad (3)$$

$$y_{jh} + y_{hj} = 1 \qquad \forall j, \ h \neq j \qquad (4)$$

$$C_{ij} \geq C_{i-1j} + p_{ij} \qquad \forall i, j \qquad (5)$$

$$\sum_{t=1}^{T} t\left(\sum_{k=1}^{K_{ij}} x_{ijkt}\right) = C_{ij} \qquad \forall i,j \qquad (6)$$

$$\sum_{k=1}^{K_{ij}} \sum_{t=1}^{T} x_{ijkt} = 1 \qquad \forall i,j \qquad (7)$$

$$\sum_{i=1}^{m} \sum_{j=1}^{n} \sum_{k=1}^{K_{ij}} \sum_{h=t}^{t+p_{ijk}-1} r_{ijk} x_{ijkh} \leq R \qquad \forall t=1,...,T \qquad (8)$$

$$y_{jh}, x_{ijkt} \in \{0,1\} \qquad \forall j, \ h \neq j, \ i,k,t \qquad (9)$$

$$C_{ij} \geq 0 \qquad \forall i,j \qquad (10)$$

El significado de (1) es evidente y no requiere comentario. Cada restricción (2) mide el tiempo de proceso real en la realización de la operación *(i,j)* una vez que se ha elegido el modo para su realización. Las restricciones (3) garantizan el no solapamiento entre operaciones consecutivas, mientras que (4) indica que si *h* precede a *j, j* no puede preceder a *h*. Las restricciones (5) garantizan que las operaciones que se realizan más tarde tienen tiempo de completación mayor. Cada restricción (6) indica que el instante en que se completa el procesamiento de la operación *(i,j)* es precisamente C_{ij}. En (7) se indica que se elige un único modo para realizar la operación *(i,j)*. Con (8) se asegura que no se excede la disponibilidad del recurso limitado en ningún instante de tiempo. Finalmente, (9) y (10) son restricciones evidentes.

2.6. Modelo de Matta y Guignard (1994)

Estos autores tratan el problema denominado *CPS* ("capacited-oriented production scheduling"). Este problema concierne a la asignación de productos que compiten entre ellos para ser procesados por líneas de producción de diferente capacidad en un número finito de periodos de tiempo.

En el modelo de producción que proponen consideran:

P = número de productos a planificar.

L = número de líneas de producción disponibles.

T = número de periodos del horizonte de planificación (usualmente semanas).

j = índice de las líneas de producción $(j=1,...,L)$.

i = índice del producto $(i=0,...,P)$; donde si a la línea j se le asigna el producto 0 quiere decir que la línea j esta parada.

t = índice del periodo de producción $(t=1,...,T)$.

s = índice del periodo de demanda $(s=1,..,T+1)$.

L_i = conjunto de líneas de producción en las que puede planificarse para ser producido el producto i .

P_j = conjunto de productos que pueden planificarse en la línea de producción j. $(0 \in P_j \ \forall j)$.

c_{ij} = costo unitario de producción del producto i en la línea j.

h_i = fin del periodo de mantenimiento del costo para el producto i .

q_{ij} = costo de cambiar el producto i a la línea j, independientemente de cual sea el producto producido previamente al producto i en la línea j (si $i=0$ y la línea no estaba parada, entonces se pone q_{0j} suficientemente grande para que sea prohibitivo)

p_{ij} = radio de producción, es decir, número de unidades del producto i producidas en la línea j durante un periodo de tiempo. Obviamente $p_{0j}=0 \ \forall j$.

d_{is} = la demanda del producto i durante el periodo s. Se asume que la demanda al final del periodo de tiempo se toma arbitrariamente como $d_{iT+1} = max_j \, p_{ij}$.

c_{ijts} = costo total de manufactura en la línea j durante el periodo t, manteniendo hasta el final del periodo s la demanda para el producto i , es decir, $c_{ijts} = (c_{ij} + (s - t)\, h_i)\, d_{is}$.

Se definen las variables de decisión:

x_{ijts} = fracción de la demanda para el producto i en el periodo s producida durante el periodo t en la línea j.

x_{ijtT+1} = holgura, en la restricción de la capacidad; es decir, unidades sobrantes del producto *i* producidas en la línea *j* desde el periodo *t* hasta el periodo *T+1* después de satisfacer la demanda para el producto i desde el periodo *t* al *T*.

$y_{ijt} = 1$, si el producto *i* se produce en la línea de producción *j* durante el periodo *t* ($y_{0jt} = 1$ quiere decir que *j* esta parada durante el periodo *t*)
$= 0$, en otro caso.

$v_{ijt} = 1$, si en la línea *j* la producción cambia al producto *i* al comienzo del tiempo *t*
$= 0$, otro caso.

y el problema se establece como:

$$v(CSP) = min \sum_{j=1}^{L} \sum_{i \in P_j} \sum_{t=1}^{T} \left\{ \sum_{s=1}^{T+1} c_{ijts} x_{ijts} + q_{ij} v_{ijt} \right\} \qquad (1)$$

s.a.:

$$\sum_{j \in L_i} \sum_{t=1}^{s} x_{itts} = 1 \qquad \forall i,\ 1 \le s \le T \qquad (2)$$

$$\sum_{s=t}^{T+1} d_{is} x_{ijts} = p_{ij} y_{ijt} \qquad \forall i,j,t \qquad (3)$$

$$\sum_{i \in P_j} y_{ijt} = 1 \qquad \forall j,t \qquad (4)$$

$$v_{ijt} \ge y_{ijt} - y_{ijt-1} \qquad \forall i, j \in L_i, t \qquad (5)$$

$$0 \le x_{ijts} \le 1 \qquad \forall i,j,t,s \qquad (6)$$

$$v_{ijt} \ge 0 \qquad \forall i, j, t \qquad (7)$$

$$0 \le y_{ijt} \le 1 \qquad \forall i, j, t \qquad (8)$$

$$y_{ijt} \text{ enteros} \qquad \forall i, j, t \qquad (9)$$

El conjunto de restricciones (2) indican que debe satisfacerse la demanda del producto *i* en el periodo *s*. (3) indican que cuando la producción del producto *i* se hace en la línea *j* durante el periodo de tiempo *t*, exactamente p_{ij} unidades del

producto se deben producir. (4) marca el hecho de que en cada línea de producción j y en cada periodo t sólo puede estar produciéndose un único producto. Las restricciones (5) indican que ocurre un cambio cuando $y_{ijt} = 1$ y $y_{ijt-1} = 0$.

Una cota inferior para el problema *CSP* puede calcularse mediante relajación lagrangiana (Geoffrion, 1974). Al dualizar las restricciones de la demanda (2) usando los multiplicadores u_{is}, obtenemos el problema lagrangiano *LR*

$$v(LR) = min \sum_{j=1}^{L} \sum_{i \in P_j} \sum_{t=1}^{T} \left\{ \sum_{s=1}^{T+1} c_{ijts} x_{ijts} + q_{ij} v_{ijt} \right\} + \sum_{i=1}^{p} \sum_{s=1}^{T} u_{is} \left(1 - \sum_{j \in L_i} \sum_{t=1}^{s} x_{ijts} \right)$$

sujeto a las restricciones (3), (4), (5), (6), (7), (8), y (9).

La función objetivo también puede escribirse en la forma:

$$v(LR) = min \sum_{j=1}^{L} \sum_{i \in P_j} \sum_{t=1}^{T} \left\{ \sum_{s=1}^{T+1} (c_{ijts} - u_{is}) x_{ijts} + C_{jT+1} x_{ijT+1} + q_{ij} v_{ijt} \right\} + \sum_{i=1}^{p} \sum_{s=1}^{T} u_{is}$$

Al resolver este problema se obtiene una cota inferior para el problema *CSP*.

2.7. Modelos de Trick (1994)

Muchos modelos tradicionales de planificación y secuenciación determinística suponen que el tiempo que un trabajo requiere en una máquina no esté bajo control, es decir, el tiempo de proceso es un dato fijado inicialmente. Sin embargo, en algunos problemas, no sólo se requiere la asignación de los trabajos a las máquinas, sino que además deben elegirse tiempos de proceso, reflejando las capacidades físicas de las máquinas. En este sentido, Trick (1994) trata el problema en el que se debe elegir la velocidad de la máquina. Considera dos tipos de modelos de múltiples máquinas con velocidad variable.

En el primer tipo de modelo, la capacidad de cada máquina se fija en el transcurso de la planificación. Se tienen n trabajos (indizados por j) y m máquinas (indizadas por i). No se asume que las máquinas sean idénticas, pero sí de características similares. Cada trabajo debe asignarse a las máquinas. Si el trabajo j se asigna a la máquina i, la máxima cantidad de tiempo que la máquina utiliza es u_{ij} incurriendo en un costo de c_{ij}. El trabajo j al menos precisa l_{ij} unidades de tiempo para ser procesado por la máquina i. Realizar los trabajos a una velocidad mayor origina un costo mayor. Así, por cada unidad de tiempo que el

procesamiento del trabajo j en la máquina i dure menos de u_{ij} se incurre en un costo no negativo s_{ij} (el costo de la velocidad). La máquina i está disponible b_i unidades de tiempo. Cada trabajo se asigna a exactamente una máquina, es decir, los trabajos son indivisibles. El objetivo es minimizar el costo total.

Una formulación de este problema podría ser:

$$min \ \sum_i \sum_j c_{ij} z_{ij} + \sum_i \sum_j s_{ij} (u_{ij} z_{ij} - x_{ij})$$

s.a.:

$$\sum_j x_{ij} \leq b_i \qquad \forall i$$

$$\sum_i z_{ij} = 1 \qquad \forall j$$

$$l_{ij} z_{ij} \leq x_{ij} \leq u_{ij} z_{ij} \qquad \forall i,j$$

$$x_{ij} \geq 0 \qquad \forall i,j$$

$$z_{ij} \geq 0 \quad enteros \quad \forall i,j$$

donde z_{ij} se interpreta como la fracción del trabajo j asignada a la máquina i , y x_{ij} es la cantidad de tiempo que el trabajo j se está procesando en la máquina i .

Esta formulación resulta adecuada para las técnicas estándar de ramificación y acotación. Trick, al proponer un método heurístico para resolver el problema mediante una relajación lineal, prefiere apoyarse en la siguiente formulación alternativa con las variables de decisión y_{ij} e y'_{ij}.

$$min \ \sum_i \sum_j c_{ij} y'_{ij} + \sum_i \sum_j (c_{ij} + s_{ij} (u_{ij} - l_{ij})) y_{ij}$$

s.a.:

$$\sum_j (l_{ij} y_{ij} + u_{ij} y'_{ij}) \leq b_i \qquad \forall i$$

$$\sum_i (y_{ij} + y'_{ij}) = 1 \qquad \forall j$$

$$y_{ij} + y'_{ij} \in \{0, 1\} \qquad \forall i,j$$

$$y_{ij} , y'_{ij} \geq 0 \qquad \forall i,j$$

donde los valores y_{ij} se interpretan como la fracción del trabajo j que se realiza usando el mínimo tiempo en la máquina i ; e y'_{ij} es la fracción usando el máximo tiempo.

Como un trabajo que utilice una cantidad de tiempo intermedia puede entenderse como que cierta porción del trabajo usa el tiempo máximo y el resto se realiza usando el tiempo mínimo, los dos modelos anteriores son equivalentes.

En el segundo tipo de modelos la cantidad de tiempo disponible de las máquinas no se conoce a priori. Se trata de planificar los trabajos indivisibles en las máquinas de modo que se minimice la suma del tiempo total de completación de los trabajos y los costos operativos de realización de los mismos. Este problema ha sido considerado por Potts (1985), Lenstra, Shmoys y Tardos (1990) sin variables de velocidad y sin costos operativos de realización.

Manteniendo la notación anterior, y considerando variables de velocidad y costos de realización, el modelo puede formularse en la forma:

$$min\ k + \sum_i \sum_j c_{ij}\, z_{ij} \ + \ \sum_i \sum_j s_{ij}\, (u_{ij}\, z_{ij} - x_{ij})$$

$$\sum_j x_{ij} \ \leq k \qquad \forall i$$

$$\sum_i z_{ij} \ = \ 1 \qquad \forall j$$

$$l_{ij}\, z_{ij} \leq x_{ij} \leq u_{ij}\, z_{ij} \qquad \forall i,j$$

$$x_{ij} \ \geq 0 \quad enteros \qquad \forall i,j$$

donde para un valor fijo de k se tiene el primero de los modelos del tipo anterior.

Sobre los modelos propuestos, Trick (1994) elabora heurísticas para resolver los problemas correspondientes.

2.8. Modelo de Mc Cormick y Pinedo (1995)

Los autores consideran el problema de planificar n trabajos con restricciones de precedencias en m máquinas uniformes, es decir, idénticas salvo en la velocidad. El trabajo j tiene tiempo de proceso p_j, y la máquina i tiene velocidad s_i. Si $m > n$, pueden eliminarse las $m - n$ máquinas más lentas y entonces puede suponerse en el modelo que $m \leq n$. Se supone que los índices de las máquinas están ordenados de modo que $s_1 \geq s_2 \geq ... \geq s_m > 0$ (de la más rápida a la más lenta) y que los índices de los trabajos están ordenados $0 < p_1 \leq p_2 \leq ... \leq p_n$ (del más corto al más largo). En el modelo se contempla la posibilidad de interrumpir el procesamiento de los trabajos. Se está interesado en encontrar los

óptimos de Pareto o puntos no dominados para los objetivos $\sum C_j$ y $C_{max}=max\{C_j\}$ (tiempo de proceso y tiempo de finalización de todos los trabajos).

Mc Cormick y Pinedo presentan un algoritmo polinomial que produce una planificación con $\sum C_j$ mínimo sujeto a que el valor de C_{max} es inferior a cierta cota D. Su algoritmo alterna entre la regla *SPT-FM* ("shortest processing time on fastest machine") y la regla *LRPT-FM* ("longest remaining processing time on fastest machine"). Los autores estudian como cambia la conducta del algoritmo cuando D varía paramétricamente. El conocimiento de la estructura de las planificaciones óptimas permite caracterizar los puntos de ruptura en la función lineal a trozos que une todos los puntos de Pareto. Como se permiten interrupciones existen a lo sumo $mn - \binom{m+1}{2} = O(mn)$ puntos de cambio de las pendientes de los segmentos de recta que unen los puntos eficientes. El algoritmo que proponen, algoritmo *P-SPT-FM-D*, ("parametric SPT-FM with deadline") puede calcular un nuevo punto de cambio en tiempo medio del $O(m^2)$. Por tanto, *P-SPT-FM-D* puede calcular todos los puntos de Pareto en un tiempo del $O(m^3n)$.

Si se ponderan los objetivos, se tiene el problema

$$min \quad \sum C_j + w\, C_{max}$$

s.a.:

$$w \geq 0 \qquad (1)$$

entonces sólo se necesitan encontrar los puntos de cambio w entre dos pendientes adyacentes w_1 y w_2 tales que $w \in [w_1, w_2]$.

Si se quiere

$$min \quad \sum C_j$$

s.a.:

$$C_{max} \leq D \qquad (2)$$

se encuentran los puntos de cambio adyacentes cuyos valores C_{max} alcanzan a D, (donde D es una fecha límite común) y encuentran el correspondiente valor $\sum C_j$ por interpolación. Puede también hacerse el razonamiento contrario si se prefiere, esto es

$$min \quad C_{max}$$

s.a.:

$$\sum C_j \leq nF \qquad (3)$$

donde F es una cota superior al tiempo medio de permanencia de los trabajos. También pueden considerarse los problemas

$$min \quad \Sigma C_j$$

s.a.:

$$C_{max} = C^*_{max} \qquad (4)$$

y

$$min \quad C_{max}$$

s.a.:

$$\Sigma C_j = \Sigma C^*_j \qquad (5)$$

donde C^*_{max} y ΣC^*_j son los valores óptimos de las funciones objetivo respectivas.

En la práctica, el decisor puede no estar satisfecho con ninguna de las soluciones dadas por (1)-(5). En estos casos, el algoritmo que proponen Mc Cormick y Pinedo genera información adicional de sensitividad que permite al decisor conocer como cambian los valores de los objetivos (1)-(5) cuando *w, D,* ó *F* cambian. De hecho, pueden predecir localmente como cambiará cada C_j (ó *w, D*, ó *F*) en un rango de variación de p_j mediante un análisis de sensitividad clásico.

Su aproximación genera una curva de puntos de Pareto enfocando primero su atención en el problema (2) ó problema *UMD* ("uniform machines with deadline"). Este problema puede entenderse como un problema de transporte considerando el trabajo *j* como una fuente de la que salen p_j unidades, y la máquina *i* como un sumidero al que llegan s_iD unidades. Desafortunadamente esta formulación no contempla la posibilidad de que varias partes de un mismo trabajo se realicen simultáneamente. Los autores desarrollan un algoritmo *SPT-FM* y *LRPT-FM* que alterna entre ambas reglas para resolver *UMD*. Luego investigan como cambia la solución dada por su algoritmo al variar *D* construyendo así el algoritmo *P-SPT-FM-D* para determinar los óptimos de Pareto del problema.

Para concluir el presente epígrafe, digamos que en la resolución de cada uno de estos modelos los autores aplican diferentes técnicas de programación matemática tales como ramificación y acotación, planos de corte, relajación lagrangiana, algoritmos heurísticos, etc. El estudio detallado de todas estas técnicas iría en una dirección distinta a los objetivos de la presente memoria.

Veremos a continuación, y para finalizar este capítulo, el análisis de un problema de planificación real correspondiente a un taller de reparaciones.

3. Modelización de un taller de reparaciones de audio, video y televisión

Planteamiento del Problema

En este epígrafe se estudia un caso práctico donde se plantean problemas de planificación. Se trata de un taller que se dedica a la reparación de videos, televisores y equipos de alta fidelidad en la isla de Tenerife.

Se pretende construir un modelo lo más representativo y adecuado posible que planifique la administración de recursos y mano de obra para la ejecución de las tareas de reparación de los aparatos de audio-video-tv que entran al taller.

Descripción del taller

El taller realiza reparaciones tanto en sus dependencias como a domicilio. Hay 6 personas dedicadas a reparaciones en el propio taller y otras 6 dedicadas a reparaciones en domicilio. Al ser un taller oficial de una conocida marca de aparatos de audio, video y tv., y según la normativa de consumo de servicio oficial, esta obligado a mantener un servicio exterior de reparaciones y atenderlas antes de las 48 horas de su solicitud. Las averías de reparación que se atienden en el exterior son sólo de televisión. Si una de estas averías alcanza cierta complejidad de reparación se convierte necesariamente en una avería de taller dejando de ser domiciliaria.

Las seis personas destinadas a reparación en taller se "especializan" de la siguiente manera:

1	*técnico de audio / sonido*	*reparaciones equipos hi-fi*
3	*técnicos de televisión*	*reparaciones aparatos de televisión*
2	*técnicos de video (1 de ellos especialista en cámaras de video)*	*reparaciones aparatos de video y cámaras de video*

El especialista en video cámaras hace las veces de Jefe de Servicio Técnico y es el encargado de distribuir el trabajo conforme va llegando al taller. Existe una séptima persona, el Director del taller, especialista en reparaciones de aparatos con tecnología Láser Disk. Tanto el Jefe de Servicio Técnico como el Director son personas *"comodín"* en el sentido de que pueden afrontar la reparación de cualquier aparato que llegue al taller. En ocasiones uno de los técnicos de

televisión puede ayudar al técnico de audio / sonido. Cuando una avería se complica sobre manera se acude a una de las personas comodín.

El horario de taller es de 8 horas diarias, 5 días a la semana. De 8:30 a 13:30 y de 16:00 a 19:00; de lunes a viernes. Es decir, 40 horas semanales. La recepción y entrega de aparatos es continua, sin días prefijados. Generalmente, y debido simplemente a hábitos de los clientes, se suelen concentrar las recepciones los primeros días de la semana, y las entregas los últimos.

Los aparatos entran en el taller por orden de solicitud (recepción). La fecha de entrega del aparato reparado se establece en 2 días después de su recepción. Los aparatos con averías más sencillas se reparan primero. Generalmente las averías más complicadas son las que presentan los modelos de aparatos que se han incorporado al mercado más recientemente, ya que estas averías no están aún catalogadas, ni se tiene ninguna referencia directa de su modo de reparación.

Normalmente, el hecho de que la reparación de un aparato requiera repuestos no presenta problemas ya que el taller tiene almacenados repuestos en el 98 % de los casos. Cuando se deben pedir repuestos, comúnmente vía Madrid, el tiempo de espera suele ser de 15 a 30 días.

Tipos de averías, frecuencias y tiempos de proceso

En el modelo se van a considerar las averías de equipos de audio, de video, y de televisión. Se han dividido los aparatos en bloques, de modo que las averías que acontecen en las componentes de un bloque concreto suelen tener el mismo grado de dificultad y el mismo tiempo de proceso, independientemente de la componente de que se trate.

Frecuencias

Los bloques en los que estructuraremos las averías de video (VCR), televisión (TV), y audio (HIFI) y las frecuencias en que aparecen son las que a continuación se citan. Los datos hacen referencia al año 1994, y, según el criterio de las personas experimentadas en este tipo de reparaciones, pueden suponerse que se mantienen más o menos constantes a lo largo de los años.

Bloques de aparatos de video	***Frecuencia de averías***
Fuentes de alimentación y CPU	*20 %*
Mecánica y servocontroles	*28 %*
Placa señal	*5 %*
Conjunto ascensor	*40 %*
Placa de frecuencia intermedia (RF/FI)	*2 %*
Estéticas (teclados, etc.)	*5 %*

Bloques de aparatos de televisión	***Frecuencia de averías***
Fuentes de alimentación	*15 %*
Oscilador de líneas	*1 %*
Oscilador vertical	*7 %*
Circuitos corrección y convergencias	*8 %*
Salida vertical	*10 %*
MAT (Muy Alta Tensión)	*25 %*
FI (Frecuencia Intermedia)	*6 %*
Sintonizador	*6 %*
Presintonías	*1 %*
Audio	*5 %*
Circuito y Señal	*10 %*
Salida RGB	*6 %*

Bloques de aparatos HI - FI	***Frecuencia de averías***
Fuente de alimentación	*15 %*
Mecánica	*60 %*
Placa señal y display	*2 %*
Placa RFFI (Radio Frecuencia, Frecuencia Intermedia)	*4 %*
Procesamiento Digital	*3 %*
Grabación / Reproducción	*10 %*
Servocontroles	*6 %*

Tiempos de proceso

Los tiempos de proceso dependerán de si la arquitectura del aparato es modular o no, y de si la avería es una avería nueva o una avería ya conocida. Los aparatos modulares se desmontan y montan más rápidamente. Las averías nuevas requieren un tiempo de diagnóstico y localización normalmente superior.

Los tiempos de proceso, en términos aproximados y en minutos de las averías que acontecen en los diferentes bloques son los que a continuación se relacionan.

	Tiempos de Proceso (minutos)			
	Primera Vez		***Sucesivas Veces***	
	Modulares	***No Modulares***	***Modulares***	***No Modulares***
Bloques de aparatos de video				
Fuentes de alimentación y CPU	*40*	*de 40 a 180*	*15*	*15*
Mecánica y servocontroles	*120*	*de 120 a 180*	*70*	*70*
Placa señal	*no existen*	*reparar: de 30 a 180 cambiar: 15*	*no existen*	*20*
Conjunto ascensor	*30*	*30*	*30*	*30*
Placa frecuencia intermedia (RF/FI) (algunas irreparables)	*reparar: 90 cambiar: 15*	*no existen*	*reparar: 90 cambiar: 15*	*no existen*
Estéticas (teclados, etc.)	*reparar: 40 a 50 cambiar: 15*	*reparar: 40 a 50 cambiar: 15*	*reparar: 40 a 50 cambiar: 15*	*reparar: 40 a 50 cambiar: 15*

	Tiempos de Proceso (minutos)			
	Primera Vez		***Sucesivas Veces***	
	Modulares	***No Modulares***	***Modulares***	***No Modulares***
Bloques aparatos televisión				
Fuente de alimentación	*40*	*de 40 a 180*	*15*	*15*
Oscilador de líneas	*25*	*25*	*25*	*25*
Oscilador vertical	*de 20 a 25*	*de 25 a días*	*de 20 a 25*	*de 25 a días*
Circuitos de corrección y de convergencias	*20*	*40*	*20*	*20*
Salida vertical	*15*	*50*	*15*	*30*
MAT (Muy Alta Tensión)	*15*	*de 40 a 180 ó más*	*15*	*15*
FI (Frecuencia Intermedia)	*reparar: 90 cambiar: 15*	*120*	*reparar: 90 cambiar: 15*	*30*
Sintonizador	*irreparable cambiar: 15*	*irreparable cambiar: 40*	*irreparable cambiar: 15*	*irreparable cambiar: 15*
Presintonías	*40*	*de 40 a 180*	*15*	*15*
Audio	*20*	*de 20 a 100*	*20*	*20*
Circuito y Señal	*de 30 a 40*	*de 30 a días*	*20*	*20*
Salida RGB	*20*	*de 20 a 60*	*de 15 a 20*	

	Tiempos de Proceso (minutos)			
	Primera Vez		***Sucesivas Veces***	
	Modulares	***No Modulares***	***Modulares***	***No Modulares***
Bloques de aparatos HI -FI				
Fuente de alimentación	*40*	*de 40 a 180*	*15*	*15*
Mecánica	*120*	*de 120 a 180*	*70*	*70*
Placa señal y display	*no existen*	*reparar: de 30 a 180 cambiar: 15*	*no existen*	*20*
Placa RFFI (algunas irreparables) (Radio Frecuencia, Frecuencia Intermedia)	*reparar: 90 cambiar: 15*	*no existen*	*reparar: 90 cambiar: 15*	*no existen*
Procesamiento Digital	*de 40 a 60*	*de 40 a 60*	*40*	
Grabación / Reproducción	*no existen*	*de 40 a días*	*no existen*	
Servocontroles	*no aparecen*	*de 60 a 120*	*no aparecen*	*de 60 a 120*

Precedencias

En el caso de que en un mismo aparato deban realizarse dos ó más reparaciones, éstas deben seguir una secuencia lógica para su reparación. Así, por ejemplo, si un televisor tiene avería en su fuente de alimentación y en su sintonizador, es claro que primero habrá que reparar su fuente de alimentación y, posteriormente, comprobar si dicho televisor sintoniza bien. Por tanto, se establecen unas precedencias entre las averías a reparar. Quizás sea mejor utilizar el término *jerarquías* en lugar de precedencias. Dichos niveles de jerarquías o precedencias vienen dados por las tablas siguientes:

Nivel	***Bloques de aparatos de video***	***Número Bloque***
1	*Fuentes de alimentación y CPU*	*1*
2	*Conjunto ascensor*	*2*
3	*Mecánica y servocontroles*	*3*
4	*Placa señal*	*4*
5	*Placa frecuencia intermedia (RF/FI)*	*5*
6	*Estéticas (teclados, etc.)*	*6*

Nivel	***Bloques de aparatos de televisión***	***Número Bloque***
1	*Fuente de alimentación*	*1*
2	*Oscilador de líneas*	*2*
3	*MAT (Muy Alta Tensión)*	*3*
4	*Salida RGB*	*4*
5	*Oscilador vertical*	*5*
5	*Salida vertical*	*6*
6	*Circuito y Señal*	*7*
7	*FI (Frecuencia Intermedia)*	*8*
8	*Presintonías*	*9*
8	*Sintonizador*	*10*
9	*Audio*	*11*
10	*Circuitos corrección y convergencias*	*12*

Nivel	***Bloques de aparatos HI - FI***	***Número Bloque***
1	*Fuente de alimentación*	*1*
2	*Mecánica*	*2*
3	*Procesamiento Digital*	*3*
4	*Servocontroles*	*4*
5	*Grabación / Reproducción (originados por 1 en su mayoría)*	*5*
6	*Placa señal y display*	*6*
7	*Placa RFFI (Radio Frecuencia, Frecuencia Intermedia)*	*7*

Posibles modelos para la resolución del problema

En el momento de modelizar un problema concreto deben respetarse su estructura y sus restricciones correspondientes. Si el problema es muy específico, puede ocurrir que ninguno de los modelos existentes para resolver problemas similares siga siendo válido para el problema en cuestión. En este ejemplo concreto se ha optado por realizar una modelización específica del problema.

a) Construcción de un modelo de Programación Entera 0-1

Para la construcción del modelo dividimos el problema en secciones independientes y planteamos un modelo para cada sección. Así, tenemos: la sección de televisión, para la cual disponemos de tres técnicos, la sección de video en la que hay dos técnicos, y la sección de audio con un único técnico. Para cada aparato consideramos tantos tipos de averías posibles como bloques característicos del aparato. Se estima que los tiempos de reparación de las averías de un determinado bloque son los mismos y en ese sentido las averías dentro de un mismo bloque pueden ser tratadas como idénticas. Se tolera la posible interrupción en la reparación de cualquier avería y se modelizan las precedencias entre sus reparaciones. Sin pérdida de generalidad se toma el tiempo como una variable discreta medida en unidades tan pequeñas y precisas como queramos. Hablándose del instante t donde $t \in \{0,1,...,T\}$ siendo T el horizonte temporal que puede fijarse por ejemplo a una semana, un mes, un trimestre, un año, etc.

Datos del Problema

De acuerdo a los datos recogidos en los epígrafes anteriores, sean $m_{TV} = 3$, $m_A = 1$, $m_V = 2$ el número de técnicos que se disponen para arreglar las averías de los n_{TV} televisores, n_A aparatos de audio, y n_V aparatos de video que se presentan en el taller, respectivamente. Cada televisor puede presentar, como vimos, a lo sumo $t_{TV} = 12$ tipos de averías distintas, mientras que cada equipo de audio puede presentar $t_A = 7$ y cada video a lo sumo $t_V = 6$ averías distintas.

Propondremos a continuación una expresión general para formular el problema de cada sección de modo independiente. Supondremos que fijada la avería i del aparato j, cualquier técnico k del conjunto de m_X técnicos capaces de arreglarla tarda el mismo tiempo en resolverla, es decir $p_{ijk} = p_{ij} \quad \forall k = 1,\ldots,m_x$, siendo p_{ij} el tiempo requerido para reparar la avería i en el aparato j; y $m_X \in \{m_{TV}, m_A, m_V\}$ viene determinado por el tipo de aparato que sea j. La suposición de los

tiempos de proceso viene motivada por la información dada por el propio taller. Dichos tiempos dependerán de sí la avería es modular o no y de si es la primera vez o no que se presenta la avería. Su valor será 0 cuando el aparato *j* no presente la avería *i*.

Otros datos asociados al problema son los siguientes: el instante r_j en el que se presenta el aparato *j* en el taller y la fecha límite, $d_j = r_j + 48\ horas$, en la que todas las averías existentes en el aparato deben estar reparadas.

Las precedencias existentes entre las t_X averías, con $t_X \in \{t_{TV}, t_A, t_V\}$, vienen dadas por los grafos dirigidos acíclicos $G_X=(V_X, A_X)$ que veremos a continuación. Los conjuntos de vértices son los tipos de averías $V_X=\{1,...,m_X\}$, y los arcos vienen dados por A_X, donde $X \in \{TV, A, V\}$ representa la sección en la que nos encontramos.

Para la sección de televisión:

$$V_{TV} = \{1, 2, 3, 4, 5, 6\}$$
$$A_{TV} = \{(1,2), (2,3), (3,4), (4,5), (4,6), (5,7), (6,7), (7,8), (8,9), (8,10), (9,11), (10,11), (11,12)\}.$$

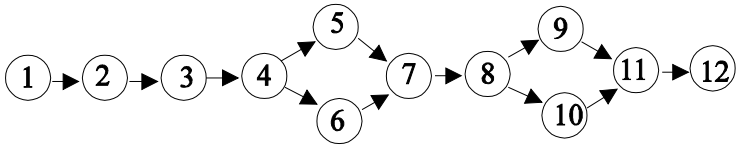


Figura 1. Grafo $G_{TV} = (V_{TV}, A_{TV})$ de precedencias entre averías de televisión.

Para la sección de audio:

$$V_A = \{1, 2, 3, 4, 5, 6, 7\}$$
$$A_A = \{(1,2), (2,3), (3,4), (4,5), (5,6), (6,7)\}.$$

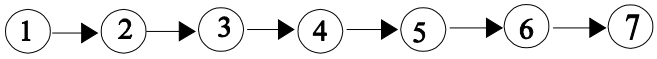


Figura 2. Grafo $G_A = (V_A, A_A)$ de precedencias entre averías de audio.

Y para la sección de video:

$$V_V = \{1, 2, 3, 4, 5, 6\}$$
$$A_V = \{(1,2), (2,3), (3,4), (4,5), (5,6)\}.$$

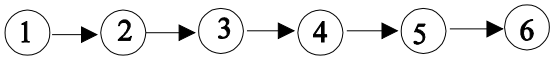


Figura 3. Grafo $G_V = (V_V, A_V)$ de precedencias entre averías de video.

Variables de Decisión

Las variables de decisión enteras 0-1 que podemos considerar en el modelo asociado a la sección $X \in \{TV, A, V\}$ son las que a continuación se indican. Los índices varían de la forma siguiente: $i = 1,...,t_X$ índice de averías; $j = 1,...,n_X$ índice de aparatos a arreglar; $k = 1,...,m_X$ índice de técnicos; $t = 0,1,...,T$ índice de instantes de tiempo.

$$x_{ijkt} = \begin{cases} 1, & \textit{si en el instante t el tecnico k esta reparando} \\ & \textit{la averia i en el aparato j} \\ 0, & \textit{otro caso} \end{cases} \qquad \forall i, \forall j, \forall k, \forall t$$

$$y_{jt} = \begin{cases} 1, & \textit{si en instante t aun no se ha completado} \\ & \textit{la reparacion del aparato j} \\ 0, & \textit{otro caso} \end{cases} \qquad \forall j, \forall t$$

$$z_{ijt} = \begin{cases} 1, & \textit{si en el instante t aun no se ha completado} \\ & \textit{la averia i del aparato j} \\ 0, & \textit{otro caso} \end{cases} \qquad \forall i, \forall j, \forall t$$

Las variables x_{ijkt} son variables indicadoras de actividad del técnico k en el instante t sobre la avería i del aparato j.

Las variables y_{jt} son variables indicadoras de las condiciones

$$\sum_{i=1}^{t_X} \sum_{k=1}^{m_X} \sum_{t_1=0}^{t} x_{ijkt_1} < \sum_{i=1}^{t_X} p_{ij} \qquad \forall j, \forall t$$

que son ciertas cuando en el instante t aún no se ha completado la reparación del aparato j.

Las variables z_{ijt} son las variables indicadoras de las condiciones

$$\sum_{k=1}^{m_X} \sum_{t_1=0}^{t} x_{ijkt_1} < p_{ij} \qquad \forall i, \forall j, \forall t$$

que son ciertas cuando en el instante t aún no se ha reparado la avería i del aparato j.

Nótese que existe la siguiente relación no lineal entre las variables:

$$y_{jt}=max_{i=1,...,t_X}\{z_{ijt}\} \quad \forall j, \forall t.$$

Objetivos

Los objetivos que pueden considerarse en el problema podrían ser funciones regulares en los tiempos de completación de las reparaciones de los aparatos. Estos tiempos vienen dados por:

$$C_j = \sum_{t=0}^{T} y_{jt} \qquad \forall j$$

Así, por ejemplo, $F_j = C_j - r_j$ denota el tiempo de permanencia del aparato j; $L_j = C_j - d_j$ la demora en su reparación; $T_j = max\ \{0, L_j\}$ la tardanza en su reparación, etc. Pudiendo tomarse por objetivos a $\sum_{j=1}^{n} C_j$, $\sum_{j=1}^{n} F_j$, $\sum_{j=1}^{n} T_j$, a los correspondientes ponderados $\sum_{j=1}^{n} \omega_j C_j$, $\sum_{j=1}^{n} \omega_j F_j$, $\sum_{j=1}^{n} \omega_j T_j$, ó bien a $C_{max}=max_j\{C_j\}$.

Restricciones del Modelo

En el modelo deben considerarse la siguiente familia de restricciones:

$$\sum_{i=1}^{t_X}\sum_{j=1}^{n_X} x_{ijkt} \le 1 \qquad \forall k, \forall t \qquad (1)$$

$$\sum_{i=1}^{t_X}\sum_{k=1}^{m_X} x_{ijkt} \le 1 \qquad \forall j, \forall t \qquad (2)$$

$$\sum_{k=1}^{m_X}\sum_{t=0}^{T} x_{ijkt} = p_{ij} \qquad \forall i, \forall j \qquad (3)$$

$$\sum_{i=1}^{t_X}\sum_{k=1}^{m_X}\sum_{t_1=0}^{t} x_{ijkt_1} - \sum_{i=1}^{t_X} p_{ij} < 1 - y_{jt} \qquad \forall j, \forall t \qquad (4)$$

$$\sum_{i=1}^{t_X}\sum_{k=1}^{m_X}\sum_{t_1=0}^{t} x_{ijkt_1} + \left(\sum_{i=1}^{t_X} p_{ij}\right) y_{jt} \ge \sum_{i=1}^{t_X} p_{ij} \qquad \forall j, \forall t \qquad (5)$$

$$\sum_{k=1}^{m_X}\sum_{t_1=0}^{t} x_{ijkt_1} - p_{ij} < 1 - z_{ijt} \qquad \forall i, \forall j, \forall t \qquad (6)$$

$$\sum_{k=1}^{m_X}\sum_{t_1=0}^{t} x_{ijkt_1} + p_{ij} z_{ijt} \ge p_{ij} \qquad \forall i, \forall j, \forall t \qquad (7)$$

$$1 - z_{i_1 jt} - x_{i_2 jkt} \geq 0 \qquad \forall (i_1, i_2) \in A_X \quad \forall j, \forall k, \forall t \quad (8)$$

Cuyo significado es el siguiente:

- (1) fijado el instante *t,* el técnico *k* no puede estar arreglando más de una avería en cualquier aparato. Son $m_X(T+1)$ restricciones.
- (2) fijado el instante *t*, a lo sumo puede estar trabajando un único técnico en el aparato *j*. Son $n_X(T+1)$ restricciones.
- (3) fijado el aparato *j*, el número total de instantes en que algún técnico está reparando la avería *i* es exactamente igual al tiempo que se tarda en reparar dicha avería. Son $t_X n_X$ restricciones.
- (4) obliga a que $y_{jt}=1 \Leftrightarrow$ en el instante *t* ***no*** se ha completado la reparación del aparato *j*. Son $n_X(T+1)$ restricciones.
- (5) obliga a que $y_{jt}=0 \Leftrightarrow$ en el instante *t* ***ya*** se ha completado la reparación del aparato *j*. Son $n_X(T+1)$ restricciones.
- (6) obliga a que $z_{ijt}=1 \Leftrightarrow$ en el instante *t* ***no*** se ha completado la reparación de la avería *i* del aparato *j*. Son $t_X n_X(T+1)$ restricciones.
- (7) obliga a que $z_{ijt}=0 \Leftrightarrow$ en el instante *t* ***ya*** se ha completado la reparación de la avería *i* del aparato *j*. Son $t_X n_X(T+1)$ restricciones.
- (8) garantiza que se respetan las relaciones de precedencias. Son $39n_{TV}(T+1)$, $6n_A(T+1)$, ó bien $10n_V(T+1)$ restricciones dependiendo de la sección en la que nos encontremos.

Nótese que si no existiesen relaciones de precedencias entre las averías podríamos prescindir de las restricciones (6), (7), y (8) y de las variables z_{ijt}.

Sí, como la realidad nos invita, tenemos que considerar relaciones de precedencia, podemos aprovechar la relación existente entre las variables z_{ijt} y las y_{jt} y sustituir las $2n_X(T+1)$ restricciones (4), (5) por las $2n_X(T+1)$ restricciones (A), y (B) siguientes:

$$y_{jt} \leq \sum_{i=1}^{t_X} z_{ijt} \qquad \forall j, \forall t \qquad \text{(A)}$$

$$\sum_{i=1}^{t_X} z_{ijt} - t_X y_{jt} \leq 0 \qquad \forall j, \forall t \qquad \text{(B)}$$

b) Construcción de un modelo de Programación Entera 0-1 Mixto

El modelo de Programación Entera 0-1 Puro tiene el inconveniente de que el número de variables crece considerablemente al tener la necesidad de

discretizar el tiempo e indizar en él. Por tanto, y como alternativa, se ha optado por construir otro modelo para el problema.

En este nuevo modelo no se divide el problema en secciones sino que se aborda de manera global. Para ello consideramos las siete personas que están en el taller, entre las que contamos las personas *"comodín"* que son el Jefe de Servicio Técnico (JST) y el Director del Taller (DT). Las personas del taller son capaces de reparar ciertos aparatos de modo preferente y, ocasionalmente, pueden reparar otros. Estos datos se recogen en la siguiente tabla, en la que *P* denota "de modo preferente" y *O* denota "ocasionalmente":

Persona:		***Puede arreglar:***				
	Especialista en:	***Audio***	***Televisión***	***Video***	***Video cámaras***	***Laser Disk***
1	*audio*	*P*	*-*	*-*	*-*	*-*
2	*tv*	*O*	*P*	*-*	*-*	*-*
3	*tv*	*-*	*P*	*-*	*-*	*-*
4	*tv*	*-*	*P*	*-*	*-*	*-*
5	*video*	*-*	*-*	*P*	*-*	*-*
6 (JST)	*comodín*	*O*	*O*	*P*	*P*	*-*
7 (DT)	*comodín*	*O*	*O*	*O*	*O*	*P*

La información de la tabla anterior nos permite construir la siguiente tabla:

	Número de personas capaces de arreglarla:		
Avería:	***de modo preferente***	***ocasionalmente***	***total***
Audio	*1*	*3*	*4*
Televisión	*3*	*2*	*5*
Video	*2*	*1*	*3*
Video cámaras	*1*	*1*	*2*
Laser Disk	*1*	*0*	*1*

De acuerdo a esta información y teniendo en cuenta los grafos de precedencias ya expuestos y las fechas de disponibilidad r_j y fechas límite d_j podemos formular el problema de la siguiente manera:

Datos del problema

Denotemos por p_{ijk} el tiempo que requeriría el técnico k para reparar la avería i del aparato j. Dicho tiempo puede ser cero si el aparato j no presenta la

avería *i;* y podemos tomarlo a $+\infty$ ó a un valor M suficientemente grande para indicar que el técnico *k* no esta cualificado para arreglar la avería en cuestión. La variación de los índices es la siguiente: $k = 1,...,7$ dado que hay siete técnicos en el taller; $j = 1,...,n$ siendo *n* el número total de aparatos, de cualquier tipo y condición, que se presentan en el taller desde el instante 0 al instante *T,* con *T* el horizonte temporal estudiado. Y, finalmente, $i = 1,...,m_j$ donde m_j es el número de averías que presenta el aparato *j*. De esta forma *(i,j)* representa la avería *i* del aparato *j*.

En el caso de que una misma avería pueda ser reparada por varios técnicos podemos estar interesados en que su reparación la lleve a cabo, no necesariamente el técnico más hábil, sino otro de modo que el más hábil quede liberado para realizar otras funciones en el taller. Tal es el caso del Jefe de Servicio Técnico y el Director de Taller que, por su experiencia, repararían más velozmente ciertas averías pero deben reservar parte de su tiempo para desempeñar labores administrativas, de abastecimiento y de planificación del taller. Estas circunstancias pueden modelizarse con unas ponderaciones ω_{jk} que indicarían el "costo" o "penalización" asociada a la asignación de cualquier avería del aparato *j* al técnico *k.*

Variables de decisión

Consideramos en el modelo las siguientes variables enteras 0-1:

$$x_{ijk} = \begin{cases} 1, & \textit{si el tecnico k repara la averia i del aparato j} \\ 0, & \textit{otro caso} \end{cases} \qquad \forall i,\ \forall j,\ \forall k \qquad (1)$$

$$\delta_{i_1 j_1 i_2 j_2 k} = \begin{cases} 1, & \textit{si el tecnico k debe arreglar ambas averias } (i_1, j_1) \textit{ e } (i_2, j_2) \\ 0, & \textit{otro caso} \end{cases} \qquad (2)$$

$$\forall (i_1, j_1),\quad \forall (i_2, j_2),\quad \forall k$$

$$\delta^1_{i_1 j_1 i_2 j_2 k} = \begin{cases} 1, & \textit{si el tecnico k debe arreglar } (i_1, j_1) \textit{ antes de } (i_2, j_2) \\ 0, & \textit{otro caso} \end{cases} \qquad (3)$$

$$\forall (i_1, j_1),\quad \forall (i_2, j_2),\quad \forall k$$

$$\delta^2_{i_1 j_1 i_2 j_2 k} = \begin{cases} 1, & \textit{si el tecnico k debe arreglar } (i_2, j_2) \textit{ antes de } (i_1, j_1) \\ 0, & \textit{otro caso} \end{cases} \qquad (4)$$

$$\forall (i_1, j_1),\quad \forall (i_2, j_2),\quad \forall k$$

y las variables continuas:

s_{ij} = *instante de comienzo de la reparación de la avería i del aparato j* $\forall i, \forall j$ (5)

Las variables (1) determinan la asignación de averías a técnicos. Las variables (2) indican, para cada par de averías, si las tiene que realizar el mismo técnico o no. En caso afirmativo, las variables (3) y (4) fijan la secuenciación sobre el técnico para garantizar que éste no tenga que simultanear la ejecución de dichas tareas. Para cada avería existente hay una variable (5) que indica el instante en el cual se comienza su reparación. Supondremos que una vez comenzada la reparación de cierta avería, se lleva a término sin interrupciones. Las interrupciones se permiten, pues, en la reparación de los aparatos y no de las averías.

Formulación del Problema

Un modelo para el problema podría ser el siguiente:

$$min \quad \max_{1\le k\le 7} \sum_{j=1}^{n}\sum_{i=1}^{m_j} \omega_{jk} p_{ijk} x_{ijk} \tag{6}$$

sujeto a:

$$\sum_{k=1}^{7} x_{ijk} = 1 \quad \forall j = 1,\ldots,n \quad \forall i = 1,\ldots,m_j \tag{7}$$

$$\sum_{j=1}^{n}\sum_{i=1}^{m_j} p_{ijk} x_{ijk} \le T_k \quad \forall k = 1,\ldots,7 \tag{8}$$

$$x_{ijk} \in \{0,1\} \quad \forall k = 1,\ldots,7 \; \forall j = 1,\ldots,n \; \forall i = 1,\ldots,m_j \tag{9}$$

$$r_j \le s_{ij} \quad \forall j = 1,\ldots,n \; \forall i = 1,\ldots,m_j \tag{10}$$

$$s_{ij} + \sum_{k=1}^{7} p_{ijk} x_{ijk} \le s_{lj} \quad \forall j = 1,\ldots,n \; \forall (i,l) \in A_j \tag{11}$$

$$s_{ij} \ge 0 \quad \forall j = 1,\ldots,n \quad \forall i = 1,\ldots,m_j \tag{12}$$

$$\left.\begin{array}{l} x_{i_1 j_1 k} = 1 \\ x_{i_2 j_2 k} = 1 \end{array}\right\} \Rightarrow \begin{cases} s_{i_1 j_1} + p_{i_1 j_1 k} \le s_{i_2 j_2} \\ \quad o\ exclusivo \\ s_{i_2 j_2} + p_{i_2 j_2 k} \le s_{i_1 j_1} \end{cases} \quad \forall k{=}1,\ldots,7 \;\; \forall (i_1,j_1) \;\; \forall (i_2,j_2) \tag{13}$$

El objetivo (6) nos indica que buscamos una planificación donde el tiempo real penalizado empleado en las reparaciones sea mínimo.

Las restricciones (7) nos indican que cada avería la repara un único técnico. Ante el peligro de saturar de trabajo al técnico más hábil, las restricciones

(8) garantizan que ningún técnico estará realizando alguna reparación más de T_k instantes de tiempo. (9) expresan el carácter binario de las variables indicadoras de actividad de los técnicos.

Las restricciones (10) recogen el hecho de que se respetan los instantes de llegada. El aparato *j* será un aparato de televisión, de audio, de video, una video cámara ó tendrá tecnología láser. En cualquier caso, las averías que de él tengan que repararse tendrán una estructura de precedencias compatible con uno de los cinco grafos asociados. El grafo correspondiente es el A_j. Las restricciones (11) garantizan que se respetan estas precedencias. La no negatividad de los instantes de comienzo de las reparaciones viene garantizada por (10) y por (12).

La condición lógica (13) indica que si un determinado técnico debe arreglar dos averías distintas, entonces tendrá que hacerlo de modo no simultáneo. Esta condición puede modelizarse utilizando las variables indicadoras δ descritas anteriormente de la siguiente manera:

$$\left.\begin{array}{l} x_{i_1 j_1 k} + x_{i_2 j_2 k} - \delta_{i_1 j_1 i_2 j_2 k} \le 1 \\ -x_{i_1 j_1 k} - x_{i_1 j_1 k} + 2\delta_{i_1 j_1 i_2 j_2 k} \le 0 \end{array}\right\} \quad \begin{array}{l} \forall k = 1,\ldots,7 \quad \forall j_1 \ne j_2 = 1,\ldots,n \\ \forall i_1 = 1,\ldots,m_{j_1} \quad \forall i_2 = 1,\ldots,m_{j_2} \end{array} \tag{14}$$

$$\left.\begin{array}{l} s_{i_1 j_1} + p_{i_1 j_1 k} - s_{i_2 j_2} \le M(1-\delta^1_{i_1 j_1 i_2 j_2 k}) \\ s_{i_2 j_2} + p_{i_2 j_2 k} - s_{i_1 j_1} \le M(1-\delta^2_{i_1 j_1 i_2 j_2 k}) \end{array}\right\} \quad \begin{array}{l} \forall k = 1,\ldots,7 \quad \forall j_1 \ne j_2 = 1,\ldots,n \\ \forall i_1 = 1,\ldots,m_{j_1} \quad \forall i_2 = 1,\ldots,m_{j_2} \end{array} \tag{15}$$

$$\delta^1_{i_1 j_1 i_2 j_2 k} + \delta^2_{i_1 j_1 i_2 j_2 k} = \delta_{i_1 j_1 i_2 j_2 k} \quad \begin{array}{l} \forall k = 1,\ldots,7 \quad \forall j_1 \ne j_2 = 1,\ldots,n \\ \forall i_1 = 1,\ldots,m_{j_1} \quad \forall i_2 = 1,\ldots,m_{j_2} \end{array} \tag{16}$$

$$\delta^1_{i_1 j_1 i_2 j_2 k}, \delta^2_{i_1 j_1 i_2 j_2 k}, \delta_{i_1 j_1 i_2 j_2 k} \in \{0,1\} \quad \begin{array}{l} \forall k = 1,\ldots,7 \quad \forall j_1 \ne j_2 = 1,\ldots,n \\ \forall i_1 = 1,\ldots,m_{j_1} \quad \forall i_2 = 1,\ldots,m_{j_2} \end{array} \tag{17}$$

Donde cada par de restricciones (14) determina el valor de δ que será 1 sólo sí el par de averías correspondiente debe ser reparado por el mismo técnico. En ese caso, (16) y (17) obligan a que una única de las variables δ^1, ó bien δ^2 sea 1 y la otra 0. Mirando el par de restricciones asociado en (15) se tiene determinado que avería debe realizarse primero por el técnico y cual después. En caso contrario, si δ es 0, (16) y (17) obligan a que las otras variables sean también nulas, con lo que las restricciones de (15) no son tales puesto que *M* es una constante suficientemente grande.

Digamos por último, que para resolver los problemas planteados podría utilizarse alguno de los métodos específicos para resolver problemas de Programación Matemática y apoyarse en el software existente.

Apéndice A

Apéndice A: Experiencia computacional para los problemas sobre una máquina.

En el capítulo II se ha estudiado el problema $1||\sum T_j$ comentando que es un problema NP-duro (Du y Leung (1990)), y se han citado diferentes algoritmos exactos existentes en la literatura. También se propuso un método heurístico para resolverlo.

En el presente apéndice se realiza un estudio computacional comparativo de los algoritmos exactos citados, el algoritmo heurístico propuesto y otros algoritmos heurísticos existentes. Para ello se han codificado los algoritmos en lenguaje de programación C y se han obtenido tiempos en un ordenador personal 80486 DX a 50 Mhz.

Los problemas test utilizados son los mismos problemas generados por Potts y van Wassenhove (1982). De acuerdo con este generador, los tiempos de proceso de cada trabajo se generan de la *U[1,100]*. Posteriormente se generan las fechas límite d_j , para lo cual se evalúa la suma de los tiempos de proceso $P = \sum p_j$ y se obtienen dichas fechas límite de la $U\left[P(1 - TF - \frac{RDD}{2}),\ P(1 - TF + \frac{RDD}{2}) \right]$ donde los parámetros *TF* y *RDD* varían en los conjuntos de valores {0.2, 0.4, 0.6, 0.8, 1.0} y {0.2, 0.4, 0.6, 0.8} respectivamente. Por tanto, para cada valor de *n*, y cada conjunto de tiempos de proceso generados se construyen 20 problemas. Para los algoritmos exactos se ha establecido un límite temporal de 5 minutos, salvo para algunos de ellos y algunos valores de *n* en los que se ha ejecutado el programa sin límite de tiempo. Los valores de *n* estudiados con los diferentes algoritmos exactos son los siguientes:

Algoritmo Exacto	***Valores de n***
Programación Dinámica Básico	*5, 10, 15*
Srinivasian	*5, 10, 15*
Lawler 77	*5, 10, 15*
Lawler 77 (Restricción de delta)	*5, 10, 15*
Schrage y Baker	*5, 10, 15, 20, 30, 40, 50, 60, 70, 80, 90*
Schrage y Baker (Elmaghraby)	*5, 10, 15, 20, 30, 40, 50, 60, 70, 80, 90*
Lawler 79	*5, 10, 15, 20, 30, 40, 50, 60, 70, 80, 90*
Lawler 79 (Elmaghraby)	*5, 10, 15, 20, 30, 40, 50, 60, 70, 80, 90*
Potts y Van Wassenhove	*5, 10, 15, 20, 30, 40, 50, 60, 70, 80, 90, 110, 120, 130, 140, 150*

y con los algoritmos heurísticos:

Algoritmo Heurístico	*Valores de n*
Wilkerson e Irwin	*10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120, 130, 140, 150*
Fry, Macleod, Vicens y Fernández	*10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120, 130, 140, 150*
Sicilia y Alcaide	*10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120, 130, 140, 150*
Holsenback y Russell	*10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120, 130, 140, 150*

Las tablas que a continuación se relacionan hacen referencia a los algoritmos exactos. Los significados de los títulos de las columnas son los siguientes:

n = número de trabajos
nprob = número de problemas generados
res = número de problemas resueltos antes del tiempo límite (5 minutos, o bien sin tiempo límite).
t_medio = promedio de los tiempos de ejecución.
t_peor = máximo de los tiempos de ejecución entre los problemas que no exceden el tiempo límite.
conjuntos = número de conjuntos generados en las recursiones de la programación dinámica empleada.

Los tiempos vienen dados en segundos CPU de un ordenador personal 80486 DX a 50 Mhz.

Programación Dinámica Básico					
n	***nprob***	***res***	***t_medio***	***t_peor***	***conjuntos***
5	*100*	*100*	*0.007*	*0.055*	*32*
10	*100*	*100*	*0.341*	*0.549*	*1024*
15	*100*	*100*	*30.212*	*30.879*	*32768*

Programación Dinámica Srinivasian					
n	***nprob***	***res***	***t_medio***	***t_peor***	***conjuntos***
5	*100*	*100*	*0.001*	*0.055*	*5*
10	*100*	*100*	*0.026*	*0.330*	*79*
15	*100*	*100*	*2.969*	*30.440*	*3796*

Programación Dinámica de Lawler					
n	***nprob***	***res***	***t_medio***	***t_peor***	***conjuntos***
5	*100*	*100*	*0.015*	*0.055*	*6*
10	*100*	*100*	*0.025*	*0.055*	*18*
15	*100*	*100*	*0.044*	*0.330*	*53*
20	*100*	*100*	*0.120*	*1.209*	*111*
30	*100*	*100*	*1.510*	*46.978*	*763*
40	*100*	*98*	*16.829*	*202.033*	*2997*
50	*100*	*88*	*24.435*	*283.462*	*3649*
60	*100*	*79*	*12.792*	*155.769*	*2724*
70	*100*	*59*	*4.395*	*32.143*	*1083*
80	*100*	*58*	*4.308*	*64.121*	*954*
90	*100*	*57*	*6.356*	*43.846*	*1287*

Programación Dinámica Lawler (Elmaghraby)					
n	***nprob***	***res***	***t_medio***	***t_peor***	***conjuntos***
5	*100*	*100*	*0.016*	*0.055*	*6*
10	*100*	*100*	*0.018*	*0.055*	*16*
15	*100*	*100*	*0.039*	*0.275*	*44*
20	*100*	*100*	*0.105*	*0.989*	*84*
30	*100*	*100*	*0.737*	*11.099*	*428*
40	*100*	*100*	*11.485*	*189.121*	*1968*
50	*100*	*93*	*17.919*	*182.637*	*2731*
60	*100*	*85*	*15.608*	*271.099*	*2418*
70	*100*	*70*	*9.992*	*84.945*	*1719*
80	*100*	*66*	*7.959*	*79.890*	*1235*
90	*100*	*69*	*11.222*	*94.560*	*1606*

DP Schrage y Baker					
n	***nprob***	***res***	***t_medio***	***t_peor***	***conjuntos***
5	*100*	*100*	*0.003*	*0.055*	*6*
10	*100*	*100*	*0.013*	*0.110*	*18*
15	*100*	*100*	*0.083*	*1.429*	*55*
20	*100*	*100*	*0.338*	*7.473*	*115*
30	*100*	*100*	*34.151*	*1090.967*	*808*
40	*100*	*79*	*14.197*	*205.604*	*467*
50	*100*	*69*	*20.091*	*243.462*	*581*
60	*100*	*61*	*33.034*	*245.000*	*742*
70	*100*	*56*	*14.947*	*169.505*	*496*
80	*100*	*56*	*21.580*	*227.143*	*493*
90	*100*	*51*	*18.234*	*254.231*	*426*

DP Schrage y Baker (Elmaghraby)					
n	**nprob**	**res**	**t_medio**	**t_peor**	**conjuntos**
5	100	100	0.004	0.055	6
10	100	100	0.013	0.165	18
15	100	100	0.079	1.429	55
20	100	100	0.334	7.418	115
30	100	100	34.171	1093.626	808
40	100	79	14.220	205.714	467
50	100	69	20.119	243.516	581
60	100	61	33.034	245.440	742
70	100	56	14.981	170.000	496
80	100	56	21.635	227.198	493
90	100	51	18.295	254.231	426

Algoritmo de descomposición de Lawler					
n	**nprob**	**res**	**t_medio**	**t_peor**	**conjuntos**
5	100	100	0.002	0.055	12
10	100	100	0.116	0.330	498
15	100	100	4.640	16.648	18639

Alg. de descomposición de Lawler (restricción de δ)					
n	**nprob**	**res**	**t_medio**	**t_peor**	**conjuntos**
5	100	100	0.003	0.055	7
10	100	100	0.073	0.275	236
15	100	100	3.108	22.747	9254

Potts y Van Wassenhove					
n	**nprob**	**res**	**t_medio**	**t_peor**	**conjuntos**
5	100	100	0.001	00.055	4
10	100	100	0.003	00.055	11
15	100	100	0.004	00.055	25
20	100	100	0.015	0.055	45
30	100	100	0.054	0.440	187
40	100	100	0.130	0.824	411
50	100	100	0.383	6.923	1179
60	100	100	0.861	24.505	2669
70	100	100	1.938	27.582	6012
80	100	100	4.751	155.879	14373
90	100	100	4.755	71.374	14420
100	100	100	10.386	225.110	29765
110	100	100	25.332	380.495	68583

120	*100*	*100*	*104.676*	*4558.846*	*274545*
130	*100*	*100*	*90.390*	*1572.912*	*230954*
140	*100*	*100*	*413.945*	*17608.297*	*1038676*
150	*100*	*100*	*591.719*	*22586.209*	*1503148*

Las próximas tablas hacen referencia a los algoritmos heurísticos. En este caso se han resuelto todos los problemas sin fijar ningún tiempo límite de ejecución. Los heurísticos han sido comparados con el algoritmo exacto de Potts y van Wassenhove, calculando el radio solución óptima / solución heurística. Los significados de los encabezados de las columnas son los siguientes:

n = número de trabajos
nprob = número de problemas generados
t_exacto = promedio de los tiempos de ejecución empleados por el algoritmo exacto.
t_heur = promedio de los tiempos de ejecución empleados por el algoritmo heurístico
opt = número de óptimos encontrados por la heurística.
% = promedio de los radios de ejecución $\frac{z^*}{z_h} x100$, donde z^* es el valor de la solución óptima y z_h el valor de la solución heurística.

Los tiempos vienen dados en segundos CPU de un ordenador personal 80486 DX a 50 Mhz.

Holsenback y Russell					
n	***nprob***	***opt***	***%***	***t_exacto***	***t_heur***
10	*100*	*93*	*99.92*	*0.004*	*0.002*
20	*100*	*80*	*99.71*	*0.015*	*0.003*
30	*100*	*64*	*99.35*	*0.051*	*0.004*
40	*100*	*52*	*99.08*	*0.115*	*0.007*
50	*100*	*41*	*98.92*	*0.336*	*0.008*
60	*100*	*34*	*98.50*	*0.788*	*0.012*
70	*100*	*35*	*98.21*	*2.027*	*0.015*
80	*100*	*33*	*98.56*	*5.004*	*0.022*
90	*100*	*32*	*98.77*	*4.992*	*0.027*
100	*100*	*31*	*97.96*	*10.960*	*0.027*
110	*100*	*30*	*98.52*	*25.332*	*0.034*
120	*100*	*31*	*98.51*	*104.676*	*0.042*
130	*100*	*30*	*98.20*	*90.390*	*0.044*
140	*100*	*28*	*97.68*	*413.945*	*0.057*
150	*100*	*28*	*97.85*	*591.719*	*0.060*

Wilkerson e Irwin					
n	***nprob***	***opt***	***%***	***t_exacto***	***t_heur***
10	*100*	*76*	*98.10*	*0.004*	*0.000*
20	*100*	*61*	*98.68*	*0.015*	*0.002*
30	*100*	*47*	*96.39*	*0.051*	*0.002*
40	*100*	*40*	*96.58*	*0.115*	*0.004*
50	*100*	*35*	*95.75*	*0.336*	*0.008*
60	*100*	*33*	*95.03*	*0.788*	*0.015*
70	*100*	*35*	*94.97*	*2.027*	*0.024*
80	*100*	*31*	*95.24*	*5.004*	*0.034*
90	*100*	*35*	*96.39*	*4.992*	*0.045*
100	*100*	*31*	*95.47*	*10.960*	*0.063*
110	*100*	*29*	*95.80*	*25.332*	*0.102*
120	*100*	*29*	*95.41*	*104.676*	*0.147*
130	*100*	*30*	*95.94*	*90.390*	*0.206*
140	*100*	*25*	*94.80*	*413.945*	*0.251*
150	*100*	*28*	*93.89*	*591.719*	*0.322*

Fry, Macleod, Vicens y Fernández					
n	***nprob***	***opt***	***%***	***t_exacto***	***t_heur***
10	*100*	*84*	*99.58*	*0.004*	*0.003*
20	*100*	*69*	*99.30*	*0.015*	*0.013*
30	*100*	*56*	*98.41*	*0.051*	*0.035*
40	*100*	*47*	*98.41*	*0.115*	*0.071*
50	*100*	*42*	*98.41*	*0.336*	*0.134*
60	*100*	*36*	*98.23*	*0.788*	*0.223*
70	*100*	*40*	*98.51*	*2.027*	*0.349*
80	*100*	*35*	*98.52*	*5.004*	*0.521*
90	*100*	*35*	*98.48*	*4.992*	*0.722*
100	*100*	*34*	*98.42*	*10.960*	*0.981*
110	*100*	*32*	*98.49*	*25.332*	*1.304*
120	*100*	*29*	*98.05*	*104.676*	*1.668*
130	*100*	*33*	*98.20*	*90.390*	*2.116*
140	*100*	*30*	*98.36*	*413.945*	*2.624*
150	*100*	*29*	*98.22*	*591.719*	*3.184*

Sicilia y Alcaide					
n	***nprob***	***opt***	***%***	***t_exacto***	***t_heur***
10	*100*	*69*	*97.75*	*0.005*	*0.002*
20	*100*	*55*	*97.57*	*0.015*	*0.014*
30	*100*	*37*	*97.20*	*0.051*	*0.045*
40	*100*	*35*	*97.98*	*0.114*	*0.108*
50	*100*	*37*	*97.97*	*0.342*	*0.236*
60	*100*	*32*	*98.12*	*0.785*	*0.432*
70	*100*	*31*	*97.26*	*2.027*	*0.754*
80	*100*	*30*	*98.37*	*5.004*	*1.168*
90	*100*	*30*	*97.98*	*4.992*	*1.761*
100	*100*	*27*	*98.42*	*10.960*	*2.593*
110	*100*	*29*	*98.75*	*25.332*	*3.680*
120	*100*	*29*	*98.52*	*104.676*	*4.954*
130	*100*	*29*	*98.14*	*90.390*	*6.521*
140	*100*	*24*	*98.24*	*413.945*	*8.423*
150	*100*	*29*	*98.25*	*591.719*	*10.774*

Conclusiones

A tenor de las tablas ilustradas se observa que los algoritmos de programación dinámica analizados presentan un comportamiento limitado en cuanto a capacidad de resolución de problemas por encima de 15 trabajos. Con el algoritmo de descomposición de Lawler, basado en su teorema de descomposición, no parecen mejorar significativamente los resultados.

No obstante, al aplicar las propiedades de dominancia de Emmons en los métodos de programación dinámica se consigue aumentar la dimensión de los problemas resueltos. Así se llegan a resolver, en tiempos aceptables y sin grandes requerimientos de memoria, incluso problemas de 90 trabajos. Además la incorporación de la regla de dominancia, basada en el lema de Elmaghraby, en el algoritmo de programación dinámica de Lawler reduce significativamente el número de conjuntos a considerar y la cantidad de memoria necesaria para ello.

El algoritmo exacto que mejor comportamiento ha mostrado en los problemas considerados (problemas de 5 a 150 trabajos) ha sido el de Potts y Van Wassenhove, de ahí que se haya tomado como referencia para estudiar los algoritmos heurísticos.

Dada la complejidad del problema, y como demuestra la experiencia computacional, los algoritmos heurísticos parecen ser la manera más adecuada de obtener soluciones aceptables en tiempos razonables cuando el número de trabajos

supera el centenar. Las tablas precedentes muestran que las heurísticas garantizan una precisión en torno al 98%. Entre ellas, atendiendo al grado de precisión de la heurística, la que mejor comportamiento tiene para un número pequeño de trabajos (menor que 100) es la de Holsenback y Russell, seguida de la de Fry, Vicens, Macleod y Fernández, la de Sicilia y Alcaide, y la de Wilkerson e Irwin. Mientras que, para problemas con más de 100 trabajos parece ser que el mejor comportamiento, siempre en términos de precisión, lo tiene la heurística de Sicilia y Alcaide, seguida de la de Fry, Vicens, Macleod y Fernández, la de Holsenback y Russell, y la de Wilkerson e Irwin.

Apéndice B

Apéndice B: Experiencia computacional para problemas sobre varias máquinas

Experiencia computacional para el problema O||Cmax.

Se ha codificado en lenguaje de programación C el procedimiento de búsqueda tabú y los algoritmos heurísticos *LPT*, *LIS*, y *SPT* propuestos en el capítulo III para el problema $O||C_{max}$. Se han considerado dos versiones del método de búsqueda tabú: *TS0* y *TS1*. La primera versión *TS0* considera sólo la estructura de vecinos *N1*, mientras que *TS1* considera *N1* y *N2* periódicamente. Los períodos vienen limitados por las "vueltas atrás" de la búsqueda para recomenzar desde el mejor grafo que se haya encontrado hasta ese momento.

La experiencia ha sido realizada en una estación de trabajo Digital DEC 5000/240 segundos. Los problemas han sido aleatoriamente generados.

Las tablas 1, 2, y 3 hacen referencia a problemas donde los tiempos de proceso son generados de la *U[1,100], U[20,100],* y *U[50,100]* respectivamente. La tabla 4 hace referencia al conjunto de 60 problemas propuesto por Taillard (1993). Los resultados tabulados son promedios de 10 problemas para cada par de valores *(m,n)*. Los números entre paréntesis *"( )"* indican el número de problemas para los que el algoritmo alcanza la cota inferior. En la tabla 4, los números entre corchetes *"[ ]"* indican el número de problemas para los cuales se alcanza una nueva cota superior. Los títulos de las columnas significan lo siguiente:

LB = promedio de las cotas inferiores.
UB = promedio de las cotas superiores (sólo tabla 4).
LPT = promedio de las soluciones propuestas por el algoritmo *LPT*.
LIS = promedio de las soluciones propuestas por el algoritmo *LIS*.
SPT = promedio de las soluciones propuestas por el algoritmo *SPT*.
TS0 = promedio de las soluciones propuestas por el algoritmo *TS0*.
TS1 = promedio de las soluciones propuestas por el algoritmo *TS1*.
% UB = promedio de los cocientes cota superior / cota inferior.
% LPT = promedio de los cocientes solución *LPT* / cota inferior.
% LIS = promedio de los cocientes solución *LIS* / cota inferior.
% SPT = promedio de los cocientes solución *SPT* / cota inferior.
% TS0 = promedio de los cocientes solución *TS0* / cota inferior.
% TS1 = promedio de los cocientes solución *TS1* / cota inferior.
tiempo_0 = tiempo en segundos CPU para *TS0*.
tiempo_1 = tiempo en segundos CPU para *TS1*.

Se observa que las heurísticas *LPT, LIS* y *SPT* son las más rápidas. Sus tiempos de ejecución son realmente pequeños, nunca superiores a un segundo. Por tanto, estos tiempos no han sido tabulados. Sin embargo, las columnas *%LPT, %LIS* y *%SPT* indican que sus soluciones están aún lejos de la cota inferior en muchos problemas. De ahí que estas heurísticas por sí solas no garanticen siempre buenas soluciones. Las heurísticas tabú propuestas utilizan a estas como soluciones iniciales. Obviamente su precisión es mejor. Tal es así que alcanzan soluciones óptimas o próximas a ellas, incluso si las soluciones semilla iniciales están alejadas.

Por otra parte, los tiempos de ejecución de las heurísticas tabú no son excesivos con respecto al tamaño del problema. Recuérdese que el problema $O||C_{max}$ es NP-duro.

Comparando los procedimientos tabú *TS0* y *TS1* se deduce que, en general, el primero de ellos da una solución más preferible. Ninguno de los algoritmos domina al otro en términos de tiempos de ejecución.

En problemas rectangulares $(m \neq n)$ los algoritmos obtienen soluciones óptimas, ó próximas a ellas, más rápidamente que en problemas cuadrados $(m=n)$. Más aún, en problemas cuadrados los tiempos de ejecución se incrementan notablemente en comparación con los problemas rectangulares con el mismo número de máquinas. Nótese que, aunque ninguno de los algoritmos *TS0, TS1* domina al otro en términos de tiempo, cuando se precisa una intensificación de la búsqueda con la estructura de vecinos *N2* en el algoritmo *TS1* se produce un crecimiento de los tiempos de ejecución con respecto a *TS0*.

Llama la atención el alto número de soluciones óptimas alcanzadas por las heurísticas tabú en las tablas 1, 2, y 3, como puede verse en las columnas *TS0* y *TS1*. Además, en la tabla 4, nótese que para ciertos tamaños de los problemas se obtienen mejores resultados que los de Taillard (1993).

Tabla 1
Estación de trabajo Digital DEC 5000/240 segundos.
$p_{i,j}$ generados uniformemente en $[1, 100]$. Resultados promedio sobre 10 problemas para cada par m, n.

m	n	LB	LPT	LIS	SPT	$TS0$	$TS1$	$\%LPT$	$\%LIS$	$\%SPT$	$\%TS0$	$\%TS1$	$tiempo0$	$tiempo1$
3	5	309.0	324.2 (3)	320.7 (5)	362.5 (2)	(10)	(10)	104.99	104.08	117.82	100.00	100.00	0.070	0.071
3	10	568.9	578.4 (5)	594.1 (2)	591.6 (3)	(10)	(10)	101.60	104.54	104.10	100.00	100.00	0.111	0.080
3	25	1404.4	1415.9 (2)	1418.5 (4)	1460.4 (1)	(10)	(10)	100.83	101.00	103.91	100.00	100.00	0.073	0.111
3	50	2686.2	2698.1 (4)	2706.9 (6)	2756.4 (1)	(10)	(10)	100.44	100.76	102.58	100.00	100.00	1.616	1.123
3	100	5352.6	5357.8 (1)	5384.2 (2)	5390.9 (2)	(10)	(10)	100.09	100.59	100.70	100.00	100.00	0.761	1.176
3	250	13166.7	13168.9 (6)	13177.7 (7)	13241.6	(10)	(10)	100.01	100.08	100.50	100.00	100.00	1.566	1.518
5	5	356.5	404.4 (2)	420.5	457.0	(10)	(10)	113.01	118.24	127.59	100.00	100.00	0.283	0.640
5	10	594.7	648.7 (1)	691.5 (1)	694.4	(10)	(10)	110.00	116.48	117.06	100.00	100.00	0.175	0.231
5	25	1427.7	1445.0 (4)	1505.3	1510.2	(10)	(10)	101.23	105.52	105.80	100.00	100.00	0.846	0.825
5	50	2771.6	2802.2 (1)	2856.1 (1)	2868.0 (2)	(10)	(10)	101.11	103.06	103.51	100.00	100.00	0.450	0.471
5	100	5469.8	5493.0 (1)	5611.5	5565.2	(10)	(10)	100.42	102.57	101.75	100.00	100.00	1.876	1.868
5	250	13289.4	13304.5 (2)	13481.3	13425.5	(10)	(10)	100.11	101.44	101.02	100.00	100.00	17.851[a]	16.736[a]
10	10	676.8	824.2	913.4	888.3	677.0 (9)	677.3 (9)	122.15	135.51	131.61	100.03	100.08	351.653[b]	391.803[c]
10	25	1487.8	1554.2	1718.5	1718.2	(10)	(10)	104.45	115.57	115.53	100.00	100.00	0.308	0.308
10	50	2866.8	2925.5	3127.3	3066.5	(10)	(10)	102.04	109.07	107.01	100.00	100.00	0.281	0.245
10	100	5555.4	5613.9	5937.3	5811.8	(10)	(10)	101.04	106.89	104.60	100.00	100.00	0.986	0.961
10	250	13396.0	13542.9	13937.8	13729.0	(10)	(10)	100.34	103.28	101.72	100.00	100.00	1.771	1.798
15	50	2889.5	3000.0	3322.8	3251.2	(10)	(10)	103.83	115.01	112.51	100.00	100.00	1.974	1.960
15	100	5589.7	5663.3	6107.2	5990.5	(10)	(10)	101.32	109.30	107.07	100.00	100.00	2.176	2.430
15	250	13541.7	13590.2	14256.3	13959.7	(10)	(10)	100.35	105.29	103.10	100.00	100.00	3.826	3.653
25	25	1587.3	2027.8	2363.6	2271.3	1669.4	1681.5	127.81	148.96	142.51	105.21	105.97	5400.0	5400.0
25	100	5710.4	5884.4	6568.5	6317.2	(10)	(10)	103.04	115.09	110.64	100.00	100.00	13.276	13.145
25	250	13635.4	13749.5	14749.4	14266.7	(10)	(10)	100.83	108.18	104.63	100.00	100.00	23.115	22.644
50	100	5748.8	6213.8	7387.9	6956.8	5769.2 (4)	5771.7 (2)	108.08	128.54	121.06	100.35	100.40	3518.974	4576.809

[a]Sólo 2 problemas más de 1 segundo, [b] sólo 4 más de 4 segundos, [c] sólo 4 más de 3 segundos.

Tabla 2
Estación de trabajo Digital DEC 5000/240 segundos.
$p_{i,j}$ generados uniformemente en $[20, 100]$ Resultados promedio sobre 10 problemas para cada par m, n.

m	n	LB	LPT	LIS	SPT	$TS0$	$TS1$	$\%LPT$	$\%LIS$	$\%SPT$	$\%TS0$	$\%TS1$	$tiempo0$	$tiempo1$
3	5	322.4	341.6 (1)	371.4 (2)	368.1 (2)	(10)	(10)	106.02	125.79	114.67	100.00	100.00	0.070	0.026
3	10	662.2	680.8 (3)	692.5 (4)	727.8	(10)	(10)	102.82	104.69	110.09	100.00	100.00	0.046	0.036
3	25	1585.3	1623.0	1633.3 (1)	1644.5 (1)	(10)	(10)	102.35	103.05	103.79	100.00	100.00	0.025	0.022
3	50	3146.6	3163.4 (1)	3155.0 (3)	3208.1 (1)	(10)	(10)	100.53	100.27	101.96	100.00	100.00	0.276	0.195
3	100	6220.4	6231.7 (1)	6281.9 (1)	6262.9 (3)	(10)	(10)	100.17	100.99	100.68	100.00	100.00	0.761	1.211
3	250	15358.6	15368.4 (2)	15375.6 (6)	15405.6 (3)	(10)	(10)	100.09	100.11	100.30	100.00	100.00	1.680	1.765
5	5	368.7	469.4	467.2	477.5	369.0 (9)	369.3 (9)	127.21	126.75	129.95	100.09	100.17	30.170[a]	36.198[b]
5	10	698.2	750.9 (1)	797.1	815.7 (1)	(10)	(10)	109.28	116.26	118.98	100.00	100.00	0.213	0.205
5	25	1627.9	1680.6	1739.2	1768.6	(10)	(10)	103.21	106.84	108.63	100.00	100.00	0.075	0.070
5	50	3182.6	3221.4 (1)	3311.5	3332.2	(10)	(10)	101.20	104.08	104.93	100.00	100.00	0.468	0.456
5	100	6361.2	6394.5 (1)	6506.5	6490.5 (2)	(10)	(10)	100.52	102.28	102.03	100.00	100.00	0.221	0.223
5	250	15570.6	15608.1	15751.1	15738.1	(10)	(10)	100.24	101.15	101.07	100.00	100.00	0.953	1.018
10	10	737.2	942.0	1029.5	1020.0	740.7 (8)	741.0 (7)	128.00	139.78	138.76	100.50	100.55	843.386[c]	965.083[c]
10	25	1669.0	1808.1	1993.3	1962.8	(10)	(10)	108.37	119.50	117.67	100.00	100.00	1.536	1.523
10	50	3289.0	3394.9	3588.8	3556.5	(10)	(10)	103.20	109.15	108.13	100.00	100.00	0.778	0.763
10	100	6423.6	6502.5	6860.8	6680.1	(10)	(10)	101.23	106.82	103.99	100.00	100.00	1.151	1.170
10	250	15680.2	15782.0	16274.0	15883.1	(10)	(10)	100.64	103.79	101.29	100.00	100.00	2.611	2.733
15	50	3289.0	3442.9	3794.2	3717.4	(10)	(10)	104.67	115.37	113.03	100.00	100.00	2.781	2.771
15	100	6462.0	6598.3	7105.1	6886.5	(10)	(10)	102.10	109.96	106.56	100.00	100.00	4.171	4.883
15	250	15835.4	15986.0	16541.8	16209.4	(10)	(10)	100.95	104.47	102.36	100.00	100.00	9.983	9.676
25	25	1742.5	2377.9	2704.7	2593.5	1934.2	1950.9	133.02	155.23	148.86	111.00	111.96	5400.0	5400.0
25	100	6468.4	6712.2	7532.1	7181.4	(10)	(10)	103.76	116.45	111.02	100.00	100.00	18.678	18.728
25	250	15953.0	16165.0	17312.4	16567.3	(10)	(10)	101.33	108.53	103.85	100.00	100.00	40.938	43.518
50	100	6562.8	7113.4	8446.1	8054.3	6687.2 (1)	6682.8 (1)	108.40	128.71	122.74	101.90	101.84	5070.365	5065.118

[a]Sólo 2 problemas más de 2 segundos, [b] sólo 3 más de 2 segundos, [c] sólo 5 más de 30 segundos.

Tabla 3
Estación de trabajo Digital DEC 5000/240 segundos.
$p_{i,j}$ generados uniformemente en $[50, 100]$ Resultados promedio sobre 10 problemas para cada par m, n.

m	n	LB	LPT	LIS	SPT	$TS0$	$TS1$	$\%LPT$	$\%LIS$	$\%SPT$	$\%TS0$	$\%TS1$	$tiempo0$	$tiempo1$
3	5	405.6	461.4	464.8 (1)	491.3 (1)	(10)	(10)	113.78	114.58	121.53	100.00	100.00	0.036	0.035
3	10	800.4	845.5 (1)	860.6 (2)	866.7 (2)	(10)	(10)	105.60	107.63	108.26	100.00	100.00	0.041	0.031
3	25	1937.6	1968.7 (2)	1991.2 (2)	2024.7	(10)	(10)	101.62	102.79	104.52	100.00	100.00	0.168	0.123
3	50	3850.2	3896.1 (1)	3928.8 (2)	3955.6	(10)	(10)	101.18	102.14	102.73	100.00	100.00	0.293	0.216
3	100	7669.2	7731.7	7725.7 (2)	7760.5	(10)	(10)	100.81	100.73	101.19	100.00	100.00	0.075	0.105
3	250	19081.6	19125.7	19127.9 (4)	19140.0 (2)	(10)	(10)	100.23	100.24	100.30	100.00	100.00	0.231	0.236
5	5	422.7	542.9	544.7	596.6	422.9 (9)	(10)	128.33	128.91	141.18	100.05	100.00	31.681[a]	27.416[a]
5	10	806.8	835.6	932.1	963.0	(10)	(10)	115.08	115.58	119.48	100.00	100.00	0.421	0.430
5	25	1970.7	2065.0	2117.8	2132.8	(10)	(10)	104.78	107.52	108.25	100.00	100.00	0.130	0.133
5	50	3890.4	3995.7	4093.9	4106.6	(10)	(10)	102.72	105.23	105.55	100.00	100.00	0.333	0.336
5	100	7710.4	7805.2	7983.4	7945.0	(10)	(10)	101.23	103.54	103.04	100.00	100.00	0.326	0.325
5	250	19112.2	19212.6	19430.2	19222.7	(10)	(10)	100.52	101.66	100.57	100.00	100.00	0.651	0.628
10	10*	835.4	1151.4	1257.6	1284.8	869.8	877.4	137.89	150.58	153.89	104.16	105.07	3034.197	3598.180
10	25	1994.1	2237.7	2414.5	2407.6	(10)	(10)	112.20	121.10	120.73	100.00	100.00	2.865	2.880
10	50	3928.2	4150.0	4508.8	4333.6	(10)	(10)	105.66	114.79	110.32	100.00	100.00	1.921	1.910
10	100	7751.6	7962.6	8410.7	8165.9	(10)	(10)	102.72	108.50	105.34	100.00	100.00	2.270	2.336
10	250	19224.4	19457.0	20232.4	19578.3	(10)	(10)	101.20	105.24	101.84	100.00	100.00	6.241	5.826
15	50	3932.8	4268.2	4678.4	4540.4	(10)	(10)	108.53	118.97	115.47	100.00	100.00	9.846	9.801
15	100	7781.4	8107.7	8705.6	8381.0	(10)	(10)	104.19	111.88	107.71	100.00	100.00	8.938	9.209
15	250	19254.8	19560.9	20562.5	19772.1	(10)	(10)	101.58	106.79	102.78	100.00	100.00	19.496	19.411
25	25	2049.1	2815.0	3218.1	3148.6	2367.6	2365.0**	137.40	157.05	153.68	115.55	115.78**	5400.0	5400.0
25	100	7807.4	8347.8	9179.6	8828.1	(10)	(10)	106.92	117.57	113.07	100.00	100.00	80.866	78.896
25	250	19298.8	19785.7	21225.3	20202.1	(10)	(10)	102.52	109.98	104.68	100.00	100.00	108.031	106.926
50	100	7863.4	8904.8	10116.8	9759.7	8320.6	8329.2	113.24	128.65	124.11	105.81	105.92	5400.0	5400.0

*Sobre 5 problemas, ** sobre 4 problemas. [a] Sólo 4 problemas más de 2 segundos.

Tabla 4
Estación de trabajo Digital DEC 5000/240 segundos.
Problemas de Taillard. Resultados promedio sobre 10 problemas para cada par m, n.

$m = n$	LB	UB	LPT	LIS	SPT	$TS0$	$TS1$	$\%UB$	$\%LPT$	$\%LIS$	$\%SPT$	$\%TS0$	$\%TS1$	$tiempo0$	$tiempo1$
4	227.4	233.0	299.9	285.2	311.9	232.7 [1]	234.0	102.46	131.85	126.31	137.33	102.55	102.88	27.438	57.343
5	306.5	311.5	411.4	427.1	447.7	311.9	313.2	101.66	133.88	139.74	145.50	101.79	102.20	177.060	361.300
7	437.8	443.3	563.2	603.7	619.3	441.0 (2) [6]	443.8 [2]	101.24	128.72	137.69	141.34	100.71	101.36	1345.479	731.166
10	598.0	601.6 (4)	789.2	874.5	904.0	602.3 (3) [3]	603.7 (1) [2]	100.58	131.86	146.26	150.09	100.70	100.94	2000.158	3354.897
15	912.4	913.7 (8)	1190.0	1325.7	1369.7	924.6 (2)	930.6 (1)	100.14	130.42	145.33	150.11	101.34	102.00	4622.958	5097.600
20	1235.4	1236.6 (7)	1593.6	1875.0	1793.5	1283.0	1295.9	100.09	129.02	151.85	145.21	103.90	104.94	5400.0	5400.0

Experiencia computacional para el problema P|prec|∑Tj

Se ha codificado en lenguaje de programación C los algoritmos propuestos en el capítulo III para el problema $P|prec|\sum T_j$. Estos algoritmos proponen una solución óptima del problema para el caso en el que el número m de máquinas sea superior al máximo número n^* de trabajos por nivel en el grafo de precedencias, y una solución heurística en caso contrario.

La experiencia ha sido realizada en una estación de trabajo HP 9000 serie 700, modelo 712, a 80 Mhz.

Los problemas test experimentados han sido generados aleatoriamente de la siguiente manera.

El número n de trabajos varía en el rango $\{10, 20, 30, 40, 50, 100, 150, 200, 250, 500\}$, mientras que el número m de máquinas toma los valores $\{5, 10, 20, 30, 40, 50\}$. Para cada par de valores (m,n) se generan 5 problemas test.

Los tiempos de proceso p_j, $j=1,...,n$ son valores enteros de la $U[1,100]$. Una vez generado cada valor p_j, se genera la correspondiente fecha límite como un valor entero $d_j \in U[p_j, p_j+20]$.

El grafo $G = (V, A)$ que modeliza las precedencias existentes entre los trabajos se genera de la siguiente manera. El conjunto de vértices $V = \{k / k=1,..,n\}$ representa los n trabajos a realizar. Para cada vértice k, el número de arcos que salen de él, es decir, su grado exterior $d_o(k)$, es un número entero aleatoriamente generado de la $U[0, \min \{5, n-k\}]$. De esta manera, para cada $k=1,...,n-1$ se generan $d_o(k)$ enteros j en $U[k+1,n]$. Todos estos arcos (k,j) constituyen el conjunto A.

En las tablas siguientes se muestran los resultados promedio obtenidos sobre 5 problemas test para cada par (m,n). En cada tabla, fijada la fila, tenemos los datos correspondientes a un valor m fijo del número de máquinas; y, fijada la columna, tenemos los datos correspondientes a un valor n fijo del número de trabajos. En la tabla 1 vienen los promedios, sobre 5 problemas test para cada par (m,n), del valor del objetivo $\sum T_j$, mientras que en la tabla 2 se recogen los tiempos promedio empleados en la resolución de dichos problemas. La tabla 3 es una tabla complementaria en la que se tabulan los valores promedio de los instantes finales de completación C_{max} de todos los trabajos que configuran el problema.

Tabla 1. *Valores promedio del objetivo $\sum T_j$*

m \ n	***10***	***20***	***30***	***40***	***50***	***100***	***150***	***200***	***250***	***500***
5	*6*	*963*	*1351*	*1366*	*3175*	*3057*	*12692*	*32067*	*57349*	*86420*
10	*8*	*903*	*1127*	*1666*	*3820*	*2994*	*6625*	*9362*	*13598*	*25035*
20	*1*	*893*	*1323*	*1487*	*3111*	*2624*	*5997*	*9344*	*12046*	*12745*
30	*5*	*745*	*1421*	*1123*	*2989*	*3191*	*6167*	*9412*	*11056*	*12951*
40	*8*	*956*	*1121*	*1169*	*3211*	*3244*	*6361*	*9378*	*11286*	*13070*
50	*3*	*993*	*1321*	*1061*	*3452*	*2812*	*6386*	*9153*	*11646*	*12998*

Tabla 2. *Tiempos promedio (en segundos).*

Máquinas\Trabajos	***10***	***20***	***30***	***40***	***50***
5	*0.014000*	*0.038000*	*0.070000*	*0.128000*	*0.188000*
10	*0.018000*	*0.044000*	*0.082000*	*0.136000*	*0.202000*
20	*0.016000*	*0.046000*	*0.088000*	*0.140000*	*0.198000*
30	*0.014000*	*0.048000*	*0.082000*	*4.546000*	*0.200000*
40	*0.014000*	*0.046000*	*0.090000*	*0.142000*	*0.208000*
50	*0.014000*	*0.046000*	*0.082000*	*0.140000*	*0.216000*

Tabla 2. *Tiempos promedio (en segundos). (continuación).*

Máquinas\Trabajos	***100***	***150***	***200***	***250***	***500***
5	*0.252000*	*0.942000*	*1.852000*	*3.144000*	*4.594000*
10	*0.264000*	*0.958000*	*1.860000*	*3.190000*	*4.830000*
20	*0.268000*	*0.968000*	*1.866000*	*3.120000*	*4.638000*
30	*0.270000*	*0.980000*	*1.910000*	*3.136000*	*4.642000*
40	*0.266000*	*0.956000*	*1.872000*	*3.118000*	*4.662000*
50	*0.272000*	*0.974000*	*1.892000*	*3.128000*	*4.651999*

Tabla 3. *Valores promedio para C_{max}.*

m \ n	***10***	***20***	***30***	***40***	***50***	***100***	***150***	***200***	***250***	***500***
5	*104*	*331*	*465*	*613*	*929*	*1019*	*1668*	*2518*	*3226*	*3877*
10	*104*	*324*	*465*	*613*	*1015*	*1064*	*1846*	*2477*	*3304*	*4353*
20	*93*	*341*	*465*	*671*	*1212*	*1077*	*1946*	*2518*	*3569*	*4188*
30	*104*	*331*	*465*	*584*	*1023*	*1314*	*1846*	*2505*	*3561*	*4371*
40	*110*	*293*	*465*	*643*	*989*	*914*	*1445*	*2705*	*3170*	*4600*
50	*98*	*311*	*465*	*641*	*1008*	*1142*	*1366*	*2128*	*3645*	*4542*

Nótese que para valores de n menores o iguales a *100*, salvo el par *(m,n)=(30,50)*, el algoritmo otorga siempre una solución en menos de *0.3* segundos. Para *n = 150, 200, 250* y *500* nunca precisa más de *1, 2, 4,* y *5* segundos, respectivamente. Esto da idea de la velocidad de convergencia del algoritmo.

Apéndice C

Apéndice C: Experiencia computacional para problemas bicriterio

En este apéndice se recoge la experiencia computacional realizada sobre un problema de planificación bicriterio. Para resolverlo se han generado aleatoriamente datos y se han programado en lenguaje de programación C los algoritmos propuestos en el epígrafe 3.1. del capítulo IV. La experiencia se ha desarrollado en un ordenador personal AT 80486 a 33 Mhz.

Se desean planificar $n = 30$ trabajos en una máquina, cuyos tiempos de proceso p_j vienen generados de una variable entera $U[1,100]$. Los pesos w_{1j}, w_{2j} son generados de una variable entera $U[1,10]$. El problema bicriterio $\alpha|\beta|(\gamma^1, \gamma^2)$ que se considera es el problema $1||(\sum_j \omega_{1j}C_j, \sum_j \omega_{2j}C_j)$. Los problemas unicriterio asociados con este problema se resuelven en tiempo polinomial por la regla *WSPT* debida a Smith (1956). Dicha regla nos indica que el trabajo i es anterior al j en la planificación activa óptima para el problema unicriterio sí, y sólo sí, $p_i / w_i \leq p_j / w_j$, siendo w_i, w_j los pesos de los trabajos i y j en dicho criterio.

Los datos vienen dados por las siguientes tablas:

T_j	p_j	w_{1j}	w_{2j}
1	15	6	10
2	16	5	2
3	87	6	4
4	4	7	7
5	66	4	3
6	38	6	6
7	74	4	2
8	2	8	9
9	41	3	5
10	13	3	8
11	100	3	4
12	41	6	9
13	34	10	10
14	58	6	10
15	83	8	9

T_j	p_j	w_{1j}	w_{2j}
16	31	5	6
17	17	5	3
18	11	10	8
19	73	7	5
20	15	4	2
21	48	10	7
22	90	7	6
23	20	5	4
24	74	9	8
25	86	6	4
26	31	4	10
27	59	3	10
28	82	9	6
29	17	9	8
30	58	7	8

Los puntos eficientes generados por el algoritmo dados en el orden de generación son:

78920	*85282*
85909	*75845*
80613	*78113*
82617	*76424*
84081	*76069*
85209	*75889*
85627	*75851*
85754	*75847*
85449	*75863*
84693	*75965*
85139	*75898*
84502	*75997*
84255	*76039*
83279	*76246*
83769	*76132*
83905	*76103*
83985	*76087*
84070	*76071*

83811	*76123*
83513	*76190*
83707	*76146*
83421	*76212*
82867	*76352*
82987	*76320*
82720	*76394*
81733	*76891*
82237	*76598*
82303	*76566*
82378	*76532*
81886	*76792*
82044	*76702*
82222	*76606*
81935	*76764*
81807	*76838*
81101	*77481*
81337	*77236*

81424	*77159*
81192	*77383*
81262	*77312*
80959	*77650*
80979	*77626*
80836	*77814*
79367	*81059*
79720	*79839*
80212	*78750*
80516	*78251*
80549	*78201*
80231	*78717*
80100	*78981*
80154	*78866*
79788	*79679*
79982	*79244*
79624	*80093*
79629	*80079*

79546	*80369*
79403	*80915*
79372	*81039*
79035	*83171*
79248	*81724*
79321	*81297*
79339	*81201*
79287	*81491*
79265	*81622*
79152	*82348*
79045	*83079*
79059	*82982*
78943	*84424*
78973	*83952*
79018	*83375*
78980	*83858*
78934	*84660*
78938	*84532*

Si ordenamos los puntos eficientes de acuerdo al orden creciente del primer criterio tenemos la lista de puntos:

78920	*85282*
78934	*84660*
78938	*84532*
78943	*84424*
78973	*83952*
78980	*83858*
79018	*83375*
79035	*83171*
79045	*83079*
79059	*82982*
79152	*82348*
79248	*81724*
79265	*81622*
79287	*81491*
79321	*81297*
79339	*81201*
79367	*81059*
79372	*81039*

79403	*80915*
79546	*80369*
79624	*80093*
79629	*80079*
79720	*79839*
79788	*79679*
79982	*79244*
80100	*78981*
80154	*78866*
80212	*78750*
80231	*78717*
80516	*78251*
80549	*78201*
80613	*78113*
80836	*77814*
80959	*77650*
80979	*77626*
81101	*77481*

81192	*77383*
81262	*77312*
81337	*77236*
81424	*77159*
81733	*76891*
81807	*76838*
81886	*76792*
81935	*76764*
82044	*76702*
82222	*76606*
82237	*76598*
82303	*76566*
82378	*76532*
82617	*76424*
82720	*76394*
82867	*76352*
82987	*76320*
83279	*76246*

83421	*76212*
83513	*76190*
83707	*76146*
83769	*76132*
83811	*76123*
83905	*76103*
83985	*76087*
84070	*76071*
84081	*76069*
84255	*76039*
84502	*75997*
84693	*75965*
85139	*75898*
85209	*75889*
85449	*75863*
85627	*75851*
85754	*75847*
85909	*75845*

A las rectas que unen estos puntos en ese orden corresponden las pendientes:

-44.428571
-32.000000
-21.600000
-15.733333
-13.428571
-12.710526
-12.000000
-9.200000
-6.928571
-6.817204
-6.500000
-6.000000
-5.954545
-5.705882
-5.333333
-5.071429
-4.000000
-4.000000

-3.818182
-3.538462
-2.800000
-2.637363
-2.352941
-2.242268
-2.228814
-2.129630
-2.000000
-1.736842
-1.635088
-1.515152
-1.375000
-1.340807
-1.333333
-1.200000
-1.188525
-1.076923

-1.014286
-1.013333
-0.885057
-0.867314
-0.716216
-0.582278
-0.571429
-0.568807
-0.539326
-0.533333
-0.484848
-0.453333
-0.451883
-0.291262
-0.285714
-0.266667
-0.253425
-0.239437

-0.239130
-0.226804
-0.225806
-0.214286
-0.212766
-0.200000
-0.188235
-0.181818
-0.172414
-0.170040
-0.167539
-0.150224
-0.128571
-0.108333
-0.067416
-0.031496
-0.012903

Las planificaciones correspondientes a los puntos obtenidos son las de las tablas siguientes, donde el número de la primera columna es el número de orden en el que el punto, asociado a la planificación que figura en la fila, fue generado por el algoritmo. Estas planificaciones vienen dadas en orden creciente del primer criterio.

1)	*8*	*4*	*18*	*29*	*1*	*2*	*13*	*17*	*20*	*23*	*10*	*21*	*16*	*6*	*12*	*26*	*24*	*30*	*28*	*14*	*15*	*19*	*22*	*9*	*25*	*3*	*5*	*7*	*27*	*11*
71)	*8*	*4*	*18*	*29*	*1*	*2*	*13*	*17*	*20*	*23*	*10*	*21*	*16*	*6*	*12*	*26*	*24*	*30*	*28*	*14*	*15*	*19*	*22*	*9*	*25*	*3*	*5*	*27*	*7*	*11*
72)	*8*	*4*	*18*	*29*	*1*	*2*	*13*	*17*	*20*	*23*	*10*	*21*	*16*	*6*	*12*	*26*	*30*	*24*	*28*	*14*	*15*	*19*	*22*	*9*	*25*	*3*	*5*	*27*	*7*	*11*
67)	*8*	*4*	*18*	*29*	*1*	*2*	*13*	*17*	*20*	*10*	*23*	*21*	*16*	*6*	*12*	*26*	*30*	*24*	*28*	*14*	*15*	*19*	*22*	*9*	*25*	*3*	*5*	*27*	*7*	*11*
68)	*8*	*4*	*18*	*29*	*1*	*2*	*13*	*17*	*20*	*10*	*23*	*21*	*16*	*6*	*12*	*26*	*30*	*24*	*14*	*28*	*15*	*19*	*22*	*9*	*25*	*3*	*5*	*27*	*7*	*11*

70)	*8*	*4*	*18*	*29*	*1*	*2*	*13*	*17*	*10*	*20*	*23*	*21*	*16*	*6*	*12*	*26*	*30*	*24*	*14*	*28*	*15*	*19*	*22*	*9*	*25*	*3*	*5*	*27*	*7*	*11*
69)	*8*	*4*	*18*	*29*	*1*	*2*	*13*	*17*	*10*	*20*	*23*	*21*	*16*	*6*	*12*	*26*	*30*	*24*	*14*	*28*	*15*	*19*	*22*	*9*	*25*	*3*	*27*	*5*	*7*	*11*
58)	*8*	*4*	*18*	*29*	*1*	*2*	*13*	*17*	*10*	*20*	*23*	*21*	*16*	*6*	*12*	*26*	*30*	*24*	*14*	*28*	*15*	*19*	*9*	*22*	*25*	*3*	*27*	*5*	*7*	*11*
65)	*8*	*4*	*18*	*29*	*1*	*13*	*2*	*17*	*10*	*20*	*23*	*21*	*16*	*6*	*12*	*26*	*30*	*24*	*14*	*28*	*15*	*19*	*9*	*22*	*25*	*3*	*27*	*5*	*7*	*11*
66)	*8*	*4*	*18*	*29*	*1*	*13*	*2*	*10*	*17*	*20*	*23*	*21*	*16*	*6*	*12*	*26*	*30*	*24*	*14*	*28*	*15*	*19*	*9*	*22*	*25*	*3*	*27*	*5*	*7*	*11*

64)	*8*	*4*	*18*	*29*	*1*	*13*	*2*	*10*	*17*	*20*	*23*	*21*	*16*	*6*	*12*	*26*	*30*	*24*	*14*	*28*	*15*	*19*	*9*	*22*	*25*	*27*	*3*	*5*	*7*	*11*
59)	*8*	*4*	*18*	*29*	*1*	*13*	*2*	*10*	*17*	*20*	*23*	*21*	*16*	*6*	*12*	*26*	*30*	*24*	*14*	*28*	*15*	*19*	*9*	*22*	*27*	*25*	*3*	*5*	*7*	*11*
63)	*8*	*4*	*18*	*29*	*1*	*13*	*10*	*2*	*17*	*20*	*23*	*21*	*16*	*6*	*12*	*26*	*30*	*24*	*14*	*28*	*15*	*19*	*9*	*22*	*27*	*25*	*3*	*5*	*7*	*11*
62)	*8*	*4*	*18*	*29*	*1*	*13*	*10*	*2*	*17*	*20*	*23*	*21*	*16*	*6*	*26*	*12*	*30*	*24*	*14*	*28*	*15*	*19*	*9*	*22*	*27*	*25*	*3*	*5*	*7*	*11*
60)	*8*	*4*	*18*	*29*	*1*	*13*	*10*	*2*	*17*	*20*	*23*	*21*	*16*	*26*	*6*	*12*	*30*	*24*	*14*	*28*	*15*	*19*	*9*	*22*	*27*	*25*	*3*	*5*	*7*	*11*

61)	*8*	*4*	*18*	*29*	*1*	*13*	*10*	*2*	*17*	*20*	*23*	*21*	*16*	*26*	*12*	*6*	*30*	*24*	*14*	*28*	*15*	*19*	*9*	*22*	*27*	*25*	*3*	*5*	*7*	*11*
43)	*8*	*4*	*18*	*29*	*1*	*10*	*13*	*2*	*17*	*20*	*23*	*21*	*16*	*26*	*12*	*6*	*30*	*24*	*14*	*28*	*15*	*19*	*9*	*22*	*27*	*25*	*3*	*5*	*7*	*11*
57)	*8*	*4*	*18*	*29*	*1*	*10*	*13*	*2*	*17*	*23*	*20*	*21*	*16*	*26*	*12*	*6*	*30*	*24*	*14*	*28*	*15*	*19*	*9*	*22*	*27*	*25*	*3*	*5*	*7*	*11*
56)	*8*	*4*	*18*	*29*	*1*	*10*	*13*	*2*	*17*	*23*	*20*	*21*	*26*	*16*	*12*	*6*	*30*	*24*	*14*	*28*	*15*	*19*	*9*	*22*	*27*	*25*	*3*	*5*	*7*	*11*
55)	*8*	*4*	*18*	*29*	*1*	*10*	*13*	*2*	*17*	*23*	*20*	*21*	*26*	*16*	*12*	*6*	*30*	*24*	*14*	*28*	*15*	*19*	*9*	*27*	*22*	*25*	*3*	*5*	*7*	*11*

53)	*8*	*4*	*18*	*29*	*1*	*10*	*13*	*2*	*17*	*23*	*20*	*21*	*26*	*16*	*12*	*6*	*30*	*14*	*24*	*28*	*15*	*19*	*9*	*27*	*22*	*25*	*3*	*5*	*7*	*11*
54)	*8*	*4*	*18*	*29*	*1*	*10*	*13*	*17*	*2*	*23*	*20*	*21*	*26*	*16*	*12*	*6*	*30*	*14*	*24*	*28*	*15*	*19*	*9*	*27*	*22*	*25*	*3*	*5*	*7*	*11*
44)	*8*	*4*	*18*	*29*	*1*	*10*	*13*	*17*	*2*	*23*	*20*	*21*	*26*	*16*	*12*	*6*	*30*	*14*	*24*	*15*	*28*	*19*	*9*	*27*	*22*	*25*	*3*	*5*	*7*	*11*
51)	*8*	*4*	*18*	*29*	*1*	*10*	*13*	*17*	*2*	*23*	*20*	*21*	*26*	*16*	*12*	*6*	*30*	*14*	*24*	*15*	*28*	*9*	*19*	*27*	*22*	*25*	*3*	*5*	*7*	*11*
52)	*8*	*4*	*18*	*29*	*1*	*10*	*13*	*17*	*2*	*23*	*20*	*21*	*26*	*16*	*12*	*6*	*30*	*14*	*24*	*15*	*28*	*9*	*27*	*19*	*22*	*25*	*3*	*5*	*7*	*11*

49)	*8*	*4*	*18*	*29*	*1*	*10*	*13*	*17*	*2*	*23*	*20*	*26*	*21*	*16*	*12*	*6*	*30*	*14*	*24*	*15*	*28*	*9*	*27*	*19*	*22*	*25*	*3*	*5*	*7*	*11*
50)	*8*	*4*	*18*	*29*	*1*	*10*	*13*	*17*	*2*	*23*	*20*	*26*	*21*	*16*	*12*	*6*	*30*	*14*	*24*	*15*	*28*	*27*	*9*	*19*	*22*	*25*	*3*	*5*	*7*	*11*
45)	*8*	*4*	*18*	*29*	*1*	*10*	*13*	*17*	*2*	*23*	*20*	*26*	*21*	*16*	*12*	*6*	*14*	*30*	*24*	*15*	*28*	*27*	*9*	*19*	*22*	*25*	*3*	*5*	*7*	*11*
48)	*8*	*4*	*18*	*29*	*1*	*10*	*13*	*17*	*2*	*23*	*20*	*26*	*21*	*12*	*16*	*6*	*14*	*30*	*24*	*15*	*28*	*27*	*9*	*19*	*22*	*25*	*3*	*5*	*7*	*11*
46)	*8*	*4*	*18*	*29*	*1*	*10*	*13*	*17*	*2*	*23*	*20*	*26*	*21*	*12*	*16*	*6*	*14*	*30*	*24*	*15*	*27*	*28*	*9*	*19*	*22*	*25*	*3*	*5*	*7*	*11*

47)	*8*	*4*	*18*	*1*	*29*	*10*	*13*	*17*	*2*	*23*	*20*	*26*	*21*	*12*	*16*	*6*	*14*	*30*	*24*	*15*	*27*	*28*	*9*	*19*	*22*	*25*	*3*	*5*	*7*	*11*
3)	*8*	*4*	*18*	*1*	*29*	*10*	*13*	*17*	*2*	*23*	*26*	*20*	*21*	*12*	*16*	*6*	*14*	*30*	*24*	*15*	*27*	*28*	*9*	*19*	*22*	*25*	*3*	*5*	*7*	*11*
42)	*8*	*4*	*18*	*1*	*29*	*10*	*13*	*17*	*2*	*23*	*26*	*20*	*21*	*12*	*16*	*6*	*14*	*30*	*24*	*27*	*15*	*28*	*9*	*19*	*22*	*25*	*3*	*5*	*7*	*11*
40)	*8*	*4*	*18*	*1*	*29*	*10*	*13*	*17*	*2*	*23*	*26*	*20*	*21*	*12*	*16*	*6*	*14*	*30*	*24*	*27*	*15*	*9*	*28*	*19*	*22*	*25*	*3*	*5*	*7*	*11*
41)	*8*	*4*	*18*	*1*	*29*	*10*	*13*	*17*	*23*	*2*	*26*	*20*	*21*	*12*	*16*	*6*	*14*	*30*	*24*	*27*	*15*	*9*	*28*	*19*	*22*	*25*	*3*	*5*	*7*	*11*

35)	*8*	*4*	*18*	*1*	*29*	*10*	*13*	*17*	*23*	*2*	*26*	*20*	*12*	*21*	*16*	*6*	*14*	*30*	*24*	*27*	*15*	*9*	*28*	*19*	*22*	*25*	*3*	*5*	*7*	*11*
38)	*8*	*4*	*18*	*1*	*29*	*10*	*13*	*17*	*23*	*26*	*2*	*20*	*12*	*21*	*16*	*6*	*14*	*30*	*24*	*27*	*15*	*9*	*28*	*19*	*22*	*25*	*3*	*5*	*7*	*11*
39)	*8*	*4*	*18*	*1*	*29*	*10*	*13*	*17*	*23*	*26*	*2*	*20*	*12*	*16*	*21*	*6*	*14*	*30*	*24*	*27*	*15*	*9*	*28*	*19*	*22*	*25*	*3*	*5*	*7*	*11*
36)	*8*	*4*	*18*	*1*	*29*	*10*	*13*	*17*	*26*	*23*	*2*	*20*	*12*	*16*	*21*	*6*	*14*	*30*	*24*	*27*	*15*	*9*	*28*	*19*	*22*	*25*	*3*	*5*	*7*	*11*
37)	*8*	*4*	*18*	*1*	*29*	*10*	*13*	*26*	*17*	*23*	*2*	*20*	*12*	*16*	*21*	*6*	*14*	*30*	*24*	*27*	*15*	*9*	*28*	*19*	*22*	*25*	*3*	*5*	*7*	*11*

26)	*8*	*4*	*18*	*1*	*29*	*10*	*13*	*26*	*17*	*23*	*2*	*20*	*12*	*16*	*21*	*6*	*14*	*30*	*27*	*24*	*15*	*9*	*28*	*19*	*22*	*25*	*3*	*5*	*7*	*11*
34)	*8*	*4*	*18*	*1*	*29*	*10*	*13*	*26*	*17*	*23*	*2*	*12*	*20*	*16*	*21*	*6*	*14*	*30*	*27*	*24*	*15*	*9*	*28*	*19*	*22*	*25*	*3*	*5*	*7*	*11*
30)	*8*	*4*	*18*	*1*	*29*	*10*	*13*	*26*	*17*	*23*	*2*	*12*	*20*	*16*	*21*	*6*	*14*	*30*	*27*	*24*	*9*	*15*	*28*	*19*	*22*	*25*	*3*	*5*	*7*	*11*
33)	*8*	*4*	*18*	*1*	*29*	*10*	*13*	*26*	*17*	*23*	*2*	*12*	*16*	*20*	*21*	*6*	*14*	*30*	*27*	*24*	*9*	*15*	*28*	*19*	*22*	*25*	*3*	*5*	*7*	*11*
31)	*8*	*4*	*18*	*1*	*29*	*10*	*13*	*26*	*17*	*23*	*12*	*2*	*16*	*20*	*21*	*6*	*14*	*30*	*27*	*24*	*9*	*15*	*28*	*19*	*22*	*25*	*3*	*5*	*7*	*11*

32)	*8*	*4*	*18*	*1*	*29*	*10*	*13*	*26*	*17*	*23*	*12*	*2*	*16*	*20*	*21*	*6*	*14*	*30*	*27*	*24*	*9*	*15*	*28*	*19*	*22*	*25*	*3*	*5*	*11*	*7*
27)	*8*	*4*	*18*	*1*	*29*	*10*	*13*	*26*	*23*	*17*	*12*	*2*	*16*	*20*	*21*	*6*	*14*	*30*	*27*	*24*	*9*	*15*	*28*	*19*	*22*	*25*	*3*	*5*	*11*	*7*
28)	*8*	*4*	*18*	*1*	*10*	*29*	*13*	*26*	*23*	*17*	*12*	*2*	*16*	*20*	*21*	*6*	*14*	*30*	*27*	*24*	*9*	*15*	*28*	*19*	*22*	*25*	*3*	*5*	*11*	*7*
29)	*8*	*4*	*18*	*1*	*10*	*29*	*13*	*26*	*23*	*17*	*12*	*16*	*2*	*20*	*21*	*6*	*14*	*30*	*27*	*24*	*9*	*15*	*28*	*19*	*22*	*25*	*3*	*5*	*11*	*7*
4)	*8*	*4*	*18*	*1*	*10*	*29*	*13*	*26*	*23*	*17*	*12*	*16*	*2*	*20*	*21*	*6*	*14*	*27*	*30*	*24*	*9*	*15*	*28*	*19*	*22*	*25*	*3*	*5*	*11*	*7*

25)	*8*	*4*	*18*	*1*	*10*	*29*	*13*	*26*	*23*	*12*	*17*	*16*	*2*	*20*	*21*	*6*	*14*	*27*	*30*	*24*	*9*	*15*	*28*	*19*	*22*	*25*	*3*	*5*	*11*	*7*
23)	*8*	*4*	*18*	*1*	*10*	*29*	*13*	*26*	*23*	*12*	*17*	*16*	*2*	*20*	*21*	*6*	*14*	*27*	*30*	*9*	*24*	*15*	*28*	*19*	*22*	*25*	*3*	*5*	*11*	*7*
24)	*8*	*4*	*18*	*1*	*10*	*29*	*13*	*26*	*23*	*12*	*17*	*16*	*2*	*20*	*21*	*14*	*6*	*27*	*30*	*9*	*24*	*15*	*28*	*19*	*22*	*25*	*3*	*5*	*11*	*7*
14)	*8*	*4*	*18*	*1*	*10*	*29*	*13*	*26*	*23*	*12*	*17*	*16*	*2*	*20*	*14*	*21*	*6*	*27*	*30*	*9*	*24*	*15*	*28*	*19*	*22*	*25*	*3*	*5*	*11*	*7*
22)	*8*	*4*	*18*	*1*	*10*	*29*	*13*	*26*	*23*	*12*	*17*	*16*	*2*	*14*	*20*	*21*	*6*	*27*	*30*	*9*	*24*	*15*	*28*	*19*	*22*	*25*	*3*	*5*	*11*	*7*

20)	*8*	*4*	*18*	*1*	*10*	*29*	*13*	*26*	*23*	*12*	*17*	*16*	*2*	*14*	*20*	*6*	*21*	*27*	*30*	*9*	*24*	*15*	*28*	*19*	*22*	*25*	*3*	*5*	*11*	*7*
21)	*8*	*4*	*18*	*1*	*10*	*29*	*13*	*26*	*23*	*12*	*17*	*16*	*14*	*2*	*20*	*6*	*21*	*27*	*30*	*9*	*24*	*15*	*28*	*19*	*22*	*25*	*3*	*5*	*11*	*7*
15)	*8*	*4*	*18*	*1*	*10*	*29*	*13*	*26*	*23*	*12*	*17*	*16*	*14*	*2*	*6*	*20*	*21*	*27*	*30*	*9*	*24*	*15*	*28*	*19*	*22*	*25*	*3*	*5*	*11*	*7*
19)	*8*	*4*	*18*	*1*	*10*	*29*	*13*	*26*	*23*	*12*	*17*	*16*	*14*	*2*	*6*	*21*	*20*	*27*	*30*	*9*	*24*	*15*	*28*	*19*	*22*	*25*	*3*	*5*	*11*	*7*
16)	*8*	*4*	*18*	*1*	*10*	*29*	*13*	*26*	*23*	*12*	*17*	*16*	*14*	*6*	*2*	*21*	*20*	*27*	*30*	*9*	*24*	*15*	*28*	*19*	*22*	*25*	*3*	*5*	*11*	*7*

17)	*8*	*4*	*18*	*1*	*10*	*29*	*13*	*26*	*23*	*12*	*17*	*16*	*14*	*6*	*21*	*2*	*20*	*27*	*30*	*9*	*24*	*15*	*28*	*19*	*22*	*25*	*3*	*5*	*11*	*7*
18)	*8*	*4*	*18*	*1*	*10*	*29*	*13*	*26*	*12*	*23*	*17*	*16*	*14*	*6*	*21*	*2*	*20*	*27*	*30*	*9*	*24*	*15*	*28*	*19*	*22*	*25*	*3*	*5*	*11*	*7*
5)	*8*	*4*	*18*	*1*	*10*	*29*	*13*	*26*	*12*	*23*	*17*	*16*	*14*	*6*	*21*	*20*	*2*	*27*	*30*	*9*	*24*	*15*	*28*	*19*	*22*	*25*	*3*	*5*	*11*	*7*
13)	*8*	*4*	*18*	*1*	*10*	*29*	*26*	*13*	*12*	*23*	*17*	*16*	*14*	*6*	*21*	*20*	*2*	*27*	*30*	*9*	*24*	*15*	*28*	*19*	*22*	*25*	*3*	*5*	*11*	*7*
12)	*8*	*4*	*18*	*1*	*10*	*29*	*26*	*13*	*12*	*23*	*17*	*16*	*14*	*6*	*21*	*20*	*27*	*2*	*30*	*9*	*24*	*15*	*28*	*19*	*22*	*25*	*3*	*5*	*11*	*7*

10)	*8*	*4*	*18*	*1*	*10*	*29*	*26*	*13*	*12*	*23*	*17*	*16*	*14*	*6*	*21*	*27*	*20*	*2*	*30*	*9*	*24*	*15*	*28*	*19*	*22*	*25*	*3*	*5*	*11*	*7*
11)	*8*	*4*	*18*	*1*	*10*	*29*	*26*	*13*	*12*	*23*	*17*	*16*	*14*	*6*	*27*	*21*	*20*	*2*	*30*	*9*	*24*	*15*	*28*	*19*	*22*	*25*	*3*	*5*	*11*	*7*
6)	*8*	*4*	*18*	*1*	*10*	*29*	*26*	*13*	*12*	*23*	*16*	*17*	*14*	*6*	*27*	*21*	*20*	*2*	*30*	*9*	*24*	*15*	*28*	*19*	*22*	*25*	*3*	*5*	*11*	*7*
9)	*8*	*4*	*18*	*1*	*10*	*29*	*26*	*13*	*12*	*23*	*16*	*17*	*14*	*27*	*6*	*21*	*20*	*2*	*30*	*9*	*24*	*15*	*28*	*19*	*22*	*25*	*3*	*5*	*11*	*7*
7)	*8*	*4*	*18*	*1*	*10*	*29*	*26*	*13*	*12*	*23*	*16*	*17*	*14*	*27*	*6*	*21*	*20*	*30*	*2*	*9*	*24*	*15*	*28*	*19*	*22*	*25*	*3*	*5*	*11*	*7*

8)	*8*	*4*	*18*	*1*	*10*	*29*	*26*	*13*	*12*	*23*	*16*	*17*	*14*	*27*	*6*	*21*	*30*	*20*	*2*	*9*	*24*	*15*	*28*	*19*	*22*	*25*	*3*	*5*	*11*	*7*
2)	*8*	*4*	*18*	*1*	*10*	*29*	*26*	*13*	*12*	*23*	*16*	*17*	*14*	*27*	*6*	*21*	*30*	*20*	*2*	*9*	*15*	*24*	*28*	*19*	*22*	*25*	*3*	*5*	*11*	*7*

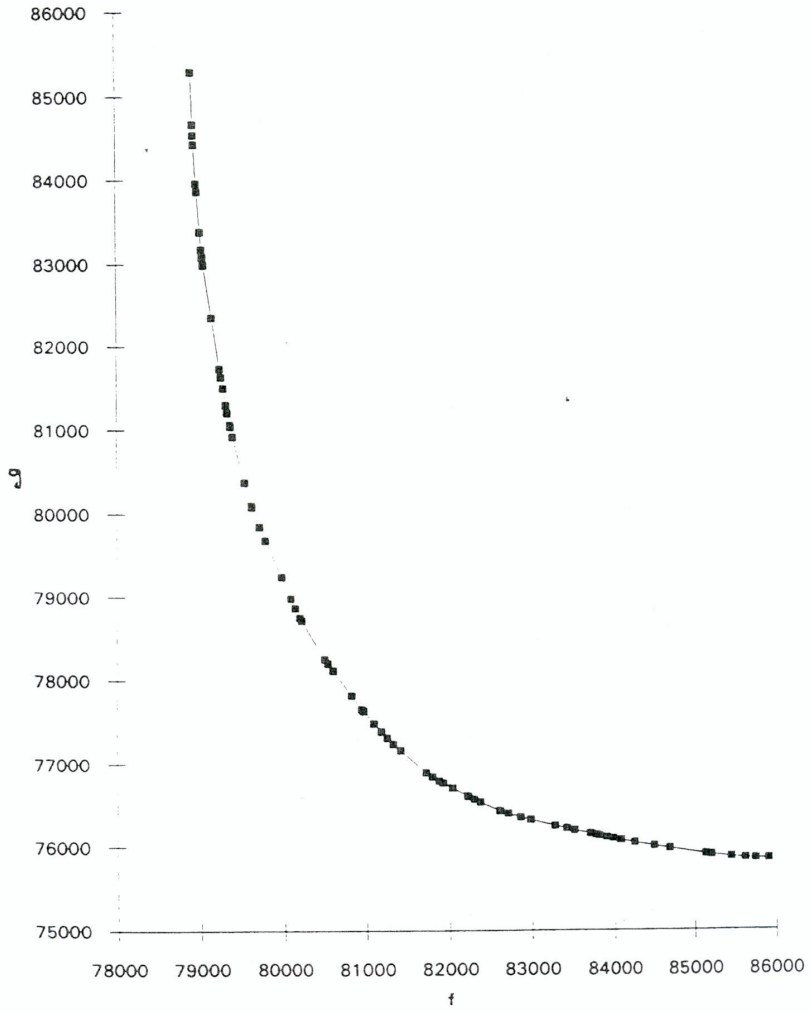
En este ejemplo concreto, al obtener los puntos eficientes extremos se observa que sólo hay un punto eficiente que no es eficiente extremo. Las coordenadas, o valores de los objetivos, para dicho punto son (79372, 81039). En el proceso de generación, este punto ocupa el número de orden 57 entre los puntos generados. La planificación correspondiente a dicho punto es:

57)	*8*	*4*	*18*	*29*	*1*	*10*	*13*	*2*	*17*	*23*	*20*	*21*	*16*	*26*	*12*	*6*	*30*	*24*	*14*	*28*	*15*	*19*	*9*	*22*	*27*	*25*	*3*	*5*	*7*	*11*

El tiempo necesario para calcular 72 puntos eficientes por un AT 80486 a 33 Mhz. ha sido de 0.2197800 segundos CPU.

La gráfica de los puntos eficientes se muestra a continuación.

eficientes


86000
85000
84000
83000
82000
81000
80000
79000
78000
77000
76000
75000
78000
79000
80000
81000
82000
83000
84000
85000
86000
f

Bibliografía

- Adiri, I., N. Aizikowitz (Hefetz) (1989) *"Open shop scheduling problems with dominated machines"* Naval Research Logistic Quarterly, 36, 273-281.
- Adiri, I., N. Amit (1983) *"Route dependent open-shop scheduling"* IIE Transaction, 15, 231-234.
- Adiri, I., N. Hefetz (1980) *"Subproblems of open shop more than two machines schedule length problem"* Operations Research, Statistics and Economics Mimeograph Series, 260, Techion, Haifa.
- Adolphson, D., T.C. Hu (1973) *"Optimal linear ordering"* SIAM J. Appl. Math. 25, 403 - 423.
- Aho, A.H., J. Hopcroft, J. Ullman (1974) *"The design and analysis of computer algorithms"* Addison-Wesley, Reading, Mass.
- Alcaide, D. (1991) *"Problemas de planificación y secuenciación"* Memoria de Licenciatura, DEIOC, Universidad de La Laguna.
- Alcaide, D., J. Sicilia (1990) *"Secuenciación de trabajos sobre una máquina con restricciones de precedencia"* Revista de la Academia Canaria de Ciencias, vol. I, pp. 217-226.
- Alcaide, D., J. Sicilia, M.T. Ramos (1992) *"The total tardiness problem with precedence constraints: an heuristic approach"* Third International Workshop on Project Management and Scheduling. Como. Italy.
- Alcaide, D., J. Sicilia, D. Vigo (1992) *"Heuristic approaches for the minimum makespan open shop problem"* AIRO'92, Acireale, Italia.
- Baker, K.R. (1974) *"Introduction to sequencing and scheduling"* John Wiley.
- Baker, K.R., E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan (1983) *"Preemptive scheduling of a single-machine to minimize maximum cost subject to release dates and precedence constraints"* Oper. Res., vol. 31, n.2, pp. 381-386.
- Baker, K.R., J.B. Martin (1974) *"An experimental comparison of solution algorithms for the single-machine tardiness problem"* Naval Research Logistic Quarterly, pp. 187-199.
- Baker, K.R., G.D. Scudder (1990) *"Sequencing with earliness and tardiness penalties: a review"* Operations Research, vol. 38, no. 1, pp. 22-36
- Baker, K.R., Z-S. Su (1974) *"Sequencing with due-dates and early start times to minimize maximum tardiness"* Naval Research Logistic Quarterly. vol 21, n.1, pp. 171-176.
- Balas, E. (1969) *"Machine sequencing via disjuntive graphs: an implicit enumeration algorithm"*. Operations Research 17, 941-957.
- Balas, E. (1985) *"On the facial structure of scheduling polyhedra"* Mathematical Programming Study, vol 24, pp. 178-218.
- Balcázar, J.L., J. Díaz, J. Gabarró (1988) *"Structural Complexity I"* EATCS Monographs on Theoretical Computer Science, vol. 11, Springer-Verlag.

- Barnes, J.W., J.J. Brennan (1977) *"An improved algorithm for scheduling jobs on identical machines"* AIIE Trans. 9, 25-31.
- Bianco, L., S. Ricciardelli (1981) *"Scheduling of a single machine to minimize total weighted completion time subject to release dates"* Istituto di Analisi dei Sistemi ed Informatica, CNR, Roma.
- Blazewicz, J. (1987) *"Selected topics in scheduling theory"* Annals of Discrete mathematics, vol 31, pp. 1-60.
- Blazewicz, J., W. Cellary, R. Slowinski, J. Weglarz (1976) *"Deterministyczne problemy szaregowania zadan na rownoleglych procesorach"* Cz. I. Zbiory Zadan Niezaleznych Podstawy Sterowania 6, 155-178.
- Blazewicz, J., W. Cellary, J. Weglarz (1977) *"A strategy for scheduling splittable tasks to reduce schedule length"* Acta Cybernet. 3, 99-106.
- Blazewicz, J., K. Ecker, G. Schmidt, J. Weglarz (1993) *"Scheduling in computer and manufacturing systems"* Springer-Verlag.
- Brucker, P., M.R. Garey, D.S. Johnson (1977) *"Scheduling equal length tasks under tree like precedence constraints to minimize maximum lateness"* Math. Oper. Res. 2, 275-284.
- Bruno, J., E.G. Coffman Jr., R. Sethi (1974) *"Scheduling independent tasks to reduce mean finishing time"* Comm. ACM 17, 382-387.
- Carlier, J. (1980) *"Probleme a une machine"* Manuscript, Institut de Programation, Universite Paris VI.
- Carlier, J., E. Pinson (1989) *"An algorithm for solving the job shop problem"* Management Science 35, 164-176.
- Cheng, T.C.E., C.C.S. Sin (1990) *"A state of the art review of parallel machine scheduling research"*. European J. Oper. Res. 47, 271-292.
- Cho, Y., S. Sahni (1978) *"Preemptive scheduling of independent jobs with release and due times on open, flow and job shops"* Technical Report 78-5, Computer Science Departament, University of Minnesota, Minneapolis.
- Church, A. (1933) *"A set of postulates for the foundation of logic"* Annals of Mathematics 25, 839-864.
- Church, A. (1936) *"An unsolvable problem of elementary number theory"* The American Journal of Mathematics 58, 345-363.
- Coffman, E.G. (ed.) (1976) *"Computer and job shop scheduling theory"* Wiley, New York.
- Coffman, E.G., M. R. Garey, D.S. Johnson (1978) *"An application of bin packing to multiprocessor scheduling"* SIAM J. Comput. 7, pp. 1-17.
- Coffman, E.G., R.L. Graham (1972) *"Optimal scheduling for two processor systems"* Acta Informatica 1, 200-213.
- Conway, R.W. , W.L. Maxwell, L.W. Miller (1967) *"Theory of scheduling"* Addison Wesley, Reading, Mass.
- Cook, S. (1971) *"The complexity of theorem proving procedures"* In Proc. 3rd ACM Symposium on the Theory of Computing, pp. 151-158.
- Daniels, R.L., J.B. Mazzola (1994) *"Flow shop scheduling with resource flexibility"* Oper. Res. vol. 42, pp. 504-522.

- Dell'Amico, M., S. Martello (1994) *"Optimal scheduling of tasks on identical parallel processors"* por aparecer en ORSA Journal on Computing.
- Dell'Amico, M., S. Martello, D. Vigo (1995) *"Heuristic algorithms for single processor scheduling with earliness and flow time penalties"* Internal Report. DEIS, Universita degli Studi di Bologna. Italia.
- Dell'Amico, M., M. Trubian (1993) *"Applying tabu search to the job shop scheduling problem"* Annals of the Operations Research, vol. 41, 231-252.
- Dietrich, B., L.F. Escudero (1990) *"Coefficient reduction for knapsack-like constraints in 0-1 programs with variable upper bounds"*. Operations Research Letters 9, 9-14.
- Dijkstra, E.W. (1959) *"A note on two problems in connection with graphs",* Numerische Mathematik, vol. 1, pp. 269.
- Dolev, D. (1981) *"Scheduling wide graphs"*
- Du, J., J. Y.-T. Leung (1990) *"Minimizing total tardiness on one machine is NP-hard"* Math. of Oper. Res. vol. 15, n. 3, 483-495.
- Eastman, W.L., S. Even, I.M. Isaacs (1964) *"Bounds for the optimal scheduling of n jobs on m processors"* Management Science 11, 268-279.
- Elmaghraby, S.E., S.H. Park (1974) *"Scheduling jobs on a number of identical machines"* A.I.I.E. Trans. 6, 1-12.
- Emmons, H. (1969) *"One-machine sequencing to minimize certain functions of job tardiness"* Oper. Res. vol 17, pp. 701-715.
- Emmons, H. (1975) *"A note on a scheduling problem with dual criteria"* Naval Research Logistic Quarterly 22, 615-616.
- Escudero, L.F., G. Pérez (1990) *"Strategies for lp-based solving a general class of scheduling problems"*. Trabajos de Investigación Operativa, 5, 3-33.
- Espinel, C. (1990) *"Cuestiones notables sobre optimización combinatoria: problemas de recubrimiento y planificación."* Tesis Doctoral, DEIOC, Universidad de La Laguna.
- Fischetti, M.; S. Martello; P. Toth (1987) *"The fixed job-schedule problem with spread-time constraints"* Oper. Res., vol. 35, 849-858.
- Fischetti, M.; S. Martello; P. Toth (1989) *"The fixed job schedule problem with working-time constraints"* Oper. Res., vol 37, pp. 395-403.
- Fischetti, M.; S. Martello; P. Toth (1992) *"Approximation algorithms for fixed job schedule problems"* Oper. Res. vol. 40, supp. 1, pp. S96-S108.
- Fischetti, M.; P. Toth (1989) *"An additive bounding procedure for combinatorial optimization problems"* Oper. Res., vol 37, n.2, pp. 319-328.
- Fisher, M.L. (1976) *"A dual algorithm for the one machine scheduling problem"* Math. Programming 11, 229-251.
- French, S. (1982) *"Sequencing and scheduling, an introduction to the mathematics of the job shop"* Ellis Horwood Series.
- Fry, T.D., L. Vicens, K. Macleod, S. Fernández (1989) *"A heuristic solution procedure to minimize $\overline{T}$ on a single machine"* Journal of the Operations Research Society, vol. 40, n. 3, pp. 293-297.
- Fujii, M., T. Kasami, K. Ninomiya (1969, 1971) *"Optimal sequencing of two equivalent processors"* SIAM J. Appl. Math. 17, 784-789, errores 20, 141.

- Gabow, H.N. (1980) *"An almost - linear algorithm for two processors scheduling"* Technical Report CU-CS-169-80 Departament of Computer Science, University of Colorado.
- Garey, M.R., D.S. Johnson (1976) *"Scheduling tasks with non uniform deadlines on two processors"* J. Assoc. Comput. Mach. 23, 461-467.
- Garey, M.R. , D.S. Johnson (1979) *"Computers and intractability: a guide of the theory of NP-completeness"* Freeman.
- Garey, M.R., D.S. Johnson, R. Sethi (1976) *"The complexity of flowshop and jobshop scheduling"* Math. Oper. Res. 1, 117-129.
- Garey, M.R., D.S. Johnson, B.B. Simons, R.E. Tarjan (1981) *"Scheduling unit-time tasks with arbitrary release times and deadlines"* Siam J. of Comput. vol 10, n.2, pp. 256-269.
- Garey, M.R., R.E. Tarjan, G.T. Wilfong (1988) *"One processor scheduling with earliness and tardiness penalties"* Mathematics of Operations Research 13, 330-348.
- Gelders, L., P.R. Kleindorfer (1974) *"Coordinating aggregate and detailed scheduling decisions in the one machine job shop: part I Theory"* Oper. Res. 22, 46-60.
- Gelders, L., P.R. Kleindorfer (1975) *"Coordinating aggregate and detailed scheduling decisions in the one machine job shop: part II Computation and structure"* Oper. Res. 23, 312-324.
- Geoffrion, A. M. (1974) *"Lagrangean relaxations for integer programming"* En *"Mathematical Programming Study 2: Approaches to integer programming."* M.L. Balinski (ed.) North-Holland, Amsterdam, 82-114.
- González, C. (1986) *"Métodos interactivos en programación multiobjetivo"* Tesis Doctoral, DEIOC, Universidad de La Laguna.
- González, T. (1977) *"Optimal mean finish time preemptive schedules"* Technical Report 220, Computer Science Departament, Pennsylvania State University.
- González, T. (1979) *"A note on open shop preemptive schedules"* IEEE Trans. Computers C-28, 782.786.
- González, T. D.S. Johnson (1980) *"A new algorithm for preemptive scheduling of trees"* J. Assoc. Comput. Mach. 27, 287-312.
- González, T., E.L. Lawler, S. Sahni (1990) *"Optimal preemptive scheduling of two unrelated processors"* ORSA Journal of Computing, 2, 219-224.
- González, T., S. Sahni (1976) *"Open shop scheduling to minimize finish time"* J. Assoc. Comput. Mach. 23, 665-679.
- González, T., S. Sahni (1978a) *"Flowshop and Jobshop schedules: complexity and approximation"* Oper. Res., vol 26, n.1, pp. 36-52.
- González, T., S. Sahni (1978b) *"Preemptive scheduling of uniform processor systems"* J. Assoc. Comput. Mach. 25, 92-101.
- Goyal, D.K. (1977) *"Non preemptive scheduling of unequal execution time tasks on two identical processors"* Technical Report CS-77-039, Computer Science Departament, Washington State University, Pullman.

- Graham, R.L.; E.L. Lawler; J.K. Lenstra; A.H.G. Rinnooy Kan (1979) *"Optimization and approximation in deterministic sequencing and scheduling: a survey"* Annals of discrete mathematics, vol 5, pp. 287-326.
- Graves, S.C., A.H.G. Rinnooy Kan, P.H. Zipkin (eds.) (1993) *"Logistics of production and inventory"* Handbooks in Operations Research and Management Science, vol. 4, North Holland.
- Gupta, J.N.D. (1986) *"Flowshop schedules with sequence dependent setup times"* Journal of the Operations Research Society of Japan, vol 29, n.3, pp. 206-219.
- Gupta, J.N.D., S.S. Reddi (1978) *"Improved dominance conditions for the three-machine flowshop scheduling problem"* Oper. Res., vol 26, n.1, pp. 200-208.
- Gupta, U.I., D.T. Lee, J.Y-T. Leung (1979) *"An optimal solution for the channel assignment problem"* IEEE Trans. Comput. C-28, 807-810.
- Hariri, A.M.A., C.N. Potts (1981) *"An algorithm for single machine sequencing with release dates to minimize total weighted completion time"* Report BW 143, Mathematisch Centrum, Amsterdam.
- Hefetz, N., I. Adiri (1979) *"An efficient optimal algorithm for the two machines, unit time, job shop, schedule length, problem"* Oper. Res. Statistics and Economics Mimeograph Series 237, Technion, Haifa.
- Hochbaum, D.S., D.B. Shmoys (1987) *"Using dual approximation algorithms in scheduling"* Journal of the ACM, 34, 144-162.
- Holsenback, J.E., R.M. Russell (1992) *"A heuristic algorithm for sequencing on one machine to minimize total tardiness"* Journal of the Operations Research Society, vol. 43, n. 1, pp. 53-62.
- Hoogeveen, J.A. (1990) *"Minimizing maximum earliness and maximum lateness on a single machine"* Report BS-R9001 CWI, Amsterdam.
- Hoogeveen, J.A. (1992a) *"Minimizing maximum promptness and maximum lateness on a single machine"* Internal Report. Eindhoven University of Technology. Eindhoven. Holanda
- Hoogeveen, J.A. (1992b) *"Single-machine scheduling to minimize a function of K maximum cost criteria"* Internal Report. Eindhoven University of Technology. Eindhoven. Holanda
- Hoogeveen, J.A., S.L. van de Velde (1992a) *"Polynomial-time algorithms for single-machine bicriteria scheduling"* Internal Report. Eindhoven University of Technology. Eindhoven. Holanda
- Hoogeveen, J.A., S.L. van de Velde (1992b) *"A new lower bound approach for single-machine multicriteria scheduling"* Internal Report. Eindhoven University of Technology. Eindhoven. Holanda
- Horn, W.A. (1972) *"Single machine job sequencing with treelike precedence ordering and linear delay penalties"* SIAM J. Appl. Math. 23, 189 - 202.
- Horn, W.A. (1973) *"Minimizing average flow time with parallel machines"* Oper. Res. 21, 846-847.
- Horn, W.A. (1974) *"Some simple scheduling algorithms"* Naval Res. Log. Quart. 21, 177-185.

- Horvath, E.C., S. Lam, R. Sethi (1977) *"A level algorithm for preemptive scheduling"* J. Assoc. Comput. Mach. 24, 32-43.
- Horowitz, E., S. Sahni (1976) *"Exact and approximate algorithms for scheduling non-identical processors"* J. Assoc. Comput. Mach. 23, 317-327.
- Hsu, N.C. (1966) *"Elementary proof of Hu'theorem on isotone mappings"* Proc. Amer. Math. Soc. 17, 111-114.
- Hu, T.C. (1961) *"Parallel sequencing and assembly line problems"* Oper. Res. 9, 841-848.
- Ibarra, O.H., C.E. Kim (1978) *"Aproximation algorithms for certain scheduling problems"* Math. Oper. Res. 3, 197-204.
- Ishii, H., T. Nishida (1986) *"Two machine open-shop scheduling problem with controllable machine speeds"* Journal of the Operations Research Society of Japan, vol 29, n.3, pp. 123-131.
- Jackson, J.R. (1955) *"Scheduling a production line to minimize maximum tardiness"* Research Report 43, Management Science Research Project, UCLA.
- Jackson, J.R. (1956) *"An extension of Johnson results on job lot scheduling"* Naval Research Logistic Quarterly, 3, 201-203.
- Jaffe, J.M. (1980) *"An analysis of preemptive multiprocessor job scheduling"* Math. Oper. Res. 5, 415-421.
- Johnson, S.M. (1954) *"Optimal two and three stage production schedules with setup includes"* Naval Research Logistic Quarterly, 1, 61-68.
- Jorge, J.M. (1992) *"Aspectos metodológicos y computacionales de la optimización multiobjetivo. El caso lineal"* Memoria de Licenciatura, DEIOC, Universidad de La Laguna.
- Karp, R.M. (1972) *"Reducibility among combinatorial problems"* En R.E. Miller, J.W. Thatcher (eds.) (1972) *"Complexity of computer computations"* Plenum Press, New York, 85-103.
- Khachiyan, L.G. (1979) *"A polynomial algorithm in linear programming"* Soviet Math. Dokl. 20, 191-194.
- Kise, H., T. Ibaraki, H. Mine (1978) *"A solvable case of the one machine scheduling problem with ready and due times"* Oper. Res. 26, 121-126.
- van Laarhoven,, P.J.M., E.H.L. Aarts, J.K. Lenstra (1988) *"Job shop scheduling by simulated annealing"* Report OS-R8809, Centre for Mathematics and Computer Science. Amsterdam.
- Labetoulle, J., E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan (1979) *"Preemptive scheduling of uniform machines subject to release dates"* Report BW 99, Mathematisch Centrum, Amsterdam.
- Lageweg, B.J., J.K. Lenstra, A.H.G. Rinnooy Kan (1976) *"Minimizing maximum lateness on one machine: computational experience and some applications"* Statist. Neerlandica 30, 25-41.
- Lageweg, B.J., J.K. Lenstra, A.H.G. Rinnooy Kan (1978) *"A general bounding scheme for the permutation flow-shop problem"* Oper. Res., vol 26, n.1, pp. 53-67.
- Lam, S., R. Sethi (1977) *"Worst case analysis of two scheduling algorithms"* Siam J. of Computing, vol 6, n.3, pp. 518-536.

- Lawler, E.L. (1973) *"Optimal sequencing of a single machine subject precedence constraints"* Management Science 19, 544 - 546.
- Lawler, E.L. (1976a) *"Combinatorial optimization: networks and matroids"* Holt, Rinehart and Winston, New York.
- Lawler, E.L. (1976b) *"Sequencing to minimize the weighted number of tardy jobs"* RAIRO Rech. Oper. 10.5 Suppl. 27-33.
- Lawler, E.L. (1977) *"A pseudopolynomial algorithm for sequencing jobs to minimize total tardiness"* Annals of Discrete Mathematics, vol. 1, pp. 331-342.
- Lawler, E.L. (1978) *"Sequencing jobs to minimize total weighted competion time subject to precedence constraints"* Ann. Discrete Math. 2, 75 - 90.
- Lawler, E.L. (1979) *"Efficient implementation of dynamic programming algorithms for sequencing problems"* Report BW 106, Mathematisch Centrum, Amsterdam.
- Lawler, E.L. (1980) *"Preemptive scheduling of precedence-constraints jobs on parallel machines"* Report BW 132, Mathematisch Centrum, Amsterdam.
- Lawler, E.L. (1981) *"Preemptive scheduling of uniform parallel machines to minimize the number of late jobs"* Report, Mathematisch Centrum, Amsterdam.
- Lawler, E.L. (1982a) *"A fully polinomial approximation scheme for the total tardiness problem"* Oper. Res. Lett. 1, 207-208.
- Lawler, E.L. (1982b) *"Scheduling a single machine to minimize the number of late jobs"* Preprint, Computer Science Division, Univ. of California, Berkeley.
- Lawler, E.L. (1983) *"Recent results in the theory of machine scheduling"* XI International Symposium on Mathematical Programming, Bonn 1982, A. Bachem et at all eds., pp. 202-234.
- Lawler, E.L., J. Labetoulle (1978) *"On preemptive scheduling of unrelated parallel processors by linear programming"* J. Assoc. Comput. Mach. 25, 612-619.
- Lawler, E.L.; J.K. Lenstra (1982) *"Machine scheduling with precedence constraints"* I.Rival (ed.) Ordered Sets, pp. 655-675.
- Lawler, E.L., J.K. Lenstra, A.H.G. Rinnooy Kan (1981) *"Minimizing maximum lateness in a two machine open shop"* Math. Oper. Res. vol. 6, 153-158.
- Lawler, E.L., J.K. Lenstra, A.H.G. Rinnooy Kan (1982) *"Recent developments in deterministic sequencing and scheduling: a survey"* en M.A.H. Dempster, J.K. Lenstra, A.H.G. Rinnooy Kan (eds.) *"Deterministic and stochastic scheduling"* NATO Advanced Study Institutes Series, serie C, vol. 84.
- Lawler, E.L., J.K. Lenstra, A.H.G. Rinnooy Kan, D.B. Shmoys (1993) *"Sequencing and scheduling: algorithms and complexity"* En S.C. Graves, A.H.G. Rinnooy Kan, P.H. Zipkin (eds.) *"Handbooks in Operations Research and Management Science",* capítulo 9, vol. 4, North Holland.
- Lawler, E.L., C.U. Martel (1980) *"Computing polymatroidal network flows"* Research Memorandum ERL M80/52, Electronic Research Laboratory, University of California, Berkeley.
- Lawler, E.L.; C.U. Martel (1989) *"Preemtive scheduling of two uniform machines to minimize the number of late jobs"* Oper. Res., vol 37, n.2, pp. 314-318.

- Lawler, E.L., J.M. Moore (1969) *"A functional equeation and its application to resource allocation and sequencing problems"* Management Science 16, 77-84.
- Lee, C-Y., S.D. Liman (1993) *"Error bound of a heuristic for the common due date scheduling problem"* ORSA Journal of Computing, vol. 5, no. 4, pp. 420-425.
- Lenstra, J.K., A.H.G. Rinnooy Kan (1978) *"Complexity of scheduling under precedence constraints"* Oper. Res., vol 26, n.1, pp. 22-35.
- Lenstra, J.K., A.H.G. Rinnooy Kan (1979) *"Computational complexity of discrete optimization problems"* Ann. Discrete Math. 4, 121-140.
- Lenstra, J.K., A.H.G. Rinnooy Kan (1980) *"Complexity results for scheduling chains on a single machine"* European Journal of Operations Research, 4, 270-275.
- Lenstra, J.K., A.H.G. Rinnooy Kan, P. Brucker (1977) *"Complexity of machine scheduling problems"* Ann. Discrete Math. 1, 343-362.
- Lenstra, J.K., D.B. Shmoys, E. Tardos (1990) *"Approximation algorithms for scheduling unrelated parallel machines"* Math. Prog. 46, 259-271.
- Liu, C.Y., R.L. Bulfin (1987) *"Scheduling ordered open-shop"* Computer and Operations Research, 14, 3, 257-264.
- Martel, C.U. (1981) *"Scheduling uniform machines with release times, deadlines and due times"* J. Assoc. Comput. Mach.
- Martello, S., P. Toth (1986) *"A heuristic approach to the bus driver scheduling problems"* European J. Opnl. Res. 24, 106-117.
- Martello, S., P. Toth (1990) *"Lower bounds and reduction procedures for the bin packing problem"* Discrete Appl. Math. 28, 59-70.
- Matta, R., M. Guignard (1994) *"Dynamic production scheduling for a process industry"* Oper. Res. vol. 42, 492-503.
- Mc Cormick, S.T., M.L. Pinedo (1995) *"Scheduling n independent jobs on m uniform machines with both flowtime and makespan objectives: a parametric analysis"* ORSA Journal on Computing, vo. 7, pp. 63-77.
- McMahon, G.B., M. Florian (1975) *"On scheduling with ready times and due dates to minimize maximum lateness"* Oper. Res. 23, 475 - 482.
- McNaughton, R. (1959) *"Scheduling with deadlines and loss functions"* Management Sci. 6, 1-12.
- Medina, P.A. (1994) *"Estudio comparativo de algoritmos para la resolución de problemas de planificación"* Proyecto fin de Diplomatura, CSI, Universidad de La Laguna.
- Monma, C.L. (1980) *"Sequencing to minimize the maximum job cost"* Oper. Res. 28, 942-951.
- Moore, J.M. (1968) *"An n job, one machine sequencing algorithm for minimizing the number of late jobs"* Management Science 15, 102-109.
- Muntz, R.R., E.G. Coffman Jr. (1969) *"Optimal preemptive scheduling on two processor systems"* IEEE Trans. Computers C-18, 1014-1020.
- Muntz, R.R., E.G. Coffman Jr. (1970) *"Preemptive scheduling of real time tasks on multiprocessor systems"* J. Assoc. Comput. Mach. 17, 324-338.

- Muth, J.F., G.L. Thompson (eds.) (1963) *"Industrial Scheduling"* Prentice Hall, Englewood Cliffs, New Jersey.
- Nakajima, K., S.L. Hakimi, J.K. Lenstra (1982) *"Complexity results for scheduling tasks in fixed intervals on two types of machines"*. SIAM J. Comput. 11, 512-520.
- Nelson, R.T., R.K. Sarin, R.L. Daniels (1986) *"Scheduling with multiple performance measures: the one machine case"* Management Science 32, 464-479.
- Ow, P.S., T.E. Morton (1989) *"The single machine early / tardy problem"* Management Science 35, 177-191.
- Pezzoli, S. (1994) *"Un algoritmo tabu-search per il problema open shop"* Tesi de Laurea, DEIS, Universita degli Studi di Bologna, Italia.
- Potts, C. (1985) *"Analysis of a linear programming heuristic for scheduling unrelated parallel machines"*. Disc. Appl. Math. 10, 155-164.
- Potts, C.N., L.N. Van Wassenhove (1982) *"A decomposition algorithm for the single machine total tardiness problem"* Operations Research Letters, vol 1, n.5, pp. 177-181.
- Potts, C.N., L.N. Van Wassenhove (1987) *"Dynamic programming and decomposition approaches for the single-machine total tardiness problem"* European Journal of Operational Research, vol 32,pp.405-414.
- Ramos, R.M. (1993) *"Cuestiones notables en teoría de grafos y optimización biobjetivo en redes"* Tesis Doctoral. DEIOC, Universidad de La Laguna.
- Rinaldi, G., A. Sassano (1977) *"On a job scheduling problem with different ready times: some properties an a new algorithm to determine the optimal solution"* Report R. 77 - 24, Istituto di Automatica, Universitá di Roma.
- Rinnooy Kan, A.H.G., B.J. Lageweg, J.K. Lenstra (1975) *"Minimizing total costs in one-machine scheduling"* Oper. Res. vol 23, n.5, pp. 908-927.
- Roy, B., B. Sussmann (1964) *"Les problemes d'ordonnancement avec contraintes disjunctives"* Note D.S. n.9. bis, SEMA, Montrouge.
- Sahni, S. (1976) *"Algorithms for scheduling independent tasks"* J. Assoc. Comput. Mach. 23, 116-127.
- Sahni, S., Y. Cho (1979) *"Nearly on line scheduling of a uniform processor system with release times"* Siam J. of Comput. 8, 275-285.
- Schrage, , K.R. Baker (1978) *"Dynamic programming solution of sequencing problems with precedence constraints"* Oper. Res. vol. 26, 444-449.
- Sen, T., S.K. Gupta (1983) *"A branch and bound procedure to solve a bicriterion scheduling problem"* IIE Transactions 15, 84-88.
- Sen, T., F.M.E. Raiszadeh, P. Dileepan (1988) *"A branch and bound approach to the bicriterion scheduling problem involving total flowtime and range of lateness"* Management Science 34, 254-260.
- Sethi, R. (1976) *"Algorithms for minimal length schedules"* en Coffman (de.) (1976) *"Computer and job shop scheduling theory"* Wiley, New York.
- Shanthikumar, J.G. (1983) *"Scheduling n jobs on one machine to minimize the maximum tardiness with minimum number tardy"* Computers and Operations Research 10, 255-266.

- Shwimer, J. (1972) *"On the N-job, one machine, sequence independent scheduling problem with tardiness penalties: a branch and bound solution"* Management Science, vol. 18, B301-313.
- Sicilia, J., D. Alcaide (1989) *"Algoritmo aproximado para la minimización de la tardanza total sobre una máquina"* Actas de las XIV Jornadas Hispano-Lusas de Matemáticas, vol. II, pp. 1061-1067, Puerto de la Cruz, Tenerife.
- Sicilia, J., D. Alcaide (1989b) *"Minimización de la tardanza total considerando diferente disponibilidad de los trabajos"* Actas de la XVIII Reunión Nacional de Estadística, Investigación Operativa e Informática, pp. 483-487, Santiago de Compostela.
- Sicilia, J., D. Alcaide (1990) *"Planificación de trabajos sobre varias máquinas"* Actas de las XV Jornadas Luso-Espanholas de Matematica, pp. 343-348, Evora, Portugal.
- Sidney, J.B. (1973) *"An extension of Moore's due date algorithm"*, en S.E. Elmaghraby (de.) (1973) Symposium on the theory of scheduling and its applications, Lecture Notes in Economics and Mathematical Systems 86, Springer, Berlín, 393-398.
- Sidney, J.B. (1975) *"Decomposition algorithms for single machine sequencing with precedence relations and deferral costs"* Oper. Res. 23, 283 - 298.
- Sidney, J.B. (1979) *"The two machine maximum flow time problem with series parallel precedence relations"* Oper. Res. 27, 782-791.
- Simons, B. (1978) *"A fast algorithm for single processor scheduling"* Proc. 19th Annual IEEE Symp. Foundations of Computer Science.
- Smith, W.E. (1956) *"Various optimizers for single stage production"* Naval Research Logistic Quarterly 3, 59 - 66.
- Spaggiari, A. (1993) *"Algoritmi euristici per i problemi di scheduling"* Tesi de Laurea, DEIS, Universita degli Studi di Bologna. Italia.
- Sundararaghavan, P.S., M.U. Ahmed (1984) *"Minimizing the sum of absolute lateness in single-machine and multimachine scheduling"* Naval Research Logistics Quarterly 31, 325-333.
- Srinivasian, V. (1971) *"A hybrid algorithm for the one machine sequencing problem to minimize total tardiness"* Naval Research Logistic Quarterly, vol. 18, n.3, septiembre.
- Szwarc, W. (1977) *"Optimal two machine orderings in the 3 x n flow shop problem"* Oper. Res. 25, 70 - 77.
- Szwarc, W. (1978) *"Dominance conditions for the three machine flow shop problem"* Oper. Res. 26, 203-206.
- Szwarc, W. (1981) *"Precedence relations of the flow-shop problem"* Oper. Res., vol 29, n.2, pp. 400-411.
- Taillard, E. (1993) *"Benchmarks for basic scheduling problems"* European J. of Operations Research, 64, 278-285.
- Trick, M.A. (1994) *"Scheduling multiple variable speed machines"* Oper. Res. vol. 42, pp. 234-248.
- Ullman, J.D. (1975) *"NP-complete scheduling problems"* J. Comput. System Sci. 10, 384-393.

- Ullman, J.D. (1976) *"Complexity of scheduling problems"* En Coffman (ed.) (1976) 139-164.
- van de Velde, S.L. (1991) *"Machine scheduling and Lagrangian relaxation"* Doctoral Thesis, CWI, Amsterdam.
- Warmuth, M.K. (1980) *"M processor unit execution time scheduling reduce to M-1 weakly connected components"* M.S. Thesis, Departament of Computer Science, University of Colorado, Boulder.
- van Wassenhove, L.N., F. Gelders (1980) *"Solving a bicriterio scheduling problem"* European Journal of Operations Research 4, 42-48.
- Wilkerson, L.J., J.D. Irwin (1971) *"An improvement algorithm for scheduling independent tasks"* A.I.I.E. Trans. 3, 239-245.