UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA MATEMÁTICA

ALGORITMOS PARA VARIANTES DEL PROBLEMA DE MINIMIZAR LA SUMA DE LOS K MAYORES RETRASOS DE TRABAJOS EN AGENDAMIENTOS EN UNA MÁQUINA

TESIS PARA OPTAR AL GRADO DE MAGÍSTER
EN CIENCIAS DE LA INGENIERÍA, MENCIÓN MATEMÁTICAS APLICADAS

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL MATEMÁTICO

RICARDO EMMANUEL ARANCIBIA CASTILLO

PROFESOR GUÍA:
JOSÉ SOTO SAN MARTÍN

MIEMBROS DE LA COMISIÓN:
MARCOS KIWI KRAUSKOPF
JOSÉ CORREA HAEUSSLER
VÍCTOR VERDUGO SILVA

Este trabajo ha sido parcialmente financiado por ANID/CONICYT FONDECYT Regular 1181180, CMM ANID PIA AFB170001, CMM ANID BASAL ACE210010 y CMM ANID BASAL FB210005

SANTIAGO DE CHILE
2021

RESUMEN DE LA MEMORIA PARA OPTAR
AL TÍTULO DE INGENIERO CIVIL MATEMÁTICO
POR: RICARDO EMMANUEL ARANCIBIA CASTILLO
FECHA: 2021
PROF. GUÍA: JOSÉ SOTO SAN MARTÍN

ALGORITMOS PARA VARIANTES DEL PROBLEMA DE MINIMIZAR LA SUMA DE LOS K MAYORES RETRASOS DE TRABAJOS EN AGENDAMIENTOS EN UNA MÁQUINA

El problema de minimizar *k-sum lateness* en una máquina es una generalización de los problemas de minimizar *maximum lateness* y *total lateness*, casos que se recuperan cuando $k = 1$ y $k = n$ respectivamente, y consiste en encontrar un agendamiento de n trabajos en una máquina, que minimice la suma de los k mayores *lateness*. Podemos resolver el problema de minimizar *maximum lateness* en tiempo polinomial mediante una regla de despacho simple que prioriza los trabajos de menor a mayor *deadline*, mientras que para resolver el problema de minimizar *total lateness* se priorizan los trabajos de acuerdo a sus tiempos de proceso, de menor a mayor.

A diferencia de los casos particulares anteriores, para el problema general del *k-sum lateness* mostramos que no existen reglas de despacho simples que permitan encontrar un agendamiento óptimo priorizando los trabajos ni según sus tiempos de proceso ni según sus *deadlines*. En esta tesis realizamos un estudio robusto del problema de *k-sum lateness* y presentamos algoritmos polinomiales para tres variantes importantes del problema. Cuando k es una constante fija, mostramos que el problema puede ser resuelto en tiempo $\mathcal{O}(n^{2k-1})$ extendiendo el trabajo de Woeginger [12] para *k-sum tardiness*. Cuando hay a lo más una cantidad P constante de tiempos de proceso distintos, reformulamos el problema de *k-sum lateness* como uno de optimización a dos niveles que puede ser resuelto en tiempo $\mathcal{O}(n^{2P+5})$. Finalmente, cuando hay a lo más una cantidad D constante de *deadlines* distintos, mostramos una serie de resultados estructurales de agendamientos óptimos que nos permiten resolver el problema en tiempo $\mathcal{O}(n^{3D+1}D)$.

A mi madre. La persona que más admiro.

Agradecimientos

Quiero agradecer en primer lugar a los profesores integrantes de mi comisión Marcos Kiwi, José Correa y Victor Verdugo por aceptar ser parte de este proceso, su buena disposición y por ayudarme a mejorar la escritura de este trabajo. Especialmente le quiero agradecer a mi profesor guía, José Soto. Para mí ha sido un honor trabajar con usted, y le agradezco profundamente por la paciencia, disponibilidad y ayudarme a encontrar caminos cuando estuve bien atascado. No podría haber elegido un mejor profesor guía. Agradezco también el financiamiento otorgado por ANID/CONICYT FONDECYT Regular 1181180, CMM ANID PIA AFB170001, CMM ANID BASAL ACE210010 y CMM ANID BASAL FB210005.

Me ha costado sentarme a escribir estos agradecimientos, porque ha sido un viaje bastante intenso. Lleno de alegrías, frustraciones, logros y fracasos, además de desafíos que nunca pensé podría sortear. Entender matemática al nivel en el que se enseña en el DIM ha sido maravilloso para mí, y nunca voy a olvidar las sensaciones que siempre que me provocó entender conceptos muy abstractos, o entender de forma profunda demostraciones de Teoremas hermosos. Tampoco olvidaré nunca el privilegio de poder compartir ideas con compañeros de carrera y con los amigos que hice en el DIM, que espero me duren toda la vida.

Creo que fui bastante afortunado de conocer tanta gente maravillosa en mi paso por la Universidad y por el departamento. Muchas gracias a Sebita y JP por ser de mis grandes amigos, por su apoyo y cariño. Al Daniel por la amistad, apoyo y por la motivación infinita que salen de nuestros proyectos. A mi mejor amiga, Claudis, a quien quiero mucho y agradezco por siempre estar ahí en las buenas y en las malas. Al Nacho, que ha sido mi amigo desde la infancia y fue clave para motivarme a seguir adelante con la carrera. Al Bolton, por las risas y los buenos momentos desde los tiempos del MBB. A Calisto, Pedro y Piero, por su amistad, por ser de las mejores personas que he conocido y por siempre disfrutar buenas conversaciones con ellos. A mis amigos franceses Pierre y Guillaume, que siempre fueron muy amables conmigo. A mis amigos de plan común Maxi, Eva, Tito y en general a los Traslov's Unicorns, con quienes me reí y me sigo riendo mucho. También le agradezco por su buena onda y cariño a la Vale, Panchito Venegas, Rodrigo L., Pablito, Zelada, Yipi, Panchito Sanhueza y Tabi. Le quiero agradecer además al profesor Patricio Felmer por darme la oportunidad de ser su auxiliar con la metodología de Resolución de Problemas, puesto que enseñar fue sin duda de las cosas más hermosas que hice en la Universidad, que marcó un antes y un después en la carrera para mí. Y por supuesto a los funcionarios del DIM, Karen, Don Óscar, Natacha, Silvia, Eterin, Cucky, Luis Mella, Juanito y Gladys, cuya pega es esencial para el departamento.

Agradezco también a Ignacio Cisternas, mi jefe, quien hace ya más de dos años me contrató como Data Scientist en Innspiral y ahora en Temis. Eres un tremendo líder y he aprendido muchísimo de ti. También le agradezco al resto del equipo de desarrollo, Pancho y Kike, de quienes también he aprendido muchísimo, tanto profesional como personalmente.

Le quiero agradecer infinito también a mi hermosa novia Constanza Eslava, por su amor, comprensión, apoyo y por estar siempre ahí para mí, en las buenas y en las malas. Te amo muchísimo.

Finalmente, le quiero agradecer a mi familia. Le quiero agradecer a mi Tía Anita, quien fue mi profesora desde segundo básico. Por confiar en mí, en mis capacidades y por involucrarse mucho más allá de lo que cualquier profesora se involucraría en la vida de un niño que la estaba pasando bastante mal. Tuve la fortuna de que la vida me regaló una segunda madre, además de la madre excelente que ya tenía. Le agradezco a mis tíos Nene y Pilo por siempre cuidarme, quererme y motivarme a aprender más. A su hija Chimu, que también fue mi primera alumna y que hoy es una gran profesional. A mi hermano Alejandro, a quien quiero mucho y confío tendrá un futuro hermoso con su familia, y que seguirá dando todo por salir adelante con su hijo Emilio. A la pareja de mi madre Claudio y su hijo Joao, quien es mi nuevo hermano, por todo lo que han aportado en nuestra vida y por ser excelentes personas. A mi hermanito Benjamín, que quienes me conocen saben es mi orgullo y uno de los principales motores que tuve para nunca rendirme durante la carrera y seguir esforzándome al máximo en la pega. Y finalmente a mi madre Jacqueline Castillo, a quien está dedicada esta tesis, y que es la persona que más admiro. Por su inteligencia, por su coraje para ser madre soltera y sacar adelante a sus dos hijos sin que nunca nos faltase nada, por su fortaleza y sabiduría infinita. Sin usted y todo su sacrificio nada de esto sería posible. Los amo infinito.

Tabla de Contenido

Índice de Ilustraciones

Capítulo 1

Introducción: El k-sum lateness problem

Dos problemas clásicos en Investigación de Operaciones y Ciencias de la Computación son el de agendar trabajos, cada uno con un *tiempo de proceso* y un *deadline* dado, en una sola *máquina* de modo tal de minimizar el retraso máximo (*maximum lateness*) o el retraso total (*total lateness*), siendo el *lateness* de un trabajo su *tiempo de completación* menos su *deadline*.

Cabe notar que estas cantidades pueden ser negativas cuando el tiempo de completación es menor al *deadline*. Existen algoritmos polinomiales que solucionan ambos problemas, que requieren ordenar dichos trabajos según *reglas de despacho*. Para el problema de *maximum lateness* se puede encontrar un agendamiento óptimo ordenando los trabajos de manera creciente con su *deadline*, mientras que para el problema de *total lateness*, se puede encontrar un agendamiento óptimo ordenando los trabajos de manera creciente con su tiempo de proceso.

En esta tesis estudiaremos una generalización de ambos problemas, en la cual para un parámetro k fijo, se desea minimizar la suma de los k mayores retrasos. Llamamos a este nuevo problema *k-sum lateness*. Notamos que cuando $k = 1$ se recupera el problema de *maximum lateness* y cuando $k = n$ se recupera el de *total lateness*. Denotaremos por D al número de *deadlines* diferentes y por P al número de tiempos de proceso diferentes. Los resultados principales de esta tesis son los siguientes.

Teorema 1.1 *El problema de minimizar* k-sum lateness *para* n *trabajos se puede resolver en tiempo polinomial* $\mathcal{O}(n^{2k-1})$*, para cada* $k \geq 2$ *fijo.*

Teorema 1.2 *El problema de minimizar* k-sum lateness *para* n *trabajos se puede resolver en tiempo polinomial* $\mathcal{O}(n^{3D+1}D)$*, para cada* D *fijo.*

Teorema 1.3 *El problema de minimizar* k-sum lateness *para* n *trabajos se puede resolver en tiempo polinomial* $\mathcal{O}(n^{2P+5})$*, para cada* P *fijo.*

Una aplicación interesante del problema en estudio consiste en la *planificación de entrega de trabajos*, donde para cada trabajo j podemos interpretar $(-d_j)$ como su *tiempo de envío*. Específicamente, consideramos una colección de trabajos con tiempos de proceso positivos y *deadlines* negativos, a agendar en una máquina. En este caso el *lateness* de cada trabajo sería la suma de su tiempo de completación con el tiempo de envío. Así, con los algoritmos polinomiales presentados en este trabajo de tesis se puede dar una respuesta al caso de estudio de minimizar la suma de los k *tiempos de entrega*, en las variantes mencionadas anteriormente.

1.1. Notación y preliminares

A lo largo de esta tesis utilizaremos la notación $J = \{1, ..., n\}$ ó $[n]$ para denotar la colección de trabajos a agendar en una única máquina. Para cada trabajo $j \in J$ denotamos por $d_j \in \mathbb{R}$ a su *deadline*, y por $p_j \geq 0$ a su tiempo de proceso. A una permutación S de J le llamamos *agendamiento*. Usaremos la notación $i \leq_S j$ si i es igual a j o si i aparece antes que j en S. Dados $A, B \subseteq [n]$ denotamos $A \leq_S B$ si $a \leq_S b$ para todo $a \in A$, $b \in B$. Se utiliza la notación de intervalos con S como subíndice para denotar colecciones de trabajos consecutivos en el agendamiento. Por ejemplo, $[i, j)_S$ son todos los trabajos consecutivos entre los trabajos i y j en el agendamiento S, incluyendo i y excluyendo j. Cada trabajo j del agendamiento se procesa en la máquina por p_j unidades de tiempo sin interrumpir, y una vez terminado se comienza a procesar el siguiente trabajo en el agendamiento de forma inmediata. Denotamos por $C(j, S) = \sum_{i \leq_S j} p_i$ al *tiempo de completación* del trabajo j en S. A continuación se presenta un agendamiento de tres trabajos con tiempos de proceso $p_1 = 3$, $p_2 = 4$, $p_3 = 5$ y *deadlines* $d_1 = 5$, $d_2 = 10$, $d_3 = 15$, en la *máquina*, utilizando un modelo de *Diagrama Gantt*, en donde los números en la parte inferior de la figura denotarán los tiempos de completación. También utilizamos alturas sobre los trabajos para ilustrar sus *deadlines* cuando sea necesario.

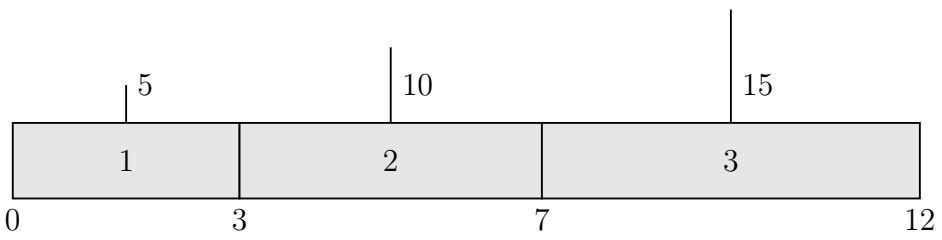

Figura 1.1: Diagrama de Gantt

Dependiendo de la posición de un trabajo j en un agendamiento S dado, se definen las siguientes cantidades.

Definición 1.4 (lateness) *Se define el* lateness *de un trabajo j en el agendamiento S como*

$$L(j, S) := C(j, S) - d_j.$$

De la definición anterior notamos que el *lateness* de un trabajo podría ser negativo. Otra función objetivo relacionada que es siempre no-negativa es el *tardiness*. Si bien no la estudiaremos en detalle, varios trabajos previos que citaremos se basan en ella.

Definición 1.5 (tardiness) *Se define el* tardiness *de un trabajo j en el agendamiento S como*

$$T(j,S) := (C(j,S) - d_j)_+ = \text{máx}\{C(j,S) - d_j, 0\}.$$

Consideremos las siguientes funciones objetivo.

Definición 1.6 (maximum lateness) *Para un agendamiento S dado, se define su* maximum lateness *como*

$$L_{\text{máx}}(S) := \max_{j \in S} L(j,S).$$

Definición 1.7 (total lateness) *Para un agendamiento S dado, se define su* total lateness *como*

$$\sigma_n(S) := \sum_{j \in S} L(j,S).$$

El objeto de estudio central de la tesis es la minimización de la siguiente función objetivo.

Definición 1.8 *Denotamos por $\sigma_k(S)$ a la suma de los k mayores* lateness *de trabajos en S. Equivalentemente,*

$$\sigma_k(S) = \max_{X \subseteq J\colon |X|=k} \sum_{j \in X} L(j,S).$$

Al conjunto de los k trabajos con mayor lateness *le llamamos k-*set *y lo denotamos por $S[k]$. Si dos trabajos tienen el mismo* lateness*, será más tardío aquel con una posición mayor en el agendamiento. Además, denotaremos por $S[k]^c$ a los trabajos de S que no están en $S[k]$.*

Dependiendo de la función objetivo que se quiera minimizar, para algunas de ellas existen reglas de ordenamiento que llamaremos *reglas de despacho*, que consisten en ordenar los trabajos de acuerdo a cierta propiedad de estos generando un agendamiento.

1.1.1. El caso $k = 1$: Maximum Lateness

Es posible generar un agendamiento óptimo S que minimiza el valor de $L_{\text{máx}}$ utilizando la regla de despacho que consiste en ordenar los trabajos en orden creciente de *deadlines*. Esta regla se denomina *EDD* por las siglas *Earliest Due Date first* [4]. En adelante podemos suponer que para cada colección de trabajos J, la regla *EDD* entrega *un único* agendamiento si entre dos trabajos con el mismo *deadline*, agendamos primero el trabajo con menor numeración. Por ejemplo, en la instancia

	1	2	3
d	2	3	2
p	4	5	6

el agendamiento que entrega *EDD* es $(1,3,2)$. De esta forma, podemos hablar de *el orden dado por* EDD y para simplificar la exposición y cuando no haya confusión posible, identificaremos *EDD* con el orden que da la misma regla.

A continuación se presenta por completitud la demostración de que *EDD* genera un *agendamiento* óptimo cuando se está minimizando $L_{\text{máx}}$.

Teorema 1.9 *([4], reformulado) La* regla de despacho EDD *genera un agendamiento óptimo para el problema de minimizar* maximum lateness $L_{\text{máx}}$.

Demostración. Sea S un agendamiento óptimo para el problema de minimizar $L_{\text{máx}}$. Probaremos que a partir de S podemos construir un agendamiento óptimo S' intercambiando trabajos adyacentes que no estén ordenados según la regla *EDD* como en la Figura 1.2.

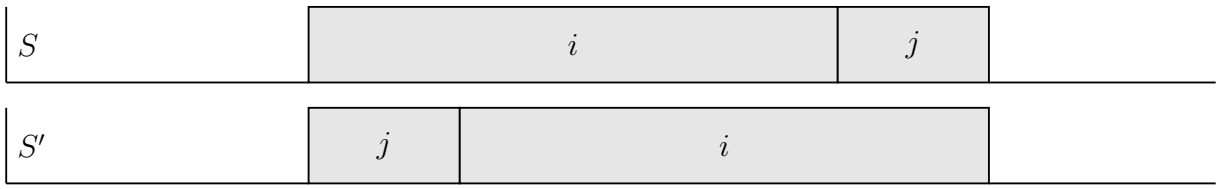

Figura 1.2: Intercambio trabajos consecutivos.

Si S no está ordenado según *EDD*, sean i, j trabajos consecutivos, con $i \leq_S j$, tales que $d_i > d_j$. Llamamos S' al agendamiento obtenido desde S intercambiando los trabajos i y j. Notemos primero que

$$\forall w \notin \{i, j\},\ L(w, S) = L(w, S'), \tag{1.1}$$

puesto que se mantienen en su misma posición y sus tiempos de completación cumplen que $C(w, S) = C(w, S')$. Mostraremos ahora que

$$\text{máx}\{L(i, S'), L(j, S')\} \leq \text{máx}\{L(i, S), L(j, S)\}. \tag{1.2}$$

En efecto, como $C(i, S) \leq C(j, S)$ y $-d_i < -d_j$, se deduce que $L(i, S) < L(j, S)$ y luego $\text{máx}\{L(i, S), L(j, S)\} = L(j, S)$.

Solo resta probar que $\text{máx}\{L(i, S'), L(j, S')\} \leq L(j, S)$. Como $C(i, S') = C(j, S)$ y $-d_i < -d_j$, tenemos que $L(i, S') < L(j, S)$. Para ver que $L(j, S') \leq L(j, S)$, notemos primero que $C(j, S') = C(i, S) - p_i + p_j$. Mas aún, como los tiempos de proceso son no negativos, tendremos que $C(j, S') \leq C(i, S) + p_j = C(j, S)$. Luego, $L(j, S') = C(j, S') - d_j \leq C(j, S) - d_j$. Es decir, $L(j, S') \leq L(j, S)$, concluyendo la demostración de (1.2).

Luego, de la igualdad (1.1) y de la desigualdad (1.2), tendremos que

$$L_{\text{máx}}(S') \leq L_{\text{máx}}(S). \tag{1.3}$$

Es decir, el agendamiento S' también será óptimo para el problema de *maximum lateness*. Se repite el procedimiento anterior hasta que S' esté ordenado según *EDD*. □

1.1.2. El caso $k = n$: Total Lateness

El problema de minimizar *total lateness* es equivalente a minimizar el *tiempo total de completación* $\sum_{i\in[n]} C(i,S)$. En efecto, basta notar que $\sum_{i\in[n]} L(i,S) = \sum_{i\in[n]} C(i,S) - d_i = \sum_{i\in[n]} C(i,S) - \sum_{i\in[n]} d_i$ y que $\sum_{i\in[n]} d_i$ es una constante, con lo cual los problemas son equivalentes. Por otro lado, como la regla de despacho *Shortest Processing Time first*, que llamaremos *SPT*, encuentra un agendamiento óptimo para el problema de minimizar el tiempo total de completación [11], también lo hará para *total lateness*.

A continuación se presenta una demostración de que *SPT* encuentra un agendamiento óptimo para *total lateness*, pues ilustra un argumento de intercambio de trabajos que utilizaremos frecuentemente a lo largo de la tesis.

Teorema 1.10 *([11], reformulado) La* regla de despacho SPT *genera un agendamiento óptimo para el problema de minimizar el* total lateness σ_n.

Demostración. Sea S un agendamiento óptimo para el problema de minimizar *total lateness* σ_n. Por contradicción, suponemos que S no está ordenado según la regla *SPT*. Entonces existen trabajos i, j consecutivos, con $i \leq_S j$, y tales que $p_i > p_j$. Llamamos S' al agendamiento obtenido de S intercambiando i con j (Figura 1.2). Veremos que $\sigma_n(S') - \sigma_n(S) = p_j - p_i < 0$, lo cual será una contradicción con el hecho de que S es óptimo. En efecto,

$$L(i,S') - L(i,S) = C(j,S) - C(i,S) = p_j, \tag{1.4}$$

donde (1.4) es puesto que $C(i,S') = C(j,S)$. Por otro lado, tenemos que,

$$\begin{aligned} L(j,S') - L(j,S) &= C(j,S') - C(j,S) \\ &= C(i,S) - p_i + p_j - C(j,S) \\ &= -p_i. \end{aligned}$$

Notamos ahora que

$$\forall w \notin \{i,j\},\ L(w,S) = L(w,S'), \tag{1.5}$$

por tanto,

$$\begin{aligned} \sigma_n(S') - \sigma_n(S) &= \sum_{h=1}^{n} L(h,S') - \sum_{h=1}^{n} L(h,S) \\ &= L(i,S') + L(j,S') - (L(i,S) + L(j,S)) \\ &= p_j - p_i < 0, \end{aligned}$$

con lo que queda demostrado el resultado. □

1.1.3. El caso general de k-sum lateness

Una *regla de despacho simple* es una regla que asigna a cada trabajo j una prioridad como función de sus parámetros (p_j, d_j) y que devuelve un agendamiento ordenando los trabajos de

mayor a menor prioridad. Como ya vimos, existen reglas de despacho simples que resuelven a optimalidad los casos $k = 1$ (*maximum lateness*, usando como prioridad $-d_j$) y $k = n$ (*total lateness*, usando como prioridad $-p_j$).

Resulta natural entonces preguntarse si existe una regla de despacho simple para resolver el caso general. El siguiente lema nos muestra que no existe dicha regla ni siquiera para el caso $k = 2$.

Lema 1.11 *No existe regla de despacho simple para minimizar* k-sum lateness.

Demostración. En efecto, basta considerar la instancia con $n = 3$ trabajos y $k = 2$, con los siguientes *deadlines* y tiempos de proceso.

	1	2	3
d	7	7	8
p	10	10	9

En la instancia anterior, el agendamiento generado tanto por la regla *EDD* como por *SPT* tiene un valor de 34, mientras el agendamiento óptimo S^* para el problema de 2-*sum lateness* está representado en la Figura 1.3. con $\sigma_2(S^*) = 33$. Notemos que este agendamiento

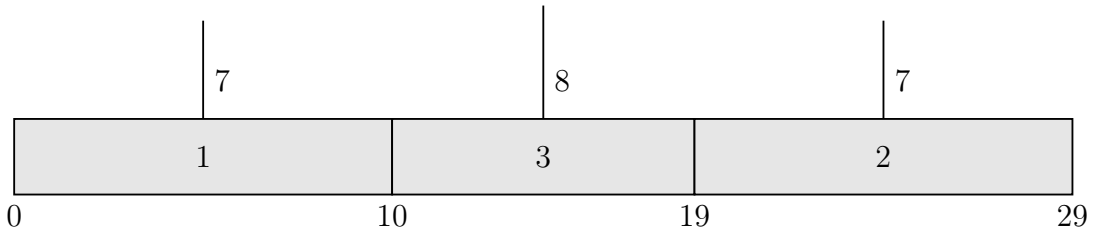

Figura 1.3: S^* óptimo para 2-*sum lateness* con trabajos 1 con $d_1 = 7$, $p_1 = 10$, 2 con $d_2 = 8$, $p_1 = 9$, 3 con $d_3 = 7$, $p_1 = 10$.

no viene de una regla de despacho simple, puesto que 1 y 2 tienen el mismo *deadline* y tiempo de proceso, pero están separados en S^*. □

El lema anterior sugiere que el problema de k-*sum lateness* es de mayor complejidad pese a que para los casos $k = 1$ y $k = n$ pueden ser resueltos en tiempo polinomial. De hecho, conjeturamos que el problema k-*sum lateness* es *NP-difícil* si k es parte de la entrada. En este trabajo de tesis se estudian variantes importantes del problema de k-*sum lateness* que admiten algoritmos polinomiales.

1.2. Organización de la tesis

En el Capítulo 2 estudiamos el problema de k-*sum lateness* cuando k es constante. Mostraremos que cuando k es pequeño, los agendamientos óptimos para este problema son similares a el agendamiento π_{EDD} generado por la regla *EDD*. En específico, probaremos que existen

agendamientos óptimos que se pueden generar a partir de π_{EDD} moviendo a lo más $k-1$ trabajos de posición. Con este resultado mostramos en el Teorema 2.5 que el problema puede ser resuelto en tiempo $\mathcal{O}(n^{2k-1})$.

En el Capítulo 3 estudiamos el problema de *k-sum lateness* cuando el número de *deadlines* distintos, que denotamos por D a lo largo de la tesis, es constante. La idea central es mostrar que existen agendamientos óptimos para el problema en los cuales si eliminamos los trabajos del *k-set*, los trabajos con igual *deadline* aparecen juntos. A las colecciones de trabajos con un único *deadline* y fuera del *k-set* las llamaremos *lagunas*, y probaremos que existe un agendamiento óptimo con a lo más D lagunas consecutivas distintas. Esto, sumado a otras ideas estructurales, nos permitirá encontrar un algoritmo con complejidad $\mathcal{O}(n^{3D+1}D)$ para resolver el problema.

En el Capítulo 4 estudiamos el caso particular del problema de *k-sum lateness* cuando el número de tiempos de procesos distintos, que llamaremos P, es constante. Mostraremos que el problema de *k-sum lateness* se puede reducir a resolver una cantidad polinomial de instancias del problema *total tardiness*. Este último problema en toda su generalidad es *NP-completo*, pero mostramos que cuando P es constante, podemos modificar un algoritmo de Lawler [7] para resolverlo en tiempo polinomial.

1.3. Trabajo relacionado y estudio bibliográfico

Existen reglas de despacho simples que encuentran un agendamiento óptimo para problemas, tales como *maximum lateness* y *total lateness*. En específico, vimos que la regla *EDD* da agendamientos óptimos para el problema de *maximum lateness* [4]. Mas aún, la regla *EDD* también encuentra agendamientos óptimos para otros problemas en una máquina, como por ejemplo para minimizar *maximum tardiness* [4]. Lawler [6] generaliza este resultado de dos formas. Primero para funciones generales, en donde cada trabajo tiene una función de costo $f_j(t)$ monótona, no-decreciente en el tiempo t en el que se completa, Lawler propone un algoritmo polinomial para minimizar el máximo de $f_j(C_j)$. Y segundo, generaliza el algoritmo para permitir restricciones de precedencias de trabajos.

Por otra parte, Smith [11] resuelve el caso de minimizar el tiempo total de completación utilizando la regla *SPT*, lo que inmediatamente dice que *total lateness* se puede resolver del mismo modo. Para funciones objetivo ligeramente más complicadas los problemas se vuelven *NP-completos*, como minimizar *total tardiness* [1], o minimizar el peso total de trabajos atrasados [5], que es *NP-difícil* incluso si todos los trabajos tienen igual *deadline*. Lawler [7] encuentra un algoritmo *pseudopolinomial* para resolver el problema de *total tardiness* con un algoritmo basado en un programa dinámico. Mas tarde, Lawler [8] encuentra un *esquema de aproximación a tiempo polinomial (PTAS)* modificando su algoritmo *pseudopolinomial* para el mismo problema.

El concepto de *k-sum optimization* fue introducido por Gupta y Punnen [2], introduciéndolo como generalización para *minsum problems* (MSP) y *minmax problems* (MMP), estos últimos también conocidos como problemas de *cuello de botella*. En dicho trabajo muestran que dada una familia de conjuntos de tamaño n fijo tales que el MSP (es decir, encontrar un conjunto factible de peso máximo) se puede resolver para cualquier función de peso en

tiempo polinomial, entonces se puede resolver el problema de encontrar el conjunto factible que minimice la suma de sus k mayores pesos en tiempo polinomial. Más tarde Punnen y Aneja [10] muestran que la condición de que todos los conjuntos tengan igual tamaño se puede relajar manteniendo el resultado. A lo largo de los años, Puerto, Rodríguez-Chía y Tamir [9] estudian diferentes formulaciones del problema de *k-sum optimization* en sus versiones continuas y enteras, de forma de poder resolver el problema mediante una minimización de un número relativamente pequeño de problemas de tipo *minsum*. En el Capítulo 4 utilizamos reformulaciones similares a estas para resolver el problema de *k-sum lateness* para el caso en que la cantidad de tiempos de proceso distintos es constante. Una función objetivo de tipo *k-sum* en agendamientos fue estudiada por Woeginger [12]. Sin embargo, en rigor esta función no cae dentro de la definición de las funciones objetivo dadas por Punnen, puesto que los pesos de los trabajos (*i.e*, sus *tardiness*) no son valores propios del trabajo sino que dependen del agendamiento considerado. Woeginger describe un algoritmo para resolver *k-sum tardiness* cuando k constante, resultado notable considerando que cuando $k = n$, el problema de *total tardiness* es *NP-completo* como fue mostrado por Du y Leung [1].

Capítulo 2

k-sum lateness con k constante

Woeginger [12] estudia el problema *k-sum tardiness* que busca minimizar la suma de los k mayores *tardiness* de un agendamiento. El resultado principal de Woeginger es un algoritmo para el problema de *k-sum tardiness*, con k fijo, de complejidad $\mathcal{O}(n^{2k-1})$, donde n representa el número de trabajos a agendar. Este algoritmo se basa en la idea de que existe un agendamiento óptimo para *k-sum tardiness* que es, en cierto sentido, cercano al orden que da *EDD*, que como vimos es óptimo para el caso $k = 1$.

En este capítulo mostramos que el esquema de Woeginger también puede ser aplicado para el problema de *k-sum lateness* y presentamos un algoritmo polinomial con complejidad $\mathcal{O}(n^{2k-1})$ para resolverlo.

Definición 2.1 *Sean π_1 y π_2 permutaciones de $[n]$. Decimos que π_2 es una permutación* m-adyacente *de π_1 si π_2 se puede obtener desde π_1 cambiando* a lo más *m trabajos de lugar. Es decir, existe una colección de m trabajos tales que al removerlos de ambos agendamientos, entonces ambas permutaciones coinciden.*

Lema 2.2 *Para cada* permutación *π, existen a lo más $\mathcal{O}(n^{2m})$ permutaciones* m-adyacentes *de π.*

Demostración. El resultado se sigue de que hay $\binom{n}{m} = \mathcal{O}(n^m)$ posibilidades para remover m elementos de π, y para cada elemento hay a lo más n posiciones diferentes para reinsertarlo. □

Woeginger muestra que los agendamientos óptimos para el problema de *k-sum tardiness* son $(k-1)$-adyacentes al agendamiento generado por la *regla de despacho EDD*. En el siguiente Teorema mostramos que la misma propiedad se tiene para el problema de *k-sum lateness*.

Teorema 2.3 *Para cada S agendamiento óptimo, existe S^* con $\sigma_k(S^*) \leq \sigma_k(S)$ y tal que S^* es* $(k-1)$-adyacente *al agendamiento dado por* EDD.

Demostración. Sea S un agendamiento óptimo para *k-sum lateness*. Denotamos por $i_1, ..., i_k$ los k trabajos mas tardíos en S, es decir, los elementos de $S[k]$, y que cumplen

$$L(i_j, S) \geq L(i_{j+1}, S).$$

Consideremos la instancia auxiliar $\tilde{S}$ que se obtiene manteniendo el ordenamiento de S y modificando los *deadlines* de los trabajos como sigue

$$\tilde{d}_j = d_j + L(j, S) \quad \forall j \in S[k]. \tag{2.1}$$
$$\tilde{d}_j = d_j + L(i_k, S) \quad \forall j \notin S[k]. \tag{2.2}$$

Es decir, los trabajos fuera del *k-set* $S[k]$ suman a su *deadline* el *lateness* del k-ésimo trabajo mas tardío $L(i_k, S)$, mientras que los trabajos del *k-set* suman a su *deadline* su *lateness* respectivo en S. Woeginger realiza esta misma transformación sumando sus *tardiness* en lugar de sus *lateness*, pero notamos que en nuestro caso los *deadlines* pueden disminuir, quedar igual o aumentar, puesto que los *lateness*, a diferencia de los *tardiness*, permiten valores negativos.

Para cualquier agendamiento T y trabajo j, usamos la notación $\tilde{C}(j, T)$ y $\tilde{L}(j, T)$ para denotar el tiempo de completación y *lateness* del trabajo j en T respectivamente, usando los nuevos *deadlines*. Asimismo, llamamos $\tilde{\sigma}_k(T)$ a la suma de las k mayores lateness de T bajo la nueva instancia. Notamos que como los trabajos no cambian de tiempo de proceso, en ambas instancias se tendrá que para todo $j \in [n]$, $\tilde{C}(j, T) = C(j, T)$.

Volvamos al agendamiento S considerado originalmente. Tenemos que para todo $j \in S[k]$

$$\tilde{L}(j, S) = \tilde{C}(j, S) - \tilde{d}_j = C(j, S) - (d_j + L(j, S)) = 0,$$

y además, para los trabajos fuera del *k-set*, $j \notin S[k]$,

$$\tilde{L}(j, S) = \tilde{C}(j, S) - \tilde{d}_j = C(j, S) - (d_j + L(i_k, S)) = L(j, S) - L(i_k, S) \leq 0,$$

puesto que i_k es el k-ésimo trabajo más tardío, y por tanto, tendrá mayor *lateness* que $j \notin S[k]$.

Luego, tendremos que

$$\tilde{L}_{\text{máx}}(S) = \max_{j \in [n]} \tilde{L}(j, S) \leq 0.$$

Llamamos EDD* al agendamiento que se obtiene al ordenar los trabajos de acuerdo a *EDD* usando los *deadlines* modificados. Como *EDD* es una regla óptima para minimizar el trabajo de mayor *lateness*, en la instancia modificada se tendrá que

$$\tilde{L}_{\text{máx}}(\text{EDD}^*) \leq \tilde{L}_{\text{máx}}(S) \leq 0.$$

Así, $\tilde{L}_{\text{máx}}(\text{EDD}^*) \leq 0$. En otras palabras, para todo $j \in [n]$, $\tilde{L}(j, \text{EDD}^*) \leq 0$. De esto se sigue directamente que $\tilde{\sigma}_k(\text{EDD}^*) \leq 0$ puesto que lo anterior nos dice que σ_k será suma de

términos negativos. Luego, analizando σ_k (*i.e*, con *deadlines* originales) de EDD*

$$\begin{aligned}
\sigma_k(\text{EDD}^*) &= \sum_{j\in \text{EDD}^*[k]} L(j, EDD^*) \\
&= \sum_{j\in \text{EDD}^*[k]} (C(j, \text{EDD}^*) - d_j) \\
&= \sum_{j\in \text{EDD}^*[k]} (\tilde{C}(j, \text{EDD}^*) - \tilde{d}_j + \tilde{d}_j - d_j) \\
&= \sum_{j\in \text{EDD}^*[k]} \tilde{L}(j, \text{EDD}^*) + \sum_{j\in \text{EDD}^*[k]} (\tilde{d}_j - d_j) \\
&= \sum_{j\in \text{EDD}^*[k]} \tilde{L}(j, \text{EDD}^*) + \sum_{j\in \text{EDD}^*[k]\cap S[k]} (\tilde{d}_j - d_j) + \sum_{j\in \text{EDD}^*[k]\setminus S[k]} (\tilde{d}_j - d_j).
\end{aligned}$$

Como para todo $j \in [n]$, $\tilde{L}(j, \text{EDD}^*) \leq 0$, tendremos que

$$\sum_{j\in \text{EDD}^*[k]} \tilde{L}(j, \text{EDD}^*) \leq 0.$$

Por otro lado, para la segunda sumatoria

$$\sum_{j\in \text{EDD}^*[k]\cap S[k]} (\tilde{d}_j - d_j) = \sum_{j\in \text{EDD}^*[k]\cap S[k]} L(j, S).$$

Y para la tercera sumatoria tendremos que

$$\begin{aligned}
\sum_{j\in \text{EDD}^*[k]\setminus S[k]} (\tilde{d}_j - d_j) &= |\text{EDD}^*[k] \setminus S[k]| L(i_k, S) \\
&= |S[k] \setminus \text{EDD}^*[k]| L(i_k, S) \\
&\leq \sum_{j\in S[k]\setminus \text{EDD}^*[k]} L(j, S),
\end{aligned}$$

donde se usó que $|\text{EDD}^*[k]| = |S[k]| = k$ y que i_k es el trabajo con menor *lateness* del *k-set* $S[k]$. Veamos ahora que EDD* también es óptimo para el problema

$$\begin{aligned}
\sigma_k(\text{EDD}^*) &\leq \sum_{j\in \text{EDD}^*[k]\cap S[k]} L(j, S) + \sum_{j\in S[k]\setminus \text{EDD}^*[k]} L(j, S) \\
&= \sum_{j\in S[k]} L(j, S) \\
&= \sigma_k(S).
\end{aligned}$$

Finalmente, se concluye el resultado notando que como los *deadlines* de los trabajos $j \in [n] \setminus \{i_1, ..., i_{k-1}\}$ fueron modificados por la misma constante, tendremos que EDD* será una permutación $(k-1)$-*adyacente* de *EDD*. En efecto, como los *deadlines* de estos trabajos fueron modificados por la misma constante $L(i_k, S)$, dichos *deadlines* estarán ordenados de la misma forma que en la colección de *deadlines* sin modificar. Luego, la posición relativa de dichos trabajos en EDD* será la misma que en *EDD* sin los *deadlines* modificados. □

El resultado anterior sugiere como algoritmo revisar todas las permutaciones $(k-1)$-adyacentes al ordenamiento que da *EDD* y quedarse con el agendamiento con menor valor.

Algoritmo 1: *k-sum lateness*, primera versión.

Data: $J = [n]$ trabajos con tiempos de proceso y *deadlines* arbitrarios.
Result: S_{best} agendamiento óptimo para σ_k

```
S_best ← ∅.
val_best ← ∞.
Generar agendamiento usando EDD.
Generar conjunto P de todas las permutaciones (k − 1)-adyacentes de EDD.
for S ∈ P do
    Calcular σ_k(S).
    if σ_k(S) ≤ val_best then
        S_best ← S.
        val_best ← σ_k(S).
    end
end
```

Teorema 2.4 *El Algoritmo 1 genera un agendamiento óptimo para* k-sum lateness.

Demostración. En efecto, como por el Teorema 2.3 existen agendamientos óptimos que son permutaciones $(k-1)$-adyacentes de *EDD*, y el algoritmo 1 revisa todas estas permutaciones y se queda con la que tiene menor σ_k, tendremos que el algoritmo encuentra un agendamiento óptimo para el problema de *k-sum lateness*. □

A continuación usando el Algoritmo 1, mostramos que *k-sum lateness* se puede resolver en tiempo polinomial cuando $k \geq 2$ es constante.

Teorema 2.5 *El problema de minimizar* k-sum lateness *para* n *trabajos se puede resolver en tiempo polinomial* $\mathcal{O}(n^{2k-1})$*, para cada* $k \geq 2$ *fijo.*

Demostración. El agendamiento π que da *EDD* se puede calcular en tiempo $\mathcal{O}(n\log(n))$ ordenando los trabajos según *deadline*. Ahora, si denotamos por $\mathcal{P}_{k-1}(EDD)$ el conjunto de las permutaciones $(k-1)$-adyacentes de π, tendremos que $|\mathcal{P}_{k-1}(EDD)| = O(n^{2k-2})$ por el Lema 2.2. No es difícil probar que se puede generar el conjunto $\mathcal{P}_{k-1}(\text{EDD})$ en tiempo $\mathcal{O}(n^{2k-2})$. Ahora bien, calcular $\sigma_k(S)$ para cada $S \in \mathcal{P}$ es lineal. En efecto, basta considerar todos los *lateness* de S en un arreglo, y encontrar el k-ésimo *lateness* mas grande en tiempo $\mathcal{O}(n)$ utilizando el algoritmo *quickselect* [3], y luego usar su *lateness* para calcular $\sigma_k(S)$. □

Capítulo 3

k-sum lateness con D constante

En este Capítulo abordaremos el caso en que se tienen a lo más una cantidad D constante de *deadlines* distintos. En las primeras 3 secciones demostramos algunas propiedades generales de agendamientos óptimos para el problema de *k-sum lateness* que no requieren que la cantidad D sea constante. En la Sección 3.1 mostramos primero que se puede suponer sin pérdida de generalidad que todos los trabajos tienen distinto tiempo de proceso. Luego definimos un tipo especial de agendamientos, llamados *agendamientos crecientes*. Estos satisfacen que para cada par de trabajos i, j con $p_i < p_j$ y $d_i \leq d_j$, se tiene que i aparece antes que j en el agendamiento. Terminamos la Sección probando que existe un agendamiento óptimo para *k-sum lateness* que es creciente. En la Sección 3.2 estudiaremos la estructura de los trabajos fuera del *k-set*. Llamamos *lagunas* a los conjuntos de trabajos fuera del *k-set* con igual *deadline*. Probaremos que existe un óptimo creciente S cuyas lagunas son conjuntos de trabajos cuyos elementos son consecutivos en S. Como consecuencia de este estudio, al final de la Sección damos una demostración alternativa del teorema central del Capítulo anterior, mostrando que existen agendamientos óptimos S que son k-adyacentes al agendamiento dado por *EDD*. En la Sección 3.3 estudiamos la estructura interna de $S[k]$ para cualquier S óptimo creciente. Llamamos *pivote* asociado a un *deadline* δ al elemento de $S[k]$ con *deadline* δ que tiene el menor tiempo de proceso. Mostramos que todo óptimo creciente satisface una propiedad llamada *SPT* por pivote. Esta propiedad permite mostrar que si conocemos el conjunto X de trabajos que estarán en el *k-set* y sabemos la posición de sus (a lo más) D pivotes entonces el orden interno de $S[k]$ está completamente determinado. En la Sección 3.4 usamos los resultados anteriores para diseñar un algoritmo para resolver *k-sum lateness* para D constante en tiempo $\mathcal{O}(n^{3D+1}D)$.

3.1. Agendamientos crecientes con tiempos de proceso distintos

Primero probaremos que en el problema de *k-sum lateness* se puede suponer sin pérdida de generalidad que todos los trabajos tienen distinto tiempo de proceso y además positivo.

Lema 3.1 *Sea I una instancia para k-*sum lateness*. Los tiempos de proceso de los trabajos*

de I se pueden perturbar adecuadamente para obtener una instancia I' con todos los tiempos de proceso distintos y distintos de cero, y tal que cualquier agendamiento óptimo de I' es agendamiento óptimo de I.

Demostración. Consideremos una instancia I en la que todos los p_i y d_i son racionales. Escalando podemos suponer que todos los p_i y los d_i son enteros y luego todos los *lateness* son números enteros. Llamamos I' a la instancia donde para cada $i \in [n]$, $p'_i = p_i + 2^{-i}/n$. De esta manera todos los tiempos de procesos son distintos entre sí, y distintos de 0. Extendemos esta notación a L', σ'_k, el *lateness* y función objetivo asociados. Para un agendamiento S y un trabajo i, se tiene que $L'(i,S) - L(i,S) > 0$ y además,

$$L'(i,S) - L(i,S) = \sum_{h \leq_S i} 2^{-h}/n \leq \sum_{h=1}^{n} 2^{-h}/n < 1/n.$$

Lo anterior nos dice que ningún trabajo aumenta su *lateness* en mas de $1/n$. Con esto tendremos que $\lfloor L'(i,S) \rfloor = L(i,S)$. Además, si $L(i,S) < L(j,S)$, entonces $1 \leq L(j,S) - L(i,S)$. Luego, $L'(i,S) = L(i,S) + \sum_{h \leq_S i} 2^{-h}/n \leq L(j,S) - 1 + \sum_{h \leq_S i} 2^{-h}/n < L(j,S) < L'(j,S)$. Se concluye que los k trabajos más tardíos en S de acuerdo a los tiempos de proceso p'_i son también k trabajos más tardíos de acuerdo a los tiempos de proceso p_i y además, $\sigma_k(S) \leq \sigma'_k(S) < \sigma_k(S) + k/n \leq \sigma_k(S) + 1$, por lo que $\lfloor \sigma'_k(S) \rfloor = \sigma_k(S)$. De aquí se tiene que cualquier agendamiento S que minimice σ'_k también minimiza σ_k. □

En virtud del lema anterior, podemos suponer en adelante que todos los trabajos tienen tiempo de proceso diferentes y positivos. Consideremos ahora las siguientes definiciones.

Definición 3.2 *Diremos que i* es dominado por *j si $p_i < p_j$ y $d_i \leq d_j$. Por otro lado, si i no domina a j ni j domina a i, diremos que ambos trabajos son* incomparables*.*

Definición 3.3 *Diremos que un* agendamiento *S es* creciente *si para cada par de trabajos $i, j \in [n]$, donde i es dominado por j, se tiene que $i \leq_S j$.*

Recordemos además que se utiliza la notación de intervalos con el agendamiento S como subíndice para denotar colecciones de trabajos consecutivos en el agendamiento. Por ejemplo, $[i,j)_S$ son todos los trabajos consecutivos entre los trabajos i y j en el *agendamiento* S, incluyendo i y excluyendo j.

Definición 3.4 *Sea S un* agendamiento *y sean $i, j \in J$ trabajos con $i \leq_S j$. La operación de intercambiar las posiciones de los trabajos i y j se llama* swap *y el* agendamiento *luego del intercambio se denota $S(j,i)$.*

El objetivo de esta Sección es probar el siguiente teorema.

Teorema 3.5 *Existe S agendamiento óptimo y* creciente *para el problema de k-*sum lateness*.*

Para probar el Teorema 3.5 necesitaremos algunos lemas sobre listas de números. Para un vector $a \in \mathbb{R}^n$ arbitrario, llamemos $s_k(a)$ a la suma de los k mayores valores de a, es decir, $s_k(a) = \max\{\sum_{i=1}^n x_i a_i : \sum_{i=1}^n x_i = k, x \in \{0,1\}^n\}$.

Lema 3.6 *Sean $b, c \in \mathbb{R}^n$ con $c_h \leq b_h, \forall h \in [n]$, entonces $s_k(c) \leq s_k(b)$.*

Demostración. Sea $\bar{x} \in \{0,1\}^n$ el vector que maximiza $s_k(c)$, i.e, $\sum_{i=1}^n \bar{x}_i c_i = s_k(c)$. Luego $s_k(c) = \sum_{i=1}^n \bar{x}_i c_i \leq \sum_{i=1}^n \bar{x}_i b_i \leq s_k(b)$. □

Lema 3.7 *Sean $a, b \in \mathbb{R}^n$ tal que a y b coinciden en todas sus coordenadas excepto en i y en j, y además se tiene $b_i + b_j \leq a_i + a_j$, $\max(b_i, b_j) \leq \max(a_i, a_j)$, entonces $s_k(b) \leq s_k(a)$*

Demostración. Sea $x \in \{0,1\}^n$ que maximiza $s_k(b)$. Tenemos tres casos.

Caso 1. $x_i = x_j = 0$.

Como a y b coinciden salvo en las coordenadas i y j, tendremos que

$$s_k(b) = \sum_{h \in [n]} x_h b_h = \sum_{h \in [n]} x_h a_h \leq s_k(a).$$

Caso 2. $x_i = x_j = 1$.

Notemos que $s_k(b) = \sum_{h \in [n]} x_h b_h = (\sum_{h \in [n]} x_h a_h) - a_i - a_j + b_i + b_j$. Como $b_i + b_j \leq a_i + a_j$, tendremos que

$$s_k(b) \leq \sum_{h \in [n]} x_h a_h \leq s_k(a).$$

Caso 3. $x_i = 0, x_j = 1 \vee x_j = 0, x_i = 1$.

Como $x \in \{0,1\}^n$ es óptimo para $s_k(b)$ tendremos que $x_i b_i + x_j b_j = \max(b_i, b_j)$. Definimos el vector y igual a x salvo en las coordenadas i y j. Se define $y_i = 1$ e $y_j = 0$ cuando $a_i \geq a_j$, y como $y_i = 0$ e $y_j = 1$ cuando $a_i < a_j$. Así, $y_i a_i + y_j a_j = \max(a_i, a_j)$ y $\sum_{h \in [n]} y_h = k$. Luego,

$$\begin{aligned}
s_k(b) = \sum_{h \in [n]} x_h b_h &= \sum_{h \in [n] \setminus \{i,j\}} x_h b_h + (x_i b_i + x_j b_j) \\
&= \sum_{h \in [n] \setminus \{i,j\}} x_h b_h + \max(b_i, b_j) \\
&\leq \sum_{h \in [n] \setminus \{i,j\}} y_h a_h + \max(a_i, a_j) \\
&= \sum_{h \in [n]} y_h a_h \\
&\leq s_k(a).
\end{aligned}$$

□

Lema 3.8 *Sean $i, j \in J$ con $i \leq_S j$, $p_i > p_j, d_i \geq d_j$. Denotamos por $H = (i,j)_S$, y por $p(H) = \sum_{h \in H} p_h$ a su tiempo de proceso total. Entonces*

1. $L(i, S(j,i)) = L(i,S) + p(H) + p_j$,
2. $L(j, S(j,i)) = L(j,S) - p(H) - p_i$,
3. $L(h, S(j,i)) = L(h,S) + p_j - p_i < L(h,S), \forall h \in H$,
4. $L(h, S(j,i)) = L(h,S), \forall h \notin [i,j]_S$.

Demostración.

1. Como $C(i, S(j,i)) = C(i,S) + p(H) + p_j$, se concluye la igualdad deseada.
2. Basta notar que $C(j, S(j,i)) = C(j,S) - p(H) - p_i$.
3. Luego del *swap*, los trabajos $h \in H$ disminuyen su tiempo de completación en $p_i - p_j$, pues $C(i,S) - C(j, S(j,i)) = p_i - p_j$. Como $p_i > p_j$, se tiene que $L(h, S(j,i)) = L(h,S) - (p_i - p_j) < L(h,S)$.
4. Los trabajos $h \notin H$ mantienen su posición y además no cambian su tiempo de completación luego del *swap*.

□

Lema 3.9 *Sean $i, j \in J$ con $i \leq_S j$, $p_i > p_j, d_i \geq d_j$. Entonces $\sigma_k(S(j,i)) \leq \sigma_k(S)$.*

Demostración. Sean $i, j \in J$ con $i \leq_S j$, $p_i > p_j, d_i \geq d_j$. Definamos los vectores $a, b, c \in \mathbb{R}^n$, con $a_h = L(h,S)$, para todo $h \in J$, $c_h = L(h, S(j,i))$, para todo $h \in J$, y donde el vector b satisface $b_i = c_i$, $b_j = c_j$, $b_h = a_h$, para todo $h \in J - i - j$.

De la propiedad 3 del Lema 3.8, $c_h \leq a_h = b_h$ para todo $h \in J - i - j$. Como $c_i = b_i$, $c_j = b_j$, concluimos que como vectores, $c \leq b$. Luego del Lema 3.6, $s_k(c) \leq s_k(b)$.

Por otro lado, a y b coinciden en todas sus coordenadas excepto en i y en j, Además, $b_i + b_j = c_i + c_j$. De las propiedades 1 y 2 del Lema 3.8, $c_i + c_j = L(i, (S(j,i)) + L(j, S(j,i)) = L(i,S) + L(j,S) + p_j - p_i = a_i + a_j + p_j - p_i$. Como $p_i > p_j$, concluimos que $b_i + b_j < a_i + a_j$. Veamos ahora que $\max(b_i, b_j) \leq \max(a_i, a_j)$. Primero probemos que $b_i \leq a_j$. En efecto, $b_i = L(i, S(j,i)) = C(i, S(j,i)) - d_i = C(j,S) - d_i \leq C(j,S) - d_j = L(j,S) = a_j$. Por otro lado, de la propiedad 2 del Lema 3.8 tenemos $b_j = L(j, S(j,i)) = L(j,S) - p(H) - p_i < L(j,S) = a_j$, con lo cual concluimos que $\max(b_i, b_j) \leq a_j \leq \max(a_i, a_j)$. Así, como tenemos las hipótesis del Lema 3.7, concluimos que $s_k(b) \leq s_k(a)$.

Finalmente, concluimos que $\sigma_k(S(j,i)) = s_k(c) \leq s_k(b) \leq s_k(a) = \sigma_k(S)$.

□

Ahora estamos en condiciones de demostrar el resultado principal de la Sección.

Demostración Teorema 3.5. De todos los agendamientos óptimos para σ_k, elijamos aquel S con el menor *total lateness* $\sigma_n(S)$. Si S no es creciente, existen i, j con $i \leq_S j$, con i dominando a j. Denotamos por $S' = S(j, i)$. Por el Lema 3.9, $\sigma_k(S') \leq \sigma_k(S)$, por lo que S' también es óptimo para σ_k. Denotamos $H = (i, j)_S$. Luego, $\sigma_n(S') - \sigma_n(S) = (p_j - p_i)(|H| + 1) < 0$, lo que contradice que S es un agendamiento óptimo con el menor *total lateness*. □

3.2. Lagunas Consecutivas Crecientes

En esta Sección veremos que existen agendamientos S óptimos, crecientes y en los cuales el conjunto $S[k]^c$ estará ordenado por *EDD*. También recuperamos la conclusión principal del Capítulo 2 como Corolario del Teorema 3.12. Definimos primero la noción de *laguna*.

Definición 3.10 *Se define una laguna como un subconjunto de trabajos $L \subseteq S[k]^c$ con un* deadline *en común, que denotaremos por d_L. Diremos que el agendamiento S cumple* lagunas consecutivas *cuando para cada laguna se cumple que sus trabajos son consecutivos en S. Si además se cumple la propiedad, $d_{L_1} \leq d_{L_2}$ entonces L_1 está agendada antes de L_2, diremos que las lagunas son* consecutivas crecientes.

Lema 3.11 *Sea S un agendamiento creciente. Si existen $i, j \in S[k]^c$, con $i \leq_S j$, consecutivos en $S[k]^c$ (no necesariamente en S) y tales que $d_i \geq d_j$, entonces el agendamiento S' que se obtiene quitando i de S e insertándolo inmediatamente después de j cumple que $\sigma_k(S') \leq \sigma_k(S)$ y que S' es creciente. Además se cumple que $\sigma_n(S') > \sigma_n(S)$.*

Demostración. Llamamos $H = (i, j]_S$, y notamos que $H - j \subseteq S[k]$, por ser i, j consecutivos en $S[k]^c$. Usaremos los siguientes resultados.

(a) Para todo $h \in H$, $L(h, S') = L(h, S) - p_i < L(h, S)$.

(b) Para todo $q \notin H + i$, $L(q, S') = L(q, S)$.

(c) $L(i, S') = C(i, S') - d_i = C(j, S) - d_i \leq C(j, S) - d_j = L(j, S)$, pues $d_i \geq d_j$.

Probemos primero que si $h \in J - i$ es tal que $L(h, S) \geq L(j, S)$, entonces $L(h, S') \geq L(j, S')$. En efecto, si $h \in H$, sumando $-p_i$ a ambos lados de la desigualdad concluimos por (a). Si $h \notin H + i$ usando (b) y (a) respectivamente, concluimos que $L(h, S') = L(h, S) \geq L(j, S) > L(j, S')$. En particular, los trabajos $w \in S[k] \subseteq J - i$ cumplen que $L(w, S) \geq L(j, S)$, pues $j \notin S[k]$. Luego, $L(w, S') \geq L(j, S')$, lo que nos dice que $j \notin S'[k]$.

Veamos ahora que $\sigma_k(S') \leq \sigma_k(S)$. Como $i \notin S[k]$, hay dos casos. Si $i \notin S'[k]$, entonces $\sigma_k(S') = \sum_{h \in S'[k]} L(h, S') \leq \sum_{h \in S'[k]} L(h, S)$, por los resultados (a) y (b). Además,

$\sum_{h \in S'[k]} L(h,S) \leq \sigma_k(S)$ por definición de $\sigma_k(S)$. Por otro lado, si $i \in S[k]$, entonces

$$\begin{aligned}\sigma_k(S') &= \sum_{h \in S'[k]} L(h,S') \\ &= \sum_{h \in S'[k]; h \neq i} L(h,S') + L(i,S') \\ &\leq \sum_{h \in S'[k]; h \neq i} L(h,S) + L(i,S'),\end{aligned}$$

por los resultados (a) y (b). Ahora bien, usando la desigualdad en (c), tendremos que

$$\sigma_k(S') \leq \sum_{h \in S'[k]; h \neq i} L(h,S) + L(j,S).$$

Como $j \notin S'[k]$, concluimos que $\sigma_k(S') \leq \sigma_k(S)$, pues por definición $\sigma_k(S)$ es la suma de los k *lateness* mas grandes en S.

Veamos ahora que S' es creciente. En efecto, para todo $h \in H - j \subseteq S[k]$ se tiene que $d_h < d_i$, puesto que si no, existe $\tilde{h} \in H - j \subseteq S[k]$ con $d_{\tilde{h}} \geq d_i \geq d_j$. Como $\tilde{h} \leq_S j$, entonces $L(\tilde{h},S) = C(\tilde{h},S) - d_{\tilde{h}} < C(j,S) - d_{\tilde{h}} \leq C(j,S) - d_j = L(j,S)$, lo cual es una contradicción con que $\tilde{h} \in S[k]$ (pues $j \notin S[k]$). Así, para todo $h \in H - j$ se tiene que $d_h < d_i$. Esto a su vez nos dice que para todo $h \in H$, $p_h > p_i$, puesto que S es creciente (si $p_h < p_i$, entonces como S creciente se tendría que $h \leq_S i$, lo cual no sucede). Luego, i es incomparable con todo $h \in H$. Además, luego de la inserción de i, en S' se cumple que i seguirá estando a la derecha de los trabajos que dominaba en S, y a la izquierda de los trabajos por los cuales era dominado en S. Lo anterior también es cierto para los trabajos de H, pues estos mantienen su posición relativa entre ellos en S' con respecto a S. Luego, S' es creciente.

Comparemos los *total lateness* de cada agendamiento. Para el trabajo i tenemos que $L(i,S') - L(i,S) = \sum_{h \in H} p_h$, pues i se inserta después de j. Por el resultado (a), se tiene que $\sum_{h \in H} L(h,S') - L(h,S) = \sum_{h \in H} -p_i = -|H|p_i$. Luego, del resultado (b), la diferencia de los *total lateness* será $\sigma_n(S') - \sigma_n(S) = \sum_{h \in H} p_h - |H|p_i$. Como para todo $h \in H$ se tiene que $p_h > p_i$, concluimos que $\sigma_n(S') > \sigma_n(S)$. □

Teorema 3.12 *Existe un agendamiento S óptimo, creciente y con lagunas consecutivas crecientes.*

Demostración. Por el Teorema 3.5 existe S agendamiento óptimo creciente. Tomamos aquel que maximiza el *total lateness* σ_n.

Mostramos primero que $S[k]^c$ está ordenado según *EDD*. Por contradicción, si no, existen trabajos $i, j \in S[k]^c$ consecutivos como trabajos de $S[k]^c$ (no necesariamente en S), tales que $d_i > d_j$. Denotamos por S' al agendamiento resultante de quitar i de S e insertarlo inmediatamente después de j. Por el Lema 3.11, tenemos que $\sigma_k(S') \leq \sigma_k(S)$, por lo cual S' también será óptimo, y además S' creciente. Como se cumple que $\sigma_n(S') > \sigma_n(S)$, tendremos

una contradicción, pues S era un agendamiento óptimo, creciente y con el mayor *total lateness* posible. Luego, los trabajos en $S[k]^c$ deben estar ordenados según *EDD*.

Para terminar la demostración, veamos que las lagunas de S deben ser consecutivas. Como en $S[k]^c$ los trabajos están ordenados por *EDD*, si las lagunas no son consecutivas, entonces existen $i, j \in S[k]^c$ consecutivos en $S[k]^c$ y $d_i = d_j$, pero no consecutivos en S. Usando nuevamente el Lema 3.11 con i y j como antes, tendremos que S' es creciente y $\sigma_k(S') \leq \sigma_k(S)$. Como además $\sigma_n(S') > \sigma_n(S)$, se tendrá una contradicción con la elección de S. Así, S es creciente y con lagunas consecutivas crecientes. □

Se concluye la Sección con un corolario importante del Teorema 3.12, que nos permite probar la conclusión principal del Capítulo 2. Es decir, que para k constante, podemos resolver el problema de *k-sum lateness* en tiempo polinomial. En particular, el siguiente corolario nos da la existencia de agendamientos óptimos k-adyacentes al agendamiento de *EDD* (resultado ligeramente mas débil que el Teorema 2.3).

Corolario 3.13 *Existe un agendamiento S óptimo que es k-adyacente al agendamiento dado por* EDD.

Demostración. Del Teorema 3.12 existe un agendamiento S óptimo, creciente y con lagunas consecutivas crecientes. Es decir, si quitamos los trabajos de $S[k]$, los trabajos que queden estarán ordenados según *EDD*. Como $|S[k]| = k$, hay a lo más k trabajos fuera de lugar con respecto a *EDD*, con lo cual S será k-adyacente al orden dado por *EDD*. □

3.3. SPT por pivotes

En las Secciones anteriores probamos que existen agendamientos S óptimos, crecientes y con lagunas consecutivas crecientes para el problema de minimizar *k-sum lateness*. En esta Sección estudiamos la estructura del conjunto $S[k]$ y en particular mostramos que este tipo de agendamientos debe cumplir *SPT* en un sentido más débil, propiedad que llamaremos *SPT por pivotes*.

Definición 3.14 *Sea $\mathcal{D}$ el conjunto de* deadlines*. Para cada $\delta \in \mathcal{D}$ llamamos J_δ al conjunto de trabajos en J con* deadline *δ. Para cada $X \subseteq J$, llamemos $\mathcal{D}(X)$ al conjunto de* deadlines *que aparecen en los trabajos de X.*

Definición 3.15 *Sea $X \subseteq J$. Para cada $\delta \in \mathcal{D}(X)$ se define el pivote asociado a δ en X como el elemento $piv_\delta(X) \in X$ con* deadline *δ y de menor tiempo de proceso. Se define $piv(X)$ como el conjunto de todos los pivotes de X.*

Notamos que en un agendamiento creciente, los trabajos de igual *deadline* están ordenados de menor a mayor tiempo de proceso. Luego, para cada *deadline* $\delta \in \mathcal{D}[S(k)]$, el pivote $\text{piv}_\delta(S[k])$ es el primer trabajo con *deadline* δ en $S[k]$.

Definición 3.16 *Un agendamiento creciente S satisface* SPT *por pivotes si para todo $j \in S[k]$ y todo $i \in S[k] \cap [piv_{d_j}(S[k]), j)_S$, $p_i < p_j$.*

En otras palabras, un agendamiento S creciente satisface *SPT* por pivotes si para cada *deadline* $\delta \in \mathcal{D}(S[k])$, y cada trabajo $j \in S[k]$ con *deadline* δ, todos los trabajos que están entre el pivote asociado a δ en $S[k]$ y j tienen menor tiempo de proceso que p_j.

Teorema 3.17 *Sea S un agendamiento óptimo para k-*sum lateness *y creciente. Entonces satisface* SPT *por pivotes.*

Demostración. Sean $i, j \in S[k]$ con $j \neq \text{piv}_{d_j}(S[k])$ e $i \in [\text{piv}_{d_j}(S[k]), j)_S$. Por contradicción, suponemos que $p_i > p_j$. Como S es creciente, i no puede ser $\text{piv}_{d_j}(S[k])$, puesto que en ese caso $d_i = d_j$ y $p_i > p_j$, lo cual no puede pasar puesto que $i \leq_S j$. Luego $i \in (\text{piv}_{d_j}(S[k]), j)_S$. Como S es creciente y estamos suponiendo que $p_i > p_j$, debemos tener también que $d_i < d_j$.

Consideremos $S(j,i)$ el agendamiento resultante de hacer un *swap* entre i y j. Notemos que $i, j \in S(j,i)[k]$. En efecto, como $i \in S[k]$ es el único trabajo que aumenta su *lateness* luego del *swap*, tendremos que $i \in S(j,i)[k]$. Por otro lado, en $S(j,i)$ se tiene que $\text{piv}_{d_j}(S[k]) \leq_{S(j,i)} j$, y como $\text{piv}_{d_j}(S[k]) \in S[k]$, tendremos que $j \in S(j,i)[k]$. En efecto, como $\text{piv}_{d_j}(S[k]) \in S[k]$, para todo $h \notin S[k]$ se cumple que $L(h,S) \leq L(\text{piv}_{d_j}(S[k]), S)$. Como todos los trabajos $h \neq i$ después del *swap* mantienen o disminuyen su *lateness*, tendremos que para los $n-k$ trabajos $h \notin S[k]$ se tiene que $L(h, S(j,i)) \leq L(h,S)$. Luego, $L(h,S(j,i)) \leq L(h,S) \leq L(\text{piv}_{d_j}(S[k]), S) = L(\text{piv}_{d_j}(S[k]), S(j,i))$, con lo cual concluimos que $\text{piv}_{d_j}(S[k]) \in S(j,i)[k]$. Luego,

$$\begin{aligned}\sigma_k(S(j,i)) &= \sum_{h \in S(j,i)[k]} L(h, S(j,i)) \\ &= \sum_{h \in S(j,i)[k]: h \notin \{i,j\}} L(h, S(j,i)) + L(i, S(j,i)) + L(j, S(j,i)).\end{aligned}$$

De los resultados en 1 y 2 en el Lema 3.8, tendremos que $L(i, S(j,i)) + L(j, S(j,i)) = L(i,S) + L(j,S) + p_j - p_i$. Luego, $L(i, S(j,i)) + L(j, S(j,i)) < L(i,S) + L(j,S)$. Así,

$$\begin{aligned}&\sum_{h \in S(j,i)[k]: h \notin \{i,j\}} L(h, S(j,i)) + L(i, S(j,i)) + L(j, S(j,i)). \\ <& \sum_{h \in S(j,i)[k]: h \notin \{i,j\}} L(h, S(j,i)) + L(i,S) + L(j,S).\end{aligned}$$

Finalmente, notando que luego del *swap* todos los trabajos salvo i, o bien disminuyen su *lateness* (en $p_i - p_j$), o bien lo mantienen igual, tendremos que

$$\begin{aligned}&\sum_{h \in S(j,i)[k]: h \notin \{i,j\}} L(h, S(j,i)) + L(i,S) + L(j,S). \\ \leq& \sum_{h \in S(j,i)[k]: h \notin \{i,j\}} L(h,S) + L(i,S) + L(j,S).\end{aligned}$$

Por definición de $\sigma_k(S)$, tendremos que $\sum_{h \in S(j,i)[k]: h \notin \{i,j\}} L(h,S) + L(i,S) + L(j,S) \leq \sigma_k(S)$, con lo que $\sigma_k(S(j,i)) < \sigma_k(S)$. Esto es una contradicción, pues S es óptimo. □

Sea S un agendamiento creciente que satisface SPT por pivote. Denotamos por $X = S[k]$, y enumeremos las posiciones en las que aparece X en S como $Q = \{q_1, q_2, \ldots, q_k\} \subseteq [n]$, de menor a mayor posición. Además sea $f\colon \mathcal{D}(X) \to [k]$ la función que a cada *deadline* $\delta \in \mathcal{D}(X)$ le asocia la posición $\ell \in [k]$ tal que el pivote $\text{piv}_\delta(X)$ está en la posición q_ℓ. Veremos que si conocemos los trabajos X, las posiciones Q que ocupan los trabajos en S (no necesariamente en orden) y además las posiciones de los pivotes en Q, dadas por la función f, entonces es posible determinar los trabajos de X en cada posición en Q en el orden en el que están en S en tiempo $\mathcal{O}(Dk)$. En otras palabras, dado X, Q y f, mostraremos un algoritmo a tiempo $\mathcal{O}(Dk)$ que devuelve el agendamiento S restringido a Q. Es decir, el algoritmo indica para cada posición $q \in Q$, el trabajo $j \in X$ que debe ir en dicho lugar. En particular, X, Q y f determinan de manera única S restringido a Q, cuando S es un agendamiento óptimo, creciente y que satisface SPT por pivotes.

Denotamos por $X[\delta]$ al conjunto de los trabajos de X con *deadline* δ, de tal forma que $X = \bigcup_{\delta_i \in \mathcal{D}} X[\delta_i]$. Supondremos que $X[\delta]$ ya está ordenado de menor a mayor tiempo de proceso. El número de *deadlines* diferentes en X será $|\mathcal{D}(X)| = m$, y ordenamos los *deadlines* $\delta_1, \ldots, \delta_m$ de modo que $1 = f(\delta_1) < \ldots < f(\delta_m) \leq k$. Por simplicidad definimos $f_i = f(\delta_i)$ y $f_{m+1} = +\infty$. Probaremos que cuando X, Q y f provienen de un agendamiento S creciente y que cumple SPT por pivotes, entonces el siguiente algoritmo devolverá una función $g\colon [k] \to X$ que a cada $\ell \in [k]$ le asigna el trabajo en $g(\ell) \in X$ que va en la posición q_ℓ en S. La idea del Algoritmo 2 es reconstruir las posiciones que debe tener cada trabajo de $X = S[k]$ en S, cuando S es un agendamiento creciente y que cumple *SPT por pivotes*. Para ello, se actualiza una cola de trabajos disponibles Y cada vez que aparezca un pivote y se elige en cada iteración el trabajo con menor tiempo de proceso.

Algoritmo 2: Generar asignación de trabajos en posiciones.

Data: $X = S[k]$, $Q = \{q_1, \ldots, q_k\}$ posiciones de $S[k]$ en S, $f\colon \mathcal{D}(X) \to [k]$ función posición de pivotes, inyectiva.

Result: $g\colon [k] \to X$ función de asignación de trabajos en X en posiciones en Q.

```
if ∃i ∈ [m] tal que Σ_{j=1}^{i-1} |X[δ_j]| < f_i − 1 then
    return Error. // X, Q y f no provienen de un agendamiento.
end
Y ← ∅. // cola trabajos a agendar.
i ← 1. // pivotes procesados.
for ℓ = 1 → k do
    if ℓ = f_i then
        Y ← Y ∪ X[δ_i].
        i ← i + 1.
    end
    Determinar j con menor tiempo de proceso en Y.
    g(ℓ) ← j.
    Remover j de Y.
end
```

El Algoritmo 2 se puede implementar en tiempo $\mathcal{O}(kD)$ debido a que se realizan $\mathcal{O}(k)$ iteraciones, y por cada iteración, dado que las $X[\delta_i]$ están ordenadas por SPT, hay que revisar el trabajo inicial de a lo más D colecciones $X[\delta_i]$ para decidir qué trabajo de Y agendar.

Teorema 3.18 *Si X, Q y f provienen de un agendamiento S creciente y que cumple* SPT *por pivotes, entonces el Algoritmo 2 entrega una función $g\colon [k] \to X$ que a cada $\ell \in [k]$ le asigna el trabajo en $g(\ell) \in X$ que va en la posición q_ℓ en S.*

Demostración. Suponemos que X, Q y f como en el algoritmo vienen de un agendamiento S creciente y que cumple SPT por pivote. Recordemos que en todo agendamiento creciente los pivotes son los elementos más a la izquierda en $S[k]$ para un *deadline* dado. Por definición, f_i corresponde al índice de la posición del i-ésimo pivote en $S[k]$ (de izquierda a derecha). Luego en las posiciones $(q_1, \ldots, q_{f_i} - 1)$ en S solo pueden haber agendados trabajos de $X[\delta_1] \cup \cdots \cup X[\delta_{i-1}]$. Sigue que para todo i, $\sum_{j=1}^{i-1} |X[\delta_j]| \geq f_i - 1$, por lo que el algoritmo debe pasar el primer chequeo (líneas 1 a 3).

Veamos por inducción en ℓ que el trabajo que entrega el algoritmo $g(\ell)$ en ℓ coincide con el trabajo asignado a q_ℓ en S. Para el caso base, notemos que en la primera iteración el trabajo en q_1 en S coincide con $g(1)$, puesto que $Y = X[\delta_1]$, y $g(1)$ recibirá $\text{piv}_{\delta_1}(S[k])$, puesto que es el trabajo con menor tiempo de proceso con *deadline* δ_1 en Y. Ahora bien, supongamos que todo está bien asignado hasta $\ell - 1$.

Sea j^* el trabajo que debería asignar g en ℓ, es decir, j^* es el trabajo con posición q_ℓ en $S[k]$, y sea $j = g(\ell)$ el trabajo que asigna el algoritmo. Supongamos que $j \neq j^*$. Luego, debemos tener $j^* \leq_S j$, pues todo está bien asignado antes de q_ℓ y j aún no está asignado al principio de la iteración ℓ. Además, antes de empezar la iteración ℓ, como i cuenta los pivotes procesados, se tiene que todos los pivotes asociados a $\delta_1, \ldots, \delta_{i-1}$ están, en el agendamiento real $S[k]$, en posiciones más a la izquierda que q_ℓ. Sea $z = \text{piv}_{d_j}(X)$ el pivote asociado al *deadline* de j. Tendremos dos casos.

Caso 1. Si j^* no es pivote, entonces Y no se actualiza en esta iteración. Luego, j y j^* están en Y desde una iteración anterior. En particular, $j \in X[\delta_1] \cup \cdots \cup X[\delta_{i-1}]$ y probamos que los pivotes de dichos trabajos están a la izquierda de la posición q_ℓ, lo cual nos dice que el pivote asociado z satisface $z \leq_S j^*$. Luego $j^* \in (\text{piv}_{d_j}(S[k]), j)_S$ y como el algoritmo elige j sobre j^* en la iteración ℓ, se debe tener que $p_j < p_{j^*}$, lo cual contradice que S cumpla SPT por pivotes.

Caso 2. Si j^* es pivote. Entonces j^* es el pivote asociado al *deadline* δ_i para el i al principio de la iteración (es decir $\ell = f_i$). El algoritmo actualiza Y al conjunto de todos los trabajos en $X[\delta_1] \cup \cdots \cup X[\delta_i]$ que no han sido asignados a posiciones anteriores a q_ℓ. En particular tenemos que $j^* \in Y$ y por ser pivote, es el trabajo de menor tiempo de proceso en $X[\delta_i]$. Como supusimos $j^* \neq j$ y tenemos que $p_j < p_{j^*}$ (si no, el algoritmo no lo hubiese elegido), entonces j debe ser un trabajo de $Y \setminus X[\delta_i]$. Como sabemos que estos trabajos tienen sus pivotes a la izquierda de la posición q_ℓ, el pivote z asociado a d_j está a la izquierda de q_ℓ. Luego $j^* \in (\text{piv}_{d_j}(S[k]), j)_S$, lo que es una contradicción pues S cumple SPT por pivotes.

Se concluye que el algoritmo entrega una función $g\colon [k] \to X$ que asigna los trabajos de

$S[k]$ a la posición que tienen en S de forma correcta.

□

3.4. Algoritmo para D constante

De las secciones anteriores sabemos que existe un agendamiento S óptimo, creciente y con lagunas consecutivas crecientes. Llamamos $\delta_1 < \delta_2 < \cdots < \delta_D$ a los *deadlines* distintos en $\mathcal{D}$. Para $i \in [D]$, denotamos L_i a la laguna asociada al *deadline* δ_i (L_i podría ser vacía para algún i). Como las lagunas consecutivas en S son crecientes tenemos que $L_1 \leq_S \cdots \leq_S L_D$ y notamos que si $L_h \neq \emptyset$, entonces a la izquierda de L_h no pueden haber trabajos $w \in S[k]$ con $d_w \geq \delta_h$. En efecto, si existe un trabajo $w \in S[k]$ a la izquierda de L_h en S con $d_w \geq \delta_h$, como $L_h \neq \emptyset$, existe $\ell_h \in L_h$ tal que $w \leq_S \ell_h$. Luego, $L(w, S) < L(\ell_h, S)$, lo cual es una contradicción, pues $\ell_h \notin S[k]$ y $w \in S[k]$.

Llamemos *isla* B_i al conjunto maximal de trabajos en $S[k]$ justo después de la laguna L_i y antes de la siguiente laguna L_{i+1}. De esta manera J se particiona en D lagunas y D islas de la siguiente forma

$$L_1 \leq_S B_1 \leq_S L_2 \leq_S B_2 \leq_S \cdots \leq_S L_D \leq_S B_D,$$

donde algunas lagunas o islas podrían ser vacías. En la Figura 3.1 se representa la estructura de los agendamientos óptimos que buscamos, donde las alturas representan los *deadlines* de los trabajos en cada colección.

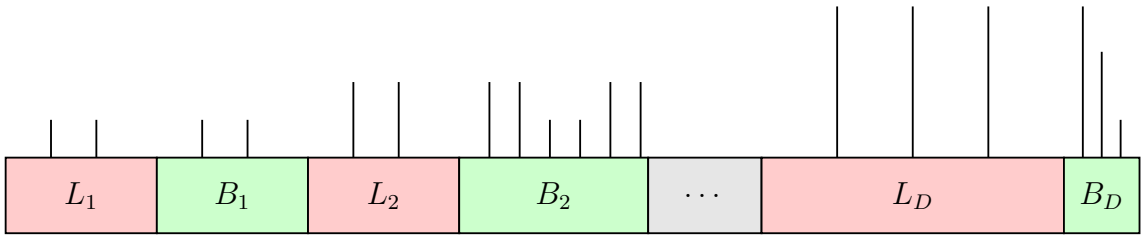

Figura 3.1: Agendamientos con lagunas e islas.

Llamemos $\alpha_i = |L_i|$ y $\beta_i = |B_i|$ para todo i. Tenemos el siguiente lema.

Lema 3.19 *Sean $J_i \subseteq [n]$ trabajos con* deadline *δ_i, ordenados por* SPT *y tales que $\{J_i\}_{i=1}^{D}$ particiona J. Si los valores $\alpha_1, \ldots, \alpha_D$ corresponden a las cardinalidades de las lagunas y $\beta_1, \ldots, \beta_D$ las cardinalidades de las islas, de un agendamiento S creciente y con lagunas consecutivas, entonces podemos determinar en tiempo $\mathcal{O}(D + k)$ los conjuntos $L_1, \ldots, L_D$ asociados a S, el conjunto de trabajos $S[k]$ y el conjunto $Q = \{q_1, \ldots, q_k\}$ de las posiciones en las que están los elementos de $S[k]$.*

Demostración. Como vimos, a la izquierda de una laguna $L_h \neq \emptyset$ no pueden haber trabajos $w \in S[k]$ con $d_w \geq \delta_h$. Además, como S es creciente entonces J_h está ordenado según

SPT. Luego, los trabajos de L_h deben ser los primeros α_h trabajos de J_h. Así, determinar todas las lagunas toma $\mathcal{O}(D)$, pues hay a lo más D lagunas distintas. Teniendo las lagunas, $S[k] = [n] \setminus \bigcup_{h\in[D]} L_h$. Las posiciones Q de $S[k]$ las podemos determinar en tiempo $\mathcal{O}(k)$ dados los α_i y β_i, pues estos para determinar las posiciones de los trabajos en las D islas. □

Supongamos ahora que S es creciente, con lagunas consecutivas y además óptimo. Del Teorema 3.18 sabemos que conociendo $X = S[k]$, las posiciones Q y la asignación f que a cada *deadline* δ_i en $\mathcal{D}(X)$ le asigna la posición q_i en la que está ubicado el pivote del *deadline* δ_i, podemos recuperar el agendamiento S restringido a las posiciones Q. Finalmente, del Lema 3.19, concluimos que para determinar S basta conocer los valores α_i, β_i y la función f. De esta forma, la idea del Algoritmo 3 es revisar todos los agendamientos que tienen una estructura de lagunas e islas como la vista en la Figura 3.1, aprovechando el hecho de que cuando D es constante, hay una cantidad a lo más polinomial en n de candidatos a (α, β, f) que se deben considerar para generarlos.

Algoritmo 3: *k-sum lateness*, segunda versión.

Data: $J = [n]$ trabajos con tiempos de proceso arbitrarios y *deadlines* $d_1, \ldots, d_D$.
Result: S_{best} agendamiento óptimo para σ_k

$S \leftarrow \emptyset$.
$S_{\text{best}} \leftarrow \emptyset, val_{\text{best}} \leftarrow \infty$.
$J_i \leftarrow \{j \in J : d_j = d_i\}, \forall i \in [D]$.
Ordenar J_i por *SPT*, $\forall i \in [D]$.
for $\alpha_1, \ldots, \alpha_D,\ \beta_1, \ldots, \beta_D \in \mathbb{N}^D$ *tal que* $\sum \alpha_i = n - k$ *y* $\sum \beta_i = k$ **do**
    if $\exists i$ *tal que* $\alpha_i > |J_i|$ **then**
        continue.
    end
    Usar Lema 3.19 para determinar $\{L_i\}_{i=1}^k$ y Q.
    $X_{\delta_i} \leftarrow J_i \setminus L_i\ \forall i \in [D]$.
    $\mathcal{F} \leftarrow \{\delta_i : X_{\delta_i} \neq \emptyset\}$, $X \leftarrow \bigcup_{\delta_i \in \mathcal{F}} X_{\delta_i}$.
    for $f : \mathcal{F} \to Q$ *función inyectiva* **do**
        Determinar $g\colon [k] \to X$ función que a cada posición q_j le asocia un único trabajo en X usando Algoritmo 2. Definimos $G := (g(1), \ldots, g(k))$.
        $B_j \leftarrow G[c(j) : c(j+1) - 1]$ con $c(j) = 1 + \sum_{h<j} \beta_h\ \forall j \in [D]$.
        Obtener S concatenando: $S \leftarrow L_1 \mid B_1 \mid \ldots \mid L_i \mid B_i \mid \ldots \mid L_D \mid B_D$.
        $val_S \leftarrow \sigma_k(S)$.
        if S *es factible* [1] $\wedge\ val_S \leq val_{best}$ **then**
            $S_{\text{best}} \leftarrow S$.
            $val_{\text{best}} \leftarrow val_S$.
        end
    end
end

[1] Decidir la factibilidad de S quiere decir que $L_j \subseteq S[k]^c$ y $B_j \subseteq S[k], \forall j \in [D]$.

Teorema 3.20 *El Algoritmo 3 encuentra un agendamiento óptimo para* k-sum lateness.

Demostración. Sea S' un agendamiento óptimo, creciente, con lagunas consecutivas crecientes. Sabemos además que S' cumple SPT por pivotes por el Teorema 3.17. Mostremos que el Algoritmo 3 revisa este agendamiento. En efecto, denotamos por $L_1(S'), \ldots, L_D(S')$ a las lagunas de S' asociadas a $\delta_1, \ldots, \delta_D$ resp., y por $B_1(S'), \ldots, B_D(S')$ a sus islas. Consideremos la iteración cuando $\alpha_1 = |L_1(S')|, \ldots, \alpha_D = |L_D(S')|$ y $\beta_1 = |B_1(S')|, \ldots, \beta_D = |B_D(S')|$. Como S' es creciente, los trabajos de cada $L_i(S')$ serán los trabajos con menor tiempo de proceso con *deadline* δ_i, por lo que las lagunas de S' coincidirán con las entregadas por el algoritmo. Es decir, $L_1(S') = L_1$. Además, esto nos dice inmediatamente que $X = S'[k]$. Luego, por como se eligieron los α_i y β_i, se tendrá que las posiciones Q' de los trabajos en $S'[k]$ en S' cumplen que $Q' = Q$. Denotamos por $f_{S'} : \mathcal{D}(S'[k]) \to Q$ a la función que a cada *deadline* δ en $\mathcal{D}(S'[k])$ le asigna la posición q_i en la que está ubicado el pivote del *deadline* δ en S'. Como $f_{S'}$ inyectiva, el algoritmo la revisa. Luego, la función que entrega el algoritmo en la línea 13 permite recuperar S' restringido a Q. Finalmente, por como están definidos los β_i, se tendrá que la concatenación en la línea 15 entrega un S tal que $S = S'$ en esta iteración. Luego, el algoritmo encuentra un agendamiento óptimo. □

A continuación usando el Algoritmo 3, mostramos que *k-sum lateness* se puede resolver en tiempo polinomial cuando D es constante.

Teorema 3.21 *El problema de minimizar* k-sum lateness *para* n *trabajos se puede resolver en tiempo polinomial* $\mathcal{O}(n^{3D+1}D)$*, para cada* D *fijo.*

Demostración. Las líneas 1 a 4 se implementan en tiempo $\mathcal{O}(n \log(n))$. El algoritmo revisa a lo más $\mathcal{O}(n^{2D})$ combinaciones de $\alpha_1, \ldots, \alpha_D, \beta_1, \ldots, \beta_D$ en el *for* de la línea 5. Analicemos cada iteración. Del Lema 3.19, determinar L_i, X y Q toma tiempo $\mathcal{O}(D+k) = \mathcal{O}(n)$. Hay a lo más $\mathcal{O}(n^D)$ funciones $f : \mathcal{F} \to Q$ inyectivas y para cada una de ellas, se ejecuta el Algoritmo 2 de complejidad $\mathcal{O}(Dk)$. En total, cada iteración toma tiempo $\mathcal{O}(n + n^D Dk)) = \mathcal{O}(n^{D+1}D)$. Luego, el Algoritmo tiene complejidad $\mathcal{O}(n \log(n) + n^{2D} n^{D+1} D) = \mathcal{O}(n^{3D+1}D)$. □

Capítulo 4

k-sum lateness con P constante

En este capítulo estudiamos el caso particular del problema de *k-sum lateness* con P, *el número de tiempos de procesos distintos*, constante. En la sección 4.1 modelamos *k-sum lateness* como un programa lineal mixto anidado. Usando dualidad mostramos que el problema es equivalente a un problema de minimizar tardanza total (*total tardiness*) paramétrico. En la Sección 4.2 mostramos que el problema paramétrico anterior, en el caso de P constante, se puede reducir a resolver una cantidad polinomial de instancias de minimizar tardanza total. Finalmente, en la Sección 4.3 veremos que podemos adaptar el algoritmo *pseudopolinomial* de Lawler [7] para resolver las instancias de *total tardiness* en tiempo polinomial en el caso en el que P es constante, completando así la descripción de un algoritmo polinomial para nuestro problema.

4.1. Formulación de *k-sum tardiness* como *PLM* anidado

Denotemos $\mathcal{S}_n$ al conjunto de permutaciones del conjunto $[n]$ de trabajos. Siguiendo la metodología propuesta por Puerto [9] para problemas de tipo *k-sum optimization*, usando variables auxiliares binarias x, podemos formular el problema de *k-sum lateness* como sigue

$$
\begin{aligned}
(P) \quad & \min_{S\in\mathcal{S}_n} \max_{x} \sum_{i=1}^{n} (C(i,S) - d_i)x_i \\
\text{s.a.} \quad & \sum_{i=1}^{n} x_i = k, \\
& x_i \in \{0,1\} \quad \forall i \in [n].
\end{aligned}
$$

Es fácil ver que la relajación lineal natural del *PLE* interno de maximización es integral, pues tiene matriz de restricciones *totalmente unimodular* y lado derecho integral. Luego, podemos

reemplazar el problema (P) por un *min-max* lineal puro. Es decir, reformulamos (P) como

$$\begin{aligned}(P') \quad & \min_{S\in\mathcal{S}_n} \max_{x} \sum_{i=1}^{n}(C(i,S)-d_i)x_i \\ & \text{s.a.} \quad \sum_{i=1}^{n} x_i = k, \\ & \qquad x_i \in [0,1] \quad \forall i \in [n].\end{aligned}$$

Tomando el dual del problema interno, y usando r como variable dual para la restricción de igualdad y la variable y_i para la restricción $x_i \leq 1$, el problema anterior es equivalente a

$$\begin{aligned}(D) \quad & \min_{S\in\mathcal{S}_n} \min_{r,y} \; kr + \sum_{i=1}^{n} y_i \\ & \text{s.a.} \quad y_i \geq C(i,S) - d_i - r \quad \forall i \in [n], \\ & \qquad y_i \geq 0 \qquad\qquad\qquad\quad \forall i \in [n].\end{aligned}$$

Ahora bien, en el problema interno de (D) notamos que en todo óptimo se tiene que $y_i = (C(i,S) - d_i - r)_+$. Es decir, y_i es la tardanza de i en S, pero usando como *deadline* $d_i + r$. Luego, podemos quitar y_i como variable y una formulación equivalente a (D) será

$$(D') \quad \min_{S\in\mathcal{S}_n} \min_{r} \; kr + \sum_{i=1}^{n}(C(i,S) - d_i - r)_+.$$

Definamos para cada $r \in \mathbb{R}$, la función $\varphi(r) = kr + \min_{S\in\mathcal{S}_n} \sum_{i=1}^{n}(C(i,S) - d_i - r)_+$ que corresponde a una función lineal en r más el mínimo *total tardiness* de un conjunto de trabajos con *deadlines* $d_i + r$. Intercambiando el orden de los mínimos en la expresión para (D'), concluimos que este se puede reformular como el problema de minimización paramétrico $\min_r \varphi(r)$. En la siguiente Sección mostraremos que la función $\varphi(r)$ es lineal por trozos y que tiene a lo más $\mathcal{O}(n^{P+1})$ puntos de quiebre, por lo cual se puede reducir a resolver una cantidad polinomial de instancias del problema de minimizar tardanzas totales.

4.2. Reducción de k-*sum lateness* a *total tardiness* cuando P es constante

Denotaremos el conjunto de todos los *lateness* posibles como $\mathcal{L} := \{C(i,S) - d_i : i \in [n],\ S \text{ agendamiento}\}$. Mostraremos primero que $|\mathcal{L}|$ es polinomial en n si P constante.

Lema 4.1 $|\mathcal{L}| = \mathcal{O}(n^{P+1})$.

Demostración. Sea $\mathcal{P}$ el conjunto de los tiempos de proceso distintos y $|\mathcal{P}| = P$. Fijemos un agendamiento S y un trabajo j. Para cada $x \in \mathcal{P}$ definamos

$$a_x(S,j) = \text{número de trabajos con tiempo de proceso } x \text{ agendados antes de } j \text{ en } S,$$

con lo cual, el tiempo de partida del trabajo j en S es

$$C(j,S) - p_j = \sum_{x \in \mathcal{P}} a_x(S,j) \cdot x.$$

Como cada $a_x(S,j) \in \{0,\ldots,n-1\}$, concluimos que para cada trabajo j sus tiempos de partida posibles pertenecen a un conjunto fijo (independiente de j y de S) de tamaño a lo más n^P. Además, el *lateness* de un trabajo se obtiene sumando una cantidad propia del trabajo (*i.e*, $p_j - d_j$) al tiempo de partida y como hay n trabajos, se concluye que $|\mathcal{L}| \leq n^{P+1}$. □

Teorema 4.2 *Para cada agendamiento S fijo, la función $f_S(r) = kr + \sum_{i=1}^{n}(C(i,S) - d_i - r)_+$ es* lineal por trozos*, continua, convexa y con puntos de quiebre contenidos en el conjunto de* lateness *posibles $\mathcal{L}$.*

Demostración. Al ser suma de funciones lineales por trozos, convexas y continuas, la función f_S es, en si misma, también lineal por trozos, convexa y continua. Veamos que los puntos de quiebre de f_S están contenidos en $\mathcal{L}$. En efecto, supongamos sin pérdida de generalidad que los trabajos están enumerados de forma creciente con su *lateness* según la permutación S. Es decir,

$$C(1,S) - d_1 \leq C(2,S) - d_2 \leq \cdots \leq C(n,S) - d_n.$$

Definamos, para $j \in [n-1]$ los intervalos semi-abiertos por la derecha $I_j(S) := [C(j,S) - d_j, C(j+1,S) - d_{j+1})$. Algunos de estos intervalos podrían ser vacíos si dos o más trabajos tienen igual *lateness*. Definiendo además $I_0(S) = (-\infty, C(1,S) - d_1)$ e $I_n = [C(n,S) - d_n, +\infty)$ tenemos que la familia $(I_j(S))_{j \in \{0,1,\ldots,n\}}$ particiona a la recta real. Más aún, para todo $j \in [n] \cup \{0\}$, la función $f_S(r)$ restringida a $I_j(S)$ es lineal, y tiene el siguiente valor

$$f_S(r) = kr + \sum_{i=1}^{n}(C(i,S) - d_i - r)_+ = kr + \sum_{i=j+1}^{n}(C(i,S) - d_i - r).$$

Así, los puntos de quiebre de f_S son subconjunto de $\{C(i,S) - d_i \colon i \in [n]\} \subseteq \mathcal{L}$. □

El Teorema 4.2 anterior nos dice que para todo agendamiento S, el problema de minimización interior en (D') es sobre una función lineal por trozos, continua y convexa, con puntos de quiebre en $\mathcal{L}$. Como las funciones de este tipo alcanzan su mínimo en un punto de quiebre, basta buscar mínimos en $\mathcal{L}$. Luego, tendremos que

$$\begin{aligned}(D') \quad & \min_{S \in \mathcal{S}_n} \min_{r} \; kr + \sum_{i=1}^{n}(C(i,S) - d_i - r)_+ \\ = \quad & \min_{S \in \mathcal{S}_n} \min_{r \in \mathcal{L}} \; kr + \sum_{i=1}^{n}(C(i,S) - d_i - r)_+.\end{aligned}$$

Por otro lado, reordenando los mínimos tendremos que

$$\begin{aligned} & \min_{S\in\mathcal{S}_n}\min_{r\in\mathcal{L}} \; kr + \sum_{i=1}^{n}(C(i,S)-d_i-r)_+ \\ (D'') \quad = \quad & \min_{r\in\mathcal{L}} \; kr + \underbrace{\min_{S\in\mathcal{S}_n}\sum_{i=1}^{n}(C(i,S)-d_i-r)_+}_{(T(r))}. \end{aligned} \tag{4.1}$$

Notemos que para un par óptimo (r^*, S^*) para el problema (D''), el agendamiento S^* también será óptimo para el problema (P) original. Esto pues, (D'') es equivalente a (D') y para pasar de (P') a (D'), se reformuló el problema dual del problema de maximización interno, dejando el problema externo de minimización sobre las permutaciones fijo. Luego, como (P') y (P) son problemas equivalentes, el agendamiento S^* óptimo de (D'') será también óptimo para (P).

Recordemos que el *tardiness* de un trabajo j en un agendamiento S con tiempo de completación $C(j,S)$ y *deadline* d_j, está definido como $T(j,S) = (C(j,S)-d_j)_+$. Luego, el problema $(T(r))$ será el de minimizar el *total tardiness* con *deadlines* $\tilde{d_j} = d_j + r$, que sabemos que es un problema *NP-completo* [1] para el caso general. Sin embargo, en la siguiente Sección veremos que en el caso particular en que P, el número de tiempos de proceso distintos, es constante, entonces existe un algoritmo polinomial que resuelve $(T(r))$.

Teorema 4.3 *El problema de minimizar* total tardiness *para n trabajos se puede resolver en tiempo polinomial* $\mathcal{O}(n^{P+4})$ *para P fijo.*

Asumiremos cierto por ahora el Teorema 4.3 y lo demostraremos en la Sección 4.3. Con lo anterior podemos plantear el siguiente algoritmo.

Algoritmo 4: *k-sum lateness*, tercera versión.

Data: $J = [n]$ trabajos con *deadlines* arbitrarios y P tiempos de proceso.
Result: (S_{best}, val_{best}) par del agendamiento óptimo y su valor para σ_k

```
S_best ← ∅.
val_best ← ∞.
Generar conjunto L de todos los lateness posibles.
for r ∈ L do
    Calcular el óptimo S_r del problema (T(r)), con valor val_(T(r))(S_r).
    val_r ← kr + val_(T(r))(S_r).
    if val_r ≤ val_best then
        S_best ← S_r.
        val_best ← val_r.
    end
end
return (S_best, val_best)
```

El siguiente resultado nos dice que el algoritmo anterior soluciona *k-sum lateness*.

Teorema 4.4 *El Algoritmo 4 genera un* agendamiento *óptimo para* k-sum lateness.

Demostración. En efecto, sabemos que las soluciones de (D'') resuelven el problema de *k-sum lateness* formulado en (P). Por el Teorema 4.2 sabemos que la función $\varphi(r) = kr + \min_{S\in\mathcal{S}_n} \sum_{i=1}^{n}(C(i,S) - d_i - r)_+$ es lineal por trozos, continua y convexa, con puntos de quiebre contenidos en $\mathcal{L}$, por lo que basta mirar el conjunto de puntos de quiebre de la función para encontrar el mínimo del problema. Como el algoritmo chequea todos los puntos de quiebre $r \in \mathcal{L}$ y para cada r encuentra la permutación S_r que minimiza (T_r) y se queda con el valor más pequeño de $\varphi(r) = kr + val_{(T(r))}(S_r)$, entonces el algoritmo encuentra un agendamiento óptimo para (P). □

A continuación usando el Algoritmo 4, mostramos que *k-sum lateness* se puede resolver en tiempo polinomial cuando P es constante.

Teorema 4.5 *El problema de* k-sum lateness *para* n *trabajos se puede resolver en tiempo polinomial* $\mathcal{O}(n^{2P+5})$*, para cada* P *fijo.*

Demostración. En efecto, del Lema 4.1 sabemos que $|\mathcal{L}| = \mathcal{O}(n^{P+1})$. No es difícil ver que dicho conjunto se puede generar en tiempo proporcional a su cardinal. Ahora bien, por cada $r \in \mathcal{L}$ se resuelve el problema de *total tardiness*, que por el Teorema 4.3 toma tiempo $\mathcal{O}(n^{P+4})$, con lo cual se concluye el resultado. □

4.3. Minimizar *total tardiness* con P constante es polinomial: Demostración del Teorema 4.3

En esta Sección estudiamos el problema de *total tardiness* cuando P, cantidad de tiempos de proceso diferentes, es constante. Para ello, nos basamos en un trabajo previo de Lawler [7]. En dicho trabajo se propone un algoritmo *pseudopolinomial* que utiliza *programación dinámica* y que encuentra una solución óptima para el problema en tiempo $\mathcal{O}(n^4 S_P)$ con $S_P = \sum_{j\in J} p_j$. En esta Sección mostraremos que dicho algoritmo se puede implementar en tiempo polinomial cuando P es constante.

Para simplificar la discusión en lo que sigue, supondremos que todos los trabajos tienen *tiempos de proceso* distintos. Esto pues, si no lo son, estos pueden ser modificados infinitesimalmente sin cambiar el problema de forma significativa. Una vez concluyamos el análisis, volveremos a considerar los tiempos de proceso originales.

Lawler [7] prueba el siguiente resultado, que es la base para poder crear un *programa dinámico* para el problema de *total tardiness*.

Teorema 4.6 *[7, Teorema 3] Suponemos que todos los trabajos están numerados de forma*

no decreciente según su deadline*, es decir,* $d_1 \leq \cdots \leq d_n$*. Sea* k *el trabajo con mayor* tiempo de proceso *según esta enumeración, i.e,* $p_k = \max_j\{p_j\}$*. Entonces existe un número entero* δ *con* $0 \leq \delta \leq n-k$ *tal que existe un agendamiento óptimo* S *en el cual a la izquierda de* k *están todos los trabajos* j *con* $j \leq k+\delta$*, y a su derecha todos los trabajos* j *con* $j > k+\delta$*.*

4.3.1. Algoritmo de Programación Dinámica

Construiremos ahora en tiempo polinomial $\mathcal{O}(n^{P+3})$ un agendamiento óptimo para el problema de *total tardiness* utilizando *programación dinámica*, basándonos en lo realizado por Lawler [7, Sección 3].

Sea $J = [n]$ una colección de trabajos numerados en orden no decreciente con sus *deadlines*. Queremos encontrar un agendamiento óptimo de los trabajos $1, \ldots, n$, cuyo primer trabajo comienza en un tiempo t. Sea k el trabajo con el mayor *tiempo de proceso*. Del Teorema 4.6 se tiene que existe un δ con $0 \leq \delta \leq n-k$ y un agendamiento óptimo S que se puede separar en tres porciones.

1. En la primera porción, los trabajos $1, \ldots, k-1, k+1, \ldots, k+\delta$, en alguna secuencia, comenzando en el tiempo t, seguidos de

2. En la segunda porción, el trabajo con mayor tiempo de proceso k, con tiempo de completación $C_k(\delta) := t + \sum_{j \leq k+\delta} p_j$, seguido de

3. En la porción final, los trabajos $k+\delta+1, \ldots, n$, en alguna secuencia, empezando en el tiempo $C_k(\delta)$.

El problema de *total tardiness* satisface el principio de optimalidad siguiente: la secuencia completa es óptima solo cuando sus subsecuencias consecutivas también son óptimas. En particular, deben ser subsecuencias óptimas las secuencias de trabajos descritas en las porciones 1 y en 3 que comienzan en t y $C_k(\delta)$ respectivamente. Esto último sugiere un *programa dinámico* como método de solución. Un agendamiento óptimo S que comienza en un tiempo t puede ser encontrado de forma recursiva desde soluciones óptimas de subproblemas S', t', con S' subconjunto propio de S y $t' \geq t$.

Notemos por otro lado que las porciones en 1 y 3 son de un tipo especial. Esto pues, ambas secuencias están conformadas por trabajos con *tiempo de proceso* menor a p_k. Se utilizará la siguiente notación

$$S(i\colon j, k) = \{w\colon i \leq_S w \leq_S j, p_w < p_k\}. \tag{4.2}$$

Luego, la secuencia en 1 será $S(1\colon k+\delta, k)$ y en 3 será $S(k+\delta\colon n, k)$.

Denotamos por $T(S(i\colon j, k), t)$ a la tardanza de algún agendamiento óptimo de los trabajos en $S(i\colon j, k)$ y que comienzan en t. Así, en virtud del principio de optimalidad y el Teorema 4.6, tendremos que

$$\begin{aligned} T(S(i\colon j, k), t) = \min_\delta\{&T(S(i\colon k'+\delta, k'), t) + \max\{0, C_{k'}(\delta) - d_{k'}\} \\ &+ T(S(k'+\delta+1\colon j, k'), C_{k'}(\delta))\}, \end{aligned} \tag{4.3}$$

donde k' se define como el trabajo de mayor *tiempo de proceso* en la subsecuencia $S(i\colon j,k)$.

$$p_{k'} = \text{máx}\{p_w : w \in S(i\colon j,k)\},$$

y además,

$$C_{k'}(\delta) = t + \sum_{w \in S(i\colon k+\delta,k')} p_w.$$

Por otro lado, las condiciones iniciales para la ecuación 4.3 están dadas por

$$T(\emptyset, t) = 0.$$
$$T(\{j\}, t) = \text{máx}\{0, t + p_j - d_j\}.$$

Observación El programa dinámico anterior se puede implementar incluso si los tiempos de proceso no son todos distintos. Para ello, basta definir una regla de desempate consistente entre trabajos de igual tiempo de proceso y así poder calcular los conjuntos $S(i\colon j,k)$ y los valores $T(S(i\colon j,k),t)$.

El siguiente resultado muestra que el problema de *k-sum tardiness* se puede resolver en tiempo $\mathcal{O}(n^{P+4})$.

Demostración del Teorema 4.3. De la discusión anterior sabemos que existe al menos un agendamiento óptimo S que puede ser construido con la recursión vista en la Sección 4.3.1. Mostraremos que podemos encontrar uno de estos agendamientos en tiempo $\mathcal{O}(n^{P+4})$.

En efecto, notemos que no hay más de $\mathcal{O}(n^3)$ subconjuntos $S(i\colon j,k)$, puesto que no hay más de n valores posibles para i, j y k. Para contar los posibles tiempos t en las que pueden comenzar estas colecciones de trabajos, dado que P es constante, basta saber para cada trabajo i, cuántos trabajos de cada *tiempo de proceso* están agendados antes de i. Sea $\mathcal{P}$ el conjunto de los tiempos de proceso distintos ($|\mathcal{P}| = P$). Para cada $x \in \mathcal{P}$ definimos

$$a_x(i) = \text{número de trabajos con tiempo de proceso } p_x \text{ antes de } i.$$

Luego,

$$t = \sum_{x=1}^{n} a_x(i) \cdot p_x.$$

Ahora bien, como

$$a_1(i) + \cdots + a_P(i) \leq n - 1, \tag{4.4}$$

tendremos que a lo más hay $\mathcal{O}(n^P)$ tiempos t en los cuales puede comenzar $S(i\colon j,k)$. Así, no hay más de $\mathcal{O}(n^{P+3})$ ecuaciones en 4.3 a resolver. Como en dicha ecuación se está computando un mínimo, cada ecuación requiere tiempo $\mathcal{O}(n)$ para resolverla. Así, se podrá encontrar un agendamiento óptimo en $\mathcal{O}(n^{P+4})$ para el problema de *total tardiness* con P constante. □

Conclusiones

El problema de minimizar *k-sum lateness* es el objeto de estudio central de esta tesis. La principal contribución del trabajo consiste en dar algoritmos polinomiales para resolver importantes casos particulares de este problema, que en su versión general conjeturamos es un problema *NP-completo*. En particular, estudiamos el problema dejando constante separadamente el parámetro k, la cantidad P de tiempos de proceso distintos y la cantidad D de *deadlines* distintos. Además, desarrollamos algoritmos basados en técnicas distintas para cada uno de estos tres casos particulares.

Para el caso en que k es constante, extendimos la idea de Woeginger [12] al problema de *k-sum lateness*, quien mostró que existen agendamientos óptimos para el problema de *k-sum tardiness* que son *parecidos* al agendamiento que entrega la regla *EDD*. Logramos para este caso un algoritmo de complejidad $\mathcal{O}(n^{2k-1})$. Conjeturamos en este capítulo que el resultado se puede generalizar a otras funciones objetivo monótonas no-decrecientes para las cuales la regla de despacho *EDD* sea óptima para el caso $k = 1$.

Realizamos aportes para la versión general del problema mostrando que existen agendamientos óptimos, crecientes y con lagunas consecutivas crecientes. También encontramos una consecuencia interesante de este resultado que nos permite recuperar la conclusión principal del Capítulo 2 de encontrar un algoritmo polinomial para el problema cuando k es constante. En el caso en que la cantidad D de *deadlines* distintos es constante, usamos los agendamientos óptimos, crecientes y con lagunas consecutivas crecientes en conjunto con la noción de *SPT* por pivote para mostrar que existen agendamientos óptimos que pueden ser particionados en bloques que en la tesis llamamos *lagunas* e *islas*. Dicha estructura nos permitió diseñar un algoritmo polinomial que encuentra una solución óptima para el problema en tiempo $\mathcal{O}(n^{3D+1}D)$. Una pregunta interesante a futuro es si es posible extender este resultado a otros problemas relacionados tales como *k-sum tardiness*, que es *NP-completo* [12].

Utilizando formulaciones dadas por Puerto [9] y modificaciones del algoritmo *pseudopolinomial* de Lawler [7] para minimizar *total tardiness*, proponemos un algoritmo polinomial cuando P, la cantidad de tiempos de proceso distintos, es constante, con tiempo $\mathcal{O}(n^{2P+5})$. La idea principal que usamos en este resultado fue notar que el problema de minimizar *total tardiness* puede ser resuelto en tiempo polinomial $\mathcal{O}(n^{P+4})$ modificando el algoritmo *pseudopolinomial* de Lawler. Sin embargo, dada la generalidad de la formulación, cualquier otra condición que permita resolver el problema de minimizar *total tardiness* en tiempo polinomial (no necesariamente P constante), permitiría resolver el problema de minimizar *k-sum lateness* en tiempo polinomial.

Finalmente, una línea de trabajo a futuro es estudiar la *NP-completitud* del problema de *k-sum lateness* para complementar los resultados de esta tesis, que en sí misma constituye un estudio robusto de casos particulares importantes de este problema.

Bibliografía

[1] J. Du and J.Y.T. Leung. Minimizing total tardiness on one machine is *NP-hard*. *Mathematics of Operations Research*, 15:483–495, 1990.

[2] S.K. Gupta and A.P. Punnen. k-sum optimization problems. *Operations Research Letters*, 9:121–126, 1990.

[3] C. A. R. Hoare. Find (algorithm 65). *Communications of the ACM*, 4:321–322, 1961.

[4] J.R. Jackson. Scheduling a production line to minimize maximum tardiness. *Research Report 43, Management Sciences Research Project, UCLA*, 1955.

[5] R. Karp. Reducibility among combinatorial problems. *Complexity of Computer Computations*, 40:85–103, 1972.

[6] E.L. Lawler. Optimal sequencing of a single machine subject to precedence constraints. *Management Science*, 19(5):544–546, 1973.

[7] E.L. Lawler. A pseudopolynomial algorithm for sequencing jobs to minimize total tardiness. *Annals of Discrete Mathematics*, 1:331–342, 1977.

[8] E.L. Lawler. A fully polynomial approximation scheme for the total tardiness problem. *Operations Research Letters*, 1(6):207–208, 1982.

[9] J. Puerto, A. Rodríguez-Chía, and A. Tamir. Revisiting k-sum optimization. *Mathematical Programming*, 165:1–26, 2016.

[10] A.P. Punnen and Y.P. Aneja. On k-sum optimization. *Operations Research Letters*, 18(5):233–236, 1996.

[11] W.E. Smith. Various optimizers for single-stage production. *Naval Research Logistics Quarterly, 3(1-2)*, 3:59–66, 1956.

[12] G. Woeginger. On minimizing the sum of k tardiness. *Information Processing Letters*, 38:253–256, 1991.