Facultad
de
Ciencias

Un algoritmo de ramificación y poda para el problema de "job shop scheduling"

(A branch and bound algorithm for the job shop scheduling problem)

Trabajo de Fin de Grado
para acceder al

GRADO EN MATEMÁTICAS

Autor: Revuelta San Emeterio, Fernando

Director: González Rodríguez, Inés

Junio – 2022

Resumen

A lo largo de este documento veremos en detalle un algoritmo de ramificación y poda propuesto por Brucker en 1994 [2] para encontrar la solución óptima de una instancia del *Job-Shop Scheduling Problem*.
Palabras clave: Job-shop scheduling problem, optimización combinatoria, algoritmo de ramificación y poda, grafo disyuntivo

Abstract

Throughout this article we will study in detail a branch and bound algorithm proposed by Brucker in 1994 [2] in order to find the optimal solution of an instance of the*Job-Shop Scheduling Problem*.
Key words: Job-shop scheduling problem, combinatorial optimization, branch and bound method, disjunctive graph

Índice general

Índice de tablas

Índice de figuras

Índice de algoritmos

Capítulo 1

Introducción

En los sistemas de manufactura contamos con unas operaciones que han de ser procesadas por unas máquinas en un periodo de tiempo. Normalmente, el número de máquinas es limitado y cada máquina solo puede procesar una operación a la vez. En general, las operaciones no pueden ser procesadas en un orden cualquiera, estas poseen ciertas restricciones de orden, los trabajos (conjunto de operaciones) por ejemplo, se rigen por unas restricciones tecnológicas.

A parte de estas restricciones, existen otras, por ejemplo, fechas de lanzamiento o finalización (suelen ser las más comunes) que definen otros tipos de problemas.

El objetivo final es establecer una fecha de inicio para cada operación sin violar las restricciones establecidas, dando lugar así a una planificación. En general, se trata de obtener la planificación que optimice distintos aspectos como, por ejemplo, el tiempo de compleción de todas las operaciones.

Centrándonos en el *job-shop scheduling problem* (también conocido como *JSP*), este puede ser formulado como un conjunto de trabajos (cada uno de ellos consta de varias operaciones que han de ser procesadas en un determinado orden) y de máquinas (una de cada tipo) de forma que cada trabajo pasa como máximo una sola vez por máquina, cada operación es procesada en una sola máquina en un tiempo determinado y cada máquina solo puede procesar una operación a la vez. Más adelante será formulado formalmente. El objetivo del *job-shop scheduling problem* es encontrar un orden en cada máquina, de forma que no se viole ninguna restricción y el tiempo de compleción de todas las operaciones sea mínimo.

El *job-shop scheduling problem* es un problema NP-duro [8], debido a esto, es muy difícil encontrar soluciones óptimas. A medida que crece el problema, lo hace también su espacio de soluciones rápidamente.

A lo largo de la historia, con el objetivo de encontrar soluciones óptimas para este problema, se han desarrollado varios algoritmos de ramificación y poda. Durante varios años, el algoritmo desarrollado por McMahon y Florian [9] fue el más eficiente, sin embargo, el primer algoritmo que resolvió el *benchmark porblem* de 10 trabajos y 10 máquinas planteado por Muth y Thompson en 1963 [10], fue el propuesto por Carlier y Pinson en 1987 [3], esto nos da una idea de la dificultad de su resolución.

Además de los algoritmos de ramificación y poda, también se han desarrollado heurísticos, siendo los más famosos aquellos basados en reglas de prioridad. Encontrar heurísticos que aporten buenas soluciones, es beneficioso también en los algoritmos de ramificación y poda, sin embargo, hasta el momento no hay ninguno que garantice una buena solución en cualquier situación.

1.1. Motivación

El problema conocido como *Job Shop Scheduling Problem* (*JSP*) consiste en planificar temporalmente la ejecución de un conjunto de tareas en un conjunto de recursos o máquinas sujeto a una serie de restricciones: restricciones de precedencia entre tareas (las tareas están organizadas en trabajos y dentro de un trabajo han de ejecutarse en un orden determinado), y restricciones de recurso (cada tarea tiene asignada una máquina concreta y cada máquina ha de ejecutar las tareas que le pertenecen de manera secuencial y sin interrupciones). Además de conseguir una planificación factible, en el sentido de que se cumplan todas las retricciones, se busca obtener una solución óptima según alguna medida de rendimiento, siendo lo más habitual minimizar el tiempo de fin de la última tarea en ejecutarse, también conocido como makespan.

El interés del *JSP* radica no solo en sus aplicaciones en diversos ámbitos industriales y empresariales sino también en su complejidad, puesto que se trata de un problema de optimización combinatoria NP-duro. Desde el ámbito de la Inteligencia Artificial y la Investigación Operativa se han propuesto y siguen proponiendo múltiples algoritmos de búsqueda exacta y aproximada para abordar este problema. De entre los exactos, uno de los más exitosos es el algoritmo de ramificación y poda de Brucker et al. [2].

1.2. Objetivos del TFG

El objetivo final de este TFG es estudiar e implementar el algoritmo propuesto en [2]. Para ello estudiaremos las distintas partes del algoritmo, en el que se combinan conceptos como, una generalización del esquema de ramificación propuesto por Grabowsky [6] y un método para fijar disyunciones propuesto por Carlier y Pinson [4]. Finalmente, implementaremos el algoritmo utilizando el lenguaje de programación *Python*.

1.3. Organización de la memoria

El resto de la memoria está organizado de la siguiente manera.

En primer lugar, en el capítulo 2 se expondrán y desarrollaran una serie de conceptos útiles para la comprensión del algoritmo.

Después, en el capítulo 3, entraremos en detalle en el esquema de ramificación y la estrategia de ramificación y poda seguida.

Más adelante, en el capítulo 4, se introducirán los conceptos de cabeza y cola de una operación en un nodo, estos son de gran importancia para el algoritmo.

Posteriormente, en el capítulo 5, se desarrollará un método sofisticado que permitirá fijar disyunciones adicionales.

El capítulo 6 le dedicaremos al cálculo de algunos elementos necesarios para poder aplicar el proceso de ramificación y poda como son las cotas inferiores o las soluciones heurísticas.

Finalmente, a lo largo del capítulo 7, mostraremos la implementación del algoritmo realizada en *Python*, así como los resultados computacionales obtenidos para diversas instancias utilizando dicha implementación.

Capítulo 2

Preliminares

A lo largo de este capítulo, expondremos una serie de conceptos, los cuales serán de utilidad para comprender el algoritmo.

2.1. Job Shop Problem

El *job-shop scheduling problem* puede ser descrito de la manera siguiente. Consideremos una planta de producción, en la cual han de realizarse un conjunto de trabajos $J_1, ..., J_n$. Para ello, contamos con una serie de máquinas $M_1, ..., M_m$. Cada trabajo J_i consiste de un cierto número de operaciones $O_{i,1}, ..., O_{i,n_i}$, que han de procesarse en ese orden sin solapamiento. Dada una operación $O_{i,k}$, esta solo puede procesarse en una única máquina $\mu_{i,k} = M_j$, esto conlleva un tiempo $p_{i,k}$. Una vez empezada una operación, esta no se puede detener. Cada trabajo puede pasar como máximo una vez por cada máquina. En cuanto a las máquinas, contamos solo con una de cada tipo y solo pueden procesar una operación a la vez.

Dadas estas restricciones, el objetivo es encontrar un orden para todas las operaciones $O_{i,k}$ con $\mu_{i,k} = M_j$ para cada máquina M_j, de forma que se respeten las restricciones de precedencia establecidas en los trabajos y el tiempo de compleción de todos los trabajos, C_{max}, sea mínimo. Visto de otro modo, nuestro objetivo es encontrar una planificación factible, de forma que el tiempo de compleción de todos los trabajos es mínimo. Una planificación factible se obtiene permutando el orden de las operaciones de las máquinas sin violar las restricciones de precedencia establecidas en los trabajos, por tanto, nos enfrentamos a un problema de optimización combinatoria.

2.2. Modelo de grafo disyuntivo

Podemos definir un grafo disyuntivo $G = (V, C \cup D)$, como un conjunto de nodos o vértices (V) y un conjunto de aristas sobre estos que pueden ser dirigidas (C), o no dirigidas (D). Estas últimas también reciben el nombre de disyunciones.

El modelo de grafo disyuntivo fue introducido por Roy y Sussman [11] en 1964.

Veamos como se puede representar el *job-shop scheduling problem* siguiendo el modelo de grafo disyuntivo.

Sea V el conjunto de nodos que representa todas las operaciones. Cada nodo posee un peso igual al tiempo de procesamiento de la operación que representa. Además

de las operaciones, V contendrá también dos nodos extra 0 y $*$, con tiempos de procesamiento nulos, estos representan el inicio y fin de la producción respectivamente.

Con el objetivo de expresar en el grafo las restricciones de precedencia establecidas por los trabajos y máquinas, se crean los conjuntos C y D.

C es el conjunto de aristas dirigidas que reflejan el orden de las operaciones en un trabajo (expresan relaciones de precedencia).

Por otro lado, para cada par de operaciones que han de ser procesadas en la misma máquina, se establece una arista no dirigida. Al conjunto de estas aristas le llamaremos D.

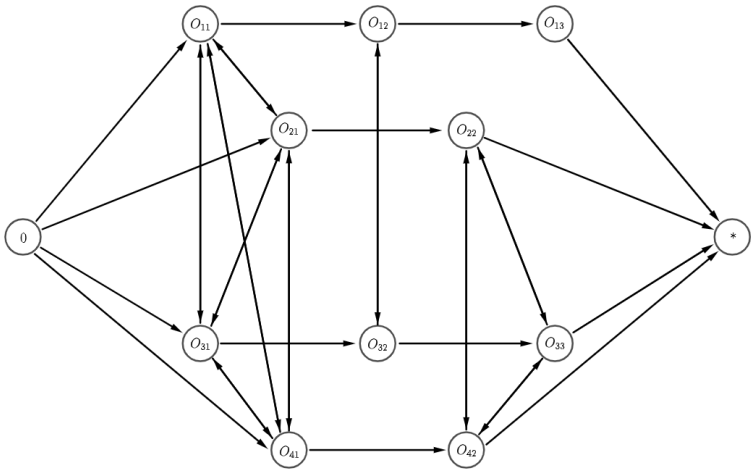

Figura 2.1: Ejemplo de un grafo disyuntivo para el *JSP*

La figura 2.1 muestra el grafo disyuntivo que representa una instancia del *job-shop scheduling problem* con 4 trabajos y 4 máquinas.

El objetivo final, como ya se ha comentado anteriormente, es establecer un orden de procesamiento de las operaciones en cada máquina sin violar ninguna restricción. En el modelo de grafo disyuntivo, esto se obtiene trasformando las aristas no dirigidas en dirigidas, a este proceso le denominaremos fijar aristas. Adicionalmente, al conjunto de aristas fijadas las llamaremos selección. Notemos que una selección S, define una planificación factible si:

i) Todas las aristas no dirigidas han sido fijadas

ii) El grafo resultante, $G(S) = (V, C \cup S)$ es acíclico.

En ese caso, diremos que la selección es completa.

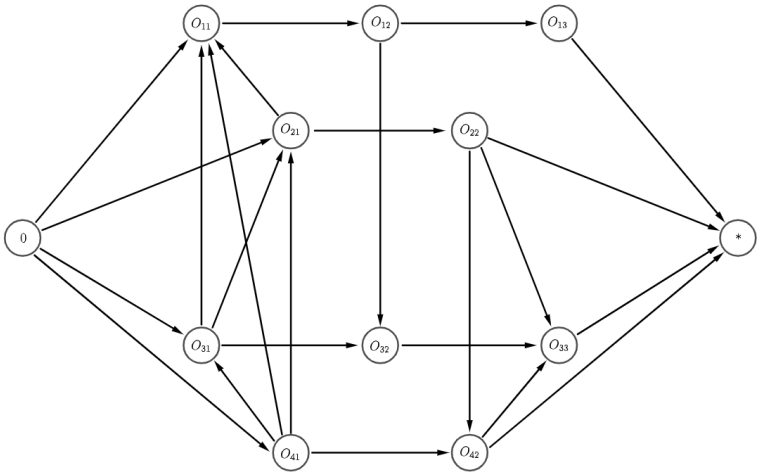

Figura 2.2: Ejemplo de una selección completa de un grafo disyuntivo para el *JSP*

La figura 2.2 muestra una selección completa para la instancia del *job-shop scheduling problem* definida en la figura 2.1.

Dada una selección completa S, el tiempo máximo de compleción de todas las operaciones, C_{max}, se corresponde con el camino de mayor peso ente 0 y $*$ en $G(S) = (V, C \cup S)$. Este camino es conocido como el camino crítico. Puede haber más de uno.

Por último, dada una selección completa, podemos considerar el concepto de bloque. Una secuencia de operaciones sucesivas en un camino crítico, $u_1, ..., u_k$ es denominada bloque, si cumple las dos propiedades siguientes:

1. La secuencia contiene al menos dos operaciones.

2. La secuencia representa el máximo número de operaciones que tienen que ser procesadas en una misma máquina, es decir, las operaciones inmediatamente anterior y posterior a la secuencia, han de procesarse en una máquina distinta a la utilizada por las operaciones de la secuencia.

Gracias a estos conceptos, denotamos como B_j el bloque j-ésimo del camino crítico.

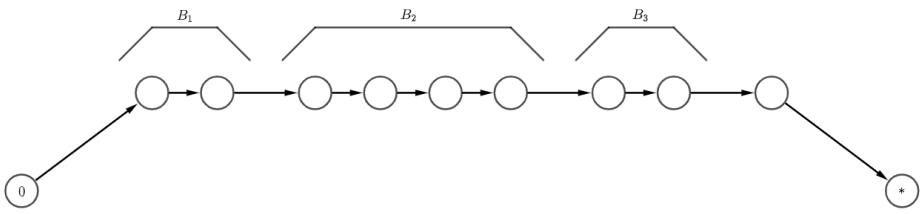

Figura 2.3: Ejemplo de bloques para el camino crítico de un grafo disyuntivo

La figura 2.3 muestra los bloques asociados a un camino crítico, que cuenta con 9 operaciones, para una instancia del *job-shop scheduling problem.*

2.3. Orden topológico de un grafo

Dado un grafo acíclico y dirigido, un ordenamiento topológico produce un ordenamiento lineal de los vértices o nodos, de forma que, si el grafo posee la arista (n, m), entonces n aparece ordenado antes que m. Este ordenamiento es útil para expresar relaciones de precedencia.

Es importante el hecho de que el grafo sea acíclico, ya que si no es el caso, no es posible establecer este orden. Si existiera un ciclo $a \to b \to c \to ... \to d \to e \to a$, el grafo contaría con las aristas $(a, b), (b, c), .., (d, e), (e, a)$, es decir, a va antes que b, b va antes que c, ..., d va antes que e y e va antes que a. Con lo cual, se obtiene que a va antes que a, haciendo imposible dicho ordenamiento lineal.

Para construirlo, basta aplicar el siguiente proceso de forma iterativa. En primer lugar, se escoge un vértice sin aristas incidentes en el (sabemos de su existencia debido a que no hay ciclos). Este se añade al ordenamiento lineal y se elimina del grafo. Se aplica el mismo proceso al grafo resultante (de nuevo, sabemos que podemos elegir un vértice con esas características, debido a que el grafo original es acíclico).

2.4. Algoritmo de ramificación y poda

La ramificación y poda se puede ver como una variante de la búsqueda por *backtracking* mejorada. Esta, suele ser utilizada para resolver problemas de optimización combinatoria.

Entrando en detalle, la estrategia de ramificación y poda consiste en una búsqueda con *backtracking* en un árbol de búsqueda donde, cada nodo, representa un conjunto de soluciones. La poda se realiza cuando se detectan nodos donde las soluciones que ofrece no son óptimas, por ello, se poda la rama correspondiente y se deja de explorar, evitando de esta forma malgastar recursos.

El algoritmo de ramificación y poda requiere, esencialmente, de dos herramientas. La primera de las herramientas se trata de un proceso de expansión en el que, dado un conjunto finito de soluciones S (asociadas a un nodo), se obtienen dos o más subconjuntos de S, $\{S_1, ..., S_k\}$ de forma que, su unión recubre S y son disjuntos dos a dos. Notemos que si S contenía una solución óptima, ahora esta se encontrará en alguno de los subconjuntos. A este proceso se le conoce como ramificación y, debido a su aplicación recursiva sobre los nodos, se logra definir una estructura de árbol.

La otra herramienta es el proceso de poda. Para ello, si se trata de un problema de minimización (como será el nuestro), establecemos para cada nodo una cota inferior del coste de las soluciones que representa. Por otro lado, vamos guardando un registro con la mejor solución encontrada hasta el momento (se inicializa en $+\infty$). Antes de procesar un nodo, si su cota inferior supera o iguala dicho registro, este deja de ser explorado y se poda la rama correspondiente, ya que las soluciones que aporta son peores o iguales que la mejor solución encontrada hasta el momento.

Por último, hay que resaltar la importancia de las cotas, ya que de estas dependerá que se pode o no. Por tanto, nos interesa que las cotas inferiores no sobrestimen el valor real (si no, perdemos soluciones prometedoras) y que se ajusten lo máximo posible a este (nos permitirá realizar más podas y ahorrar recursos).

Capítulo 3

Ramificación y poda para el *JSP*

3.1. Algoritmo de ramificación y poda aplicado al *JSP*

A lo largo de este capítulo, desarrollaremos la estrategia de ramificación y poda, propuesta en [2], que utiliza este algoritmo. Este, puede ser visto como un árbol de búsqueda donde cada nodo se corresponde con un grafo disyuntivo. Inicialmente, solo contaremos con la raíz, donde todavía no ha sido fijada ninguna disyunción, por lo tanto, representa todas las posibles soluciones del problema. Los sucesores de un nodo cualquiera son calculados fijando nuevas disyunciones, de este modo, el grafo disyuntivo correspondiente a cada sucesor representa todas las posibles soluciones que, además de respetar las relaciones de precedencia del padre, respetan las nuevas relaciones de precedencia fijadas en el sucesor en cuestión. El algoritmo trata a los sucesores de cada nodo de forma recursiva empezando por la raíz. El procesado de un nodo termina cuando, o bien se ha alcanzado una solución (selección completa), o bien se puede demostrar que este no contiene una solución óptima.

Entrando más en detalle, sea r un nodo de nuestro árbol de búsqueda. Dicho nodo se corresponde con un grafo disyuntivo $G(FD_r) = G(V, C \cup FD_r)$, donde FD_r denota el conjunto de disyunciones fijadas en el nodo r. Dicho nodo representa todas las posibles soluciones que respetan las relaciones de precedencia fijadas en el mismo, las denotaremos $Y(r)$.

El proceso de ramificación consisten en dividir $Y(r)$ en subconjuntos que lo recubran, disjuntos dos a dos $Y(s_1), ..., Y(s_k)$. De nuevo, $Y(s_i)$ representa el conjunto de posibles soluciones que respetan las relaciones de precedencia fijadas en grafo disyuntivo $G(FD_{s_i}) = G(V, C \cup FD_{s_i})$ donde $FD_r \subset FD_{s_i}$, es decir, $G(F_{s_i})$ se obtiene fijando nuevas disyunciones además de las de FD_r.

De esta forma, para el nodo r, se obtendrán sus sucesores $s_1, ..., s_q$ que serán tratados recursivamente.

Por otro lado, para poder deducir si un nodo r no contiene soluciones óptimas, se le asocia un valor $LB(r)$ que acota inferiormente el valor objetivo (C_{max}) de todas las soluciones en $Y(r)$. Además de estas cotas inferiores, contamos con un valor, UB, que contiene el valor objetivo de la mejor solución encontrada hasta el momento. Este nos servirá para realizar la poda cada vez que $LB(r) \geq UB$, ya que, en ese caso, las soluciones de $Y(r)$ son peores o iguales que la mejor solución encontrada hasta el momento y, por tanto, no debemos malgastar recursos explorando la rama en cuestión.

En el caso de que el grafo disyuntivo asociado a un nodo cuente con ciclos (no

representa una solución factible) se fija $LB(r) = \infty$, de esta forma será podado.

3.2. Esquema de ramificación

Este se basa en una solución o planificación factible del nodo del árbol de búsqueda en cuestión, $G(S) = G(V, C \cup S)$, siendo S una selección completa. Para el cálculo de dicha solución se hace uso de un heurístico, tal y como se explica en la sección 6.2.

En dicha solución, consideramos el camino crítico P, con longitud $L(S)$, así como los bloques del mismo.

Una vez introducidos estos elementos, se puede enunciar el siguiente teorema que es la base de la estrategia de ramificación.

Teorema 3.1. *Consideramos una instancia cualquiera del job-shop scheduling problem. Sea S una selección completa. Si existe otra selección completa S', de forma que $L(S') < L(S)$, al menos una operación de un bloque de $G(S)$ ha de procesarse antes de la primera o después de la última operación del bloque correspondiente en $G(S')$.*

Demostración. Sea $P = (0, u_1^1, u_2^1, ..., u_{m_1}^1, ..., u_1^k, u_2^k, ..., u_{m_k}^k, *)$ el camino crítico en $G(S) = (V, C \cup S)$ donde el superíndice agrupa operaciones consecutivas que han de procesarse en una misma máquina y el subíndice indica la posición de la operación dentro del agrupamiento. En el caso de que se tratarse de un bloque, sabemos que $m_i > 1$. Supongamos que existe S' selección completa de forma que $L(S') < L(S)$ y ninguna operación de ningún bloque de $G(S)$ es procesada antes de la primera o después de la última del bloque correspondiente en $G(S')$. Debido a esto, sabemos que $G(S') = G(V, C \cup S')$ contiene las siguientes aristas:

- $u_1^j \to u_i^j,\ \forall j \in \{1, ..., k\}$ *tal que* $m_j > 1 \wedge \forall i \in \{2, ..., m_j\}$
- $u_i^j \to u_{m_j}^j,\ \forall j \in \{1, ..., k\}$ *tal que* $m_j > 1 \wedge \forall i \in \{1, ..., m_j - 1\}$

Por otro lado, sabemos que las operaciones que se encuentran entre la primera y la última de cada bloque, en una selección completa se encontrarán totalmente ordenadas, pues son operaciones procesadas en una misma máquina.

Además, dadas dos operaciones contiguas en P que tengan distinta máquina, la arista que los une ha de corresponder a una restricción de trabajo, luego $G(S')$ cuenta con estas aristas también.

Debido a estos hechos, $G(S')$ contiene el siguiente camino:

$$(0, u_1^1, v_2^1, ..., v_{m_1-1}^1, u_{m_1}^1, ..., u_1^k, v_2^k, ..., v_{m_k-1}^k, u_{m_k}^k, *)$$

donde $v_2^j, ..., v_{m_j-1}^j$ es una permutación de $u_2^j, ..., u_{m_j-1}^j\ \forall j \in \{1, ..., k\}$.

Dado que $G(S')$ contiene este camino, su longitud proporciona una cota inferior de $L(S')$ (tiene ese camino, pero puede tener otro más largo). Tomando $v_1^j = u_1^j$ y $v_{m_j}^j = u_{m_j}^j\ \forall j \in \{1, ..., k\}$ por comodidad con la notación, tenemos entonces:

$$L(S') \geq \sum_{j=1}^{k}(\sum_{i=1}^{m_j} p_{v_i^j}) = \sum_{j=1}^{k}(\sum_{i=1}^{m_j} p_{u_i^j}) = L(S)$$

Lo cual es una contradicción, ya que habíamos supuesto $L(S') < L(S)$.

La igualdad central, se puede tomar debido a que se trata de una permutación y la suma es conmutativa.

□

Debido al teorema anterior, se obtiene la siguiente consecuencia:
Dada una instancia cualquiera del *job-shop scheduling problem* y dadas dos selecciones completas, S y S' de forma que $L(S') < L(S)$, entonces se da alguna de las dos siguientes condiciones:

i) Al menos una operación de un bloque B en $G(S)$ (diferente de la primera operación de B) ha de ser procesada antes que el resto de operaciones de B en $G(S')$.

ii) Al menos una operación de un bloque B en $G(S)$ (diferente de la última operación de B) ha de ser procesada después que el resto de operaciones de B en $G(S')$

Si no se da ninguna de las dos, entramos en contradicción con el teorema 3.1. Volviendo al árbol de búsqueda, sea r un nodo e $y \in Y(r)$ una solución calculada utilizando un heurístico. Consideremos la selección completa S asociada a y, el camino crítico de $G(S)$ y los bloques $B_1, ..., B_k$ definidos en este. Para cada bloque $B_j = u_1^j, ..., u_{m_j}^j$ consideramos los siguientes subconjuntos:

i) $E_j^B = B_j \setminus \{u_1^j\}$

ii) $E_j^A = B_j \setminus \{u_{m_j}^j\}$

El primero de ellos lo llamaremos *before-candidates* y el segundo de ellos *after-candidates*.

Para cada *before-candidate* obtendremos un nuevo sucesor s del nodo r moviendo dicha operación al principio de su bloque. Del mismo modo, para cada *after-candidate*, obtendremos un nuevo sucesor moviendo la operación en cuestión al final de su bloque. El hecho de mover una operación al principio o al final de su bloque establece implícitamente unas relaciones de precedencia, ya que sabemos que dentro de la máquina, la operación a mover se procesará, o bien antes, o bien después, del resto de operaciones del bloque. Para mover una operación $l \in E_j^B$ al principio del bloque B_j, basta fijar las disyunciones $\{l \to i : i \in B_j \setminus \{l\}\}$. Del mismo modo, para mover una operación $l \in E_j^A$ al final del bloque B_j, basta fijar las disyunciones $\{i \to l : i \in B_j \setminus \{l\}\}$. Sin embargo, estas no son las únicas aristas que podemos fijar. Veamos que se pueden fijar más aristas gracias a las siguientes ideas.

En primer lugar, sea s un sucesor inmediato del nodo r, generado al mover la operación $l \in E_j^B$. Como los nodos se exploran de manera recursiva, tras hacer *backtracking* a r desde s, se han examinado todas los soluciones en $Y(r)$ con las disyunciones $\{l \to i : i \in B_j \setminus \{l\}\}$ fijadas (además de las de r). Debido a esto, sabemos que, en las soluciones que surgen a partir del resto de sucesores inmediatos de r que quedan por explorar, l no se procesa antes que el resto de operaciones de B_j, dicha solución habría sido encontrada en s o en sus sucesores.

Durante la exploración de s y sus sucesores, puede que se haya encontrado un nuevo UB que mejore el anterior.

Por otro lado, consideremos una permutación arbitraria $E_1, ..., E_{2k}$ de todos los conjuntos E_j^B y E_j^A. Diremos que E_t es predecesor de $E_{t'}$ si $t < t'$. Esta permutación define un orden de ramificación, por lo tanto, si generamos un sucesor s de r moviendo la operación $l \in E_t$, los nodos correspondientes a las operaciones $l' \in E_1 \cup ... \cup E_{t-1}$ ya han sido procesados.

Teniendo esto en cuenta, en el nodo s generado al mover la operación $l \in E_t$ podemos fijar además las siguientes aristas:

i) $F_j = \{u_1^j \to i,\ \forall i \in \{u_2^j, ..., u_{m_j}^j\}\}$ para cada predecesor E_j^B de E_t

ii) $L_j = \{i \to u_{m_j}^j,\ \forall i \in \{u_1^j, ..., u_{m_j-1}^j\}\}$ para cada predecesor E_j^A de E_t

Esto se puede hacer ya que, una vez procesado un conjunto E_j^B, han sido inspeccionadas todas las soluciones en las que otra operación del bloque es la primera, por ello, a la hora de inspeccionar el siguiente conjunto, podemos establecer u_1^j como la primera operación, ya que solo cabe esa posibilidad. Del mismo modo ocurre para los E_j^A, que nos permiten fijar $u_{m_j}^j$ como la última operación del bloque.

Notemos por tanto que, a la hora de ramificar un nodo r, consideramos todas las soluciones en $Y(r)$ que puede que mejoren la heurística y además, los conjuntos de soluciones que ofrecen los sucesores inmediatos de r, son disjuntos dos a dos. La primera afirmación es debida al Teorema 3.1 ya que, si hay alguna solución en $Y(r)$ que mejora la solución heurística, se encuentra en estos subconjuntos. Para ver la segunda afirmación, basta tener en cuenta las aristas que se fijan en cada sucesor inmediato y notaremos que al menos diferirán en la primera o última operación de un bloque.

Hasta el momento, no se ha especificado la forma de escoger la permutación de los conjuntos E_j^B y E_j^A. Uno de los objetivos es fijar el mayor número de disyunciones lo antes posible. Esto tiene una serie de ventajas como por ejemplo:

i) Aumenta el valor de las cotas inferiores, ya que hay mas relaciones de precedencia que respetar.

ii) Aumenta el número de sucesores con ciclos, por el mismo motivo que en el punto anterior.

iii) El heurístico busca soluciones más rápido (esto se verá en el algoritmo utilizado para calcular la solución heurística expuesto en la sección 6.2).

Por ello, escogeremos dichos conjuntos en orden no creciente en cuanto a la cardinalidad de los bloques se refiere, tomando los E_j^A como sucesores directos de los E_j^B. De esta forma, conseguimos fijar el mayor número de disyunciones lo antes posible ya que, observando el proceso de ramificación, notamos que los sucesores de un nodo cuentan con las disyunciones fijadas en su padre, así como las disyunciones fijadas al mover una operación al inicio o al final de un bloque. El número de estas últimas dependerá de la longitud del bloque. Por otro lado, en los nodos hermanos fijamos las disyunciones correspondientes a los conjuntos F_j y L_j que dependen de la longitud de los bloques de los predecesores.

El orden sería por tanto el siguiente:

$$E_{2i-1} = E_{\pi(i)}^B, \quad E_{2i} = E_{\pi(i)}^A \quad \forall i \in \{1, ..., k\}$$

donde π es una permutación del conjunto $\{1, ..., k\}$ de forma que $|B_{\pi(i)}| \geq |B_{\pi(j)}|$ si $i < j$.

Además, teniendo en cuenta las características del esquema de ramificación, bajo ciertas circunstancias, es posible eliminar los sucesores correspondientes a los *before-candidates* en el primer bloque, y los correspondientes a los *after-candidates* en el último bloque. Esto se debe al siguiente resultado, basado en las ideas de [6] para el problema de una máquina.

Proposición 3.2. *Sea r un nodo del árbol de búsqueda e $y \in Y(r)$ la solución heurística calculada para dicho nodo. Sea $P = (0, u_1^1, u_2^1, ..., u_{m_1}^1, ..., u_1^k, u_2^k, ..., u_{m_k}^k, *)$ un camino crítico en y, donde el superíndice agrupa operaciones consecutivas que han de procesarse en una misma máquina y el subíndice indica la posición de la operación dentro del agrupamiento. Sean $s_1, ..., s_m$ todos los sucesores de r, exceptuando los correspondientes a los before-candidates en el primer bloque si $m_1 > 1$, y los correspondientes a los after-candidates en el último bloque si $m_k > 1$. Tras haber procesado los nodos $s_1, ..., s_m$ y haber hecho backtracking a r, las soluciones pertenecientes al conjunto $Y(r) \setminus \cup_{i=1}^m Y(s_i)$ no mejoran a y.*

Demostración. Supongamos que han sido procesados los sucesores $s_1, ..., s_m$, debido al esquema de ramificación, es posible fijar ciertas aristas en el resto de sucesores, por lo tanto, el resto de soluciones contarán con ellas. Consideramos $y' \in Y(r) \setminus \cup_{i=1}^m Y(s_i)$, veamos que podemos encontrar en y' un camino al menos tan pesado como P, de este modo, es fácil deducir que $C_{max}(y') \geq C_{max}(y)$.

Sea j_1 el índice del primer bloque y j_k el índice del último bloque. Debido a lo comentado previamente e independientemente del los valores m_1 y m_k, en y' encontramos las siguientes aristas:

- $u_1^j \rightarrow u_i^j,\ \forall j \in \{1, ..., k\} \setminus \{j_1\}$ *tal que* $m_j > 1 \wedge \forall i \in \{2, ..., m_j\}$
- $u_i^j \rightarrow u_{m_j}^j,\ \forall j \in \{1, ..., k\} \setminus \{j_k\}$ *tal que* $m_j > 1 \wedge \forall i \in \{1, ..., m_j - 1\}$

Además, dadas dos operaciones contiguas en P que tengan distinta máquina, la arista que las une ha de corresponder a una restricción de trabajo, luego contamos con estas aristas también.

Por otro lado, sabemos que en una selección completa las operaciones que se encuentran entre la primera y la última de cada bloque distinto del primero y el último se encontraran totalmente ordenadas, pues son operaciones realizadas en una misma máquina.

De todo ello, se deduce que y' cuenta con el siguiente camino:

$$u_{m_{j_1}}^{j_1}, ..., u_1^j, v_2^j, ..., v_{m_j-1}^j, u_{m_j}^j, ..., u_1^{j_k}$$

donde $v_2^j, ..., v_{m_j-1}^j$ es una permutación de $u_2^j, ..., u_{m_j-1}^j\ \forall j \in \{j_1 + 1, ..., j_k - 1\}$.
En cuanto al primer bloque, consideramos dos casos en función del valor de m_1.

i) Si $m_1 \not> 1$, han sido procesados los sucesores correspondientes a los *before-candidates* en el primer bloque y por tanto, contamos con las siguientes aristas $\{u_1^{j_1} \rightarrow u_i^{j_1}\}\ \forall i \in \{2, ..., m_{j_1}\}$. Utilizando un razonamiento similar al anterior, se deduce que y' cuenta con el siguiente camino:

$$0, u_1^1, u_1^2, ..., u_1^{j_1-1}, u_1^{j_1}, v_2^{j_1}, ..., v_{m_{j_1}-1}^{j_1}, u_{m_{j_1}}^{j_1}$$

ii) Si $m_1 > 1$, el primer agrupamiento se corresponde con un bloque y por tanto, $j_1 = 1$. Como ya hemos comentado anteriormente, la operación $u^{j_1}_{m_{j_1}}$ es procesada después del resto de operaciones del mismo bloque. Estas últimas, presentaran un orden determinado en una solución, por los motivos expuestos anteriormente. Como además, cualquier operación es sucesora del 0, en y' encontraremos el camino:

$$0, ..., v^{j_1}_1, ..., v^{j_1}_{m_{j_1}-1}, u^{j_1}_{m_{j_1}}$$

donde $v^{j_1}_1, ..., v^{j_1}_{m_{j_1}-1}$ es una permutación de $u^{j_1}_1, ..., u^{j_1}_{m_{j_1}-1}$.

Para el último bloque, se puede aplicar un razonamiento similar.

Finalmente, en función de los valores que tomen m_1 y m_k, podemos plantear cuatro casos. Debido a todo lo comentado, en cada uno de ellos es posible construir en y' un camino al menos tan pesado como P.

i) Si $m_1, m_k \not> 1$, y' cuenta con el siguiente camino:

$$(0, u^1_1, u^2_1, ..., u^{j_1-1}_1, u^{j_1}_1, v^{j_1}_2, ..., v^{j_1}_{m_{j_1}-1}, u^{j_1}_{m_{j_1}}, ..., u^{j_k}_1, v^{j_k}_2, ..., v^{j_k}_{m_{j_k}-1}, u^{j_k}_{m_{j_k}}, u^{j_k+1}_1, ..., u^{k-1}_1, u^k_1, *)$$

ii) Si $m_1, m_k > 1$, y' cuenta con el siguiente camino:

$$(0, ..., v^{j_1}_1, ..., v^{j_1}_{m_{j_1}-1}, u^{j_1}_{m_{j_1}}, ..., u^{j_k}_1, v^{j_k}_2 ..., v^{j_k}_{m_{j_k}}, ..., *)$$

Notemos que ha de ser, $j_1 = 1$ y $j_k = k$.

iii) Si $m_1 > 1$ y $m_k \not> 1$, y' cuenta con el siguiente camino:

$$(0, ..., v^{j_1}_1, ..., v^{j_1}_{m_{j_1}-1}, u^{j_1}_{m_{j_1}}, ..., u^{j_k}_1, v^{j_k}_2, ..., v^{j_k}_{m_{j_k}-1}, u^{j_k}_{m_{j_k}}, u^{j_k+1}_1, ..., u^{k-1}_1, u^k_1, *)$$

Notemos que ha de ser, $j_1 = 1$.

iv) Si $m_1 \not> 1$ y $m_k > 1$, y' cuenta con el siguiente camino:

$$(0, u^1_1, v^1_2, ..., v^1_{m_1-1}, u^1_{m_1}, ..., u^{j_1}_1, v^{j_1}_2, ..., v^{j_1}_{m_{j_1}-1}, u^{j_1}_{m_{j_1}}, ..., u^{j_k}_1, v^{j_k}_2 ..., v^{j_k}_{m_{j_k}}, ..., *)$$

Notemos que ha de ser, $j_k = k$.

Tomemos $v^j_1 = u^j_1$ y $v^j_{m_j} = u^j_{m_j}$, en aquellos casos donde no se halla cambiado por comodidad con la notación. En cualquier caso, se tiene que:

$$\begin{aligned} C_{max}(y') &\geq \sum_{j=1}^{k} (\sum_{i=1}^{m_j} p_{v^j_i}) \\ &= \sum_{j=1}^{k} (\sum_{i=1}^{m_j} p_{u^j_i}) = C_{max}(y) \end{aligned}$$

Para la primera desigualdad, hemos de tener en cuenta que puede existir otro camino más pesado en y', así como el hecho de que no hemos tenido en cuenta el peso de los caminos entre 0 y $v^{j_1}_1$ cuando $m_1 > 1$, ni entre $v^{j_k}_{m_{j_k}}$ y $*$ cuando $m_k > 1$, las cuales son cantidades no negativas. La primera igualdad se da, ya que se trata de una permutación y la suma es conmutativa. La ultima igualdad, se da por definición.

De esta cadena de desigualdades, se deduce el resultado. □

Por otro lado, el proceso de ramificación es también útil para evitar los ciclos de longitud dos. Esto se puede hacer durante el cálculo de los *before-candidates* y *after-candidates*. Dado un bloque $B_l = u_1^l, ..., u_{m_l}^l$, si la disyunción $i \rightarrow j$ $(i, j \in B_l)$ ha sido fijada, la operación i no será incluida en E_l^A, en caso contrario, se produciría un ciclo y por lo tanto, el sucesor en cuestión no contendría soluciones factibles. Del mismo modo, j no será incluida en E_l^B.

3.3. Algoritmo

Teniendo en cuenta los aspectos comentados a lo largo de este capítulo, estamos en condiciones de formular el algoritmo de ramificación y poda. Su pseudocódigo es el siguiente:

```
1: procedure Ramificación y Poda(r) /*r nodo del árbol de búsqueda*/
2:     Calcular una solución y ∈ Y(r) utilizando un heurístico
3:     si C_max(y) < UB entonces
4:         UB = C_max(y)
5:     Calcular un camino crítico de y, P
6:     Calcular los bloques de P
7:     Calcular los conjuntos E_j^B y E_j^A
8:     mientras ∃i ∈ E_j^v para algún j ∈ {1, ..., k} y v ∈ {A, B} hacer
9:         Eliminar la operación i de E_j^v
10:        Fijar las disyunciones para el correspondiente sucesor s
11:        Calcular una cota inferior para el nodo s, LB(s)
12:        si LB(s) < UB entonces
13:            Ramificación y Poda(s)
```

Algoritmo 3.1: Algoritmo de ramificación y poda para el *JSP*

Esta se trata de la versión más primitiva del algoritmo, a la cual, hemos de añadir distintos aspectos mencionados previamente como, por ejemplo, el fijado de aristas adicionales al finalizar la inspección de todos los sucesores provenientes de un conjunto, establecer el orden de los bloques o verificar si podemos eliminar algún sucesor.

Notemos que la inspección de un nodo finaliza si:

i) La cota inferior supera o iguala el UB, en ese caso, las soluciones que ofrece el nodo son peores o iguales que la mejor solución obtenida hasta el momento.

ii) El camino crítico de la solución heurística no contiene bloques, esto quiere decir que el camino más largo corresponde a un trabajo y por tanto, todas las soluciones de ese nodo tienen ese mismo camino y no es posible mejorar.

iii) Los conjuntos E_j^B y E_j^A están vacíos para todos los bloques B_j, en ese caso, se han acabado de procesar todos los sucesores del nodo en cuestión.

Finalmente, no se ha especificado el orden de inspección de las operaciones dentro de un conjunto E_j. Basándonos en la evidencia de [2], tomaremos los *before-candidates* en orden no decreciente en las cabezas y los *after-candidates*, en orden no decreciente en las colas. Los conceptos cabeza y cola serán introducidos en el siguiente capítulo.

Capítulo 4

Cabezas y colas

4.1. Definiciones

Dado un nodo cualquiera de nuestro árbol de búsqueda, a cada operación i le asociaremos una cabeza y una cola. Estos conceptos son importantes, por ejemplo, para el cálculo de cotas inferiores, así como para el cálculo de la solución heurística. También nos permiten definir el orden de ramificación.

Para el cálculo de las cabezas y colas, tendremos en cuenta las aristas dirigidas (relación de precedencia en los trabajos) y aquellas disyunciones que hayan sido fijadas hasta el momento, por lo tanto, dependen del nodo r en el que nos encontremos.

Definición 4.1. Sea r un nodo del árbol de búsqueda, una cabeza r_i de una operación i es una cota inferior del tiempo de inicio más temprano posible para la operación i en cualquier solución de $Y(r)$.

Definición 4.2. Sea r un nodo del árbol de búsqueda, una cola q_i de una operación i es una cota inferior del periodo de tiempo comprendido entre la finalización de la operación i y el tiempo de compleción de todas las operaciones en cualquier solución de $Y(r)$.

Claramente, podemos establecer $r_0 = 0$ y $q_* = 0$.

4.2. Cálculo de las cabezas y colas

Una forma de obtener la cabeza r_i de una operación i en un nodo r, sería calcular el camino más pesado posible desde el inicio 0 hasta i en el grafo disyuntivo $G(FD_r) = (V, C \cup FD_r)$, sin tener en cuenta el peso de la operación i. Para que pueda iniciar la operación en cuestión, han de haberse finalizado todos sus predecesores. Sea $y \in Y(r)$, se puede comprobar por inducción en el recorrido en orden topológico del grafo definido por y que, en dicha solución, el valor del camino más pesado desde 0 hasta i (en el grafo definido por y) coincide con el tiempo de inicio más temprano posible para i (la cabeza de 0 se define como 0). Por lo tanto, concluimos que el camino más pesado desde 0 hasta i en $G(FD_r)$ se trata de una cota inferior para el tiempo de inicio más temprano posible de i en todas las soluciones de $Y(r)$ ya que, a medida que fijamos más aristas, el camino más pesado se mantiene o aumenta hasta llegar a una solución, donde se da la igualdad.

Del mismo modo, para cada operación i, podemos obtener una cola q_i, calculando el camino más pesado posible entre la operación i y el fin de todas las operaciones $*$, en el mismo grafo disyuntivo definido anteriormente, de nuevo, sin tener en cuenta el peso de la operación i. Para que todas las operaciones finalicen, al menos han de finalizar todos los sucesores de i. Sea $y \in Y(r)$, se puede comprobar por inducción en el recorrido en orden topológico en sentido inverso del grafo definido por y que, en dicha solución, este valor (en el grafo definido por y) acota inferiormente el periodo de tiempo comprendido entre el fin del procesado de i y el fin del procesado de todas las operaciones (la cola de $*$ se define como 0). Esto se debe a que no se tienen en cuenta otras operaciones que no estén entre sus predecesores (puede que otras operaciones tarden más en completarse). Por todo esto, concluimos que se trata de una cota inferior para todas las soluciones de $Y(r)$ ya que, a medida que fijamos más aristas, el camino más pesado se mantiene o aumenta hasta llegar a una solución, donde ya hemos visto que se trata de una cota inferior.

Para obtener buenas cotas inferiores para la poda, es necesario obtener unas buenas cabezas y colas. Por ello, introduciremos el siguiente método más sofisticado para su cálculo.

Proposición 4.3. *Sea $P(i)$ el conjunto de los predecesores disyuntivos de la operación i en un nodo r cualquiera. El valor*

$$\max_{J \subseteq P(i)} \left\{ \min_{j \in J} r_j + \sum_{j \in J} p_j \right\}$$

es una cota inferior para el tiempo de inicio más temprano de la operación i en cualquier solución alcanzable desde el nodo actual.

Demostración. Sea $J \subseteq P(i)$ cualquiera. Como el conjunto J está formado por predecesores de i, es necesario completar todas las operaciones de este conjunto para poder iniciar i. Veamos que la expresión $\min_{j \in J} r_j + \sum_{j \in J} p_j$ acota inferiormente el tiempo de compleción de J en cualquier solución alcanzable desde el nodo actual. Por un lado, por definición, la cabeza r_i de una operación i es una cota inferior del tiempo de inicio más temprano posible para la operación i en cualquier solución de $Y(r)$, por ello, el mínimo de las cabezas acota inferiormente el tiempo de inicio más temprano posible de cualquier operación de J en cualquier solución de $Y(r)$, es decir, el tiempo de inicio más temprano de J en cualquier solución de $Y(r)$.

Por otro lado, la suma acota inferiormente el tiempo de procesamiento de J, pues no tiene en cuenta si existen tiempos de espera entre las mismas, además, no pueden ser procesadas en paralelo, pertenecen a la misma máquina.

Por todo ello, esta expresión acota inferiormente el tiempo de compleción de todas las operaciones de J y, por tanto, el inicio más temprano de la operación i en cualquier solución alcanzable desde el nodo actual. □

De esta forma, si tomamos el predecesor en el trabajo $h(i)$, el valor $r_{h(i)} + p_{h(i)}$ nos ofrece también una cota inferior por los mismos motivos expuestos anteriormente.

Teniendo todo lo anterior en cuenta, podemos dar la siguiente expresión recursiva para el cálculo la cabeza r_i de la operación i:

$$r_0 = 0$$

$$r_i = \text{máx}\left\{ r_{h(i)} + p_{h(i)},\ \max_{J \subseteq P(i)} \left\{ \min_{j \in J} r_j + \sum_{j \in J} p_j \right\} \right\}$$

Del mismo modo, aplicando estas ideas a las colas, podemos calcular la cola q_i de la operación i, con la siguiente expresión recursiva:

$$q_* = 0$$

$$q_i = \text{máx}\left\{ p_{k(i)} + q_{k(i)},\ \max_{J \subseteq S(i)} \left\{ \sum_{j \in J} p_j + \min_{j \in J} q_j \right\} \right\}$$

donde, $k(i)$ es el sucesor en el trabajo de i y $S(i)$ es el conjunto de sucesores disyuntivos de i.

4.3. Revisión de ciclos

Es posible aprovechar el cálculo de las cabezas para detectar la presencia de ciclos. Para este propósito, utilizaremos el recorrido en orden topológico de un grafo, introducido en el capítulo 2.

Diremos que una operación está etiquetada si ya se ha calculado su cabeza. Consideramos el conjunto D, que contendrá todas las operaciones que están listas para ser etiquetadas, es decir, operaciones que todavía no están etiquetadas y cuentan con la propiedad de que todos sus predecesores están etiquetados. Inicializamos $D = \{0\}$. A la hora de etiquetar una operación $i \in D$, i es eliminado de D y se revisan los sucesores i para posibles inserciones. El proceso finaliza cuando D queda vacío.

Una vez definido este proceso de etiquetado, contamos con el siguiente resultado.

Teorema 4.4. *Sea r un nodo del árbol de búsqueda. El grafo disyuntivo $G(FD_r) = (V, C \cup FD_r)$ no contiene ciclos si y solo si se ha asignado una nueva cabeza al nodo $*$ a través del proceso de etiquetado anterior.*

Demostración. Supongamos primero que el grafo disyuntivo $G(FD_r) = (V, C \cup FD_r)$ no contiene ciclos. En ese caso, procedamos por reducción al absurdo y supongamos que no se ha asignado una nueva cabeza al nodo $*$. Si esto ocurre, quiere decir que algún predecesor de $*$ no ha sido etiquetado. Continuando con esta estrategia y teniendo en cuenta que el número de operaciones es finito, como hemos supuesto que el grafo es acíclico y todos los nodos a excepción del 0 tienen predecesores, llegaremos a la conclusión de que 0 no ha sido etiquetado (recorremos las operaciones en orden inverso al topológico, no es necesario pasar por todas las operaciones), lo cual es una contradicción que viene de suponer que el nodo $*$ no ha sido etiquetado.

Recíprocamente, supongamos ahora que al nodo $*$ se le ha asignado una nueva cabeza. Para ver que el grafo es acíclico, basta aplicar el mismo razonamiento usado en la sección 2.3. Si existiese un ciclo, para etiquetar cualquier operación del mismo, habría sido necesario etiquetarla a ella misma previamente, lo cual no es posible y por lo tanto, el nodo $*$ no sería etiquetado (etiquetar este nodo requiere que todas las operaciones estén etiquetadas, pues es sucesor de todos los nodos). □

Capítulo 5

Método para fijar disyunciones adicionales

5.1. Pares primales y duales

Como ya hemos comentado en el capítulo 3, uno de los objetivos del proceso de ramificación, es conseguir fijar el mayor número de disyunciones lo antes posible cuando vamos de un nodo r a sus sucesores, ya que esto reporta varios beneficios.

A lo largo de este capítulo, presentaremos un método propuesto por Carlier y Pinson [4] para fijar disyunciones adicionales entre operaciones que comparten una misma máquina, dado un nodo del árbol de búsqueda. Este método es independiente del proceso de ramificación.

Hasta el momento, hemos utilizado la palabra solución para referirnos a cualquier selección completa. En realidad, solo estamos interesados en soluciones y, tales que $C_{max}(y) < UB$ y por lo tanto, centraremos la búsqueda en estas soluciones, el resto no aporta ningún beneficio a la búsqueda y hemos de evitar malgastar recursos explorándolas. En lo que resta de capítulo, utilizaremos la palabra solución para referirnos a aquellas que son de interés. De igual manera, utilizaremos $Y(r)$ para referirnos a todas las soluciones alcanzables desde el nodo r. Notemos que, a pesar de este cambio, las cabezas y colas calculadas siguen siendo válidas (las nuevas soluciones, son un subconjunto de las anteriores).

En primer lugar, consideremos el valor UB, que contiene el valor objetivo de la mejor solución encontrada hasta el momento, y ciertas cotas inferiores que presentaremos a continuación. Veamos que, en el caso de que alguna cota inferior supere el UB, podemos establecer ciertas relaciones de precedencia.

Proposición 5.1. *Sea r un nodo del árbol de búsqueda e I el conjunto de operaciones que han de ser procesadas en una máquina. Sea $c \in I$ y $J \subseteq I \setminus \{c\}$, entonces:*

i) Si se da la siguiente desigualdad

$$\min_{j \in J \cup \{c\}} r_j + \sum_{j \in J \cup \{c\}} p_j + \min_{j \in J} q_j \geq UB \tag{5.1}$$

en cualquier solución de $Y(r)$, c ha de procesarse después de todas las operaciones de J. En ese caso, diremos que (J, c) es un par primal.

ii) Si se da la siguiente desigualdad

$$\min_{j\in J} r_j + \sum_{j\in J\cup\{c\}} p_j + \min_{j\in J\cup\{c\}} q_j \geq UB \tag{5.2}$$

en cualquier solución de $Y(r)$, c ha de procesarse antes de todas las operaciones de J. En ese caso, diremos que (c, J) es un par dual.

Demostración. Para ver este resultado, basta proceder por reducción al absurdo. Lo veremos para la primera parte de la proposición. La segunda se obtiene de manera análoga.

Supongamos que se da 5.1 para un conjunto J y una operación c como los descritos en el enunciado. Consideramos una solución $y \in Y(r)$. Supongamos por reducción al absurdo que c no es procesada después de J en y, esto quiere decir que, o bien c es procesada antes que J, o bien, c es procesada entre las operaciones de J.

En primer lugar, veamos que la parte izquierda de la desigualdad 5.1 acota inferiormente el valor objetivo de dicha solución.

Por un lado, el mínimo de las cabezas acota inferiormente el tiempo de inicio más temprano de $J \cup \{c\}$ en y. Por otro lado, la suma acota inferiormente el tiempo de procesamiento de $J \cup \{c\}$, pues no tiene en cuenta si existen tiempos de espera entre las mismas, además, no pueden ser procesadas en paralelo, pertenecen a la misma máquina. Por último, el mínimo de las colas acota inferiormente el periodo de tiempo entre el fin de las operaciones de J y el fin de todas las operaciones en y. En esta solución, c no es procesada después de J, luego el fin de las operaciones de J coincide con el fin de las operaciones de $J \cup \{c\}$.

Por todo lo comentado anteriormente, es claro que esta expresión acota inferiormente el valor objetivo de y. Por lo tanto, como hemos supuesto que se da 5.1, llegamos a una contradicción con la definición de solución, ya que el peso de su camino crítico ($C_{max}(y)$) no puede ser mayor o igual que UB. Esta contradicción viene de suponer que existe una solución con esta característica. Concluiríamos por tanto, que c ha de procesarse después de todas las operaciones de J en cualquier solución de $Y(r)$.

El mismo razonamiento puede ser utilizado para ver el segundo caso. □

Sea (J, c) un par primal, en ese caso, podemos fijar las siguientes disyunciones $\{j \to c : \ j \in J\}$. A estas las llamaremos aristas primales.

Del mismo modo, sea (c, J) un par dual, en ese caso, podemos fijar las siguientes disyunciones $\{c \to j : \ j \in J\}$. A estas las llamaremos aristas duales.

Estas disyunciones pueden ser fijadas ya que, en caso contrario, tenemos garantizado que no se tratan de soluciones y por tanto, no estamos interesados explorarlas.

Además, en caso que $|J| = 1$, contamos con el siguiente resultado también.

Proposición 5.2. *Sea r un nodo del árbol de búsqueda e I el conjunto de operaciones que han de ser procesadas en una máquina. Sean $c, j \in I$ con $c \neq j$, si se da la siguiente desigualdad*

$$r_c + p_c + p_j + q_j \geq UB \tag{5.3}$$

j ha de ser procesado antes que c en cualquier solución de $Y(r)$.

Demostración. Al igual que en la proposición anterior, procedamos por reducción al absurdo.

Supongamos que se da 5.3 para unas operaciones c y j como las descritas en el enunciado. Consideramos una solución $y \in Y(r)$. Supongamos por reducción al absurdo que c es procesada antes que j en y. Veamos que, en ese caso, la parte izquierda de la desigualdad 5.3 acota inferiormente el valor objetivo de dicha solución.

En primer lugar, sabemos que r_c acota inferiormente el tiempo de inicio más temprano de la operación c en y.

Por otro lado, la expresión $p_c + p_j$ acota inferiormente el tiempo de procesamiento de ambas operaciones, pues no tiene en cuenta si existe tiempo de espera entra el procesado de ambas, además, no pueden ser procesadas en paralelo, pues pertenecen a una misma máquina.

Por último, q_j acota inferiormente el periodo de tiempo comprendido entre el fin de la operación j y el fin de todas las operaciones en y.

Como hemos supuesto que en y, c es procesada antes que j, deducimos que la parte izquierda de la desigualdad 5.3 se trata de una cota inferior para el valor objetivo de dicha solución ($C_{max}(y)$). Por lo tanto, como hemos supuesto también que se da 5.3, llegamos a una contradicción con la definición de solución. Esta viene de suponer la existencia de una solución con dicha característica. □

Si nos encontramos bajo las hipótesis de la proposición 5.2 para unas operaciones c y j, podemos fijar la disyunción $j \to c$, a estas las llamaremos aristas directas. De nuevo, estas pueden ser fijadas ya que en otro caso, tenemos garantizado que no se tratan de soluciones.

Sea r un nodo del árbol de búsqueda e I el conjunto de operaciones que han de ser procesadas en una máquina. El siguiente método consigue fijar todas las aristas directas del conjunto I:

```
1: procedure SELECT(I) /*I conjunto de operaciones de una máquina*/
2:     para todo c, j ∈ I, c ≠ j hacer
3:         si r_c + p_c + p_j + q_j ≥ UB ∧ j → c no fijada entonces
4:             Fijar la disyunción j → c
```

Algoritmo 5.1: Algoritmo que permite fijar todas las aristas directas de una máquina

La complejidad temporal de este método es $\theta(|I|^2)$, ya que ha de revisar todos los pares ordenados de operaciones distintas que existen en I.

5.2. Problemas primales y duales

Dado un par primal (J, c), este nos permite deducir cierta información. Del mismo modo, dado un par dual (c, J) se puede obtener la misma información de manera análoga.

Sea (J, c) un par primal, entonces la operación c ha se ser procesada después de todas las operaciones de J en cualquier solución alcanzable desde el nodo del árbol de búsqueda actual. Por lo tanto, obtenemos el siguiente resultado.

Lema 5.3. *Sea r un nodo del árbol de búsqueda y sea (J, c) un par primal entonces,*

en cualquier solución de $Y(r)$, la operación c no puede empezar antes de:

$$r_J = \max_{J' \subseteq J} \left\{ \min_{j \in J'} r_j + \sum_{j \in J'} p_j \right\} \tag{5.4}$$

Demostración. Sea $y \in Y(r)$, veamos que dicho valor acota inferiormente el tiempo de compleción de todas las operaciones de J en y. Como c ha de procesarse después de J en y por ser (J, c) un par primal, se deduce el resultado.

Sea $J' \subseteq J$ cualquiera, para completar todas las operaciones de J hemos de completar todas las de J'. Teniendo en cuenta la definición de cabeza y de mínimo de un conjunto, notemos que el mínimo que aparece en la expresión, acota inferiormente el tiempo de inicio más temprano de J' en y. De igual manera, notemos que el sumatorio acota inferiormente el tiempo de procesamiento de J', pues no se tiene en cuenta los posibles tiempos de espera entre el procesado de las operaciones, ni se pueden procesar en paralelo (pertenecen a una misma máquina). Por lo tanto, concluimos que se trata de una cota inferior del tiempo de compleción de todas las operaciones de J' en y, y por tanto, de J también. □

Sea r un nodo del árbol de búsqueda, consideremos un par primal (J, c). Podemos suponer que r_J configura una cabeza para c ya que, en otro caso, tenemos garantizado que no se tratan de soluciones y no estamos interesados en ellas. En el caso de que $r_c < r_J$, podemos actualizar el valor de r_c con el de r_J, si este fuese el caso, las aristas primales $\{j \to c : j \in J\}$ serían fijadas por el método *SELECT*. Esto se puede deducir gracias al resultado que introduciremos a continuación.

Lema 5.4. *Sea r un nodo del árbol de búsqueda y sea (J, c) un par primal. Si $r_c \geq r_J$, entonces el método SELECT fija todos los arcos primales asociados a (J, c).*

Demostración. Debido a que (J, c) se trata de un par primal, sabemos que:

$$\min_{j \in J \cup \{c\}} r_j + \sum_{j \in J \cup \{c\}} p_j + \min_{j \in J} q_j \geq UB$$

Como $r_c \geq r_J$, tomando $J' = \{j\}\ \forall j \in J$ en la expresión 5.4, se obtiene que $r_c \geq r_j + p_j\ \forall j \in J$ y por tanto, al tratarse de cantidades no negativas, $r_c \geq r_j\ \forall j \in J$. Debido a esto, se tiene que $\min_{j \in J} r_j = \min_{j \in J \cup \{c\}} r_j$ y por tanto:

$$\min_{j \in J} r_j + \sum_{j \in J \cup \{c\}} p_j + \min_{j \in J} q_j \geq UB \tag{5.5}$$

Queremos ver que $r_c + p_c + p_i + q_i \geq UB\ \forall i \in J$, de este modo, el método *SELECT* fijará las aristas deseadas. Sea $i \in J$ cualquiera. Como $r_c \geq r_J$ entonces:

$$r_c + p_c + p_i + q_i \geq r_J + p_c + p_i + q_i$$

Tomando $J' = J$ en la expresión 5.4, deducimos que:

$$r_J \geq \min_{j \in J} r_j + \sum_{j \in J} p_j$$

y por tanto:

$$r_J + p_c + p_i + q_i \geq \min_{j \in J} r_j + \sum_{j \in J} p_j + p_c + p_i + q_i$$

Por último, desprendiéndonos del termino no negativo p_i, teniendo en cuenta la definición de mínimo de un conjunto y la expresión 5.5, obtenemos que:

$$\min_{j \in J} r_j + \sum_{j \in J} p_j + p_c + p_i + q_i \geq \min_{j \in J} r_j + \sum_{j \in J} p_j + p_c + \min_{j \in J} q_j \geq UB$$

De esta forma, obtenemos una cadena de desigualdades que nos permite deducir el resultado. □

Sea r un nodo del árbol de búsqueda y c una operación cualquiera. Consideremos ahora un par primal (J_c, c) tal que $r_{J_c} \geq r_J$ para todo par primal (J, c). Si actualizamos el valor de r_c con el de r_{J_c}, gracias al lema 5.4, sabemos que el método *SELECT* fijaría todas las aristas primales asociadas a todos los pares primales (J, c). Debido a este hecho, es interesante encontrar el valor r_{J_c} para cada operación, ya que esto nos permitiría fijar varias aristas adicionales. Todas estas ideas nos hacen plantearnos el siguiente problema.

Problema primal 5.5. *Sea r un nodo del árbol de búsqueda y sea c una operación. ¿Existe un par primal (J, c) tal que $r_c < r_J$? Si existe, encontrar $r_{J_c} = \max\{r_J : (J, c)$ es par primal$\}$*

Para su resolución, nos apoyaremos en el llamado *Jackson Preemtive Schedule* (*JPS*) para todas las operaciones que han de ser procesadas en la misma máquina que c. El *JPS* es una planificación solución del siguiente problema.

Problema 5.6 (Problema de una máquina relajado)**.** *Sea I un conjunto de operaciones. Estas serán procesadas en una misma máquina. A cada operación i, se le asocia un tiempo de inicio más temprano r_i (también denominado cabeza), un tiempo de procesamiento p_i y un tiempo restante para finalizar la operación q_i (también denominado cola). El objetivo es encontrar una planificación en la que está permitido interrumpir tareas, de forma que minimize el siguiente valor:*

$$C^*_{max} = \max_{i \in I}\{C_i + q_i\} \tag{5.6}$$

donde C_i denota el tiempo de compleción de la operación i.

El *JPS* obtiene una solución utilizando la siguiente regla: en cada instante t, donde t es el tiempo de inicio más temprano o el tiempo de finalización de una operación, se planifica la operación i tal que $q_i = \max\{q_j : r_j \leq t \wedge C_j > t\}$. En caso de empate, en cuanto a la prioridad en las colas se refiere, se elige la operación que haya sido procesada antes en lo que llevamos de planificación. Si ninguna de ellas ha sido procesada hasta el momento, se elige una cualquiera. En otras palabras, cada vez que se pueda iniciar una nueva operación o se finalice la actual, se elige la operación no acabada, que está lista para ser procesada y tiene una mayor cola, deshaciendo los empates como hemos comentado anteriormente.

El siguiente resultado nos garantiza que dicha solución es óptima.

Proposición 5.7. *El JPS ofrece la solución óptima al problema 5.6 (problema de una máquina relajado).*

Demostración. Consideramos el problema 5.6 para un conjunto de operaciones I. En primer lugar, tomemos la operación $j_0 \in I$ de forma que:

$$C^*_{max}(JPS) = C_{j_0} + q_{j_0}$$

Gracias al lema A.3, sabemos que $\exists K \subseteq I$ de forma que:

i) $j_0 \in K$

ii) $C_{j_0} = \text{mín}_{k \in K}\, r_k + \sum_{k \in K} p_k$

iii) $q_k \geq q_{j_0}\ \forall k \in K$

Gracias a los puntos i) y iii) se deduce que $q_{j_0} = \text{mín}_{k \in K}\, q_k$, teniendo en cuenta el punto ii) también, se tiene:

$$C^*_{max}(JPS) = \min_{k \in K} r_k + \sum_{k \in K} p_k + \min_{k \in K} q_k \tag{5.7}$$

Consideremos una planificación P para el problema definido al inicio de la prueba. Si consideramos de nuevo el conjunto K en la planificación P, es claro que:

$$\max_{j \in K}\{C_j + q_j\} \leq \max_{j \in I}\{C_j + q_j\} = C^*_{max}(P) \tag{5.8}$$

Por otro lado, es claro que la expresión $\text{mín}_{k \in K}\, r_k + \sum_{k \in K} p_k$ se trata de una cota inferior del tiempo de compleción de K en cualquier planificación (el mínimo de las cabezas acota inferiormente el inicio de K y el sumatorio acota inferiormente el tiempo de procesamiento de K, pues no tiene en cuenta posibles tiempos de espera), en concreto, lo es para P. Suponiendo que en P, la ultima operación de K procesada es i_0, se tiene:

$$\min_{k \in K} r_k + \sum_{k \in K} p_k \leq \max_{j \in K} C_j = C_{i_0} \tag{5.9}$$

Finalmente, debido a que $i_0 \in K$ por definición, se tiene que:

$$\min_{k \in K} q_k \leq q_{i_0} \tag{5.10}$$

Teniendo en cuenta todo lo comentado anteriormente, se deduce que:

$$\begin{aligned} C^*_{max}(JPS) &= \min_{k \in K} r_k + \sum_{k \in K} p_k + \min_{k \in K} q_k \\ &\leq C_{i_0} + q_{i_0} \leq \max_{j \in K}\{C_j + q_j\} \\ &\leq \max_{j \in I}\{C_j + q_j\} = C^*_{max}(P) \end{aligned}$$

La primera igualdad, se da por 5.7. Para ver la primera desigualdad, basta tener en cuenta 5.9 y 5.10. La segunda desigualdad, se da por ser $i_0 \in K$. El resto de la expresión, se debe a 5.8.
Gracias a esta última expresión, se deduce la optimalidad del *JPS*. □

Por otro lado, gracias al resultado que aparece a continuación, sabemos que el valor $C^*_{max}(JPS)$ para cierta instancia del problema 5.6, da una cota inferior del valor objetivo de todas las soluciones alcanzables desde el nodo del árbol de búsqueda actual.

Proposición 5.8. *Sea r un nodo del árbol de búsqueda e I el conjunto de operaciones que han de procesarse en una máquina para el JSP. El valor $C^*_{max}(JPS)$, para el JPS que surge como solución del problema 5.6 para las operaciones de I (tomando las cabezas y colas obtenidas en el nodo r), ofrece una cota inferior para los valores objetivos de las soluciones de $Y(r)$.*

Demostración. Consideremos el *JPS* definido en el enunciado. Utilizando un desarrollo similar al que encontramos en el inicio de la prueba de la proposición 5.7, es fácil ver que $\exists K \subseteq I$ de forma que:

$$C^*_{max}(JPS) = \min_{k\in K} r_k + \sum_{k\in K} p_k + \min_{k\in K} q_k \tag{5.11}$$

Por otro lado, tengamos en cuenta también, el hecho de que es indiferente hablar de cabezas y colas en el *JPS* que en el nodo nodo r, pues estas coinciden.

Finalmente, es claro que la expresión que aparece a la derecha de la igualdad en 5.11, acota inferiormente el valor objetivo de las soluciones de $Y(r)$. En cualquier solución, hemos de procesar las operaciones de K. El mínimo de las cabezas acota inferiormente el tiempo de inicio de K en cualquier solución de $Y(r)$. Del mismo modo, el mínimo de las colas acota inferiormente el periodo de tiempo comprendido entre fin del procesado de K y el fin del procesado de todas las operaciones en cualquier solución de $Y(r)$. Por último, el sumatorio de los pesos acota inferiormente el tiempo de procesamiento de K, pues no se tienen en cuenta tiempos de espera y además, no pueden ser procesadas en paralelo, pues todas las operaciones son procesadas en una misma máquina ($K \subseteq I$).

Una vez visto que se trata de una cota inferior y teniendo en cuenta 5.11, es fácil deducir el resultado. □

A continuación, introduciremos una serie de conceptos relacionados con el *JPS* que nos serán de utilidad más adelante.

Dado un *JPS* para un conjunto de operaciones I y sea $c \in I$ fijo. Como ya hemos comentado anteriormente, denotaremos por C_j el tiempo de compleción de la operación j. Denotaremos por K^+_c el conjunto de operaciones de I con mayor prioridad en relación a las colas que c y que son completadas después de r_c, es decir:

$$K^+_c = \{j \in I : q_j > q_c \wedge C_j > r_c\} \tag{5.12}$$

Por otro lado, denotaremos por p^+_j el tiempo de procesado restante de la operación j tras r_c. Notemos que $p^+_j > 0 \ \forall j \in K^+_c$.

Finalmente, sea $K \subseteq K^+_c$, podemos definir:

$$t_K = r_c + p_c + \sum_{j\in K} p^+_j + \min_{j\in K} q_j \tag{5.13}$$

Gracias a esta notación, podemos definir un segundo problema primal.

Problema primal 5.9. *Sea r un nodo del árbol de búsqueda e I el conjunto de operaciones que han de ser procesadas en una máquina para el JSP. Sea $c \in I$ fijo. Dado el JPS para las operaciones de I (obtenido tomando las cabezas y colas de r) ¿Existe algún subconjunto no vacío $K \subseteq K^+_c$ de forma que $t_K \geq UB$? Si existe, encontrar el subconjunto K^*_c de cardinal máximo.*

Este segundo problema primal, que inicialmente ha sido introducido sin motivo alguno, está fuertemente relacionado con el primero. Gracias a las ideas de [4] se obtiene el siguiente resultado que relaciona ambos.

Teorema 5.10. *Sea r un nodo del árbol de búsqueda e I el conjunto de operaciones que han de ser procesadas en una máquina para el JSP. Consideramos el JPS para las operaciones de I (obtenido tomando las cabezas y colas de r). Sea $c \in I$ fijo. Entonces, r_{J_c} existe si y solo si K_c^* existe ($K_c^* \neq \emptyset$). En ese caso, se da la siguiente relación:*

$$r_{J_c} = \max_{j \in K_c^*} C_j \tag{5.14}$$

La prueba de este resultado, se encuentra en el apéndice A.

Gracias a este resultado, la resolución del problema primal 5.5 es equivalente a la resolución del problema primal 5.9. En este último, hemos de encontrar el conjunto $K_c^* \subseteq K_c^+$ para una operación c dada. Esto es una tarea más fácil gracias al siguiente resultado.

Proposición 5.11. *Si el conjunto K_c^* que surge como solución de una instancia cualquiera del problema primal 5.9 es no vacío, entonces tiene la siguiente forma:*

$$K_c^* = \{j \in K_c^+ : q_j \geq q_{j_0}\} \text{ para algún } j_0 \in K_c^+ \tag{5.15}$$

Demostración. Basta escoger j_0 de forma que $q_{j_0} = \min_{j \in K_c^*} q_j$ ($j_0 \in K_c^* \subseteq K_c^+$) y tener en cuenta el lema A.9. □

Por otro lado, como hemos comentado anteriormente, se puede seguir un desarrollo similar para poder fijar los arcos duales.

En primer lugar, dado un par dual (c, J), tomamos en cuenta el siguiente valor:

$$q_J = \max_{J' \subseteq J} \left\{ \sum_{j \in J'} p_j + \min_{j \in J'} q_j \right\} \tag{5.16}$$

Siguiendo demostraciones simétricas a las dadas en en los lemas 5.3 y 5.4, es fácil ver que este valor es un candidato a cola para la operación c en las soluciones alcanzables desde el nodo del árbol de búsqueda actual, así como el hecho de que si $q_c \geq q_J$, el método *SELECT* fijará las aristas duales asociadas.

De estos hechos, se deduce la necesidad de plantear y resolver un nuevo problema similar al problema primal 5.5 para los pares duales.

Problema dual 5.12. *Sea r un nodo del árbol de búsqueda y sea c una operación. ¿Existe un par dual (c, J) tal que $q_c < q_J$? Si existe, encontrar $q_{J_c} = \max\{q_J : (c, J)$ es par dual$\}$*

Para resolver el problema primal 5.5, nos basábamos en el llamado *JPS*. Para resolver este nuevo problema, tomaremos en consideración el denominado *Dual Jackson Preemptive Schedule* (*DJPS*) [5] para las operaciones que comparten máquina con c, tomando las cabezas y colas del nodo del árbol de búsqueda en cuestión. El *DJPS* se construye del mismo modo que el *JPS* invirtiendo los roles de las cabezas y colas, es decir, las colas actúan como cabezas y viceversa.

De nuevo, utilizando demostraciones simétricas a las dadas en las proposiciones 5.7 y 5.8, se obtiene que el *DJPS* ofrece la solución óptima a un problema similar al problema 5.6, donde las colas actúan como tiempo de inicio más temprano, las cabezas como colas y se busca minimizar el siguiente valor:

$$C'_{max} = \max_{i \in I}\{C_i + r_i\} \tag{5.17}$$

donde C_i denota el tiempo de compleción de la operación i. Además, el valor C'_{max} para el *DJPS* introducido anteriormente ofrece una cota inferior para los valores objetivos de las soluciones alcanzables desde el nodo del árbol de búsqueda en cuestión. Para la obtención de estos dos últimos resultados, es necesario contar con la versión dual del lema A.3, que se puede encontrar en [5].

Finalmente, debido a que en esta nueva planificación las colas actúan como cabezas, siguiendo las demostraciones dadas, es posible obtener los resultados ya obtenidos para las cabezas, ahora para las colas.

Al igual que en el caso de los pares primales, introduciremos la notación correspondiente.

Dado un *DJPS* para un conjunto de operaciones I, sea $c \in I$ fijo. Como ya hemos comentado anteriormente, denotaremos por C_j el tiempo de compleción de la operación j. Denotaremos por D_c^+ el conjunto de operaciones de I con mayor prioridad en relación a las cabezas que c y que son completadas después de q_c, es decir:

$$D_c^+ = \{j \in I : r_j > r_c \wedge C_j > q_c\} \tag{5.18}$$

Por otro lado, denotaremos por s_j^+ el tiempo de procesado restante de la operación j tras q_c. Notemos que $s_j^+ > 0 \ \forall j \in D_c^+$.

Finalmente, sea $D \subseteq D_c^+$, podemos definir:

$$t_D = \min_{j \in D} r_j + \sum_{j \in D} s_j^+ + p_c + q_c \tag{5.19}$$

De esta forma, es posible plantear un problema similar al problema primal 5.9 con la nueva notación dada y el *DJPS* ya mencionado.

Problema dual 5.13. *Sea r un nodo del árbol de búsqueda e I el conjunto de operaciones que han de ser procesadas en una máquina para el JSP. Sea $c \in I$ fijo. Dado el DJPS para las operaciones de I (obtenido tomando las cabezas y colas de r) ¿Existe algún subconjunto no vacío $D \subseteq D_c^+$ de forma que $t_D \geq UB$? Si existe, encontrar el subconjunto D_c^* de cardinal máximo.*

Finalmente, siguiendo las demostraciones dadas como ya hemos comentado, se obtiene el siguiente teorema:

Teorema 5.14. *Sea r un nodo del árbol de búsqueda e I el conjunto de operaciones que han de ser procesadas en una máquina para el JSP. Consideramos el DJPS para las operaciones de I (obtenido tomando las cabezas y colas de r). Sea $c \in I$ fijo. Entonces, q_{J_c} existe si y solo si D_c^* existe ($D_c^* \neq \emptyset$). En ese caso, se da la siguiente relación:*

$$q_{J_c} = \max_{j \in D_c^*} C_j \tag{5.20}$$

Al igual que en el el caso de los pares primales y, de nuevo, siguiendo la demostración de la proposición 5.11, se obtiene el siguiente resultado.

Proposición 5.15. *Si el conjunto D_c^* que surge como solución de una instancia cualquiera del problema primal 5.13 es no vacío, entonces tiene la siguiente forma:*

$$D_c^* = \{j \in D_c^+ : r_j \geq r_{j_0}\} \text{ para algún } j_0 \in D_c^+ \tag{5.21}$$

5.3. Integración

Una vez visto el desarrollo anterior, estamos en condiciones de exponer el procedimiento para fijar disyunciones adicionales.

Para ello, hemos de contar previamente con los procedimientos para fijar las aristas primales y duales.

A continuación, se expondrá el procedimiento que permite fijar las aristas primales, el proceso para fijar las aristas duales es similar, basta seguir los mismos pasos trabajando con el *DJPS* y la notación introducida al final de la sección anterior para los pares duales.

Realizaremos la siguiente secuencia de acciones para cada máquina. En primer lugar, hemos de computar el *JPS* para las operaciones de la máquina en cuestión (tomando las cabezas y colas del nodo del árbol de búsqueda en el que nos encontramos). Tras ello, basta seguir los siguientes pasos para cada operación c procesada en dicha máquina:

i) Calcular el conjunto K_c^+. Si dicho conjunto es vacío, el problema 5.5 no tiene solución (lema A.5).

ii) Calcular la operación $j_0 \in K_c^+$ con menor cola, de forma que se de la siguiente desigualdad:

$$r_c + p_c + \sum_{\{j \in K_c^+ : q_j \geq q_{j_0}\}} p_j^+ + q_{j_0} \geq UB$$

Si no existe dicha operación, entonces el problema primal 5.9 no tiene solución y por tanto, tampoco la tiene el problema primal 5.5 (teorema 5.10). En otro caso, establecemos:

$$K_c^* = \{j \in K_c^+ : q_j \geq q_{j_0}\}$$

iii) Si $K_c^* \neq \emptyset$, entonces podemos calcular $r_{J_c} = \max_{j \in K_c^*} C_j$ y establecer $r_c = r_{J_c}$.

Finalmente, al pasar el método *SELECT* para las operaciones de dicha máquina, quedarán fijas todas las aristas primales asociadas a las operaciones de la máquina en cuestión, en el nodo del árbol de búsqueda en el que nos encontramos.

Una vez contamos con estos procedimientos, veamos la estructura del proceso para fijar disyunciones adicionales:

i) Cálculo de todos los arcos primales.

ii) Cálculo de nuevas cabezas y colas.

iii) Cálculo de todos los arcos duales.

iv) Cálculo de nuevas cabezas y colas.

Los pasos ii) y iv), son necesarios ya que, si se han fijado nuevas aristas en los pasos anteriores, es posible que los valores de las cabezas y colas aumenten. Del mismo modo, el cálculo de nuevas cabezas y colas puede permitir fijar nuevas disyunciones adicionales. Por todo ello, se repetirán estos pasos hasta que no sea posible fijar ninguna disyunción más.

Por último, notemos que el paso ii) es solo necesario si se ha conseguido fijar algún arco primal, en caso contrario, los valores de las cabezas y colas se mantendrían.

Capítulo 6

Herramientas para el proceso de ramificación y poda

6.1. Cálculo de cotas inferiores

Echando un vistazo al algoritmo de ramificación y poda descrito en el capítulo 3 (Algoritmo 3.1), se observa que, dado un nodo r, para cada sucesor s se calcula una cota inferior del valor objetivo de todas las soluciones alcanzables desde el mismo, $LB(s)$. Si en alguno de los sucesores ocurre que $LB(s) \geq UB$, entonces la inspección del sucesor en cuestión no es necesaria, ya que sabemos que las soluciones que nos ofrece son peores o iguales que la mejor encontrada hasta el momento. Sin embargo, todavía no se ha especificado como se realizará el cálculo de las cotas inferiores.

Basándonos en la evidencia aportada en [2], es beneficioso calcular varias cotas inferiores en distintos momentos del algoritmo, si alguna de ellas sobrepasa el UB, la inspección de ese nodo finaliza.

Sea r un nodo del árbol de búsqueda y s un sucesor de r, veamos entonces cuales son estas cotas para el nodo s:

i) La primera de las cotas inferiores se calcula durante el tratamiento de los conjuntos E_l^B y E_l^A del nodo r. Si s se ha obtenido moviendo una operación i al principio del bloque B, el siguiente valor es una cota inferior para las soluciones del nodo s:

$$r_i + p_i + \max \left\{ \max_{j \in B \setminus \{i\}} (p_j + q_j), \sum_{j \in B \setminus \{i\}} p_j + \min_{j \in B \setminus \{i\}} q_j \right\} \tag{6.1}$$

donde las cabezas y colas han sido tomadas del nodo r.

Del mismo modo, se puede obtener una cota inferior para las soluciones del nodo s, en el caso de que este se haya obtenido moviendo la operación i al final del bloque B:

$$\max \left\{ \max_{j \in B \setminus \{i\}} (r_j + p_j), \min_{j \in B \setminus \{i\}} r_j + \sum_{j \in B \setminus \{i\}} p_j \right\} + p_i + q_i \tag{6.2}$$

de nuevo, las cabezas y colas han sido tomadas del nodo r.

ii) La segunda cota inferior, se obtiene durante el cálculo de las cabezas y colas de s. En este caso, para cada operación se mira si el valor $r_i + p_i + q_i$ es mayor o igual que el UB, en ese caso, la inspección del nodo finaliza. Además de esos valores, también se utilizan los casos particulares q_0 y r_* como cotas inferiores.

iii) La última de las cotas inferiores, se obtiene gracias al *Jackson Preemptive Schedule* (también llamado *JPS*), introducido en el capítulo 5. Este se calcula para las operaciones de cada máquina durante el proceso para fijar disyunciones adicionales en un nodo del árbol de búsqueda (tomando las cabezas y colas de las operaciones del mismo nodo), es este caso, nos centraremos en s. El valor $C^*_{max}(JPS)$ (para los *JPS* indicados anteriormente) definido en el capítulo 5 también, ofrece una cota inferior para todas las soluciones alcanzables desde el nodo del árbol de búsqueda en el que nos encontramos.

Proposición 6.1. *Sea r un nodo del árbol de búsqueda y s un sucesor de r. Las expresiones que acabamos de introducir, acotan inferiormente el valor objetivo de todas las soluciones de $Y(s)$.*

Demostración. Veamos el resultado para cada expresión:

i) En este caso, veremos el resultado para el caso en el que s se obtiene moviendo una operación i, delante del bloque B. El resultado para el otro caso, se obtiene de forma análoga.

Esta cota es comprobada durante el procesado del nodo r, por ello, trabajaremos con las cabezas y colas de r.

En primer lugar, es claro que $r_i + p_i$ acota inferiormente el tiempo de compleción de la operación i en cualquier solución de $Y(r)$, en particular para las de $Y(s)$ también. Por otro lado, sabemos que la operación i es procesada antes que el resto de operaciones del bloque B en todas las soluciones de $Y(s)$. Teniendo esto en consideración, veamos que, para cualquiera de las expresiones que encontramos dentro del máximo en 6.1, se consigue acotar inferiormente el valor objetivo de las soluciones de $Y(s)$.

Para la primera expresión, sea $j \in B \setminus \{i\}$ cualquiera, $p_j + q_j$ acota inferiormente el periodo de tiempo comprendido entre el inicio de la operación j y la finalización de todos los trabajos en cualquier solución de $Y(r)$, en particular para las de $Y(s)$ también. Como no se tiene en cuenta si existe tiempo de espera entre el procesado de las operaciones i y j y las operaciones no se pueden procesar en paralelo, pues se procesan en la misma máquina, queda claro que $r_i + p_i + p_j + q_j$, acota inferiormente el valor objetivo de una solución cualquiera de $Y(s)$.

Para la segunda expresión, notemos los siguientes hechos. El sumatorio acota inferiormente el tiempo de procesamiento de $B \setminus \{i\}$, pues no tiene en cuenta si existen tiempos de espera entre el procesado de las operaciones, además, como las operaciones pertenecen a la misma máquina, no pueden ser procesadas en paralelo.

Por otro lado, utilizando el mismo razonamiento de la prueba de la proposición 4.3, el mínimo que aparece, acota inferiormente el periodo de tiempo comprendido entre el fin de todas las operaciones de $B \setminus \{i\}$ y el fin de todos los trabajos en cualquier solución de $Y(r)$, en particular para las de $Y(s)$ también lo acota.

Como además tampoco se tiene en cuenta si existe tiempo de espera entre el procesado de i y el procesado de la primera operación de $B \setminus \{i\}$, ni se pueden procesar en paralelo (de nuevo, pertenecen a la misma máquina), se deduce que $r_i + p_i + \sum_{j \in B \setminus \{i\}} p_j + \min_{j \in B \setminus \{i\}} q_j$ acota inferiormente el valor objetivo de cualquier solución de $Y(s)$.

Tras haber visto ambas cotas, se deduce el resultado.

ii) Para ver el resultado en este tipo de cotas, basta tener en cuenta la definición de las cabezas y colas de una operación. Para ver los casos particulares, basta con aplicar la definición de nuevo.

iii) El resultado ya ha sido visto en la proposición 5.8.

□

Por último, notemos que, debido al procedimiento introducido en el capítulo 5 para fijar disyunciones adicionales, el valor de las cabezas y colas puede aumentar. Por ello, es recomendable comprobar estas dos últimas cotas inferiores cada vez que se fije alguna disyunción nueva.

6.2. Cálculo de la solución heurística

Echando de nuevo un vistazo al algoritmo de ramificación y poda descrito en el capítulo 3 (Algoritmo 3.1), notemos que es necesario realizar el cálculo de una solución heurística en cada nodo, sin embargo, hasta el momento no hemos especificado cómo hacerlo.

Nuevamente, teniendo en cuenta la evidencia aportada en [2] (basada en un estudio de [7]), presentamos el siguiente método para calcular una solución heurística. En este, se parte del grafo disyuntivo del nodo del árbol de búsqueda en cuestión, sobre el que se fijan nuevas disyunciones de forma iterativa atendiendo a un criterio heurístico. La solución es calculada siguiendo estos pasos:

i) En primer lugar, se calcula el conjunto C que contiene todas las operaciones que están listas para ser planificadas, es decir, aquellas operaciones que todavía no han sido planificadas, con la característica de que todos sus predecesores ya han sido planificados. Inicialmente, $C = \{0\}$. De esta forma, recorremos las operaciones en el orden topológico del grafo disyuntivo, respetando las relaciones de precedencia ya fijadas y por tanto, la solución mantendrá la estructura del nodo del árbol de búsqueda en el que nos encontremos.

ii) Sea $u \in C$ una operación tal que $r_u + p_u = \min_{c \in C}\{r_c + p_c\}$. Sea M_k la máquina en la que se tiene que procesar u. Definimos un nuevo conjunto $\overline{C}$ (también llamado conjunto conflicto) formado por las operaciones de la máquina de u que están listas para ser planificadas y podrían comenzar antes de que u termine:

$$\overline{C} = \{c \in C : \ r_c < r_u + p_u \wedge \text{c es procesada en } M_k\}$$

iii) Para cada operación $c \in \overline{C}$, se obtiene una cota inferior del valor objetivo para las soluciones en las que se ha planificado c como siguiente operación. Se escoge la operación $\overline{c} \in \overline{C}$ con menor cota inferior. Una vez ha sido planificada

la operación $\overline{c}$, puede que se fijen nuevas disyunciones ($\overline{c}$ es procesada antes que el resto de las operaciones de la misma máquina que todavía no han sido planificadas). Por lo tanto, hemos de actualizar las cabezas y colas si fijamos aristas con las que no contamos en el nodo del árbol de búsqueda actual, de esta forma, contamos con mayor información a la hora de planificar la próxima operación (las cabezas determinan la operación u y el conjunto $\overline{C}$ y tanto las cabezas como las colas son utilizadas para calcular la cota inferior). No obstante, notemos que no es necesario realizar el cálculo de las cabezas y colas al completo, basta con propagar las cabezas a partir de los nuevos sucesores de $\overline{c}$ que acabamos de fijar, pues para el resto de iteraciones solo se requiere de esta información.

iv) Se revisan todos los sucesores de $\overline{c}$ para posibles inserciones. Tras esto, $\overline{c}$ es eliminado de C y se vuelve al paso ii).

El pseudocódigo del algoritmo sería el siguiente:

```
1: procedure SOLUCIÓN HEURÍSTICA(r) /*r nodo del árbol de búsqueda*/
2:     Copiar el nodo r /*Trabajaremos sobre el, no perdemos información*/
3:     Inicializar C = {0}
4:     mientras C ≠ ∅ hacer
5:         Escoger u ∈ C tal que r_u + p_u = mín_{c∈C}{r_c + p_c}
6:         Construir el conjunto conflicto C̄
7:         Calcular una cota inferior para cada operación de C̄
8:         Escoger c̄ ∈ C̄ con menor cota inferior
9:         Planificar la tarea c̄ fijando las disyunciones correspondientes
10:        Actualizar cabezas y colas si fuese necesario
11:        Actualizar el conjunto C
12:    return La planificación obtenida
```

Algoritmo 6.1: Algoritmo para el cálculo de una solución heurística

En cuanto a la cota inferior utilizada en el algoritmo, la que ofreció mejores resultados en [7] fue la siguiente.

Sea T el conjunto de las operaciones de la máquina M_k que todavía no han sido planificadas (notemos que $\overline{C} \subseteq T$). Tomaremos como cota inferior el valor $C^*_{max}(JPS)$ del JPS obtenido para el conjunto T, planificando al principio la operación para la cual queremos calcular la cota inferior.

Por último, notemos que cuantas más disyunciones haya fijadas en el nodo del árbol de búsqueda en el que nos encontramos, menos tiempo se tardará en calcular una solución heurística. Esto se debe a que, a medida que fijamos disyunciones, se establece un orden en cada máquina, por ello, disminuye el tamaño de los conjuntos C y $\overline{C}$. En el caso de C, aparecen menos operaciones por cada máquina, pues unas necesitarán del procesado de otras. En el caso de $\overline{C}$, el número de conflictos disminuye al contar con más disyunciones, pues esta información queda reflejada en las cabezas, utilizadas para determinar dicho conjunto. Todo esto garantiza elecciones más rápidas.

Capítulo 7

Implementación y resultados computacionales

7.1. Implementación

Una vez descrito el algoritmo y sus diferentes componentes, estamos en condiciones de implementarlo. Para ello, se ha escogido el lenguaje de programación *Python*.

En cuanto a la implementación, para representar el problema contamos con tres clases.

En primer lugar, la clase *operacion* representa el estado de una operación para un nodo del árbol de búsqueda de una instancia cualquiera del *JSP*. Esta cuenta con un identificador, un peso, una máquina en la que se procesa y sucesores y predecesores en la máquina así como sucesores y predecesores en el trabajo. En cuanto a métodos, se han establecido observadores y modificadores.

Por otro lado, contamos con la clase *nodo*, que representa un nodo del árbol de búsqueda de una instancia cualquiera del *JSP*. Esta cuenta con una lista de operaciones, que representa las operaciones del problema para dicho nodo, una lista de listas de operaciones, que almacenan las operaciones agrupadas por máquinas (esto se hace por cuestiones de eficiencia, será explicado más adelante), una lista que almacena los valores de las cabezas para las operaciones y otra lista que almacena los valores de las colas para las operaciones. En cuanto a los métodos, esta cuenta con los necesarios para realizar el cálculo de cabezas y colas, el fijado de disyunciones adicionales, el cálculo de la solución heurística y la obtención del camino crítico, los bloques así como los conjuntos de ramificación. Adicionalmente contamos con los observadores oportunos.

Finalmente, contamos con la clase *ProblemaJSP*, que representa una instancia del *JSP*. Esta cuenta con una lista de operaciones que representa las operaciones relativas a dicha instancia del *JSP*. Cuenta además con una lista de listas de operaciones, donde se almacenan las operaciones agrupadas por máquinas. Como hemos comentado anteriormente, esto se hace por cuestiones de eficiencia, pues en distintos momentos del algoritmo es necesario acceder a dicha información, de esta forma, evitamos computarla cada vez que sea necesaria. Esta información es compartida con los nodos. Finalmente, cuenta con una variable auxiliar que representa el número de nodos expandidos en cada ejecución del algoritmo de ramificación y poda. En cuanto a métodos, contamos con el método recursivo relativo al algoritmo de ramificación y

poda, así como otro método que prepara y realiza la llamada al anterior. Se ha implementado además, una versión iterativa del mismo algoritmo, equivalente a la de los dos métodos anteriores. Adicionalmente, contamos con los observadores oportunos.

7.2. Resultados computacionales

Una vez descrito el esquema general de la implementación, pasaremos a comentar los resultados computacionales obtenidos para diversas instancias del *JSP*.

En cuanto a las instancias computadas, las tres primeras se corresponden respectivamente con los 6×6, 10×10 y 20×5 *benchmark problems* propuestos en [10]. La cuarta y quinta instancia, se corresponden con los problemas 10×10 planteados en [1]. El resto de ellas, son las denominadas *Lawrence instances*, propuestas en [1] también.

Los resultados obtenidos son los siguientes:

Instancia	Dimensión	Solución óptima	Tiempo	Nodos expandidos
ft06	6 Trabajos, 6 Máquinas	55	0 s	5
ft10	10 Trabajos, 10 Máquinas	930	61 min 22 s	68948
ft20	20 Trabajos, 5 Máquinas	*1193	-	-
abz5	10 Trabajos, 10 Máquinas	1234	28 min 24 s	28599
abz6	10 Trabajos, 10 Máquinas	943	3 min 45 s	4148
la01	10 Trabajos, 5 Máquinas	666	5 s	175
la02	10 Trabajos, 5 Máquinas	655	9 s	370
la03	10 Trabajos, 5 Máquinas	597	22 s	705
la04	10 Trabajos, 5 Máquinas	590	30 s	810
la05	10 Trabajos, 5 Máquinas	593	0 s	1
la06	15 Trabajos, 5 Máquinas	926	4 min 19 s	604
la07	15 Trabajos, 5 Máquinas	890	2 min 29 s	504
la08	15 Trabajos, 5 Máquinas	863	32 min 7 s	2805
la09	15 Trabajos, 5 Máquinas	951	36 s	241
la10	15 Trabajos, 5 Máquinas	958	59 s	270
la11	20 Trabajos, 5 Máquinas	1222	42 min 56 s	694
la12	20 Trabajos, 5 Máquinas	1039	162 min 22 s	1667
la13	20 Trabajos, 5 Máquinas	1150	27 min 6 s	398
la14	20 Trabajos, 5 Máquinas	1292	1 min 51 s	21
la15	20 Trabajos, 5 Máquinas	1207	86 min 59 s	972
la16	10 Trabajos, 10 Máquinas	945	14 min 56 s	21746
la17	10 Trabajos, 10 Máquinas	784	3 min 53 s	3973
la18	10 Trabajos, 10 Máquinas	848	24 min 54 s	27661
la19	10 Trabajos, 10 Máquinas	842	52 min 59 s	59332
la20	10 Trabajos, 10 Máquinas	902	63 min 45 s	75084

Tabla 7.1: Tabla que almacena la información relativa a los resultados computacionales para distintas instancias.

La tabla 7.1 incluye, para cada instancia, su dimensión, el valor objetivo de la

solución óptima encontrada, así como el tiempo empleado y el número de nodos expandidos. Un nodo se considera expandido en el momento en el que comienza a ser procesado.

Cabe destaque que la instancia ft20 no pudo ser resuelta en una sola ejecución del algoritmo, pues se excedió el uso de memoria. Se ha dejado indicado el valor objetivo de la mejor solución encontrada.

Apéndice A

Demostración del teorema 5.10

A continuación, realizaremos la demostración del teorema 5.10. Previo a ello, mostraremos una serie de resultados que nos serán útiles para las pruebas.

A.1. Resultados previos

Lema A.1. *Sea r un nodo del árbol de búsqueda y sea (J, c) un par primal. Si existe una solución en r, entonces se tiene:*

$$\min_{j \in J} q_j > q_c$$

Demostración. Para ver la desigualdad, supongamos por reducción al absurdo que $\min_{j \in J} q_j \leq q_c$. Pueden darse dos casos:

- Caso 1: $r_c \leq \min_{j \in J} r_j$
  En este caso tenemos:

  $$\begin{aligned} &\min_{j \in J} r_j + \sum_{j \in J \cup \{c\}} p_j + \min_{j \in J \cup \{c\}} q_j \geq r_c + \sum_{j \in J \cup \{c\}} p_j + \min_{j \in J} q_j \\ &= \min_{j \in J \cup \{c\}} r_j + \sum_{j \in J \cup \{c\}} p_j + \min_{j \in J} q_j \geq UB \end{aligned}$$

  Para la primera desigualdad, basta tener en cuenta la hipótesis realizada en el Caso 1 y la suposición inicial, realizada por reducción al absurdo. Para la igualdad, utilizamos de nuevo la hipótesis del Caso 1. La última desigualdad es debida a que (J, c) es un par primal.

- Caso 2: $r_c > \min_{j \in J} r_j$
  De nuevo, tenemos las siguientes relaciones:

  $$\min_{j \in J} r_j + \sum_{j \in J \cup \{c\}} p_j + \min_{j \in J \cup \{c\}} q_j = \min_{j \in J \cup \{c\}} r_j + \sum_{j \in J \cup \{c\}} p_j + \min_{j \in J} q_j \geq UB$$

  Para la igualdad, hemos de tener en cuenta la suposición realizada en el Caso 2, esta nos garantiza que r_c no es la menor de las cabezas en $J \cup \{c\}$. Además, hemos de tener en cuenta también la suposición inicial. De nuevo, la desigualdad es debida a que (J, c) es un par primal.

En ambos casos se llega a que (c, J) es un par dual, por lo tanto, no es posible que exista solución, pues c debería ser procesada antes y después de las operaciones de J. Esto último contradice el enunciado, pues es una de las hipótesis. □

Nota: En lo que sigue, podemos suponer que existe una solución en r. Si es cierto, el proceso para fijar disyunciones adicionales es correcto. En caso contrario, este proceso no es correcto, sin embargo es beneficioso también, pues puede que fijemos nuevas aristas con lo que pueden que aumenten las cotas inferiores y podemos antes un nodo que no posee soluciones. De esta forma, se reducen las hipótesis para aplicar el lema A.1.

Lema A.2. *Consideramos un JPS para un conjunto de operaciones I. Sea $J \subseteq I$ y $J' \subseteq J$, entonces:*

$$\min_{i \in J'} r_i + \sum_{i \in J'} p_i \leq \max_{j \in J} C_j$$

Demostración. Como $J' \subseteq J$, para completar todas las operaciones de J, hemos de completar todas las de J', por lo tanto, se tiene que:

$$\max_{j \in J'} C_j \leq \max_{j \in J} C_j$$

Por otro lado, es claro que la expresión $\min_{i \in J'} r_i + \sum_{i \in J'} p_i$ acota inferiormente el tiempo de compleción de todas las operaciones de J', pues esta no tiene en cuenta posibles tiempos de espera ni posibles operaciones que se procesen entre las operaciones de J', además, las operaciones son procesadas en una mísma máquina. De esto se deduce que:

$$\min_{i \in J'} r_i + \sum_{i \in J'} p_i \leq \max_{j \in J'} C_j$$

Gracias a la transitividad, se obtiene el resultado final. □

Lema A.3. *Consideramos un JPS para un conjunto de operaciones I. Sea $j \in I$, entonces, existe $K \subseteq I$ tal que:*

i) $j \in K$

ii) $C_j = \min_{k \in K} r_k + \sum_{k \in K} p_k$

iii) $q_k \geq q_j \ \forall k \in K$

Demostración. Sea u el tiempo más grande, menor o igual que C_j de forma que, en $[u-1, u]$ o bien, ninguna operación es procesada, o bien, se procesa una operación i con $q_i < q_j$. Este u siempre puede ser encontrado, pues es el máximo de un conjunto que al menos contiene el tiempo inicial. A continuación, consideramos el conjunto de las operaciones que han sido procesadas en $[u, C_j]$, lo llamaremos K. En relación a K y al intervalo de tiempo $[u, C_j]$, podemos deducir algunas propiedades:

i) $u < C_j$, en $[C_j - 1, C_j]$ es procesada j ($q_j \not< q_j$), por lo tanto $j \in K$ y $K \neq \emptyset$.

ii) No existen espacios en $[u, C_j]$, si existiesen, u sería mayor, lo cual es una contradicción.

iii) $q_k \geq q_j \ \forall k \in K$, si no, entonces existe $k_0 \in K$ tal que $q_{k_0} < q_j$ y por lo tanto, u sería mayor, llegando de nuevo a una contradicción.

iv) $\text{mín}_{k\in K} r_k = u$. Si $\text{mín}_{k\in K} r_k > u$, en $[u, \text{mín}_{k\in K} r_k]$ no es procesada ninguna operación, eso quiere decir que $[u, C_j]$ tiene un espacio, lo cual contradice el punto ii). Por otro lado, si $\text{mín}_{k\in K} r_k < u$ quiere decir que existe $k_0 \in K$ de forma que, $r_{k_0} < u$, es decir, puede empezar en $u-1$. Además, como $k_0 \in K$, entonces es procesada en $[u, C_j]$ y por tanto en $u-1$ no está terminada. En $[u-1, u]$ sabemos que, o bien, no hay ninguna operación procesada, o bien, es procesada una operación i con $q_i < q_j$ y por lo tanto $q_i < q_j \leq q_{k_0}$ por iii) . En cualquier caso, llegaríamos a la conclusión de que en $[u-1, u]$ se procesa k_0, lo cual entra en contradicción con la elección de u ($q_j \leq q_{k_0}$ por iii)).

v) $C_j - u = \sum_{k\in K} p_k$. Para ver dicha relación, tengamos en cuenta los siguientes hechos:

- Todas las operaciones de K comienzan después de u. Se deduce del punto iv).
- Todas las operaciones de K finalizan antes de C_j. En primer lugar, notemos que todas las operaciones de K pueden comenzar antes de C_j, pues son procesadas en $[u, C_j]$. Sea $k \in K$, si $q_k > q_j$, es claro que será procesada antes que j, pues puede empezar antes de que acabe j y tiene mayor prioridad. Si $q_k = q_j$, debido a como se forma el *JPS*, en especial, a la forma en la que se deshacen los empates, la operación k ha sido procesada completamente antes que j, o completamente después que j. Por ser $k \in K$, k es procesada en $[u, C_j]$ y deducimos que k ha de ser procesada completamente antes que j, con lo que se obtiene el resultado.
- Gracias al punto ii), sabemos que no existen espacios en $[u, C_j]$.

Teniendo en cuenta estos tres puntos, queda claro que en el intervalo $[u, C_j]$ se procesan íntegramente la operaciones de K (y solo las de K, por definición de dicho conjunto) y por lo tanto, se deduce el resultado.

vi) Teniendo en cuenta la información obtenida en los puntos iv) y v), se deduce que $C_j = \text{mín}_{k\in K} r_k + \sum_{k\in K} p_k$

Finalmente, notemos que los puntos i), iii) y vi) concluyen la demostración, ya que muestran que el conjunto K, definido al principio de la prueba, cumple las propiedades buscadas. □

Lema A.4. *Consideramos el JPS para un conjunto de operaciones I. Sea $J \subseteq I$ y $J' \subseteq J$. Si $\text{máx}_{j\in J} C_j \leq r_c$ (para una operación c cualquiera), entonces:*

$$\min_{j\in J'} r_j + \sum_{j\in J'} p_j \leq r_c$$

Demostración. Se tiene que:

$$\min_{j\in J'} r_j + \sum_{j\in J'} p_j \leq \max_{j\in J} C_j \leq r_c$$

La primera desigualdad proviene de aplicar el lema A.2. Para ver la segunda, basta aplicar la hipótesis del enunciado. Esta cadena de desigualdades nos da el resultado. □

Lema A.5. *Sea r un nodo del árbol de búsqueda y sea I el conjunto de operaciones que han de realizarse en una máquina para el JSP. Consideramos el JPS para las operaciones de I (tomando las cabezas y colas del nodo r). Sea $c \in I$ fijo, si $K_c^+ = \emptyset$ entonces, no existe ningún par primal (J, c) en r, de forma que $r_c < r_J$.*

Demostración. En primer lugar, notemos que es indiferente hablar de las cabezas y colas en r que en el *JPS*, pues estas coinciden. Supongamos que $K_c^+ = \emptyset$. Sea (J, c) un par primal en r. Entonces, sabemos que $\max_{j \in J} C_j \leq r_c$, si este no fuese el caso, entonces existe una operación $j \in J$ de forma que $C_j > r_c$. Como además sabemos que $q_j > q_c$ (lema A.1), entonces $j \in K_c^+$, lo cual es una contradicción.
Sea $J' \subseteq J$, con la información que acabamos de obtener, estamos en condiciones de aplicar el lema A.4, con lo que se obtiene que:

$$\min_{j \in J'} r_j + \sum_{j \in J'} p_j \leq r_c$$

Esto nos permite deducir que $r_J \leq r_c$ y por lo tanto, se obtiene el resultado. □

Lema A.6. *Sea r un nodo del árbol de búsqueda y sea I el conjunto de operaciones que han de realizarse en una máquina para el JSP. Consideramos el JPS para las operaciones de I (tomando las cabezas y colas del nodo r). Sea $c \in I$ fijo y sea (J, c) un par primal en r de forma que $J^+ = J \cap K_c^+ \neq \emptyset$, entonces se tiene que $t_{J^+} \geq UB$.*

Demostración. En primer lugar, notemos que es indiferente hablar de las cabezas y colas en r que en el *JPS*, pues estas coinciden. Sea (J, c) un par primal, gracias al lema A.1 sabemos que $q_j > q_c \ \forall j \in J$. Pueden producirse dos casos:

- Caso 1: $p_j^+ = p_j \ \forall j \in J$
  En este caso, se tiene que $J^+ = J$ y por lo tanto:

$$t_{J^+} = r_c + p_c + \sum_{j \in J^+} p_j^+ + \min_{j \in J^+} q_j = r_c + p_c + \sum_{j \in J} p_j + \min_{j \in J} q_j$$
$$\geq \min_{j \in J \cup \{c\}} r_j + \sum_{j \in J \cup \{c\}} p_j + \min_{j \in J} q_j \geq UB$$

  La segunda igualdad, se obtiene debido a que $p_j^+ = p_j \ \forall j \in J$ y a que $J^+ = J$. La primera desigualdad, se debe a que $c \in J \cup \{c\}$ y la última, a que (J, c) es un par primal.

- Caso 2: $\exists \ j \in J / p_j^+ < p_j$
  En este caso, existe una operación de J que se procesa antes de r_c en el *JPS* definido en el enunciado, este hecho nos da la siguiente desigualdad:

$$r_c \geq \min_{j \in J} r_j + \sum_{j \in J} p_j^-$$

  Como sabemos que existe una operación de J que se procesa antes de r_c, su cabeza será menor que r_c y por lo tanto, el mínimo de las cabezas de J también. Por otro lado, en el sumatorio solo tenemos en cuenta los tiempos de procesamiento hasta antes de r_c (por definición de p_j^-). Como además no tenemos en cuenta posibles tiempos de espera ni posibles procesamientos de otras operaciones que no estén en J, se trata de una cota inferior. Notemos además, los dos siguientes hechos:

i) $\min_{j\in J^+} q_j \geq \min_{j\in J} q_j$. Esto es debido a que $J^+ \subseteq J$.

ii) $\sum_{j\in J} p_j = \sum_{j\in J} p_j^- + \sum_{j\in J} p_j^+ = \sum_{j\in J} p_j^- + \sum_{j\in J^+} p_j^+$. La primera igualdad, se obtiene por definición. Para la segunda, sea $j \in J$, si $j \notin J^+$, ha de ser que $j \notin K_c^+$. Como hemos mencionado al inicio de la prueba, $q_j > q_c$, por lo tanto ha de ser que $C_j \leq r_c$, entonces, $p_j^+ = 0$ y por lo tanto, podemos despreciar esas operaciones del sumatorio.

Finalmente, gracias a la información que acabamos de obtener, se deduce que:

$$\begin{aligned} t_{J^+} &= r_c + p_c + \sum_{j\in J^+} p_j^+ + \min_{j\in J^+} q_j \\ &\geq \min_{j\in J} r_J + \sum_{j\in J} p_j^- + p_c + \sum_{j\in J^+} p_j^+ + \min_{j\in J^+} q_j \\ &\geq \min_{j\in J\cup\{c\}} r_J + \sum_{j\in J\cup\{c\}} p_j + \min_{j\in J} q_j \geq UB \end{aligned}$$

Para ver la primera desigualdad, basta tener en cuenta la cota inferior de r_c dada al inicio de este caso. Para ver la segunda desigualdad, notemos que $J \subseteq J \cup \{c\}$ y los puntos i) y ii) expuestos anteriormente en esta prueba. La ultima desigualdad, es consecuencia de que (J, c) es un par primal.

En ambos casos hemos visto que $t_{J^+} \geq UB$ y por tanto, finaliza la prueba. □

Lema A.7. *Consideramos el JPS para un conjunto de operaciones I. Sea $c \in I$ fijo y sean K_1^+ y K_2^+ subconjuntos de K_c^+ que satisfacen que $t_{K_1^+} \geq UB$ y que $t_{K_2^+} \geq UB$, entonces, el conjunto $K = K_1^+ \cup K_2^+$ satisface que $t_K \geq UB$. Por lo tanto, si existe algún subconjunto de K_c^+ cumpliendo dicha desigualdad, existe un único conjunto al que denominaremos como K_c^* de forma que $K_c^* \subseteq K_c^+$, $t_{K_c^*} \geq UB$ y que sea de cardinal maximal.*

Demostración. Podemos suponer sin perdida de generalidad que $\min_{j\in K_1^+} q_j \leq \min_{j\in K_2^+} q_j$. Consideramos el conjunto $K = K_1^+ \cup K_2^+$, entonces tenemos que:

$$\begin{aligned} t_K &= r_c + p_c + \sum_{j\in K} p_j^+ + \min_{j\in K} q_j \\ &\geq r_c + p_c + \sum_{j\in K_1^+} p_j^+ + \min_{j\in K_1^+} q_j \\ &= t_{K_1^+} \geq UB \end{aligned}$$

Para la primera desigualdad, basta tener en cuenta que $p_j^+ > 0\ \forall j \in K$ debido a que $K \subseteq K_c^+$ (por serlo K_1^+ y K_2^+), que $K_1^+ \subseteq K$ y que $\min_{j\in K} q_j = \min_{j\in K_1} q_j$ (se deduce de la suposición realizada al iniciar la demostración). Por último, notemos que la expresión que aparece a la derecha de la primera desigualdad se corresponde con el valor $t_{K_1^+}$, que sabemos que cumple la siguiente propiedad, $t_{K_1^+} \geq UB$. Gracias a este argumento, es fácil deducir el resultado procediendo por reducción al absurdo. □

Lema A.8. *Sea r un nodo del árbol de búsqueda e I el conjunto de operaciones que han de ser procesadas en una máquina para el JSP. Consideramos el JPS para las*

operaciones de I (tomando las cabezas y colas del nodo r). Sea $c \in I$ fijo y sea (J, c) un par primal. Si existe K_c^*, entonces:

$$r_J = \max_{J' \subseteq J} \left\{ \min_{j \in J'} r_j + \sum_{j \in J'} p_j \right\} \leq \max_{j \in K_c^*} C_j$$

Demostración. En primer lugar, notemos que es indiferente hablar de las cabezas y colas en r que en el *JPS*, pues estas coinciden. Gracias al lema A.1, sabemos que $q_j > q_c \ \forall j \in J$. Por otro lado, sea $j \in J$ se pueden dar dos casos:

- Caso 1: $C_j \leq r_c$
  En este caso se tiene que $C_j \leq r_c < \max_{i \in K_c^*} C_i$. La ultima desigualdad se debe a que $K_c^* \subseteq K_c^+$ y por lo tanto, $C_i > r_c \ \forall i \in K_c^*$

- Caso 2: $C_j > r_c$
  Debido a que $q_j > q_c$ (comentado al inicio de la prueba) y a la hipótesis de este caso, se deduce que $j \in K_c^+$. Consideramos el conjunto J, por lo visto anteriormente, $j \in J^+ = J \cap K_c^+$ y podemos aplicar el lema A.6, de donde se deduce que J^+ (subconjunto de K_c^+ por definición) satisface que $t_{J^+} \geq UB$. Por lo tanto, ha de ser $J^+ \subseteq K_c^*$, si este no fuese el caso, podemos considerar el conjunto $K_c^* \cup J^+$ (subconjunto de K_c^+ por serlo cada uno de ellos), por el lema A.7, sabemos que $t_{K_c^* \cup J^+} \geq UB$ y como $|K_c^* \cup J^+| > |K_c^*|$ (ya que hemos supuesto que $J^+ \not\subseteq K_c^*$), llegamos a una contradicción con la definición de K_c^*. Por lo tanto, se tiene que $j \in J^+ \subseteq K_c^*$, con lo que se deduce que $C_j \leq \max_{i \in K_c^*} C_i$

En cualquier caso, hemos visto que $C_j \leq \max_{i \in K_c^*} C_i \ \forall j \in J$. Gracias al lema A.2 y a la información que acabamos de obtener, se deduce lo siguiente:

$$\min_{i \in J'} r_i + \sum_{i \in J'} p_i \leq \max_{j \in J} C_j \leq \max_{i \in K_c^*} C_i \ \forall J' \subseteq J$$

Con lo que se concluye la prueba. □

Lema A.9. *Consideramos el JPS para un conjunto de operaciones I. Sea $c \in I$ fijo. Supongamos que existe el conjunto K_c^*. Dada una operación $j \in I$, si $j \in K_c^+$ y $q_j \geq \min_{i \in K_c^*} q_i$, entonces $j \in K_c^*$.*

Demostración. Supongamos que $j \notin K_c^*$. Consideremos el conjunto $K_c^* \cup \{j\} \subseteq K_c^+$, dicho conjunto satisface que $t_{K_c^* \cup \{j\}} \geq UB$. Esto se debe a que $p_j^+ > 0$ ($j \in K_c^+$) y a que $q_j \geq \min_{i \in K_c^*} q_i$. Como $j \notin K_c^*$, entonces $|K_c^* \cup \{j\}| > |K_c^*|$. Llegamos entonces a una contradicción con la definición de K_c^*, esta viene de suponer que $j \notin K_c^*$. □

A.2. Demostración del teorema 5.10

Gracias a estos resultados previos, estamos en condiciones de realizar la prueba del teorema 5.10.

Teorema 5.10. *Sea r un nodo del árbol de búsqueda e I el conjunto de operaciones que han de ser procesadas en una máquina para el JSP. Consideramos el JPS para*

las operaciones de I (obtenido tomando las cabezas y colas de r). Sea $c \in I$ fijo. Entonces, r_{J_c} existe si y solo si K_c^* existe ($K_c^* \neq \emptyset$). En ese caso, se da la siguiente relación:

$$r_{J_c} = \max_{j \in K_c^*} C_j \tag{5.14}$$

Demostración. En primer lugar, notemos que es indiferente hablar de las cabezas y colas en r que en el *JPS*, pues estas coinciden. A continuación, veamos ambas implicaciones, durante el desarrollo de una de ellas se verá la relación expuesta en el enunciado, con lo que quedaría demostrado.

$\Rightarrow$) Supongamos que existe r_{J_c}, en ese caso, se tiene que:

$$r_{J_c} = \min_{j \in J_c^*} r_j + \sum_{j \in J_c^*} p_j$$

Para algún $J_c^* \subseteq J_c$, de forma que (J_c, c) es un par primal.

Consideramos el *JPS* expuesto en el enunciado. Para ver que $K_c^* \neq \emptyset$, basta ver que existe un conjunto K de forma que, $K \subseteq K_c^+$ y $t_K \geq UB$. De esa forma, K_c^* es el máximo (en cuanto a cardinal se refiere) de un conjunto no vacío y por lo tanto, este tampoco lo será.

Teniendo en cuenta la definición de r_{J_c} y el lema A.2 se tiene:

$$r_c < r_{J_c} = \min_{j \in J_c^*} r_j + \sum_{j \in J_c^*} p_j \leq \max_{j \in J_c} C_j$$

Esto nos permite deducir que existe una operación $j \in J_c$ de forma que $C_j > r_c$. Por otro lado, como (J_c, c) es un par primal, gracias al lema A.1 se deduce que $q_j > q_c$. Estos dos últimos datos nos indican que $j \in K_c^+$. Finalmente, considerando el conjunto J_c y aplicando el lema A.6 (es posible pues $j \in J_c^+$), se obtiene que el conjunto J_c^+ (subconjunto de K_c^+ por definición) satisface que $t_{J_c^+} \geq UB$. De esta forma, hemos encontrado el conjunto buscado y podemos deducir que $K_c^* \neq \emptyset$.

$\Leftarrow$) Supongamos que $K_c^* \neq \emptyset$. Para esta implicación, nos centraremos en el estudio del *JPS* descrito en el enunciado para poder encontrar r_{J_c}. Distinguimos dos casos:

- Caso 1: $\min_{j \in K_c^*} r_j < r_c$
  Consideramos el intervalo de tiempo $[r_c - 1, r_c]$. Sea j la operación procesada en dicho intervalo. Sabemos que en dicho intervalo es procesada una operación, pues al menos la operación con menor cabeza de K_c^* está disponible en ese intervalo de tiempo, en $r_c - 1$ puede ser empezada (por la hipótesis del caso) y no está terminada ($K_c^* \subseteq K_c^+$ y la operaciones de K_c^+ son procesadas después de r_c por definición). Además, debido a lo que acabamos de comentar se tiene:

  $$q_j \geq \min_{i \in K_c^*} q_i = q_{j_1}$$

  Si no fuese el caso, la operación con menor cabeza de K_c^* sería procesada en $[r_c - 1, r_c]$ (acabamos de ver que está disponible y en ese caso, tendría

mayor cola) y no j, siendo estas operaciones distintas (en este caso tendrían distinta cola).
Sea δ el tiempo más grande, menor o igual que r_c, de forma que en $[\delta-1, \delta]$, o bien, la máquina está vacía, o bien, es procesada una operación i con $q_i < q_{j_1}$. Utilizando un argumento similar al de la prueba del lema A.3, deducimos que siempre podemos encontrar dicho δ. Sea J^- el conjunto de operaciones procesadas en $[\delta, r_c]$ en el JPS y sea $J_c = J^- \cup K_c^*$, se puede deducir la siguiente información:

i) $\delta < r_c$ ya que en $[r_c - 1, r_c]$ es procesada j con $q_j \geq q_{j_1}$. Por lo tanto, $j \in J^-$ y entonces $J^- \neq \emptyset$.

ii) En $[\delta, r_c]$ no hay espacios, si los hubiese δ sería mayor.

iii) $q_j \geq q_{j_1}\ \forall j \in J^-$, si no fuese el caso, δ sería mayor.

iv) $\text{mín}_{j \in J^-} r_j = \delta$. Si $\text{mín}_{j \in J^-} r_j > \delta$, entonces en $[\delta, \text{mín}_{j \in J^-} r_j]$ habría un espacio, lo cual contradice el punto ii). Si $\text{mín}_{j \in J^-} r_j < \delta$, entonces existe una operación $t \in J^-$ que en $\delta - 1$ puede empezar y además, no está terminada ($t \in J^-$). En el intervalo $[\delta - 1, \delta]$, o bien, no es procesada ninguna operación, o bien, es procesada una operación i con $q_i < q_{j_1}$ y por lo tanto, $q_i < q_t$ (punto iii)). En cualquier caso, llegaríamos a la conclusión de que en dicho intervalo es procesada t, lo cual supone una contradicción con la definición de δ ($q_t \geq q_{j_1}$ por el punto iii)).

v) Debido al punto ii), al punto iv) (este nos dice que las operaciones no han sido procesadas antes de δ) y a la definición de p_j^-, tenemos que:

$$r_c - \delta = \sum_{j \in J^-} p_j^-$$

Si tenemos en cuenta el punto iv) de nuevo, llegamos a la siguiente expresión:

$$r_c = \min_{j \in J^-} r_j + \sum_{j \in J^-} p_j^-$$

vi) $\text{mín}_{j \in J^-} r_j = \delta \leq \text{mín}_{j \in K_c^*} r_j$. Si $\delta > \text{mín}_{j \in K_c^*} r_j$, entonces existe una operación $t \in K_c^*$ disponible en $\delta - 1$ pues puede iniciar y no está terminada ($t \in K_c^* \subseteq K_c^+$, las operaciones de K_c^+ son procesadas después de r_c y $d < r_c$ por el punto i)). Por otro lado, sabemos que en el intervalo $[\delta - 1, \delta]$ o bien, no es procesada ninguna operación, o bien, es procesada una operación i con $q_i < q_{j_1}$ y por lo tanto, $q_i < q_t$ ($t \in K_c^*$). En ambos caso, llegamos a la conclusión de que t ha de procesarse en dicho intervalo, entrando en contradicción con la definición de δ ($q_t \geq q_{j_1}$ por ser $t \in K_c^*$).

A continuación, veamos que (J_c, c) es un par primal. Es claro que $J_c \subseteq I$

y que $c \notin J_c$, además se tiene que:

$$\begin{aligned}
&\min_{j\in J_c\cup\{c\}} r_j + \sum_{j\in J_c\cup\{c\}} p_j + \min_{j\in J_c} q_j \\
&= \min_{j\in J_c} r_j + \sum_{j\in J_c} p_j^- + \sum_{j\in J_c} p_j^+ + p_c + \min_{j\in J_c} q_j \\
&\geq \min_{j\in J^-} r_j + \sum_{j\in J^-} p_j^- + \sum_{j\in J_c} p_j^+ + p_c + \min_{j\in J_c} q_j \\
&= r_c + \sum_{j\in J_c} p_j^+ + p_c + \min_{j\in J_c} q_j \\
&\geq r_c + \sum_{j\in K_c^*} p_j^+ + p_c + \min_{j\in K_c^*} q_j \geq UB
\end{aligned}$$

Para la primera igualdad, hemos de tener en cuenta la hipótesis del caso y las definiciones de p_j^- y p_j^+. La primera desigualdad se debe al punto vi), a que $J^- \subseteq J_c$ y a que los p_j^- son cantidades no negativas. Para ver la segunda igualdad, basta tener en cuenta el punto v). La segunda desigualdad, se da debido a que $K_c^* \subseteq J_c$, los p_j^+ son cantidades no negativas y al punto iii). La última desigualdad se da por definición de K_c^*.
Esta cadena de desigualdades nos permiten deducir que (J_c, c) es un par primal. Además, gracias al lema A.1 sabemos también que:

$$q_j > q_c \ \forall j \in J_c \tag{A.1}$$

Consideremos ahora la operación $j_0 \in K_c^*$ de forma que $C_{j_0} = \max_{j\in K_c^*} C_j$. Gracias al lema A.3, sabemos que existe un conjunto $K \subseteq I$ de forma que:

$$j_0 \in K \text{ por lo tanto, } K \neq \emptyset$$

$$C_{j_0} = \min_{i\in K} r_i + \sum_{i\in K} p_i$$

$$q_{j_0} = \min_{i\in K} q_i$$

Siendo K el conjunto de operaciones procesadas en el periodo $[u, C_{j_0}]$ para un u determinado (dicha elección se justifica en la prueba del lema A.3). Es claro que tiene que ser $u \geq \delta$, si $u < \delta$, entonces fijémonos que el intervalo $[\delta - 1, \delta]$ cae dentro de $[u, C_{j_0}]$. En dicho intervalo sabemos que, o bien, no se procesa ninguna operación, o bien, se procesa una operación i con $q_i < q_{j_1}$ y por lo tanto, $q_i < q_{j_1} \leq q_{j_0}$. En cualquier caso, vemos que no es posible, pues contradicen los puntos ii) y iii) respectivamente, de la demostración del lema A.3 que necesariamente han de cumplirse (propiedades relativas al intervalo $[u, C_{j_0}]$). Teniendo también en cuenta el punto iv) de dicha prueba, se tiene que:

$$\min_{j\in K} r_j = u \geq \delta \tag{A.2}$$

Veamos a continuación, que $K \subseteq J_c$. Sea $k \in K$, puede ocurrir:

- $k \in J^-$, por lo tanto $k \in J_c$, pues $J^- \subseteq J_c$.

- $k \notin J^-$. Debido a A.2, sabemos que $r_k \geq \delta$. Por otro lado, como $k \notin J^-$, entonces k no ha sido procesada en el intervalo $[\delta, r_c]$, por lo tanto, solo queda una posibilidad, k ha sido procesada íntegramente después de r_c. Además, debido a que $k \in K$, $j_0 \in K_c^*$, $K_c^* \subseteq J_c$ y a A.1 se tiene:

$$q_k \geq q_{j_0} \geq q_{j_1} > q_c \tag{A.3}$$

Debido a A.3 y a que k ha sido procesada íntegramente después de r_c, se deduce que $k \in K_c^+$. De nuevo, gracias a A.3, podemos aplicar el lema A.9 y obtenemos que $k \in K_c^*$, por lo tanto, $k \in J_c$ pues $K_c^* \subseteq J_c$.

En cualquier caso hemos visto que $k \in J_c$, por lo tanto, $K \subseteq J_c$.
Tomemos ahora $J_c^* = K$, tenemos:

i) $J_c^* \subseteq J$ siendo (J_c, c) un par primal.

ii) $r_{J_c} \geq \min_{i \in J_c^*} r_i + \sum_{i \in J_c^*} p_i = C_{j_0}$. Esto se debe a que $J_c^* \subseteq J_c$, $J_c^* = K$ y a la definición de K. Por último, como $j_0 \in K_c^* \subseteq K_c^+$, $C_{j_0} > r_c$ pues las operaciones de K_c^+ son procesadas después de r_c, por lo tanto, se concluye que $r_{J_c} > r_c$.

iii) Gracias al lema A.8, a la definición de K y a que $J_c^* = K$, sea (J, c) un par primal, se tiene:

$$r_J \leq \max_{j \in K_c^*} C_j = C_{j_0} = \min_{i \in J_c^*} r_i + \sum_{i \in J_c^*} p_i$$

En particular, tomando $J = J_c$, se deduce que $r_{J_c} = \min_{i \in J_c^*} r_i + \sum_{i \in J_c^*} p_i$ y por lo tanto, $r_{J_c} = \max\{r_J : (J, c) \text{ es par primal}\}$.

Finalmente, se tiene que :

$$r_{J_c} = \min_{i \in J_c^*} r_i + \sum_{i \in J_c^*} p_i = C_{j_0} = \max_{j \in K_c^*} C_j$$

Siendo esta la igualdad buscada en la prueba. De esta forma, se concluye la prueba para este caso.

- Caso 2: $\min_{j \in K_c^*} r_j \geq r_c$
Debido a la hipótesis, se tiene que $p_j^+ = p_j \ \forall j \in K_c^*$. Sea j la operación procesada en $[r_c, r_c + 1]$. Sabemos que la máquina no está vacía, pues c puede empezar y no está finalizada en ese momento. En función de j podemos distinguir dos subcasos:

  - Caso 2.1: $q_j < \min_{i \in K_c^*} q_i$
  Tomemos $J_c = K_c^*$, es claro que $J_c \subseteq I$ y que $c \notin J_c$, además se tiene que:

$$\begin{aligned} &\min_{j \in J_c \cup \{c\}} r_j + \sum_{j \in J_c \cup \{c\}} p_j + \min_{j \in J_c} q_j \\ &= r_c + p_c + \sum_{j \in J_c} p_j^+ + \min_{j \in J_c} q_j \\ &= r_c + p_c + \sum_{j \in K_c^*} p_j^+ + \min_{j \in K_c^*} q_j \geq UB \end{aligned}$$

La primera igualdad, se debe a la hipótesis del Caso 2 y a que $p_j^+ = p_j \ \forall j \in K_c^*$ (comentado al inicio del Caso 2). La segunda igualdad, se debe a que $J_c = K_c^*$. Por último, la desigualdad final se da por definición de K_c^*. Gracias a esto, se deduce que (J_c, c) es un par primal. Además, gracias al lema A.1 sabemos que:

$$q_j > q_c \ \forall j \in J_c \tag{A.4}$$

Por otro lado, consideremos de nuevo la operación $j_0 \in K_c^*$ de forma que $C_{j_0} = \max_{j \in K_c^*} C_j$. Gracias al lema A.3, sabemos que existe un conjunto $K \subseteq I$ de forma que:

$$j_0 \in K \text{ por lo tanto, } K \neq \emptyset$$

$$C_{j_0} = \min_{i \in K} r_i + \sum_{i \in K} p_i$$

$$q_{j_0} = \min_{i \in K} q_i$$

Siendo K el conjunto de operaciones procesadas en el periodo $[u, C_{j_0}]$, para un u determinado (dicha elección se justifica en la prueba del lema A.3).
A continuación, veamos que $K \subseteq J_c$. Sea $k \in K$. Por un lado, se tiene que:

$$q_k \geq q_{j_0} \geq \min_{i \in K_c^*} q_i > q_c \tag{A.5}$$

La primera desigualdad, se da por ser $k \in K$ y por definición de q_{j_0}, la segunda, por ser $j_0 \in K_c^*$ y la última, por ser $K_c^* = J_c$ y por A.4.
Por otro lado, se tiene que $u > r_c$. Si $u \leq r_c$, entonces, $u \leq r_c < C_{j_0}$ y por lo tanto, $j \in K$, con lo que tendríamos que $q_j \geq q_{j_0}$ (por definición de q_{j_0}). Sin embargo, sabemos que $q_{j_0} \geq \min_{i \in K_c^*} q_i > q_j$. La primera desigualdad se da por ser $j_0 \in K_c^*$ y la segunda por la hipótesis del Caso 2.1. Esto se trata de una contradicción, que viene de suponer que $u \leq r_c$.
Observando la prueba del lema A.3, sabemos que $u = \min_{i \in K} r_i$, como $u > r_c$, se tiene que $r_i > r_c \ \forall i \in K$, en particular, $r_k > r_c$. Debido a este último dato y a A.5, se obtiene que $k \in K_c^+$. Además, gracias a A.5 de nuevo, podemos aplicar el lema A.9 obteniendo así que $k \in K_c^* = J_c$. Hemos visto por tanto que $K \subseteq J_c$.
Tomemos $J_c^* = K$, entonces tenemos:

i) $J_c^* \subseteq J_c$ siendo (J_c, c) un par primal.
ii) $r_{J_c} \geq \min_{i \in J_c^*} r_i + \sum_{i \in J_c^*} p_i = C_{j_0}$. Esto se debe a que $J_c^* \subseteq J_c$, $J_c^* = K$ y a la definición de K. Por último, como $j_0 \in K_c^* \subseteq K_c^+$, $C_{j_0} > r_c$ pues las operaciones de K_c^+ son procesadas después de r_c, por lo tanto, se concluye que $r_{J_c} > r_c$.
iii) Gracias al lema A.8, a la definición de K y a que $J_c^* = K$, sea (J, c) un par primal, se tiene:

$$r_J \leq \max_{j \in K_c^*} C_j = C_{j_0} = \min_{i \in J_c^*} r_i + \sum_{i \in J_c^*} p_i$$

En particular, tomando $J = J_c$, se deduce que $r_{J_c} = \min_{i \in J_c^*} r_i + \sum_{i \in J_c^*} p_i$ y por lo tanto, $r_{J_c} = \max\{r_J : (J, c) \text{ es par primal}\}$.

Finalmente, se tiene que :

$$r_{J_c} = \min_{i \in J_c^*} r_i + \sum_{i \in J_c^*} p_i = C_{j_0} = \max_{j \in K_c^*} C_j$$

Siendo esta la igualdad buscada en la prueba. De esta forma, se concluye la prueba para el Caso 2.1.

- Caso 2.2: $q_j \geq \min_{i \in K_c^*} q_i$

  Por comodidad, diremos que $q_{j_1} = \min_{i \in K_c^*} q_i$. Sea δ el tiempo más grande posible, menor o igual que r_c, de forma que en el intervalo $[\delta - 1, \delta]$, o bien, no es procesada ninguna operación, o bien, es procesada una operación i tal que $q_i < q_{j_1}$. Utilizando un argumento similar al de la prueba del lema A.3, deducimos que siempre podemos encontrar dicho δ. Sea J^- el conjunto de operaciones procesadas en el intervalo $[\delta, r_c]$ y sea $J_c = J^- \cup K_c^*$. Veamos que (J_c, c) es un par primal. En función de J^- distinguimos dos casos:

  - $J^- = \emptyset$

    En este caso, $J_c = K_c^*$. Es claro que $J_c \subseteq I$ y que $c \notin J_c$, además se tiene que:

    $$\begin{aligned} &\min_{j \in J_c \cup \{c\}} r_j + \sum_{j \in J_c \cup \{c\}} p_j + \min_{j \in J_c} q_j \\ &= r_c + p_c + \sum_{j \in J_c} p_j^+ + \min_{j \in J_c} q_j \\ &= r_c + p_c + \sum_{j \in K_c^*} p_j^+ + \min_{j \in K_c^*} q_j \geq UB \end{aligned}$$

    La primera igualdad, se debe a la hipótesis del Caso 2 y a que $p_j^+ = p_j \ \forall j \in K_c^*$ (comentado al inicio del Caso 2). La segunda igualdad, se debe a que $J_c = K_c^*$. Por último, la desigualdad final se da por definición de K_c^*. Gracias a esto, se deduce que (J_c, c) es un par primal.

  - $J^- \neq \emptyset$

    En este caso, podemos deducir la siguiente información:

    i) No existen espacios en $[\delta, r_c]$, si hubiese alguno, δ seria mayor.

    ii) $q_j \geq q_{j_1} \ \forall j \in J^-$, si este no fuese el caso, δ seria mayor.

    iii) $\min_{j \in J^-} r_j = \delta$. Si $\min_{j \in J^-} r_j > \delta$, entonces en el intervalo $[\delta, \min_{j \in J^-} r_j]$ habría un espacio y por lo tanto, lo habría en $[\delta, r_c]$, contradiciendo el punto i). Si $\min_{j \in J^-} r_j < \delta$, entonces existe una operación $t \in J^-$ que en $\delta - 1$ puede empezar y además no está terminada ($t \in J^-$). En el intervalo $[\delta - 1, \delta]$, o bien, no es procesada ninguna operación, o bien, es procesada una operación i con $q_i < q_{j_1}$ y por lo tanto, $q_i < q_t$ (punto ii)). En cualquier caso, llegaríamos a la conclusión de que en dicho intervalo es procesada t, lo cual supone una contradicción con la definición de δ ($q_t \geq q_{j_1}$ por el punto ii)).

iv) Debido al punto i), al punto iii) y a la definición de p_j^-, tenemos que:

$$r_c - \delta = \sum_{j \in J^-} p_j^-$$

Si tenemos en cuenta el punto iii) de nuevo, llegamos a la siguiente expresión:

$$r_c = \min_{j \in J^-} r_j + \sum_{j \in J^-} p_j^-$$

v) $\min_{j \in J^-} r_j = \delta \leq \min_{j \in K_c^*} r_j$. Si $\delta > \min_{j \in K_c^*} r_j$, entonces existe una operación $t \in K_c^*$ disponible en $\delta - 1$, pues puede iniciar y no está terminada ($t \in K_c^* \subseteq K_c^+$ y las operaciones de K_c^+ son procesadas después de r_c). Por otro lado, sabemos que en el intervalo $[\delta - 1, \delta]$, o bien, no es procesada ninguna operación, o bien, es procesada una operación i con $q_i < q_{j_1}$ y por lo tanto, $q_i < q_t$ ($q_t \geq q_{j_1}$ por ser $t \in K_c^*$). En ambos caso, llegamos a la conclusión de que t ha de procesarse en dicho intervalo, entrando en contradicción con la definición de δ ($q_t \geq q_{j_1}$ por ser $t \in K_c^*$).

De nuevo, es claro que $J_c \subseteq I$ y que $c \notin J_c$, además se tiene que:

$$\begin{aligned}
&\min_{j \in J_c \cup \{c\}} r_j + \sum_{j \in J_c \cup \{c\}} p_j + \min_{j \in J_c} q_j \\
&= \min_{j \in J_c} r_j + \sum_{j \in J_c} p_j^- + \sum_{j \in J_c} p_j^+ + p_c + \min_{j \in J_c} q_j \\
&\geq \min_{j \in J^-} r_j + \sum_{j \in J^-} p_j^- + \sum_{j \in J_c} p_j^+ + p_c + \min_{j \in J_c} q_j \\
&= r_c + \sum_{j \in J_c} p_j^+ + p_c + \min_{j \in J_c} q_j \\
&\geq r_c + \sum_{j \in K_c^*} p_j^+ + p_c + \min_{j \in K_c^*} q_j \geq UB
\end{aligned}$$

Para la primera igualdad, hemos de tener en cuenta que $\min_{j \in J_c} r_j \leq \min_{j \in J^-} r_j = \delta \leq r_c$ ($J^- \subseteq J_c$, punto iii) y definición de δ) y las definiciones de p_j^- y p_j^+. La primera desigualdad se debe al punto v), a que $J^- \subseteq J_c$ y a que los p_j^- son cantidades no negativas. Para ver la segunda igualdad, basta tener en cuenta el punto iv). La segunda desigualdad se da debido a que $K_c^* \subseteq J_c$, los p_j^+ son cantidades no negativas y al punto ii). La desigualdad final se da por definición de K_c^*.

Esta cadena de desigualdades, nos permiten deducir que (J_c, c) es un par primal.

En cualquier caso, hemos visto que (J_c, c) es un par primal. Además, gracias al lema A.1 se tiene:

$$q_j > q_c \ \forall j \in J_c \tag{A.6}$$

Por otro lado, consideremos de nuevo la operación $j_0 \in K_c^*$ de forma que $C_{j_0} = \max_{j \in K_c^*} C_j$. Gracias al lema A.3, sabemos que existe un conjunto $K \subseteq I$ de forma que:

$$j_0 \in K \text{ por lo tanto, } K \neq \emptyset$$

$$C_{j_0} = \min_{i \in K} r_i + \sum_{i \in K} p_i$$

$$q_{j_0} = \min_{i \in K} q_i$$

Siendo K el conjunto de operaciones procesadas en el periodo $[u, C_{j_0}]$ para un u determinado (dicha elección se justifica en la prueba del lema A.3).
Es claro que tiene que ser $u \geq \delta$, si $u < \delta$, entonces fijémonos que el intervalo $[\delta - 1, \delta]$ cae dentro de $[u, C_{j_0}]$. En dicho intervalo, sabemos que, o bien, no se procesa ninguna operación, o bien, se procesa una operación i con $q_i < q_{j_1}$ y por lo tanto, $q_i < q_{j_1} \leq q_{j_0}$ ($j_0 \in K_c^*$). En cualquier caso, vemos que no es posible, pues contradicen los puntos ii) y iii) respectivamente, de la demostración del lema A.3 que necesariamente han de cumplirse (propiedades relativas al intervalo $[u, C_{j_0}]$). Teniendo también en cuenta el punto iv) de dicha prueba, se tiene que:

$$\min_{j \in K} r_j = u \geq \delta \tag{A.7}$$

Veamos a continuación, que $K \subseteq J_c$. Sea $k \in K$, puede ocurrir:

- $k \in J^-$, por lo tanto $k \in J_c$, pues $J^- \subseteq J_c$.
- $k \notin J^-$. Debido a A.7, sabemos que $r_k \geq \delta$. Por otro lado, como $k \notin J^-$, entonces k no ha sido procesada en el intervalo $[\delta, r_c]$, por lo tanto, solo queda una posibilidad, k ha sido procesada íntegramente después de r_c. Además, debido a que $k \in K$, $j_0 \in K_c^*$, $K_c^* \subseteq J_c$ y a A.6 se tiene:

  $$q_k \geq q_{j_0} \geq q_{j_1} > q_c \tag{A.8}$$

  Debido a A.8 y a que k ha sido procesada íntegramente después de r_c, se deduce que $k \in K_c^+$. De nuevo, gracias a A.8, podemos aplicar el lema A.9 y obtenemos que $k \in K_c^*$, por lo tanto, $k \in J_c$ pues $K_c^* \subseteq J_c$.

En cualquier caso, hemos visto que $k \in J_c$, por lo tanto, $K \subseteq J_c$.
Tomemos ahora $J_c^* = K$, tenemos:

i) $J_c^* \subseteq J_c$ siendo (J_c, c) un par primal.
ii) $r_{J_c} \geq \min_{i \in J_c^*} r_i + \sum_{i \in J_c^*} p_i = C_{j_0}$. Esto se debe a que $J_c^* \subseteq J_c$, $J_c^* = K$ y a la definición de K. Por último, como $j_0 \in K_c^* \subseteq K_c^+$, $C_{j_0} > r_c$ pues las operaciones de K_c^+ son procesadas después de r_c, por lo tanto, se concluye que $r_{J_c} > r_c$.
iii) Gracias al lema A.8, a la definición de K y a que $J_c^* = K$, sea (J, c) un par primal, se tiene:

$$r_J \leq \max_{j \in K_c^*} C_j = C_{j_0} = \min_{i \in J_c^*} r_i + \sum_{i \in J_c^*} p_i$$

En particular, tomando $J = J_c$, se deduce que $r_{J_c} = \min_{i \in J_c^*} r_i + \sum_{i \in J_c^*} p_i$ y por lo tanto, $r_{J_c} = \max\{r_J : (J, c) \text{ es par primal}\}$.

Finalmente, se tiene que :

$$r_{J_c} = \min_{i \in J_c^*} r_i + \sum_{i \in J_c^*} p_i = C_{j_0} = \max_{j \in K_c^*} C_j$$

Siendo esta la igualdad buscada en la prueba. De esta forma, se concluye la prueba para el Caso 2.2 y por lo tanto, para el Caso 2, finalizando así la prueba del teorema 5.10.

□

Bibliografía

[1] J. Adams, E. Balas, and D. Zawack. The shifting bottleneck procedure for job-shop scheduling. *Management Science*, 34:391–401, 1988.

[2] P. Brucker, B. Jurisch, and B. Sievers. A branch and bound algorithm for the job-shop scheduling problem. *Discrete Applied Mathematics*, 49:107–127, 1994.

[3] J. Carlier and E. Pinson. An algorithm for solving the job-shop problem. *Management Science*, 35:164–176, 1989.

[4] J. Carlier and E. Pinson. A practical use of jackson's preemptive schedule for solving the job shop problem. *Annals of Operations Research*, 26:269–287, 1990.

[5] J. Carlier and E. Pinson. Adjustment of heads and tails for the job-shop problem. *European Journal of Operational Research*, 78:146–161, 1994.

[6] J. Grabowski, E. Nowicki, and S. Zdrzalka. A block approach for single machine scheduling with relase dates and due dates. *European Journal of Operational Research*, 26:278–285, 1986.

[7] B. Jurisch and B. Sievers. Ein branch and bound verfahren fur des job shop scheduling problem. *Osnabrucker Schriften zur Mathematik Reihe P*, Heft 122.

[8] J. Lenstra, A. H. G. R. Kan, and P. Brucker. Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, 1:343–362, 1977.

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

[10] J. Muth and G. Thompson. Industrial scheduling. *Prentice-Hall, Englewood Cliffs, NJ*, 1963.

[11] B. Roy and B. Sussmann. Les problemes d'ordonnancement avec contraintes disjonctives. *SEMA*, Note DS:9, 1964.