UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MORELOS

INSTITUTO DE INVESTIGACIÓN EN CIENCIAS BÁSICAS Y APLICADAS

CENTRO DE INVESTIGACIÓN EN CIENCIAS

Casos especiales solucionables en tiempo polinomial del problema de calendarización con tiempos de liberación y fechas límites en una máquina

TESIS

QUE PARA OBTENER EL GRADO DE

DOCTOR EN CIENCIAS

PRESENTA:

Maria Elisa Chinos Olivan

Director de Tesis:

Dr. Nodari Vakhania

Cuernavaca, Morelos Diciembre 2018

Esta tesis fue evaluada por el siguiente jurado:

Presidente:	Dr. Raúl Monroy Borja
Secretario:	Dr. Antonio Daniel Rivera López
Vocal:	Dr. Federico Alonso Pecina
Vocal:	Dr. Rodrigo Palacios Saldaña
Vocal:	Dr. José Cripín Zavala Díaz
Suplente:	Dra. Larissa Sbitneva
Suplente:	Dr. Nodari Vakhania

II

Lista de publicaciones relacionadas con esta tesis

1. N. Vakhania, E. Chinos and C. Zavala. An efficient Heuristic for a discrete optimization problem. Journal of Computer Science Technology Updates. Volume 2 Issue 2, Pages 38-45. DOI: http://dx.doi.org/10.15379/2410-2938.2015.02.02.05. ISSN (online): 2410-2938. http://www.cosmosscholars.com/current-issue-jcstu/79-abstracts/jcstu/483-abstract-an-efficient-heuristic-for-a-discrete-optimization-problem. 2015.

2. E. Chinos and N. Vakhania. Polynomially Solvable and NP-hard Special Cases for Scheduling with Heads and Tails. RECENT ADVANCES in MATHEMATICS and COMPUTATIONAL SCIENCE. (MCSS 16), p.141-145. ISBN: 978-1-61804-367-2 (http://www.wseas.us/elibrary/conferences/2016/barcelona/MCSS/MCSS-17.pdf) 2016.

3. E. Chinos and N. Vakhania. Scheduling jobs with two release times and tails on a single machine. INTERNATIONAL JOURNAL OF MATHEMATICAL MODELS AND METHODS IN APPLIED SCIENCES. Volume 10, p.303-3089. ISSN: 2074-1278 http://www.naun.org/main/NAUN/ijmmas/2016/a782001-aan.pdf. 2016.

4. E. Chinos and N. Vakhania. Adjusting scheduling model with release and due dates in production planning. Cogent Engineering. 4:1,DOI:10.1080/23311916.2017.1321175.2017.

Resumen

Los problemas de *optimización combinatoria* (CO, por sus siglas en inglés) constituyen una clase relevante de problemas prácticos con una naturaleza discreta.

Los problemas de CO se pueden clasificar en dos clases: los problemas P, los cuales se pueden resolver en tiempo polinomial con respecto al tamaño de una instancia, y los problemas NP-duro. Existen algoritmos eficientes para los problemas de la clase P, y se piensa que no existen algoritmos eficientes para los problemas NP-duro.

El problema de calendarización que estudiamos en este proyecto de investigación es de la clase de los problemas de *optimización combinatoria*. El término de calendarización se refiere a la asignación de un conjunto de solicitudes sobre un conjunto dado de recursos a lo largo del tiempo con el objetivo de optimizar un criterio objetivo dado. Las solicitudes son llamadas *trabajos* o *tareas* y los recursos son llamados *máquinas* o *procesadores*, mientras que el objetivo es elegir el orden de procesamiento de los trabajos sobre las máquinas así como cumplir con un criterio objetivo dado.

Diferentes características de los trabajos y de las máquinas junto con diferentes criterios de optimalidad dan origen a una gran cantidad de problemas de calendarización.

El problema de calendarización básico que consideramos en este proyecto de investiagación es el siguiente: n trabajos tienen que ser calendarizados en una máquina. Cada trabajo i llega a ser disponible en su *tiempo de liberación* o *cabeza* r_i, y necesita un tiempo de *procesamiento continuo* p_i sobre la máquina. La máquina sólo puede procesar un trabajo a la vez. Una vez que el trabajo j es completado este trabajo todavía necesita un tiempo de entrega (constante) q_i para su completés total (los trabajos son entregados

por una unidad independiente y este no toma tiempo en la máquina). Todos estos parámetros son enteros. Nuestro objetivo es encontrar una secuencia de trabajos sobre la máquina que minimice el máximo tiempo de completés total.

Garey y Johnson en 1979 demostraron que el problema de calendarizacin que consideramos en este trabajo denotado por $1|r_j, q_j|C_{\text{máx}}$ es NP-duro en el sentido estricto. Un método aproximado para solucionar el problema $1|r_j, q_j|C_{\text{máx}}$ es la llamada heurística extendida de Jackson (JE-heurística), dada por Schrage en 1971. La JE-heurística iterativamente, en cada tiempo de calendarización t (dado por el tiempo de liberación o completés de un trabajo), entre los trabajos liberados en este tiempo t calendariza uno con el mayor tiempo de entrega.

Explorando las propiedades estructurales inherentes del problema, derivamos nuevas condiciones de optimalidad cuando algunos casos prácticos del problema $1|r_j, q_j|C_{\text{máx}}$ pueden resolverse de manera óptima en un tiempo polinomial. En particular, proporcionamos condiciones explícitas que conducen a una solución eficiente del problema $1|r_j, q_j|C_{\text{máx}}$ mediante una mera aplicación de la JE-heurística. Además estudiamos otras propiedades estructurales útiles de los JE-calendarios (los creados por la JE-heurística) que conducen a la solución óptima de otras versiones de nuestro problema con el mismo costo computacional que el de la JE-heurística.

Finalmente, nos enfocamos en un caso especial de nuestro problema con solo dos tiempos permitidos de liberación de los trabajo r_1 y r_2 con $r_1 < r_2$ (denotado como $1|\{r_1, r_2\}, \{d_1, d_2\}|L_{\text{máx}}$), demostramos que este problema es NP-duro. Para este problema, también buscamos reglas de dominio estrictas y condiciones de optimalidad que se pueden verificar en tiempo polinomial.

Abstract

Combinatorial optimization (CO) problems constitute a significant class of practical problems with a discrete nature.

A CO problem is characterized by a finite set of the so-called feasible solutions, defined by a given set of restrictions, and an objective function for these feasible solutions, which typically needs to be optimized, i.e., minimized or maximized: the problem is to find an optimal solution, that is, one minimizing the objective function.

The CO problems are partitioned into two basic types, type P , which are polynomially solvable ones, and $NP-hard$ problems. Intuitively, there exist efficient (polynomial in the size of the problem) solution methods or algorithms for the problems from the first class, whereas no such algorithms exist for the problems of the second class.

The scheduling problem that we study in this project is of the class of problems of *combinatorial optimization.*The term *scheduling* refers to the assignment of a set of requests to the given set of resources over time with the objective to optimize a given objective criterion. The requests are called *jobs* or *tasks* and a resources are called *machines* or *processors*, whereas the aim is to choose the order of processing the jobs on the machines so as to meet a given objective criteria. Different characteristics of jobs and machines together with different optimality criteria originate a vast amount of the scheduling problems.

A basic scheduling problem that we consider in this thesis is as follows: n jobs have to be scheduled on a single machine. Each job j becomes available at its release time r_j . A released job can be assigned to the machine that has to process job j for p_j time units. The machine can handle at most one job at a time. Once it completés j this job still needs a (constant) delivery time q_j

for its full completion (the jobs are delivered by an independent unit and this takes no machine time). All above parameters are integers. Our objective is to find a job sequence on the machine that minimizes the maximum job full completion time.

Garey and Johnson in 1979 showed that our scheduling problem denoted by $1|r_j, q_j|C_{\text{máx}}$ it is NP-hard in the strict sense.

An approximate method to solve this problem is the so called extended Jackson heuristic (JE-heuristic), given by Schrage in 1971. The JE-heuristic iteratively, at each scheduling time t (given by job release or completion time), among the jobs released by time t schedules one with the the largest delivery time (or smallest due-date).

Exploring the inherent structural properties of the problem, here we derive new optimality conditions when practical special cases of our problem can be solved optimally in a low degree polynomial time. In particular, we provide explicit conditions that lead to an efficient solution of the problem by a mere application of J-heuristic. Then we study further useful structural properties of the J-schedules (ones created by J-heuristic) leading to the optimal solution of other versions of the problem with the same computational cost as that of J-heuristic.

Finally, we focus on a special case of our problem with only two allowable job release times r_1 and r_2 with $r_1 < r_2$ (abbreviated $1|\{r_1, r_2\}, \{d_1, d_2\}|L_{\text{máx}}$). Although the latter problem remains NP-hard, it admits stricter dominance rules and optimality conditions leading to the corresponding polynomial-time verification procedures (and the reduction of the search space).

A mi familia.

Agradecimientos

Primeramente quiero darle gracias a mi madre por todo su apoyo, sus cuidados y consejos, ella ha sido una pieza importante para lograr todas mis metas. Gracias a Dios y a ella hoy puedo culminar con esta etapa de mi vida.

Gracias a mis hermanos, simplemente no sé que haría sin ellos.

Quiero agradecer al Dr. Nodari Vakhania, por su tiempo, asesoramiento y rigor científico. Sin su esfuerzo y compromiso con el proyecto, simple y sencillamente no hubiera sido posible la realización del presente trabajo de investigación.

A mi comité tutoral, mi más sincero agradecimiento por todo su apoyo y compresión.

A mi jurado evaluador, les agradezco profundamente su tiempo y dedicacin para corregir el manuscrito final y fungir como miembros de mi jurado evaluador.

Agradezco al Consejo Nacional de Ciencia y Tecnología, CONACyT, por el soporte económico otorgado a lo largo de este periodo de estudios a través de la beca doctoral con número 258736.

A la Coordinación de Estudios de Posgrado en Ciencias de la UAEM.

A mis amigos y amigas que han estado presente a lo largo de esta etapa de mi vida, gracias por su cariño, comprensión y su apoyo incondicional, soy muy afortunada por tenerlos.

Contenido

Capítulo 1

Introducción

Es común en nuestra vida cotidiana el deseo de optimizar el tiempo, hacer nuestros deberes en un tiempo determinado o arreglar tantas cosas como sea posible en un tiempo limitado. A veces la optimización de nuestro tiempo no es un asunto puramente personal, porque depende de los factores y el entorno que nos rodea. Deseamos organizar este proceso de una manera que sea beneficiosa para nosotros. Como ejemplo, damos un problema con el que muchos de nosotros nos enfrentamos diariamente. Una familia con n miembros vive en un departamento que cuenta con una taza de baño, una regadera, una plancha y un espejo. Cada miembro de la familia se levanta diariamente a una hora fija, cada uno necesita un tiempo determinado en la taza de baño, en la regadera, para planchar y frente al espejo. La taza de baño, la regadera, la plancha y el espejo son nuestros recursos y pueden ser utilizados por un máximo de una persona a la vez. Dado que cada miembro de la familia tiene que salir de la casa en una hora determinada (para llegar al trabajo/escuela), nos enfrentamos al problema de optimización de cómo organizar las actividades de cada miembro de la familia en cada uno de los recursos, para minimizar el tiempo total necesario para que todos los miembros de la familia cumplan con todas sus actividades. Estos tipos de problemas se tratan en optimización discreta.

1.1. Problemas de optimización combinatoria

Los problemas de *optimización combinatoria* (CO) constituyen una clase significante de problemas prácticos con una naturaleza discreta. Estos surgieron a finales de los años 40 del siglo XX. Con un rápido crecimiento en la industria, se ha incrementado la demanda de soluciones óptimas para nuevos problemas de manejo de recursos y de logística. Para el desarrollo de métodos de solución efectiva, estos problemas son formalizados y modelados matemáticamente.

Un problema CO se caracteriza por contar con un conjunto finito de *soluciones factibles*, definidas por un conjunto de restricciones, y una *función objetivo* que debe minimizarse (o maximizarse) para las soluciones factibles. Entonces, el problema es encontrar una *solución óptima*, esto es, una que minimice la función objetivo. Como el número de soluciones factibles de un problema de optimización combinatoria normalmente es finito, teóricamente, encontrar una solución óptima es trivial: basta enumerar todas las soluciones factibles, calculando para cada una de ellas la función objetivo y seleccionando alguna con un valor objetivo óptimo. Desafortunadamente, una enumeración exhaustiva de todas las soluciones factibles normalmente es imposible en la práctica (a excepción de problemas muy pequeños). Aún para problemas con un tamaño moderado (digamos, 30 ciudades para el problema del agente viajero clásico o 10 trabajos sobre 10 máquinas en el problema de calendarización job-shop), tal enumeración completa puede tardar cientos de siglos en una computadora moderna. Además, esta situación no se puede cambiar aunque en el futuro se desarrollen computadoras más rápidas.

Los problemas CO se pueden clasificar en dos clases: problemas P, que se pueden resolver en un tiempo polinomial con respecto al tamaño de una instancia y problemas NP-duro. Existen algoritmos eficientes para los problemas de la clase P, e intuitivamente se piensa que no existen algoritmos eficientes para los problemas NP-duro. Informalmente hablando, el tamaño de una instancia es la cantidad de memoria necesaria para representar los datos y parámetros de un problema en particular. El número de soluciones factibles de un problema de optimización NP-duro crece exponencialmente con respecto al tamaño de la instancia. Más aún, todos los problemas NP-duro tienen una *complejidad computacional (tiempo)* similar entre ellos, en el sentido de que si se encontrara un algoritmo en tiempo polinomial eficiente para cualquiera de ellos, tal algoritmo podría producir otro algoritmo en

tiempo polinomial para cualquier problema NP-duro. Todos los problemas NP-duro pertenecen a la clase NP, lo cual garantiza que cualquier solución factible a un problema NP-duro puede ser verificada en un tiempo polinomial. En el mismo tiempo, se cree que es muy poco probable que un problema NP-duro pueda ser resuelto en tiempo polinomial. Un problema es NP-duro en el sentido estricto si aún con representación unaria de los parámetros del problema siguen siendo NP-duro. En código unario todos los datos numéricos son codificados por cadenas de unos (más específicamente un entero d es representado por una secuencia de d unos).

1.2. Problemas de calendarización

Los *problemas de calendarización* son problemas importantes de optimización combinatoria. El término de calendarización se refiere a la asignación de un conjunto de solicitudes sobre un conjunto dado de recursos a lo largo del tiempo con el objetivo de optimizar un criterio objetivo dado. Las solicitudes son llamadas *trabajos* o *tareas* y los recursos son llamados *máquinas* o *procesadores*, mientras que el objetivo es elegir el orden de procesamiento de los trabajos sobre las máquinas así como cumplir con un criterio objetivo dado. Por lo tanto, un problema de calendarización se caracteriza por tres componentes distintos: las tareas o actividades que se desean realizar, los recursos disponibles para su implementación, y los objetivos que se quieren lograr (que identifican la mejor solución). Diferentes características de los trabajos y las máquinas junto con diferentes criterios de optimalidad dan origen a una gran cantidad de problemas de calendarización.

Un problema de calendarización básico que consideramos en este proyecto de investigación es el siguiente: n trabajos tienen que ser calendarizados en una máquina. Cada trabajo i llega a estar disponible en su *tiempo de liberación*, o *cabeza* r_i, y necesita un tiempo de *procesamiento continuo* p_i sobre la máquina. La máquina sólo puede procesar un trabajo a la vez. Una vez que el trabajo j es completado este trabajo todavía necesita un tiempo de entrega q_i (constante) para su completés total (los trabajos son entregados por una unidad independiente y este no toma tiempo en la máquina). Todos estos parámetros son enteros. Nuestro objetivo es encontrar una sucesión de trabajos sobre la máquina que minimice el tiempo máximo de completés total.

El problema es conocido como NP-duro en el sentido estricto (Garey and Johnson en [41]). De acuerdo a la notación convencional de tres campos introducida por Graham [58], el problema bajo investigación es abreviado como $1|r_j, q_i|C_{\text{máx}}$: el primer campo indica la máquina (solo un procesador), el segundo campo especifica los parámetros de los trabajos, y en el tercer campo se da el criterio objetivo. En el segundo campo el tiempo de procesamiento es omitido ya que este es un parámetro obligatorio para cualquier problema de calendarización.

El problema $1|r_j, q_j|C_{max}$ se considera como uno de los primeros problemas de optimización combinatoria NP-duros, y ha sido muy usado en muchos procedimientos de solución para varios problemas de calendarización más complejos (Lancia [14]). Además, el problema $1|r_j, q_j|C_{max}$ juega un papel central en la vida real en algunas aplicaciones en la industría (Sourirajan y Uzsoy [32]).

Existe una formulación equivalente del problema $1|r_j, q_i|C_{\text{máx}}$, donde el tiempo de entrega q_i de cada trabajo i es remplazado por una *fecha límite* d_i, que es el tiempo deseado de completés del trabajo i. Aquí la *tardanza* máxima de cualquier trabajo $L_{\text{máx}}$ (a saber, la diferencia entre el tiempo de completés del trabajo y su fecha límite) debe ser minimizada.

Cabe destacar, que además de las tardanzas de los trabajos, existen otros criterios objetivos orientados con fechas límite. Uno de estos criterios es minimizar el número de trabajos tardíos, donde el trabajo es tardío si es completado después de su fecha límite. En la versión de factibilidad del problema se busca un calendario con trabajos no tardíos. Obviamente, si en una solución óptima de la versión de minimización de la máxima tardanza de cualquier trabajo no es más que cero, entonces esta es una solución factible de la versión de factibilidad también; o de otra manera (la tardanza máxima de cualquier trabajo es positiva), aquí no existe una solución factible para la versión de factibilidad. Vice-versa, un algoritmo para el problema de factibilidad puede ser usado para resolver la versión de minimización: iterativamente incrementamos las fechas límite de todos los trabajos hasta encontrar un calendario factible con fechas límite modificadas.

Dada una instancia del problema $1\,|r_i,\, q_i|\, C_{max}$, podemos obtener una instancia equivalente del problema $1\,|r_i|\, L_{max}$ de la siguiente manera: tomamos una constante adecuada suficientemente grande K (no menos que la cola del trabajo con cola máxima) y definimos la fecha límite de cada trabajo i como

$d_i = K - q_i$. Vice-versa, dada una instancia del problema $1\,|r_i|\,L_{max}$, una instancia equivalente del problema $1\,|r_i,\, q_i|\,C_{max}$ puede ser obtenida, pero definiendo la cola de cada trabajo i como $q_i = D - d_i$, donde D es una constante apropiada (no menos que la fecha límite del trabajo con fecha límite máxima). En la figura 1.1 damos un ejemplo de la construcción de una instancia con fechas límite a partir de una instancia con tiempos de entrega. Se puede ver que el par de instancias definidas de esta manera son equivalentes; es decir, siempre que el tiempo máximo de completés total para la versión $1\,|r_i,\, q_i|\,C_{max}$ es minimizado, la tardanza máxima del trabajo en $1\,|r_i|\,L_{max}$ es minimizada, y vice-versa (ver Bratley en [57] para más detalles).

$r_1 = 0$	$p_1 = 5$	$q_1 = 4$
$r_2 = 0$	$p_2 = 10$	$q_2 = 0$
$r_3 = 11$	$p_3 = 1$	$q_3 = 25$
$r_4 = 11$	$p_4 = 4$	$q_4 = 7$

Sea $D = 30$.

$r_1 = 0$	$p_1 = 5$	$d_1 = 30 - 4 = 26$
$r_2 = 0$	$p_2 = 10$	$d_2 = 30 - 0 = 30$
$r_3 = 11$	$p_3 = 1$	$d_3 = 30 - 25 = 5$
$r_4 = 11$	$p_4 = 4$	$d_4 = 30 - 7 = 23$

Figura 1.1: Ejemplo de la transformación de una instancia con tiempos de entrega a fechas límites.

1.3. Métodos de solución

Existen algoritmos exactos que se ejecutan en tiempo polinomial para problemas de la clase P, pero es poco probable que existan tales algoritmos para un problema NP-duro. De aquí, se deriva el enfoque de resolver un problema NP-duro de forma aproximada.

Los algoritmos voraces y heurísticos son de los algoritmos aproximados más simples. Un algoritmo *heurísticos* puede encontrar la solución exacta de un problema en la clase P y una solución aproximada de un problema NP-duro. Un algoritmo heurístico es un algoritmo de tiempo polinomial que crea una o más soluciones factibles. Un algoritmo *voraz* crea una solución en n iteraciones del ciclo externo, donde n es el número de objetos de la instancia

correspondiente. Debido a que el tamaño de un problema de optimización discreta es un polinomio de n, el número de iteraciones en un algoritmo voraz es también polinomial con respecto al tamaño del problema. Tal algoritmo crea una solución factible completa iterativamente, añadiendo un elemento a la solución parcial actual tomando esta decisión final en cada iteración. En este camino, el espacio de búsqueda se reduce aadiendo un solo elemento en cada paso. Esto reduce drásticamente el espacio de búsqueda posible. Esta reducción del tamaño de búsqueda puede conducir a la construcción de una solución muy lejos de la óptima.

Los algoritmos voraces y heurísticos puede crear una solución óptima para un problema de la clase P (aunque no cualquier problema en P puede ser resuelto de manera óptima por un algoritmo voraz). Sin embargo, éste no es el caso para un problema NP-duro. No existe un algoritmo heurístico/voraz que pueda resolver de forma óptima un problema NP (a menos que P = NP, lo cual es poco probable). Debido a que la mayoría de los problemas de optimización combinatoria son NP-duro, frecuentemente no se puede evitar aceptar una solución que no es óptima. En este camino, es natural y práctico pensar acerca del diseño y análisis de *algoritmos aproximados* de tiempo polinomial, es decir, unos que entregan una solución con un alejamiento garantizado de una óptima en tiempo polinomial. Un algoritmo voraz aproximado no garantiza la mejor aproximación posible, pero tiene la ventaja que es simple en su implementación y corre más rápido.

El *radio de ejecución* de un algoritmo aproximado A, mide la calidad de este algoritmo aproximado. Un *algoritmo k-aproximado* es un algoritmo en el cual en el peor caso tendrá un radio de aproximación k del óptimo. Otra medida usada comúnmente es el error absoluto, que es la diferencia entre el valor de la función objetivo obtenida por el algoritmo A y valor de la solución óptima. Los algoritmos heurísticos y aproximados son importantes y comunes para los problemas de calendarización, los cuales forman una parte considerable de los problemas de optimización combinatoria.

Como ya mencionamos, los algoritmos heurísticos no garantizan la solución óptima. Cuando es importante determinar una solución óptima del problema; es necesario recurrir a diferentes algoritmos. Hay enfoques posibles para determinar la solución exacta de un problema de optimización combinatoria NP-duro; entre ellos, los de enumeración implícita.

En un problema de optimización combinatoria las técnicas enumerativas, enu-

meran o listan todas (enumeración explícita) o algunas (enumeración implícita) de las posibles soluciones parciales de un problema. En la enumeración implícita, se intenta eliminar de la enumeración explícita, un conjunto de soluciones parciales que no producirán soluciones completas óptimas. Estas soluciones parciales, deben encontrarse mediante algún procedimiento complejo matemático, que garantice la no exclución de alguna solución óptima contenida en la solución parcial. No todas las técnicas enumerativas pueden realizar procedimientos de poda de soluciones de la misma forma. El modelo matemático usado no siempre es el mismo y este influye significativamente en el proceso de selección de soluciones parciales.

Toda solución factible en un problema de calendarización, puede ser clasificada como una solución parcial o como una solución completa. Las técnicas enumerativas construyen un árbol de soluciones T que va siendo obtenido por un algoritmo solución, que enumera las soluciones parciales o completas.

Dicho árbol de soluciones T, cumple un rol fundamental en el proceso de encontrar la solución óptima del problema, a partir de seleccionar la mejor solución completa entre todas las obtenidas. Cada nodo h de T, representa una solución parcial o completa en el caso de ser el nodo h una hoja (nodo terminal de una rama del árbol) de T. La ráiz de T es conciderada un caso especial, en el cual están contenidas todas las soluciones parciales o completas del problema. Un algoritmo enumerativo que construye a T, pasa por determinados estados h (nodos del árbol), en el proceso de encontrar la solución del problema. Cada estado h repesenta una solución parcial o completa σ_h y a su vez define el subárbol de soluciones parciales o completas que pueden ser obtenidas a partir de h.

Dentro de las técnicas heurísticas podemos mencionar el método de Búsqueda en Haz (Beam Search en ingles). Al igual que las técnicas enumerativas, este método enumera las soluciones, creando un árbol de soluciones T, la diferencia radica en que el método de Búsqueda en Haz básicamente trunca el árbol de soluciones T, permitiendo la exploración en T de sólo los k nodos más prometedores en cada nivel l de T.

La búsqueda de haz utiliza la búsqueda de amplitud para crear su árbol de búsqueda. En cada nivel del árbol genera todos los sucesores de los estados en el nivel actual, clasificándolos en orden creciente de costo heurístico. Sin embargo, solo almacena un número predeterminado, β, de los mejores estados en cada nivel (llamado ancho del haz). Solo esos estados se expanden a

continuación. Cuanto mayor sea el ancho del haz, menos estados se podan. Con un ancho de haz infinito, no se podan los estados y la búsqueda de haces es idéntica a la búsqueda de amplitud. El ancho del haz limita la memoria requerida para realizar la búsqueda. Dado que un estado de objetivo podría ser podado, la búsqueda de haz sacrifica la eficiencia (la garantía de que un algoritmo terminará con una solución, si existe). La búsqueda de haces no necesariamente es óptima (es decir, no hay garantía de que encuentre la mejor solución). Devuelve la primera solución encontrada.

1.4. Estado del arte

Siendo el problema $1|r_j, q_j|C_{max}$ NP-duro en el sentido estricto, ha sido extensamente estudiado en la literatura de calendarización; en esta sección mencionamos los trabajos relevantes que existen en el estudio de este problema.

En la subsección 1.4.1 hacemos mención de los trabajos enfocados en identificar casos del problema $1|r_j, q_j|C_{max}$ que tienen solución polinomial. En la subsección 1.4.2 citamos algunos algoritmos y esquemas de aproximación diseñados para el problema $1|r_j, q_j|C_{max}$. Y finalmente en la sección 1.4.3 mencionamos los trabajos relacionados con métodos exactos para solucionar el problema $1|r_j, q_j|C_{max}$.

1.4.1. Casos especiales que se solucionan en tiempo polinomial

Varios autores se han enfocado en identificar casos del problema $1|r_j, q_j|C_{max}$ que tienen una solución polinomial. Jackson y Smith en 1955 y 1956 respectivamente, fueron los primeros en enfocarse en el estudio de este problema. Jackson en [30] probó que cuando las fechas de liberación de los trabajos son iguales, calendarizando los trabajos en orden no creciente de sus colas, el problema se soluciona en forma óptima en $O(n\,log\,n)$. Similar-mente, si todas las colas son iguales, entonces calendarizando los trabajos en orden no creciente de sus tiempos de liberación se obtiene un calendario óptimo.

Unos años más tarde, en 1978, Dessouky y Larson en [37] mostraron que los siguientes casos especiales se solucionan en tiempo polinomial:

- $r_i \geq r_j + p_j$ o $r_j \geq r_i + p_i$ para todos los pares de trabajos i y j.
- $q_i \geq q_j + p_j$ o $q_j \geq q_i + p_i$ para todos los pares de trabajos i y j.
- $\text{máx}_j \ r_j < \text{mín}_j \ (r_j + p_j)$ y $\text{máx}_j \ q_j < \text{mín}_j \ (q_j + p_j)$.

Para el problema $1|r_j|L_{\text{máx}}$, Smith en [62] estudió un caso particular con la restricción que cada trabajo debe ser completado en su fecha límite. Mostró que la tardanza máxima es minimizada si los trabajos son ordenados de cuerdo a sus fechas límite. En 1974, Horn en [61] dio algunos algoritmos simples para encontrar calendarios óptimos para minimizar la tardanza máxima para el caso en una sola máquina. Mostró que si todos los tiempos de procesamientos son iguales a uno y todos los tiempos de liberación son enteros, entonces el problema se soluciona en forma óptima en $O(n \ log \ n)$ tiempos por el algoritmo de Schrage: siempre que un trabajo llega a estar disponible, calendariza un trabajo disponible con la cola más larga (Schrage, 1971). Frederickson [16] mejoró el resultado de Horn y proporcionó un algoritmo en tiempo lineal.

En 1981, M.R. Garey et al. en [42], propusieron un algoritmo polinomial cuando los tiempos de procesamiento de todos los trabajos son igual a una constante P. Si todos los tiempos de liberación, tiempos de procesamiento y fechas límite son restringidos de tal modo que cada r_j está en el intervalo $[d-d_j-p_j-a, \ d-d_j-a]$, para alguna constante a y una constante adecuada d, entonces el problema se puede solucionar en tiempo $O(n \log n)$, ver Hoogeven [61] (1995). Entre algunos trabajos relevantes de N. Vakhania mencionamos los siguientes: en 2004 N. Vakhania [45] propuso un algoritmo $O(n^2 log \, n)$ para el caso cuando los tiempos de procesamiento de los trabajos son restringidos a P o $2P$. Para otro criterio relativo, N. Vakhania en [46] (2009) dio un algoritmo de complejidad $O(n^3 \log n)$ que minimiza el número de trabajos tardíos con tiempos de liberación sobre una sola máquina cuando es permitido interrumpir los trabajos. En el 2013, N. Vakhania y F. Wener [50], dieron una solución en tiempo polinomial de la versión en la cual el tiempo de procesamiento máximo de cualquier trabajo y la diferencia entre los tiempos de liberación de los trabajos son acotados por una constante. En este mismo año, N. Vakhania dio un algoritmo de tiempo polinomial para trabajos de igual longitud sobre una sola máquina de complejidad $O(n^2 \log n)$.

En el 2016, en [52], N. Vakhania construyó un algoritmo polinomial que encuentra una solución óptima cuando los tiempos de procesamiento de los

trabajos son mutuamente divisibles.

En N. Vakhania y F. Pecina [54] (2017) describieron un algoritmo heurístico para el problema de calendarización con tiempos de liberación y fechas límites con dos objetivos: minimizar la tardanza máxima y encontrar un calendario factible en el cual el trabajo con la tardanza máxima es la mínima posible entre todos los calendarios factibles. Su algoritmo está basado en la idea de dividir el conjunto entero de trabajos sobre dos categorías básicas conteniendo los trabajos urgentes y no urgentes.

Tian et al. [65] (2006) estudiaron el problema de calendarización con interrupciones de n trabajos con igual longitud sobre una sola máquina para minimizar la tardanza total sujeto a sus tiempos de liberación. Ellos dieron un algoritmo de complejidad $O(n^2)$ para resolver el problema.

Además, se ha mostrado que el problema $1|r_j, q_j|C_{\text{máx}}$ es útil para solucionar problemas de calendarización con multiprocesadores. Por ejemplo, para la versión de factibilidad con m máquinas idénticas y trabajos de igual longitud, algoritmos con tiempos de complejidad $O(n^3 \log\log n)$, $O(n^2 m)$ y $O(n^3 \log n \log p_{\text{máx}})$ fueron propuestos por Simons [62] (1983), Simons & Warmuth [5] (1989) y N. Vakhania (2012) respectivamente. Pessan et al. [7] (2008) consideró el caso del problema con máquinas paralelas no relacionadas. Probó que si $r_j \leq r_j + 1$ y $q_j \leq q_j + 1$ para todo $j = 1, \ldots, n$, entonces el problema se soluciona en tiempo polinomial $o(n\ log\ n)$.

1.4.2. Algoritmos y esquemas de aproximación

Entre los primeros trabajos relacionados con la construcción de algoritmos de aproximación para el problema $1|r_j, q_j|C_{\text{máx}}$, se encuentra el trabajo de Kise et al. [18] (1979). Ellos estudiaron el desempeño de seis algoritmos de aproximación incluido el algoritmo de Schrage. Concluyeron que uno de estos algoritmos es eficiente, y da una buena solución aproximada para el problema $1|r_j|L_{\text{máx}}$. Recomiendan este algoritmo para usos prácticos.

Otro trabajo relevante, es el trabajo de Hall y Shmoys [33] (1992). Ellos construyeron un algoritmo de aproximación 4/3 con complejidad $O(n^2 \log n)$.

Kacem y Kellerer [19] (2014) diseñaron un algoritmo de aproximación para el problema en una sola máquina con el objetivo de minimizar la tardanza máxima cuando los trabajos tienen diferente tiempos de liberación y tiempos

de entrega bajo la suposición que no hay tiempo de inactividad, es decir, el calendario no puede tener ningún tiempo de inactividad entre dos trabajos consecutivos en la máquina. Su trabajo fue motivado por aplicaciones industriales interesantes en el área de producción.

En un trabajo reciente del 2018 de Kacem y Kellerer [20], consideraron cuatro problemas de calendarización en una sola máquina con tiempos de liberación. Para cada uno de los cuatro problemas establecieron la existencia de un esquema de aproximación en tiempo polinomial. N. Vakhania en [55] (2018), describió un esquema de aproximación en tiempo polinomial simple para el problema de calendarización en una sola máquina donde los trabajos no son liberados simultáneamente y con el objetivo de minimizar la tardanza máxima.

1.4.3. Métodos exactos

Diseñar algoritmos exactos para solucionar el problema $1|r_j, q_j|C_{max}$ ha sido de gran interés para muchos investigadores desde los primeros trabajos de Dessouky y Margenthaler [38](1972), Baker y Su [31] (1974), McMahon & Florian [15] (1975), Lageweg et al. [3] (1976) y Carlier [23] (1982).

Otros trabajos relevantes relacionados con métodos exactos basados en técnicas de enumeración fueron propuestos por Grabowski et al. en [27] (1986), Shantikumar en [28] (1983) y Gupta et al en [29] (1999). Grabowski et al. en [27] propusieron un algoritmo nuevo de tipo corte y poda para solucionar el problema de calendarización de trabajos sobre una máquina con tiempos de liberación y fechas límite. Baptiste et al [56] presentaron un algoritmo de tipo corte y poda para el problema de calendarización en una sola máquina donde el objetivo es minimizar el número de trabajos tardíos. Shantikumar en [28] presentó un algoritmo de corte y poda para obtener el calendario óptimo que minimice la tardanza máxima con un número mínimo de trabajos tardíos. Gupta et al en [29] desarrollaron un algoritmo de corte y poda para solucionar el problema de caledarización en una sola máquina con el objetivo de minimizar la tardanza máxima de cualquier trabajo, sujeto a la restricción que el número de trabajos tardíos es mínimo. Las pruebas computacionales mostraron que el algoritmo que propusieron es efectivo en resolver problemas con hasta 100 trabajos. Algunos algoritmos más recientes para intentar solucionar el problema $1|r_j, q_j|C_{max}$ fueron diseñados por Sadykov y Lazarev

[59] (2005), Pan y Shi [63] (2006) y Briand et al. [6] (2008).

[23], Carlier y Pinson [24] y Brinkkotter y Brucker [60] y trabajos más recientes de Gharbi y Labidi [2] y Della Croce y T'kindt [13]. Carlier y Pinson [25] han usado una sencilla generalización de la JE-heurística para la solución del problema de job-shop con multiprocesador con máquinas idénticas. Esto además puede ser adoptado para el caso cuando las máquinas paralelas no son relacionadas, ver [49]. La JE-heurística puede ser útil para que los cálculos sean paralelos en calendarizar job-shop (Perregaard and Clausen [40]), y también para los problemas de calendarización de lotes con tiempos de liberación Condotta et al. [1].

1.5. Resultados obtenidos

Tomando como referencia que el problema de calendarización $1|r_j, q_j|C_{max}$ es NP-duro en el sentido estricto, fijamos nuestro objetivo en encontrar nuevos casos que se pueden solucionar polinomialmente de este problema; estos casos son presentados en el capítulo 3.

Explorando las propiedades estructurales inherentes del problema $1|r_j, q_j|C_{max}$, derivamos nuevas condiciones de optimalidad cuando casos prácticos de este problema pueden resolverse de forma óptima en un tiempo polinomial (ver sección 3.1). En particular, proporcionamos condiciones explícitas que conducen a una solución eficiente de nuestro problema mediante sólo la aplicación de la JE-heurística. Además estudiamos otras propiedades estructurales útiles de los JE-calendarios (los creados por JE-heurística) que conducen a la solución óptima de otras versiones del problema $1|r_j, q_j|C_{max}$ con el mismo costo computacional que el de la JE-heurística.

Finalmente, nos enfocamos en un caso especial de nuestro problema con solo dos tiempos permitidos de liberación de los trabajo r_1 y r_2 con $r_1 < r_2$ y dos fechas límites d_1 y d_2 $(d_1 > d_2)$ (este problema es denotado como $1|\{r_1, r_2\}, \{d_1, d_2\}|L_{\text{máx}}$), demostramos que este problema es NP-duro (ver sección 3.2). Para el problema $1|\{r_1, r_2\}, \{d_1, d_2\}|L_{\text{máx}}$, también buscamos reglas de dominio estrictas y condiciones de optimalidad que se pueden verificar en tiempo polinomial; estas condiciones son presentadas en la sección 3.3.

Las condiciones de optimalidad que encontramos las incorporamos a un algoritmo de enumeración implícita; en el caso que se cumplan las condiciones, el algoritmo va a seguir las condiciones de propiedad correspondiente. Estas condiciones nos permitirán eliminar subconjuntos de soluciones implícitas en una búsqueda explícita.

En el siguiente capítulo damos algunos conceptos básicos y preliminares, así como describimos la heurística extendida de Jackson y mostramos que esta heurística es 2-aproximada. En el capítulo 3 estudiamos las condiciones que se pueden identificar eficientemente, y los casos cuando nuestro problema genérico puede ser solucionado en tiempo polinomial. Además mostramos que el caso especial del problema con sólo dos tiempos permitidos de entrega sigue siendo NP-duro, y damos relaciones de dominio estrictas y otras versiones particulares que pueden ser solucionadas eficientemente.

Capítulo 2

Preliminares

En este capítulo, en la sección 2.1 presentamos el problema de estudio de este proyecto de investigación. En la sección 2.2, describimos un método aproximado para solucionar el problema $1|r_j, q_j|C_{max}$; este método, es la llamada heurística extendida de Jackson ofrecida por Schrage en 1971. Como veremos en la sección 2.4, esta heurística, en el caso peor, da una solución dos veces peor que la solución óptima. En la sección 2.3 damos algunas definiciones y conceptos básicos que nos permitieron explorar las propiedades estructurales del problema.

2.1. El problema de estudio de este proyecto

El problema de calendarización que estudiamos en este proyecto de investigación es el siguiente: Consideramos n trabajos $i(i = 1, \ldots, n)$ y una máquina disponible desde el tiempo 0. Cada trabajo i llega a estar disponible en su *tiempo de liberación* o *cabeza* r_i. Necesita un tiempo de *procesamiento continuo* p_i sobre la máquina. Necesita un tiempo adicional después de su completés sobre la máquina, el *tiempo de entrega* o *cola* q_i (cabe notar que q_i no requiere tiempo en la máquina en nuestro modelo). Los valores de las cabezas y colas son números enteros no negativos, mientras que un tiempo de procesamiento es un entero positivo. Como mencionamos en la sección 1.2, el problema descrito anteriormente es denotado como $1|r_j, q_i|C_{\text{máx}}$.

Un *calendario factible* S asigna a cada trabajo i un tiempo de inicio $t_i(S)$,

tal que $t_i(S) \geq r_i$ y $t_i(S) \geq t_k(S) + p_k$, para cualquier trabajo k incluido más temprano que un trabajo i en S; la primera desigualdad dice que un trabajo no puede ser empezado antes de su tiempo de liberación, y la segunda refleja la restricción de que la máquina sólo puede procesar un trabajo a la vez. El *tiempo de completés* de un trabajo i, $c_i(S) = t_i(S) + p_i$; el *tiempo de completés total* de un trabajo i en S, $C_i(S) = c_i(S) + q_i$. Nuestro objetivo es encontrar un *calendario óptimo*, es decir, un calendario factible S con el valor mínimo del tiempo de completés total máximo $|S| = max_i C_i$ (llamado el *makespan*).

2.2. Heurística extendida de Jackson

Una de las heurísticas más antiguas y usadas comúnmente en la teoría de calendarización es la de Jackson. La heurística extendida de Jackson se usa ampliamente para construir soluciones factibles cuando se calendarizan trabajos con tiempos de liberación y colas en una sola máquina (o un grupo de máquinas paralelas). Además es una herramienta eficiente en la solución de problemas más complicados de calendarización en la industria. El problema original ocurre como un auxiliar y se utiliza para la obtención de estimaciones inferiores en algoritmos de enumeración implícita (sin embargo, incluso este último problema es fuertemente NP-duro). Al mismo tiempo, se sabe que en el peor de los casos, la heurística extendida de Jackson da una solución que es dos veces peor que la óptima.

La heurística extendida de Jackson, calendariza iterativamente en cada tiempo de calendarización t (dado por el tiempo de liberación o por el tiempo de completés de cualquier trabajo), entre todos los trabajos liberados al tiempo t uno con la cola más larga (o fecha límite más pequeña).

Una descripción más detallada de esta heurística es la siguiente: la JE-heurística distingue n instantes de calendarización (los momentos de tiempo en que un trabajo es asignado a la máquina). Inicialmente, el tiempo de calendarización más temprano, es ajustado al tiempo del trabajo con mínimo tiempo de liberación. Entre todos los trabajos liberados en ese tiempo, el trabajo con la cola más larga (la fecha límite mínima, alternativamente) se asigna a la máquina. Iterativamente, el siguiente tiempo de calendarización, es el máximo entre el tiempo de completés del último trabajo asignado en la máquina y el mínimo tiempo de liberación de los trabajos aún no asignados (como ningún trabajo se puede empezar antes de que la máquina esté activa

o antes de su tiempo de liberación). Y de nuevo, entre todos los trabajos liberados en este tiempo de calendarización un trabajo con la cola más larga (fecha límite mínima) se asigna a la máquina.

En las figuras 2.1 y 2.2 ilustramos la construcción del calendario generado por la JE-heurística para las versiones con colas y fecha límites respectivamente.

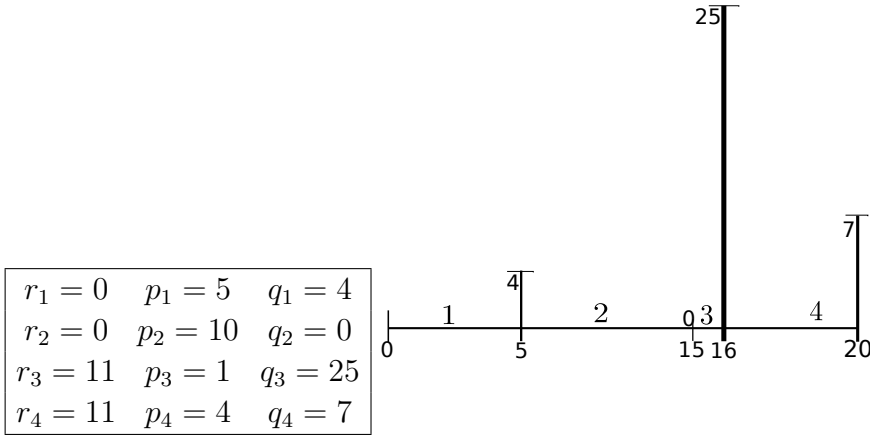


Figura 2.1: Ejemplo de una instancia con 4 trabajos y del calendario generado por la JE-heurística para esta instancia. Observemos que el tiempo de liberación mínimo de los trabajos de esta instancia es 0. Entonces, la JE-heurística escoge de los trabajos liberados a este tiempo el de mayor fecha de entrega y lo asigna. Los trabajos que están liberados al tiempo 0 de esta instancia, son los trabajos 1 y 2, entonces el primer trabajo asignado por la JE-heurística es el trabajo 1 con $q_1 = 4$ y tiempo de procesamiento $p_1 = 5$. Una vez que el trabajo 1 fue asignado, el tiempo actual en el calendario parcial es 5 éste está dado por el tiempo de completés del trabajo 1. Otra vez, de los trabajos liberados a este tiempo la JE-heurística calendariza el trabajo con tiempo de entrega máximo. En este caso, solamente el trabajo 2 está liberado al tiempo $t = 5$, entonces el siguiente trabajo calendarizado por la JE-heurística es el trabajo 2, este procedimiento se vuelve a repetir iterativamente. Las líneas verticales representan los tiempos de entrega de los trabajos.

$r_1 = 0$	$p_1 = 5$	$d_1 = 26$
$r_2 = 0$	$p_2 = 10$	$d_2 = 30$
$r_3 = 11$	$p_3 = 1$	$d_3 = 5$
$r_4 = 11$	$p_4 = 4$	$d_4 = 23$

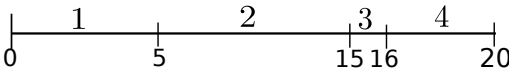


Figura 2.2: Ejemplo de una instancia con fechas límites y del calendario generado por la JE-heurística para esta instancia. El procedimiento de construcción del calendario, es similar al de la figura 2.1, pero en este caso, la JE-heurística, en lugar de escoger el trabajo con tiempo de entrega más grande, escoge el trabajo con menor fecha límite.

Observemos que la JE-heurística no crea huecos (intervalos de tiempo en el que ningún trabajo es calendarizado en la máquina) que puedan ser evitados calendarizando siempre un trabajo liberado listo una vez que la máquina llega a estar disponible, mientras haya trabajos liberados aún no calendarizados en cada tiempo de calendarización, la JE-heurística da prioridad a un trabajo más urgente (es decir, uno con fecha límite más pequeña o alternativamente con tiempo de entrega más largo).

Ya que el número de tiempos de calendarización es $O(n)$ y en cada tiempo de calendarización se busca un elemento mínimo/máximo de una lista ordenada, el tiempo de complejidad de la heurística es $O(n\ log\ n)$.

La JE-heurística da una aproximación en el peor caso de radio 2 (ver sección 2.4), es decir, esta da una solución que es a lo más dos veces peor que una solución óptima.

Se demostró que la JE-heurística es una herramienta eficiente para solucionar una amplia gama de problemas de calendarización.

En el problema clásico de calendarización tipo job-shop para la versión con interrupciones, la JE-heurística aplicada para un problema especial en una sola máquina, da una cota inferior, ver por ejemplo, Carlier [23], Carlier y Pinson [24] y Brinkkotter y Brucker [60] y trabajos más recientes de Gharbi y Labidi [2] y Della Croce y T'kindt [13]. Carlier y Pinson [25] han usado una sencilla generalización de la JE-heurística para la solución del problema de job-shop con multiprocesador con máquinas idénticas. Esto además puede ser adoptado para el caso cuando las máquinas paralelas no son relacionadas, ver [49]. La JE-heurística puede ser útil para que los cálculos sean paralelos en calendarizar job-shop (Perregaard and Clausen [40]), y también para los

problemas de calendarización de lotes con tiempos de liberación Condotta et al. [1].

2.3. Conceptos básicos y definiciones

El JE-calendario tiene propiedades estructurales útiles que nos permitieron deducir nuestros resultados. En está sección presentamos algunas definiciones básicas tomadas de N. Vakhania (2003) [44] que nos ayudaron a exponer estas propiedades.

Un JE-calendario puede contener un *hueco*, que es un intervalo de tiempo en el cual ningún trabajo es procesado por la máquina. Asumimos que un hueco de longitud cero $(c_j,\, t_i)$ ocurre siempre que el trabajo i empieza en su tiempo de inicio más temprano posible, es decir, en su tiempo de liberación, inmediatamente después del tiempo de completés del trabajo j.

Un *bloque* en un JE-calendario, es una parte consecutiva sin hueco la cual consiste de trabajos calendarizados sucesivamente, y es precedida y seguida por un hueco (posiblemente un hueco de longitud cero).

Un JE-calendario tiene propiedades estructurales beneficiosas. Las siguientes definiciones básicas tomadas de [45], nos ayudaron a expresar estas propiedades.

Entre todos los trabajos en un JE-calendario S, distinguimos uno el cual tiene su tiempo de completés total máximo en S; el último trabajo calendarizado es llamado el *overflow job* en S. Denotamos el overflow job en S por $o(S)$. El *bloque crítico* de S, $B(S)$, es el bloque que contiene a $o(S)$.

Un trabajo e es llamado *trabajo emergente* en el calendario S si $e \in B(S)$ y $q_e < q_{o(S)}$. El último trabajo emergente calendarizado antes del overflow job $o(S)$ es llamado *activo* y es denotado por l.

El *kernel* de S, $K(S)$, consiste del conjunto de trabajos calendarizados en S, que están entre el trabajo emergente activo l y el overflow job $o(S)$, sin incluir l pero que incluye $o(S)$ (observemos que la cola de cualquier trabajo del $K(S)$ no es menor que la del overflow job $o(S)$). Denotamos por $r(K)$ el tiempo de liberación mínimo de los trabajos del kernel K.

Denotamos por $E\,(S)$ el conjunto de todos los trabajos emergentes en el calendario S calendarizados antes del $K(S)$.

La figura 2.3 ilustra las nociones introducidas anteriormente.

$r_1 = 3$	$p_1 = 5$	$q_1 = 4$
$r_2 = 0$	$p_2 = 10$	$q_2 = 0$
$r_3 = 11$	$p_3 = 1$	$q_3 = 25$
$r_4 = 18$	$p_4 = 4$	$q_4 = 7$
$r_5 = 20$	$p_5 = 10$	$q_5 = 10$
$r_6 = 19$	$p_6 = 3$	$q_6 = 15$

$c_2 = 0 + 10 = 10$	$C_2 = 10 + 0 = 10$
$c_1 = 10 + 5 = 15$	$C_1 = 15 + 4 = 19$
$c_3 = 15 + 1 = 16$	$C_3 = 16 + 25 = 41$
$c_4 = 18 + 4 = 22$	$C_4 = 22 + 7 = 29$
$c_6 = 22 + 3 = 25$	$C_6 = 25 + 15 = 40$
$c_5 = 25 + 10 = 35$	$C_5 = 35 + 10 = 45$

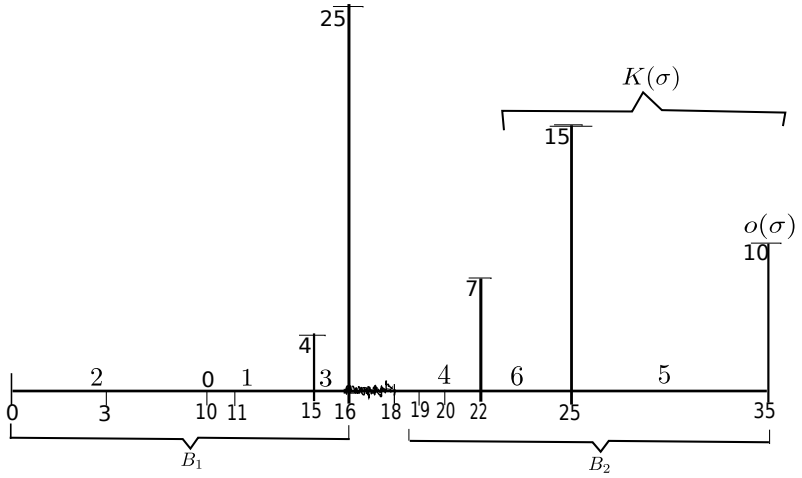


Figura 2.3:
Ejemplo de una instancia con 6 trabajos y del calendario σ para esta instancia. Observemos que este calendario σ, tiene un hueco dado por el intervalo $[16, 18]$ y por lo tanto contiene dos bloques, B_1 y B_2. En este ejemplo, el trabajo 5 es el overflow job y éste está contenido en el bloque B_2, por lo tanto el bloque B_2 es el bloque crítico. El kernel $K(\sigma)$ está formado por los trabajos 6 y 5. Y hay solamente un trabajo emergente, el trabajo 4.

Si un calendario S no es óptimo, entonces debe existir un trabajo emergente que produce el retraso de los trabajos del kernel $K(S)$ y del overflow job $o(S)$ que debe ser calendarizado más temprano (ver Lema 6 en la página 31). Denotamos la cantidad de este retraso por $\Delta_l = c_l(S) - r(K(S))$.

Observación 1. *En un calendario factible, los trabajos del kernel $K(\sigma)$ pueden ser calendarizados más temprano por a lo más Δ_l unidades de tiempo.*

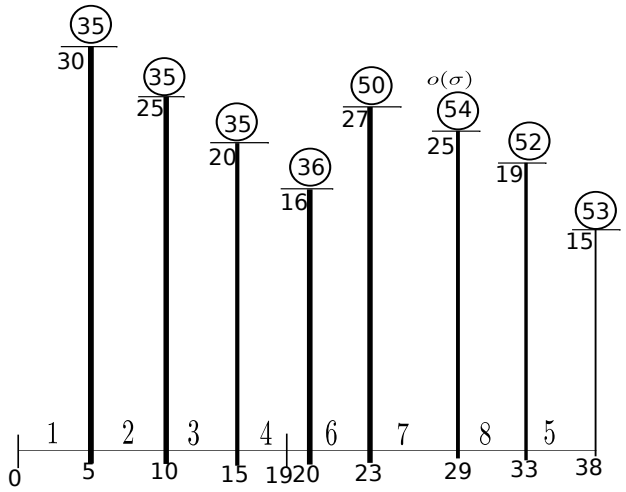
Demostración. Sigue de la definición de Δ_l y del hecho que ningún trabajo

del kernel $K(\sigma)$ es liberado más temprano que en el tiempo $r(K(\sigma))$. □

Para reducir la tardanza máxima ($|S|$) en S, *aplicamos* un trabajo emergente e a el $K(S)$, es decir, *recalendarizamos* el trabajo e después del $K(S)$. Técnicamente, hacemos esto en dos pasos: primero incrementamos artificialmente el tiempo de liberación del trabajo e, asignando a éste una magnitud no menos que el tiempo de liberación del último trabajo en el $K(S)$, entonces aplicamos la JE-heurística a la instancia del problema modificada en este camino. Ya que el tiempo de liberación del trabajo e no es menor que cualquier otro trabajo del $K(S)$, y q_e es menor que la cola de cualquier trabajo del $K(S)$ (por la definición de trabajo emergente), la JE-heurística dará prioridad a los trabajos del $K(S)$ y el trabajo e será calendarizado después de todos esos trabajos.

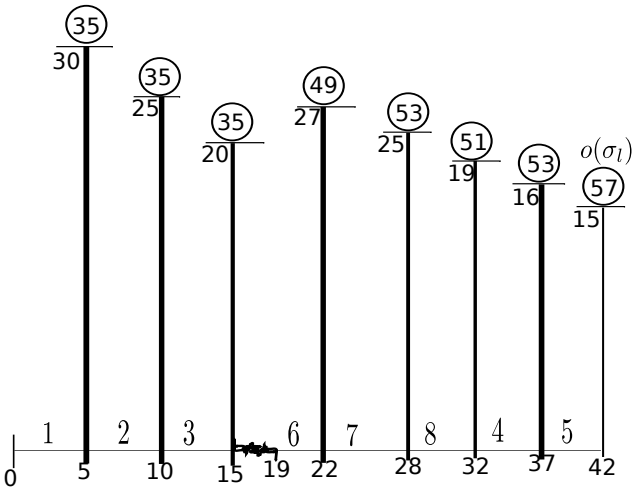
Si recalendarizamos el trabajo emergente activo l, entonces es fácil ver que habrá un hueco antes de los trabajos del kernel $K(S)$, si ningún otro trabajo emergente calendarizado en S después del kernel $K(S)$ consigue incluirse antes del kernel $K(S)$ (tomando la posición anterior del trabajo l). En general, mientras aplicamos cualquier trabajo emergente $e \in E(S)$, evitamos tal escenario, incrementando artificialmente el tiempo de liberación de cualquier trabajo emergente que puede otra vez empujar los trabajos del kernel $K(S)$ en el nuevo calendario construido que llamamos *complementario* para el calendario S y lo denotamos por S_e.

En la figura 2.5 damos un ejemplo de la construcción del calendario σ_l a partir del calendario σ de la figura 3.11.



$r_1 = 0$	$p_1 = 5$	$q_1 = 30$
$r_2 = 0$	$p_2 = 5$	$q_2 = 25$
$r_3 = 0$	$p_3 = 5$	$q_3 = 20$
$r_4 = 0$	$p_4 = 5$	$q_4 = 16$
$r_5 = 0$	$p_5 = 5$	$q_5 = 15$
$r_6 = 19$	$p_6 = 3$	$q_6 = 27$
$r_7 = 19$	$p_7 = 6$	$q_7 = 25$
$r_8 = 19$	$p_8 = 4$	$q_8 = 19$

Figura 2.4: Ejemplo de una instancia con 8 trabajos y del calendario σ para esta instancia. Observemos que en el calendario σ, el overflow job es el trabajo 7 y este contiene dos trabajos emergentes, los trabajos 3 y 4 (en este caso el trabajo 4 es el trabajo emergente activo). Los números dentro de los círculos son los tiempos de completés total del trabajo correspondiente.



$r_1 = 0$	$p_1 = 5$	$q_1 = 30$
$r_2 = 0$	$p_2 = 5$	$q_2 = 25$
$r_3 = 0$	$p_3 = 5$	$q_3 = 20$
$r_4 = 19$	$p_4 = 5$	$q_4 = 16$
$r_5 = 19$	$p_5 = 5$	$q_5 = 15$
$r_6 = 19$	$p_6 = 3$	$q_6 = 27$
$r_7 = 19$	$p_7 = 6$	$q_7 = 25$
$r_8 = 19$	$p_8 = 4$	$q_8 = 19$

Figura 2.5: Tomando como base la estructura del calendario de la figura 3.11, construimos el calendario σ_4 (el calendario generado al recalendarizar el trabajo 4 después del kernel $K(\sigma)$). Esto lo hacemos de la siguiente manera: aumentamos el tiempo de liberación del trabajo 4 (no menos que el tiempo de liberación mínimo del kernel $K(\sigma)$, en este caso, no menos que 19) y aplicamos la JE-heurística.

2.4. La cota en el peor caso de la JE-heurística

Esta sección está enfocada en demostrar que la JE-heurística da una aproximación en el peor caso de radio 2, es decir, esta da una solución que es a lo más dos veces peor que una solución óptima.

Como un complemento de este proyecto de investigación presentamos los siguientes resultados tomados de [53]. Cabe destacar que los resultados presentados en esta sección son dados con la versión de fechas límites, es decir, para el problema $1|r_j|L_{max}$.

Lema 1. *La tardanza máxima (makespan) de un kernel K no puede ser*

reducida si el trabajo calendarizado más temprano en K empieza en el tiempo $r(K)$. Si un $JE-calendario$ S contiene un kernel con esta propiedad, entonces, es óptimo.

Demostración. Recordemos que todo trabajo $i \neq o(\sigma)$ del kernel K, $q_i \geq o(S)$ y que los trabajos en K forman una secuencia sin ningún hueco. Entonces, debido a que el trabajo más temprano en K empieza en su tiempo de liberación, ninguna reordenación de los trabajos en K puede reducir la máxima tardanza, que es $L_o(S)$. En este caso, no existe un calendario factible $S^{'}$ con $L(S^{'}) < L_o(S)$, es decir S es óptima. □

Así, el calendario σ es óptimo si la condición del Lema 1 se cumple. Por otra parte, debe existir un trabajo menos urgente que $o(\sigma)$ calendarizado antes de los trabajos del kernel K, que atrasa los trabajos del kernel (y el overflow job $o(\sigma)$). Pero, recalendarizando tal trabajo más tarde, los trabajos del kernel K pueden ser reiniciados más temprano. Necesitamos algunas definiciones adicionales para definir formalmente esta operación.

Supongamos que el trabajo i está calendarizado antes del trabajo j en el $JE-calendario$ S. Decimos que el trabajo i empuja al trabajo j en S, si la JE-heurística recalendariza al trabajo j más temprano, siempre que el trabajo i, es forzado a estar calendarizado después del trabajo j.

Debido a que el trabajo calendarizado más temprano del kernel K no empieza en su tiempo de liberación (ver Lema 1), este es inmediatamente precedido y empujado por un trabajo l con $d_l > d_{o(\sigma)}$. En general, podemos tener más de un trabajo calendarizado antes del kernel K en el bloque B (un bloque que contiene al kernel K).

Del Lema 1 y de la definición de trabajo emergente inmediatamente se tiene el siguiente corolario:
Corolario 1. *Si S contiene un kernel que no tiene trabajos emergentes, entonces S es óptimo.*

Denotamos como T^S el makespan (máximo tiempo de completés total) del JE-calendario S, y T^* (L^*_{max}, respectivamente) para el makespan óptimo (tardanza, respectivamente).
Lema 2. *$T^\sigma - T^* < p_l$ ($L^\sigma_{max} - L^*_{max} < p_l$), donde l es el trabajo emergente activo del kernel $K \in \sigma$.*

Demostración. Es necesario mostrar que el tiempo de retraso impuesto por el

trabajo l para los trabajos del kernel K en el JE-calendario σ es menos que p_l. Ningún trabajo del kernel K está liberado en el tiempo de inicio del trabajo l en σ, de lo contrario la JE-heurística tendría que incluir un trabajo del kernel en lugar de l. En el mismo tiempo, el trabajo más temprano del kernel K es calendarizado inmediatamente después del trabajo l en σ. Entonces la diferencia entre el tiempo de inicio del trabajo anterior y el momento de tiempo $r(K)$ es menos que p_l. Del Lema 1 se obtiene la sentencia. □

El Lema 2 define implícitamente una cota inferior de $T^\sigma - p_l$ derivada de la solución de la heurística extendida de Jackson sin interrupciones. Esta cota inferior puede ser mejorada usando el siguiente concepto: Definimos el retraso del kernel $k \in \sigma$, $\delta(k,l) = c_l - r(k)$ (l presenta otra vez el trabajo emergente activo para el kernel K).

Lema 3. *$L^* = T^\sigma - \delta(k,l)$ ($L_{o(\sigma)} - \delta(k,l)$, respectivamente) es una cota inferior del makespan óptimo T^* (tardanza L^*_{max}, respectivamente).*

Demostración. La demostración es similar a la del Lema 2, con una observación extra que el retraso para el trabajo calendarizado más temprano del kernel K es definido por $\delta(K,l)$. □

Observemos que $\delta(k,l) < p_l$, y, en la práctica, $\delta(K,l)$ puede ser drásticamente más pequeño que p_l.

Podemos ver que la JE-heurística obtiene un desempeño de radio 2 para la versión $1\,|r_j,\, q_j|\, C_{max}$ (observemos que la estimación de la aproximación para la versión con fechas límite con el objetivo de minimizar la tardanza máxima es menos apropiada: por ejemplo, la tardanza óptima podría ser negativa).

Lema 4. *La JE-heurística da una solución $2-$aproximada para el problema $1\,|r_j,\, q_j|\, C_{max}$, es decir, $T^\sigma/T^* < 2$.*

Demostración. Si no existe un trabajo emergente activo para $K \in \sigma$, entonces σ es óptimo por el Corolario 1. Supongamos que l existe; entonces, $p_l < T^*$ (como el trabajo l tiene que ser calandarizado en S^* y hay al menos un trabajo más en este). Entonces por Lema 2,

$T^\sigma/T^* < (T^\sigma + p_l)/T^* = 1 + p_l/T^* < 1 + 1 = 2.$ □

Con el Lema 4 se llegó al objetivo de esta sección.

La cota en el peor caso de 2, puede ser bastante mala en la práctica, cuando la calidad de la solución es importante, es decir, cuando se requiere una solución con valor objetivo mejor que dos veces peor que el valor óptimo.

Varios algoritmos eficientes son modificaciones de la heurstica extendida de Jackson. Potts [8] propuso una modificación de la JE-heurística que repetidamente aplica la JE-heurística $O(n^2\ log\ n)$ tiempos y obtiene una mejor aproximación de radio 3/2. Nowicki y Smutnicki [12] presentaron un algoritmo más eficiente de aproximación 3/2 en cual corre en $O(n\ log\ n)$ tiempos. Mastrolilli [39] mejoró los resultados de Nowicki y Smutnicki derivando un esquema de aproximación en tiempo polinomial cuyo tiempo de ejecución depende sólo linealmente de n.

Capítulo 3

Condiciones de optimalidad

En este capítulo presentamos los resultados que obtuvimos en el estudio del problema $1|r_j, q_j|C_{max}$. Damos nuevas condiciones de optimalidad cuando casos especiales del problema $1|r_j, q_j|C_{max}$ pueden ser solucionados en forma óptima en tiempo polinomial; estas condiciones fueron obtenidas explorando las propiedades estructurales inherentes del problema. Más específicamente, presentamos condiciones explícitas que llevan a una solución eficiente del problema mediante la aplicación de la JE-heurística (ver sección 3.1). Estudiamos las propiedades estructurales útiles de los JE-calendarios (los calendarios creados por la JE-heurística) que llevan a la solución óptima de otras versiones del problema con el mismo costo computacional que el de la JE-heurística.

Finalmente, nos enfocamos en un caso especial del problema $1|r_j, q_j|C_{max}$ cuando solamente dos tiempos de liberación r_1 y r_2 ($r_1 < r_2$), y dos colas q_1 y q_2 ($q_1 < q_2$) son permitidos (este problema es denotado como $1|\{r_1, r_2\}, \{q_1, q_2\}|C_{\text{máx}}$); demostramos que este problema es $NP - duro$ (ver sección 3.2). En base al resultado de la sección 3.2, mostramos que el problema $1|\{r_1, r_2\}, \{q_1, q_2\}|C_{\text{máx}}$ permite reglas de dominación estricta y condiciones de optimalidad que se pueden verificar en tiempo polinomial (sección 3.3).

Los resultados expuestos en este capítulo fueron previamente publicados en [9], [10] y [11].

Observamos que de acuerdo a la importancia y a la complejidad de cada resultado obtenido lo llamamos Lema o Teorema.

Recordemos que cada trabajo i tiene su tiempo de liberación r_i. Sean $r^1 \leq \ldots \leq r^k$ los tiempos de liberación distintos de los trabajos de una instancia del problema $1|r_j, q_j|C_{max}$.

A lo largo de éste capítulo denotaremos como $J_i\,(i = 1, \ldots, k)$ al conjunto de trabajos liberados al tiempo r^i.

3.1. Los casos y condiciones para la solución del problema $1|r_j, q_j|C_{max}$ en tiempo polinomial

En esta sección presentamos las condiciones que nos llevan a la solución eficiente del problema $1|r_j, q_j|C_{max}$. Estas condiciones fueron obtenidas inmediatamente del análisis del JE-calendario σ (proporcionan un tiempo de decisión $O(n \log n)$).

El primer caso que obtuvimos en el que el problema $1|r_j, q_j|C_{max}$ tiene solución óptima en tiempo polinomial es el siguiente:

Lema 5. *Si el overflow job $o(\sigma)$ está calendarizado en su tiempo de liberación, entonces el calendario σ es óptimo.*

Demostración. Si $o(\sigma)$ está calendarizado en su tiempo de liberación $r_{o(\sigma)}$, entonces su tiempo de inicio $t_{o(\sigma)} = r_{o(\sigma)}$. Entonces, el trabajo $o(\sigma)$ empieza en su tiempo de inicio más temprano en σ y finaliza en su tiempo de completés minímo. Por lo tanto, $o(\sigma)$ no puede ser recalendarizado más temprano que en σ, es decir, no existe un calendario σ' tal que $\left|\sigma'\right| < |\sigma|$ y σ es óptima. □

En la figura 3.1, ilustramos el Lema 5.

$r_1 = 1$	$p_1 = 4$	$q_1 = 10$
$r_2 = 3$	$p_2 = 2$	$q_2 = 8$
$r_3 = 7$	$p_3 = 4$	$q_3 = 20$
$r_4 = 10$	$p_4 = 6$	$q_4 = 9$
$r_5 = 17$	$p_5 = 3$	$q_5 = 5$

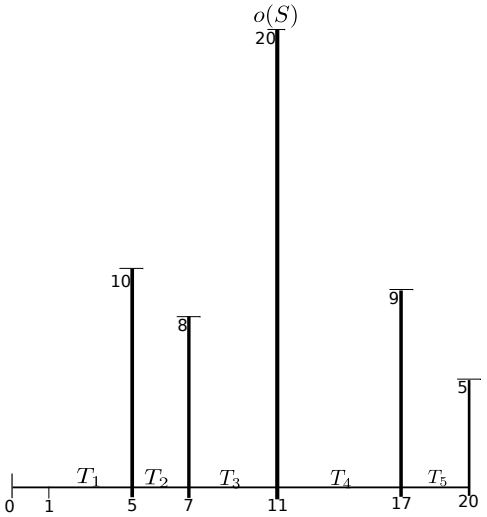


Figura 3.1: Ejemplo de una instancia con 5 trabajos y del calendario generado por la JE-heurística para esta instancia. Observemos que el trabajo 3 es el trabajo con tiempo de completés máximo, por lo tanto es el overflow job. Además observemos que el trabajo 3 está calendarizado en su tiempo de liberación $r_3 = 7$, por el Lema 5 este calendario es óptimo.

Si el overflow job $o(\sigma)$ no está calendarizado en su tiempo de liberación, entonces el tiempo de inicio de $o(\sigma)$ debe ser el tiempo de completés de algún otro trabajo.

Otro caso cuándo el calendario σ es óptimo, es cuando las colas de todos los trabajos calendarizados en el bloque crítico antes del overflow job $o(\sigma)$ son más largas que la cola de $o(\sigma)$:

Lema 6. *Si el bloque crítico $B(\sigma)$ no contiene ningún trabajo emergente (equivalentemente, no tiene kernel), entonces, el calendario σ es óptimo.*

Demostración. Asumamos que σ contiene un bloque crítico $B(\sigma)$, el cual no contiene un trabajo emergente, es decir, para todo $e \in B(\sigma)$ calendarizado

antes del trabajo $o(\sigma)$, $q_e \geq q_{o(\sigma)}$. Supongamos que hay una secuencia no vacía E de trabajos en $B(\sigma)$ calendarizados antes de $o(S)$. Si recalendarizamos algunos de los trabajos de E después del $o(\sigma)$ en el calendario resultante σ', al menos un trabajo e será completado no más temprano que en el momento $c_{o(\sigma)}$. Pero como e no es trabajo emergente, $q_e \geq q_{o(\sigma)}$, por lo tanto el tiempo de completés total de e, C_e, será no menos que $C_{o(\sigma)}$. Entonces $\left|\sigma'\right| > |\sigma|$. Por lo tanto, el calendario σ es óptimo.

Si el conjunto E es vacío entonces $o(\sigma)$ está calendarizado en σ en su tiempo de liberación, por lo tanto, por el Lema 5, el calendario σ es óptimo.

El caso cuando σ no tiene kernel es similar. □

La figura 3.2 muestra un ejemplo de un calendario que no contiene trabajos emergentes.

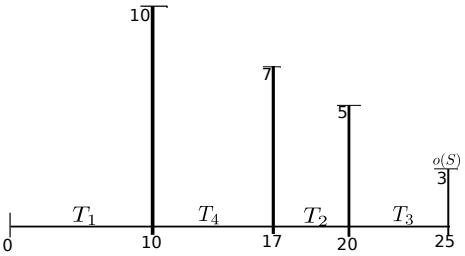


Figura 3.2: Ejemplo de un calendario σ que no contiene trabajos emergentes. Observemos que el trabajo 3 con cola $q_3 = 3$ es el overflow job en este calendario, y los trabajos restantes tienen cola mayor que 3.

Los siguientes Lemas (7 y 8) son resultados conocidos en la literatura; los presentamos en este trabajo como un complemento. En las figuras 3.3 y 3.4 ilustramos los respectivos resultados.

Lema 7. *Si el tiempo de liberación r_i $(i = 1, \ldots, n)$ de todos los trabajos es igual a una constante r, entonces el calendario σ es óptimo.*

Demostración. Como todos los trabajos i $(i = 1, \ldots, n)$ son liberados en el tiempo r, entonces, la JE-heurística en cada iteración calendariza el trabajo i con la cola más larga. La JE-heurística genera un calendario σ tal que los trabajos son calendarizados en orden no creciente de sus colas. Entonces, el calendario σ no puede contener un trabajo emergente, y por el Lema 6, el calendario σ es óptimo. □

$r_1 = 0$	$p_1 = 10$	$q_1 = 10$	$c_1 = 0 + 10 = 10$	$C_1 = 10 + 10 = 20$
$r_2 = 0$	$p_2 = 3$	$q_2 = 5$	$c_4 = 10 + 7 = 17$	$C_4 = 17 + 7 = 24$
$r_3 = 0$	$p_3 = 5$	$q_3 = 3$	$c_2 = 17 + 3 = 20$	$C_2 = 20 + 5 = 25$
$r_4 = 0$	$p_4 = 7$	$q_4 = 7$	$c_3 = 20 + 5 = 25$	$C_3 = 25 + 3 = 28$

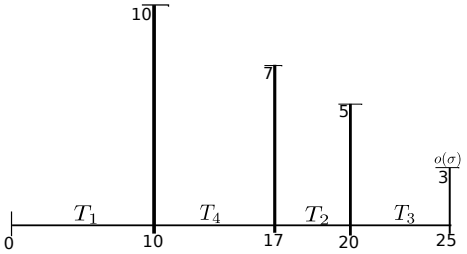


Figura 3.3: Ejemplo de una instancia en la que todos los trabajos tienen tiempo de liberación igual y del calendario generado por la JE-heurística para esta instancia. Observemos que el calendario no contiene trabajos emergentes.

Lema 8. *Sea σ un JE-calendario con trabajos i $(i = 1, ...n)$ de tiempos de liberación enteros r_i y tiempos de procesamiento $p_i = 1$ $(i = 1, ..., n)$. Entonces el calendario σ es óptimo.*

Demostración. Dado que el tiempo de liberación r_i de cada trabajo i es un número entero y su tiempo de procesamiento es 1, durante la ejecución de algún trabajo ningún otro trabajo más urgente puede ser liberado. De aquí , en cada tiempo de calendarización t, el algoritmo de Jackson extendido, entre todos los trabajos liberados al mismo tiempo, incluye uno con la cola más larga.

Además, en σ, cada trabajo es calendarizado en su tiempo de liberación r_i o en el tiempo de completés de otro trabajo. Si el trabajo $o(\sigma)$ es calendarizado en el tiempo $t = r_{o(\sigma)}$, entonces el calendario σ es óptimo por Lema 5. Si $o(\sigma)$ es calendarizado en el tiempo de completés de otro trabajo k tal que el trabajo $o(\sigma)$ fue liberado antes del tiempo de completés de este trabajo, entonces $q_k \geq q_{o(\sigma)}$. Esto implica, el calendario σ no puede contener un trabajo emergente y es óptimo por Lema 6. □

$r_1 = 2$	$p_1 = 1$	$q_1 = 10$	$c_1 = 2 + 1 = 3$	$C_1 = 3 + 10 = 13$
$r_2 = 3$	$p_2 = 1$	$q_2 = 15$	$c_2 = 3 + 1 = 4$	$C_2 = 4 + 15 = 19$
$r_3 = 4$	$p_3 = 1$	$q_3 = 5$	$c_3 = 4 + 1 = 5$	$C_3 = 5 + 5 = 10$
$r_4 = 7$	$p_4 = 1$	$q_4 = 3$	$c_4 = 7 + 1 = 8$	$C_4 = 8 + 3 = 11$
$r_5 = 8$	$p_5 = 1$	$q_5 = 20$	$c_5 = 8 + 1 = 9$	$C_5 = 9 + 20 = 29$

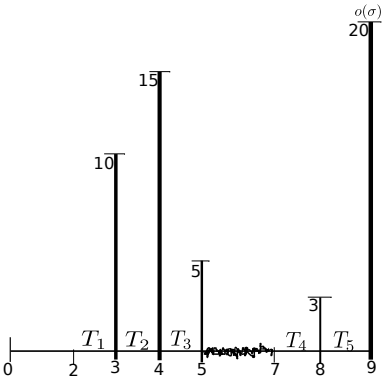


Figura 3.4: Ejemplo de una instancia donde todos los trabajos tienen tiempo de procesamiento $p_i = 1$ $(i = 1 \dots 5)$, y del calendario σ generado por la JE-heurística para esta instancia.

De ahora en adelante seguimos con la presentación de nuestros resultados.

Lema 9. *El conjunto de trabajos en* $J = \{i_j\}_{j=1,\dots,n}$ *es ordenado por una secuencia no decreciente de sus tiempos de liberación. Entonces el calendario* σ *es óptimo si para cualesquiera par de trabajos vecinos* i_j *y* i_{j+1} $(j = 1, \dots, n-1)$, $r_{i_{j+1}} \geq r_{i_j} + p_{i_j}$.

Demostración. Por la condición de nuestra afirmación, cada trabajo i_j $(j = 1, \dots, n)$ es calendarizado en σ en el intervalo $\left[r_{i_j},\, r_{i_j} + p_{i_j}\right]$. Así, el tiempo de inicio de cada trabajo en σ es $t_j = r_{i_j}$, es decir, cada trabajo i_j empieza en su tiempo de inicio más temprano en σ. Esto es cierto, en particular, para el trabajo $o(\sigma)$ y por lo tanto, por Lema 6, el calendario σ es óptimo. □

En la figura 3.5 ilustramos con un ejemplo el Lema 9.

$r_1 = 0$	$p_1 = 5$	$q_1 = 10$	$c_1 = 0 + 5 = 5$
$r_2 = 5$	$p_2 = 3$	$q_2 = 15$	$c_2 = 5 + 3 = 8$
$r_3 = 10$	$p_3 = 6$	$q_3 = 30$	$c_3 = 10 + 6 = 16$
$r_4 = 16$	$p_4 = 4$	$q_4 = 20$	$c_4 = 16 + 4 = 20$
$r_5 = 21$	$p_5 = 2$	$q_5 = 15$	$c_5 = 21 + 2 = 23$
$r_6 = 23$	$p_6 = 10$	$q_6 = 6$	$c_6 = 23 + 10 = 33$

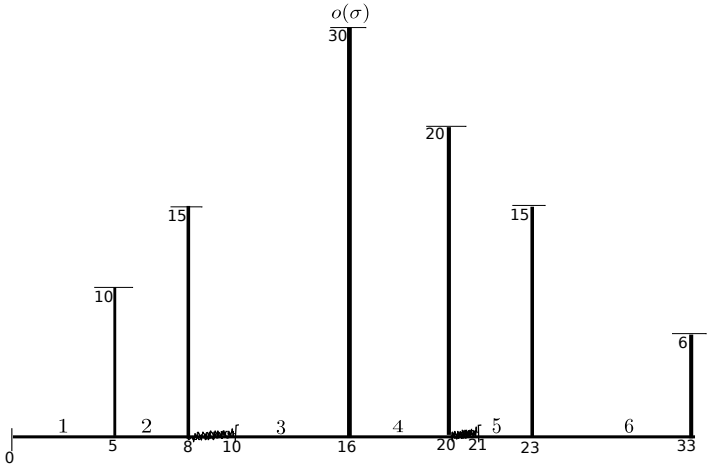


Figura 3.5: Ejemplo de una instancia en la que cualquier par de trabajos vecinos i_j y i_{j+1} cumplen con la condición, $r_{i_{j+1}} \geq r_{i_j} + p_{i_j}$. Observamos que en el calendario σ, cada trabajo i_j es calendarizado en el intervalo $\left[r_{i_j},\, r_{i_j} + p_{i_j}\right]$.

El siguiente resultado es un caso especial de cuando el overflow job está liberado al tiempo r_1.

Teorema 1. *Para una instancia dada del problema* $1|r_j, q_j|C_{\text{máx}}$*, el JE-calendario inicial* σ *es óptimo si* $o(\sigma) \in J_1$.

Demostración. Por la condición, en cualquier tiempo de calendarización entre los tiempos de liberación $r^2, \ldots, r^k$, el trabajo $o(\sigma)$ fue liberado. Entonces por la JE-heurística , para cada trabajo i calendarizado en σ antes del trabajo $o(\sigma)$, $q_i \geq q_{o(\sigma)}$. Por lo tanto, σ no contiene un trabajo emergente y es óptimo por Lema 6. □

En la sección 2.3 mencionamos que para reducir la tardanza máxima en un calendario σ, es necesario recalendarizar un trabajo emergente e después del

$K(\sigma)$. En el siguiente resultado mostramos qué pasa cuando un trabajo e es calendarizado dentro del $K(\sigma)$.

Asumimos que K es cualquier kernel que forma parte de algún JE-calendario (K puede ser un kernel de otro JE-calendario distinto de S). Entonces decimos que el trabajo emergente $e \in E(S)$ *es calendarizado dentro del kernel* K si existe al menos un trabajo de este kernel calendarizado antes y depués del trabajo e en el calendario S.

Lema 10. *Si el trabajo e es calendarizado dentro de kernel $K(\sigma)$ en el calendario S entonces $|S| > |\sigma|$.*

Demostración. Observemos que ningún trabajo del kernel $K(\sigma)$ calendarizado antes del trabajo e puede ser overflow job en el calendario S, y ningún trabajo e puede ser overflow job en S porque después del trabajo e existe calendarizado al menos un trabajo $j \in K(\sigma)$ con $q_j \geq q_e$. Además, ningún trabajo k calendarizado después del kernel $K(\sigma)$ puede ser overflow job en el calendario S.

Del Lema 10 concluimos que sólo un trabajo del kernel $K(\sigma)$ calendarizado después del trabajo e puede ser overflow job en el calendario S. Pero por el desplazamiento forzado a la derecha impuesto por el trabajo e, el trabajo j del kernel $K(\sigma)$ calendarizado al último en el calendario σ no pude completarse más temprano en el calendario S que el trabajo $o(\sigma)$ fue completado en el calendario σ, es decir, $c_j(S) \geq c_{o(\sigma)}(\sigma)$.

En el mismo tiempo, por la definición de kernel $K(\sigma)$ el trabajo j no es menos urgente que trabajo $o(\sigma)$, es decir, $q_j \geq q_{o(\sigma)}$. Entonces $C_j(S) \geq C_{o(\sigma)}(\sigma)$ y por lo tanto $|S| \geq |\sigma|$. □

Como una consecuencia de los Lemas 6 y 10 obtenemos el siguiente corolario:

Corolario 2. *En cualquier calendario con tiempo de completés menor que σ, un trabajo emergente es recalendarizado después del $K(\sigma)$.*

De ahora en adelante denotaremos como $J_i^{'}$ $(i = 1, \ldots, k-1)$ al conjunto unión de los conjuntos $J_1, J_2 \ldots, J_i$, es decir, $J_i^{'} = J_1 \cup J_2 \cup \ldots \cup J_i$.

Observación 2. *Los trabajos del conjunto $J_i^{'}$ $(i = 1, \ldots, k-1)$ calendarizados dentro del intervalo (r^i, r^{i+1}) se encuentran en el calendario σ en orden no creciente de sus colas.*

Demostración. Primero observamos que todos los trabajos del conjunto $J_i^{'}$ al tiempo r^i están liberados Entonces nuestra afirmación es válida porque la

JE-heurística en cada tiempo de calendarización $t \in (r^i, r^{i+1})$, entre todos los trabajos aún no calendarizados liberados, repetidamente incluye uno con la cola más larga. □

Sea $j^{[i]}$ el primer trabajo del conjunto $J_i^{'}$ durante la construcción del calendario σ, tal que $t_{j^{[i]}} + p_{j^{[i]}} > r^{i+1}$ ($t_{j^{[i]}}$ y $p_{j^{[i]}}$, son los tiempos de inicio y procesamiento de trabajo $j^{[i]}$ respectivamente), para todo $i = 1, \ldots, k-1$. Sea q_{max}^{i+1} es la cola más larga entre todas las colas de los trabajos liberados al tiempo r^{i+1}.

Lema 11. *Bajo la condición $q_{j^{[i]}} \geq q_{max}^{i+1}$ para todo $i = 1, \ldots, k-1$, entonces el calendario σ es óptimo.*

Demostración. Por la condición, el calendario σ no contiene trabajos emergentes y por lo tanto el calendario σ es óptimo por el Lema 6. □

Observación 3. *Los trabajos del conjunto J_j son calendarizados en orden no creciente de sus colas dentro del intervalo (r^i, r^s) en el calendario σ, para todo i, j y s tal que $1 \leq i \leq j < s \leq k$.*

Demostración. Por definición de la JE-heurística. □

Observación 4. *Los trabajos del conjunto J_j calendarizados antes del tiempo r^i, para todo $j < i$, tienen su cola más larga que los trabajos del conjunto J_j calendarizados despueś del tiempo r^i.*

Demostración. Por definición de la JE-heurística. □

Observación 5. *Si hay un trabajo j en el kernel $K(\sigma)$ liberado al tiempo r^i, entonces cualquier trabajo emergente en el calendario σ es liberado al tiempo $t \in \{r^1, \ldots, r^{i-1}\}$.*

Demostración. Por definición de la JE-heurística, cualquier trabajo i liberado al tiempo r^i calendarizado antes del trabajo j, su cola $q_i \leq q_j$. Entonces ninguno de tales trabajos puede ser un trabajo emergente para el kernel $K(\sigma)$ y la observación es válida. □

Observación 6. *Si el calendario σ contiene un trabajo emergente $e \in J_i$, entonces el kernel $K(\sigma)$ no puede contener trabajos del conjunto $J_1 \cup \ldots \cup J_i$.*

Demostración. Cualquier trabajo $j \in J_1 \cup \ldots \cup J_i$, no incluido al tiempo r^i en el calendario σ no puede ser más urgente que el trabajo e, de otra manera la JE-heurística incluiría aquel antes del kernel $K(\sigma)$ en lugar del trabajo e. Nuestra afirmación es válida ya que el trabajo e es un trabajo emergente para el kernel $K(\sigma)$ y todos los trabajos del kernel $K(\sigma)$ deben ser más urgentes que el trabajo j. □

Ahora, Qué podemos decir cuando el primer trabajo calendarizado del $K(\sigma)$ tiene tiempo de inicio igual a su tiempo de liberación? Como respuesta a esta pregunta damos el Lema 12.

Sea $Q(\sigma)$ la secuencia de trabajos i en el calendario σ calendarizados en el bloque crítico $B(\sigma)$ antes del overflow job $o(\sigma)$ tal que sus colas $q_i \geq q_{o(\sigma)}$.
Lema 12. *Si el tiempo de inicio del primer trabajo de la secuencia $Q(\sigma)$ es su tiempo de liberación, decimos r^i, es decir, el bloque crítico $B(\sigma)$ empieza al tiempo r^i, entonces el calendario σ es óptimo.*

Demostración. Por la condición de nuestra afirmación y por la definición del conjunto $Q(\sigma)$, el calendario σ no contiene un trabajo emergente y por lo tanto es óptimo por Lema 6. □

En lo que sigue, nuestro objetivo es dar algunos conceptos y observaciones que nos ayudaron a demostrar que: Si el calendario σ_e no tiene hueco y el overflow job en este calendario está en el conjunto $\{o(\sigma), e, k\}$, entonces $|\sigma_e| < |\sigma|$.

Sea $J[e]$ el conjunto de trabajos calendarizados inmediatamente después del kernel $K(\sigma)$ en σ, tal que para cualquier trabajo j de este conjunto, $q_j > q_e$.

De ahora en adelante, denotamos por k el trabajo con mayor tiempo de completés total en el calendario σ entre todos los trabajos calendarizados después de los trabajos del conjunto $J[e]$. Se puede ver que, el trabajo $k \in J_i$, para cualquier $i = 1, \ldots, k$.

Cabe destacar que el orden de los trabajos calendarizados en el calendario σ entre el trabajo e y el kernel $K(\sigma)$, los trabajos del kernel $K(\sigma)$ y del conjunto $J[e]$ podría ser diferente en el calendario σ y σ_e.

Sea j el trabajo del conjunto $J[e]$ calendarizado al último en el calendario σ.
Observación 7. *Si el calendario σ_e no tiene hueco, entonces el conjunto de trabajos calendarizados en el intervalo $[r^1, c_j(\sigma)]$ en este calendario es el*

mismo que en el calendario σ.
Observación 8. *Si el calendario* σ_e *no tiene hueco, entonces* $C_k(\sigma_e) = C_k(\sigma)$.

Demostración. Por Observación 7, el conjunto de trabajos calendarizados en el calendario σ_e en el intervalo $[r^1, c_j(\sigma)]$ es el mismo que en el calendario σ (posiblemente las secuencias correspondientes son diferentes). Del mismo modo, todos los trabajos calendarizados después de los trabajos del conjunto $J[e]$ en el calendario σ no tendrán desplazamiento en el calendario σ_e debido a que el trabajo e es incluido antes de ellos y no surge un nuevo hueco en el calendario σ_e. Entonces, el tiempo de inicio del trabajo k en ambos calendarios σ_e y σ es el mismo y nuestra afirmación es válida. □

Con ayuda de las Observaciones 7 y 8, y de otros criterios referentes a la estructura de un $JE - calendario$, obtuvimos el siguiente resultado:
Teorema 2. *Si el calendario* σ_e *no tiene hueco y el overflow job en este calendario está en el conjunto* $\{o(\sigma), e, k\}$, *entonces* $|\sigma_e| < |\sigma|$.

Demostración. Primero, supongamos que $o(\sigma)$ es el overflow job en el calendario σ_e. Nosotros sostenemos que $o(\sigma)$ es el último trabajo calendarizado del kernel $K(\sigma)$ en el calendario σ_e. Supongamos que algún trabajo $j \in K(\sigma) \setminus o(\sigma)$ es calendarizado después del trabajo $o(\sigma)$ en el calendario σ_e. Entonces $c_{o(\sigma)}(\sigma_e) < c_j(\sigma_e)$. Por la definición del kernel, cualquier trabajo $j \in K(\sigma) \setminus o(\sigma)$, $q_j \geq q_{o(\sigma)}$. Entonces, $C_j(\sigma_e) > C_{o(\sigma_e)}$ que contradice el hecho que $o(\sigma)$ es el overflow job en el calendario σ_e. Ahora, como el calendario σ_e no tiene hueco, $o(\sigma)$ es desplazado a la izquierda p_e unidades de tiempo en el calendario σ_e, es decir, $C_{o(\sigma)}(\sigma_e) = C_{o(\sigma)}(\sigma) - p_e$ y entonces $C_{o(\sigma)}(\sigma_e) < C_{o(\sigma)}(\sigma)$ y obtenemos que $|\sigma_e| < |\sigma|$.

Ahora supongamos que el trabajo e es el overflow job en el calendario σ_e, entonces el trabajo e es calendarizado después de todos los trabajos del conjunto $J[e]$ en el calendario σ_e. En efecto, si existe un trabajo $j \in J[e]$ calendarizado después del trabajo e, entonces $c_e(\sigma_e) < c_j(\sigma_e)$. Pero, $q_j > q_e$ (por la definición del conjunto $J[e]$), $C_e(\sigma_e) < C_j(\sigma_e)$, lo cual contradice al hecho que el trabajo $e$es overflow job en el calendario σ_e.

Además, por la Observación 7, el conjunto de trabajos calendarizados en el calendario σ_e en el intervalo $[r^1, c_j(\sigma)]$ es el mismo que en el calendario σ (mientras que el trabajo e es calendarizado en este intervalo). Como el

trabajo e es calendarizado después de todos los trabajos del conjunto $J[e]$ en el calendario σ_e, el tiempo de completés del trabajo e en el calendario σ_e, es igual al tiempo de completés del último trabajo calendarizado, decimos j, del conjunto $J[e]$ en el calendario σ. Pero $q_e < q_j$, y $C_e(\sigma_e) < C_j(\sigma)$, mientras $C_j(\sigma) \leq C_{o(\sigma)}(\sigma)$ y así $C_e(\sigma_e) < C_{o(\sigma)}(\sigma)$. Hemos probado que $|\sigma_e| < |\sigma|$.

Finalmente suponemos que k es el overflow job en el calendario σ_e. Por Observación 8, $C_k(\sigma_e) = C_k(\sigma)$. Pero $C_k(\sigma) < C_{o(\sigma)}(\sigma)$ y $C_k(\sigma_e) < C_{o(\sigma)}(\sigma)$ y por lo tanto $|\sigma_e| \ < |\sigma|$. □

3.2. La NP-dureza del problema: $1|\{r_1, r_2\}, \{q_1, q_2\}|C_{\text{máx}}$

A continuación, enfocamos nuestra atención sobre un caso especial del problema $1|r_j, q_j|C_{max}$ permitiendo solo 2 tiempos de liberación y 2 colas. Como vimos anteriormente, la JE-heurística proporciona un calendario óptimo si tenemos un solo tiempo de liberación o una sola cola (o fecha límite), es decir, los problemas $1|q_j|C_{\text{máx}}$ y $1|r_j|C_{\text{máx}}$ ($1||L_{\text{máx}}$ y $1|r_j, d_j = d|L_{\text{máx}}$ se solucionan en un tiempo casi lineal $n \log n$ (el problema general $1|r_j, q_j|C_{\text{máx}}$ siendo fuertemente NP-duro). Ahora mostramos que tenemos un problema NP-duro permitiendo solo dos tiempos de liberación distintos o colas; es decir, el problema $1|\{r_1, r_2\}, \{q_1, q_2\}|C_{\text{máx}}$ es NP-duro. Por conveniencia, consideramos la versión con fechas límite $1|\{r_1, r_2\}, \{d_1, d_2\}|L_{\text{máx}}$.

Teorema 3. $1|\{r_1, r_2\}, \{d_1, d_2\}|L_{\text{máx}}$ *es NP-duro.*

Demostración. Aplicamos la reducción de un problema de Suma de subconjuntos para la versión de factibilidad de $1|\{r_1, r_2\}, \{d_1, d_2\}|L_{\text{máx}}$, en el cual deseamos conocer si existe un calendario factible en el que todos los trabajos satisfacen sus fechas límite (es decir, en el cual todos los trabajos son completados no más tarde que sus fechas límite). El problema de subconjuntos consiste de un conjunto finito de números enteros $C = \{c_1, c_2, \ldots, c_n\}$ y un número entero $B \leq \sum_{i=1}^{n} c_i$. Este problema de decisión da un "si"de respuesta si y sólo si existe un subconjunto de C tal que la suma de sus elementos es igual a B. Dada una instancia arbitraria de Suma de subconjuntos, construimos nuestra instancia de calendarización con $n+1$ trabajos con longitud total de $\sum_{\iota=1}^{n} c_\iota + 1$ de modo siguiente:

Tenemos *n trabajos de partición 1,...,n* liberados al tiempo $r_1 = 0$ con $p_i = c_i$, $r_i = 0$ y $d_i = \sum_{\iota=1}^{n} c_\iota + 1$, para $i = 1, \ldots, n$. Además de estos trabajos de partición, tenemos otro trabajo *separador* I, liberado al tiempo $r_2 = B$ con $p_I = 1$, $r_I = r_2$ y fecha límite $d_I = B+1$. Observamos que esta transformación que crea una instancia de $1|\{r_1, r_2\}, \{d_1, d_2\}|L_{\text{máx}}$ es polinomial debido aque el número de trabajos es acotado por un polinomio de n, y todas las magnitudes pueden ser representadas en código binario en $O(n)$ bits.

Ahora probamos que existe un calendario factible en el cual todos los trabajos satisfacen sus fechas límite si y sólo si existe una solución para el problema de Suma de subconjuntos. En una dirección, supongamos que hay una solución del problema de Suma de subconjuntos formada por los números del conjunto $C' \subseteq C$. Sea J' el conjunto correspondiente a los trabajos de partición en nuestra instancia de calendarización. Construimos un calendario factible en el que todos los trabajos satisfacen sus fechas límite (Ver figura 3.6). Primero calendarizamos los trabajos de partición del conjunto J' en el intervalo $[0, B]$ en una manera sin retraso (usando la heurística extendida de Jackson). Entonces calendarizamos el trabajo separador al tiempo B completado en su fecha límite $B+1$ y el resto de los trabjos de partición empiezan a partir del tiempo $B+1$, también en una manera sin retraso por la heurística extendida de Jackson . Observemos que el último trabajo calendarizado será completado exactamente en el tiempo $\sum_{\iota=1}^{n} c_\iota + 1$ y todos los trabajos se cumplen a su fecha límite. Por lo tanto, existe un calendario factible en el cual todos los trabajos son completados no más tarde que su fecha límite si el problema Suma de subconjuntos tiene una solución.

Figura 3.6: Esquema de un calendario factible proporcionado por una solución del problema de SUMA DE SUBCONJUNTOS.

En la otra direción: supongamos que existe un calendario factible S en el cual todos los trabajos son completados no más tarde que en sus fechas límite. Entonces, en este calendario, el trabajo separador debe ser calendarizado en el intervalo $[B+1]$, mientras que el último trabajo de partición debe ser completado al tiempo $\sum_{\iota=1}^{n} c_\iota + 1$. Pero esto sólo es posible si el intervalo $[0, B]$ en el calendario S se llena completamente por los trabajos de partición, es decir,

debe haber un subconjunto correspondiente J' de los trabajos de partición que llenen completamente el intervalo $[0, B]$ en el calendario S (si este intervalo no está lleno completamente en el calendario S entonces el tiempo de completés del último trabajo de partición calendarizado será $\sum_{\iota=1}^{n} c_\iota + 1$ más la longitud total del hueco dentro del intervalo $[0, B]$ y entonces el trabajo no se completaría antes de su fecha límite, ver figura 3.7). Los tiempos de procesamiento de los trabajos en el conjunto J' forman la solución correspondiente del problema de Suma de subconjuntos.

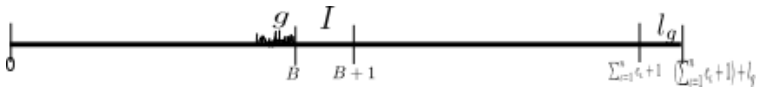


Figura 3.7: Un calendario no factible (g es hueco y lg es la longitud total del hueco).

□

3.3. Casos especiales tratables del problema $1|\{r_1, r_2\}, \{q_i\}|C_{\text{máx}}$

En la sección anterior mostramos que el problema $1|\{r_1, r_2\}, \{q_i\}|C_{\text{máx}}$, es NP-duro, en base a eso, en esta sección establecemos algunas propiedades que nos garantizan la solución óptima para el problema $1|\{r_1, r_2\}, \{q_i\}|C_{\text{máx}}$, estas propiedades pueden ser verificadas en tiempo $O(n \log n)$.

Para el primer resultado presentado en esta sección, asumimos que la secuencia de trabajos $\{i_j\}|j = 1, \ldots, m)$ es enumerada de tal forma que los primeros m trabajos de la secuencia tienen tiempo de liberación r_1 y los siguientes $n - m$ trabajos tienen tiempo de liberación r_2 $(m < n)$. Recordemos que denotamos como J_1 y J_2 a los conjuntos de trabajos que se liberan en el tiempo r_1 y r_2, respectivamente.

Lema 13. *Si* $\sum_{j=1}^{m} p_{i_j} + r_1 \leq r_2$ *entonces el calendario* σ *es óptimo.*

Demostración. Por lo que $\sum_{j=1}^{m} p_{i_j} + r_1 \leq r_2$, la JE-heurística en las primeras m iteraciones calendarizará todos los trabajos liberados al tiempo r_1. Como

los trabajos i_j, $(j = 1, \ldots, m)$ se liberan simultáneamente en el tiempo r_1, estos son calendarizados optimamente de acuerdo con el Lema 7.

Por la condición de nuestra afirmación, todos los trabajos liberados en el tiempo r_1 son completados al tiempo r_2, y por eso todos los trabajos en el conjunto J_2 son disponibles para ser calendarizados al tiempo r_2. Entonces, la JE-heurística calendarizará todos esos trabajos en forma óptima (por Lema 7).

Ahora, uniendo los dos calendarios parciales anteriores, obtenemos un calendario óptimo completo (si hay un aumento de un hueco entre las dos porciones de este calendario entonces este es inevitable). □

En la siguiente figura ilustramos el Lema 13.

$r_1 = 2$ y $r_2 = 21$

$r_1' = 2$	$p_1 = 3$	$q_1 = 19$	$c_1 = 2 + 3 = 5$	$C_1 = 5 + 19 = 24$
$r_2' = 2$	$p_2 = 5$	$q_2 = 15$	$c_2 = 5 + 5 = 10$	$C_2 = 10 + 15 = 25$
$r_3' = 21$	$p_3 = 4$	$q_3 = 25$	$c_4 = 10 + 10 = 20$	$C_4 = 20 + 5 = 25$
$r_4' = 2$	$p_4 = 10$	$q_4 = 5$	$c_3 = 21 + 4 = 25$	$C_3 = 25 + 25 = 50$
$r_5' = 21$	$p_5 = 6$	$q_5 = 10$	$c_5 = 25 + 6 = 31$	$C_5 = 31 + 10 = 41$
$r_6' = 21$	$p_6 = 2$	$q_6 = 3$	$c_6 = 31 + 2 = 33$	$C_6 = 33 + 3 = 36$

$$\sum_{j=1}^{3} p_{i_j} + r_1 = (3 + 5 + 10) + 2 = 20 < 21 = r_2$$

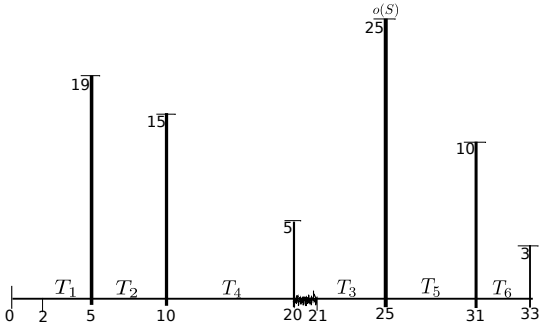


Figura 3.8: Ejemplo de una instancia en la que se cumple la condición: $\sum_{j \in J_1} p_{i_j} + r_1 \leq r_2$ y del calendario generado por la JE-heurística. En esta instancia $r^1 = 2$ y $r^2 = 21$.

Si no se cumple la condición del Lema 13, entonces podemos tener el siguiente

caso:

Lema 14. *Si $k < m$ es tal que para los primeros k trabajos con la cola más larga en el calendario σ la igualdad $\sum_{l=1}^{k} p_{i_{j_l}} = r_2 - r_1$ se cumple, entonces el calendario σ es óptimo.*

Demostración. Distinguimos los siguientes dos casos basados en el hecho que para los primeros k trabajos con la cola más larga liberados al tiempo r_1, $\sum_{l=1}^{k} p_{i_{j_l}} = r_2 - r_1$ se satisface para los siguientes casos:

Caso 1: $k = m$. En este caso $\sum_{l=1}^{m} p_{i_{j_l}} = r_2 - r_1$ y por Lema 9 el calendario σ es óptimo (con la igualdad siendo cumplida para cualesquiera dos trabajos vecinos).

Caso 2: $k < m$. Sea $\sigma^{'}$ un JE-calendario para los primeros k trabajos. Debido a que todos esos trabajos son liberados en el tiempo r_1, ellos son calendarizados en forma óptima por Lema 7. Además, $\sum_{l=1}^{k} p_{i_{j_l}} \leq r_2 - r_1$, el tiempo de inicio más temprano de los trabajos $i_{j_{k+1}}, \ldots, i_{j_m}$ es r_2. A partir de este tiempo r_2, todos los $n - k$ trabajos restantes no calendarizados aún se hacen simultáneamente disponibles. Sea $\sigma^{''}$ el JE-calendario para esos trabajos. Como todos los trabajos aún no calendarizados son disponibles al tiempo r_2, $\sigma^{''}$ es óptimo por Lema 7.

Ahora sea σ el JE-calendario obtenido al juntar los JE-calendarios $\sigma^{'}$ y $\sigma^{''}$. Observemos que el calendario σ no tiene hueco, este consiste de un sólo bloque. Por nuestra construcción, no existe un trabajo emergente en el calendario σ, y este es óptimo por Lema 6. □

En la figura 3.9 ilustramos el Lema 14. En la instancia dada en esta figura $r^1 = 2$ y $r^2 = 15$.

$r_1 = 2$	$p_1 = 3$	$q_1 = 15$	$c_2 = 2 + 5 = 7$	$C_2 = 7 + 20 = 27$
$r_2 = 2$	$p_2 = 5$	$q_2 = 20$	$c_1 = 7 + 3 = 10$	$C_1 = 10 + 15 = 25$
$r_3 = 18$	$p_3 = 4$	$q_3 = 19$	$c_6 = 10 + 3 = 13$	$C_6 = 13 + 12 = 25$
$r_4 = 2$	$p_4 = 5$	$q_4 = 10$	$c_4 = 13 + 5 = 18$	$C_4 = 18 + 10 = 28$
$r_5 = 18$	$p_5 = 2$	$q_5 = 8$	$c_3 = 18 + 4 = 22$	$C_3 = 22 + 19 = 41$
$r_6 = 2$	$p_6 = 3$	$q_6 = 12$	$c_5 = 22 + 2 = 24$	$C_5 = 24 + 8 = 32$
$r_7 = 2$	$p_7 = 6$	$q_7 = 5$	$c_7 = 24 + 6 = 30$	$C_7 = 30 + 5 = 35$

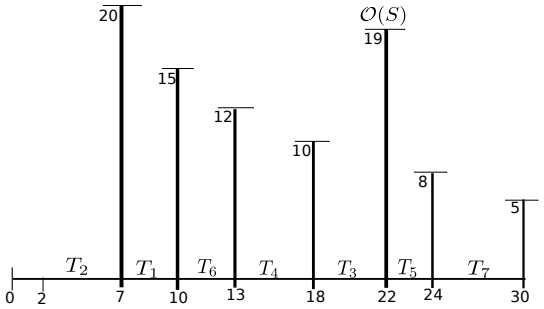


Figura 3.9: Observemos en este ejemplo que la suma de los tiempos de procesamiento de los primeros 4 trabajos liberados al tiempo r^1 con la cola más larga es igual a $r^2 = 15$.

Sea i, $i \leq k$, el trabajo de aumento más temprano de J_1 durante la construcción del calendario σ, tal que $t_i + p_i > r_2$, y sea $q^{'}$ la cola más larga entre todas las colas de los trabajos liberados al tiempo r_2.
Lema 15. *Si $q_i \geq q^{'}$, entonces σ es óptima.*

Demostración. El calendario σ no contiene trabajos emergentes porque $q_i \geq q^{'}$, entonces, por el Lema 6 el calendario σ es óptimo. □

Debido al Teorema 1, a continuación, asumimos que $o(\sigma) \in J_2$ y que $B(\sigma)$ contiene al menos un trabajo emergente (en caso contrario, el calendario σ es óptimo por Lema 6).
Observación 9. $K(\sigma) \subset J_2$

Demostración. Por contradicción: supongamos que el kernel $K(\sigma)$ contiene al menos un trabajo $j \in J_1$. Entonces para todo trabajo $i \in J_1$ calendarizado antes del kernel $K(\sigma)$, se verifica $q_i \geq q_j$ (por la JE-heurística). Como $o(\sigma)$ es el trabajo con la cola más pequeña del kernel $K(\sigma)$ (por la JE-heurística

y la definición del kernel $K(\sigma)$), se tiene $q_j \geq q_{o(\sigma)}$ y entonces $q_i \geq q_{o(\sigma)}$. Esto significa que el calendario σ no puede tener trabajos emergentes, y llegamos a una contradicción. □

Observación 10. *Para el problema* $1|\{r_1, r_2\}, \{q_i\}|C_{\text{máx}}$, $E(\sigma) \subset J_1$ *y para cualquier trabajo emergente* $e \neq l$, $q_e \geq q_l$.

Demostración. La primera afirmación se debe al hecho que el trabajo calendarizado más temprano del kernel $K(\sigma)$ en el calendario σ pertenece al conjunto J_2 el cual, a su vez, proviene de la JE-heurística. La segunda afirmación se debe a que la JE-heurística calendariza todos los trabajos liberados al tiempo r_1 (en particular, los trabajos de $E(\sigma)$) en orden no creciente de sus colas (en particular, los liberados al tiempo r_2 después de los trabajos del kernel $K(\sigma)$) mientras que el trabajo l es el último trabajo calendarizado del conjunto $E(\sigma)$ en el calendario σ. □

Observación 11. *El kernel* $K(\sigma)$ *contiene los trabajos con la cola más larga del conjunto* J_2 *calendarizados en orden no creciente de sus colas.*

Demostración. Por la observación 10, $E(\sigma) \subset J_1$. Por definición de trabajo emergente activo, l es el último trabajo calendarizado antes del kernel $K(\sigma)$ liberado al tiempo r_1 y el primer trabajo con tiempo de completés más grande o igual a r_2. Puede no existir un trabajo del conjunto J_2 calendarizado antes del trabajo l en el calendario σ (el trabajo l empieza estrictamente antes del tiempo r_2). Entonces por la JE-heurística, el primer trabajo del kernel $K(\sigma)$ debe ser uno con la cola más larga del conjunto J_2 y los siguientes trabajos deben ser calendarizados en orden no creciente de sus colas. Además, todos estos trabajos pertenecen al conjunto J_2 por la Observación 9. □

Recordemos que para un JE-calendario S, el calendario complementario S_l siempre contiene un hueco antes de los trabajos del $K(S)$, es decir, el trabajo calendarizado más temprano del $K(S)$ empieza en su tiempo de liberación en el calendario S_l. En general, para cualquier trabajo emergente $e \in E(S)$, el calendario complementario S_e puede contener un hueco o no, dependiendo de la longitud del trabajo (comparada a la del trabajo l). Veremos esto con más detalle más adelante.

Observación 12. *El orden de los trabajos calendarizados después del tiempo* r_2, *y, antes y después del trabajo* e *(particularmente los trabajos del* $K(\sigma)$*) es el mismo en los calendarios* σ *y* σ_e.

Demostración. Observemos que todos los trabajos del kernel $K(\sigma)$ y los trabajos calendarizados después de este kernel en σ se liberaron al tiempo r_2. Por la JE-heurística, deberían haber sido calendarizados esos trabajos en orden no-creciente de sus colas en el calendario σ. Entonces, la JE-heurística también calendarizará todos los trabajos formados en el mismo orden en el calendario σ_e, es decir, los trabajos antes del trabajo e y después de este trabajo (liberados al tiempo r_2), en particular, unos del kernel $K(\sigma)$ serán incluidos en el mismo orden en ambos calendarios. □

Sea $J[e]$ el conjunto de todos los trabajos j calendarizados después de r^2 en el calendario σ tal que $q_e < q_j < q_{o(\sigma)}$. De la definición del conjunto $J[e]$ se deduce que todos los trabajos del conjunto $J[e]$ son calendarizados inmediatamente después del kernel $K(\sigma)$ y antes del trabajo e en σ_e.

En la figura 3.10 damos un ejemplo de un calendario σ y de los conjuntos $J[e]$ para ese calendario.

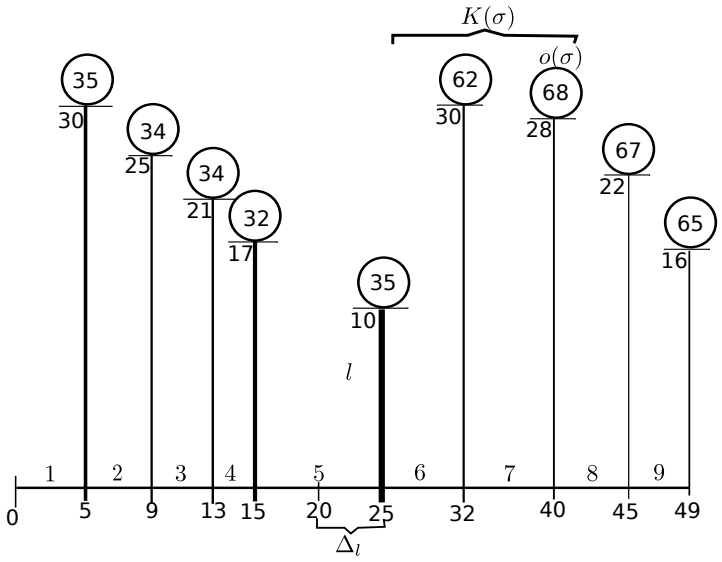


Figura 3.10: Observemos que en este calendario σ, el overfow job es el trabajo 7 y los trabajos emergentes son los trabajos {2,3,4,5}. Los conjuntos: $J[2] = \{\emptyset\}$, $J[3] = \{8\}$, $J[4] = \{8\}$ y $J[5] = \{8, 9\}$.

Observación 13. $J[e] \subset J_2$.

Demostración. Primero observemos que el conjunto $J[e]$ no tiene ningún trabajo con su cola igual que la del trabajo emergente e. Además, $K(\sigma) \subset J_2$ y $q_k > q_l$, para cada trabajo $k \in K(\sigma)$. Por lo tanto, por la JE-heurística, para cada trabajo $j \in J_1$ calendarizado en σ antes de $K(\sigma)$, se tiene $q_j \geq q_l$, y para cada trabajo $i \in J_1$ calendarizado después de $K(\sigma)$ se tiene, $q_i \leq q_l$. Entonces $q_i \leq q_e$ como $q_e \geq q_l$ y nuestra Observación sigue de la definición del conjunto $J[e]$. □

Proposición 1. *Sin perdida de generalidad, podemos asumir que todos los trabajos j con $q_j = q_e$ calendarizados despueś del kernel $K(\sigma)$ en σ son incluidos después del trabajo e en el calendario complementario σ_e.*
Observación 14. *En el calendario σ_e, los trabajos del kernel $K(\sigma)$ y los del conjunto $J[e]$ son desplazados a la izquierda por la misma cantidad, mientras que todos los trabajos j con $q_j \leq q_e$ son desplazados a la derecha por la misma cantidad.*

Demostración. Observemos que en la costrucción del calendario σ, la JE-heurística calendariza todos los trabajos en orden no creciente de sus colas después del tiempo r_2 (como todos los trabajos se liberan en este tiempo). En la construcción del calendario σ_e, los mismos trabajos más el trabajo e son disponibles después del tiempo r_2. En particular, se puede ver que si $p_e \geq \Delta_l$, todos los trabajos del kernel $K(\sigma)$ y los del conjunto $J[e]$ serán desplazados a la izquierda por Δ_l, y si $p_e < \Delta_l$ este desplazamiento será igual a p_e, mientras que cualquier trabajo j con $q_j \leq q_e$ será calendarizado después del trabajo e y será desplazado a la derecha correspondientemente. □

Un calendario sigma σ_e puede contener o no contener un hueco; esto depende del tiempo de procesamiento del trabajo emergente e y del valor de Δ_l (el empuje del Kernel de σ). En el siguiente Lema, damos las condiciones necesarias para saber cuándo el calendario σ_e tiene o no tiene hueco.
Lema 16. *Sea e un trabajo emergente del calendario σ. Y σ_e el calendario complementario del calendario σ.*

1. *Si $p_e > \Delta_l$, entonces σ_e tiene un hueco de longitud $p_e - \Delta_l$.*
2. *Si $p_e \leq \Delta_l$, entonces σ_e no tiene hueco.*

Demostración. Primero observemos que Δ_l es ahora $c_l(\sigma) - r_2$. Por la definición de σ_e, ningún trabajo $j \in J_1$ con $q_j < q_e$ calendarizado después de $K(\sigma)$

en σ será calendarizado después de $K(\sigma)$ en σ_e. Si $p_e \geq \Delta_l$, en calendario σ_e, los trabajos del $K(\sigma)$ y del conjunto $J[e]$ tendrán el desplazamiento a la izquierda correspondiente a la longitud Δ_l (el cual corresponde al máximo tiempo posible por Observación 1). Entonces σ_e tendrá un hueco de longitud $p_e - \Delta_l$. Si $p_e < \Delta_l$, los trabajos del kernel $K(\sigma)$ y los del conjunto $J[e]$ tendrán un desplazamiento a la izquierda correspondiente a la longitud p_e, de aquí σ_e no tiene hueco. □

En la figura 3.11 damos un ejemplo de un calendario generado por la JE-heurística. En base a este calendario, damos un ejemplo de un calendario complementario que contiene hueco y uno que no contiene (ver figuras 3.12 y 3.13, respectivamente).

$r_1 = 0$	$p_1 = 5$	$q_1 = 25$
$r_2 = 0$	$p_2 = 4$	$q_2 = 20$
$r_3 = 0$	$p_3 = 4$	$q_3 = 21$
$r_4 = 0$	$p_4 = 2$	$q_4 = 17$
$r_5 = 0$	$p_5 = 10$	$q_5 = 10$
$r_6 = 20$	$p_6 = 7$	$q_6 = 30$
$r_7 = 20$	$p_7 = 8$	$q_7 = 28$
$r_8 = 20$	$p_8 = 5$	$q_8 = 22$
$r_9 = 20$	$p_9 = 4$	$q_9 = 16$

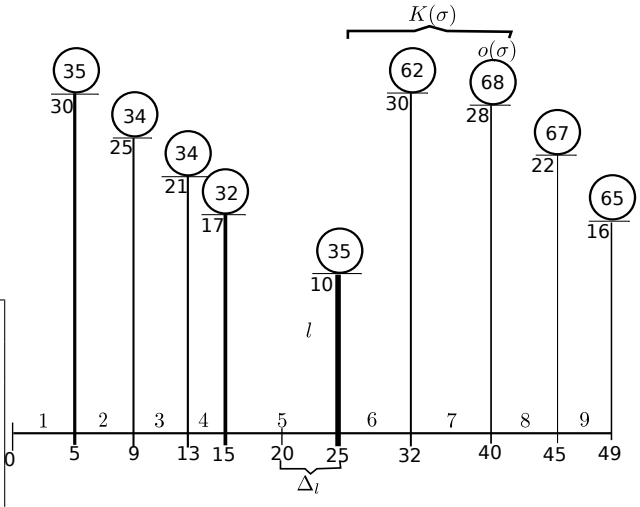


Figura 3.11: Ejemplo de una instancia y del calendario σ generado por la JE-heurística para esta instancia. En este ejemplo $r^1 = 0$ y $r^2 = 20$. Observemos que el tiempo de completés del trabajo emergente activo en el calendario σ es, $c_l(\sigma) = 25$, entonces el retraso del kernel es $\Delta_l = c_l(\sigma) - r^2 = 25 - 20 = 5$.

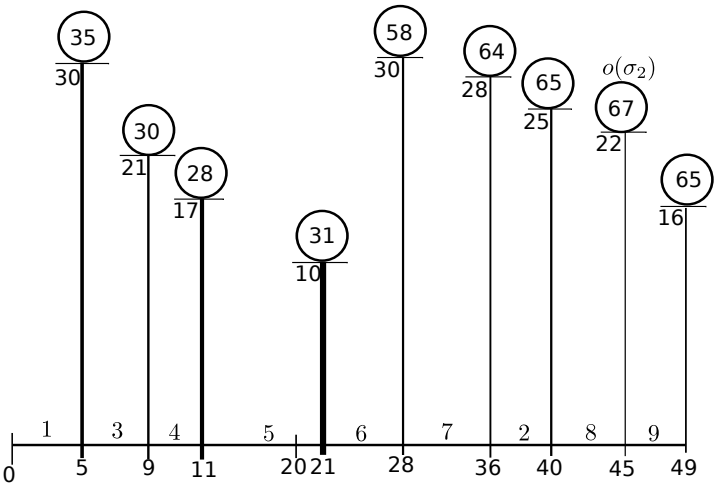


Figura 3.12: Calendario complementario σ_2 (Recordemos que σ_2 es el calendario resultante al recalendarizar el trabajo emergente 2). Observemos que el trabajo emergente 2 tiene tiempo de procesamiento $p_2 = 4$. $p_2 < \Delta_l = 5$. Observemos que el calendario σ_2 no contiene hueco.

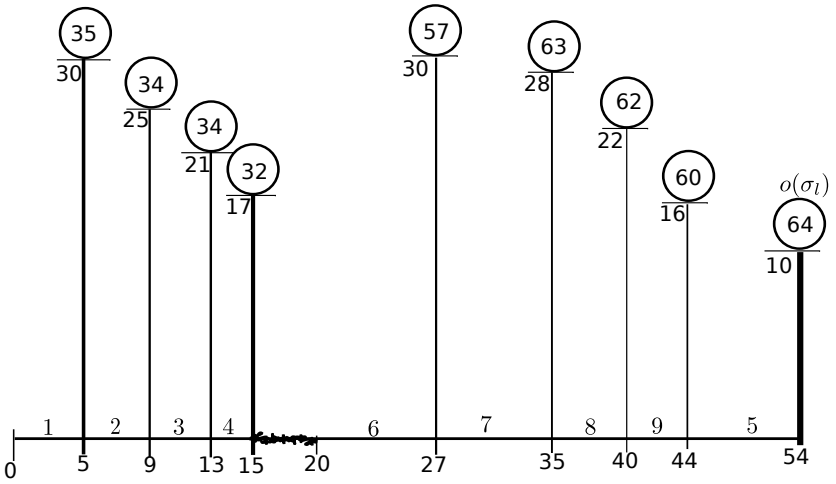


Figura 3.13: Calendario σ_5. El trabajo emergente 5 tiene tiempo de procesamiento $p_5 = 10$. Observemos que : $p_5 > \Delta_l = 5$ y que el calendario complementario σ_5 contiene hueco.

Por el Lema (16) definimos la longitud del hueco δ_e en el calendario σ_e de la

manera siguiente:

$$\delta_e = \begin{cases} 0 & si \quad p_e \leq \Delta_l \\ p_e - \Delta_l & if \quad p_e > \Delta_l \end{cases}$$

para todo $e \in E(\sigma)$.

Usando la definición de δ_e, definimos el tiempo de completés del trabajo emergente e en el calendario σ_e de la siguiente manera:

Lema 17. *El tiempo de completés total del trabajo e en σ_e es,*

$$C_e(\sigma_e) \quad = \quad c_{o(\sigma)}(\sigma) + \sum_{j \in J[e]} p_j + \delta_e + q_e.$$

Demostración. Por la definición del conjunto $J[e]$, el trabajo e será calendarizado en σ_e después de todos los trabajos de $J[e]$ y antes de todos los trabajos i con $q_i = q_e$ (ver Proposición 1). Por la Observación 1, Δ_l es el desplazamiento a la izquierda máximo posible para los trabajos del $K(\sigma)$ en σ_e. El tiempo de completés total del trabajo e en σ_e dependerá de la longitud de este desplazamiento a la izquierda. Consideremos los siguientes dos casos: Si $p_e > \Delta_l$, entonces los trabajos del $K(\sigma)$ y los del conjunto $J[e]$ tendrán un desplazamiento de longitud Δ_l a la izquierda en el calendario σ_e. Recordemos que el último trabajo del kernel $K(\sigma)$ calendarizado en σ es $o(\sigma)$. Entonces

$$C_e(\sigma_e) \quad = \quad (c_{o(\sigma)}(\sigma) + \sum_{j \in J[e]} p_j) - \Delta_l + p_e + q_e.$$

Sea $\delta_e = p_e - \Delta_l$, entonces $p_e = \delta_e + \Delta_l$. Por lo tanto

$$\begin{aligned} C_e(\sigma_e) &= c_{o(\sigma)}(\sigma) + \sum_{j \in J[e]} p_j - \Delta_l + \delta_e + \Delta_l + q_e \\ &= c_{o(\sigma)}(\sigma) + \sum_{j \in J[e]} p_j + \delta_e + q_e. \end{aligned}$$

En este caso σ_e tiene un hueco de longitud δ_e.

Si $p_e \leq \Delta_l$, entonces, por la definición del conjunto $J[e]$ y la construcción del calendario σ_e, los trabajos del kernel $K(\sigma)$ y del conjunto $J[e]$ serán desplazados por la cantidad p_e en el calendario σ_e. Esto implica que

$$\begin{aligned} C_e(\sigma_e) &= (c_{o(\sigma)}(\sigma) + \sum_{j \in J[e]} p_j) - p_e + p_e + q_e \\ &= c_{o(\sigma)}(\sigma) + \sum_{j \in J[e]} p_j + q_e. \end{aligned}$$

Vemos que en este caso σ_e no tiene hueco. □

En la figura 3.14 explicamos cómo podemos aplicar el Lema 17 para deducir el tiempo de completés total de un trabajo emergente e en el calendario σ_e a partir de la estructura del calendario σ. En la figura 3.15 ilustramos el calendario σ_e para verificar que efectivamente el tiempo de completés del trabajo e es igual que el deducido en la figura 3.14.

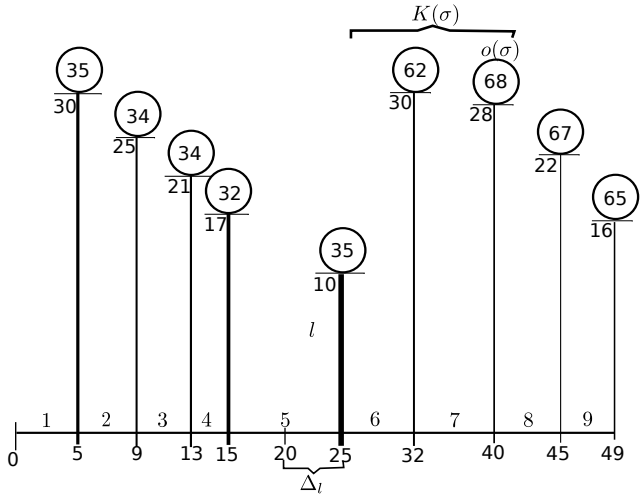


Figura 3.14: En este ejemplo mostramos que a partir de cualquier calendario σ dado, podemos deducir mediante la ecuación del Lema 17 el tiempo de completés del trabajo emergente e en el calendario σ_e. Por ejemplo, el tiempo de completés del trabajo emergente 5 en σ_5 es: $C_5(\sigma_5) = c_{o(\sigma)}(\sigma) + \sum_{j \in J[5]} p_j + \delta_5 + q_5 = 40 + 5 + 4 + 5 + 10 = 64$ debido a que $c_{o(\sigma)}(\sigma) = 40$, $\sum_{j \in J[5]} p_j = 5 + 4$, $\delta_5 = 5$ y $q_5 = 10$.

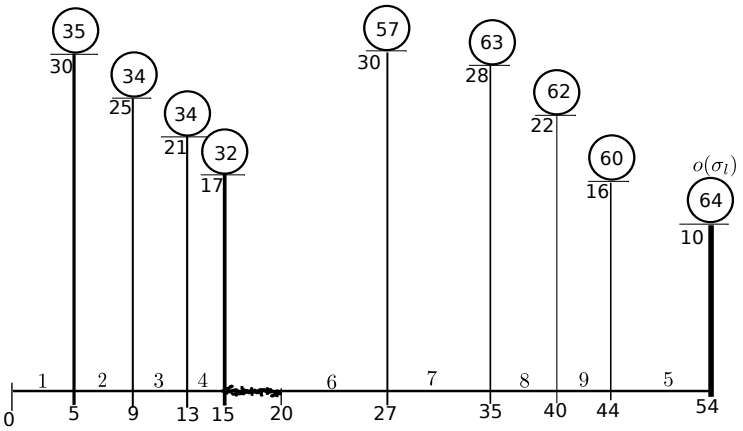


Figura 3.15: Calendario σ_5. Observemos que el tiempo de completés del trabajo 5 es: $C_5(\sigma_5) = 64$.

Consideremos la siguiente desigualdad:

$$c_{o(\sigma)}(\sigma) + \sum_{j \in J[e]} p_j + \delta_e + q_e \geq C_{o(\sigma)}(\sigma). \tag{3.1}$$

Lema 18. *Si $p_e \geq \Delta_l$ y la desigualdad (3.1) para el trabajo $e \in E(\sigma)$ se cumple, entonces en un calendario óptimo S_{opt}, el trabajo e está calendarizado antes del kernel $K(\sigma)$.*

Demostración. Como $p_e \geq \Delta_l$, σ_e tiene un hueco por Lema 16 y el kernel $K(\sigma)$ empieza en su tiempo más temprano de inicio $r(K(\sigma)) = r_2$ en el calendario σ_e. Por la desigualdad (3.1), $C_e(\sigma_e) \geq C_{o(\sigma)}(\sigma)$; por eso, $|\sigma_e| \geq |\sigma|$. Entonces tenemos que el trabajo e no puede ser calendarizado despué del kernel $K(\sigma)$ en el calendario S_{opt}. □

Como una consecuencia del Lema 18 obtenemos el siguiente resultado.
Teorema 4. *Si $p_e \geq \Delta_l$ para cada trabajo $e \in E(\sigma)$ y la desigualdad (3.1) se satisface para el trabajo e, entonces el calendario σ es óptimo.*

Demostración. Recordemos que si el calendario σ no es óptimo, el kernel $K(\sigma)$ debe ser reiniciado más temprano. Esto es posible solo si al menos un trabajo $e \in E(\sigma)$ es recalendarizado después del kernel $K(\sigma)$. Supongamos

que σ_e es un calendario con el mejor (más pequeño) makespan que el calendario σ. Entonces, similarmente, se verifica que, por la condición del Teorema, la desigualdad (3.1) se cumple $C_e(\sigma_e) \geq C_{o(\sigma)}(\sigma)$ para cada $e \in E(\sigma)$. Entonces por Lema 18, el calendario σ debe ser óptimo. □

Ahora, si el trabajo emergente activo cumple la desigualdad (3.1), entonces podemos deducir que todo trabajo emergente e con tiempo de procesamiento $p_e \geq p_l$ también cumple la desigualdad (3.1), y por lo tanto, por el Lema 18 se puede concluir que cualquiera de estos trabajos está calendarizado antes del kernel $K(\sigma)$. Para demostrar esto, definimos el siguiente conjunto: Sea $E(S,l) = \{e \in E(S) \setminus \{l\} \mid p_e \geq p_l\}$.

Lema 19. *Si la desigualdad (3.1) se cumple para el trabajo l, entonces en S_{opt} cualquier trabajo $e \in E(\sigma,l)$ está calendarizado antes del kernel $K(\sigma)$.*

Demostración. Por el Lema 18, l está calendarizado en S_{opt} antes del $K(S_{opt})$.

Sea k el último trabajo calendarizado de $J[l]$ en σ (recordemos que $J[l]$ es el conjunto de todos los trabajos j calendarizados después de r_2 en el calendario σ tal que $q_l \leq q_j < q_{o(\sigma)}$). Por la JE-heurística, el trabajo k tiene la cola menor de los trabajos del conjunto $J[l]$. Por la definición del conjunto $J[l]$, sin perdida de generalidad, supongamos que la JE-heurística calendarizara a l después del trabajo k en el calendario σ_l. Además, l va a empezar al tiempo de completés del trabajo k en el calendario σ_l, es decir, $t_{l(\sigma_l)} = c_{k(\sigma_l)}$ (por Proposición 1 y debido a que la JE-heurística, cualquier trabajo i calendarizado después de k en σ, su cola $q_i < q_k$ y por lo tanto $q_i < q_l$, por la definición del conjunto $J[l]$).

Se puede ver que para todo trabajo $e \in E(\sigma,l)$, $\delta_e \geq \delta_l$ (porque $p_e \geq p_l$ y por la definición de δ_e).

Además, observemos que el trabajo k está calendarizado después de los trabajos del $K(\sigma)$ en σ_e y el trabajo e puede estar calendarizado antes o después del trabajo k en σ_e. Respectivamente, distinguimos los siguientes dos casos:

- Si el trabajo e está calendarizado en σ_e después del trabajo k, el tiempo de inicio del trabajo e en σ_e es igual al tiempo de inicio del trabajo l en σ_l, es decir, $t_{e(\sigma_e)} = t_{l(\sigma_l)}$ ya que para todo trabajo i calendarizado después de l en σ_l, $q_i < q_l$, y $q_i < q_e$ ya que $q_e \geq q_l$. Como la desigualdad (3.1) se cumple para l, además se cumple para el trabajo e porque

$\delta_e \geq \delta_l$ y $q_e \geq q_l$. Por Lema 18, e está calendarizado antes del $K(\sigma)$ en el calendario S_{opt}.

- Si el trabajo e está calendarizado en σ_e antes del trabajo k, los trabajos calendarizados después de e en σ_e (en particular los trabajos del conjunto $J[l]$) son desplazados a la derecha, por lo tanto, el tiempo de completés del trabajo k en σ_e es mayor que el tiempo de completés del trabajo k en σ_l. Además, dado que $\delta_e \geq \delta_l$, el tiempo de completés del trabajo k en σ_e es más grande o igual que el tiempo de completés del trabajo l en σ_l ($c_{k(\sigma_e)} \geq c_{l(\sigma_l)}$). Como $q_k \geq q_l$, tenemos $C_{k(\sigma_e)} \geq C_{l(\sigma_l)}$ y por lo tanto $|\sigma_e| \geq |\sigma_l|$. Entonces, el trabajo l no puede ser calendarizado después del $K(\sigma)$ en el calendario S_{opt} ya que el trabajo l satisface la desigualdad (3.1), en consecuencia $|\sigma_e| \geq |\sigma|$.

Hemos mostrado que el trabajo e no puede estar calendarizado después del $K(\sigma)$ en el calendario S_{opt} y el Lema está demostrado. □

A continuación, denotamos por k al trabajo con tiempo de completés total máximo en el calendario σ entre todos los trabajos calendarizados después de los trabajos del conjunto $J[e]$. Se puede ver que el trabajo k está en el conjunto J_1 o J_2.

Observación 15. *Si el overflow job del calendario σ_e está calendarizado después de r^2, entonces el overflow job pertenece al conjunto $\{o(\sigma), e, k\}$.*

Demostración. De la Observación 12, tenenos que los trabajos calendarizados después del tiempo r_2 y antes del trabajo e en el calendario σ_e tienen el mismo orden de procesamiento como en el calendario σ. Del mismo modo, por la Observación 14 los trabajos del kernel $K(\sigma)$ y los del conjunto $J[e]$ tienen el mismo desplazamiento p_e a la izquierda en el calendario σ_e. Justo después de los últimos trabajos, el trabajo e es calendarizado en el calendario σ_e (Proposición 1) y, el orden de procesamiento y los tiempos de inicio/completés de los trabajos siguientes restantes no serán cambiados. Nuestra proposición sigue de las definiciones de los trabajos $o(\sigma)$, e and k. □

Un caso en el que podemos asegurar que el calendario σ_e es óptimo, es el siguiente:

Lema 20. *El calendario σ_e es óptimo si $p_e \geq \Delta_l$ y $o(\sigma) = o(\sigma_e)$.*

Demostración. Dado que $p_e > \Delta_l$, los trabajos del kernel $K(\sigma)$ son desplazados a la izquierda en el calendario σ_e por un tiempo Δ_l. Por definición de Δ_l y debido al hecho que todos los trabajos del kernel $K(\sigma)$ se liberan al tiempo r_2, el trabajo más temprano de este kernel empieza en su tiempo de inicio más temprano en σ_e, es decir, $t_{K(\sigma)}(\sigma_e) = r_2$. Por la JE-heurística, para cualquier trabajo $i \in K(\sigma) \setminus \{o(\sigma)\}$, tenemos $q_i \geq q_{o(\sigma)}$. Tenemos que el tiempo de completés total del trabajo $o(\sigma)$ en el calendario σ_e es el mejor posible y el calendario σ_e es óptimo. □

Otro caso en el que podemos asegurar que el calendario σ_e es óptimo, es el siguiente:
Teorema 5. *Si el trabajo $o(\sigma_e)$ es calendarizado antes del $K(\sigma)$ en el calendario σ_e, entonces el calendario σ_e es óptimo.*

Demostración. Dado que $K(\sigma) \subset J_2$ (por Observación 9) y que el primer trabajo del $K(\sigma)$ es el trabajo con la cola más larga del conjunto J_2, todos los trabajos calendarizados antes del kernel $K(\sigma)$ están en el conjunto J_1. En este caso el overflow job $o(\sigma_e)$ está en el conjunto J_1. Por la JE-heurística, todos los trabajos calendarizados antes de kernel $K(\sigma)$ son calendarizados en orden no creciente de sus colas (ver también el Lema 7). Entonces, en este caso no puede existir un calendario $\sigma_e^{'}$ tal que $\left|\sigma_e^{'}\right| < |\sigma_e|$ y por ello e calendario σ_e es óptimo. □

En seguida ilustramos el Teorema 5. Los calendarios σ y σ_l se interpretan en las figuras 3.16 y 3.17, respectivamente (los números dentro de los crculos son los tiempos de completés total de los trabajos correspondientes).

$r_1 = 0$	$p_1 = 5$	$q_1 = 40$
$r_2 = 0$	$p_2 = 4$	$q_2 = 30$
$r_3 = 0$	$p_3 = 10$	$q_3 = 8$
$r_4 = 10$	$p_4 = 10$	$q_4 = 16$
$r_5 = 10$	$p_5 = 6$	$q_5 = 11$

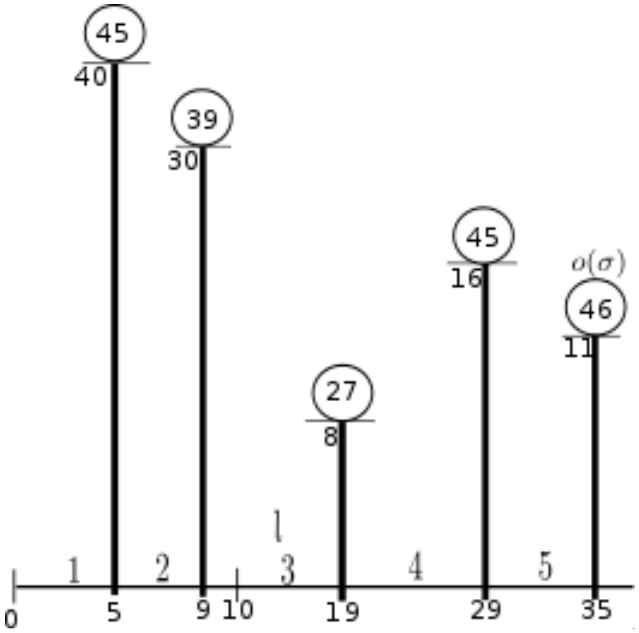


Figura 3.16: Calendario σ.

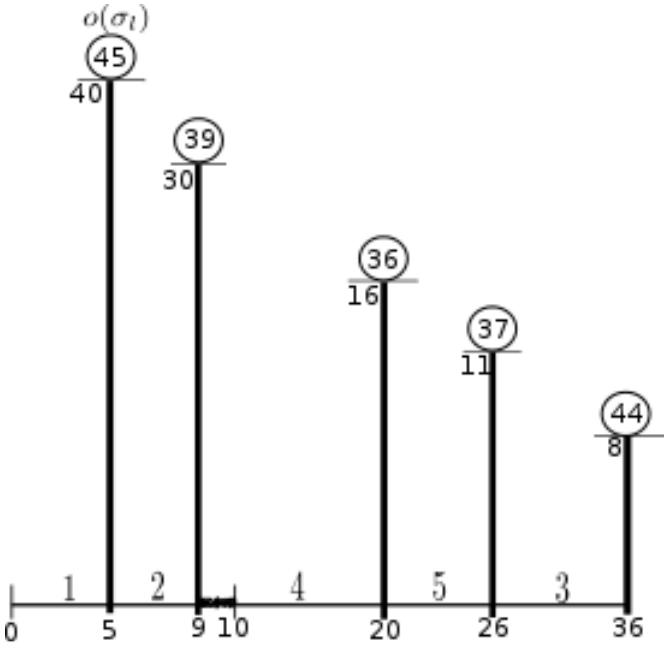


Figura 3.17: Calendario σ_l, en este calendario el overflow job $o(\sigma_l)$ es calendarizado antes del kernel $K(\sigma)$.

Observación 16. *Si en el calendario σ_e el overflow job $o(\sigma_e)$ es calendarizado antes del kernel $K(\sigma)$, entonces $o(\sigma_e) \notin E(\sigma) \setminus \{e\}$.*

Demostración. Por la definición de trabajo emergente, todos los trabajos del conjunto $E(\sigma)$ tienen la cola más corta que los trabajos del kernel $K(\sigma)$, mientras que estos trabajos, excepto el trabajo e, son calendarizados antes de los trabajos de este kernel en el calendario σ_e. Entonces, ninguno de ellos (especialmente el trabajo e) puede ser overflow job en el calendario σ_e. □

3.3.1. Propiedades para trabajos de igual longitud en el conjunto J_1

En esta sección consideramos casos especiales para el problema $1|r_j, q_j|C_{max}$ cuando todos los trabajos liberados al tiempo r_1 tienen el mismo tiempo de procesamiento p. Especificamos este caso como $1\,|\{r_1, r_2\}, q_j, p_{r_1,j} = p|\,C_{max}$ ($p_{r_1,j} = p$ designa que todos los trabajos liberados en el tiempo r_1 tienen

tiempo de procesamiento p).

Recordemos que el lado izquierdo de la desigualdad (3.1) es el tiempo de completés total del trabajo e en el calendario σ_e. En la sección 3.3, denotamos como k al trabajo con tiempo de completés total máximo en el calendario σ entre todos los trabajos calendarizados después de los trabajos del conjunto $J[e]$.

Nuestro siguiente objetivo, es definir el tiempo de completés del trabajo k en el calendario σ_e, para lograrlo, definimos el siguiente conjunto:

$J(e,k) = \{i \mid q_k \leq q_i \leq q_e\}$.

De la definición de la longitud de hueco δ_e (sección 3.3) y del conjunto $J(e,k)$, podemos definir el tiempo de completés total del trabajo k en el calendario σ_e.

Lema 21. *El tiempo de completés total del trabajo k en el calendario σ_e es:*

$$C_k(\sigma_e) \;=\; c_{o(\sigma)}(\sigma) + \sum_{j \in J[e]} p_j + \delta_e + \sum_{i \in J(e,k)} p_i + p_k + q_k.$$

Demostración. Se demuestra de manera análoga a la demostración del Lema 17. □

Ahora consideremos la siguiente desigualdad,

$$c_{o(\sigma)}(\sigma) + \sum_{j \in J[e]} p_j + \delta_e + \sum_{i \in J(e,k)} p_i + p_k + q_k \geq C_{o(\sigma)}(\sigma) \tag{3.2}$$

Lema 22. *En el problema $1\,|\{r_1, r_2\}, q_j, p_{r_1,j} = p|\, C_{max}$, en cualquier calendario σ_e hay un hueco antes del kernel $K(\sigma)$. Además, $\delta_e = \delta_l$ para cada trabajo $e \in E(\sigma) \setminus \{l\}$.*

Demostración. Tenemos que $p_e > \Delta_l$, porque $p_e = p_l = p$ para todo $e \in E(\sigma) \setminus \{l\}$. Por Lema 16, el calendario σ_e tiene un hueco de longitud $p_e - \Delta_l$ y $\delta_e = \delta_l$ para todo $e \in E(\sigma) \setminus \{l\}$. □

Lema 23. *Si el tiempo de completés del trabajo k cumple la desigualdad (3.2), entonces cada trabajo $e \in E(\sigma)$ es calendarizado antes del kernel $K(\sigma)$ en S_{opt}.*

Demostración. La desigualdad (3.2) implica que $C_k(\sigma_l) \geq C_{o(\sigma)}(\sigma)$; por eso, $|\sigma_l| \geq |\sigma|$. Como el tiempo de completés total del trabajo k en σ es menor que el tiempo de completés total del overflow job $o(\sigma)$ (en caso contrario, el trabajo k debería ser overflow job en el calendario σ) y $C_k(\sigma_l) \geq C_{o(\sigma)}(\sigma)$, entonces el trabajo l no puede ser calendarizado después del kernel $K(\sigma)$ en el calendario S_{opt}. Por Lema 22, $\delta_e = \delta_l$ para cada trabajo emergente e. Entonces el tiempo de inicio del trabajo k en el calendario σ_e es el mismo como en σ_l. Por lo tanto, $C_k(\sigma_e) \geq C_{o(\sigma)}(\sigma)$ y $|\sigma_e| \geq |\sigma|$ para cada trabajo $e \in E(\sigma)$. Entonces ningún trabajo $e \in E(\sigma)$ puede ser calendarizado después del kernel $K(\sigma)$ en el calendario S_{opt}. □

Teorema 6. *Para el problema* $1\,|\{r_1, r_2\}, q_j, p_{r_1,j} = p|\,C_{max}$*, el calendario* σ *es óptimo si*

1. *La desigualdad (3.1) se satisface para el trabajo* l *en el calendario* σ_l*.*

2. *La desigualdad (3.2) se satisface para el trabajo* k *en el calendario* σ_l*.*

Demostración. Para la primera afirmación, por el Lema 19, cualquier trabajo $e \in E(\sigma)$ es calendarizado en S_{opt} después del kernel $K(\sigma)$. Esto implica que cada trabajo $e \in E(\sigma) \setminus \{l\}$ satisface la desigualdad (3.1) y por el Teorema 4, σ es óptima.

Para la segunda afirmación, por Lema 23 cada trabajo $e \in E(\sigma)\setminus\{l\}$ debe ser calendarizado antes del kernel $K(\sigma)$ in S_{opt}, lo que implica que el calendario σ es óptimo. □

3.3.2. Trabajos emergentes cortos

En esta subsección estudiamos el caso cuando la aplicación de un trabajo emergente corto, es decir, uno con tiempo de procesamiento menor que Δ_l, nos da propiedades útiles acerca de optimalidad. Recordemos que el Lema 16 nos dice que al recalendarizar un trabajo emergente corto e del calendario σ, el calendario resultante σ_e no tiene hueco. En está subsección, sea j el trabajo del conjunto $J[e]$ calendarizado al último en el calendario σ. Y sea k el trabajo con el tiempo de completés total máximo entre todos los trabajos calendarizados después de los trabajos de $J[e]$ en el calendario σ.

La Propiedad 1, junto con el Lema 24 y la observación 17 (presentados un poco más adelante), nos ayudaran a demostrar el resultado principal de esta subsección, el Teorema 7.

Propiedad 1. *Si $p_e \leq \Delta_l$ entonces,*

1. *$c_e(\sigma_e) = c_j(\sigma)$ si el conjunto $J[e]$ no es vacío.*
2. *$c_e(\sigma_e) = c_{o(\sigma)}(\sigma)$ si el conjunto $J[e]$ es vacío.*

Demostración. Si el conjunto $J[e]$ no es vacío, entonces consideremos el conjunto de los trabajos emergentes del calendario σ incluidos entre el trabajo e y el kernel $K(\sigma)$, los trabajos del kernel $K(\sigma)$ y unos del conjunto $J[e]$.Observemos que esos son los trabajos que serán incluidos antes del trabajo e en el calendario σ_e. Esos trabajos serán desplazados a la izquierda por p_e unidades de tiempo (ver Lema 16). Dado que el trabajo j es uno de esos trabajos, entonces el trabajo j será desplazado a la izquierda por las mismas unidades de tiempo, y el trabajo e empezará inmediatamente después del trabajo j en el calendario σ_e (si hay trabajos calendarizados después del kernel σ con cola igual a q_e, aquellos serán incluidos después del trabajo e, por la Proposición 1). Tenemos que el tiempo de completés del trabajo e en el calendario σ_e es igual al del trabajo j en el calendario σ.

La demostración es análoga cuando el conjunto $J[e]$ es vacío. □

Lema 24. *Si $p_e < \Delta_l$, entonces $C_e(\sigma_e) < C_{o(\sigma)}(\sigma)$.*

Demostración. Por la Propiedad 1, $c_e(\sigma_e) = c_j(\sigma)$ si el conjunto $J[e]$ no es vacío y $q_e < q_j$ (por la definición del conjunto $J[e]$). Esto implica, $C_e(\sigma_e) < C_j(\sigma)$. Pero $C_j(\sigma) < C_{o(\sigma)}(\sigma)$ y nuestra afirmación se cumple. Si el conjunto $J[e]$ es vacío, por la propiedad 1, $c_e(\sigma_e) = c_{o(\sigma)}(\sigma)$ y $q_e < q_{o(\sigma)}$ (por la definición del overflow job), entonces $C_e(\sigma_e) < C_{o(\sigma)}(\sigma)$. □

Observación 17. *Si $p_e < \Delta_l$, entonces $C_k(\sigma_e) = C_k(\sigma)$.*

Demostración. Dado que el calendario σ_e no tiene hueco (Lema 16) todos los trabajos emergentes calendarizados entre el trabajo e y el kernel $K(\sigma)$, los trabajos del kernel $K(\sigma)$ y unos del conjunto $J[e]$ tendrán el mismo desplazamiento a la izquierda de p_e unidades en el calendario σ_e. Y a la vez, todos los trabajos calendarizados después de los trabajos del conjunto $J[e]$ en el calendario σ no tendrán ningún desplazamiento en el calendario σ_e porque

el trabajo e será incluido antes de aquellos. Entonces, el tiempo de inicio del trabajo k en ambos calendarios σ_e y σ es el mismo y nuestra afirmación es válida. □

Teorema 7. *Si* $p_e < \Delta_l$, *entonces* $|\sigma_e| \leq |\sigma|$.

Demostración. Por la Proposición 15 los overflow jobs potenciales en el calendario σ_e son $o(\sigma)$, e y k. Dado que los trabajos del kernel $K(\sigma)$ tienen un desplazamiento a la izquierda de tamao p_e en el calendario σ_e, $C_{o(\sigma)}(\sigma_e) = C_{o(\sigma)}(\sigma) - p_e$ y entonces $C_{o(\sigma)}(\sigma_e) < C_{o(\sigma)}(\sigma)$. Por Lema 24, $C_e(\sigma_e) < C_{o(\sigma)}$ y por Proposición 17, $C_k(\sigma_e) = C_k(\sigma)$. Pero $C_k(\sigma) \leq C_{o(\sigma)}(\sigma)$ consecuentemente $C_k(\sigma_e) \leq C_{o(\sigma)}(\sigma)$. Por eso, el tiempo de completés total de los trabajos $o(\sigma)$, e y k en el calendario σ_e no es más que en el calendario σ. □

El último resultado presentado en esta tesis, es el siguiente:
Teorema 8. *El calendario* σ_e *es óptimo si* $p_e < \Delta_l$, $k \in J_1$ *y* $k = o(\sigma_e)$.

Demostración. El calendario σ_e es mejor que el calendario σ debido al Teorema 7. Además, mostramos que no existe mejor calendario que σ_e. Es suficiente considerar un reordenamiento donde consideramos sólo los trabajos emergentes (ver Lema 6). Para este fin, primero observamos que $C_{o(\sigma)}(\sigma) \geq C_k(\sigma)$. En otras palabras, no es necesario desplazar el kernel $K(\sigma)$ formado a la izquierda y mostramos que el trabajo k no puede ser desplazado a la derecha. Además, si S un calendario en el cual algunos trabajos emergentes son recalendarizados después del kernel $K(\sigma)$.

Dado que $k \in J_1$, tenemos $q_e \geq q_k$ para cualquier trabajo emergente $e \in E(\sigma)$ (debido a la JE-heurística). Por eso, la JE-heurística incluirá cualquier trabajo emergente antes del trabajo k en el calendario S. Consideramos las siguientes dos posibilidades.

1. Si el calendario S no tiene hueco, entonces el conjunto de trabajos calendarizados antes del trabajo k en ambos calendarios σ_e y S es el mismo, el trabajo k se empieza y se completa en el calendario S al mismo tiempo que en el calendario σ_e. Entonces, $|S| = |\sigma_e|$.

2. Si el calendario S tiene un hueco, entonces dado que en el calendario S todos los trabajos emergentes recalendarizados están incluidos antes

del trabajo k, el último trabajo será desplazado a la derecha por una distancia igual al del hueco. Entonces $C_k(S) > C_k(\sigma_e)$ y por lo tanto $|S| > |\sigma_e|$.

Hemos mostrado que no existe un calendario mejor que σ_e y la observación es completada. □

Desafortunadamente, el Teorema 8 no puede ser generalizado para el caso cuando $k = o(\sigma_e)$ y $p_e \geq \Delta_l$, porque se forzaría a solucionar el problema de suma de subconjuntos (ver sección 3.2). Entonces el problema de cuello de botella se ocurre cuando esas condiciones se cumplen. Ilustramos esto en las figuras 3.18, 3.19 y 3.20.

$r_1 = 0$	$p_1 = 5$	$q_1 = 30$
$r_2 = 0$	$p_2 = 4$	$q_2 = 25$
$r_3 = 0$	$p_3 = 6$	$q_3 = 20$
$r_4 = 0$	$p_4 = 10$	$q_4 = 13$
$r_5 = 18$	$p_5 = 7$	$q_5 = 16$
$r_6 = 0$	$p_6 = 3$	$q_6 = 10$
$r_7 = 0$	$p_7 = 8$	$q_7 = 6$
$r_8 = 18$	$p_8 = 4$	$q_8 = 27$
$r_9 = 18$	$p_9 = 3$	$q_9 = 25$
$r_{10} = 18$	$p_{10} = 2$	$q_{10} = 3$

$c_1 = 0 + 5 = 5$	$C_1 = 5 + 30 = 35$
$c_2 = 5 + 4 = 9$	$C_2 = 9 + 25 = 34$
$c_3 = 9 + 6 = 15$	$C_3 = 15 + 20 = 35$
$c_4 = 15 + 10 = 25$	$C_4 = 25 + 13 = 38$
$c_8 = 25 + 4 = 29$	$C_8 = 29 + 27 = 56$
$c_9 = 29 + 3 = 32$	$C_9 = 32 + 25 = 57$
$c_5 = 32 + 7 = 39$	$C_5 = 39 + 16 = 55$
$c_6 = 39 + 3 = 42$	$C_6 = 42 + 10 = 52$
$c_7 = 42 + 8 = 50$	$C_7 = 50 + 6 = 56$
$c_{10} = 50 + 2 = 52$	$C_{10} = 52 + 3 = 55$

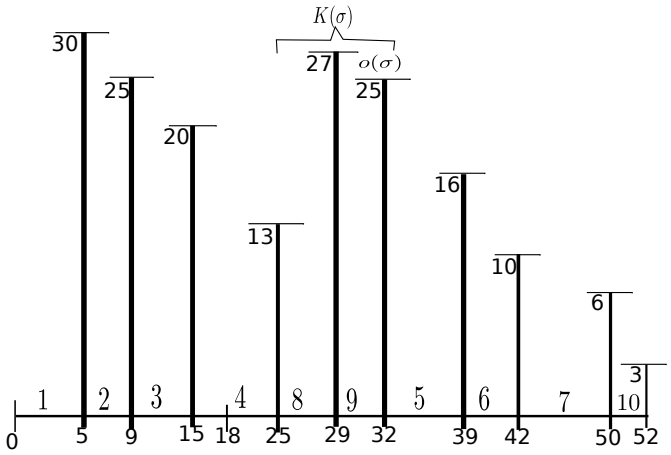


Figura 3.18: Calendario σ, $|\sigma| = 57$, con el trabajo emergente $l = 4$ y $o(\sigma) = 9$; $J[l] = 5, 6$ and $k = 7$.

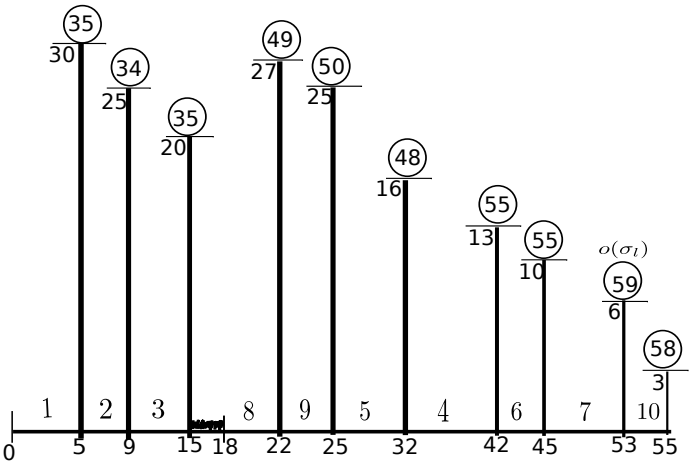


Figura 3.19: Calendario σ_l, su makespan es 59 y este no es óptimo porque $|\sigma_l| > |\sigma|$. Aquí la desigualdad (3.1) no se satisface para el trabajo l, el trabajo $k = 7$ es el overflow job en σ_l.

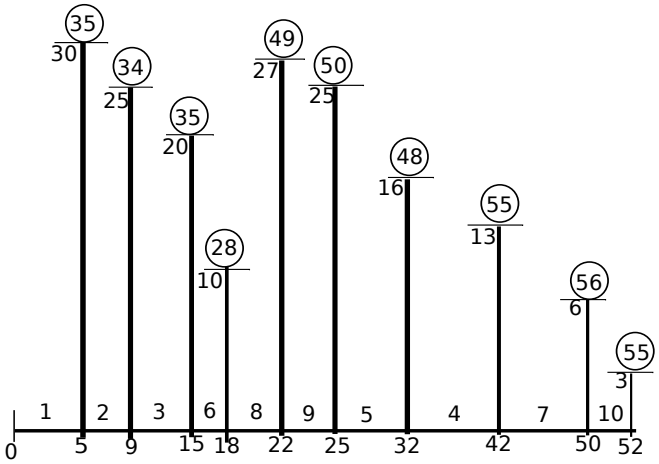


Figura 3.20: Un calendario óptimo S_{opt} con makespan 56.

Capítulo 4

Conclusión y trabajo a futuro

Basándonos en el análisis de los JE-calendarios, hemos establecido condiciones cuando una solución óptima puede ser obtenida eficientemente. El análisis presentado lleva a relaciones explícitas estrictas y obtener soluciones en tiempo polinomial de estos casos particulares. Realizamos un estudio similar para un caso especial NP-duro del problema $1|r_j, q_j|C_{\text{máx}}$ con sólo dos tiempos de liberación permitidos de los trabajos.

Nuestra investigación esencialmente dependió de la partición del calendario entero sobre dos tipos básicos de intervalos que contiene trabajos urgentes (kernel) y no urgentes (emergentes), y tratando de extraer los conflictos que pueden ocurrir entre estos dos tipos de intervalos. El análisis correspondiente permite una visión intuitiva sobre la perspectiva general y conflictos potenciales que pueden ocurrir mientras usamos reglas de una simple heurística para la solución del problema en la fase de calendarización.

Las condiciones obtenidas se pueden introducir en un esquema como las condiciones de dominio. Como parte de la conclusión describimos un algoritmo enúmerativo. En base a este esquema se pueden construir algoritmos heurísticos como Beam Search.

En el algoritmo enumerativo que consideramos cada nodo $h \in T$ representa un calendario completo, un poco más adelante describiremos la construcción del árbol T.

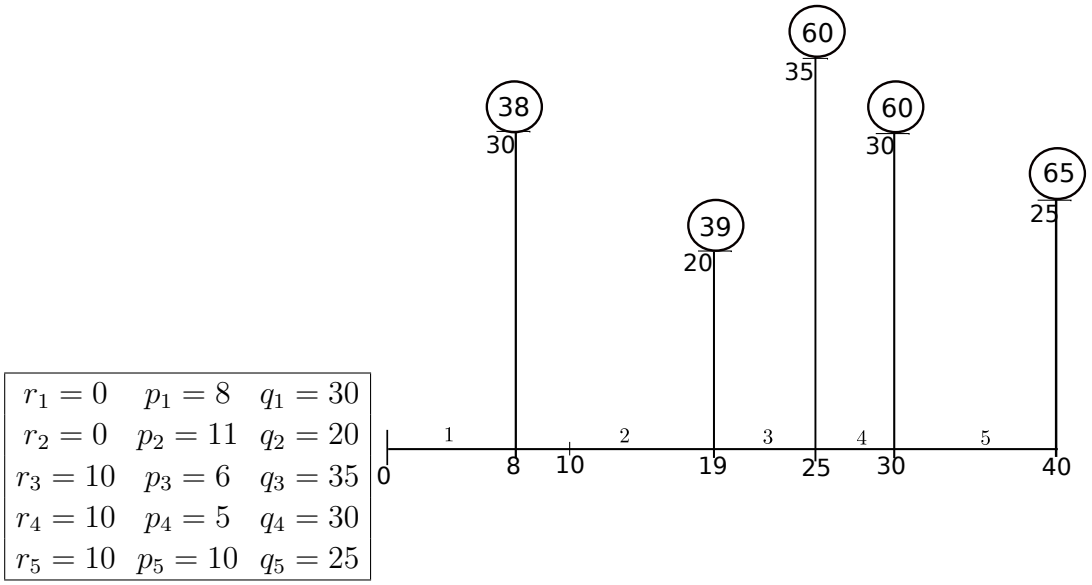
Enseguida describiremos las soluciones parciales que enumeramos en este árbol T.

Un calendario factible es particionado sobre dos tipos de segmentos, rígidos y flexibles. Los segmentos rígidos son los ocupados por los trabajos de los kernels, y los segmentos flexibles por el resto de los trabajos. Cada kernel es definido por dos segmentos flexibles, uno que le precede y otro que lo sucede. Denotamos como $B(K)$ el segmento flexible que precede inmediatamente al kernel K.

Para nuestro método de enumeración es importante estudiar la estructura de cada solución factible que estamos generando.

En el algoritmo descrito un poco más adelante, consideramos todos lo posibles subconjuntos de trabajos disponibles para calendarizados dentro de los segmentos flexibles surgidos, de esta manera garantizamos que nuestro árbol de solución va a contener la solución óptima. Inicialmente, kernel $K(\sigma)$, es detectado en calendario σ generado por la JE-heurística. El kernel $K(\sigma)$ define dos segmentos flexibles. El primero de estos segmentos es calendarizado pero restringiendo el conjunto de trabajos para ser incluidos dentro de este segmento a un subconjunto de los trabajos disponibles J. Así, ningún trabajo diferente de este subconjunto puede ser incluido sobre este segmento que es calendarizado por la JE-heurística aplicada a este subconjunto de trabajos.La JE-heurística es aplicada a el conjunto de trabajos de $J \cup K(\sigma)$ así todos ellos son incluidos antes del segmento que sucede a $K(\sigma)$ en σ es calendarizado (algunos trabajos del subconjunto J pueden ser incluidos después de los trabajos del kernel $K(\sigma)$). Una vez que todos los trabajos de $J \cup K(\sigma)$ son incluidos, se aplica la JE-heurística a el conjunto de los trabajos restantes. Un nuevo JE- calendario obtenido en este camino es denotado por $\sigma(J, K(\sigma))$. Un JE-calendario es creado para cada posible subconjunto J, y las correspondientes soluciones alternativas son representadas como brazos alternativos en la solución de árbol T. La raíz de este árbol representa el calendario σ, mientras que cada otro nodo $h \in T$ representa el JE-calendario $S^h = S^{h'}(J, K(S^{h'}))$ donde h' es el padre del nodo h. El correspondiente subconjunto J es asociado con cada arista $(h', h) \in T$.

En seguida ejemplificamos los conceptos anteriores.



$r_1 = 0$	$p_1 = 8$	$q_1 = 30$
$r_2 = 0$	$p_2 = 11$	$q_2 = 20$
$r_3 = 10$	$p_3 = 6$	$q_3 = 35$
$r_4 = 10$	$p_4 = 5$	$q_4 = 30$
$r_5 = 10$	$p_5 = 10$	$q_5 = 25$

Figura 4.1: Ejemplo de una instancia con 5 trabajos y del calendario σ para esta instancia. En este calendario, $K(\sigma) = \{3, 4, 5\}$. $B(K) = [0, 19]$.

Los conjuntos de trabajos que se pueden calendarizar en el intervalo $[0, 19]$ son: $\{\emptyset\}$, $\{1\}$, $\{2\}$ y $\{1, 2\}$. En las figuras 4.4, 4.2 y 4.3 ilustramos los calendarios originados por cada uno de estos subconjuntos.

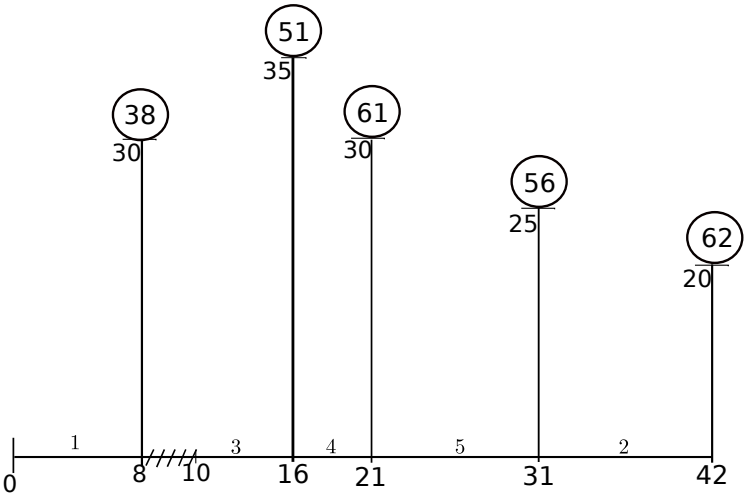


Figura 4.2: Calendario dado por el conjunto J={1}. Calendario óptimo.

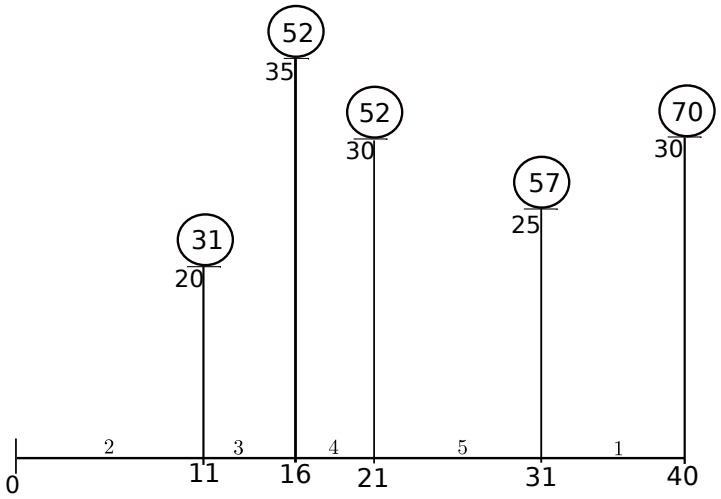


Figura 4.3: Calendario dado por el conjunto J={2}.

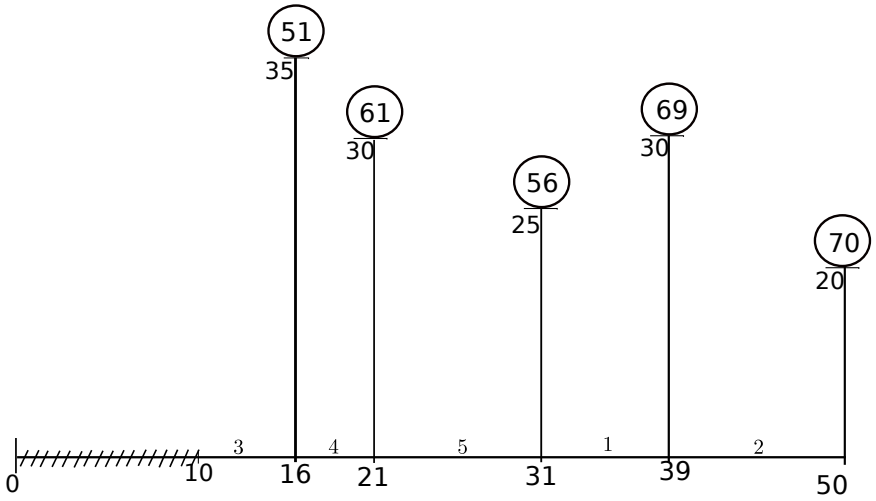


Figura 4.4: Calendario dado por el conjunto $J = \{\emptyset\}$.

El algoritmo que en seguida describimos, considera diferentes posibles subconjuntos para llenar el bin $B(K(S^h))$. El intervalo de tiempo de este segmento en el nuevo calendario construido por la JE-heurística no es prioridad fija, más bien depende de un conjunto particular J. El algoritmo en seguida descrito aplica busqueda en anchura explorando el nodo abierto próximo más

a la izquierda en el árbol (parcial). Sea $r(K) = min_{i \in K} r_i$.

Inicialmente, $h := 0$; S^h:=σ. Iterativamente, en cada nodo h:

Caso 1) Si el bloque crítico $B(\sigma)$ no contiene ningún trabajo emergente, entonces el calendario σ es óptimo (Teorema 6).

Caso 2) Para cualquier nodo sucedor h de nodo σ; si el bloque crítico del calendario S^h no contiene trabajos emergentes y $o(S^h) = o(\sigma)$, entonces parar S^h es óptimo.

Caso 3) Si overflow job $o(S^h)$ está calendarizado en su tiempo de liberación en el calendario S^h, entonces parar, calendario S^h es óptimo (Teorema 5).

Caso 4) Si block crítico del calendario S^h no contiene trabajos emergentes, entonces el nodo h es cerrado, es decir, declarar este como una hoja del árbol.

Caso 5) Si el tiempo de inicio del $K(S^h)$ es $r(K(S^h))$ y $K(S^h) = K(\sigma)$, entonces parar (S^h es óptimo).

Caso 6) Si el bloque crítico del calendario S^h contiene trabajos emergentes, entones todos los posibles subconjuntos de los trabajos liberados dentro del bin $B(K(S^h), S^h)$ son considerados y el número correspondiente de brazos alternativos de nodo h son generados; con cada nuevo nodo creado $h^{'}$, un JE-calendario correspondiente $S^{h^{'}} = S^h(J, K(S^h))$ es asociado, donde J es el subconjunto elegido de trabajos.

Caso 7) Al completar cualquiera de los pasos anteriores, sea h el nodo abierto más cercano a la izquierda y repita desde el paso (a00); si no hay tal nodo, entonces pare.

Una vez que hemos descrito nuestro algoritmo enumerativo podemos construir un algoritmo heurístico Beam Search, donde este básicamente trunca el árbol de soluciones T, permitiendo la exploración en T de sólo los k nodos más prometedores en cada nivel l de T.

En el futuro estamos contemplando generalizar nuestros resultados para las versiones generalizadas del problema incluyendo las versiones con multiprocesadores, con tiempos de mantenimiento y tiempos de procesamiento inexactos, así como versiones de dos objetivos.

Bibliografía

[1] A. Condotta, S. Knust and N.V. Shakhlevich. Parallel batch scheduling of equal-length jobs with release and due dates. *Journal of Scheduling*, 13, 463-477 (2010).

[2] A. Gharbi and M. Labidi. Jackson's Semi-Preemptive Scheduling on a Single Machine. Computers & Operations Research 37, 2082-2088 (2010).

[3] B. J. Lageweg, J. K. Lenstra and A. H. G. Rinnooy Kan. Minimizing maximum lateness on one machine: computational experience and some applications. Statistica Neerlandica 30:2541 (1976).

[4] B. Simons. Multiprocessor scheduling of unit-time jobs with arbitrary release times and deadlines. *SIAM J. Comput.* 12, 294-299 (1983).

[5] B. Simons, M. Warmuth. A fast algorithm for multiprocessor scheduling of unit-length jobs. *SIAM J. Comput.* 18, 690-710 (1989).

[6] C. Briand, S. Ourari and B. Bouzouia B. An efficient ILP formulation for the single machine scheduling problem. submitted to RAIRO-Operations Research, (2008).

[7] C. Pessan, J.L. Bouquard, E. Nron. An unrelated parallel machines model for an industrial production resetting problem. European Journal of Industrial Engineering , 2:153-171 (2008).

[8] C .N. Potts. Analysis of a heuristic for one machine sequencing with release dates and delivery times. *Operations Research* 28, p.1436-1441 (1980).

[9] E. Chinos and N. Vakhania. Polynomially Solvable and NP-hard Special Cases for Scheduling with Heads and Tails. RECENT ADVANCES in MATHEMATICS and COMPUTATIONAL SCIENCE. (MCSS 16), p.141-145. ISBN: 978-1-61804-367-2 (http://www.wseas.us/elibrary/conferences/2016/barcelona/MCSS/MCSS-17.pdf) 2016.

[10] E. Chinos and N. Vakhania. Scheduling jobs with two release times and tails on a single machine. INTERNATIONAL JOURNAL OF MATHEMATICAL MODELS AND METHODS IN APPLIED SCIENCES. Volume 10, p.303-3089. ISSN: 2074-1278 http://www.naun.org/main/NAUN/ijmmas/2016/a782001-aan.pdf. 2016.

[11] E. Chinos and N. Vakhania. Adjusting scheduling model with release and due dates in production planning. Cogent Engineering. 4:1,DOI:10.1080/23311916.2017.1321175.2017.

[12] E. Nowicki and C. Smutnicki. An approximation algorithm for single-machine scheduling with release times and delivery times. Discrete Applied Mathematics 48:6979 (1994).

[13] F. Della Croce and V. T'kindt. Improving the preemptive bound for the single machine dynamic maximum lateness problem. *Operations Research Letters* 38 589591 (2010).

[14] G. Lancia. Scheduling jobs with release dates and tails on two unrelated parallel machines to minimize the makespan. European Journal of Operational Research 2(120):277-288 (2000).

[15] G. McMahon and M. Florian. On scheduling with ready times and due dates to minimize maximum lateness. *Operations Research* 23, 475–482 (1975).

[16] G. N. Frederickson. Scheduling unit time tasks with integer release times and deadlines. Information Processing Letters ;16:171173 (1983).

[17] H. Heck and S. Roberts. A note on the extension of a result on scheduling with secondary criteria. Naval Research Logistics Quarterly, 19:59-66 (1972).

[18] H. Kise, T. Ibaraki and H. Mine. Performance analysis of six approximation algorithms for the one-machine maximum lateness scheduling problem with ready times. Journal of Operations Research Society of Japan 22:205-223 (1979).

[19] I. Kacem and H. Kellerer. Approximation algorithms for no idle time scheduling on a single machine with release times and delivery times. Discrete Applied Mathematics 164: 154-160 (2014).

[20] I. Kacem and H. Kellerer. Approximation schemes for minimizing the maximum lateness on a single machine with release times under non-availability or deadline constraints (2018).

[21] J. A. Hoogeveen. Minimizing maximum promptness and maximum lateness on a single machine. *Math. Oper. Res* 21, 100-114 (1995).

[22] J. A. Hoogeveen and S. L. Vande Velde. Minimizing total completion time and maximum cost simultaneously is solvable in polynomial time. Operations Research Letters, 17:205-208 (1995).

[23] J. Carlier. The one–machine sequencing problem. *European J. of Operations Research*. 11, 42–47 (1982).

[24] J. Carlier and E. Pinson. An Algorithm for Solving Job Shop Problem. *Management Science*, 35, 164-176 (1989).

[25] J. Carlier and E. Pinson. Jakson's pseudo preemptive schedule for the $Pm/r_i, q_i/C_{max}$ problem. *Annals of Operations Research* 83, 41-58 (1998).

[26] J. Du and J. Y. T. Leung. Minimizing total tardiness on one machine is NP-hard. Mathematics of Operations Research, 15(3): 483-495 (1990).

[27] J. Grabowski, E. Nowicki, S. Zdrzaka. A block approach for single-machine scheduling with release dates and due dates. European Journal of Operational Research, 2: 278-285 (1986).

[28] J. G. Shantikumar. Scheduling n jobs on one machine to minimize the maximum tardiness with minimum number. Computers and Operations Research, 10(3):255-266 (1983).

[29] J.N.D. Gupta, A.M.A. Hariri and C.N. Potts. Singlemachine scheduling to minimize maximum tardiness with minimum number of tardy jobs. Annals of Operations Research, 92:107-123 (1999).

[30] J.R. Jackson. Schedulig a production line to minimize the maximum tardiness. *Manegement Scince Research Project*, University of California, Los Angeles, CA (1955).

[31] K. R. Baker, Z. Su. Sequencing with due-dates and early start times to minimize maximum tardiness. Naval Research Logistics Quarterly, 21:171176 (1974).

[32] K. Sourirajan and R. Uzsoy. Hybrid decomposition heuristics for solving large-scale scheduling problems in semiconductor wafer fabrication, Journal Of Scheduling 10:41-65 (2007).

[33] L.A. Hall and D.B. Shmoys. Jackson's rule for single-machine scheduling: Making a good heuristic better, Mathematics of Operations Research 17 22–35 (1992).

[34] L. Schrage. Obtaining optimal solutions to resource constrained network scheduling problems, unpublished manuscript (march, 1971).

[35] L. Van Wassenhove and L. F. Gelders. Solving a bicriterion scheduling problem. European Journal of Operational Research, 4:42-48 (1980).

[36] L. Wan, J. Yuan. Single-machine scheduling to minimize the total earliness and tardiness is strongly NP-hard. Operations Research Letters 41: 363365 (2013).

[37] M. I. Dessouky and R. E Larson. Symmetry and optimally properties of the single machine problem. AIIE Transactions, 10(2):170-175 (1978).

[38] M. I. Dessouky and C. R. Margenthaler. The one-machine sequencing problem with early starts and due dates. AIIE Transactions;4:214-222 (1972).

[39] M. Mastrolilli. Efficient approximation schemes for scheduling problems with release dates and delivery times. Journal of Scheduling 6:521-531 (2003).

[40] M. Perregaard and J.Clausen. Parallel branch-and-bound methods for the job-shop scheduling problem. *Annals of Operations Research* 83, 137-160 (1998).

[41] M.R. Garey and D.S. Johnson. Computers and Intractability: A Guide to the Theory of NP–completeness. Freeman, San Francisco (1979).

[42] M.R. Garey, D.S. Johnson, B.B. Simons and R.E. Tarjan. Scheduling unit–time tasks with arbitrary release times and deadlines. *SIAM J. Comput.* 10, 256–269 (1981).

[43] N. Potts, L.N Van Wassenhove. A decomposition algorithm for the single machine total tardiness problem. Operations Research Letters, 5: 177-181 (1982).

[44] N. Vakhania. A better algorithm for sequencing with release and delivery times on identical processors. *Journal of Algorithms* 48, p.273-293 (2003)

[45] N. Vakhania. Single-Machine Scheduling with Release Times and Tails. *Annals of Operations Research*, 129, p.253-271 (2004)

[46] N. Vakhania. "Scheduling jobs with release times preemptively on a single machine to minimize the number of late jobs". *Operations Research Letters* 37, 405-410 (2009)

[47] N.Vakhania. Branch less, cut more and minimize the number of late equal-length jobs on identical machines. *Theoretical Computer Science* 465, 49–60 (2012).

[48] N.Vakhania. A study of single-machine scheduling problem to maximize throughput. *Journal of Scheduling* Volume 16, Issue 4, pp 395-403 (2013).

[49] N. Vakhania and E.Shchepin. Concurrent operations can be parallelized in scheduling multiprocessor job shop, *J. Scheduling* 5, 227-245 (2002).

[50] N. Vakhania and F. Werner. Minimizing maximum lateness of jobs with naturally bounded job data on a single machine in polynomial time. *Theoretical Computer Science* 501, p. 7281 (2013).

[51] N. Vakhania, D. Perez and L.Carballo.Theoretical Expectation versus Practical Performance of Jackson's Heuristic. *Mathematical Problems in Engineering* Volume 2015, Article ID 484671, 10 pages http://dx.doi.org/10.1155/2015/484671 (2015).

[52] N. Vakhania. Dynamic restructuring algorithm for minimizing maximum lateness. Unpublished. https://doi.org/10.13140/rg.2.2.21584.71684 (2016).

[53] N. Vakhania, E. Chinos and C. Zavala. An efficient Heuristic for a discrete optimization problem. Journal of Computer Science Technology Updates. Volume 2 Issue 2, Pages 38-45. DOI: http://dx.doi.org/10.15379/2410-2938.2015.02.02.05. ISSN (online): 2410-2938. http://www.cosmosscholars.com/current-issue-jcstu/79-abstracts/jcstu/483-abstract-an-efficient-heuristic-for-a-discrete-optimization-problem. 2015.

[54] N. Vakhania and F. Alonso. Partial enumeration in polynomial time for scheduling one machine.

[55] N. Vakhania. On polynomial-time approximation for scheduling single machine. DOI: 10.13140/RG.2.2.29007.59040.

[56] P. Baptiste, L. Peridy and E. Pinson. A Branch and Bound to Minimze the Number of Late Jobs on a Single Machine with Release Time Constraints, (2000).

[57] P. Bratley, M. Florian and P. Robillard. On sequencing with earliest start times and due–dates with application to computing bounds for $(n/m/G/F_{max})$ problem. *Naval Res. Logist. Quart.* 20, 57–67 (1973)

[58] R.L. Graham. E.L. Lawler, J.L. Lenstra, and A.H.G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a servey. *Ann. Discrete Math.* 5 287-326 (1979).

[59] R. Sadykov and A. Lazarev. Experimental comparison of branch-and-bound algorithms for the $1|r_j|L_{max}$ problem. Proceedings of the Seventh International Workshop MAPSP05. Italy, Sienna p. 239-241 (2005).

[60] W. Brinkkotter and P.Brucker. Solving open benchmark instances for the job-shop problem by parallel head–tail adjustments. *J. of Scheduling* 4, 53–64 (2001).

[61] W. A. Horn. Some simple scheduling algorithms. Naval Research Logistics Quarterly, 21: 177-185 (1974).

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

[63] Y. Pan and L. Shi. Branch-and-bound algorithms for solving hard instances of the one-machine sequencing problem. European Journal of Operational Research 168(3):1030-1039 (2006).

[64] Z. L. Chen, W. Lu y G. Tang. Single machine scheduling with discretely controllable procesing times. Operations Research Letters, 21:69-76.

[65] Z. Tian, C. T. Ng. y T. C. E. Cheng. An $O(n^2)$ algorithm for scheduling equal-length preemptive jobs on a single machine to minimize total tardiness. Journal of Scheduling, 9(4): 343-364 (2006).