Modelos de optimización en programación de tareas

Víctor Lallana Campos

Trabajo de Fin de Grado en Matemáticas
Universidad de Zaragoza

Directores del trabajo:
Herminia I. Calvete Fernández
José Ángel Iranzo Sanz

28 de junio de 2021

Prólogo

El objeto de estudio de esta memoria es la programación de tareas. Esta consiste en la asignación de diferentes tareas en un periodo determinado de tiempo a un conjunto de máquinas siguiendo una determinada secuencia con el propósito de optimizar alguna función objetivo.

En el primer capítulo de esta memoria se introduce en qué consiste la programación de tareas, así como la notación relacionada más importante, de la que se lleva a cabo una breve introducción. En este mismo capítulo, se presentan distintas clasificaciones dependiendo de las características del estudio. Por último, se considera el problema de programación de tareas con una única máquina en el cual se centran los siguientes capítulos. Para este problema se estudian cuatro modelos cuyos objetivos son minimizar el tiempo de finalización total ponderado de las tareas, minimizar la peor desviación de la fecha de entrega, minimizar el número de tareas tardías y minimizar la tardanza total de todas las tareas.

En el segundo capítulo se presentan dos formulaciones del modelo de optimización entera mixta correspondiente a los cuatro problemas estudiados. Estas formulaciones se distinguen por el tipo de variables que utilizan. En la primera se utilizan variables de precedencia que indican si una tarea precede o no a otra. En la segunda se utilizan variables de posición que indican si una tarea es realizada o no en una determinada posición de la secuencia o programación. El código desarrollado para la resolución de los distintos modelos con CPLEX Studio se incluye en el apéndice.

En el tercer capítulo se estudian los modelos desde la perspectiva de aprovechar sus propiedades combinatorias para el desarrollo de algoritmos. Se presentan los algoritmos y los teoremas que justifican que proporcionan una solución óptima. En el apéndice correspondiente se presentan los códigos en C++ desarrollados para implementar los algoritmos, así como un ejemplo de su aplicación.

En el cuarto capítulo se presentan los resultados del estudio computacional que se ha realizado para evaluar y comparar las dos aproximaciones al problema presentadas en el segundo y tercer capítulo.

Summary

Scheduling is a decision-making process which consists in the allocation of resources to different tasks over given time periods. This project addresses the problem of scheduling a set of tasks in a set of machines aiming to optimize different objectives.

Scheduling has a key role in the actual society and, as a result, its use has now spread to a wide variety of fields. For example, it can be used to schedule tasks in a runway at an airport, to schedule crews at a construction site or even to schedule functions of the CPU. Over the years, the study of scheduling has been approached in two different ways. On the one hand, the first approach focuses on the study of its properties and the later application of some resolution algorithm based on combinatorics. On the other hand, the second approach seeks to develop mathematical optimization models and use the general techniques proposed to solve them.

In scheduling we have a machine environment in which certain tasks have to be processed. This set of tasks will be denote as N and n will be the number of tasks, $n = |N|$. Generally, subscript i will refer to a machine and j to the task. The following concepts are introduced. Let p_j be the processing time of the task j, d_j the due date of the task j and w_j the weight of the task j. Scheduling problems can be described as a triplet $\alpha|\beta|\gamma$, where α represents the machine environment, β gives characteristics and restrictions of the problem and γ refers to the goal of the problem. Based on it, in the first chapter we give a classification for each of the three environments described in the previous triplet.

In this project we focus on the study of the problem of scheduling in a single machine environment. This model is considered one of the most important models as it can be applied to the resolution of more complex environments. The single machine problem consists of the scheduling of a finite number of tasks, all of them available at the initial moment, which must be processed on a single machine. Depending on the requirements of the system, the goal could vary ranging from minimizing the total lateness of the tasks to avoiding certain task to be late. In particular, we have focussed on the four main single machine problems which are: $1||\sum w_jC_j$ which aims to minimize the total weighted completion time, $1||L_{max}$ which aims to minimize the maximum lateness, $1||\sum U_j$ which aims to minimize the number of tardy jobs and $1||\sum T_j$ which aims to minimize the total tardiness.

In the second chapter of the project, we study the mixed integer optimization models of each of the four main single machine problems. We present two different formulations of each problem, depending on the type of variable used. The first modelling is based on precedence variables. For this purpose, we define the variable y_{jk}, which is a binary variable with value 1 if the task j goes in the schedule before the task k and 0 otherwise. We also define the variable C_j which represents the completion time of task j, the variable $Lmax$ which represents the maximum lateness, the variable U_j which is a binary variable that represents whether the task j is late or not and, finally, the variable T_j which is the total tardiness of task j. The second modelling is based on position variables. We define the binary variable x_{jk} which has value 1 if the task j goes in the position k and 0 otherwise, and the positional date variable γ_k which defines the completion time of the task at position k. For this second modelling we define a new binary variable U_k which has value 1 if the task that is processed in the place k is late and 0 otherwise. Moreover, we also use in this second modelling the variables previously defined C_j, $Lmax$ and T_j. The codes of the two modellings have been implemented on the computer software CPLEX Studio and are available at the corresponding appendix.

In the third chapter, we focus on the modelling based on combinatory properties. For each problem we present a different polynomial or pseudo-polynomial algorithm capable of solving it. We also state

the most important theorems and its proofs, that justify the correctness of the algorithm. The polynomial algorithms are based in the WSPT (Weighted Shortest Processing Time first) rule for the model $1||\sum w_j C_j$ and in the EDD (Earliest Due Date) rule for the model $1||L_{max}$. For the model $1||\sum U_j$ the polynomial algorithm of Hodgson-Moore is presented. Finally, for the model $1||\sum T_j$ a pseudo-polynomial algorithm based on Dynamic Programming is presented. The algorithms have been coded in C++ programming language. Each code can be seen in the corresponding appendix.

In the last chapter we have made a computational study comparing approaches presented in the chapters 2 and 3. We have generated several random datasets with different task sizes aiming to compare the computational time required to solve the problem. We have concluded that, in general, polynomial algorithms based on combinatory properties are more suitable than the corresponding optimization model in terms of the computational time required. Which is the best optimization model depends on the scheduling problem considered. Finally, the optimization model based on positional variables is better than the optimization model based on precedence variables and the algorithm based on Dynamic Programming for the problem in which the total tardiness is minimized.

Índice general

Capítulo 1

Programación de tareas

1.1. Introducción

La programación o secuenciación de tareas, *scheduling* en inglés, consiste en la asignación de recursos a diferentes tareas o trabajos durante periodos de tiempo determinados.

Dada una colección de tareas que requieren ser procesadas en un cierto conjunto de máquinas, el problema considerado consiste en programar para cada tarea uno o más intervalos de tiempo en una o más máquinas. Esta programación puede estar sujeta a ciertas restricciones y trata de optimizar algunos criterios de rendimiento. En consecuencia, el propósito de la programación de tareas es optimizar uno o más objetivos con el fin de conseguir unas prestaciones idóneas y adecuadas a las necesidades del usuario.

La programación de tareas tiene en la actualidad un papel fundamental. Debido a la complejidad que puede llegar a tener esta programación, su uso se ha ido extendiendo a lo largo de diversos campos. Dichas aplicaciones pueden variar desde procesos como la programación de tareas en las pistas de aterrizaje de un aeropuerto, en máquinas de una planta de ensamblaje, en las funciones de una CPU o en la secuenciación de operarios en un taller.

A lo largo del tiempo, el estudio de la programación de tareas que es, en general, computacionalmente complejo [1] ha sido abordado de dos formas distintas tratando de buscar una solución óptima de manera eficiente. La primera línea de investigación se centra en el estudio de las propiedades de cada problema particular para, posteriormente, aplicar algún algoritmo basado en la combinatoria como método de resolución. Por otro lado, la segunda trata de formular modelos de optimización matemática y aplicar las técnicas generales desarrolladas para el modelo. En esta memoria se han estudiado ambas aproximaciones.

1.2. Definiciones y notación

En el problema de la programación de tareas se tiene un sistema en el que se dispone de un entorno de máquinas en el que han de procesarse un conjunto de tareas, que denominamos N. Sea n el número total de tareas, es decir, $n = |N|$. Generalmente, el subíndice j se referirá a una tarea y el subíndice i se referirá a una máquina. Se introducen los siguientes conceptos:

Tiempo de procesado (p_{ij}). Representa el tiempo que tarda la máquina i en realizar la tarea j. Si la tarea j solo ha de procesarse en una máquina se omite el índice i.

Tiempo de llegada al sistema (r_j). Es el momento en que la tarea j llega al sistema para realizarse, es decir, lo más pronto que la tarea j puede comenzar a ser procesada.

Fecha de entrega (d_j). Es la fecha acordada con el cliente para entregar la tarea j. En algunos problemas es posible exceder la fecha de entrega con cierta penalización.

Coste o peso (w_j). Denota la importancia de la tarea j con respecto a las demás tareas que deben realizarse. A menudo, representa un coste por la permanencia de dicha tarea en el sistema.

El objetivo a minimizar será, en general, una función de los tiempos de proceso de las tareas, que dependerá de la secuencia en la que son procesadas. En otros casos, el objetivo será una función de las fechas de vencimiento. Por ello, se introducen los siguientes conceptos:

Tiempo de finalización (C_j). Es el tiempo en el que la tarea j sale del sistema porque ha sido completada. Es decir, es el tiempo de finalización en la última máquina donde requiere procesamiento.

Retraso (L_j). Se define el retraso de la tarea j como $L_j = C_j - d_j$. Esta función será positiva siempre que la tarea j se complete tarde con respecto a su fecha de entrega y será negativa cuando la tarea j sea completada con antelación.

Tardanza (T_j). Se define la tardanza de una tarea j como

$$T_j = \max\{C_j - d_j, 0\} = \max\{L_j, 0\}.$$

La tardanza representa las unidades de tiempo que la tarea tarda en completarse si se termina más tarde que su fecha de entrega.

Adelanto (E_j). El adelanto de la tarea j se define como

$$E_j = \max\{d_j - C_j, 0\} = \max\{-L_j, 0\}.$$

Representa las unidades de tiempo que la tarea j se completa antes de su fecha de entrega, si lo hace.

Penalización unitaria (U_j). Indica si la tarea j se ha completado a tiempo y se define como:

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

1.3. Clasificación de los problemas

Los problemas de programación de tareas se describen mediante una terna $\alpha \mid \beta \mid \gamma$. En esta terna, α representa el entorno de máquina, β proporciona las características y restricciones del problema y, finalmente, γ hace referencia al objetivo a minimizar o maximizar en el problema.

1.3.1. Campo α: entorno de máquina

La siguiente clasificación muestra los tipos de entorno de máquina más usuales en los problemas de programación de tareas, así como el símbolo que los identifica en el campo α:

Máquina única (1). Las tareas han de procesarse en una única máquina.

Máquinas idénticas en paralelo (Pm). En este entorno hay m máquinas idénticas y la tarea j requiere una única operación que puede ser realizada en cualquiera de las m máquinas. Si la tarea j solo puede procesarse en cualesquiera de las máquinas que pertenecen a un subconjunto dado M_j, entonces se escribe M_j en el campo α.

Máquinas en paralelo con diferentes velocidades (Qm). Hay m máquinas en paralelo pero estas tienen distintas velocidades que no dependen de la tarea.

Máquinas en paralelo no relacionadas (Rm). En este entorno hay m máquinas en paralelo, pero cada máquina i puede procesar la tarea j a una velocidad diferente.

General Shop (Gm). En este entorno hay n tareas diferentes y cada una de ellas tiene una o varias subtareas. Cada tarea no puede realizarse en más de una máquina simultáneamente y cada máquina no puede procesar más de una tarea a la vez.

Open Shop (Om). Este entorno es un caso particular del General Shop. En él se tienen m máquinas y n tareas compuestas de tantas subtareas como máquinas hay. Así, cada subtarea $i = 1, ..., m$ debe ser procesada en la máquina i.

Flow Shop (Fm). En este entorno hay m máquinas en serie. Cada tarea j consiste en tantas subtareas como máquinas hay y cada subtarea $i = 1, ..., m$ debe ser procesada en la máquina i.

Job Shop (Jm). En este entorno hay m máquinas y en él, cada tarea tiene definida la secuencia que debe seguir. Así, dentro de una misma tarea, cada subtarea j debe ser procesada antes que la subtarea $j+1$.

1.3.2. Campo β: restricciones

En este apartado se describen algunas de las restricciones más comunes existentes en los problemas de programación de tareas. Si el campo β está vacío, se considera que todas las tareas y máquinas del sistema están disponibles desde el comienzo del horizonte de planificación.

Fecha de inicio (r_j). Si este símbolo aparece en el campo β, entonces la tarea j no podrá empezar a ser procesada antes de su fecha de inicio r_j. Si r_j no aparece como restricción, entonces la tarea puede comenzar en cualquier momento. A diferencia de las fechas de inicio, las fechas de entrega no son una restricción que se especifica en el campo β.

Interrupciones ($prmp$). Las interrupciones implican que no es necesario mantener una tarea en una máquina una vez empezado su proceso hasta su finalización. Así, es posible interrumpir el procesamiento de una tarea y sustituir dicha tarea por otra. En este caso, cuando se vuelve a reanudar la tarea interrumpida en dicha máquina solo es necesario procesar la tarea con su tiempo restante de procesamiento. Siempre que el problema no lo indique explícitamente en este campo, las interrupciones no están permitidas.

Restricciones de precedencia ($prec$). Estas restricciones consisten en la imposición de que una o varias tareas se completen antes de que otra tarea pueda empezar a procesarse. Si cada tarea tiene a lo sumo un predecesor y un sucesor, estas restricciones se denominan *cadenas*.

Tiempo de preparación dependiente de la secuencia (s_{jk}). Esta restricción representa el tiempo de preparación necesario entre el procesamiento de las tareas j y k. En particular, denota el tiempo que es preciso para que la tarea k pueda procesarse. Así, s_{0k} será el tiempo de preparación para la tarea k si esta es la primera de la secuencia. Por su parte, s_{j0} hace referencia al tiempo de limpieza después de realizar la tarea j si dicha tarea es la última en la secuencia. Si, además, el tiempo de preparación necesario entre ambas tareas depende de la máquina lo denotaremos por s_{ijk}.

Procesamiento por lotes ($batch(b)$). Una máquina es capaz de procesar un número de tareas b simultáneamente, es decir, puede procesar un lote de hasta b tareas al mismo tiempo. Durante el procesamiento de un lote, el tiempo de procesado de las tareas puede no ser el mismo, pero el lote entero está completado cuando se han completado todas sus tareas. Por tanto, el tiempo de finalización del lote coincide con el tiempo de finalización de la tarea con el mayor tiempo de proceso.

Restricciones de elegibilidad de la máquina (M_j). Como ya se ha indicado anteriormente, el símbolo M_j aparece en el campo β cuando se dispone de m máquinas en paralelo pero no todas las máquinas son capaces de procesar la tarea j. El símbolo M_j denota el conjunto de máquinas capaces de procesar la tarea j.

1.3.3. Campo γ: funciones objetivo

El objetivo que se propone minimizar para abordar el problema condiciona el procedimiento que se aplica para su resolución. A continuación, se relacionan las funciones objetivo más usuales en programación de tareas:

Máximo retraso (L_{max}). Se define como $\max\{L_1,...,L_n\}$ y mide la peor desviación de la fecha de entrega.

Tiempo de finalización total ponderado ($\sum w_jC_j$). Dependiendo del significado del peso w_j en el problema, puede dar una idea, por ejemplo, del coste total de la programación de las tareas.

Número de tareas con retraso ($\sum U_j$). Identifica el total de tareas no realizadas antes de la fecha de entrega.

Tardanza total ($\sum T_j$). Mide la tardanza total de las tareas.

1.4. Programación de tareas con una única máquina

En esta memoria, nos centraremos en el estudio de la programación de tareas en una única máquina. Los modelos considerados, además de ser importantes en sí mismos, lo son porque, en muchos casos, se aplican en la resolución de problemas más complejos.

La programación de tareas en una única máquina consiste en establecer la secuencia en la que debe realizarse un número finito de tareas que han de procesarse en la única máquina del sistema. Suponemos que cada tarea j está disponible desde el instante inicial del proceso. Como la máquina solo puede procesar una tarea cada vez, el resto de las tareas permanecerán en cola hasta que sean procesadas y abandonen el sistema una vez hayan sido completadas. El objetivo considerado podrá ser, dependiendo de las necesidades del sistema, minimizar la penalización por el retraso en algunas tareas o garantizar que ciertas tareas no excedan su fecha de entrega.

Notemos que cuando solo hay una máquina, si la función objetivo es una función monótona del tiempo de finalización de las tareas entonces solo hay que tener en cuenta aquellas programaciones que no consideran ni interrupciones ni tiempos de inactividad de la máquina. En efecto, consideremos que la tarea j interrumpe su procesado. Es decir, supongamos que j es programada en $[t_1,t_2]$ y $[t_3,t_4]$ donde $t_1 < t_2 < t_3 < t_4$.

Si se reorganiza la programación de manera que la parte de la tarea j que se procesa en $[t_1,t_2]$ es programada entre $t_3-(t_2-t_1)$ y t_3 y por tanto, todo lo programado entre t_2 y t_3 se adelanta t_2-t_1 unidades de tiempo, entonces se elimina la interrupción de la tarea j sin incrementar el valor de la función objetivo y sin crear otras nuevas interrupciones. Así, continuando el proceso se obtendría una solución óptima del problema en la que no son necesarias las interrupciones.

En esta memoria se estudian los modelos:

Modelo $1||\sum w_jC_j$, en el que el objetivo es minimizar el tiempo de finalización total ponderado.

Modelo $1||L_{max}$, en el que se minimiza la peor desviación de la fecha de entrega.

Modelo $1||U_j$, en el que se minimiza el número de tareas tardías.

Modelo $1||\sum T_j$, en el que se minimiza la tardanza total.

Las buenas propiedades de estos modelos, en comparación con el resto de modelos de programación de tareas, han permitido desarrollar algoritmos polinomiales y pseudopolinomiales. Es posible resolver tres de los modelos de una única máquina en tiempo polinomial. El estudio de los capítulos 1 y 3 se ha realizado a partir de los textos de Pinedo [2], Brucker [1] y Sule [3]. El capítulo 2 se ha trabajado a partir del artículo de Keha et al. [5] y del artículo de Baker y Keler [6].

Capítulo 2

Modelos de optimización entera

En este capítulo el objetivo es presentar modelos de optimización entera para cada uno de los problemas de programación de tareas con una única máquina considerados, que pueden ser resueltos con técnicas estándar desarrolladas para estos modelos. En particular, en esta memoria se utilizará el software de propósito general CPLEX Studio.

2.1. Modelización basada en variables que indican precedencia

Se define la variable binaria y_{jk} con $j,k \in N, j < k$:

$$y_{jk} = \begin{cases} 1, & \text{si la tarea } j \text{ se realiza antes que la } k \\ 0, & \text{en caso contrario} \end{cases}$$

Se definen también las variables:

C_j: variable no negativa que representa el tiempo de finalización de la tarea j, $j \in N$.

$Lmax$: variable no restringida que representa la peor desviación de la fecha de entrega.

U_j: variable binaria que toma el valor 1 si la tarea finaliza después de su fecha de entrega y cero en caso contrario, $j \in N$.

T_j: variable no negativa que representa la tardanza total de la tarea j, $j \in N$.

En las siguientes subsecciones se van a formular los modelos de optimización basados en estas variables correspondientes a los cuatro problemas de programación de tareas indicados en el apartado 1.4

2.1.1. Modelo $1||\sum w_j C_j$

El modelo de optimización entera mixta se formula como:

$$\min \quad \sum_{j=1}^{n} w_j C_j \tag{2.1a}$$

sujeto a

$$C_j \geq p_j \qquad j \in N \tag{2.1b}$$
$$C_j + p_k \leq C_k + M(1 - y_{jk}) \qquad j,k \in N,\ \ j < k \tag{2.1c}$$
$$C_k + p_j \leq C_j + M y_{jk} \qquad j,k \in N,\ \ j < k \tag{2.1d}$$
$$C_j \geq 0 \qquad j \in N \tag{2.1e}$$
$$y_{jk} \in \{0,1\} \qquad j,k \in N,\ \ j < k \tag{2.1f}$$

siendo M una constante suficientemente grande. En general, el valor de M es suficientemente grande si se toma como la suma de los tiempos de procesamiento de todas las tareas. La función objetivo (2.1a) minimiza el tiempo de finalización total ponderado. Las restricciones (2.1b) aseguran que el tiempo de finalización de cada tarea es mayor o igual que su tiempo de procesamiento. Las restricciones (2.1c) y (2.1d) son disyuntivas y su función es asegurar que, o bien la tarea j es procesada antes que la k, o bien la k es procesada antes que la j. Es decir, o bien $C_j + p_k \leq C_k$, o bien $C_k + p_j \leq C_j$. Finalmente, las restricciones, (2.1e) y (2.1f), garantizan que las variables C_j son no negativas y las variables y_{jk} son binarias, respectivamente.

2.1.2. Modelo $1||L_{max}$

El modelo de optimización entera mixta se formula como:

$$\min \quad \max_{j=1,\ldots,n} \{C_j - d_j\}$$

sujeto a

(2.1b) - (2.1f)

La función objetivo, que no es lineal, minimiza la peor desviación de la fecha de entrega. Para linealizarla, basta considerar la variable *Lmax* y formular el modelo como:

$$\min \quad Lmax \tag{2.3a}$$

sujeto a

$$\text{(2.1b) - (2.1f)} \tag{2.3b}$$

$$Lmax \geq C_j - d_j \qquad j \in N \tag{2.3c}$$

La función objetivo minimiza la variable *Lmax* que, teniendo en cuenta las restricciones (2.3c), representa la peor desviación de la fecha de entrega. Las restricciones (2.3b) han sido explicadas en la sección 2.1.1. Las restricciones (2.3c) garantizan que la variable *Lmax* es mayor o igual que las desviaciones de todas las tareas de sus fechas de entrega.

2.1.3. Modelo $1||\sum U_j$

El modelo de optimización entera mixta se formula como:

$$\min \quad \sum_{j=1}^{n} U_j \tag{2.4a}$$

sujeto a

$$\text{(2.1b) - (2.1f)} \tag{2.4b}$$

$$C_j \leq d_j + MU_j \qquad j \in N \tag{2.4c}$$

$$U_j \in \{0,1\} \qquad j \in N \tag{2.4d}$$

La función objetivo minimiza el número de tareas que no satisfacen su fecha de entrega. Las restricciones (2.4b) han sido ya definidas en la sección 2.1.1. Las restricciones (2.4c), en las que M es una constante suficientemente grande, garantizan que si hay retraso en la finalización de la tarea, es decir, $C_j > d_j$, entonces $U_j = 1$. En general, el valor de M se toma como la suma de los tiempos de procesamiento de todas las tareas. Por otro lado, si $C_j \leq d_j$, las restricciones (2.4c) no exigen nada sobre el valor de U_j pero el hecho de que la función objetivo minimice garantiza que $U_j = 0$. Las restricciones (2.4d) aseguran que las variables U_j son binarias.

2.1.4. Modelo $1||\sum T_j$

El modelo de optimización entera mixta se formula como:

$$\min \quad \sum_{j=1}^{n} T_j \tag{2.5a}$$

sujeto a

$$\text{(2.1b) - (2.1f)} \tag{2.5b}$$

$$T_j \geq C_j - d_j \qquad j \in N \tag{2.5c}$$

$$T_j \geq 0 \qquad j \in N \tag{2.5d}$$

La función objetivo minimiza la tardanza total. Las restricciones dadas en (2.5b) han sido explicadas en la sección 2.1.1. Las restricciones (2.5c) y (2.5d) garantizan que T_j mide la tardanza de la tarea j. Aunque, nada impediría en las restricciones que T_j tomase un valor mayor que $C_j - d_j$ si $C_j > d_j$, el hecho de que la función objetivo se minimice asegura que, en este caso, $T_j = C_j - d_j$. Por otro lado, si $C_j < d_j$, obviamente, $T_j = 0$.

2.2. Modelización basada en variables de posición

Se define la variable binaria de posición, x_{jk} con $j,k \in N$:

$$x_{jk} = \begin{cases} 1, & \text{si la tarea } j \text{ se realiza exactamente en la posición } k \\ 0, & \text{en caso contrario} \end{cases}$$

Además, se define la variable de fecha posicional, γ_k, que representa el tiempo de finalización de la tarea que se realiza en la posición k, $k = 1,...,n$. Se utilizarán también las variables C_j, *Lmax* y T_j ya introducidas en la sección 2.1. En las siguientes subsecciones se van a formular los modelos de optimización basados en estas variables.

2.2.1. Modelo $1||\sum w_j C_j$

El modelo de optimización entera mixta se formula como:

$$\min \quad \sum_{j=1}^{n} w_j C_j \tag{2.6a}$$

sujeto a

$$\sum_{k=1}^{n} x_{jk} = 1 \qquad j \in N \tag{2.6b}$$

$$\sum_{j=1}^{n} x_{jk} = 1 \qquad k \in N \tag{2.6c}$$

$$\gamma_1 \geq \sum_{j=1}^{n} p_j x_{j1} \tag{2.6d}$$

$$\gamma_k \geq \gamma_{k-1} + \sum_{j=1}^{n} p_j x_{jk} \qquad k = 2,...,N \tag{2.6e}$$

$$C_j \geq \gamma_k - M(1 - x_{jk}) \qquad j,k \in N \tag{2.6f}$$

$$\gamma_k \geq 0 \qquad k \in N \tag{2.6g}$$

$$x_{jk} \in \{0,1\} \qquad j,k \in N \tag{2.6h}$$

$$C_j \geq 0 \qquad j \in N \tag{2.6i}$$

siendo M una constante suficientemente grande. En general, el valor de M es suficientemente grande si se toma como la suma de los tiempos de procesamiento de todas las tareas. La función objetivo (2.6a) minimiza el tiempo de finalización total ponderado. Las restricciones (2.6b) y (2.6c) aseguran que cada tarea es asignada a una única posición y que cada posición se asigna a una única tarea. Las restricciones (2.6d) y (2.6e) aseguran el tiempo de finalización de la tarea en la posición k. Las restricciones (2.6f) garantizan que el tiempo de finalización de la tarea j es no anterior al momento en que termina. En efecto, si $x_{jk} = 1$, es decir, se realiza en la posición k entonces $C_j \geq \gamma_k$. Cuando $x_{jk} = 0$ esa restricción no restringe el valor de C_j. Aunque estas restricciones solo garantizan que $C_j \geq \gamma_k$, el hecho de que la función objetivo minimice asegura que $C_j = \gamma_k$ si la tarea j se realiza en la posición k. Las restricciones, (2.6g) y (2.6h), garantizan que las variables γ_k y las variables x_{jk} son no negativas y binarias, respectivamente. Por último, las restricciones (2.6i) aseguran la no negatividad de las variables C_j, $j \in N$.

2.2.2. Modelo $1||L_{max}$

Con el mismo argumento de linealización de la primera función objetivo explicada en la sección 2.1.2, el modelo de optimización entera mixta se formula como:

$$\min \quad Lmax \tag{2.7a}$$

sujeto a

$$\text{(2.6b) - (2.6e), (2.6g) - (2.6h)} \tag{2.7b}$$

$$Lmax \geq \gamma_k - \sum_{j=1}^{n} d_j x_{jk} \qquad k \in N \tag{2.7c}$$

La función objetivo minimiza la peor desviación de las fechas de entrega. Las restricciones (2.7b) han sido explicadas en la sección 2.2.1. Las restricciones (2.7c) garantizan que la variable $Lmax$ es mayor o igual que las desviaciones de todas las tareas de sus fechas de entrega, ya que $\sum_{j=1}^{n} d_j x_{jk}$ proporciona la fecha de entrega de la tarea que se realiza en la posición k.

2.2.3. Modelo $1||\sum U_j$

Se introduce la variable $\tilde{U}_k$ que vale 1 si la tarea que se realiza en la posición k se realiza después de su fecha de entrega y 0 en caso contrario.

El modelo de optimización entera mixta se formula como:

$$\min \quad \sum_{k=1}^{n} \tilde{U}_k \tag{2.8a}$$

sujeto a

$$\text{(2.6b) - (2.6e), (2.6g) - (2.6h)} \tag{2.8b}$$

$$\gamma_k \leq \sum_{j=1}^{n} d_j x_{jk} + M\tilde{U}_k \qquad k \in N \tag{2.8c}$$

$$\tilde{U}_k \in \{0,1\} \qquad k \in N \tag{2.8d}$$

La función objetivo minimiza el número de tareas que no satisfacen su fecha de entrega. Las restricciones (2.8b) han sido ya definidas en la sección 2.2.1. Las restricciones (2.8c), en las que M es una constante suficientemente grande que se toma como en 2.1.1, garantizan que si $\gamma_k > \sum_{j=1}^{n} d_j x_{jk}$, es decir, hay retraso en la finalización de la tarea que se realiza en la posición k entonces $\tilde{U}_k = 1$. Por otro lado, si $\gamma_k \leq \sum_{j=1}^{n} d_j x_{jk}$, las restricciones (2.8c) no exigen nada sobre el valor de $\tilde{U}_k$ pero la función objetivo garantiza que $\tilde{U}_k = 0$. Las restricciones (2.8d) aseguran que la variable $\tilde{U}_k$ sea binaria.

2.2.4. Modelo $1||\sum T_j$

El modelo de optimización entera mixta se formula como:

$$\min \quad \sum_{j=1}^{n} T_j \tag{2.9a}$$

sujeto a

$$\text{(2.6b) - (2.6i)} \tag{2.9b}$$

$$T_j \geq C_j - d_j \qquad j \in N \tag{2.9c}$$

$$T_j \geq 0 \qquad j \in N \tag{2.9d}$$

La función objetivo minimiza la tardanza total. Las restricciones (2.9b) han sido explicadas en la sección 2.2.1. Las restricciones (2.9c) y (2.9d) garantizan que T_j mide la tardanza de la tarea j. Como ya se ha indicado anteriormente, aunque nada impediría en las restricciones que T_j tomase un valor mayor que $C_j - d_j$, si $C_j > d_j$, el hecho de que la función objetivo minimice asegura que, en este caso, $T_j = C_j - d_j$. Por otro lado, si $C_j < d_j$, obviamente, $T_j = 0$.

Capítulo 3

Modelización basada en propiedades combinatorias

En este capítulo se estudian las propiedades combinatorias de los cuatro problemas de programación de tareas en una única máquina estudiados en este trabajo que han permitido desarrollar algoritmos eficientes para la resolución del problema.

3.1. Modelo $1||\sum w_jC_j$.

Este modelo en el que se trata de minimizar el tiempo de finalización total ponderado puede resolverse de manera sencilla aplicando la regla 'el tiempo ponderado de procesamiento más corto, primero' (en inglés *Weighted Shortest Processing Time first, WSPT rule*). Es decir, las tareas han de ordenarse en orden decreciente de w_j/p_j.

Teorema 3.1. *La regla WSPT es óptima para el modelo* $1||\sum w_jC_j$.

Demostración. Probaremos que la regla es óptima por contradicción.

Suponemos en primer lugar una secuenciación o programación S de n tareas que es óptima pero no cumple la regla WSPT. Si las tareas aún no han sido programadas en S en orden decreciente del cociente w_j/p_j, entonces debe haber al menos dos tareas adyacentes, j seguida por k, tales que

$$\frac{w_j}{p_j} < \frac{w_k}{p_k} \tag{3.1}$$

Supongamos que la tarea j empieza a procesarse en el tiempo t e intercambiamos las posiciones de las tareas j y k, dejando sin modificar la posición del resto de las tareas. Una vez realizado el intercambio, llamamos a la nueva programación S'. El tiempo de finalización total ponderado de las tareas que se van a realizar antes de procesar las tareas j y k será el mismo en ambas programaciones. De igual manera, tampoco se verá afectado el tiempo ponderado de finalización de las tareas que se realizan después de procesar ambas tareas. La única diferencia en la función objetivo entre ambas programaciones será la producida por las tareas j y k, como puede verse en la figura 3.1.

Bajo la programación S, el tiempo de finalización total ponderado de las tareas j y k es

$$w_jC_j + w_kC_k = (t+p_j)w_j + (t+p_j+p_k)w_k,$$

mientras que en la nueva programación S' será:

$$w_kC'_k + w_jC'_j = (t+p_k)w_k + (t+p_k+p_j)w_j,$$

Aplicando la desigualdad (3.1):

$$w_jC_j + w_kC_k - w_kC'_k - w_jC'_j = p_jw_k - p_kw_j > 0$$

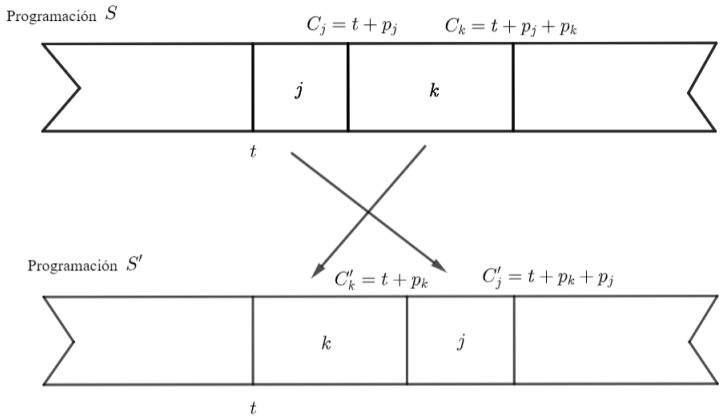

Figura 3.1: Intercambio entre las tareas j y k

Por tanto, la suma de los tiempos de finalización ponderados de S' es estrictamente menor que la suma de los tiempos de finalización ponderados de la programación S. Esto contradice que S sea óptima. Por tanto, S ha de cumplir la regla WSPT y por ello, la regla WSPT es óptima para el modelo $1||\sum w_j C_j$. □

En el Algoritmo 1 se describe el procedimiento que permite resolver el problema. Este algoritmo aplica directamente la regla WSPT. En el apéndice A.1. se incluye un ejemplo de aplicación del algoritmo.

Algoritmo 1: Regla WSPT

1 Datos: $N, p_j, w_j, j \in N$;
2 **para** $j = 1$ **a** n **hacer**
3 | Obtener ratio $S_j := \frac{w_j}{p_j}$;
4 **fin**
5 Ordenar las tareas en orden descendente de S_j;
6 **devolver** *Programación ordenada de tareas;*

Como se indicó en el capítulo 1, entre las restricciones de precedencia que pueden introducirse están las denominadas cadenas. En ellas, cada tarea con restricción de precedencia va precedida por una tarea específica y debe ser seguida por otra tarea específica. A continuación se describe un modelo particular del modelo general $1||\sum w_j C_j$, para el que también se ha desarrollado un algoritmo polinomial basado en las relaciones entre los pesos y los tiempos de procesamiento.

Modelo $1|prec\ chain|\sum w_j C_j$

El siguiente teorema indica, dadas dos cadenas de tareas, la cadena de tareas que debería procesarse en primer lugar. Por simplificar la notación, supondremos que las dos cadenas son $\text{Cad}_1 = \{1,...,k\}$ y $\text{Cad}_2 = \{k+1,...,n\}$ en la que las tareas han de procesarse en orden creciente de índices.

Teorema 3.2. *Si*

$$\frac{\sum_{j=1}^{k} w_j}{\sum_{j=1}^{k} p_j} > \frac{\sum_{j=k+1}^{n} w_j}{\sum_{j=k+1}^{n} p_j}, \tag{3.2}$$

el tiempo de finalización total ponderado de procesar Cad_1 seguida de Cad_2 es menor que el tiempo de finalización total ponderado si se procesan en el orden inverso.

Demostración. Si se programa la Cad_1 seguida de Cad_2, el tiempo de finalización ponderado es

$$w_1 p_1 + ... + w_k \sum_{j=1}^{k} p_j + w_{k+1} \sum_{j=1}^{k+1} p_j + ... + w_n \sum_{j=1}^{n} p_j,$$

mientras que si se programan en orden inverso, el tiempo de finalización total ponderado es

$$w_{k+1} p_{k+1} + ... + w_n \sum_{j=k+1}^{n} p_j + w_1 \left(\sum_{j=k+1}^{n} p_j + p_1 \right) + ... + w_k \sum_{j=1}^{n} p_j.$$

Simplificando los términos comunes y aplicando la desigualdad 3.2 se obtiene la conclusión del teorema. □

Dada la cadena $\text{Cad}_1 = \{1, ..., k\}$ se define el ρ-factor de la cadena como

$$\rho(1, ..., k) = \frac{\sum_{j=1}^{l^*} w_j}{\sum_{j=1}^{l^*} p_j} \tag{3.3}$$

siendo l^* la tarea para la que se cumple que

$$\frac{\sum_{j=1}^{l^*} w_j}{\sum_{j=1}^{l^*} p_j} = \max_{1 \leq l \leq k} \left\{ \frac{\sum_{j=1}^{l} w_j}{\sum_{j=1}^{l} p_j} \right\}$$

Basándose en este concepto, el siguiente teorema indica cuántas de las tareas de una cadena han de procesarse sin interrupción por tareas de otras cadenas.

Teorema 3.3. *Sea la cadena $Cad_1 = \{1, ..., k\}$ y la tarea l^* identificada por su ρ-factor. Entonces existe una programación óptima que procesa las tareas $1, ..., l^*$ una detrás de otra sin interrupción alguna por tareas de otras cadenas.*

Demostración. Lo probaremos por contradicción. Supongamos que existe una programación óptima en la que la secuencia de tareas $1, ..., l^*$ es interrumpida por una tarea, v, de otra cadena. Es decir, la programación óptima contiene la secuencia, que llamamos S, $1, ..., u, v, u+1, ..., l^*$. Veamos que, o bien la secuencia $S' : v, 1, ..., l^*$, o bien la secuencia $S'' : 1, ..., l^*, v$, proporciona un tiempo de finalización total ponderado menor manteniendo el resto de las tareas sin modificación con S. Si no es menor con S', entonces tendrá que serlo con S''.

Por el teorema 3.2, si el tiempo de finalización total ponderado con S es menor que con S' entonces

$$\frac{w_v}{p_v} < \frac{w_1 + w_2 + ... + w_u}{p_1 + p_2 + ... + p_u}$$

Análogamente, si el tiempo de finalización total ponderado con S es menor que con S'' entonces

$$\frac{w_v}{p_v} > \frac{w_{u+1} + w_{u+2} + ... + w_{l^*}}{p_{u+1} + p_{u+2} + ... + p_{l^*}}$$

Como la tarea l^* es aquella que determina el ρ-factor de la cadena Cad_1, entonces

$$\frac{\sum_{j=1}^{l^*} w_j}{\sum_{j=1}^{l^*} p_j} \geq \frac{\sum_{j=1}^{u} w_j}{\sum_{j=1}^{u} p_j}$$

Desarrollando la expresión se obtiene:

$$\sum_{j=1}^{l^*} w_j \sum_{j=1}^{u} p_j \geq \sum_{j=1}^{u} w_j \sum_{j=1}^{l^*} p_j \Rightarrow \sum_{j=1}^{u} w_j \sum_{j=1}^{u} p_j + \sum_{j=u+1}^{l^*} w_j \sum_{j=1}^{u} p_j \geq \sum_{j=1}^{u} w_j \sum_{j=1}^{u} p_j + \sum_{j=1}^{u} w_j \sum_{j=u+1}^{l^*} p_j$$

Y por lo tanto

$$\frac{w_{u+1} + w_{u+2} + ... + w_{l^*}}{p_{u+1} + p_{u+2} + ... + p_{l^*}} \geq \frac{w_1 + w_2 + ... + w_u}{p_1 + p_2 + ... + p_u}$$

Así, si la secuencia S es mejor que S'' se tiene que

$$\frac{w_v}{p_v} > \frac{w_{u+1} + w_{u+2} + ... + w_{l^*}}{p_{u+1} + p_{u+2} + ... + p_{l^*}} \geq \frac{w_1 + w_2 + ... + w_u}{p_1 + p_2 + ... + p_u}$$

Por lo que S' es mejor que S. Así llegamos a una contradicción. El otro caso es igual. Por tanto, bajo una programación óptima la secuencia de tareas $1, ..., l^*$ no es interrumpida por ninguna tarea de otra cadena. Si se interrumpe la cadena por más de una tarea el razonamiento sería análogo. □

Basado en este teorema se ha desarrollado un algoritmo polinomial que indica el orden en el que se realizan las secciones de cada cadena. Así, dadas las cadenas $R_1, ..., R_h$ con conjunto de tareas ordenadas $N_1 = \{i_1^1, ..., i_{k_1}^1\}, ..., N_h = \{i_1^h, ..., i_{k_h}^h\}$, el siguiente procedimiento permite resolver el problema. En primer lugar, se obtiene la tarea l^* que determina el ρ-factor y se calcula el ρ-factor de cada cadena según la definición (3.3). Una vez calculados todos los ρ-factores, se selecciona la cadena R_k que tiene un mayor ρ-factor. Se añade el conjunto de tareas $i_1, ..., i_{l^*}^k$ de dicha cadena a la programación ordenada y se elimina $i_1, ..., i_{l^*}^k$ del conjunto de tareas a programar de la cadena.

Una vez eliminado el conjunto de tareas a programar de la cadena R_k, con las tareas restantes en R_k, se obtiene la nueva tarea l^* que determina el nuevo ρ-factor y se vuelve a repetir el proceso a partir de la comparación de ρ-factores descrito antes. Si, por el contrario, en la cadena R_k no quedan más tareas por programar después de haber quitado el conjunto de tareas $i_1, ..., i_{l^*}^k$ de ella, de entre las demás cadenas con tareas por programar se selecciona la cadena que tiene un mayor ρ-factor y se repite el proceso anterior hasta llegar a la programación ordenada de tareas. En el apéndice A.1. se incluye un ejemplo de aplicación de este algoritmo.

Si suponemos que en el problema hay una única cadena, por ejemplo, $Cad = \{1, ..., r\}$, en el modelo de optimización presentado en la sección 2.2.1, para garantizar la relación de precedencia de esta cadena habrían de añadirse las restricciones:

$$x_{jk} \leq \sum_{h=1}^{k-1} x_{j-1,h} \qquad j, h \in Cad$$

Estas restricciones garantizan que dos tareas consecutivas de una misma cadena tengan el orden necesario de precedencia. En efecto, si la tarea j ocupa el lugar k entonces la tarea $j-1$ ha de ocupar un lugar anterior a k. Así, si $x_{jk} = 1$ se garantiza que la tarea $j-1$ ha sido programada con anterioridad. Si, por otra parte, $x_{jk} = 0$, entonces no restringen nada.

3.2. Modelo $1||L_{max}$

En esta sección consideramos en primer lugar un modelo general en el que se minimiza la función objetivo h_{max} definida como $h_{max} = \max\{h_1(C_1), ..., h_n(C_n)\}$ siendo h_j funciones de coste no decrecientes de los tiempos de finalización de las tareas y en el que pueden existir relaciones de precedencia entre las tareas. Este modelo de minimización incluye el modelo $1||L_{max}$ como caso particular.

Para el modelo $1|prec|h_{max}$ se ha desarrollado un algoritmo polinomial basado en la regla 'Las últimas las de menor coste' (conocida en inglés como *"Lowest Cost Last"*, LCL). Este algoritmo es de tipo 'backward', es decir, la ordenación correcta de las tareas será la secuencia inversa a la proporcionada

por el algoritmo, de manera que la última tarea de la programación será la que se obtiene en primer lugar según el algoritmo.

En el problema, $C_{max} = \sum_{j=1}^{n} p_j$, es independiente de la programación. Sea J el conjunto de tareas ya programadas, que se procesan en el intervalo $[C_{max} - \sum_{j\in J} p_j, C_{max}]$. Sea J^c el complementario de J. Sea J' el conjunto de tareas que pueden ser programadas inmediatamente antes de las tareas de J, es decir, el conjunto de tareas cuyos sucesores están en J. El conjunto J' se denomina conjunto de tareas programables.

De acuerdo con la regla LCL se calcula, para $j \in J'$ el valor $h_j\left(\sum_{k\in J^c} p_k\right)$ y se selecciona la tarea j^* tal que

$$h_{j^*}\left(\sum_{k\in J^c} p_k\right) = \min_{j\in J'}\left\{h_{j^*}\left(\sum_{k\in J^c} p_k\right)\right\}$$

Una vez obtenida j^*, se añade al conjunto J y se elimina de J^c. Se modifica, a continuación, el conjunto J' de las tareas listas para ser programadas. Si una vez añadida j^* se tiene que $J^c = \emptyset$, ya se tiene la programación ordenada de tareas. Si, por el contrario, aún quedan tareas por programar en J^c, se recalculan las funciones de coste, repitiendo el proceso anterior hasta obtener la programación ordenada de tareas. En el apéndice A.2. se incluye un ejemplo de aplicación del algoritmo.

Teorema 3.4. *La regla LCL proporciona una programación óptima para el modelo* $1|prec|h_{max}$.

Demostración. Probaremos que la regla proporciona una programación óptima por contradicción.

Supongamos que en una iteración dada, la tarea j^{**} del conjunto J', no tiene el mínimo coste de realización $h_{j^*}\left(\sum_{k\in J^c} p_k\right)$ entre las tareas incluidas en J'. En ese caso, la tarea con el mínimo coste, j^*, deberá ser secuenciada en una iteración posterior y ha de aparecer en la programación antes que la tarea j^{**}. Es posible que otras tareas sean realizadas entre las tareas j^* y j^{**}.

Para ver que esta programación de las tareas no es óptima, tomamos la tarea j^* y la programamos inmediatamente después de la tarea j^{**}. Todas las tareas en la secuencia original que se realizan entre las tareas j^* y j^{**}, incluyendo a j^{**}, se completarían antes. Por tanto, la única tarea cuyo coste de finalización se incrementa será la j^*. Sin embargo, su coste de finalización actual es, por definición, menor que el coste de finalización de la tarea j^{**} en la secuencia original. Por tanto, el coste máximo de finalización disminuye después de la reorganización de la tarea j^* y con ello, se llega a una contradicción. □

El modelo $1||L_{max}$ estudiado en esta memoria es un caso particular del modelo $1|prec|h_{max}$, en el que $h_j = C_j - d_j$ y no existen precedencias entre las cadenas. En este modelo, la aplicación del algoritmo LCL permite ordenar las tareas en orden creciente de sus fechas de entrega. Esta regla es conocida como 'Fecha de entrega más temprana, primero' (conocida en inglés como *"Earliest Due Date"*, EDD). En ella, en caso de empate entre fechas de entrega, las tareas pueden ser seleccionadas arbitrariamente por el usuario. En el Algoritmo 2 se describe el algoritmo EDD.

Algoritmo 2: Algoritmo EDD.

1 Datos: $J^c = \{1,...,n\}$, $J = \emptyset$, d_j, $j \in N$;
2 **para** $j \in J^c$ **hacer**
3 | Seleccionar j^* la tarea con menor d_{j^*};
4 **fin**
5 Añadir j^* al conjunto J y eliminarlo de J^c;
6 **si** $J^c \neq \emptyset$ **entonces**
7 | Parar;
8 **en otro caso**
9 | Ir a la línea 2;
10 **fin**
11 **devolver** *Programación ordenada de tareas;*

3.3. Modelo $1||\sum U_j$

En esta sección el objetivo es minimizar el número de tareas que no han sido completadas antes de su fecha de entrega. En una programación óptima para este modelo, se realizan, en primer lugar, un conjunto de tareas que cumplirán sus fechas de entrega. A continuación, se realizan las tareas que no cumplen sus fechas de entrega cuyo orden de realización no es relevante. El primer conjunto de tareas tendrá que ser programado según el algoritmo EDD explicado en la sección 3.2. Es decir, aquellas con una fecha de entrega más temprana, irán primero.

Sea J el conjunto de las tareas que en una programación óptima son completadas antes de su fecha de entrega y sea J^d el conjunto de las tareas que no satisfacen su fecha de entrega. Tras reordenar las tareas en orden creciente de sus fechas de entrega $d_1 \leq d_2 \leq ... \leq d_n$, el algoritmo añade sucesivamente siguiendo este orden las tareas al conjunto J. Si al añadir una tarea k a J dicha tarea se realiza más tarde de la fecha de entrega, entonces, de entre todas las tareas ya programadas, se elimina de J aquella con mayor tiempo de procesamiento y se añade a J^d.

Por tanto, al aplicar las iteraciones del algoritmo, conocido como algoritmo de Hodgson-Moore, se crean conjuntos $J_1,...,J_n$ de manera que el subconjunto J_k (subconjunto J en la $k-$ésima iteración) está formado por las tareas que son candidatas a satisfacer su fecha de entrega. Finalmente, el conjunto J_n será el compuesto por todas las tareas que satisfacen su fecha de entrega en la programación óptima generada, cuando se realizan según el orden en el que se han añadido a J. En el Algoritmo 3 se describe el algoritmo de Hodgson-Moore. En el apéndice A.3. se incluye un ejemplo de aplicación del algoritmo de Hodgson-Moore.

Algoritmo 3: Algoritmo de Hodgson-Moore

```
1  Datos: J = ∅, J^d = ∅, d_j, p_j, j ∈ N y t := 0;
2  Ordenar las tareas en orden no decreciente de d_j;
3  para k = 1 a n hacer
4      J := J ∪ {k};
5      t := t + p_k;
6      si t > d_k entonces
7          Tomar la tarea j tal que p_j = max_{h∈J} p_h;
8          J := J \ {h};
9          t := t − p_h;
10         J^d := J^d ∪ {h};
11     fin
12 fin
13 devolver Programación ordenada de tareas;
```

Teorema 3.5. *El algoritmo de Hodgson-Moore proporciona una programación óptima para el modelo* $1||\sum U_j$.

Demostración. Sea j la primera tarea suprimida del conjunto J en el paso 8 del Algoritmo 3. Veamos que existe una programación óptima $P = (J, J^d)$ con $j \in J^d$.

Sea k la tarea añadida al conjunto J en el paso 4 del Algoritmo 3 al descartar la tarea j. Se tiene que $p_j = \max_{h \in J} p_h$ y $k \in J$. Por tanto, en la secuencia $\{1, 2, ..., j-1, j+1, ..., k\}$ ninguna tarea es realizada con retraso ya que en la secuencia $\{1, 2, ..., k-1\}$ ninguna tarea se realiza con retraso y, además, $p_k \leq p_j$. Entonces, se reemplaza la tarea j por k y se reordenan las tareas en orden no decreciente de sus fechas de entrega.

Sea $P' = (J', (J^d)')$ una secuencia óptima con $j \in J'$. Existe una secuencia óptima $\{h_1, ..., h_m, ..., h_r, h_{r+1}, ..., h_n\}$ con

$$(J^d)' = \{h_{r+1}, ..., h_n\} \tag{3.4}$$

$$d_{h_1} \leq ... \leq d_{h_r} \tag{3.5}$$

$$\{h_1, ..., h_m\} \subseteq \{1, ..., k\} \tag{3.6}$$

$$\{h_{m+1}, ..., h_r\} \subseteq \{k+1, ..., n\} \tag{3.7}$$

$$j \in \{h_1, ..., h_m\} \tag{3.8}$$

Como $d_1 \leq d_2 \leq ... \leq d_n$, existe siempre m de modo que se cumplen (3.6) y (3.7). Además, como $j \in J' \cap \{1, ..., k\}$ se cumple (3.8). Como $\{h_1, ..., h_m\} \subseteq J'$, ninguna tarea en $\{h_1, ..., h_m\}$ se realiza con retraso. Por otro lado, si hay una tarea con retraso en $\{1, ..., k\}$ y, por tanto, en cualquier programación $\{h_1, ..., h_m\} \subsetneq \{1, ..., k\}$. Esto implica que se tiene una tarea r con $1 \leq r \leq k$ y $r \notin \{h_1, ..., h_m\}$. Luego eliminamos la tarea j de $\{h_1, ..., h_m\}$ y la reemplazamos por r.

Si se ordenan las tareas del conjunto $\{h_1, ..., h_m\} \cup \{r\} \setminus \{j\} \subseteq \{1, ..., k\} \setminus \{j\}$ en orden no decreciente de sus fechas de entrega, todas ellas serán realizadas a tiempo al tener $\{1, ..., k\} \setminus \{j\}$ dicha propiedad. Al añadir las tareas $h_{m+1}, ..., h_r$ al conjunto $\{h_1, ..., h_m\} \cup \{r\} \setminus \{j\}$ y ordenarlos de nuevo en orden no decreciente de sus fechas de entrega, todas las tareas serán realizadas a tiempo ya que al tener $p_r \leq p_j$ se cumple que

$$\sum_{i=1}^{m} p_{h_i} - p_j + p_r \leq \sum_{i=1}^{m} p_i$$

y por lo tanto, obtenemos que una programación óptima será $P = (J, J^d)$ con $j \in J^d$.

Por último, veamos por inducción sobre el número de tareas que se cumple el teorema. El algoritmo es correcto si $k = 1$. Supongamos que es cierto para problemas con $k-1$ tareas y veamos si se cumple para k tareas. Sea $P = (J, J^d)$ la programación construida por el algoritmo y sea $P' = (J', (J^d)')$ una secuencia óptima con $j \in (J^d)'$. Por ser P' óptima se cumple que $|J| \leq |J'|$. Aplicando el algoritmo al conjunto de tareas $\{1, ..., j-1, j+1, ..., n\}$ se obtiene una programación óptima de la forma $(J, J^d \setminus \{j\})$. Como $(J', (J^d)' \setminus \{j\})$ es una programación factible para el conjunto de tareas $\{1, ..., j-1, j+1, ..., n\}$, entonces $|J'| \leq |J|$. Por tanto, $|J| = |J'|$ y P es óptima. □

3.4. Modelo $1||\sum T_j$

En esta sección el objetivo es minimizar la tardanza total. Este problema es NP-duro y para él se ha desarrollado un algoritmo pseudo-polinomial basado en programación dinámica.

Teorema 3.6. *Si $p_j \leq p_k$ y $d_j \leq d_k$, entonces existe una programación óptima en donde la tarea j es realizada antes de la tarea k.*

Demostración. Sea S una programación óptima y supongamos que la tarea j sigue a la tarea k en S, es decir, $C_k \leq C_j$. Al ser $d_j \leq d_k$, entonces $\max\{0, T - d_k\} \leq \max\{0, T - d_j\}$ para todo T. Sea C'_k el tiempo de finalización de la tarea k en la nueva programación S' donde la tarea k es programada en la posición inmediatamente después de la tarea j. Como $C'_k - d_k = C_j - d_k$, entonces $\max\{0, C'_k - d_k\} = \max\{0, C_j - d_k\} \leq \max\{0, C_j - d_j\}$.

Como la programación S es óptima, la nueva programación S' no puede ser mejor, luego se verifica la igualdad y por lo tanto, la programación S' es óptima.

Además, si la tarea j sigue a la tarea k y ambas tareas son realizadas a tiempo, al cambiar la tarea k a la posición inmediatamente después de j la tarea k continúa realizándose a tiempo de manera que en la nueva programación la tardanza total no se incrementa. □

En el siguiente teorema se estudia cómo afecta a la programación óptima una modificación en las fechas de entrega.

Teorema 3.7. *Sea C'_k el tiempo de finalización más tardío de la tarea k en cualquier programación óptima y sea S' esta programación. Supongamos que se modifica la fecha de entrega de la tarea k que pasa a ser $\max\{d_k, C'_k\}$, sin que se modifiquen las restantes fechas de entrega. Sea S'' una programación óptima para las nuevas fechas de entrega. Entonces S'' es óptima para el problema original.*

Demostración. Sea S' una programación de las tareas 1 a n y sea $V'(S)$ la tardanza total con respecto a las fechas de entrega originales y $V''(S)$ la tardanza total con respecto a las nuevas fechas de entrega. Podemos escribir

$$V'(S') = V''(S') + A_k$$

y

$$V'(S'') = V''(S'') + B_k$$

Si $C'_k \leq d_k$ entonces $\max\{d_k, C'_k\} = d_k$. Por tanto las fechas de entrega son las mismas en ambos problemas y las programaciones que son óptimas para el nuevo problema lo son para el original.

Si $C'_k > d_k$, entonces $\max\{d_k, C'_k\} = C'_k$. Entonces

$$A_k = C'_k - d_k$$

$$B_k = \max\{0, \min\{C''_k, C'_k\} - d_k\}$$

siendo C''_k el tiempo de finalización de la tarea k en la programación S''. Como $\max\{0, \min\{C''_k, C'_k\} - d_k\} \leq \max\{0, C_k - d_k\} = C'_k - d_k$. Entonces $A_k \geq B_k$.

Como S'' es óptima para las nuevas fechas de entrega, $V''(S') \geq V''(S'')$. Por tanto

$$V'(S') = V''(S') + A_k \geq V''(S'') + A_k = V'(S'') + A_k - B_k \geq V'(S'')$$

.

Luego S'' ha de ser óptima para las fechas originales de entrega. □

En lo que sigue, supondremos que las tareas 1 a n están ordenadas en orden no decreciente de sus fechas de entrega $d_1 \leq ... \leq d_n$. Sea k la tarea con el mayor tiempo de procesamiento $p_k = \max\{p_1, ..., p_n\}$. Aplicando el teorema 3.6, existe una programación óptima en la que las tareas $\{1, ..., k-1\}$, en algún orden, se programan antes de la tarea k. De las tareas $\{k+1, ..., n\}$ algunas pueden ser programadas antes de la tarea k y otras después en la programación óptima. El siguiente teorema se centra en las tareas $k+1$ a n.

Teorema 3.8. *Existe un entero δ, con $0 \leq \delta \leq n-k$ y una programación óptima S en la que la tarea k es precedida por todas las tareas j cumpliendo $j \leq k+\delta$ y es seguida por las tareas j que cumplen que $j > k+\delta$.*

Demostración. Sea C'_k el tiempo de finalización más tardío de la tarea k en cualquier programación óptima con respecto a las fechas de entregas $d_1, ..., d_n$.

Sea S'' una programación óptima con respecto a las fechas de entrega $d_1, ..., d_{k-1}, \max\{C'_k, d_k\}$, $d_{k+1}, ..., d_n$. Sea C''_k el tiempo de finalización de la tarea k en esta programación.

Por el teorema 3.7, S'' es una programación óptima para las fechas de entregas originales. Por tanto, $C''_k \leq C'_k \leq \max\{C''_k, d_k\}$ por la definición de C'_k.

Podemos suponer que la tarea k no es precedida en S'' por ninguna tarea con una fecha de entrega posterior a $\max\{C'_k, d_k\}$. Notemos que, si este fuera el caso, esta tarea se realizaría a tiempo y cambiar su posición para insertarla después de la tarea k no incrementa la función objetivo. Además, por el teorema 3.6, la tarea k ha de ser precedida por todas las tareas con una fecha de entrega anterior a $\max\{C'_k, d_k\}$. Por tanto δ puede elegirse como el mayor entero tal que $d_{k+\delta} \leq \max\{C'_k, d_k\}$. □

En el Algoritmo 4 se describe, para cada $j \geq k$, una secuencia óptima en la cual el conjunto de tareas denominado $I_1 = \{1, .., j\} \setminus \{k\}$ es realizado antes que la tarea k. Mientras que el conjunto de tareas $I_2 = \{j+1, ..., n\}$ se procesa a continuación de la tarea k. Al final del proceso se obtiene la programación óptima de las n tareas empezando en el tiempo t.

En el apéndice A.4. se incluye un ejemplo de aplicación del algoritmo de programación dinámica.

Algoritmo 4: Minimizar la tardanza total

Datos: N, t, p_j, d_j,$j \in N$;
Hacer $I = N$;
si $I = \emptyset$ **entonces**
    σ^* :=secuencia vacía;
en otro caso
    Sean $i_1 < i_2 < ... < i_r$ las tareas en I;
    Encontrar i_k tal que $p_{i_k} := \max\{p_i \mid i \in I\}$;
    $f^* = \infty$;
    para $j = k$ **a** r **hacer**
        $I_1 := \{i_v \mid 1 \leq v \leq j; v \neq k\}$;
        $t_1 := t$;
        $\sigma_1 :=$ aplicar Algoritmo 4 al conjunto de datos: I_1, t_1, p_j, d_j,$j \in N$;
        $I_2 := \{i_v \mid j < v \leq r\}$;
        $t_2 := t + \sum_{v=1}^{j} p_{i_v}$;
        $\sigma_2 :=$ aplicar Algoritmo 4 al conjunto de datos: I_2, t_2, p_j, d_j,$j \in N$;
        $\sigma := \sigma_1 \circ i_k \circ \sigma_2$;
        Hacer $f(\sigma,t) =$ la tardanza total de la programación σ;
        si $f(\sigma,t) < f^*$ **entonces**
            $\sigma^* := \sigma$;
            $f^* := f(\sigma,t)$;
        fin
    fin
fin
devolver *Programación ordenada de tareas* σ^**;*

Capítulo 4

Estudio computacional

En este capítulo, se presentan los resultados del estudio comparativo realizado sobre los modelos de optimización descritos en el capítulo 2 y los algoritmos descritos en el capítulo 3 para los modelos $1||\sum w_j C_j$, $1||L_{max}$, $1||\sum U_j$ y $1||\sum T_j$.

4.1. Entorno del estudio computacional

Los modelos de optimización descritos en el capítulo 2, tanto los basados en variables que indican precedencia como los basados en variables de posición, se han resuelto usando la versión académica del software CPLEX Studio en su versión 20.1.0. Los códigos pueden verse en el apéndice B. Por otro lado, los algoritmos basados en propiedades combinatorias descritos en el capítulo 3 han sido implementados mediante el uso del lenguaje de programación C++. Para su desarrollo se ha utilizado el entorno de desarrollo integrado de código abierto Code::Blocks. Los códigos pueden verse en el apéndice C. Todas las pruebas han sido realizadas en un ordenador portátil personal 'Honor Magicbook 14' con sistema operativo Windows 10.0, procesador AMD Ryzen 5 3500U y 8GB de memoria RAM.

En el estudio computacional se han generado problemas con 20, 40 y 60 tareas. Para cada uno de ellos se han generado tres conjuntos de tareas. Para cada tarea j, su tiempo de procesamiento, p_j, se ha generado aleatoriamente como un entero con distribución uniforme $[0,100]$. Los costes o pesos w_j se han generado de igual manera siguiendo una distribución uniforme $[0,10]$. La fecha de entrega de cada tarea j es un entero generado de manera uniforme en el intervalo $[P(L-R/2),P(L+R/2)]$ donde $P = \sum_{j=1}^{n} p_j$ y L,R son parámetros constantes. Estos valores han sido propuestos en [5]. De entre los valores relativos a la distribución propuestos para R y L, en esta memoria se han seleccionado $R = 0{,}5$ y $L = 0{,}8$.

4.2. Análisis de los resultados

A continuación se presentan, para cada problema de programación de tareas considerado, los resultados obtenidos en términos del valor de la función objetivo y el tiempo de CPU invertido en su resolución en segundos. En CPLEX se fijó un tiempo límite de 600 segundos como criterio de parada. Si se alcanza ese límite, en la columna de la función objetivo se muestra el valor de la mejor solución factible obtenida hasta ese momento y se indica en la tabla con un asterisco, para hacer notar que la solución puede no ser óptima.

4.2.1. Modelo $1||\sum w_j C_j$

Los resultados obtenidos se presentan en la tabla 4.1. En ella, se comprueba que los tiempos de cálculo son mucho menores cuando se aplica el algoritmo WSPT. De hecho, CPLEX no ha podido garantizar la solución óptima en ninguno de los problemas. No obstante, comparando los valores de la

Tabla 4.1: Resultados obtenidos para el modelo $1||\sum w_j C_j$ (tiempo de CPU en segundos)

			Modelos de optimización			
	Algoritmo WSPT		Variables de precedencia		Variables de posición	
Tareas	Función objetivo	Tiempo CPU	Función objetivo	Tiempo CPU	Función objetivo	Tiempo CPU
20_1	43779	1	43779*	600	43779*	600
20_2	42514	1	42514*	600	42514*	600
20_3	48253	1	48253*	600	48253*	600
40_1	125837	1	125837*	600	125837*	600
40_2	83113	1	83113*	600	83114*	600
40_3	120607	1	120607*	600	120775*	600
60_1	288645	1	288693*	600	294824*	600
60_2	309988	1	310001*	600	314844*	600
60_3	349219	1	349225*	600	356528*	600

función objetivo con el óptimo proporcionado por el algoritmo WSPT, en todos los problemas con 20 tareas ha alcanzado la solución óptima. Para los problemas de 40 tareas, la ha obtenido únicamente para el modelo con variables de precedencia. Para los problemas con 60 tareas no la ha obtenido con ninguno de los modelos. En general, el modelo con variables de precedencia proporciona los mejores resultados entre los dos modelos de optimización entera.

4.2.2. Modelo $1||L_{max}$

Tabla 4.2: Resultados obtenidos para el modelo $1||L_{max}$ (tiempo de CPU en segundos)

			Modelos de optimización			
	Algoritmo EDD		Variables de precedencia		Variables de posición	
Tareas	Función objetivo	Tiempo CPU	Función objetivo	Tiempo CPU	Función objetivo	Tiempo CPU
20_1	174	1	174*	600	174	1
20_2	108	1	108*	600	108	1
20_3	118	1	118*	600	118	1
40_1	280	1	280*	600	280	1
40_2	319	1	319*	600	319	95
40_3	255	1	255*	600	255	5
60_1	413	1	436*	600	413	4
60_2	373	1	856*	600	373	5
60_3	380	1	480*	600	380	7

Los resultados se presentan en la tabla 4.2. En ella, se observa que los tiempos de cálculos son mucho menores cuando se aplica, o bien el algoritmo EDD, o bien el modelo de optimización con variables de posición. Ambos han llegado a la solución óptima en un tiempo de CPU reducido aunque el algoritmo EDD es más rápido ya que, en general, nunca ha necesitado más de un segundo para alcanzar la solución óptima. CPLEX no ha podido garantizar la solución óptima en ninguno de los problemas para el modelo con variables de precedencia. Comparando los valores de la función objetivo de dicha modelización con el óptimo proporcionado por el algoritmo EDD, por ejemplo, en todos los problemas con 20 y 40 tareas ha alcanzado la solución óptima. Para los problemas con 60 tareas no la ha obtenido en ninguno de los problemas.

4.2.3. Modelo $1||\sum U_j$

Los resultados se presentan en la tabla 4.3. El valor de la función objetivo será el total de tareas que han sido realizadas con retraso, es decir, sin cumplir su fecha de entrega. En ella, se comprueba que los

Tabla 4.3: Resultados obtenidos para el modelo $1||\sum U_j$ (tiempo de CPU en segundos)

			Modelos de optimización			
	Algoritmo Hosgson-Moore		Variables de precedencia		Variables de posición	
Tareas	Función objetivo	Tiempo CPU	Función objetivo	Tiempo CPU	Función objetivo	Tiempo CPU
20_1	3	1	3	3	3	1
20_2	2	1	2*	600	2	1
20_3	2	1	2	112	2	1
40_1	3	1	7*	600	3	2
40_2	4	1	5*	600	4	7
40_3	3	1	5*	600	3	15
60_1	5	1	16*	600	5	10
60_2	4	1	21*	600	4	7
60_3	4	1	15*	600	4	9

tiempos de cálculos son mucho menores cuando se aplica el algoritmo de Hodgson-Moore y el modelo con variables de posición. Aunque CPLEX ha obtenido la solución óptima para el modelo con variables de precedencia en dos problemas con solo 20 tareas, en general no es capaz de proporcionar la solución óptima: en todos los problemas con 40 y 60 tareas, CPLEX no ha obtenido la solución óptima.

4.2.4. Modelo $1||\sum T_j$

En un estudio preliminar se observó que la resolución de los problemas con 40 y 60 tareas era muy costosa en tiempo de cálculo (>2 horas) tanto con el algoritmo basado en programación dinámica como con ambos modelos de optimización. Por ello, se optó por considerar seis conjuntos de 20 tareas denotados por $\{D_1,...,D_6\}$. Para cada conjunto de tareas se han generado tres problemas distintos que denotaremos por $\{D_{j1},D_{j2},D_{j3}\}$, $j=\{1,...,6\}$. Los distintos problemas estudiados corresponden a los siguientes valores de L y R mencionados en la sección 4.1:

$L\backslash R$	0,4	0,8	1,4
0,5	D_1	D_2	D_3
0,7	D_4	D_5	D_6

Debido a la forma en la que se han definido los intervalos correspondientes a la generación de d_j, cuando $R=1{,}4$ y $L=0{,}5$ la cota inferior de la distribución sería negativa y en ningún caso las fechas de entrega pueden ser menores que cero al estar todas las tareas disponibles en el instante 0. Por ello, se ha fijado el valor inferior de dicho intervalo en 0.

Para la resolución de los problemas, se estableció un tiempo límite para CPLEX de 600 segundos y no se fijó ningún límite para el algoritmo basado en la programación dinámica. Los resultados se presentan en la tabla 4.4. En este caso, puede observarse que el modelo con variables de posición tiene los mejores resultados en tiempo de cálculo ya que, salvo en el problema D_{21}, proporciona la solución óptima, necesitando entre 1 y 120 segundos con una media de 17 segundos. Para el modelo de optimización con variables de precedencia CPLEX ha proporcionado la solución óptima en tres de los conjuntos en tiempos de cálculo muy pequeños (entre 1 y 9 segundos) mientras que se ha aplicado el criterio de parada en los otros tres conjuntos. En estos casos, comparando con la solución óptima proporcionada por el algoritmo basado en programación dinámica, puede verse que en 6 de los 9 conjuntos la solución proporcionada también es la óptima. El algoritmo basado en programación dinámica proporciona la solución óptima en todas las tareas, pero precisa tiempos grandes de cálculo (entre 254 y 1765 segundos).

Por otro lado, no parece haber una relación directa entre el tiempo invertido en la resolución según la aproximación utilizada. Problemas "fáciles de resolver" con los modelos de optimización tienen tiempos muy diferentes de cálculo con el algoritmo basado en la programación dinámica. Finalmente, los problemas en los que $L=0{,}7$ tienen, en general, menores valores de la función objetivo, es decir, menor tardanza total. Esto puede explicarse porque las fechas de entrega son mayores y es, por tanto,

Tabla 4.4: Resultados obtenidos para el modelo $1||\sum T_j$ (tiempo de CPU en segundos)

			Modelos de optimización			
	Programación dinámica		Variables de precedencia		Variables de posición	
Tareas	Función objetivo	Tiempo CPU	Función objetivo	Tiempo CPU	Función objetivo	Tiempo CPU
D_{11}	1070	942	1322*	600	1070	26
D_{12}	1467	1765	1467*	600	1467	10
D_{13}	1414	1232	1414*	600	1414	7
D_{21}	1233	978	1267*	600	1233*	600
D_{22}	1876	432	1876*	600	1876	112
D_{23}	372	254	372*	600	372	6
D_{31}	146	652	146	1	146	2
D_{32}	362	378	362	2	362	11
D_{33}	725	646	725	9	725	120
D_{41}	202	864	205*	600	202	1
D_{42}	144	756	144*	600	144	1
D_{43}	341	920	341*	600	341	1
D_{51}	0	624	0	1	0	1
D_{52}	0	486	0	5	0	1
D_{53}	0	522	0	2	0	1
D_{61}	14	565	14	2	14	3
D_{62}	0	784	0	1	0	1
D_{63}	195	492	195	1	195	1

menos frecuente que no se cumpla la fecha de entrega. Además, estos problemas, con el modelo con variables de posición se resuelven todos en alrededor de un segundo.

4.3. Conclusiones del estudio computacional

Al comparar los resultados obtenidos podemos concluir que no existe una aproximación que sea la mejor en todos los modelos. En general, como sería esperable, los algoritmos polinomiales proporcionan los mejores tiempos de cálculo (problemas $1||\sum w_j C_j$, $1||L_{max}$, $1||\sum U_j$).

Refiriéndonos a la comparación entre las formulaciones como modelos de optimización entera, la formulación con variables de posición es mejor para los modelos $1||L_{max}$, $1||\sum U_j$ y $1||\sum T_j$ mientras que la formulación con variables de precedencia es mejor para el modelo $1||\sum w_j C_j$.

Por último, para el modelo $1||\sum T_j$ la formulación con variables de posición es claramente la mejor forma de abordar el problema, seguida del algoritmo basado en programación dinámica. No obstante, debe hacerse constar que, en este caso, el estudio ha incluido solo problemas con 20 tareas.

Bibliografía

[1] P. BRUCKER, *Scheduling Alhorithms*, 5.ª ed., Springer, 2007.

[2] M. L. PINEDO, *Scheduling. Theory, Algorithms, and Systems*, 4.ª ed., Springer, 2011.

[3] D. R. SULE, *Production planning and industrial Scheduling*, 2.ª ed., CRC Press, 2008.

[4] J. Y-T. LEUNG, *Handbook of Scheduling. Algorithms, Models, and Performance Analysis*, Chapman & Hall/CRC, 2004.

[5] A. B. KEHA, K. KHOWALA Y J.W. FOWLER, Mixed integer programming formulations for single machine scheduling problems, *Computers & Industrial Engineering* **56** (2009), 357–367.

[6] K. R. BAKER Y B. KELLER, Solving the single-machine sequencing problem using integer programming, *Computers & Industrial Engineering* **59** (2010), 730–735.

Apéndice A

Ejemplos de los algoritmos presentados en el capítulo 3

A.1. Ejemplo del modelo $1||\sum w_jC_j$

Ejemplo 1. Sea el problema $1||\sum w_jC_j$ con los siguientes tiempos de procesamiento y pesos:

Tareas	1	2	3	4	5	6	7
w_j	1	18	12	8	8	17	16
p_j	3	6	6	5	4	8	9

Se calculan para todas las tareas el ratio S_j definido como $S_j = w_j/p_j$

Tareas	1	2	3	4	5	6	7
S_j	1/3	3	2	8/5	2	17/8	16/9

Al ordenar los valores de S_j en orden decreciente y aplicar la regla WSPT se obtiene el orden de realización de las tareas:

$$2 \to 6 \to 3 \to 5 \to 7 \to 4 \to 1$$

Ejemplo 2. Con los mismos datos que en el ejemplo anterior, suponemos ahora que entre las tareas existen restricciones de tipo cadena de la siguiente manera:

$$1 \to 2$$

$$3 \to 4 \to 5$$

$$6 \to 7$$

En primer lugar se calcula el ρ-factor de las tres cadenas. El ρ-factor de la primera cadena es $(1+18)/(3+6) = 19/9$ y viene determinado por la tarea 2. La segunda cadena tendrá un ρ-factor de $12/6 = 2$ y está determinado por la tarea 3. Por último, el ρ-factor de la tercera cadena es $17/8$, determinado por la tarea 6.

El mayor ρ-factor es el correspondiente a la cadena 1 y por ello, las tareas 1 y 2 serán las primeras en realizarse. Se vuelve a realizar el mismo proceso con las dos cadenas restantes. En ellas, la tercera cadena tiene el mayor ρ-factor y por tanto, la tarea 6 se realiza a continuación.

Se calcula el nuevo ρ-factor de la cadena 3 actualizada tras eliminar la tarea 6. Como solo hay una única tarea, esta será la que determine el ρ-factor$= 16/9$. Al comparar el nuevo ρ-factor de la cadena 3 con el ρ-factor de la segunda, el de la segunda cadena es mayor y por tanto, la tarea 3 será la siguiente en ser programada.

Se calcula de nuevo el ρ-factor de la cadena 2 con las tareas 4 y 5 restantes, que es $16/9$. Como los dos ρ-factores restantes son iguales da igual que conjunto de tareas se realiza en último lugar.

Al ordenar todas las tareas se obtiene la programación: $1 \to 2 \to 6 \to 3 \to 7 \to 4 \to 5$.

A.2. Ejemplo del modelo $1||L_{max}$

Ejemplo 3. Sea el modelo $1||L_{max}$, en este ejemplo se estudia el modelo general $1||h_{max}$ con los siguientes tiempos de procesamiento y funciones h_j:

Tareas	1	2	3	4	5	6	7
p_j	4	8	12	7	6	9	9
$h_j(C_j)$	$3C_1$	77	C_3^2	$1{,}5C_4$	$70+\sqrt{C_5}$	$1{,}6C_6$	$1{,}4C_7$

En el inicio, $J=\emptyset, J^c=\{1,...,7\}$ y $\sum_{j=1}^{7} p_j = 55$. Por tanto:

Tareas	1	2	3	4	5	6	7
$h_j(55)$	165	77	3025	82,5	77,41	88	77

Como se puede observar, $\min_{j=1,...,7} h_j(55) = 77$ que corresponde a la tarea 2 ó la 7. Por tanto, la última tarea en completarse sería la 2 o la 7. Como en ambos casos llegaremos a una solución óptima, seguimos el procedimiento tomando como última tarea la 2.

Ahora se tiene que $J=\{2\}, J^c=\{1,3,4,5,6,7\}$. Las seis tareas restantes tendrán un tiempo de procesamiento de 47 unidades de tiempo. Es decir, ahora $C_{max}=47$. Repitiendo el proceso para las tareas del conjunto J^c:

Tareas	1	3	4	5	6	7
$h_j(47)$	141	2209	70,5	76,85	75,2	65,8

Como $\min_{j\in J^c} h_j(47) = 65{,}8$ que corresponde a la tarea 7, esta tarea se elimina de J^c y se añade a J. La tarea 7 precederá a la tarea 2 en la programación óptima. Siguiendo el proceso realizado se obtiene que la programación óptima es: $3\to 5\to 1\to 6\to 4\to 7\to 2$.

A.3. Ejemplo del modelo $1||\sum U_j$

Ejemplo 4. Sea el problema $1||\sum U_j$ con los siguientes tiempos de procesamiento y fechas de entrega

Tareas	1	2	3	4	5	6	7	8
p_j	6	8	12	10	10	11	5	7
d_j	8	42	44	24	26	26	70	75

En primer lugar, como para aplicar el algoritmo de Hodgson-Moore las tareas han de estar ordenados según la fecha de entrega en orden ascendente, reordenamos la tabla:

Tareas	1	4	5	6	2	3	7	8
p_j	6	10	10	11	8	12	5	7
d_j	8	24	26	26	42	44	70	75

Inicialmente $J=\emptyset$, $J^d=\emptyset$ y $t=0$.

Notamos que las tareas 1, 4 y 5 pueden ser realizadas en primer lugar cumpliendo sus fechas de entrega. Observamos que $p_1+p_4+p_5=26\leq d_5$ y, por lo tanto, las tres tareas son realizadas a tiempo.

A continuación, seleccionamos la tarea 6. Como podemos observar, será imposible cumplir su fecha de entrega ya que siguiendo el orden inicial y añadiendo 6 tendríamos que $t=37>d_6$. Por lo tanto, según el algoritmo, desechamos la tarea con mayor tiempo de procesamiento de J y la añadimos al conjunto J^d. En este caso, será la misma tarea 6 la descartada. Luego se tiene que $J=\{1,4,5\}$ y $J^d=\{6\}$.

Siguiendo el mismo proceso, añadimos la tarea 2 que cumple su fecha de entrega al conjunto J. Al añadir la tarea 3 al conjunto J tendríamos que $t = 46 > d_3$. Por lo tanto, aplicando el algoritmo, descartamos aquella tarea con mayor tiempo de procesamiento. Como en este caso es la tarea 3, tenemos que $J^d = \{3,6\}$ son las tareas descartadas y cuya fecha de entrega será imposible de satisfacer.

Procediendo de la misma manera tenemos que las tareas 7 y 8 pueden ser realizadas satisfaciendo su fecha de entrega y por lo tanto, la programación óptima es: $1 \to 4 \to 5 \to 2 \to 7 \to 8 \to 6 \to 3$. En ella, las tareas $1,4,5,2,7$ y 8 se realizan antes de su fecha de entrega, mientras que no lo hacen las tareas 3 y 6. Es decir, 2 de las 8 tareas no se realizan a tiempo.

A.4. Ejemplo del modelo $1||\sum T_j$

Ejemplo 5. Sea el problema $1||\sum T_j$ con los siguientes tiempos de procesamiento y fechas de entrega:

Tareas	1	2	3	4	5	6	7	8
p_j	6	18	12	10	10	11	5	7
d_j	8	42	44	24	26	26	70	75

En primer lugar, ordenamos las tareas según la fecha de entrega en orden ascendente para ver si podemos aplicar el lema 3.6 y así usar el criterio de eliminación. Reordenamos la tabla:

Tareas	1	4	5	6	2	3	7	8
p_j	6	10	10	11	18	12	5	7
d_j	8	24	26	26	42	44	70	75

Como podemos observar, para $j = 1,4,5,6$ y $k = 2$ se cumple que $p_j \leq p_k$ y $d_j \leq d_k$. Por tanto, existe una secuencia óptima en donde la tarea j es realizada antes de la tarea k.

Tras comprobar que las tareas $1,4,5$ y 6 van en primer lugar, escribimos de nuevo la tabla con las tareas restantes. Para evitar confusiones a la hora de aplicar el algoritmo, en esta segunda parte del problema, identificaremos la tarea 2 como 1 y la tarea 3, 7 y 8 como 2, 3 y 4, respectivamente. Por tanto las nuevas tareas a programar son:

Tareas	1	2	3	4
p_j	18	12	5	7
d_j	42	44	70	75

Como las condiciones iniciales han de empezar en $t = 0$, restaremos a las fechas de entrega las 37 unidades de tiempo que ya se llevan consumidas por la realización de las tareas de número original 1, 4, 5 y 6.

Tareas	1	2	3	4
p_j	18	12	5	7
d_j	5	7	33	38

La tarea 1 es la que tiene un tiempo de procesamiento mayor y por lo tanto será la tarea k. De esta manera tenemos que existe δ de manera que $0 \leq \delta \leq n-k = 3$.

De antemano podemos saber que la nueva tarea 3 precederá a la 4 al tener el tiempo de procesamiento y la fecha de entrega menor. Como las fechas de entrega ya están ordenadas, por el teorema 3.8 tenemos cuatro posibles ordenaciones de las tareas.

Caso 1.

La tarea número 1 va en primer lugar y las tareas $2,3$ y 4 le siguen. Veamos cuál sería la secuencia óptima del conjunto $J(2,4,1)$, donde la 2 será la primera tarea del conjunto, la 4 la última y el 1 denota la tarea con tiempo de procesamiento mayor de todas las tareas del conjunto $\{2,3,4\}$.

Renumeramos las tareas $2,3,4$ para volver a aplicar la secuencia de manera que pasan a ser $1',2',3'$ y su tiempo inicial será $t = 18$.

De las tres tareas, aquella con mayor p_j es la $1'$ y por tanto $\exists \delta'$ tal que $0 \leq \delta' \leq n-k = 3-1' = 2$.

Si $\delta' = 0$, entonces la ordenación sería
$1' + 2' - 3' \rightarrow$ Tardanza=$(18+12-7)+(30+5-33)+(35+7-38) = 29$
Si $\delta' = 1$, entonces la ordenación sería
$2' + 1' + 3' \rightarrow$ Tardanza=$(18+5-33)+(23+12-7)+(35+7-38) = 32$
Si $\delta' = 2$, entonces la ordenación sería
$2' + 3' + 1' \rightarrow$ Tardanza=$(18+5-33)+(23+7-38)+(30+12-7) = 35$

Por tanto, la secuencia óptima de todas ellas sería la primera en la que las tareas vienen ordenadas en orden ascendente $(2-3-4)$. Lo cual hace que volviendo al problema principal la secuencia óptima sea: $1-2-3-4$ cuya tardanza será de:

Tardanza = Tardanza 1 + Tardanza $\{2-3-4\} = (0+18-5)+29 = 42$.

Caso 2.

La tarea número 1 iría precedida por la 2 que en este caso va en primer lugar. A continuación irían las tareas 3 y 4. Como sabemos que la 3 precede siempre a la 4 ya se tiene la secuencia óptima $(2-1-3-4)$ y solo tendremos que calcular su tardanza.

Tardanza = Tardanza 2 + Tardanza 1 + Tardanza $\{3,4\}$=
$= (12-7)+(30-5)+((30+5-33)+(35+7-38)) = 36$

Caso 3.

La tarea número 1 iría precedida por el conjunto $J(1,3,1)$, es decir las tareas 2 y 3. Por último iría la tarea 4. Dentro del conjunto $J(1,3,1)$ es obvio que la tarea 2, de mayor p_j irá en primer lugar para así reducir la tardanza. Por tanto, la secuencia óptima de este caso sería: $2-3-1-4$. Calculemos su tardanza como antes:

Tardanza = Tardanza $\{2,3\}$ + Tardanza 1 + Tardanza 4=$((12-7)+0)+(35-5)+(42-38) = 39$

Caso 4.

En este último caso, la tarea de mayor tiempo de procesamiento irá en último lugar mientras que el conjunto $J(1,4,1)$ irá en primer lugar. Notar que en el primer caso ya hemos obtenido la secuencia óptima para dicho subconjunto: $2-3-4$. Por ello, la secuencia óptima de este caso sería: $2-3-4-1$.
Tardanza = Tardanza $\{2,3,4\}$ + Tardanza $1 = ((12-7)+0+0)+(42-5) = 42$

Por tanto, una vez estudiados todos los casos posibles para los distintos valores de δ ya podemos concluir que la secuencia óptima será aquella con el menor valor de la función objetivo. En nuestro caso el caso 2 será el óptimo y por tanto, la secuencia buscada será $2-1-3-4$.

Como antes habíamos cambiado la numeración de las tareas, volvemos a las tareas iniciales y de esta manera la solución óptima que resolverá nuestro problema será la siguiente:

$$1 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 3 \rightarrow 2 \rightarrow 7 \rightarrow 8$$

Apéndice B

Programación en CPLEX Studio

B.1. Modelo $1||\sum w_j C_j$

WeigComTime.mod

```
/*********************************************
 * OPL 20.1.0.0 Model
 * Author: victo
 * Creation Date: 27 abr. 2021 at 18:10:09
 *********************************************/
//Datos (se leen del archivo .dat)
int nI = ...;
int nJ = ...;
range I = 1..nI;
range J = 1..nJ;
float d[I][J] = ...;

//Variables
dvar int C[J];
dvar boolean y[J][J];
dexpr float M=sum(j in J)(d[2][j]);


//Función objetivo
minimize sum(j in J )( d[1][j]*C[j] );

//Restricciones
subject to {

  forall( j in J ){
     C[j] >= d[2][j];
  }

  forall(j,k in J : j<k){
     C[j]+d[2][k] <= C[k]+M*(1-y[j][k]);
  }

  forall(j,k in J : j<k){
     C[k]+d[2][j] <= C[j]+M*y[j][k];
  }

  forall( j in J ){
     C[j] >= 0;
  }

}
```

2WeigComTime.mod

```
/*********************************************
 * OPL 20.1.0.0 Model
 * Author: victo
 * Creation Date: 19 jun. 2021 at 13:28:19
 *********************************************/
//Datos (se leen del archivo .dat)
int nI = ...;
int nJ = ...;
range I = 1..nI;
range J = 1..nJ;
float d[I][J] = ...;

//Variables
dvar int C[J];
dexpr float M=sum(j in J)(d[2][j]);
dvar boolean u[J][J];
dvar int gamma[J];



//Función objetivo
minimize sum(j in J )( d[1][j]*C[j] );

//Restricciones
subject to {

  forall( j in J ){
     sum(k in J)(u[j][k])==1;
  }
  forall( k in J ){
     sum(j in J)(u[j][k])==1;
  }

  gamma[1]>=sum(j in J)(d[2][j]*u[j][1]);

  forall(k in J: k>1){
  gamma[k]>=gamma[k-1]+sum(j in J)(d[2][j]*u[j][k]);
   }

  forall( k in J ){
    gamma[k]>=0;
  }
  forall(j,k in J){
     C[j]>= gamma[k]-M*(1-u[j][k]);
  }

  forall( j in J ){
     C[j] >= 0;
  }

}

//Para escribir un archivo con la solución encontrada
execute{
  var archivoSalida = new IloOplOutputFile("solucion2WComTime60c.txt");

  archivoSalida.writeln("Los Cj asociados a cada trabajo son: ");
  for(var j in thisOplModel.J){
  archivoSalida.write(thisOplModel.C[j], " ");
}
archivoSalida.writeln();
  archivoSalida.close();
```

B.2. Modelo $1||L_{max}$

MaxLateness2.mod

```
/*********************************************
 * OPL 20.1.0.0 Model
 * Author: victo
 * Creation Date: 27 abr. 2021 at 17:37:49
 *********************************************/
//Datos (se leen del archivo .dat)
int nI = ...;
int nJ = ...;
range I = 1..nI;
range J = 1..nJ;
float d[I][J] = ...;

//Variables
dvar int C[J];
dvar boolean y[J][J];
dvar float LMAX;
dexpr float M=sum(j in J)(d[2][j]);



//Función objetivo
minimize LMAX;

//Restricciones
subject to {

  forall( j in J ){
     C[j] >= d[2][j];
  }

  forall(j,k in J : j<k){
     C[j]+d[2][k] <= C[k]+M*(1-y[j][k]);
  }

  forall(j,k in J : j<k){
     C[k]+d[2][j] <= C[j]+M*y[j][k];
  }

  forall( j in J ){
     C[j] >= 0;
  }

  forall(j in J){
    LMAX>=(C[j]-d[3][j]);
  }
}

//Para escribir un archivo con la solución encontrada
execute{
  var archivoSalida = new IloOplOutputFile("solucionMLateness40.txt");

  archivoSalida.writeln("Los Cj asociados a cada trabajo son: ");
  for(var j in thisOplModel.J){
  archivoSalida.write(thisOplModel.C[j], " ");
}
archivoSalida.writeln();
  archivoSalida.writeln("El valor de LMAX es: ");
  archivoSalida.write(thisOplModel.LMAX, " ");
  archivoSalida.close();
}

```

Página 1

2MaxLateness.mod

```
/*********************************************
 * OPL 20.1.0.0 Model
 * Author: victo
 * Creation Date: 19 jun. 2021 at 14:03:31
 *********************************************/
//Datos (se leen del archivo .dat)
int nI = ...;
int nJ = ...;
range I = 1..nI;
range J = 1..nJ;
float d[I][J] = ...;

//Variables
dvar float LMAX;
dvar boolean u[J][J];
dvar int gamma[J];


//Función objetivo
minimize LMAX;
//Restricciones
subject to {

  forall( j in J ){
     sum(k in J)(u[j][k])==1;
  }
  forall( k in J ){
     sum(j in J)(u[j][k])==1;
  }

  gamma[1]>=sum(j in J)(d[2][j]*u[j][1]);

  forall(k in J: k>1){
  gamma[k]>=gamma[k-1]+sum(j in J)(d[2][j]*u[j][k]);
   }

  forall( k in J ){
    gamma[k]>=0;
  }
  forall(k in J){
    LMAX>=gamma[k]-sum(j in J)(d[3][j]*u[j][k]);
   }

}



//Para escribir un archivo con la solución encontrada
execute{
  var archivoSalida = new IloOplOutputFile("solucion1MLateness60c.txt");
  archivoSalida.writeln("El valor de LMAX es: ");
  archivoSalida.write(thisOplModel.LMAX, " ");
  archivoSalida.close();
}

```

B.3. Modelo $1||\sum U_j$

TardyJobs2.mod

```
/*********************************************
 * OPL 20.1.0.0 Model
 * Author: victo
 * Creation Date: 27 abr. 2021 at 12:24:37
 *********************************************/
//Datos (se leen del archivo .dat)
int nI = ...;
int nJ = ...;
range I = 1..nI;
range J = 1..nJ;
float d[I][J] = ...;

//Variables
dvar int C[J];
dvar boolean y[J][J];
dexpr float M=sum(j in J)(d[2][j]);
dvar boolean U[J];

//Función objetivo
minimize sum(j in J) U[j];

//Restricciones
subject to {

  forall( j in J ){
     C[j] >= d[2][j];
  }

  forall(j,k in J : j<k){
     C[j]+d[2][k] <= C[k]+M*(1-y[j][k]);
  }

  forall(j,k in J : j<k){
     C[k]+d[2][j] <= C[j]+M*y[j][k];
  }

  forall( j in J ){
     C[j] >= 0;
  }

  forall(j in J){
    C[j]<=d[3][j]+M*U[j];
  }
}

//Para escribir un archivo con la solución encontrada
execute{
  var archivoSalida = new IloOplOutputFile("solucionTardyJobs40c.txt");

  archivoSalida.writeln("Los Cj asociados a cada trabajo son: ");
  for(var j in thisOplModel.J){
  archivoSalida.write(thisOplModel.C[j], " ");
}
archivoSalida.writeln();
  archivoSalida.writeln("Los trabajos que no cumplen su fecha de entrega son: ");
  for(var j in thisOplModel.J){
  archivoSalida.write(thisOplModel.U[j], " ");
}
  archivoSalida.close();
}

```

Página 1

2TardyJobs.mod

```
/*********************************************
 * OPL 20.1.0.0 Model
 * Author: victo
 * Creation Date: 19 jun. 2021 at 16:52:14
 *********************************************/
//Datos (se leen del archivo .dat)
int nI = ...;
int nJ = ...;
range I = 1..nI;
range J = 1..nJ;
float d[I][J] = ...;

//Variables
dvar boolean u[J][J];
dvar int gamma[J];
dexpr float M=sum(j in J)(d[2][j]);
dvar boolean U[J];

//Función objetivo
minimize sum(j in J) U[j];
//Restricciones
subject to {

  forall( j in J ){
     sum(k in J)(u[j][k])==1;
  }
  forall( k in J ){
     sum(j in J)(u[j][k])==1;
  }

  gamma[1]>=sum(j in J)(d[2][j]*u[j][1]);

  forall(k in J: k>1){
  gamma[k]>=gamma[k-1]+sum(j in J)(d[2][j]*u[j][k]);
   }

  forall( k in J ){
    gamma[k]>=0;
  }
  forall(k in J){
    gamma[k]<=sum(j in J)(d[3][j]*u[j][k])+M*U[k];
  }
}


//Para escribir un archivo con la solución encontrada
execute{
  var archivoSalida = new IloOplOutputFile("solucion1TardyJobs60c.txt");

archivoSalida.writeln();
  archivoSalida.writeln("Los trabajos que no cumplen su fecha de entrega son: ");
  for(var j in thisOplModel.J){
  archivoSalida.write(thisOplModel.U[j], " ");
}
  archivoSalida.close();
}

```

B.4. Modelo $1||\sum T_j$

TotalTardiness2.mod

```
/*********************************************
 * OPL 20.1.0.0 Model
 * Author: victo
 * Creation Date: 27 abr. 2021 at 17:42:22
 *********************************************/
//Datos (se leen del archivo .dat)
int nI = ...;
int nJ = ...;
range I = 1..nI;
range J = 1..nJ;
float d[I][J] = ...;

//Variables
dvar int C[J];
dvar boolean y[J][J];
dexpr float M=sum(j in J)(d[2][j]);
dvar int T[J];


//Función objetivo
minimize sum(j in J )( d[1][j]*T[j] );

//Restricciones
subject to {

  forall( j in J ){
     C[j] >= d[2][j];
  }

  forall(j,k in J : j<k){
     C[j]+d[2][k] <= C[k]+M*(1-y[j][k]);
  }

  forall(j,k in J : j<k){
     C[k]+d[2][j] <= C[j]+M*y[j][k];
  }

  forall( j in J ){
     C[j] >= 0;
  }

  forall( j in J ){
     T[j] >= C[j]-d[3][j];
  }

  forall( j in J ){
     T[j] >= 0;
  }

}

//Para escribir un archivo con la solución encontrada
execute{
  var archivoSalida = new IloOplOutputFile("solucionTTardiness20.txt");

  archivoSalida.writeln("Los Cj asociados a cada trabajo son: ");
  for(var j in thisOplModel.J){
  archivoSalida.write(thisOplModel.C[j], " ");
}
archivoSalida.writeln();
  archivoSalida.writeln("La tardanza de cada tarea es: ");
  for(var j in thisOplModel.J){
```

Página 1

TotalTardiness2.mod

```
63   archivoSalida.write(thisOplModel.T[j], " ");
64 }
65   archivoSalida.close();
66 }
67
```

2TTardines.mod

```
/*********************************************
 * OPL 20.1.0.0 Model
 * Author: victo
 * Creation Date: 19 jun. 2021 at 18:50:08
 *********************************************/
//Datos (se leen del archivo .dat)
int nI = ...;
int nJ = ...;
range I = 1..nI;
range J = 1..nJ;
float d[I][J] = ...;

//Variables
dvar int C[J];
dexpr float M=sum(j in J)(d[2][j]);
dvar boolean u[J][J];
dvar int gamma[J];
dvar int T[J];


//Función objetivo
minimize sum(j in J )( T[j] );

//Restricciones
subject to {

  forall( j in J ){
     sum(k in J)(u[j][k])==1;
  }
  forall( k in J ){
     sum(j in J)(u[j][k])==1;
  }

  gamma[1]>=sum(j in J)(d[2][j]*u[j][1]);

  forall(k in J: k>1){
  gamma[k]>=gamma[k-1]+sum(j in J)(d[2][j]*u[j][k]);
   }

  forall( k in J ){
    gamma[k]>=0;
  }
  forall(j,k in J){
     C[j]>= gamma[k]-M*(1-u[j][k]);
  }

  forall( j in J ){
     C[j] >= 0;
  }
    forall( j in J ){
     T[j] >= C[j]-d[3][j];
  }

  forall( j in J ){
     T[j] >= 0;
  }

}
//Para escribir un archivo con la solución encontrada
execute{
  var archivoSalida = new IloOplOutputFile("solucion2TTardiness5.txt");
```

2TTardines.mod

```
63
64   archivoSalida.writeln("Los Cj asociados a cada trabajo son: ");
65   for(var j in thisOplModel.J){
66   archivoSalida.write(thisOplModel.C[j], " ");
67 }
68 archivoSalida.writeln();
69   archivoSalida.writeln("La tardanza de cada tarea es: ");
70   for(var j in thisOplModel.J){
71   archivoSalida.write(thisOplModel.T[j], " ");
72 }
73   archivoSalida.close();
74 }
```

Apéndice C

Programación en C++

C.1. Algoritmo WSPT

```
#include <cstdlib>
#include <cstdio>
#include <iostream> //para salida y entrada de datos por consola
#include <fstream>   //para salida y entrada de datos del fichero
using namespace std;
int main()
{
    int filas, columnas;
    const char* nombre_fichero1 = "100trabajosWSPT.txt"; //nombre del fichero de los datos
    ifstream fichero1 (nombre_fichero1);
    fichero1>>filas;
    fichero1>>columnas;
    float** matriz;
    matriz = new float*[filas];
    for (int i=0; i<filas; i++)
    {
        matriz[i]=new float[columnas];
    }
    for (int i=0; i<filas; i++)
    {
        for (int j=0; j<columnas; j++)
        {
            fichero1>>matriz[i][j];
        }
    }
    fichero1.close();
    float division[100], aux, auxi,auxil,axu;
    for(int j=0;j<columnas;j++){
        division[j]=matriz[1][j]/matriz[2][j];
    }
    int pos; //ordeno las tareas en función al ratio de la regla WSPT
    for(int j = 0; j < columnas; j++){
        pos=j;
        aux=division[j];
        axu=matriz[1][j];
        auxi=matriz[0][j];
        auxil=matriz[2][j];
        while((pos>0)&&(division[pos-1]<aux)){
            division[pos]=division[pos-1];
            matriz[0][pos]=matriz[0][pos-1];
            matriz[2][pos]=matriz[2][pos-1];
            matriz[1][pos]=matriz[1][pos-1];
            pos--;
        }
        matriz[0][pos]=auxi;
        matriz[2][pos]=auxil;
        matriz[1][pos]=axu;
        division[pos]=aux;
    }
    int suma1=0, matriz1[100][100];  //creo una matriz para calcular los Cj asociados a cada tarea
    for(int j = 0; j < columnas; j++){
        matriz1[0][j]=matriz[0][j];
        suma1=suma1+matriz[2][j];
        matriz1[1][j]=suma1;
    }
    int fin=0;
    for (int j=0;j<columnas;j++){
        fin=fin+matriz1[1][j]*matriz[1][j];
```

```
    }
    ofstream archivo1("resultado4.txt");
    archivo1<<"Segun la regla WSPT, las tareas ordenadas son: " << endl;
    for (int j = 0; j < columnas; j++){ // Imprime las posiciones de las tareas
        archivo1<<matriz[0][j]<<" ";
    }
    archivo1<<endl;
    archivo1 <<"Los Cj asociados a cada tarea son: " << endl;
```

```
67      for (int j = 0; j < columnas; j++){ // Imprime las posiciones de los Cj
68          archivo1 <<matriz1[1][j] << " ";
69      }
70      archivo1<<endl;
71      archivo1<<"El valor total de la FO es: "<<fin<<endl;
72      archivo1.close();
73      return 0;
74  }
```

C.2. Algoritmo EDD

```
#include <stdlib.h>
#include <stdio.h>
#include <iostream> //para salida y entrada de datos por consola
#include <fstream>   //para salida y entrada de datos del fichero
using namespace std;
int main(void)
{
    int filas, columnas;
    const char* nombre_fichero1 = "20trabajos.txt"; //nombre del fichero de los datos
    ifstream fichero1 (nombre_fichero1);
    fichero1>>filas;
    fichero1>>columnas;
    float** matriz;
    matriz = new float*[filas];
    for (int i=0; i<filas; i++)
    {
        matriz[i]=new float[columnas];
    }

    for (int i=0; i<filas; i++)
    {
        for (int j=0; j<columnas; j++)
        {
            fichero1>>matriz[i][j];
        }
    }
    fichero1.close();
    float aux,auxi, auxil;
    int pos;
    for(int j = 0; j < columnas; j++){     //ordenamos las tareas en función a dj
        pos=j;
        aux=matriz[2][j];
        auxi=matriz[0][j];
        auxil=matriz[1][j];
        while((pos>0)&&(matriz[2][pos-1]>aux)){
            matriz[2][pos]=matriz[2][pos-1];
            matriz[0][pos]=matriz[0][pos-1];
            matriz[1][pos]=matriz[1][pos-1];
            pos--;
        }
        matriz[2][pos]=aux;
        matriz[0][pos]=auxi;
        matriz[1][pos]=auxil;
    }
    cout <<"Segun la regla EDD, las tareas ordenadas son: " << endl;
    for (int j = 0; j < columnas; j++){
        cout <<matriz[0][j]<<" \t";
    }
    cout<<endl;
    int suma1=0;
    int matriz1[100][100]; //creo una nueva matriz para obtener los Cj de cada tarea
    for(int j = 0; j < columnas; j++){
        matriz1[0][j]=matriz[0][j];
        suma1=suma1+matriz[1][j];
        matriz1[1][j]=suma1;
        matriz1[2][j]=matriz[2][j];
    }
    int retraso[100]; //calculo el retraso que lleva cada tarea
    for(int j = 0; j < columnas; j++){
         if(matriz1[1][j]>matriz1[2][j]){
            retraso[j]=matriz1[1][j]-matriz1[2][j];
         }
    }
    int LMAX=0;
     for(int j = 0; j < columnas; j++){
         if(retraso[j]>LMAX){
```

```
67              LMAX=retraso[j];
68          }
69       }
70       cout<<LMAX<<endl;
71      ofstream archivo1("resultado1.txt"); //creo el fichero de salida
72      archivo1<<"Segun la regla EDD, las tareas ordenadas son: " << endl;
73      for (int j = 0; j < columnas; j++){
74          archivo1<<matriz[0][j]<< " ";
75      }
76      archivo1<<endl;
77      archivo1<<"Los Cj asociados a cada tarea son: " << endl;
78      for (int j = 0; j < columnas; j++){ // Imprime las posiciones
79          archivo1 <<matriz1[1][j] << " ";
80      }
81      archivo1<<endl;
82      archivo1<<"El valor de LMAX sera: "<< LMAX<<endl;
83      archivo1.close();
84      system("PAUSE"); //para que no se cierre al terminar el programa
85      return 0;
86  }
```

C.3. Algoritmo Hodgson-Moore

```
#include <stdlib.h>
#include <stdio.h>
#include <iostream> //para salida y entrada de datos por consola
#include <fstream>   //para salida y entrada de datos del fichero
#include <vector>
using namespace std;
using std::vector;
int main(void)
{
    int filas, columnas;
    const char* nombre_fichero1 = "40trabajos.txt"; //nombre del fichero de los datos
    ifstream fichero1 (nombre_fichero1);
    fichero1>>filas;
    fichero1>>columnas;
    float** matriz0;
    matriz0 = new float*[filas];
    //creo una matriz inicial con los datos a la que acudir al llamar posteriormente
    for (int i=0; i<filas; i++)
    {
        matriz0[i]=new float[columnas];
    }
    for (int i=0; i<filas; i++)
    {
        for (int j=0; j<columnas; j++)
        {
            fichero1>>matriz0[i][j];
        }
    }
    fichero1.close();
    int matriz[100][100];       //creo una nueva matriz para la ordenación
    for (int i=0; i<filas; i++)
    {
        for (int j=0; j<columnas; j++)
        {
            matriz[i][j]=matriz0[i][j];
        }
    }
    float aux, auxi, auxil;
    int pos;
    for(int j = 0; j < columnas; j++){      //ordenamos las tareas en función a dj
        pos=j;
        aux=matriz[0][j];
        auxi=matriz[1][j];
        auxil=matriz[2][j];
        while((pos>0)&&(matriz[2][pos-1]>auxil)){
            matriz[0][pos]=matriz[0][pos-1];
            matriz[1][pos]=matriz[1][pos-1];
            matriz[2][pos]=matriz[2][pos-1];
            pos--;
        }
        matriz[0][pos]=aux;
        matriz[1][pos]=auxi;
        matriz[2][pos]=auxil;
    }
    int suma;
    suma=0;
    int maximo=0,maxi=0;
    vector<int> s1,s2;
    for(int j = 0; j < columnas; j++){
        suma=suma+matriz[1][j];
        s1.insert(s1.end(),matriz[0][j]);
        int maximo2=0,k;
        //si la fecha de entrega es menor que el tiempo que llevamos entonces entra al bucle
        if(suma>matriz[2][j]){
            for(int i = 0; i < s1.size(); i++){
                if(maximo2<matriz0[1][s1[i]-1]){
```

```
67                      maximo2=matriz0[1][s1[i]-1];
68                      k=i;
69                  }
70              }
71              s2.push_back(s1[k]); //añado la tarea k a s2 que es la que tiene mayor pj
72              suma=suma-matriz0[1][s1[k]-1];//resto el pj de la tarea k al tiempo total
73              s1.erase(s1.begin()+k); //quito la tarea k de s1
74          }
75      }
76      int retraso=0;  //calculo el retraso de las tareas que están en s2
77      for (size_t i = 0; i < s2.size(); i++) {
78          suma=suma+matriz0[1][s2[i]-1];
79          retraso=retraso+suma-matriz0[2][s2[i]-1];
80      }
81      ofstream archivo1("resultado40.txt"); //creo el fichero con los datos que me interesan
82      archivo1 <<"Las tareas realizadas a tiempo ordenadas por orden de realizacion son: " << endl;
83      for (size_t i = 0; i < s1.size(); i++) {
84          archivo1 << s1[i] << " ";
85      }
86      archivo1 << endl;
87      archivo1<<"Mi conjunto de tareas no realizadas a tiempo son:"<<endl;
88      for (size_t i = 0; i < s2.size(); i++) {
89          archivo1<< s2[i] << " ";
90      }
91      archivo1<<endl;
92      archivo1<<"El retraso total de la programacion es: "<< retraso<<endl;
93      archivo1.close();
94          system("PAUSE"); //para que no se cierre al terminar el programa
95          return 0; //Debe devolver algo el main, lo que e de la gana
96  }
```

C.4. Programación dinámica

```cpp
#include <stdlib.h>
#include <stdio.h>
#include <iostream> //para salida y entrada de datos por consola
#include <fstream>   //para salida y entrada de datos del fichero
#include <vector>
#include<algorithm>
using namespace std;

//Variables globales
vector< vector<int> > matriz;
int sumaLema;


bool existeEnVector(vector<int> v, int busqueda) { //función que me busque un entero en un vector
    return find(v.begin(), v.end(), busqueda) != v.end();
}
vector<int> secuenciar(double t, vector <int> I){
    //función para secuenciar las tareas de un conjunto en función del tiempo inicial t
    vector<int> sigmax;
    if(I.size()==0){
        return sigmax;
    }
    else{
        int maximo2=0;
        int k;
        for(int j = 0; j < I.size(); j++){ //tomamos k como el maximo de las tareas
            if(maximo2<matriz[1][I[j]-1]){
                maximo2=matriz[1][I[j]-1];
                k=j;
            }
        }
        float f1=1e6; //f1 suficientemente grande
        for(int j=k; j<I.size();j++){
            vector<int> I1;
            int suma1=0;
            for(int v=0;v<j+1;v++){
                suma1=suma1+matriz[1][I[v]-1];
                if(v!=k){
                    I1.push_back( I[v] );
                }
            }
            int t1=t;
            vector<int> sigma1=secuenciar(t1,I1);
            vector<int> I2;
            for(int v=j+1;v<I.size();v++){
                    I2.push_back( I[v] );
                }
            int t2=t+suma1;
            vector<int> sigma2=secuenciar(t2,I2);
            vector<int> sigma=sigma1;
            sigma.push_back(I[k]);

            sigma.insert(sigma.end(), sigma2.begin(),sigma2.end());

            /*
            cout<<"sigma1\t "; for(int i =0; i< sigma1.size(); i++) cout<<sigma1[i]<<"\t"; cout<<endl;
            cout<<"tk "; cout<<I[k]<<"\t"; cout<<endl;
            cout<<"sigma2\t"; for(int i =0; i< sigma2.size(); i++) cout<<sigma2[i]<<"\t"; cout<<endl;
            cout<<"sigma "; for(int i =0; i< sigma.size(); i++) cout<<sigma[i]<<"\t"; cout<<endl;
            */
            //Mirar si se mejora la f.o.
            int valorFO = 0;
            int tiempoAcumulado =0;

            for(int i = 0; i< sigma.size(); i++){
                tiempoAcumulado += matriz[1][sigma[i]-1];
```

```
67
68                  int diferencia = tiempoAcumulado - (matriz[2][sigma[i]-1] - sumaLema);
69                  if(diferencia > 0){
70                      valorFO += diferencia;
71                  }
72              }
73              if( valorFO < f1){
74                  f1 = valorFO;
75                  sigmax = sigma;
76              }
77          }
78          cout<<".----------"<<endl;
79          //for(int i =0; i< I.size(); i++) cout<<I[i]<<"\t"; cout<<endl;
80
81          cout<<"FO = "<< f1<<endl;
82
83          for(int i =0; i< I.size(); i++) cout<<sigmax[i]<<"\t"; cout<<endl;
84          cout<<".----------"<<endl;
85          return sigmax;
86      }
87  }
88  int main(void)
89  {
90      int filas, columnas;
91      const char* nombre_fichero1 = "30trabajos.txt"; //nombre del fichero de los datos
92      ifstream fichero1 (nombre_fichero1);
93      fichero1>>filas;
94      fichero1>>columnas;
95      vector < vector<int> > matriz2(filas);
96      matriz.resize(filas);
97      for(int i = 0; i<filas; i++) matriz[i].resize(columnas);
98      for(int i = 0; i<filas; i++) matriz2[i].resize(columnas);
99      for (int i=0; i<filas; i++)
100     {
101         for (int j=0; j<columnas; j++)
102         {
103             fichero1>>matriz[i][j];
104         }
105     }
106     fichero1.close();
107     for (int i=0; i<filas; i++)
108     {
109         for (int j=0; j<columnas; j++)
110         {
111             matriz2[i][j]=matriz[i][j];
112         }
113     }
114     //Ordeno la matriz en función de la fecha de entrega.
115     float aux, auxi, auxil;
116     int pos;
117     for(int j = 0; j < columnas; j++){
118         pos=j;
119         aux=matriz2[0][j];
120         auxi=matriz2[1][j];
121         auxil=matriz2[2][j];
122         while((pos>0)&&(matriz2[2][pos-1]>auxil)){
123             matriz2[0][pos]=matriz2[0][pos-1];
124             matriz2[1][pos]=matriz2[1][pos-1];
125             matriz2[2][pos]=matriz2[2][pos-1];
126             pos--;
127         }
128         matriz2[0][pos]=aux;
129         matriz2[1][pos]=auxi;
130         matriz2[2][pos]=auxil;
131     }
132     //Tareas que no entran en programación dinámica al ser realizadas en orden.
```

```
133     int suma,total;
134     suma=0;
135     total=0;
136     vector<int> s1,s2;
137     for(int j = 0; j < 1; j++){
138         if(matriz2[2][j]<=matriz2[2][j+1] && matriz2[1][j]<=matriz2[1][j+1]){
139             s1.insert(s1.end(),matriz2[0][j]);
140             suma=suma+matriz2[1][j];
141             total=total+1;
142         }
143     }
144 
145     for(int j = 1; j < columnas; j++){
146         if(matriz2[2][j]<=matriz2[2][j+1] && matriz2[1][j]<=matriz2[1][j+1]
147            && existeEnVector(s1, matriz2[0][j-1])){
148             s1.insert(s1.end(),matriz2[0][j]);
149             suma=suma+matriz2[1][j];
150             total=total+1;
151         }
152     }
153     cout <<"Los trabajos realizados en primer lugar (por el lema) son: " << endl;
154     for (size_t i = 0; i < s1.size(); i++) {
155         cout << s1[i] << endl;
156     }
157     cout<<".----------"<<endl;
158     //Trabajamos con el nuevo tiempo y las tareas restantes
159     sumaLema = suma;
160     int num;
161     num=0; //numero de tareas en programacion dinamica
162     for(int j = total; j < columnas; j++){
163           num=num+1;
164     }
165 //Creo la nueva matriz solo con las tareas que me interesan
166     for(int j = total; j < columnas; j++){
167     cout<<matriz2[0][j]<<" "<<matriz2[1][j]<<" "<<matriz2[2][j]<<endl;
168     }
169     cout<<".----------"<<endl;
170     vector<int> trabajosOrdenados2;
171     for(int j = total; j < columnas; j++){
172         trabajosOrdenados2.insert(trabajosOrdenados2.end(),matriz2[0][j]);
173     }
174     secuenciar(0,trabajosOrdenados2);
175     system("PAUSE"); //para que no se cierre al terminar el programa
176     return 0; //Debe devolver algo el main, lo que e de la gana
177 }
178 
```