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

ALGORITMOS DE APROXIMACIÓN PARA LA PROGRAMACIÓN DE TRABAJOS DIVISIBLES CON TIEMPOS DE INSTALACIÓN EN MÁQUINAS PARALELAS

TESIS PARA OPTAR AL GRADO DE MAGÍSTER EN GESTIÓN DE OPERACIONES
MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL MATEMÁTICO

VÍCTOR IGNACIO VERDUGO SILVA

PROFESOR GUÍA:
JOSÉ RAFAEL CORREA HAEUSSLER

PROFESOR CO-GUÍA:
JOSÉ CLAUDIO VERSCHAE TANNENBAUM

MIEMBROS DE LA COMISIÓN:
JOSÉ ANTONIO SOTO SAN MARTÍN
FERNANDO IVÁN ORDOÑEZ PIZARRO

Agradece financiamiento de CONICYT y al Núcleo Milenio Información y Coordinación en Redes.

SANTIAGO DE CHILE
ENERO 2014

RESUMEN DE LA TESIS
PARA OPTAR AL GRADO DE
MAGÍSTER EN GESTIÓN DE OPERACIONES
RESUMEN DE LA MEMORIA
PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL MATEMÁTICO
POR: VÍCTOR VERDUGO SILVA
FECHA: 24/01/2013
PROF. GUÍA: JOSÉ CORREA

ALGORITMOS DE APROXIMACIÓN PARA LA PROGRAMACIÓN DE TRABAJOS DIVISIBLES CON TIEMPOS DE INSTALACIÓN EN MÁQUINAS PARALELAS

En este trabajo se estudian problemas de programación de tareas en un entorno de máquinas paralelas. A diferencia de la literatura clásica, asumimos que los trabajos pueden ser divididos en distintas partes, cada una de las cuales puede ser procesada en distintas máquinas. Antes de procesar cualquier parte de un trabajo, la máquina debe prepararse y requiere un tiempo de instalación. Primero se estudia el problema de minimizar la suma ponderada de tiempos de completación, para el cual se obtiene una $(2+\varepsilon)$-aproximación cuando los tiempos de instalación son todos iguales. Este resultado corresponde al primer algoritmo de aproximación de factor constante para este problema. Usando técnicas similares se diseña una 2-aproximación para el caso de una ponderación uniforme de los trabajos, que en particular mejora el factor 2.781 obtenido por Schalekamp et al. [22]. Finalmente, con un algoritmo de *programación en lista*, se obtiene una 4-aproximación para el problema original con tiempos de instalación dependientes del trabajo.

Posteriormente se estudia el problema en máquinas no relacionadas, donde los tiempos de proceso e instalación dependen de cada máquina. Los algoritmos diseñados en esta sección están basados en técnicas de redondeo de relajaciones lineales. La primera relajación que se estudia permite diseñar una 3-aproximación para el problema. Al realizar un paso de *lift and project* sobre una restricción es posible fortalecer la relajación, lo que permite diseñar una $(1+\phi)$-aproximación, donde $\phi = \frac{\sqrt{5}+1}{2}$. Respecto a la inaproximabilidad del problema se demuestra una cota inferior igual a $\frac{e}{e-1}$ basada en un resultado de Feige [8] para *Max-k-Cover*. Usando la relajación lineal fuerte se muestra una 2-aproximación para la versión del problema en que cada trabajo posee un conjunto restringido de máquinas en las que puede ser procesado, teniendo igual tiempo de instalación y procesamiento en todas ellas. Finalmente, se estudia relajaciones basadas en *configuraciones* sobre trabajos, es decir, las variables corresponden a vectores que representan la asignación de un trabajo a máquinas en una cierta programación. El programa lineal de configuraciones de trabajos posee una cantidad infinita de variables, sin embargo, se demuestra que es posible restringirse a una cantidad finita de ellas y que además es posible aproximar este programa lineal en tiempo polinomial a un factor de $1+\varepsilon$. Determinar el gap de integralidad de esta relajación queda como una pregunta abierta.

AGRADECIMIENTOS

Agradezco a mi familia, su apoyo y cariño ha sido siempre fundamental. Agradezco también a mi amigos, por todos los momentos que hemos vivido y que sin duda seguiremos teniendo.

A José Correa y José Verschae, muchas gracias por todo el tiempo, apoyo y consejos que me han dado en este tiempo. Son muy buenos investigadores y también excelentes personas. He aprendido mucho de ustedes.

También agradezco a Leen Stougie, por haberme recibido durante mi estadía en Amsterdam. El trabajo que realizamos allá dio origen a esta tesis.

Quiero agradecer a los profesores y funcionarios del DIM y MGO por la excelente labor que realizan.

Finalmente, agradezco a Conicyt y al Núcleo Milenio de Información y Coordinación en Redes por el financiamiento otorgado para sacar adelante este trabajo.

Tabla de contenido

Tabla de contenido IV

Índice de figuras V

1. Introducción 1
1.1. Problemas de programación de tareas . . . 1
1.1.1. Clasificación de los problemas de programación de tareas . . . 1
1.2. Algoritmos de aproximación . . . 3
1.2.1. Esquemas de aproximación a tiempo polinomial . . . 4
1.2.2. Minimizando tiempo total de completación: $1|r_j|\sum C_j$ . . . 5
1.2.3. Programando máquinas no relacionadas: $R||C_{\text{máx}}$ . . . 6
1.3. Programando tareas con división del trabajo y tiempos de instalación . . . 10
1.4. Estructura de los capítulos y contribuciones de este trabajo . . . 10

2. Algoritmos de aproximación para $P|s_j, \textbf{split}|\sum w_j C_j$ 12
2.1. Descripción del problema y trabajo relacionado . . . 12
2.2. Contribución de este trabajo . . . 13
2.3. Programando órdenes de trabajos . . . 13
2.4. Programación de tareas con tiempos de instalación . . . 19

3. Algoritmos de aproximación para $R|\textbf{split,setup}|C_{\text{máx}}$ 25
3.1. Definición del problema y trabajo relacionado . . . 25
3.2. Contribución de este trabajo . . . 26
3.3. Una 3-aproximación . . . 27
3.3.1. Construcción del conjunto A . . . 28
3.4. Una $(1+\phi)$-aproximación . . . 32
3.5. Una 2-aproximación bajo asignación restringida . . . 37
3.6. Cota inferior en la aproximabilidad del problema . . . 39
3.7. PL de configuraciones . . . 40
3.7.1. PL de configuraciones para las máquinas . . . 40
3.7.2. PL de configuraciones para trabajos . . . 42
3.7.3. Proyección de [CLP] al espacio de asignaciones . . . 46

Bibliografía 49

Índice de figuras

3.1. Cada componente conexa de $G'(x)$ es un pseudo-árbol. 28
3.2. Las aristas con línea continua corresponden al conjunto A. 29
3.3. Ejemplo que muestra que el mejor factor para [LST] es 3. 32
3.4. Ejemplo que muestra que el mejor factor para [$\text{LST}_{\text{strong}}$] es $1 + \phi$. 36
3.5. Gadget para la reducción desde Max-k-Cover. 40
3.6. Ejemplo que muestra que el mejor factor posible para [MCLP] es $1 + \phi$. . . 42

Capítulo 1

Introducción

1.1. Problemas de programación de tareas

Los problemas de *scheduling* aparecen cuando es necesario distribuir algún recurso escaso para la realización de un conjunto de actividades, con el fin de optimizar sobre alguna medida de rendimiento. Dependiendo de la situación, estos recursos y actividades pueden aparecer en distintas formas. Recursos pueden ser máquinas de ensamblaje en una planta productora, CPU, memoria y dispositivos I/O en un computador, pistas de un aeropuerto, mecánicos en un taller de reparación automotriz, etc. Actividades pueden ser las operaciones realizadas durante el ensamblaje en un planta, la ejecución de algún programa computacional, despegues y aterrizajes en las pistas del aeropuerto, los autos reparados en el taller, etc. El objetivo a optimizar puede variar dependiendo también de la situación, requiriendo por ejemplo en un caso usar lo menos posible los recursos para llevar a cabo las actividades, mientras que en otro caso se podría buscar minimizar el tiempo total que toma realizarlas.

El estudio de este tipo de problemas tuvo sus orígenes en los años '50. Fue en esa época en que investigadores de distintas áreas comenzaron a desarrollar algoritmos para encontrar buenas soluciones a estos problemas. Por ejemplo, desde el punto de vista industrial, el desarrollo de técnicas que permitan encontrar buenas soluciones permite que la empresa se mantenga competitiva. En los años '60 también llamó la atención de investigadores en Ciencias de la Computación, puesto que los recursos computacionales en ese tiempo eran bastante escasos, por lo cual buscar una buena forma de ejecutar los distintos procesos en un computador era un problema clave. Posteriormente, con el desarrollo de la Teoría de Complejidad Computacional, diversos investigadores probaron que habían muchos de estos problemas que eran difíciles, en un sentido que precisaremos más adelante.

1.1.1. Clasificación de los problemas de programación de tareas

En lo que sigue, nos referiremos por *máquinas* a los recursos disponibles, y por *trabajos* a las distintas actividades por realizar. No es difícil notar que un problema de scheduling no solo depende de cuantas máquinas y trabajos dispongamos, sino que también de las características de ellos. Por ejemplo, es distinto el problema en que todas las máquinas son idénticas al problema en que las máquinas poseen distintas velocidades, y en consecuencia un trabajo puede ser procesado más rápido en una máquina que en otra. También

podría ocurrir que hubiesen trabajos que simplemente no pueden ser asignados a cualquier máquina debido al nivel de especialización que requiera la tarea. Y además, el problema también depende del objetivo sobre el que se busca optimizar, pues a pesar de tener el mismo ambiente de máquinas y trabajos el cambio en el objetivo puede generar una solución totalmente distinta.

Usualmente, en un problema de programación de tareas los tiempos de procesamiento de un trabajo j en la máquina i se denota por p_{ij}. En presencia de pesos para los trabajos, estos se denotarán por w_j, y el tiempo de completación del trabajo j en la programación se denota por C_j. Dada la gran diversidad de problemas que se generan al variar cada uno de las partes involucradas en el problema, en el año 1979, Graham et al. [11] introdujeron una notación de tres campos, representada en el objeto $\alpha|\beta|\gamma$. El primer campo α refiere al ambiente de las máquinas, el β al ambiente de los trabajos y γ al objetivo sobre el cual se optimiza.

1. Campo α.

   - $\alpha = 1$ (*Una máquina*) Hay solo una máquina en el sistema.
   - $\alpha = P$ (*Máquinas idénticas*) En este ambiente las máquinas son idénticas y pueden funcionar en paralelo.
   - $\alpha = Q$ (*Máquinas relacionadas*) En este ambiente las máquinas también funcionan en paralelo pero a diferentes velocidades. Cada máquina i es representada por su velocidad v_i y el tiempo que tarda en ser procesado completamente un trabajo en esta máquina es $p_{ij} = p_j/v_i$.
   - $\alpha = R$ (*Máquinas no relacionadas*) Al igual que en los casos anteriores, las máquinas funcionan en paralelo, pero ahora cada máquina puede procesar cada trabajo a una velocidad distinta. En este caso el tiempo que tarda la máquina i en procesar completamente un trabajo j lo denotamos simplemente por p_{ij}.

   Cuando el número de máquinas paralelas no es parte del input se agrega un m a los campos anteriores: Pm, Qm y Rm.

2. Campo β.
   Este campo asociado a las características de los trabajos puede tener múltiples entradas.

   - $\beta =$ pmtn (*Interrupciones*) Los trabajos pueden ser interrumpidos para seguir siendo procesados posteriormente. Cada parte puede ser procesada en distintas máquinas pero no simultáneamente. Se hará referencia a este ambiente cuando se hable de *interrupción* de trabajos.
   - $\beta = r_j$ (*Tiempos de disponibilidad*) Cada trabajo posee un tiempo a partir del cual está disponible en el sistema.
   - $\beta =$ prec (*Restricciones de precedencias*) Esta característica hace referencia al hecho de que ciertos trabajos no pueden ser procesados hasta que otros hayan sido finalizados. Estas precedencias son representadas mediante un digrafo acíclico, en donde cada nodo representa un trabajo, y si la arista (i, j) está entonces el trabajo i debe ser procesado antes que el trabajo j. Si es que cada

trabajo posee a lo más un antecesor en el orden y a lo más un sucesor, entonces escribimos *chains* en vez de *prec*.

- β = split (*Trabajos divisibles*) Los trabajos pueden ser divididos en distintas partes, que a diferencia del ambiente con *interrupciones*, estas si pueden ser procesadas simultáneamente en distintas máquinas. Se hará referencia a este ambiente cuando se hable de *trabajos divisibles* o *división de trabajos*.
- β = setup (*Tiempos de instalación*) Antes de que cada trabajo sea procesado en una máquina, se requiere un tiempo de instalación o preparación de la máquina. Denotamos por s_{ij} al tiempo de instalación requerido para el trabajo j en la máquina i. Cuando los tiempos de instalación solo dependen del trabajo simplemente se denotará por s_j a a este tiempo.

3. Campo γ. Este campo representa la función objetivo a minimizar.

   - $\gamma = C_{\text{máx}}$ (*Makespan*) El objetivo es minimizar el tiempo en que el último trabajo es completado.
   - $\gamma = \sum_j C_j$ (*Suma de los tiempos de completación*) El objetivo es minimizar la suma de los tiempos en que cada trabajo es completado.
   - $\gamma = \sum_j w_j C_j$ (*Suma ponderada de los tiempos de completación*) Cada trabajo tiene un peso positivo w_j, y el objetivo es minimizar la suma de los tiempos de completación de los trabajos, ponderados por su respectivo peso.

1.2. Algoritmos de aproximación

El poder de computación desarrollado durante las últimas décadas, ha permitido que problemas que antes resultaban inmanejables ahora se puedan tratar obteniendo resultados en un tiempo corto. Decidir niveles de inventario, scheduling, ruteo de vehículos, clasificación de datos, diseño de sistemas de búsqueda y localización son algunos de los problemas estudiados por el área de la *optimización discreta*. Sin embargo, muchos de estos problemas resultan ser difíciles. Desde el punto de vista de la teoría de complejidad computacional son problemas $\mathcal{NP}$-*duros*, es decir, a menos que $\mathcal{P} = \mathcal{NP}$, no existen algoritmos *eficientes* para encontrar la solución óptima. Por *eficiente* nos referimos a un algoritmo que corre en tiempo acotado por un polinomio en el tamaño de la instancia, codificada en binario.

Entonces la pregunta que surge es, ¿cómo lidiar con este tipo de problemas? Existe un trade-off entre buscar soluciones óptimas y el tiempo de ejecución del algoritmo. Una forma de atacar el problema es diseñando un algoritmo que busque soluciones exactas, usando *branch y bound*, *branch y cut* y técnicas de Programación Entera, por ejemplo. Sin embargo, a pesar de que estos algoritmos pueden correr en tiempo razonable para instancias pequeñas, para instancias muy grandes los tiempos de ejecución se pueden volver inmanejables. Otra opción, es buscar algoritmos eficientes que no necesariamente entreguen la solución óptima al problema. Dentro de estos algoritmos, los *algoritmos de aproximación* son algoritmos eficientes que a pesar de que no encuentran necesariamente el óptimo al problema, entregan una solución cercana al óptimo, en un sentido que se precisa a continuación.

Definición 1.1. Dado un problema de minimización Π con función de costos c, un algoritmo $\mathcal{A}$ a tiempo polinomial es una *ρ-aproximación* para Π si para todo instancia I del problema, se tiene que el output $\mathcal{A}(I)$ del algoritmo satisface

$$c(\mathcal{A}(I)) \leq \rho \cdot \text{opt}(I),$$

donde $\text{opt}(I)$ es el costo de la solución óptima al problema Π. Cuando ρ no depende del tamaño de la instancia diremos que $\mathcal{A}$ es una *algoritmo de aproximación de factor constante*.

1.2.1. Esquemas de aproximación a tiempo polinomial

La pregunta natural que surge al estudiar algoritmos de aproximación para un problema es cuan bueno podría llegar a ser el factor de aproximación. En algunos casos, es posible obtener aproximaciones arbitrariamente buenas, por ejemplo, para *Knapsack* [14] y el *Euclidean Travelling Salesman Problem* [2].

Definición 1.2. Dado un problema de minimización, un *esquema de aproximación a tiempo polinomial (PTAS)* es una colección de algoritmos $\{A_\varepsilon\}_{\varepsilon>0}$ tal que A_ε es una $(1+\varepsilon)$-aproximación.

Es importante notar que ε no es parte del input del problema, y por lo tanto el tiempo de ejecución podrá ser exponencial en $1/\varepsilon$. Cuando el tiempo de ejecución es también polinomial en $1/\varepsilon$ decimos que $\{A_\varepsilon\}_{\varepsilon>0}$ es un *esquema de aproximación a tiempo completamente polinomial (FPTAS).*

A lo largo de este capítulo hemos asumido que las instancias de los distintos problemas de optimización han sido codificadas en binario. El tamaño binario de una instancia I, denotado como $|I|$, se define con el número de bits necesarios para codificar I en binario. Si denotamos por I_u a la codificación de I en unario, su tamaño unario $|I_u|$ corresponde el número de bits necesarios para escribir I_u.

Definición 1.3. Un algoritmo a tiempo *pseudo-polinomial* es un algoritmo cuyo tiempo de ejecución en una instancia I es acotado por un polinomio en $|I_u|$.

De esta forma, hay problemas que a pesar de ser $\mathcal{NP}$-duros, admiten un algoritmo pseudo-polinomial que lo resuelve. Por ejemplo, *Knapsack* es $\mathcal{NP}$-duro, pero admite un algoritmo pseudo-polinomial basado en programación dinámica que lo resuelve de manera óptima [29]. Sin embargo, hay problemas $\mathcal{NP}$-duros para los cuales tampoco existe un algoritmo pseudo-polinomial que los resuelva, bajo el supuesto que $\mathcal{P} \neq \mathcal{NP}$.

Definición 1.4. Un problema Π es *fuertemente $\mathcal{NP}$-duro* si todo problema en $\mathcal{NP}$ puede ser reducido en tiempo polinomial a Π de manera que los números en la reducción están escritos en unario.

El siguiente teorema muestra que son pocos los problemas $\mathcal{NP}$-duros que admiten un FPTAS.

Teorema 1.5 ([10])**.** *Sea Π un problema de optimización fuertemente $\mathcal{NP}$-duro. Entonces Π no admite un FPTAS, salvo que $\mathcal{P} \neq \mathcal{NP}$.*

La importancia de este hecho es que varios problemas de programación de tareas se reducen desde *3-Partition*, el cual es fuertemente $\mathcal{NP}$-duro [10]. En particular, en el Capítulo 2 estudiaremos un problema que se reduce desde *3-Partition*, y por lo tanto un PTAS es lo mejor que se esperaría poder diseñar, salvo que $\mathcal{P} = \mathcal{NP}$. En la siguiente sección mostraremos dos ejemplos de algoritmos de aproximación, en el contexto de programación de tareas.

1.2.2. Minimizando tiempo total de completación: $1|r_j|\sum C_j$

El primer ejemplo que veremos corresponde a un problema de programación de tareas en una máquina. El algoritmo que se muestra tiene la particularidad de hacer uso de otro problema de programación de tareas para el cual si se conoce una forma eficiente de resolverlo. Se tiene n trabajos, cada uno con tiempo de procesamiento $p_j > 0$ y tiempos de disponibilidad en el sistema $r_j \geq 0$. Una programación para este problema en una asignación de los trabajos a la máquina donde a lo más un trabajo es procesado en un instante de tiempo, ningún trabajo comienza a ser procesado antes de su tiempo de disponibilidad y cada trabajo una vez que empieza a ser procesado debe ser completado antes que cualquier otro trabajo comience a ser procesado. En la notación de tres campos introducidas en la Sección 1.1.1 corresponde al problema $1|r_j|\sum C_j$. Este problema es $\mathcal{NP}$-duro [16], por lo que buscar un algoritmo de aproximación es una buena estrategia para abordarlo.

La versión con interrupciones de este problema, es decir, $1|r_j, \text{pmtn}|\sum C_j$, puede ser resuelto de manera óptima en tiempo polinomial [4] vía la regla de procesamiento *Shortest Remaining Processing Time First* (SRPT): en cada instante de tiempo se procesa el trabajo con menor cantidad de trabajo pendiente siempre y cuando esté disponible en el sistema y no haya sido completado aun. Sea C_j^P el tiempo de completación del trabajo j en una programación óptima con interrupciones y denotemos por opt la suma de los tiempos de completación en una programación óptima para $1|r_j|\sum C_j$. Dado que una programación óptima para $1|r_j|\sum C_j$ es factible para la versión del problema con interrupciones, se sigue que $\sum_{j=1}^{n} C_j^P \leq \text{opt}$. El siguiente algoritmo nos entrega una programación para $1|r_j|\sum C_j$, donde C_j^N denotará el tiempo de completación del trabajo j en la programación sin interrupciones.

Algorithm 1 Programando según SRPT

1: Encontramos una programación con interrupciones óptima usando SRPT.
2: Ordenamos los trabajos según el tiempo en que finalizan en la programación con interrupciones.
3: Definimos $C_1^N = r_1 + p_1$.
4: **for** $j = 2, \ldots, n$ **do**
5: El trabajo j es completado en el tiempo $C_j^N = \text{máx}\{C_{j-1}^N, r_j\} + p_j$.
6: **end for**

Es decir, una vez que los trabajos han sido ordenados según $C_1^P \leq C_2^P \leq \cdots \leq C_n^P$, se procesa el trabajo 1 desde su tiempo de disponibilidad en el sistema r_1 hasta el tiempo en que es completado $r_1 + p_1$. Después el trabajo 2 se comienza a procesar tan pronto como sea

posible una vez que el trabajo 1 es completado, es decir, desde el tiempo máx$\{r_1 + p_1, r_2\}$ hasta máx$\{r_1 + p_1, r_2\} + p_2$. El resto de los trabajos se procesan análogamente. En el siguiente teorema se prueba que este algoritmo corresponde a una 2-aproximación para $1|r_j|\sum C_j$.

Teorema 1.6. *El algoritmo Programando según SRPT es una 2-aproximación para el problema* $1|r_j|\sum C_j$.

Demostración. Dado que en la programación con interrupciones óptima el trabajo j es procesado después de $1, \ldots, j-1$ se tiene que

$$C_j^P \geq \max_{k\in\{1,\ldots,j\}} r_k \quad \text{y} \quad C_j^P \geq \sum_{k=1}^{j} p_k.$$

Además, por construcción de la programación se tiene que $C_j^N \geq \max_{k\in\{1,\ldots,j\}} r_k$. Consideremos un tiempo en que la máquina está en desuso. Estos tiempos ocurren solo cuando el siguiente trabajo por procesar aun no ha sido liberado al sistema. Luego, entre el tiempo $\max_{k\in\{1,\ldots,j\}} r_k$ y C_j^N no existe ningún período de tiempo donde la máquina se encuentre en desuso. De esta forma, se tiene que

$$C_j^N \leq \max_{k\in\{1,\ldots,j\}} r_k + \sum_{k=1}^{j} p_k \leq 2C_j^P,$$

donde la última desigualdad proviene de las cotas inferiores en C_j^P mencionadas anteriormente. Se obtiene finalmente que

$$\sum_{j=1}^{n} C_j^N \leq 2\sum_{j=1}^{n} C_j^P \leq 2 \cdot \text{opt}.$$

□

Como se pudo ver en este ejemplo, el factor de aproximación del algoritmo depende fuertemente de la calidad de las cotas inferiores en el óptimo del problema. La forma en como se obtienen estas cotas depende mucho del problema, por lo que no hay una receta general para obtenerlas.

1.2.3. Programando máquinas no relacionadas: $R||C_{\max}$

Un esquema que ha resultado ser muy útil en el diseño de algoritmos de aproximación consiste en encontrar una formulación basada en programación entera para el problema de optimización y luego relajar las restricciones de integralidad. A continuación, se muestra un ejemplo también en el ámbito de programación de tareas, basado en una relajación lineal para el problema.

Se tiene n trabajos, cada uno de los cuales debe ser asignado exactamente a una de m máquinas. Si el trabajo j es asignado a la máquina i, entonces requiere p_{ij} unidades de tiempo para ser procesado sin interrupciones. El objetivo es encontrar una programación que minimice el máximo tiempo de procesamiento requerido por alguna máquina. En la

notación introducida en la sección 1.1.1 corresponde al problema $R||C_{\text{máx}}$. Este problema es $\mathcal{NP}$-duro [10] y el primer algoritmo de aproximación a tiempo polinomial y de factor constante para este problema fue desarrollado por Lenstra et al. [17].

El algoritmo se basa en una formulación lineal para el problema de decisión asociado: cada máquina i está disponible por T unidades de tiempo para procesar trabajos y la variable $x_{ij} \in \{0, 1\}$ indica si el trabajo j es asignado a la máquina i en la programación. Sea $J_i(t) = \{j \in J : p_{ij} \leq t\}$ los trabajos que pueden ser procesados en la máquina i en un tiempo no mayor a t, y $M_j(t) = \{i \in M : p_{ij} \leq t\}$ las máquinas que pueden procesar el trabajo j en un tiempo no mayor a t. El siguiente teorema es clave para el desarrollo del algoritmo.

Teorema 1.7 (Lenstra et al. [17])**.** *Sea* $[LP(t)]$ *la relajación dada por:*

$$[LP(t)]:$$

$$\sum_{i \in M_j(t)} x_{ij} = 1 \qquad \text{para todo } j \in \{1, \ldots, n\}, \tag{1.1}$$

$$\sum_{j \in J_i(t)} x_{ij} p_{ij} \leq T \qquad \text{para todo } i \in \{1, \ldots, m\}, \tag{1.2}$$

$$x_{ij} \geq 0 \qquad \text{para todo } j \in J_i(t),\ i \in \{1, \ldots, m\}. \tag{1.3}$$

Si $[LP(t)]$ *posee solución, entonces cualquier punto extremo de este polítopo puede ser redondeado a una solución factible* $\bar{x}$ *de* $[IP(t)]$ *dado por:*

$$[IP(t)]:$$

$$\sum_{i \in M_j(t)} x_{ij} = 1 \qquad \text{para todo } j \in \{1, \ldots, n\}, \tag{1.4}$$

$$\sum_{j \in J_i(t)} x_{ij} p_{ij} \leq T + t \qquad \text{para todo } i \in \{1, \ldots, m\}, \tag{1.5}$$

$$x_{ij} \in \{0, 1\} \qquad \text{para todo } j \in J_i(t),\ i \in \{1, \ldots, m\}. \tag{1.6}$$

Además, este procedimiento puede ser realizado en tiempo polinomial.

Nótese que en [LP(t)] se tiene que $x_{ij} = 0$ para todo i, j tales que $p_{ij} > t$. El siguiente lema muestra que una solución extrema de este PL pocas variables no nulas.

Lema 1.8. *Todo punto extremo de* $[LP(t)]$ *posee a lo más* $m + n$ *variables no nulas.*

Demostración. Sea v el número de variables de [LP(t)]. Todo punto extremo del polítopo que define [LP(t)] está determinado por v restricciones linealmente independientes que se satisfacen con igualdad, y por lo tanto al menos $v - (m + n)$ variables serán nulas en el punto extremo. □

Dado un punto extremo x de [LP(t)], definamos el grafo $G(x) = (J \cup M, E(x))$, donde $J = \{1, \ldots, n\}$ y $M = \{1, \ldots, m\}$ representan los trabajos y máquinas respectivamente, y $E(x) = \{ij : x_{ij} > 0\}$ el conjunto de aristas. Este grafo es bipartito, puesto que solo pueden haber arcos entre un nodo $j \in J$ y otro $i \in M$. Además, el Lema 1.8 nos dice que $|E(x)| \leq m + n = |J \cup M|$, es decir, el número de aristas en el grafo es a lo más el número de vértices. A continuación probaremos que toda componente conexa de $G(x)$ también satisface esta propiedad, y en consecuencia $G(x)$ es un *pseudo-bosque*.

Definición 1.9. Decimos que un grafo conexo $G = (V, E)$ es un *pseudo-árbol* si $|E| \leq |V|$. Un *pseudo-bosque* es un grafo tal que cada una de sus componentes conexas es un pseudo-árbol.

Es decir, un pseudo-árbol es un árbol con a lo más una arista extra, la que podría generar un ciclo.

Lema 1.10. *Para todo punto extremo x el grafo $G(x)$ es un pseudo-bosque.*

Demostración. Sea C una componente conexa de $G(x)$. Denotemos por J_C y M_C los trabajos y máquinas en C respectivamente, x_C la restricción de x sobre aquellos trabajos en J_C y máquinas en M_C, y $x_{\bar{C}}$ el resto de las variables de x. Reordenando las componentes supongamos que $x = (x_C, x_{\bar{C}})$. También denotemos por P_C a la submatriz de P donde las filas son M_C y las columnas J_C.

Primero probemos que x_C es punto extremo de $[\mathrm{LP}(P_C, t)]$. Supongamos que no, es decir, existen x_1, x_2 soluciones factibles de $[\mathrm{LP}(P_C, t)]$ tales que $x_C = \frac{1}{2}(x_1 + x_2)$. Pero entonces $x = \frac{1}{2}((x_1, x_{\bar{C}}) + (x_2, x_{\bar{C}}))$ donde $(x_1, x_{\bar{C}}), (x_2, x_{\bar{C}})$ son soluciones factibles para $[\mathrm{LP}(t)]$, lo que contradice que x sea punto extremo. Luego, como x_C es punto extremo de $[\mathrm{LP}(P_C, t)]$, gracias al Lema 1.8 sabemos que el número de aristas en $G(x_C)$ es a lo más el número de vértices y como C es conexo concluimos que $G(x_C)$ es un pseudo-árbol. En consecuencia $G(x)$ un pseudo-bosque. □

Ahora ya se tienen las herramientas para poder demostrar el Teorema 1.7.

Demostración Teorema 1.7. Sea x un punto extremo de $[\mathrm{LP}(t)]$, $E_1 = \{ij \in E(x) : x_{ij} = 1\}$ y $J_1 = \{j \in J : \text{existe } i \in M \text{ tal que } x_{ij} = 1\}$. Notemos que los arcos en E_1 corresponden a aquellos trabajos que son asignados de manera íntegra a alguna máquina, es decir, tienen grado 1 en el grafo $G(x)$. Hacemos $\bar{x}_{ij} = 1$ para todo $ij \in E_1$ y el grafo $G'(x) = (J \setminus J_1 \cup M, E \setminus E_1)$ obtenido al borrar estos trabajos y los arcos respectivos sigue siendo un pseudo-bosque. Si una componente conexa de $G'(x)$ es un árbol, lo enraizamos en cualquier vértice, y cada trabajo en el árbol lo emparejamos con alguna de sus máquinas hijas en el árbol. Notemos que esto siempre es posible, pues el grado de todo trabajo en $G'(x)$ es mayor o igual a 2, y como cada vértice tiene a lo más un padre ninguna máquina del árbol será emparejada con más de un trabajo. En el caso que una componente conexa de $G'(x)$ posea un ciclo, emparejamos los trabajos del ciclo tomando aristas en el ciclo de manera alternada. Esto es posible pues $G'(x)$ es bipartito, y entonces el ciclo es de largo par. Si los arcos del ciclo son retirados obtenemos una colección de árboles para los cuales hacemos el mismo procedimiento descrito anteriormente para árboles, considerándolos enraizados en los vértices del ciclo. Si denotamos por E_2 al emparejamiento obtenido en $G'(x)$, hacemos $\bar{x}_{ij} = 1$ para todo $ij \in E_2$ y $\bar{x}_{ij} = 0$ para $ij \in E(x) \setminus (E_1 \cup E_2)$. Notemos que en $\bar{x}$ cada trabajo es asignado a exactamente una máquina, y por lo tanto

$$\sum_{i \in M_j(t)} \bar{x}_{ij} = 1$$

para todo $j \in J$. En consecuencia $\bar{x}$ es una solución factible de $[\mathrm{IP}(t)]$, pues cada trabajo en $\bar{x}$ es asignado a alguna máquina a la cual era procesado fraccionariamente en x. Además,

en E_2 cada máquina ha sido emparejada con a lo más un trabajo, y por lo tanto se tendrá que para todo $i \in M$

$$\sum_{j \in J_i(t)} p_{ij}\bar{x}_{ij} \leq \sum_{j \in J_i(t)} p_{ij}x_{ij} + t \leq T + t.$$

□

Volviendo al problema, notemos que si asignamos cada trabajo j a la máquina tal que p_{ij} es mínimo y denotamos el makespan de esta programación como α, este valor corresponderá a una cota superior en el makespan óptimo T^*. Por otro lado, α/m corresponde a una cota inferior en el makespan óptimo, y por lo tanto $T^* \in [\alpha/m, \alpha]$. El makespan de cualquier programación es un número entero, y por lo tanto adivinaremos el valor T^* realizando una búsqueda binaria en el intervalo $[\alpha/m, \alpha]$.

Algorithm 2 Programación de máquinas no relacionadas

1: Mediante una búsqueda binaria en $[\alpha/m, \alpha]$, se encuentra $T_A := \text{mín}\{T \in [\alpha/m, \alpha] : [\text{LP}(T)] \text{ es factible }\}$.
2: Se busca un punto extremo x de $[\text{LP}(T_A)]$.
3: Se construye el grafo $G(x)$ y se asigna los trabajos a máquinas según $\bar{x}$.

Teorema 1.11. *El algoritmo anterior es una 2-aproximación para* $R||C_{\text{máx}}$.

Demostración. Claramente $T_A \leq T^*$, pues $LP(T^*)$ es factible. Sea x el punto extremo de $LP(T_A)$ encontrado y $\bar{x}$ la asignación de trabajos construida a partir de x. Gracias al Teorema 1.7 se tiene que para todo $i \in M$, $\sum_{j \in J_i(T_A)} p_{ij}\bar{x}_{ij} \leq T_A + T_A = 2T_A \leq 2T^*$. □

El análisis del algoritmo anterior no puede ser mejorado para obtener un factor de aproximación menor a 2. Para ver esto, basta encontrar una colección de instancias tales que al redondear un punto extremo se obtenga una solución con costo creciente a 2 veces el valor óptimo. Por ejemplo, consideremos una instancia con m máquinas y $m^2 - m + 1$ trabajos. El primer trabajo posee tiempo de procesamiento $p_{i1} = m$ para toda máquina i, y el resto de los trabajos posee tiempo de procesamiento igual a 1 en todas las máquinas. Nótese que la carga total sobre las máquinas en cualquier programación será $m \cdot 1 + (m^2 - m) = m^2$, y por lo tanto el *makespan* está acotado inferiormente por m. Si se asigna el primer trabajo completamente en alguna de las máquinas, y en cada una de las $m - 1$ máquinas se asignan m de los trabajos unitarios, el makespan será exactamente m y por lo tanto esta programación es óptima. Para $T = m$, supongamos que el punto extremo encontrado en $[\text{LP}(m)]$ es el que asigna una unidad del trabajo 1 en cada máquina, y $m - 1$ trabajos unitarios completos en cada una de las máquinas. Al construir la asignación $\bar{x}$ a partir de este punto extremo se obtiene una solución que asigna completamente el trabajo 1 en alguna de las máquinas, y además asigna $m - 1$ trabajos unitarios en esa máquina, y por lo tanto el makespan será $2m - 1$. Luego, la razón entre el makespan de la programación que retorna el algoritmo y el óptimo es $(2m - 1)/m \nearrow 2$.

A pesar de que este algoritmo no permite obtener un factor de aproximación menor a 2, podría existir otro algoritmo con factor de aproximación menor basado en la relajación $[\text{LP}(t)]$. Esta idea motiva la definición de gap de integralidad para una relajación lineal.

Definición 1.12 (Gap de integralidad)**.** Dado un programa entero [IP] y su relajación [LP], el gap de integralidad corresponde a $\rho = \sup_I \frac{\text{opt}_{[\text{IP}]}(I)}{\text{opt}_{[\text{LP}]}(I)}$, donde I son las instancias del problema.

En el caso de $R||C_{\text{máx}}$, el gap de integralidad es menor o igual a 2 para la relajación [LP(t)], pues al algoritmo que se mostró anteriormente toma un punto extremo de la relajación lineal y construye un solución del problema entero cuyo costo es a lo más el doble.

1.3. Programando tareas con división del trabajo y tiempos de instalación

Durante el presente trabajo se estudia el problema de programar tareas en un entorno de máquinas paralelas, pero bajo el supuesto de que los trabajos pueden ser divididos en distintas partes, cada una de las cuales puede ser procesada en distintas máquinas. En la notación introducida en la Sección 1.1.1, este ambiente de trabajos corresponde a *split*. A diferencia del ambiente *pmtn*, diferentes partes de un mismo trabajo pueden ser procesadas de manera simultánea en máquinas distintas. Además, antes de procesar cada parte de un trabajo en alguna máquina, existe un tiempo en que la máquina debe ser preparada para procesar el nuevo trabajo, llamado *tiempo de instalación*.

Más específicamente, y usando la notación de tres campos, en esta tesis se estudian los problemas $P|s_j, \text{setup}|\sum w_j C_j$ y $R|\text{split,setup}|C_{\max}$. En el primer problema el tiempo instalación sólo depende del trabajo, mientras que en el segundo problema el tiempo de instalación depende del trabajo y de la máquina en la cual se procese. En este ambiente, el tiempo de completación C_j del trabajo j corresponde al punto en el tiempo en que se termina de procesar la última parte del trabajo.

Problemas con tiempos de instalación aparecen en diversas aplicaciones en planificación de producción, por ejemplo, en la industria textil [24] y en la construcción de componentes semiconductores [15] usados en telecomunicación. También se ha estudiado este ambiente de trabajos en problemas de planificación de recursos en caso de desastre [28]. Para tener una visión más detallada sobre artículos vinculados a programación de tareas con tiempos de instalación ver el survey de Allahverdi et al. [1].

1.4. Estructura de los capítulos y contribuciones de este trabajo

En el Capítulo 2 se estudia el problema $P|s_j, \text{setup}|\sum w_j C_j$. El resultado principal de esta sección es la existencia de una $(2+\varepsilon)$-aproximación cuando los tiempos de instalación son todos iguales, que corresponde al primer algoritmo de aproximación de factor constante para este problema. La idea tras el algoritmo es dividir cada trabajo j en partes de tamaño $2s_j$, donde s_j es el tiempo de instalación para este trabajo. Luego, cada trabajo j ahora se ve como un grupo L_j de trabajos, y entonces se estudia este nuevo problema.

Se demuestra que al hacer esta transformación en la instancia los tiempos de completación crecen en un factor de 2, y por lo tanto un PTAS para el problema de programar grupos entrega la $(2+\varepsilon)$-aproximación. Usando técnicas similares a las usadas para este algoritmo, se puede diseñar una 2-aproximación para el problema de minimizar $\sum C_j$, que en particular mejora el factor obtenido por Schalekamp et al. [22]. Finalmente, usando un algoritmo de *programación en lista*, se obtiene una 4-aproximación para el problema original con tiempos de instalación dependientes del trabajo.

En el Capítulo 3 se estudia el problema $R|\text{split,setup}|C_{\text{máx}}$. Los algoritmos diseñados en esta sección están basados en programación lineal, donde x_{ij} representa la fracción del trabajo j que es asignado en la máquina i. La primera relajación que se estudia es

$$[\text{LST}]: \quad \sum_{i\in M} x_{ij} = 1 \qquad \text{para todo } j \in J, \tag{1.7}$$

$$\sum_{j\in J} x_{ij}(p_{ij}+s_{ij}) \leq C^* \qquad \text{para todo } i \in M, \tag{1.8}$$

$$x_{ij} = 0 \qquad i \in M, j \in J : s_{ij} > C^*, \tag{1.9}$$

$$x_{ij} \geq 0 \qquad \text{para todo } i \in M, j \in J. \tag{1.10}$$

a partir de la cual se diseña una 3-aproximación para el problema. Nótese que la desigualdad 1.8 se puede ver como la relajación lineal de

$$\sum_{j\in J}(x_{ij}p_{ij} + y_{ij}s_{ij}) \leq C^* \qquad \text{para todo } i \in M, \tag{1.11}$$

$$x_{ij} \leq y_{ij} \qquad \text{para todo } i \in M \text{ y } j \in J, \tag{1.12}$$

donde y_{ij} es una variable binaria que indica si se procesa o no el trabajo j en la máquina i. Al realizar un paso de *lift and project* sobre la desigualdad 1.11 es posible fortalecer la relajación anterior, y como consecuencia es posible diseñar una $(1+\phi)$-aproximación para el problema, donde $\phi = \frac{\sqrt{5}+1}{2}$. Respecto a la inaproximabilidad del problema, se mejora la cota inferior de 3/2 heredada de $R||C_{\text{máx}}$, a la constante $\frac{e}{e-1}$, basada en un resultado de Feige [8] sobre *Max-k-Cover*. Usando la relajación lineal fuerte se muestra una 2-aproximación para la versión del problema en que cada trabajo posee un conjunto restringido de máquinas en las que puede ser procesado, teniendo igual tiempo de instalación y procesamiento en todas ellas. Finalmente, se estudia las relajaciones basadas en *configuraciones* sobre máquinas y trabajos, es decir, aquí las variables corresponden a vectores que representan el estado de un trabajo o de una máquina en cierta programación. El PL de configuraciones de trabajos posee una cantidad infinita de variables, sin embargo, se demuestra que es posible restringirse a una cantidad finita de ellas y que además es posible resolver este PL en tiempo polinomial a un factor de $1+\varepsilon$. Determinar el gap de integralidad de esta relajación queda como una pregunta abierta de este trabajo.

Capítulo 2

Algoritmos de aproximación para $P|s_j, \textbf{split}|\sum w_jC_j$

2.1. Descripción del problema y trabajo relacionado

El problema estudiado en esta sección corresponde a programar trabajos que pueden ser divididos y en donde cada parte puede ser procesada independientemente y de forma simultánea en distintas máquinas. Para cada trabajo $j \in J$, el tiempo de procesamiento se denota por $p_j \in \mathbb{Z}_+$ y su peso en la programación es $w_j \in \mathbb{Z}_+$. Además, antes de procesar una parte de algún trabajo, las máquinas deben correr un tiempo de instalación $s_j \in \mathbb{Z}_+$, en el cual no pueden procesar ningún otro trabajo. El tiempo de instalación puede depender del trabajo que se vaya a procesar. Las máquinas serán paralelas e idénticas. Lo que se busca es minimizar la suma ponderada de los tiempos de completación de los trabajos. Usando la notación de tres campos se representa este problema como $P|s_j, \text{split}|\sum w_jC_j$. En Schalekamp et al. [22] se probó que $P|s_j, \text{split}|\sum w_jC_j$ es fuertemente $\mathcal{NP}$-duro incluso cuando $s_j = s$ para todo $j \in J$. En ausencia de tiempos de instalación, este problema se puede resolver de manera óptima en tiempo polinomial dividiendo los trabajos de manera equitativa en cada máquina y procesando las partes en cada máquina de acuerdo a la *regla de Smith*. Esta regla consiste en ordenar los trabajos de acuerdo a p_j/w_j creciente.

La literatura en algoritmos de aproximación para este problema es pequeña. Potts y Wassenhove [21] realizaron una revisión sobre algoritmos y complejidad computacional para distintos problemas de programación de tareas que involucra una decisión sobre si dividir o no cierto trabajo o grupo de trabajos, considerando también tiempos de instalación y diversas funciones objetivo. Sin embargo, en su trabajo queda abierta la complejidad computacional del problema estudiado en este capítulo. Más tarde, Xing y Zhang [32] mostraron que ciertos problemas que permitían dividir trabajos en ausencia de tiempos de instalación se pueden resolver en tiempo polinomial. Recientemente, Schalekamp et al. [22] mostraron una 2.781-aproximación para el problema $P|s, \text{split}|\sum C_j$ y un PTAS para $Pm|s, \text{split}|\sum w_jC_j$, que se constituyen en los primeros de su tipo para estos problemas. La complejidad del problema $P|s, \text{split}|\sum C_j$ permanece como una pregunta abierta.

2.2. Contribución de este trabajo

En la Sección 2.4 se desarrolla una $(2+\varepsilon)$-aproximación para $P|s_j, \text{split}|\sum w_j C_j$ cuando $s_j = s$ para todo trabajo j. Este esquema constituye en el primer resultado de existencia de un algoritmo de aproximación para este problema. Cuando los pesos de todos los trabajos son iguales, se diseña una 2-aproximación. Este resultado mejora la 2.781-aproximación de Schalekamp et al. [22]. Ambos resultados se basan en el estudio de un problema de programación de grupos de trabajos, el cual se estudia en la Sección 2.3. En esa sección se estudia la complejidad computacional del problema, y se logra probar la equivalencia con un problema de programación de tareas en una máquina. Usando un resultado probado por Megow y Verschae [19] se prueba la existencia de un PTAS en el caso con pesos, y se prueba que puede ser resuelto de manera óptima en tiempo polinomial cuando todos los grupos pesan lo mismo. Para este último se hace uso de una conocida regla de programación en una máquina, *Shortest Processing Time First* (SPT). En el caso con pesos, también se prueba que la regla de Smith es una 2-aproximación, la cual tiene como corolario una 4-aproximación para el problema $P|s_j, \text{split}|\sum w_j C_j$.

2.3. Programando órdenes de trabajos

Antes de tratar propiamente el problema $P|s_j, \text{split}|\sum w_j C_j$, estudiaremos otro problema de problema de programación, para el cual obtendremos resultados que nos ayudarán a diseñar los algoritmos de aproximación para el primer problema. Se tiene un conjunto de trabajos J y un conjunto $O \subseteq \mathcal{P}(J)$ de grupos de trabajos, tales que $L \cap L' = \emptyset$ para todo $L, L' \in O$. Dentro de cada grupo L, todo trabajo $j \in L$ posee el mismo tiempo de procesamiento $q_j = q_L$. Cada trabajo debe ser asignado a alguna máquina y cuando comienza a procesarse no puede ser interrumpido. Cada grupo L además tiene asociado un peso w_L, y $C_L = \max\{C_j : j \in L\}$ corresponde al tiempo de completación del grupo, es decir, el tiempo en que se completa el último trabajo del grupo L. Lo que se busca es encontrar la programación que minimice $\sum_{L \in O} w_L C_L$. Denotaremos este problema como $P|\text{part}, q_j = q_L|\sum w_L C_L$. En este problema no hay tiempos de instalación involucrados.

Observación 2.1. Notemos que los trabajos de un mismo grupo que son procesados dentro de una misma máquina no afectan el tiempo de completación del grupo cuando son permutados, de hecho, desde el punto de vista de tiempos de completación y costo de la programación el permutar trabajos de un mismo grupo inclusive en distintas máquinas no genera ningún cambio. Esto es porque todos los trabajos dentro de un grupo tienen el mismo tiempo de procesamiento. Si esto no fuese asi, la observación no sería cierta.

Definición 2.2 (Programación en Lista)**.** Diremos que una programación corresponde a una *programación en lista* si se tiene un orden sobre los trabajos y los trabajos se van procesando en la máquina con menos carga de acuerdo a este orden.

Consideremos ahora un caso particular de este problema en el cual $q_L = q_{L'}$ para todo $L, L' \in O$, es decir, todos los trabajos involucrados poseen el mismo tiempo de procesamiento q. A esta versión restringida la denotaremos por $P|\text{part}, q_j = q|\sum w_L C_L$. El siguiente lema nos permite probar que siempre existe una programación en lista que es óptima para el problema.

Lema 2.3. *Sea S una programación con tiempos de completación $C_{L_1} \leq C_{L_2} \leq \cdots \leq C_{L_{|O|}}$. Consideremos la programación en lista dada por el orden $L_1 \leq L_2 \leq \cdots L_{|O|}$, y denotemos por C'_{L_j} el tiempo de completación del grupo L_j en la programación en lista. Entonces $C'_{L_j} \leq C_{L_j}$ para todo grupo $L_j \in O$.*

Demostración. Dado que todos los trabajos poseen tiempo de procesamiento q, el tiempo de completación del grupo L_j en la programación en lista es $C'_{L_j} = \left\lceil \frac{q}{m} \sum_{k \leq j} |L_k| \right\rceil$. Por otro lado, en la programación S se han completado todos los grupos $L_1, \ldots, L_j$ hasta el tiempo C_{L_j}, y por lo tanto, $mC_j \geq q \sum_{k \leq j} |L_k|$. Luego, $C_{L_j} \geq \frac{q}{m} \sum_{k \leq j} |L_k|$, pero como $C_{L_j} \in \mathbb{Z}_+$ concluimos que $C_{L_j} \geq \left\lceil \frac{q}{m} \sum_{k \leq j} |L_k| \right\rceil = C'_{L_j}$. □

Al tomar una programación óptima S^* y reprocesar los grupos de acuerdo a los tiempos de completación en S^*, el lema anterior nos dice que los nuevos tiempos de completación no crecen, y por lo tanto la programación en lista será también óptima.

Consideremos el problema en una máquina $1||\sum w_j \lceil C_j/\ell \rceil$. Es decir, se tiene un conjunto J de n trabajos, en donde cada uno posee un peso w_j y un tiempo de procesamiento p_j. Estos trabajos deben programarse en una máquina, sin interrupciones y lo que se busca es minimizar $\sum_{j \in J} w_L \lceil C_j/\ell \rceil$, donde C_j corresponde al tiempo de completación del trabajo j. Probaremos que este problema es equivalente a $P|\text{part}, q_j = q|\sum w_L C_L$.

Lema 2.4. *Dada una instancia I_p para el problema $P|part, q_j = q|\sum w_L C_L$, existe una instancia I_s para el problema en una máquina $1||\sum w_j \lceil C_j/\ell \rceil$ tal que $opt(I_p) = opt(I_s)$. Recíprocamente, dada una instancia I'_s de $1||\sum w_j \lceil C_j/\ell \rceil$, existe una instancia I'_p de $P|part, q_j = q|\sum w_L C_L$ tal que $opt(I'_s) = opt(I'_p)$.*

Demostración. Dada una instancia I_p para el primer problema, construimos una instancia I_s para el problema en una máquina como sigue: tenemos un conjunto O de trabajos, el tiempo de procesamiento de un trabajo L es $p_L = q|L|$, el peso del trabajo L es w_L, y $\ell = m$ corresponde al número de máquinas. Sabemos que existe una programación óptima para el problema en máquinas paralelas que corresponde a una programación en lista. Sea σ el orden sobre los grupos en esta programación. Sabemos que costo en este caso corresponde

$$\sum_{L \in O} w_L \left\lceil \frac{q}{m} \sum_{K:\sigma(K) \leq \sigma(L)} |K| \right\rceil .$$

Por otra parte, el tiempo de completación de un trabajo L al procesarlos de acuerdo al orden σ es $C_L = \sum_{K:\sigma(K) \leq \sigma(L)} p_K$. Luego, el costo de la programación en el problema en una máquina al considerar el orden σ sobre los trabajos corresponde a

$$\sum_{L \in O} w_L \left\lceil \frac{C_L}{m} \right\rceil = \sum_{L \in O} w_L \left\lceil \frac{1}{m} \sum_{K:\sigma(K) \leq \sigma(L)} p_K \right\rceil = \sum_{L \in O} w_L \left\lceil \frac{q}{m} \sum_{K:\sigma(K) \leq \sigma(L)} |K| \right\rceil ,$$

y por lo tanto $\text{opt}(I_p) \geq \text{opt}(I_s)$. Sin embargo, si existiese otro orden π tal que generase un costo menor en el problema en una máquina, ese orden induciría una programación en lista para el problema en máquinas paralelas con costo también menor, lo cual no puede ser

pues σ es óptimo. Luego, $\text{opt}(I_p) = \text{opt}(I_s)$. Recíprocamente, dada una instancia I'_s para el problema en una máquina, construimos una instancia para el problema en máquinas paralelas: se tiene un grupo por cada trabajo j, donde cada grupo posee p_j trabajos con tiempo de procesamiento $q = 1$ cada uno. El peso del grupo j es w_j y el número de máquinas paralelas es ℓ. Sea σ' el orden sobre los trabajos que minimiza el costo de la programación en una máquina. Luego, se tiene que

$$\sum_{j \in J} w_j \left\lceil \frac{C_j}{\ell} \right\rceil = \sum_{j \in J} w_j \left\lceil \frac{1}{\ell} \sum_{k:\sigma'(k) \leq \sigma'(j)} p_k \right\rceil .$$

El lema sigue por una argumento análogo al anterior. □

Claramente las construcciones hechas en el lema anterior se pueden realizar en tiempo polinomial. Luego, gracias a la equivalencia entre estos problemas podemos estudiar la complejidad computacional y desarrollar algoritmos para el problema en una máquina, los cuales también servirán para el problema en máquinas paralelas. El siguiente teorema nos dice que el problema en una máquina es $\mathcal{NP}$-duro, y por lo tanto el problema en máquinas paralelas también.

Teorema 2.5. *El problema* $1||\sum w_j\lceil C_j/m\rceil$ *es fuertemente* $\mathcal{NP}$*-duro.*

Demostración. Reduciremos desde el problema 3-Partition: dados $3n$ enteros positivos $a_1, \ldots, a_{3n}$ tales que $\sum_{j=1}^{3n} a_j = nB$ y $B/4 < a_j < B/2$ para todo $j \in \{1, \ldots, 3n\}$, decidir si existe alguna 3-partición, es decir, $A_1, A_2, \ldots, A_n$ de $\{a_1, a_2, \ldots, a_{3n}\}$ tal que $|A_i| = 3$ y $\sum_{j \in A_i} a_j = B$ para todo $i \in \{1, \ldots, n\}$. Este problema es fuertemente $\mathcal{NP}$-duro [10]. Dada una instancia de 3-Partition, construimos una instancia para el problema en una máquina: tenemos $\ell = B$ máquinas y $3n$ trabajos. Además los tiempos de procesamiento y pesos son $p_j = w_j = a_j$ para todo $j \in \{1, \ldots, 3n\}$.

Supongamos que existe una partición $A_1, A_2, \ldots, A_n$ que es una 3-partición. Consideremos una programación dada por procesar cada trabajo en un conjunto A_i de manera consecutiva, para todo i, y bajo alguna permutación de esta partición. Denotemos por $\sigma_i : A_i \to \{1, 2, 3\}$ el orden dentro de cada conjunto A_i en que se procesan sus tres

trabajos. Tenemos que el costo de la programación es:

$$\begin{aligned}
\sum_j w_j \left\lceil \frac{C_j}{m} \right\rceil &= \sum_{i=1}^{n} \sum_{j \in A_i} a_j \left\lceil \frac{C_j}{B} \right\rceil \\
&= \sum_{i=1}^{n} \sum_{j \in A_i} a_j \left\lceil \frac{1}{B} \left((i-1)B + \sum_{k:\sigma_i(k) \leq \sigma_i(j)} a_k \right) \right\rceil \\
&= \sum_{i=1}^{n} \sum_{j \in A_i} a_j \left\lceil i - 1 + \frac{1}{B} \sum_{k:\sigma_i(k) \leq \sigma_i(j)} a_k \right\rceil \\
&= \sum_{i=1}^{n} i \sum_{j \in A_i} a_j \\
&= B \sum_{i=1}^{n} i \\
&= \frac{B}{2} n(n+1) := \text{opt.}
\end{aligned}$$

Esto implica que el costo de la programación será independiente de la 3-partición que se considere. Nótese que $\sum_{k:\sigma_i(k) \leq \sigma_i(j)} a_k \leq B$ para todo i y para todo $k \in A_i$, y es por ello que se tiene la cuarta igualdad. Consideremos ahora las funciones $H_i : \mathbb{R}_+ \to \{0, 1\}$ para $i \in \{1, \ldots, n\}$, tales que

$$H_i(t) = \begin{cases} 1 & \text{si } t > (i-1)B, \\ 0 & \text{si } t \leq (i-1)B. \end{cases}$$

Luego, se tiene:

$$\begin{aligned}
\sum_j w_j \left\lceil \frac{C_j}{\ell} \right\rceil &= \sum_j a_j \left\lceil \frac{C_j}{B} \right\rceil \\
&= \sum_j a_j \sum_{i=1}^{n} H_i(C_j) \\
&= \sum_{i=1}^{n} \sum_j a_j H_i(C_j) \\
&= \sum_{i=1}^{n} \sum_{j:C_j > (i-1)B} a_j \\
&\geq \sum_{i=1}^{n} (n-i+1)B \\
&= \frac{B}{2} n(n+1).
\end{aligned}$$

La desigualdad se debe a que aquellos trabajos que finalizan en un tiempo posterior a $(i-1)B$ deben usar completamente la máquina a partir de ese tiempo, y por lo tanto la carga total de ellos debe ser al menos $nB - (i-1)B = (n-i+1)B$. La igualdad se

alcanza solamente cuando algún trabajo comienza a procesarse en cada tiempo $(i-1)B$ para $i \in \{1, \ldots, n\}$, lo cual ocurre si y solo si existe una 3-partición. Luego, si no existe una 3-partición cualquier programación tendrá costo mayor a opt. □

A pesar de que el problema es fuertemente $\mathcal{NP}$-duro, es posible encontrar un esquema de aproximación a tiempo polinomial. De hecho, cuando todos los trabajos tengan el mismo peso será posible resolver el problema de manera óptima en tiempo polinomial programando los trabajos ordenados de menor a mayor tiempo de procesamiento, es decir, $p_1 \leq p_2 \leq \cdots \leq p_n$. A este orden lo denotaremos por SPT *(Shortest Processing Time First)*.

Lema 2.6. *Existe un PTAS para el problema* $1||\sum w_j\lceil C_j/\ell\rceil$. *Cuando* $w_j = w_k$ *para todo* j, k, *entonces se puede resolver de manera óptima en tiempo polinomial al procesar los trabajos según el orden SPT.*

Demostración. Sea $f : \mathbb{R}_+ \to \mathbb{R}$ tal que $f(t) = \lceil t/m\rceil$. Esta función es positiva y no decreciente, y por lo tanto existe un PTAS para $1||\sum w_j f(C_j)$. Esto fue probado por Megow y Verschae [19]. Ahora supongamos que todos los pesos son iguales. Probaremos que procesar los trabajos según el orden dado por SPT es óptimo. Supongamos que S es una programación óptima, la cual viene dada por procesar los trabajos en un orden π distinto de SPT. En este orden deben haber dos trabajos consecutivos, digamos j y k, tales que $p_j > p_k$ y $j <_\pi k$. Supongamos que el trabajo j comienza a procesarse en el tiempo t, y consideremos la programación dada al permutar j y k. Llamemos π' a la nueva programación. Es importante notar que el tiempo de completación del resto de los trabajos se mantiene igual, pues el tiempo total usado por j y k se mantiene invariante en posición y duración. Bajo la programación π' se tiene que j comienza en el tiempo $t + p_k$, y k comienza en el tiempo t. Luego, se tiene que:

$$\begin{aligned}
\sum_j \left\lceil \frac{C_j^\pi}{\ell} \right\rceil - \sum_j \left\lceil \frac{C_j^{\pi'}}{\ell} \right\rceil &= \left\lceil \frac{C_j^\pi}{\ell} \right\rceil + \left\lceil \frac{C_k^\pi}{\ell} \right\rceil - \left(\left\lceil \frac{C_j^{\pi'}}{\ell} \right\rceil + \left\lceil \frac{C_k^{\pi'}}{\ell} \right\rceil \right) \\
&= \left\lceil \frac{t+p_j}{\ell} \right\rceil + \left\lceil \frac{t+p_j+p_k}{\ell} \right\rceil - \left\lceil \frac{t+p_k+p_j}{\ell} \right\rceil - \left\lceil \frac{t+p_k}{\ell} \right\rceil \\
&= \left\lceil \frac{t+p_j}{\ell} \right\rceil - \left\lceil \frac{t+p_k}{\ell} \right\rceil \\
&\leq 0.
\end{aligned}$$

Al iterar este proceso se obtiene la programación según orden SPT, y en cada iteración el costo de la programación no aumenta. Luego, se concluye que SPT es óptimo. □

Corolario 2.7. *Existe un PTAS para el problema* $P|part, q_j = q|\sum w_L C_L$. *Cuando los pesos de cada grupo son iguales, entonces la programación en lista dada por SPT es óptima.*

Demostración. Sea $\{A_\varepsilon\}_{\varepsilon>0}$ el PTAS para el problema $1||\sum w_j\lceil C_j/\ell\rceil$, y denotemos por $\{B_\varepsilon\}_{\varepsilon>0}$ la siguiente colección de algoritmos:

Algorithm 3 Algoritmo $B_\varepsilon(I_p)$

1: Transforma I_p en una instancia I_s como se hizo en el Lema 2.4.
2: Corre A_ε en la instancia I_s. Sea σ_ε el orden en que se programan los trabajos.
3: Se ejecuta una programación en lista para I_p dada por el orden σ_ε.

Gracias a lo probado en el Lema 2.4 se tiene que

$$\text{costo}(B_\varepsilon(I_p)) = \text{costo}(A_\varepsilon(I_s)) \leq (1+\varepsilon)\text{opt}(I_s) = (1+\varepsilon)\text{opt}(I_p),$$

y por lo tanto $\{B_\varepsilon\}_{\varepsilon>0}$ es un PTAS para $P|\text{part}, q_j = q|\sum w_L C_L$. De manera análoga, en presencia de pesos iguales para todos los grupos, sea B_{SPT} el algoritmo que entrega una programación en lista para I_p. Luego, se tiene que:

$$\text{costo}(B_{\text{SPT}}(I_p)) = \text{costo}(\text{SPT}(I_s)) = \text{opt}(I_s) = \text{opt}(I_p).$$

□

Nótese que el problema en una máquina es fuertemente $\mathcal{NP}$-duro, y por lo tanto la existencia de un FPTAS no es posible salvo que $\mathcal{P} = \mathcal{NP}$. En ese sentido, en esta sección hemos podido estudiar de manera completa y bastante cerrada el problema $P|\text{part}, q_j = q|\sum w_L C_L$. En cuanto al problema más general, $P|\text{part}, q_j = q_L|\sum w_L C_L$, mostraremos que un algoritmo de programación en lista es una 2-aproximación. El orden usado para la programación procesa los trabajos de manera que

$$\frac{q_1|L_1|}{w_1} \leq \cdots \leq \frac{q_n|L_n|}{w_n}.$$

Esta regla de procesamiento se conoce como *regla de Smith*. Al orden lo denotamos como WSPT *(Weighted Shortest Processing time First)*.

Teorema 2.8. *El algoritmo de programación en lista dado por el orden WSPT es una 2-aproximación para $P|part, q_j = q_L|\sum w_L C_L$.*

Demostración. Consideremos una máquina, la cual posee una velocidad de procesamiento constante igual a m. Si procesamos los grupos en esta máquina, veremos cada grupo L_j como un trabajo j con tiempo de procesamiento $p_j = q_j|L_j|$, y cuando el objetivo es minimizar $\sum w_j C_j$ se sabe que procesar los trabajos según la regla de Smith es óptimo [26]. Consideremos entonces los trabajos ordenados según WSPT y llamemos $C_{L_j}^{\text{WSPT}}$ al tiempo de completación del grupo L_j en la programación en lista dada por WSPT, y llamemos C_j^{WSPT} al tiempo de completación del trabajo j en la máquina de velocidad m según la regla de Smith. Luego, se tiene que:

$$C_{L_j}^{\text{WSPT}} \leq C_j^{\text{WSPT}} + q_j = \frac{1}{m}\sum_{k\leq j} p_k + q_j = \frac{1}{m}\sum_{k\leq j} q_j|L_j| + q_j \leq C_{L_j}^{\sigma} + C_{L_j}^{\sigma} = 2C_{L_j}^{\sigma}.$$

La última desigualdad es debido a que $\frac{1}{m}\sum_{k\leq j} q_j|L_j|$ y q_j son cotas inferiores en el tiempo de completación del grupo j en la programación en lista σ óptima. □

Observación 2.9. Respecto al algoritmo de programación en lista según Smith, el mejor factor de aproximación posible es exactamente 2, y en ese sentido nuestro algoritmo es óptimo para este factor. Para ello, consideremos la instancia I_m donde se tienen m máquinas, un grupo L_1 de un trabajo tal que $q_1 = 1$, $w_1 = 2$ y otro grupo L_2 de m trabajos tales que $q_2 = 1$, $w_2 = m$. Lo óptimo para esta instancia es programar primero los m trabajos de L_2, uno en cada máquina, y a continuación el único trabajo de L_1 en alguna de las máquinas. El costo de esta programación es $\text{opt}(I_m) = 2 \cdot 2 + m \cdot 1 = m + 4$. Sin embargo, el orden WSPT de estos trabajos es primero L_1 y luego L_2, pues $\frac{1}{2} = \frac{q_1|L_1|}{w_1} < \frac{q_2|L_2|}{w_2} = \frac{m}{m} = 1$. El costo de la programación según Smith es $1 \cdot 2 + 2 \cdot m = 2m + 2$, y por lo tanto

$$\frac{\sum w_{L_j} C_{L_j}^{\text{WSPT}}(I_m)}{\text{opt}(I_m)} = \frac{2m+2}{m+4} \nearrow 2.$$

2.4. Programación de tareas con tiempos de instalación

Ahora volvamos al problema original $P|s_j, \text{split}|\sum w_j C_j$. La idea es hacer uso de los resultados que tenemos para el problema de programar grupos, usando esos algoritmos como subrutinas en los algoritmos diseñados en esta sección. Al igual que en la sección anterior, primero tenemos un lema de tipo estructural que nos permitirá optimizar sin pérdida de generalidad sobre un conjunto de programaciones con ciertas características especiales.

Lema 2.10. *Existe una programación óptima donde el orden en que las partes de distintos trabajos son procesados no depende de la máquina.*

Demostración. Consideremos una programación óptima S con tiempos de completación $C_1 \leq C_2 \leq \cdots \leq C_n$. Consideremos los trabajos ordenados de acuerdo a estos tiempos de completación, y reprocesemos las partes en cada máquina de acuerdo a este orden. Denotemos por C'_j los nuevos tiempos de completación y llamamos S' a esta nueva programación. Probaremos que $C'_j \leq C_j$ para todo $j \in J$. En efecto, sea q_{ij} el tiempo total que es procesado el trabajo j en la máquina i en la programación S, y sea $y_{ij} = s_j + q_{ij}$ si $q_{ij} > 0$ e $y_{ij} = 0$ en otro caso. Sea C_{ij} el tiempo en que finaliza el trabajo j en la máquina i. Fijemos un trabajo j y una máquina i, y sea $k = \text{argmax}\{C_{ir} : 1 \leq r \leq j\}$, es decir, k corresponde al trabajo que en la programación S termina a lo más en tiempo C_j y que en la máquina i es la última en procesarse. Luego,

$$C_j \geq C_k \geq C_{ik} \geq \sum_{r=1}^{j} y_{ir} = C'_{ij}.$$

La última igualdad es producto de la forma en que reprograman los trabajos en S', y las desigualdades son debido a la elección de k. Luego, se tiene que $C_j \geq C_{ij}$ para toda máquina i, y por lo tanto $C_j \geq C'_j$. Luego, la programación S' también es óptima.

□

El lema anterior posee algunos corolarios de gran utilidad. Dado que el orden en que se procesan los trabajos en cada máquina es independiente de la máquina, todas las partes de

un mismo trabajo se procesan de manera consecutiva, y por lo tanto podemos considerar sin pérdida de generalidad aquellas programaciones en que solo se procesa a lo más una parte de un trabajo en cada máquina. De esta forma, las programaciones para nuestro problema quedan determinadas por un par (X, σ) donde $X = (x_{ij})$ es una matriz que indica la fracción del trabajo j que se procesa en la máquina i, para todo par i, j, y un orden σ que dice el orden en que los trabajos se procesan en cada máquina.

El siguiente algoritmo se encarga de realizar el puente entre el problema estudiado en la sección anterior, y el problema con división de trabajos y tiempos de instalación. Denotemos las instancias I de $P|s_j, \text{split}|\sum w_j C_j$ como tuplas $(s_j, p_j, w_j)_{j\in J}$.

Algorithm 4 Algoritmo Constructor$((s_j, p_j, w_j)_{j\in J})$

1: Por cada trabajo j, sea L_j un grupo de tamaño $\lceil p_j/s_j \rceil$ donde cada trabajo posee tiempo de procesamiento $q_{L_j} = 2s_j$.
2: El peso de cada grupo L_j es $w_{L_j} = w_j$.
3: Se retorna la instancia $I_p = (L_j, q_{L_j}, w_{L_j})_{j\in J}$ instancia para el problema en máquinas paralelas $P|\text{part}, q_j = q_L|\sum w_L C_L$.

Luego, la instancia construida por el algoritmo Constructor corresponde a una instancia para el problema $P|\text{part}, q_j = q_L|\sum w_L C_L$. Veremos que al realizar esta transformación de la instancia perdemos a lo más un factor 2 en el valor de la programación óptima. Denotemos por $C^\sigma_{L_j}$ el tiempo de completación del grupo L_j en la programación en lista dada por un orden σ para la instancia I_p construida por el algoritmo Constructor, y denotemos por C^σ_j el tiempo de completación del trabajo j en la programación (X, σ) para la instancia original dada al reutilizar el espacio usado por el grupo j en la máquina i. En el siguiente lema se especifica mejor como hacer esto.

Observación 2.11. Si denotamos por M_j las máquinas que procesan j en una programación para I_p, se tiene $2s_j|M_j| \leq 2s_j\lceil p_j/s_j \rceil$ y entonces $|M_j| \leq \lceil p_j/s_j \rceil$. La desigualdad se debe a que en cada máquina $i \in M_j$ se procesa al menos un trabajo de tamaño $2s_j$, y por lo tanto $2s_j|M_j|$ es una cota inferior en el tiempo total de procesamiento del grupo.

Lema 2.12. *Para todo $j \in J$, se tiene que $C^\sigma_j \leq C^\sigma_{L_j} \leq 2C^\sigma_j$.*

Demostración. Sea y_{ij} el tiempo total que se procesa el trabajo j en la máquina i de acuerdo a la programación en lista dada por σ, en la instancia I_p. Sea $x_{ij} = (y_{ij} - s_j)/p_j$ para todo $i \in M_j$, y $x_{ij} = 0$ para el resto de las máquinas. Para la primera desigualdad basta verificar que (X, σ) definido de esta forma es una programación factible para el

problema con división de trabajos.

$$\begin{aligned}
\sum_{i\in M} x_{ij} &= \sum_{i\in M_j} \frac{y_{ij}-s_j}{p_j} \\
&= \frac{1}{p_j}\sum_{i\in M_j} y_{ij} - \frac{s_j}{p_j}|M_j| \\
&= \frac{2s_j}{p_j}\left\lceil \frac{p_j}{s_j}\right\rceil - \frac{s_j}{p_j}|M_j| \\
&\geq \frac{2s_j}{p_j}\left\lceil \frac{p_j}{s_j}\right\rceil - \frac{s_j}{p_j}\left\lceil \frac{p_j}{s_j}\right\rceil \\
&= \frac{s_j}{p_j}\left\lceil \frac{p_j}{s_j}\right\rceil \\
&\geq 1.
\end{aligned}$$

La primera desigualdad es producto de la Observación 2.11. Para probar la segunda desigualdad, veremos que al duplicar el tiempo disponible para cada trabajo j en cada máquina i es suficiente para procesar completamente cada grupo j:

$$\begin{aligned}
\sum_{i\in M_j} 2(p_j x_{ij} + s_j) &\geq 2p_j + 2s_j|M_j| \\
&= 2s_j\left(\frac{p_j}{s_j} + |M_j|\right) \\
&\geq 2s_j\left(\frac{p_j}{s_j} + 1\right) \\
&\geq 2s_j\lceil p_j/s_j\rceil \\
&= 2s_j|L_j|.
\end{aligned}$$

□

Observación 2.13. Nótese que al definir $x_{ij} = (y_{ij} - s_j)/p_j$, en el Lema 2.12 se probó que $\sum_{i\in M} x_{ij} \geq 1$, y entonces se podría estar sobreestimando el valor de x_{ij} en alguna máquina. Sin embargo, sabemos que esto da origen a una programación que es factible para el problema con trabajos divisibles simplemente escalando los valores x_{ij} vía el factor $p_j/(2s\lceil p_j/s\rceil - s|M_j|)$. Este valor se obtiene al imponer que $\sum_{i\in M_j} x_{ij} = 1$.

Consideremos ahora $s_j = s$ para todo j. Dado $\varepsilon > 0$, el siguiente algoritmo toma una instancia I_p construida por el algoritmo Constructor y corre el algoritmo B_ε. Sea σ_ε el orden en que fueron procesados los grupos de trabajos y y_{ij} tiempo total que el grupo j fue procesado en la máquina i. El algoritmo lo que hace a continuación es retornar una programación factible (X, σ_ε) para el problema $P|s, \text{split}|\sum w_j C_j$ como se detalla a continuación.

Algorithm 5 Algoritmo Programador(ε)

1: Se construye la instancia $I_p = \text{Constructor}(s, (p_j, w_j)_{j\in J})$.
2: Se ejecuta el algoritmo B_ε en la instancia I_p.
3: Sea σ_ε el orden en que fueron procesados los grupos de trabajos y y_{ij} tiempo total que el grupo L_j fue procesado en la máquina i.
4: Para todo $j \in J$, sea M_j las máquinas que procesan el grupo L_j en la programación en lista σ_ε.
5: Sea $x_{ij} = (y_{ij} - s)/p_j$ para todo $i \in M_j$, y $x_{ij} = 0$ para todo $i \notin M_j$.
6: Retornar (X, σ_ε).

Observación 2.14. Es importante notar que el algoritmo Programador(ε) corre en tiempo polinomial en el tamaño de la instancia original del problema con división de trabajos y tiempos de instalación. Para ello, nótese que el tiempo y_{ij} que un trabajo j es procesado en una máquina se puede calcular de la siguiente forma: sea $C_{0i} = 0$ para todo $i \in M$, supongamos que conocemos C_{ji} para todo $i \in M$, y consideremos las máquinas ordenadas de menor a mayor carga. En caso de igualdad, se rompe el empate de manera arbitraria. Entonces, $y_{ij} = C_{(j+1)i} - C_{ji}$, donde

$$C_{(j+1)i} = \begin{cases} C_{ji} + 2s_j \lfloor |L_{j+1}|/m \rfloor + 2s_j & \text{si } 1 \le i \le \lceil p_j/s_j \rceil \pmod m, \\ C_{ji} + 2s_j \lfloor |L_{j+1}|/m \rfloor & \text{en otro caso.} \end{cases}$$

El algoritmo de Euclides calcula $\lfloor a/b \rfloor$ en tiempo $\mathcal{O}(\log \text{máx}\{a, b\})$. En nuestro caso, tenemos que

$$\begin{aligned} \log \text{máx}\{L_j, m\} &= \log \text{máx}\{\lceil p_j/s_j \rceil, m\} \\ &\le \log(\lceil p_j/s_j \rceil \cdot m) \\ &\le \log(p_j \cdot m) \\ &\le \log p_j + \log m, \end{aligned}$$

y en consecuencia será polinomial en el tamaño de la entrada de nuestra instancia original. Esta observación permanece válida incluso cuando los tiempos de instalación dependen del trabajo, y por lo tanto los algoritmos mostrados en el resto del capítulo también son a tiempo polinomial.

Del Lema 2.12 sabemos que (X, σ_ϵ) es factible para el problema con división de trabajos. El siguiente teorema corresponde al resultado más relevante de este capítulo, pues entrega el primer esquema de aproximación para el problema $P|s, \text{split}|\sum w_j C_j$. También corresponde al primer algoritmo de factor constante para el problema.

Teorema 2.15. *El algoritmo Programador($\varepsilon/2$) es una $(2+\varepsilon)$-aproximación para el problema $P|s, split|\sum w_j C_j$.*

Demostración. Sea (X^*, σ^*) la programación óptima para $P|s, \text{split}|\sum w_j C_j$ en una instancia I, y sea σ el orden para la programación en lista óptima para $P|\text{part}, q_j =$

$q|\sum w_L C_L$ en la instancia I_p. Luego,

$$\begin{aligned}\sum_j w_j C_j^{\sigma_\varepsilon} &\leq \sum_j w_{L_j} C_{L_j}^{\sigma_\varepsilon} \\ &\leq (1+\varepsilon/2)\sum_j w_{L_j} C_{L_j}^{\sigma} \\ &\leq (1+\varepsilon/2)\sum_j w_{L_j} C_{L_j}^{\sigma^*} \\ &\leq 2(1+\varepsilon/2)\sum_j w_j C_j^{\sigma^*}. \\ &= (2+\varepsilon)\cdot \text{opt}.\end{aligned}$$

La primera y la última desigualdad son consecuencia del Lema 2.12. La segunda desigualdad se debe a que σ_ε es el orden retornado por el algoritmo B_ε, la tercera desigualdad es consecuencia de la optimalidad de σ. □

Ahora, supongamos que $s_j = s$ y $w_j = w$ para todo j, es decir, consideremos el problema $P|s, \text{split}|\sum C_j$. El siguiente algoritmo es una variante del algoritmo Programador. Se construye la instancia I_p mediante el algoritmo Constructor, y la instancia que se obtiene se puede resolver de manera óptima gracias al Lema 2.6.

Algorithm 6 Algoritmo $\text{Programador}_{\text{SPT}}$

1: Se construye la instancia $I_p = \text{Constructor}((s, p_j)_{j\in J})$.
2: Se programa la instancia I_p de acuerdo a la programación en lista dada por SPT.
3: Sea y_{ij} tiempo total que el grupo L_j fue procesado en la máquina i.
4: Para todo $j \in J$, sea M_j las máquinas que procesan el grupo L_j en la programación dada por SPT.
5: Sea $x_{ij} = (y_{ij} - s)/p_j$ para todo $i \in M_j$, y $x_{ij} = 0$ para todo $i \notin M_j$.
6: Retornar (X, SPT).

Teorema 2.16. *El algoritmo Programador_{SPT} es una 2-aproximación para $P|s, split|\sum C_j$.*

Demostración. Sea (X^*, σ^*) la programación óptima para $P|s, \text{split}|\sum C_j$ en una instancia I. Al tiempo de completación del trabajo j en la programación (X, SPT) lo denotamos C_j^{SPT}. Luego,

$$\begin{aligned}\sum_j C_j^{\text{SPT}} &\leq \sum_j C_{L_j}^{\text{SPT}} \\ &\leq \sum_j C_{L_j}^{\sigma^*} \\ &\leq 2\sum_j C_j^{\sigma^*} \\ &= 2\cdot \text{opt}.\end{aligned}$$

La primera y tercera desigualdad son consecuencia del Lema 2.12, y la segunda desigualdad se debe a la optimalidad de SPT para I_p. □

Esto mejora la 2.781-aproximación de Schalekamp et al. [22]. Además, el análisis de los algoritmos no puede ser mejorado para obtener un factor menor:

Teorema 2.17. *Para todo $\varepsilon > 0$, existe una instancia para el problema $P|s, split|\sum C_j$ tal que $\sum w_j C_j^{SPT}/opt \geq 2 - \varepsilon$.*

Demostración. Consideremos la instancia I_m donde $p_1 = s_1 = 1$. El óptimo para esta instancia es dividir el trabajo entre las m máquinas de manera equitativa, y por lo tanto $C_1 = 1 + 1/m$. El algoritmo $\text{Programador}_{\text{SPT}}$ genera un grupo L_1 de tamaño $\lceil p_1/s_1 \rceil = 1$, donde el tiempo de procesamiento del trabajo es $2 \cdot s_1 = 2$. Luego, el óptimo para esta instancia es programar el único trabajo de L_1 en la primera máquina, obteniendo $y_{11} = 2$, $M_1 = \{1\}$ y $y_{i1} = 0$ para todo $i \in \{2, \cdots, m\}$. Luego, la programación (X, SPT) en este caso corresponde a $x_{11} = 1$, $x_{i1} = 0$ para $i \in \{2, \cdots, m\}$, y por lo tanto $C_1^{\text{SPT}} = s_1 + p_1 x_{11} = 2$. Tenemos entonces que $\sum w_j C_j^{\text{SPT}}(I_m)/\text{opt}(I_m) = \frac{2}{1+1/m}$, y por lo tanto basta tomar $m \geq \frac{2-\varepsilon}{\varepsilon}$. □

Observación 2.18. Las instancias $\{I_m\}_{m \in \mathbb{N}}$ en la demostración anterior también son instancias para $P|s, \text{split}|\sum w_j C_j$, y por lo tanto, para todo $\varepsilon > 0$ no existe un $c < 2$ tal que $\text{Programador}(\varepsilon)$ sea una $(c + \varepsilon)$-aproximación.

Finalmente, para el problema en que los tiempos de instalación dependen del trabajo, probaremos que un algoritmo basado en la regla de Smith posee un factor de aproximación constante:

Algorithm 7 Algoritmo $\text{Programador}_{\text{Smith}}$

1: Se construye la instancia $I_p = \text{Constructor}((s_j, p_j, w_j)_{j \in J})$.
2: Se programa la instancia I_p de acuerdo a la programación en lista dada por WSPT.
3: Sea y_{ij} tiempo total que el grupo L_j fue procesado en la máquina i.
4: Para todo $j \in J$, sea M_j las máquinas que procesan el grupo L_j en la programación dada por WSPT.
5: Sea $x_{ij} = (y_{ij} - s_j)/p_j$ para todo $i \in M_j$, y $x_{ij} = 0$ para todo $i \notin M_j$.
6: Retornar (X, WSPT).

Teorema 2.19. *El algoritmo $\text{Programador}_{Smith}$ es una 4-aproximación para el problema $P|s_j, split|\sum w_j C_j$.*

Demostración. Sea (X^*, σ^*) la programación óptima para $P|s_j, \text{split}|\sum w_j C_j$ en una instancia I y sea σ el orden para la programación en lista óptima para $P|\text{part}, q_j = q_L|\sum w_L C_L$ en la instancia I_p. Al tiempo de completación del trabajo j en la programación (X, WSPT) lo denotamos C_j^{WSPT}. Luego,

$$\sum_j C_j^{\text{WSPT}} \leq \sum_j C_{L_j}^{\text{WSPT}} \leq 2 \sum_j C_{L_j}^{\sigma} \leq 2 \sum_j C_{L_j}^{\sigma^*} \leq 4 \sum_j C_j^{\sigma^*} = 4 \cdot \text{opt}.$$

La primera y la cuarta desigualdad son consecuencia del Lema 2.12. La segunda es gracias a que la programación en lista según WSPT es una 2-aproximación (Teorema 2.8), y la tercera desigualdad se debe a la optimalidad de σ. □

Capítulo 3

Algoritmos de aproximación para $R|$split,setup$|C_{\text{máx}}$

3.1. Definición del problema y trabajo relacionado

El problema de programar máquinas no relacionadas, $R||C_{\text{máx}}$ en la notación de Graham et al. [11], ha concentrado una gran atención de la comunidad de investigadores. En el Capítulo 1 se mostró la 2 aproximación existente para este problema, basada en programación lineal. Además, en el mismo artículo, Lenstra et al. [17] probaron que no existe un algoritmo de aproximación de factor $3/2 - \varepsilon$ salvo que $\mathcal{P} = \mathcal{NP}$.

Una generalización natural para este problema es la que se trata en este capítulo. Ahora cada trabajo puede ser dividido, y cada parte se puede procesar independientemente y de manera simultánea en diferentes máquinas. Además, las máquinas requieren un tiempo de instalación antes de procesar cualquier parte de algún trabajo. Formalmente, se tiene un conjunto M de m máquinas y un conjunto J de n trabajos. El tiempo de procesamiento de un trabajo j en la máquina i es $p_{ij} \in \mathbb{Z}_+$ y el tiempo de instalación de un trabajo j en la máquina i es $s_{ij} \in \mathbb{Z}_+$. Una *programación* en este contexto corresponde a un vector $x \in [0,1]^{M \times J}$, donde x_{ij} representa la fracción del trabajo j que se procesa en la máquina i, y por lo tanto $\sum_{i \in M} x_{ij} = 1$ para todo $j \in J$. Solo si $x_{ij} > 0$ la máquina i requerirá s_{ij} unidades de tiempo para la instalación, y durante este tiempo la máquina no puede procesar ningún otro trabajo. De esta forma, la *carga total* sobre la máquina i para la programación x será $\sum_{j \in J: x_{ij} > 0}(x_{ij}p_{ij} + s_{ij})$. Se denota este problema como $R|$split,setup$|C_{\text{máx}}$. Nótese que al considerar $p_{ij} = 0$ para todo i, j se puede recuperar el problema $R||C_{\text{máx}}$ interpretando los tiempos de instalación como los tiempos de procesamiento de los trabajos.

El poder encontrar un algoritmo de aproximación de factor menor a 2 para $R||C_{\text{máx}}$ se ha convertido en un importante problema abierto en el área [31]. Desde el trabajo de Lenstra et al. [17] se ha hecho un esfuerzo considerable en tratar de responder, al menos, parcialmente a esta pregunta. En la versión con *asignación restringida*, los tiempos de procesamiento son de la forma $p_{ij} \in \{p_j, \infty\}$ para todo i, j. Un caso especial de esta versión, en el cual cada trabajo puede ser asignado solo a dos máquinas, fue estudiado por Ebelendr et al. [7]. Ellos notaron que la cota inferior de 3/2 se mantiene, pero que puede obtenerse una 7/4-aproximación. Svensson [27] pudo bajar la barrera del 2 en el caso general de

asignación restringida, probando que se puede obtener una $(33/17 + \varepsilon \approx 1{,}9412 + \varepsilon)$-aproximación. Este algoritmo se basa un PL de *configuraciones de las máquinas*, donde cada variable indica el subconjunto de trabajos asignadas a una máquina. Sin embargo, esta relajación posee un gap de 2 para $R||C_{\text{máx}}$. Este tipo de relajaciones han sido bastante estudiadas para la versión max-min del problema, [30, 5, 9, 13, 3, 20].

La mayor parte de las referencias que involucran trabajos divisibles se concentra en el diseño de heurísticas. Los resultados teóricos son muy escasos, y atacan principalmente el problema en máquinas paralelas idénticas. Xing y Zhang [32] encontraron una $(7/4 - 1/m)$-aproximación para minimizar el makespan, y después Chen et al. [6] mejoraron este resultado mostrando una 5/3-aproximación. El ambiente con interrupciones es cercano al de trabajos divisibles, sin embargo, no permite procesar simultáneamente partes del mismo trabajo. Se puede ver [23, 18, 21] para mayor detalle sobre problemas relacionados que admiten interrupción de trabajos.

3.2. Contribución de este trabajo

El problema $R|\text{split,setup}|C_{\text{máx}}$ no ha sido estudiado antes, y por lo tanto el objetivo principal de este trabajo es poder encontrar cierta estructura en el problema y entender mejor su relación con el problema $R||C_{\text{máx}}$. Primero se estudia la relajación usada por Lenstra et al. [17], comprobando que el mismo algoritmo no sirve en este caso. Esto se debe a que su algoritmo podría asignar completamente un trabajo con un tiempo de procesamiento muy grande en una máquina, mientras que en el óptimo éste es dividido. Por otro lado, si se usa la solución retornada por la relajación para asignar los trabajos se podría estar pagando muchas veces el tiempo de instalación. Podemos ver entonces que existe un trade-off entre dividir, e instalar. El algoritmo para redondear desarrollado en esta tesis adapta la técnica usada en [17] de manera que se redondean aquellas variables que superen cierto umbral, lo cual garantiza no estar pagando excesivamente los tiempos de instalación. Esto permite diseñar una 3-aproximación, que se detalla en la Sección 3.3. Además, se muestra una colección de instancias que permite probar que el gap de integralidad de esta relajación es 3, y por lo tanto el algoritmo usado es el mejor posible en términos de aproximación para esta relajación.

En la Sección 3.4 se mejora el factor de aproximación, reforzando el PL anterior. Mediante un refinamiento de la técnica usada para la 3-aproximación se logra obtener una $(1 + \phi)$-aproximación, la cual también será la mejor posible en términos de aproximación para este PL. Este factor también será el mejor posible para la relajación basada en configuraciones de máquinas. En cuanto a la inaproximabilidad del problema, en la Sección 3.6 se prueba que no es posible obtener una $(\frac{e}{e-1} - \varepsilon)$-aproximación salvo que $\mathcal{P} = \mathcal{NP}$, para todo $\varepsilon > 0$. En el caso de asignación restringida, cuando $p_{ij} \in \{p_j, \infty\}$ y $s_{ij} \in \{s_j, \infty\}$, se muestra una 2-aproximación, factor que coincide con el encontrado para asignación restringida en [17] para $R||C_{\text{máx}}$. Una propiedad importante que comparten los tres algoritmos diseñados, es que en cada máquina se procesa a lo más un trabajo que ha sido dividido.

Con el objetivo de lograr una mejora en términos del gap de integralidad, en la Sección

3.7.2 se propone una relajación basada en *configuraciones de los trabajos*, es decir, corresponde a vectores de dimensión m que describen la asignación de un cierto trabajo en las distintas máquinas. El PL resultante hace infactibles las soluciones obtenidas usando los PL's anteriores para las colecciones de instancias en donde alcanza el gap de integralidad, y por lo tanto esta relajación se constituye en un buen candidato para obtener un mejor factor de aproximación. A pesar que el PL posee infinitas variables, es posible demostrar que se puede restringir a una cantidad finita de ellas, que corresponden a configuraciones especiales llamadas *maximales*. Además, es posible resolver este PL a un factor de $(1+\varepsilon)$ en tiempo polinomial, mediante separación en el dual. Finalmente, se estudia la proyección de este LP en el *espacio de asignaciones*, y se deriva un conjunto de desigualdades que definen este polítopo. Determinar el gap de integralidad del PL de configuraciones de los trabajos es una pregunta que queda abierta en este trabajo.

3.3. Una 3-aproximación

La 3-aproximación que se muestra en esta sección está basada en una generalización del algoritmo diseñado por Lenstra, Shmoys y Tardos [17]. Usando la idea mostrada en la Sección 1.2.3, el tiempo que se encuentre disponible una máquina para procesar trabajos lo denotamos por C^*. Consideremos el siguiente PL de factibilidad, donde la variable x_{ij} representa la fracción del trabajo j que es procesada en la máquina i. Este PL es una relajación para el problema de decisión asociado $R|\text{split,setup}|C_{\text{máx}}$, pues permite que los tiempos de instalación sean fraccionarios:

$$[\text{LST}]: \quad \sum_{i\in M} x_{ij} = 1 \qquad \text{para todo } j \in J, \tag{3.1}$$

$$\sum_{j\in J} x_{ij}(p_{ij}+s_{ij}) \leq C^* \qquad \text{para todo } i \in M, \tag{3.2}$$

$$x_{ij} = 0 \qquad i \in M, j \in J : s_{ij} > C^*, \tag{3.3}$$

$$x_{ij} \geq 0 \qquad \text{para todo } i \in M, j \in J. \tag{3.4}$$

Sea x es un punto extremo de [LST]. Se define el grafo $G(x) := (J \cup M, E(x))$, donde $E(x) = \{ij : x_{ij} > 0\}$. Al igual que para el problema $R||C_{\text{máx}}$, el siguiente lema es clave para diseñar el algoritmo de aproximación.

Lema 3.1. *Para todo x punto extremo de* [LST]*, cada componente conexa de $G(x)$ es un pseudo-árbol, es decir, es un árbol con a lo más una arista extra que crea un ciclo.*

La demostración de este lema se omite pues es completamente análoga a la mostrada en el Lema 1.10. Sea $E_+ = \{ij \in E(x) : x_{ij} > 1/2\}$ y $J_+ = \{j \in J : \text{existe } i \in M \text{ tal que } ij \in E_+\}$, es decir, J_+ son aquellos trabajos que en la solución fraccionaria son asignados en una fracción mayor a $1/2$ en alguna máquina. En el procedimiento de redondeo de la solución fraccionaria, un trabajo $j \in J_+$ será asignado completamente a aquella máquina $i \in M$ tal que $ij \in E_+$. Veamos ahora como trabajar con el resto de los trabajos. Sea $G'(x) = G[(J \cup M) \setminus J_+]$ el subgrafo inducido por los vértices no removidos. Dado que $G'(x)$ es un subgrafo de de $G(x)$, cada componente de $G'(x)$ es también un pseudo-árbol. Además, cada $ij \in E(G'(x))$ satisface que $0 < x_{ij} \leq 1/2$.

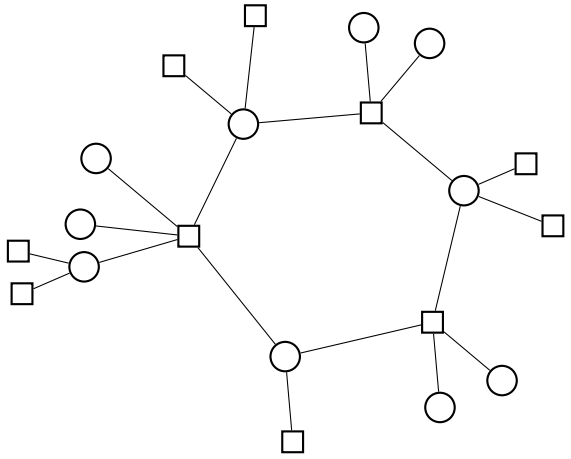

Figura 3.1: Cada componente conexa de $G'(x)$ es un pseudo-árbol.

Definición 3.2. Dado $A \subseteq E(G'(x))$, decimos que una máquina $i \in M$ es A-balanceada si existe a lo más un trabajo $j \in J \setminus J_+$ tal que $ij \in A$. Decimos que un trabajo $j \in J \setminus J_+$ es A-procesado si existe a lo más una máquina $i \in M$ tal que $ij \in A$ y $x_{ij} > 0$.

En lo que sigue, buscamos un conjunto A tal que cada máquina sea A-balanceada y cada trabajo en $J \setminus J_+$ sea A-procesado. Este conjunto será suficiente para diseñar la 3-aproximación, pues veremos que para todo trabajo $j \in J \setminus J_+$ será suficiente asignarlo entre las máquinas i tales que $ij \in A$. Además, cada máquina i al ser A-balanceada tendrá asignado a ella a lo más un trabajo $j \in J \setminus J_+$, por lo que su carga total aumentará de manera controlada al agregar el tiempo de instalación faltante.

3.3.1. Construcción del conjunto A

Consideremos una componente conexa T de $G'(x)$, la cual gracias al Lema 3.1 sabemos que es un pseudo-árbol. Denotemos por $C = j_1 i_1 j_2 i_2 \cdots j_\ell i_\ell j_1$ el único ciclo de T, en caso de existir, el cual sabemos es de largo par pues el grafo $G'(x)$ es bipartito. Los trabajos en el ciclo son $J(C) = \{j_1, \ldots, j_\ell\}$ y las máquinas $M(C) = \{i_1, \ldots, i_\ell\}$. En el ciclo definimos el emparejamiento perfecto $K_C = \{(j_k, i_k) : k \in \{1, \ldots, \ell\}\}$. En el bosque $T \setminus K_C$, denotamos por (T_u, u) el árbol enraizado en u, para todo $u \in M(C)$. Nótese que al borrar el emparejamiento perfecto del ciclo que no es K_C, no habrán dos vértices de $M(C)$ en la misma componente conexa. Para todo $u \in M(C)$, al dirigir las aristas de (T_u, u) desde la raíz obtenemos un árbol dirigido en donde cada nivel consiste solo de máquinas o solo de trabajos. Se borran todas las aristas que salen desde los vértices que representan máquinas, es decir, las que inciden en algún trabajo, y al resto de las aristas de (T_u, u) las denotamos por A_u. Sea $A := K_C \cup \bigcup_{u \in M(C)} A_u$. Los siguientes dos lemas prueban que el conjunto A satisface lo que se busca.

Lema 3.3. *Todo trabajo $j \in J \setminus J_+$ es A-procesado.*

Demostración. Consideremos primero un trabajo $j_k \in J(C)$. Por construcción, la única arista removida es $i_{k-1}j_k$, y por lo tanto j_k es A-procesado para todo $k \in \{1, \ldots, \ell\}$. Ahora consideremos un trabajo $j \in T_u$. Todos los vértices en el árbol dirigido (T_u, u) tienen una

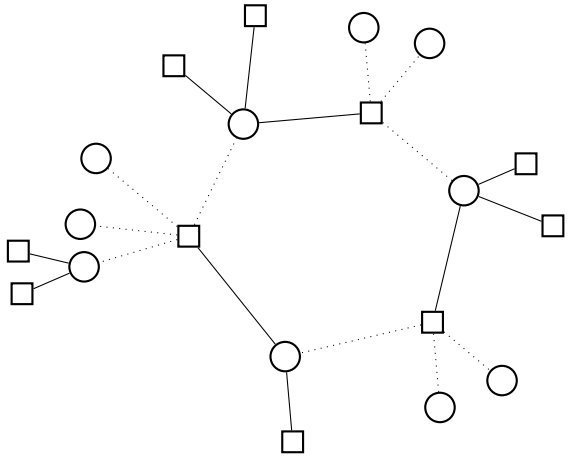

Figura 3.2: Las aristas con línea continua corresponden al conjunto A.

arista entrante (exceptuando la raíz u), y en particular, todo trabajo tiene una máquina por padre. Del proceso de construcción de A_u se tiene que la arista que une un trabajo con su padre es la única borrada, y por lo tanto cada trabajo $j \in T_u$ es A-procesado. □

Lema 3.4. *Toda máquina $i \in M$ es A-balanceada.*

Demostración. Consideremos primero una máquina $i \in M(C)$. Por construcción, se han borrado todas las aristas entre i_k y algún trabajo en T_{i_k}, por lo que la única arista que permanece es $j_k i_k$. Luego, i_k es A-balanceada para todo $k \in \{1, \ldots, \ell\}$. Tomemos ahora una máquina $i \in V(T_u)$ con $i \neq u$. Por construcción, la única arista que permanece en A es la que incide en i, y por lo tanto i es A-balanceada. □

De esta forma el conjunto A descrito satisface las propiedades que buscábamos para desarrollar el redondeo del punto extremo x. En la Figura 3.2 se puede apreciar la construcción del conjunto A sobre el pseudo-árbol de la Figura 3.1. El siguiente algoritmo retorna una nueva asignación $\tilde{x}$ y un vector de coordenadas binarias $\tilde{y}_{ij} \in \{0, 1\}$ el cual indica si un trabajo j ha sido asignado fraccionariamente a la máquina i.

Algorithm 8 Redondeo(x)

1: Construimos $G(x)$, $G'(x)$, y el conjunto A.
2: Para todo $ij \in E_+$, $\tilde{x}_{ij} \leftarrow 1$ y $\tilde{y}_{ij} \leftarrow 1$;
3: Para todo $ij \in A$, $\tilde{x}_{ij} \leftarrow \dfrac{x_{ij}}{\sum_{k:kj\in A} x_{kj}}$ y $\tilde{y}_{ij} \leftarrow 1$;
4: Para todo $ij \in E \setminus (E_+ \cup A)$, $\tilde{x}_{ij} \leftarrow 0$ y $\tilde{y}_{ij} \leftarrow 0$.

La carga de la máquina i usando la programación $(\tilde{x}, \tilde{y})$ es $\sum_{j\in J}(\tilde{x}_{ij}p_{ij} + \tilde{y}_{ij}s_{ij})$, y el makespan de la programación es $C_{\max}(\tilde{x}, \tilde{y}) = \max_{i\in M} \sum_{j\in J}(\tilde{x}_{ij}p_{ij} + \tilde{y}_{ij}s_{ij})$.

Teorema 3.5. *El vector $\tilde{x}$ es una asignación válida y $C_{\max}(\tilde{x}, \tilde{y}) \leq 3C^*$.*

Demostración. Verifiquemos que $\tilde{x}$ asigna completamente cada trabajo, es decir, $\sum_{i\in M} x_{ij} = 1$ para todo $j \in J$. Si $j \in J_+$ entonces existe una única máquina $i \in M$ tal que $ij \in E_+$ y por lo tanto es asignado completamente en esa máquina. Si $j \in J \setminus J_+$, entonces

$$\sum_{i\in M} \tilde{x}_{ij} = \sum_{i:ij\in A} \tilde{x}_{ij} + \sum_{i:ij\in E\setminus(E_+\cup A)} \tilde{x}_{ij} = \sum_{i:ij\in A} \frac{x_{ij}}{\sum_{k:kj\in A} x_{kj}} = 1.$$

Veamos ahora que $C_{\text{máx}}(\tilde{x}, \tilde{y}) \leq 3C^*$. Primero notemos que para todo $ij \in E_+$, tenemos que $1 = \tilde{y}_{ij} = \tilde{x}_{ij} \leq 2x_{ij}$, puesto que $ij \in E_+$ si y solo si $x_{ij} > 1/2$. Por otro lado, tenemos que para todo $j \in J \setminus J_+$ se tiene que $\sum_{k:kj\in A} x_{kj} \geq 1/2$, y como es A-procesado existe a lo más una máquina que procesa fraccionariamente j y que no está en A. Luego, para toda máquina $i \in M$ tenemos que

$$\begin{aligned}
\sum_{j\in J}(\tilde{x}_{ij}p_{ij} + \tilde{y}_{ij}s_{ij}) &= \sum_{j:ij\in E_+}(\tilde{x}_{ij}p_{ij} + \tilde{y}_{ij}s_{ij}) + \sum_{j:ij\in A}(\tilde{x}_{ij}p_{ij} + \tilde{y}_{ij}s_{ij}) \\
&= \sum_{j:ij\in E_+}(\tilde{x}_{ij}p_{ij} + \tilde{y}_{ij}s_{ij}) + \sum_{j:ij\in A}\left(\frac{x_{ij}}{\sum_{k:kj\in A} x_{kj}}p_{ij} + s_{ij}\right) \\
&\leq \sum_{j:ij\in E_+} 2x_{ij}(p_{ij} + s_{ij}) + \sum_{j:ij\in A}(2x_{ij}p_{ij} + s_{ij}) \\
&\leq 2\sum_{j\in J} x_{ij}(p_{ij} + s_{ij}) + \sum_{j:ij\in A} s_{ij} \\
&\leq 2C^* + \sum_{j:ij\in A} s_{ij}.
\end{aligned}$$

Dado que la máquina i es A-balanceada tenemos que $|\{j : ij \in A\}| \leq 1$. Además, como $ij \in A \subseteq E(x) = \{ij : x_{ij} > 0\}$ la restricción 3.3 de [LST] implica que $s_{ij} \leq C^*$. Como consecuencia se tiene $\sum_{j:ij\in A} s_{ij} \leq C^*$ y por lo tanto concluimos que

$$C_{\text{máx}}(\tilde{x}, \tilde{y}) = \max_{i\in M}\sum_{j\in J}(\tilde{x}_{ij}p_{ij} + \tilde{y}_{ij}s_{ij}) \leq 3C^*.$$

□

El valor de C^* que se usará para extraer un punto extremo y luego redondearlo corresponderá al menor racional tal que [LST] es factible. Notemos que [LST] es factible para opt, y por lo tanto $C^* \leq$ opt, que es la cota inferior que nos permite concluir la 3-aproximación. A continuación se explica como obtener C^*. Consideremos el conjunto $J \times M$ ordenado de acuerdo a un orden total $<_s$ tal que $(i, j) <_s (i', j')$ si $s_{ij} < s_{i'j'}$. Sea F_r el conjunto de los r primeros pares en $(J \times M, <_s)$, para todo $1 \leq r \leq nm$. Sea [LST(F_r)] el programa lineal que fija en 0 aquellas variables en F_r^c, es decir,

$$\begin{aligned}
[\text{LST}(F_r)]\text{:} \quad \text{mín} \quad & C & & \\
\text{s.t.} \quad & \sum_{i\in M} x_{ij} = 1 & & \text{para todo } j \in J, \\
& \sum_{j\in J} x_{ij}(p_{ij} + s_{ij}) \leq C & & \text{para todo } i \in M, \\
& x_{ij} \geq 0 & & \text{para todo } (i, j) \in F_r.
\end{aligned}$$

Denotemos por C^r el mínimo para [LST(F_r)], y x^r un punto extremo en que se alcanza tal valor. Una vez obtenida esta solución, se verifica si $x_{ij} = 0$ para todo (i, j) tal que $s_{ij} > C^r$, que es la restricción (3.3) de [LST]. Si esto ocurre, entonces x^r es factible para [LST] al considerar C^r, y por lo tanto $C^r \geq C^*$. Sea F el conjunto de los r tales que x^r es factible para [LST] al considerar C^r.

Lema 3.6. *El menor valor que hace factible a* [LST] *es* $C^* = \min_{r \in F} C^r$ *y este valor se puede calcular en tiempo polinomial.*

Demostración. Se tiene que $C^* \leq C^r$ para todo $r \in F$, y por lo tanto $C^* \leq \min_{r \in F} C^r$. Por otro lado, sea x^* un punto extremo de [LST] para el menor valor C^*. Por factibilidad de x^* se tiene que $x_{ij} = 0$ si $s_{ij} > C^*$, y por lo tanto (x^*, C^*) es factible para [LST(F_{nm-r^*})] con $r^* = |\{(i,j) : s_{ij} > C^*\}|$. Luego, $C^* \geq C^{r^*} \geq \min_{r \in F} C^r$ y por lo tanto $C^* = \min_{r \in F} C^r$. Para calcular este mínimo basta recorrer todos los nm posibles valores de r, de menor a mayor, y para cada r verificar que x^r satisface la restricción (3.3) de [LST] se puede realizar en a lo más nm operaciones. □

Teorema 3.7. *Existe una 3-aproximación para* $R|split,setup|C_{\text{máx}}$.

Demostración. Sea $(\tilde{x}, \tilde{y})$ el output de Redondeo(x) y opt el makespan óptimo para el problema $R|$split,setup$|C_{\text{máx}}$. Del Teorema 3.5 tenemos que $C_{\text{máx}}(\tilde{x}, \tilde{y}) \leq 3C^*$, pero también tenemos que $C^* \leq$ opt, pues [LST] es factible para opt. Luego, $C_{\text{máx}}(\tilde{x}, \tilde{y}) \leq 3 \cdot \text{opt}$. □

Finalizamos esta sección mostrando que 3 es el mejor factor de aproximación posible para la relajación [LST]:

Teorema 3.8. *Para todo* $\varepsilon > 0$*, existe una instancia del problema* $R|split,setup|C_{\text{máx}}$ *tal que* $\text{opt}/C^* \geq 3 - \varepsilon$*, donde* C^* *es el menor racional tal que* [LST] *es factible.*

Demostración. Exhibiremos una sucesión de instancias $\{I_k\}_{k \in \mathbb{N}}$ tal que $\frac{\text{opt}(I_k)}{C^*(I_k)} \nearrow 3$ cuando $k \to \infty$. La instancia I_k tiene $2k+1$ trabajos. Por cada trabajo j se tiene un conjunto M_j de k máquinas, tales que $M_j \cap M_{j'} = \emptyset$ si $j \neq j'$. Sea $s_{ij} = 1$ y $p_{ij} = 2k$ para cada $j \in J$ y $i \in M_j$, y $s_{ij} = p_{ij} = \infty$ si $i \in M_{j'}$ con $j \neq j'$. Se tiene también un conjunto extra M' de k máquinas, donde $p_{ij} = 1$ y $s_{ij} = 0$ para todo $j \in J$ y $i \in M'$. En la Figura 3.3 se puede apreciar la construcción de la instancia I_k.

Probaremos que opt $= 3$ y que [LST] es factible para $C^* = 1 + 1/2k$. Para ver que opt $= 3$, hay que notar que existen dos casos. Si todos los trabajos en J son asignados en las máquinas de M', el makespan es 3, puesto que son k máquinas, $2k+1$ trabajos y el tiempo de instalación de cada trabajo es 1 en cada máquina de M'. De otra forma, existe un trabajo $j \in J$ que es completamente asignado en las máquinas de M_j. El makespan mínimo sobre las máquinas de M_j se alcanza cuando se procesa una fracción $1/k$ del trabajo j en cada máquina $i \in M_j$, y por lo tanto la carga sobre cada una de estas máquinas será $1 + p_{ij}/k = 1 + 2k/k = 3$. Ahora consideremos la siguiente asignación fraccionaria de los trabajos,

$$x_{ij} = \begin{cases} 1/2k & \text{si } j \in J \text{ y } i \in M_j \cup M', \\ 0 & \text{en otro caso.} \end{cases}$$

Dado que $|M_j \cup M'| = 2k$, cada trabajo será asignado completamente. Si $i \in M'$, entonces

$$\sum_{j \in J} x_{ij}(p_{ij} + s_{ij}) = \sum_{j \in J} x_{ij} = \frac{2k+1}{2k} = 1 + \frac{1}{2k} = C^*,$$

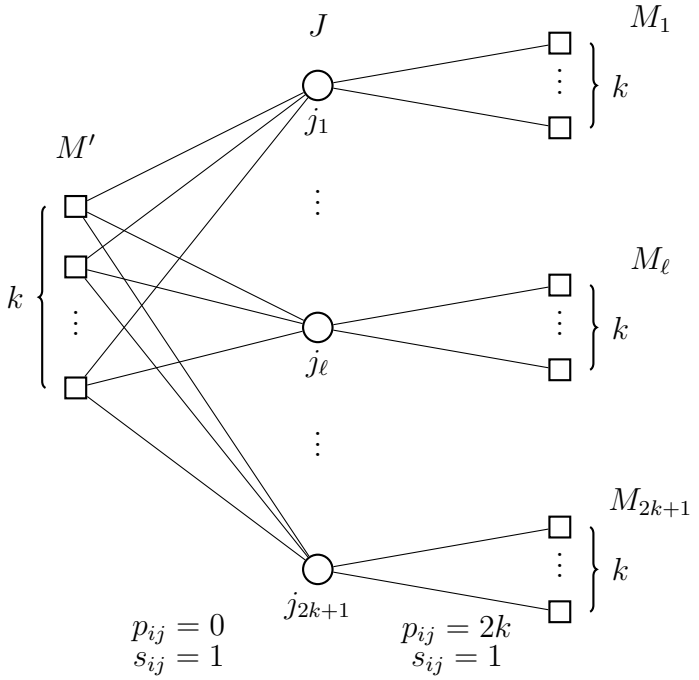

Figura 3.3: Ejemplo que muestra que el mejor factor para [LST] es 3.

y si $i \in M_j$,

$$\sum_{j \in J} x_{ij}(p_{ij} + s_{ij}) = \frac{1}{2k}(2k+1) = 1 + \frac{1}{2k} = C^*.$$

□

Concluimos entonces que $\dfrac{\text{opt}(I_k)}{C^*(I_k)} \geq \dfrac{3}{1+1/k} \nearrow 3$.

3.4. Una $(1+\phi)$-aproximación

En la sección anterior se mostró una 3-aproximación para el problema, pero además se probó que es lo mejor posible que se puede obtener usando la relajación [LST]. Para poder mejorar el factor de aproximación, reforzaremos la relajación usada en la sección anterior. Para ello, observemos que la desigualdad 3.2 de [LST] corresponde a la relajación lineal del siguiente programa mixto con variables binarias y_{ij}:

$$\sum_{j \in J}(x_{ij}p_{ij} + y_{ij}s_{ij}) \leq C^* \qquad \text{para todo } i \in M, \tag{3.5}$$

$$x_{ij} \leq y_{ij} \qquad \text{para todo } i \in M \text{ y } j \in J. \tag{3.6}$$

Sea i una máquina y j algún trabajo. Si consideramos la desigualdad correspondiente a la máquina i en 3.5 y la multiplicamos por y_{ij}, tenemos que:

$$y_{ij}x_{ij}p_{ij} + y_{ij}^2 s_{ij} \leq \sum_{j \in J}(y_{ij}x_{ij}p_{ij} + y_{ij}^2 s_{ij}) \leq C^* y_{ij}.$$

Si $C^* > s_{ij}$, entonces la desigualdad anterior implica siguiente desigualdad lineal:

$$x_{ij}\frac{p_{ij}}{C^* - s_{ij}} \leq y_{ij}, \tag{3.7}$$

puesto que toda solución factible del programa mixto satisface $y_{ij}x_{ij} = x_{ij}$ y $y_{ij}^2 = y_{ij}$. En una solución óptima de la relajación lineal, y_{ij} tomará el menor valor posible de modo que satisfaga las desigualdades 3.6 y 3.7. Luego, definiendo $\alpha_{ij} = \text{máx}\left\{1, \frac{p_{ij}}{C^* - s_{ij}}\right\}$ si $C^* > s_{ij}$ y $\alpha_{ij} = 1$ en otro caso, obtenemos la siguiente relajación:

$$[\text{LST}_{\text{strong}}]: \quad \sum_{i\in M} x_{ij} = 1 \qquad \text{para todo } j \in J, \tag{3.8}$$

$$\sum_{j\in J} x_{ij}(p_{ij} + \alpha_{ij}s_{ij}) \leq C^* \qquad \text{para todo } i \in M, \tag{3.9}$$

$$x_{ij} = 0 \qquad i \in M, j \in J : s_{ij} > C^*, \tag{3.10}$$

$$x_{ij} \geq 0 \qquad \text{para todo } i \in M, j \in J. \tag{3.11}$$

Esta relajación es al menos tan buena como [LST] en términos de su gap de integralidad, pues $\alpha_{ij} \geq 1$. Sea x un punto extremo de $[\text{LST}_{\text{strong}}]$ y el grafo $G(x)$ como en la sección anterior. En este grafo cada componente conexa es un pseudo-árbol, y la demostración se omite pues es la misma mostrada en el Lema 1.10. Dado $\beta \in (1/2, 1)$, sea $E_+^\beta = \{ij \in E(x) : x_{ij} > \beta\}$ y $J_+^\beta = \{j \in J : \text{existe } i \in M \text{ tal que } ij \in E_+^\beta\}$. Nótese que en la sección anterior se escogió $\beta = 1/2$. La idea, es obtener un factor de aproximación parametrizado en β, y luego optimizar sobre β. Sea $G'_\beta(x) = G[(J \cup M) \setminus J_+^\beta]$ y A el conjunto construido tal como en la sección anterior. Usando el algoritmo Redondeo(x) obtenemos una programación $(\tilde{x}, \tilde{y})$. El siguiente lema, nos permitirá analizar el factor de aproximación que nos entrega este procedimiento.

Lema 3.9. *Sea $\beta \in (1/2, 1)$. Luego,* $\max\limits_{\mu\in[0,1]} \left\{\mu + \text{máx}\left\{\frac{1}{\beta}, \frac{1-\mu}{1-\beta}\right\}\right\} = \text{máx}\left\{\frac{1}{1-\beta}, 1 + \frac{1}{\beta}\right\}$.

Demostración. Sea $f : [0,1] \to \mathbb{R}$ tal que $f(\mu) = \mu + \text{máx}\left\{\frac{1}{\beta}, \frac{1-\mu}{1-\beta}\right\}$. Podemos escribir f se la siguiente forma:

$$f(\mu) = \begin{cases} \frac{1-\mu\beta}{1-\beta} & \text{si } \mu \in [0, 2 - 1/\beta], \\ 1 + \frac{1}{\beta} & \text{si } \mu \in (2 - 1/\beta, 1]. \end{cases}$$

Al ser una función lineal por pedazos, el máximo en el intervalo $[0,1]$ se alcanza en el conjunto $\{0, 2 - 1/\beta, 1\}$. Al evaluar, se obtiene que

$$\begin{aligned} \max_{u\in[0,1]} f(\mu) &= \text{máx}\{f(0), f(2 - 1/\beta), f(1)\} \\ &= \text{máx}\left\{\text{máx}\left\{\frac{1}{\beta}, \frac{1}{1-\beta}\right\}, \frac{1}{\beta}, 1 + \frac{1}{\beta}\right\} \\ &= \text{máx}\left\{\frac{1}{1-\beta}, \frac{1}{\beta}, 1 + \frac{1}{\beta}\right\}. \end{aligned}$$

Como $\beta \in (1/2, 1)$, se tiene que $\frac{1}{\beta} < \frac{1}{1-\beta}$ y por lo tanto $\max_{u\in[0,1]} f(\mu) = \max\left\{\frac{1}{1-\beta}, 1+\frac{1}{\beta}\right\}$. □

Teorema 3.10. . *Dado $\beta \in (1/2, 1]$, el vector $\tilde{x}$ es una asignación válida y $C_{\max}(\tilde{x}, \tilde{y}) \leq \max\left\{\frac{1}{1-\beta}, 1+\frac{1}{\beta}\right\} C^*$.*

Demostración. El hecho que $\tilde{x}, \tilde{y}$ sea una asignación factible se sigue del mismo argumento usado en el Teorema 3.5. Para toda arista $ij \in E_+$ tenemos que $x_{ij} > \beta$, y entonces $1 = \tilde{x}_{ij} = \tilde{y}_{ij} < 1/\beta \cdot x_{ij}$. Como todo trabajo $j \in J \setminus J_+$ es A-procesado, a lo más una máquina que procesa j no está considerada en A, y por lo tanto $\sum_{k:kj\in A} x_{kj} \geq 1-\beta$ y por lo tanto $\tilde{x}_{ij} \leq \frac{x_{ij}}{1-\beta}$. Luego, para una máquina i se tendrá

$$\begin{aligned}\sum_{j\in J}(\tilde{x}_{ij}p_{ij} + \tilde{y}_{ij}s_{ij}) &= \sum_{j:ij\in E_+}(\tilde{x}_{ij}p_{ij} + \tilde{y}_{ij}s_{ij}) + \sum_{j:ij\in A}(\tilde{x}_{ij}p_{ij} + \tilde{y}_{ij}s_{ij}) \\ &\leq \frac{1}{\beta}\sum_{j:ij\in E_+} x_{ij}(p_{ij}+s_{ij}) + \frac{1}{1-\beta}\sum_{j:ij\in A} x_{ij}p_{ij} + \sum_{j:ij\in A} s_{ij}.\end{aligned}$$

Como la máquina i es A-balanceada, existe a lo más un trabajo j tal que $ij \in A$ (si tal trabajo no existe, entonces la carga sobre i es a lo más C^*/β). Sea $j(i)$ ese trabajo, y definamos $\mu_i = s_{ij(i)}/C^*$. Se observa que

$$\begin{aligned}x_{ij(i)}(p_{ij(i)} + \alpha_{ij(i)}s_{ij(i)}) &\geq x_{ij(i)}p_{ij(i)}\left(1 + \frac{s_{ij(i)}}{C^* - s_{ij(i)}}\right) \\ &= x_{ij(i)}p_{ij(i)}\left(1 + \frac{\mu_i}{1-\mu_i}\right) = x_{ij(i)}p_{ij(i)}\frac{1}{1-\mu_i}.\end{aligned}$$

SI usamos esta última desigualdad en la que teníamos previamente,

$$\begin{aligned}\sum_{j\in J}(\tilde{x}_{ij}p_{ij} + \tilde{y}_{ij}s_{ij}) &\leq \frac{1}{\beta}\sum_{j:ij\in E_+} x_{ij}(p_{ij}+s_{ij}) + \frac{1}{1-\beta}x_{ij(i)}p_{ij(i)} + s_{ij(i)} \\ &\leq \frac{1}{\beta}\sum_{j:ij\in E_+} x_{ij}(p_{ij}+s_{ij}) + \frac{1-\mu_i}{1-\beta}x_{ij(i)}(p_{ij(i)} + \alpha_{ij(i)}s_{ij(i)}) + \mu_i C^* \\ &\leq \max\left\{\frac{1}{\beta}, \frac{1-\mu_i}{1-\beta}\right\}\sum_{j\in J} x_{ij}(p_{ij} + \alpha_{ij}s_{ij}) + \mu_i C^* \\ &\leq \left(\mu_i + \max\left\{\frac{1}{\beta}, \frac{1-\mu_i}{1-\beta}\right\}\right) C^* \\ &\leq \max_{\mu\in[0,1]}\left\{\mu + \max\left\{\frac{1}{\beta}, \frac{1-\mu}{1-\beta}\right\}\right\} C^* \\ &= \max\left\{\frac{1}{1-\beta}, 1+\frac{1}{\beta}\right\} C^*.\end{aligned}$$

En la última igualdad se usó el Lema 3.9. □

Corolario 3.11. *La programación $(\tilde{x}, \tilde{y})$ satisface que $C_{\max}(\tilde{x}, \tilde{y}) \leq (1+\phi)C^*$, donde $\phi = (1+\sqrt{5})/2 \approx 1{,}618$.*

Demostración. En el Teorema 3.10, se demostró que $C_{\text{máx}}(\tilde{x},\tilde{y}) \leq \text{máx}\left\{\frac{1}{1-\beta}, 1+\frac{1}{\beta}\right\} C^*$ para todo $\beta \in (1/2, 1]$. Luego, se tiene que

$$C_{\text{máx}}(\tilde{x},\tilde{y}) \leq C^* \min_{\beta\in(1/2,1]} \text{máx}\left\{\frac{1}{1-\beta}, 1+\frac{1}{\beta}\right\}.$$

Este mínimo se alcanza en único punto β^*, el cual satisface $\frac{1}{1-\beta^*} = 1+\frac{1}{\beta^*}$, es decir, $\beta^{*2}+\beta^*-1=0$. De las dos soluciones de esta ecuación, la que minimiza la función es $\beta^* = (\sqrt{5}-1)/2$, y entonces el valor de la función en ese punto es

$$1+1/\beta^* = 1+(1+\sqrt{5})/2 = 1+\phi.$$

□

Teorema 3.12. *Existe una* $(1+\phi \approx 2{,}618)$*-aproximación para* $R|split,setup|C_{\text{máx}}$*, donde* $\phi = \frac{1+\sqrt{5}}{2}$.

El valor de C^* se busca de manera análoga a la descrita en la sección anterior, y la demostración se omite por ser completamente análoga a la del Teorema 3.7. Al igual que en la sección anterior, veremos que el factor $(1+\phi)$ es el mejor posible para la relajación $[\text{LST}_{\text{strong}}]$.

Teorema 3.13. *Para todo* $\varepsilon > 0$*, existe una instancia del problema* $R|split,setup|C_{\text{máx}}$ *tal que* $opt/C^* \geq 1+\phi-\varepsilon$*, donde* C^* *es el menor racional tal que* $[\text{LST}_{\text{strong}}]$ *es factible.*

Demostración. Al igual que en el Teorema 3.8, mostraremos una sucesión de instancias $\{I_k\}_{k\in\mathbb{N}}$ tales que $\text{opt}(I_k)/C^*(I_k) \nearrow 1+\phi$. La instancia I_k posee dos conjuntos de trabajos J y J' de igual tamaño k, y por cada trabajo $j_\ell \in J$ tenemos un trabajo $j'_\ell \in J'$. Para cada par de trabajos j_ℓ, j'_ℓ tenemos una máquina padre i^ℓ_p de ambos, tales que ambos trabajos pueden ser procesados en esta máquina con $p_{i^\ell_p j_\ell} = p_{i^\ell_p j'_\ell} = 0$ y $s_{i^\ell_p j_\ell} = s_{i^\ell_p j'_\ell} = \phi/2$. Además, cada trabajo $j \in J \cup J'$ puede ser procesado por una máquina hija $i_c(j)$ tal que $s_{i_c(j)j} = 0$ y $p_{i_c(j)j} = 1+\phi$, y se tiene también un trabajo j_t que puede ser procesado en todas las máquinas padre, con tiempo de instalación 1 y tiempo de procesamiento 0. Para todos los pares de trabajos y máquinas que no están conectados se tiene $s_{ij} = p_{ij} = \infty$.

Veremos que para esta instancia el óptimo es $1+\phi$, y que $[\text{LST}_{\text{strong}}]$ es factible para $C^* = 1+1/k$. Para ver que $1+\phi$ es el óptimo, sea $i_p(j_\ell)$ la máquina que procesa completamente el trabajo j^t en una solución óptima (este trabajo no se reparte entre varias máquinas en una solución óptima). Los trabajos j_ℓ y j'_ℓ tampoco se dividen en una solución óptima. Si alguno de estos trabajos se asigna completamente a su máquina hija, el makespan corresponde a $1+\phi$. De otra forma, j_ℓ y j'_ℓ son ambas asignadas a $i_p(j_\ell) = i_p(j'_\ell)$, y en ese caso el makespan sería $1+\phi/2+\phi/2 = 1+\phi$. Ahora veamos que la solución fraccionaria siguiente es factible para $C^* = 1+1/k$:

$$x_{ij} = \begin{cases} \phi-1 & \text{para } j \in J\cup J', i = i_p(j), \\ 2-\phi & \text{para } j \in J\cup J', i = i_c(j), \\ \frac{1}{k} & \text{para } j = j^t, i = i_p(j) \text{ para } j \in J, \\ 0 & \text{en otro caso.} \end{cases}$$

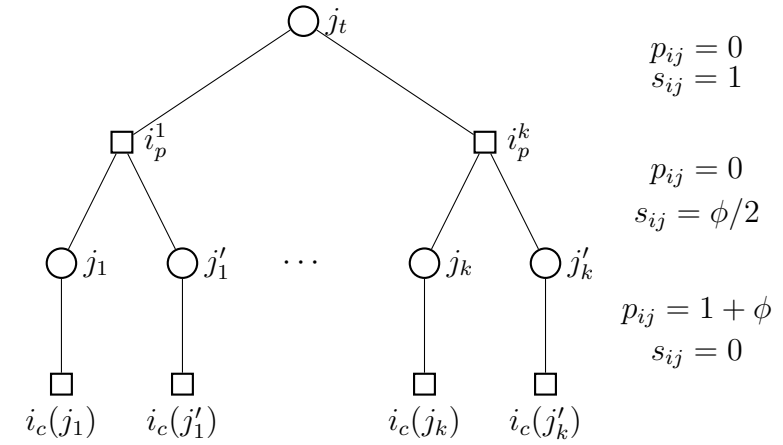

Figura 3.4: Ejemplo que muestra que el mejor factor para $[\mathrm{LST}_{\mathrm{strong}}]$ es $1+\phi$.

Si $j = j^t$,

$$\sum_{i \in M} x_{ij} = \sum_{\ell=1}^{k} x_{i_p(\ell)j} = k \cdot \frac{1}{k} = 1.$$

Si $j \in J \cup J'$,

$$\sum_{i \in M} x_{ij} = x_{i_p(j)j} + x_{i_c(j)j} = \phi - 1 + 2 - \phi = 1.$$

Ahora, si $i = i_p(j)$ para algún $j \in J$,

$$\begin{aligned}
\sum_{j \in J \cup J' \cup \{j^t\}} x_{ij}(p_{ij} + \alpha_{ij} s_{ij}) &= \frac{1}{k}(0 + 1 \cdot 1) + (\phi - 1)\left(0 + 1 \cdot \frac{\phi}{2}\right) + (\phi - 1)\left(0 + 1 \cdot \frac{\phi}{2}\right) \\
&= \frac{1}{k} + \phi(\phi - 1) \\
&= \frac{1}{k} + \left(\frac{\sqrt{5}+1}{2}\right)\left(\frac{\sqrt{5}-1}{2}\right) \\
&= \frac{1}{k} + \frac{5-1}{4} \\
&= 1 + \frac{1}{k}.
\end{aligned}$$

Finalmente, si $i = i_c(j)$ para algún $j \in J$,

$$\begin{aligned}
\sum_{j \in J \cup J' \cup \{j^t\}} x_{ij}(p_{ij} + \alpha_{ij} s_{ij}) &= (2 - \phi)(1 + \phi + \alpha_{i_c(j)j} \cdot 0) \\
&= (2 - \phi)(1 + \phi) \\
&= \left(\frac{3 - \sqrt{5}}{2}\right)\left(\frac{3 + \sqrt{5}}{2}\right) \\
&= \frac{9 - 5}{4} \\
&= 1.
\end{aligned}$$

□

3.5. Una 2-aproximación bajo asignación restringida

En esta sección consideramos un ambiente de programación con asignación restringida, es decir, para cada trabajo $j \in J$ existe un conjunto de máquinas M_j tales que $p_{ij} = p_j$, $s_{ij} = s_j$ si $i \in M_j$ y $p_{ij} = s_{ij} = \infty$ si $i \notin M_j$. El objetivo a minimizar es el mismo que antes, el makespan. Usaremos la misma relajación de la sección anterior para diseñar una 2-aproximación para este problema:

$$[\text{LST}_{\text{strong}}]: \quad \sum_{i \in M_j} x_{ij} = 1 \qquad \text{para todo } j \in J, \tag{3.12}$$

$$\sum_{j \in J: i \in M_j} x_{ij}(p_j + \alpha_j s_j) \leq C^* \qquad \text{para todo } i \in M, \tag{3.13}$$

$$x_{ij} = 0 \qquad \text{para todo } i \in M,\ j \in J : s_{ij} > C^*, \tag{3.14}$$

$$x_{ij} \geq 0 \qquad \text{para todo } j \in J \text{ y } i \in M_j, \tag{3.15}$$

donde $\alpha_j = \max\{1, p_j/(C^* - s_j)\}$. Sea x una solución extrema de este PL y consideremos $G'(x) = (J \cup M, E'(x))$ donde $E'(x) = \{ij : 0 < x_{ij} < 1\}$. Aquellas variables que valen 0 o 1 las fijamos en ese valor y no son consideradas en el proceso de redondeo. Al igual que antes, cada componente conexa de $G'(x)$ es un pseudo-árbol. Sea A construido sobre $G'(x)$ de igual forma que en 3.3.1. Tal como se hizo en esa sección, sea T una componente conexa de $G'(x)$ y u en el ciclo C de T (en caso que exista), donde denotamos por (T_u, u) el árbol enraizado en u. Dado j, sea $u(j)$ el único nodo en el ciclo T tal que j es un nodo de $T_{u(j)}$ y sea $ch(j)$ los hijos de j en $T_{u(j)}$. Nótese que los elementos en $ch(j)$ son únicamente máquinas. Consideremos un emparejamiento perfecto K_C en el ciclo C, y si j es un trabajo en el ciclo C definimos $ch'(j) = ch(j) \cup \{m_j\}$, donde m_j es la única máquina en el ciclo C tal que $m_j j \in K_C$. Si j es un trabajo que no está en el ciclo, entonces $ch'(j) = ch(j)$. Vamos a ver que con C^* unidades extra de tiempo en cada máquina de $ch'(j)$ es suficiente para poder procesar completamente el trabajo j, es decir, se podrá distribuir el trabajo j sobrecargando de manera controlada cada una de estas máquinas.

El lema que sigue nos dará una cota inferior sobre el grado de los nodos de trabajo en el grafo $G'(x)$. Denotamos por $\delta(v)$ el conjunto de nodos incidentes a v en el grafo $G'(x)$.

Lema 3.14. *Para cualquier solución extrema x, el grado $|\delta(j)|$ de de un trabajo j en el grafo $G'(x)$ es mayor o igual a* $\max\{2, \lceil \alpha_j \rceil\}$.

Demostración. Es claro que $|\delta(j)| \geq 2$, pues $x_{ij} < 1$ para todo $ij \in E'(x)$. Consideremos la variable $y_{ij} = 1$ cuando $0 < x_{ij} < 1$ y 0 en otro caso. Notemos que si $ij \in E'(x)$ entonces $s_j < C^*$, y $x_{ij}\alpha_j \leq 1$. En efecto:

$$x_{ij}\alpha_j = x_{ij} \max\left\{1, \frac{p_j}{C^* - s_j}\right\} \leq \frac{x_{ij}p_j}{C^* - s_j} \leq 1,$$

donde la última cota es debido a la restricción 3.13. Luego, con esto podemos acotar el grado:

$$|\delta(j)| = \sum_{i \in M_j} y_{ij} = \sum_{i \in M_j} 1 \geq \sum_{i \in M_j} \alpha_j x_{ij} = \alpha_j.$$

De la integralidad de $|\delta(j)|$ se concluye que $|\delta(j)| \geq \lceil \alpha_j \rceil$. □

Sea i una máquina en $ch'(j)$. La solución entregada por [$\text{LST}_{\text{strong}}$] ya asigna $x_{ij}(p_j + \alpha_j s_j)$ unidades de tiempo en la máquina i para procesar al trabajo j. Sea $L_{ij} := x_{ij}(p_j + \alpha_j s_j) + C^* - s_j$ el tiempo total disponible para procesar el trabajo j si incrementamos el tiempo disponible en la máquina i en C^* unidades de tiempo (esto es descontando las s_j unidades de tiempo de instalación). El siguiente lema muestra que la disponibilidad total $\sum_{i \in ch'(j)} L_{ij}$ de las máquinas en $ch'(j)$ al incrementar su disponibilidad en C^* unidades de tiempo es suficiente para procesar el trabajo j:

Lema 3.15. *Para todo trabajo j se tiene que $\sum_{i \in ch'(j)} L_{ij} \geq p_j$.*

Demostración. Para cada trabajo j, existe exactamente una máquina que es incidente a j en $G(x)$, que no está en $ch'(j)$. Llamemos i_j^* a esta máquina. Luego, gracias al lema anterior se tiene

$$\begin{aligned}\sum_{i \in ch'(j)} L_{ij} &= |ch'(j)|(C^* - s_j) + (p_j + \alpha_j s_j) \sum_{i \in ch'(j)} x_{ij} \\ &\geq (|\delta(j)| - 1)(C^* - s_j) + (1 - x_{i_j^* j})(p_j + \alpha_j s_j) \\ &\geq \text{máx}\{1, \lceil \alpha_j \rceil - 1\}(C^* - s_j) + (1 - x_{i_j^* j})(p_j + \alpha_j s_j).\end{aligned}$$

Cuando $\alpha_j = 1$ podemos acotar la última expresión por $C^* - s_j \geq p_j$. Cuando $\alpha_j > 1$ tenemos que $\alpha_j = p_j/(C^* - s_j)$ y nuevamente usaremos el lema anterior, específicamente que $x_{ij} \leq 1/\alpha_j$,

$$\begin{aligned}\sum_{i \in ch'(j)} L_{ij} &\geq (\lceil \alpha_j \rceil - 1)(C^* - s_j) + (1 - x_{i_j^* j})(p_j + \alpha_j s_j) \\ &\geq (\lceil \alpha_j \rceil - 1)(C^* - s_j) + \left(1 - \frac{1}{\alpha_j}\right)(p_j + \alpha_j s_j) \\ &= (\lceil \alpha_j \rceil - 1)(C^* - s_j) + (\alpha_j - 1)\left(\frac{p_j}{\alpha_j} + s_j\right) \\ &= (\lceil \alpha_j \rceil - 1)(C^* - s_j) + (\alpha_j - 1)C^* \\ &\geq (\lceil \alpha_j \rceil + \alpha_j - 2)(C^* - s_j).\end{aligned}$$

Cuando $\alpha_j > 1$ tenemos que $\lceil \alpha_j \rceil \geq 2$, y por lo tanto acotamos inferiormente la última expresión por $\alpha_j(C^* - s_j) = p_j$, lo cual prueba el lema. □

Teorema 3.16. *Existe una 2-aproximación para el problema de programar máquinas no relacionadas bajo asignación restringida.*

Demostración. El lema anterior prueba que podemos procesar cada trabajo j en las máquinas $ch'(j)$ cuando incrementamos la disponibilidad de cada máquina en C^* unidades de tiempo, lo que aumenta el makespan a lo más a $2C^*$, pues $ch'(j) \cap ch'(k) = \emptyset$ si $j \neq k$. El menor valor C^* tal que [$\text{LST}_{\text{strong}}$] sea factible se encontrará de la misma forma que en las secciones anteriores. □

Si hacemos $p_j = 0$ para todo trabajo $j \in J$ tendremos que $\alpha_j = 1$, y en este caso [$\text{LST}_{\text{strong}}$] recupera la relajación para la versión de $R||C_{\text{máx}}$ bajo asignación restringida, la cual posee un gap de integralidad igual a 2. En consecuencia, el gap de integralidad de [$\text{LST}_{\text{strong}}$] bajo asignación restringida es 2, lo cual hace al algoritmo anterior lo mejor posible para esta relajación.

3.6. Cota inferior en la aproximabilidad del problema

En las secciones 3.3 y 3.4 se diseñaron algoritmos de aproximación para el problema $R|\text{split,setup}|C_{\text{máx}}$ y en la Sección 3.5 un algoritmo de aproximación para la versión bajo asignación restringida. El siguiente teorema entrega un resultado negativo respecto a la aproximabilidad del problema, es decir, acota inferiormente el factor de aproximación que pueda entregar algún algoritmo bajo el supuesto de que $\mathcal{P} \neq \mathcal{NP}$.

Teorema 3.17. *Para todo $\varepsilon > 0$, no existe una $\left(\frac{e}{e-1} - \varepsilon\right)$-aproximación para el problema $R|\text{split,setup}|C_{\text{máx}}$, salvo que $\mathcal{P} = \mathcal{NP}$.*

Demostración. Usaremos una reducción desde el problema Max-k-Cover: dado un conjunto de elementos $U = \{e_1, \ldots, e_m\}$ y $S_1, \ldots, S_n \subseteq U$, encontrar k de tales subconjuntos que maximicen el número de elementos cubiertos, es decir, encontrar $A \subseteq \{1, \ldots, n\}$ con $|A| = k$ tal que $|\bigcup_{\ell \in A} S_\ell|$ sea máximo. Feige [8] probó que es $\mathcal{NP}$-duro distinguir entre aquellas instancias en las cuales los elementos pueden ser cubiertos por k subconjuntos disjuntos y aquellas instancias en las cuales no hay k subconjuntos que cubran más de una fracción $(1 - 1/e + \delta)$ de los elementos, para cualquier $\delta > 0$. Este resultado se mantiene cierto incluso si consideramos instancias en donde $|S_\ell| = m/k$, para todo $\ell \in \{1, \ldots, k\}$.

Dada una instancia de Max-k-Cover donde cada subconjunto posee cardinal m/k, construimos una instancia para $R|\text{split,setup}|C_{\text{máx}}$ como sigue: se tienen n trabajos, uno por cada subconjunto S_j, se tiene un conjunto M_U de $n-k$ máquinas *universales*, en donde cada trabajo j tiene tiempo de instalación 1 y tiempo de procesamiento 0. Por cada elemento se tiene una máquina *representante* e_i, en donde $s_{e_i j} = 0$ y $p_{e_i j} = m/k = |S_j|$ si $e_i \in S_j$, y $s_{e_i j} = 2$ y $p_{e_i j} = m/k = |S_j|$ si $e_i \notin S_j$. Consideremos una programación con makespan menor a 2. Esto es posible si $n-k$ trabajos son asignados a las máquinas universales y los k trabajos restantes son asignados a las máquinas representantes. El makespan dependerá de los elementos cubiertos por los conjuntos representados por los trabajos. Primero notemos que si uno de estos trabajos es asignado a una máquina que representa un elemento fuera del conjunto entonces el makespan sería automáticamente mayor o igual a 2. Por un lado, si los k conjuntos son una partición $\{e_1, \ldots, e_m\}$ con cardinal m/k cada uno, entonces cada uno de estos trabajos puede ser asignado fraccionariamente sobre cada uno de los elementos que cubre, de manera equitativa. Dado que cada máquina universal procesa exactamente un trabajo y cada máquina representante es asignada con una fracción k/m, con tiempo de procesamiento m/k y tiempo de instalación 0, se tendrá que el makespan en este caso es exactamente 1. Por otro lado, si los k conjuntos restantes cubren una fracción menor o igual a $(1-1/e+\delta)$ de los elementos, tenemos que cubrir un tiempo de procesamiento total igual a $m/k \cdot k = m$ sobre $(1-1/e+\delta)m$ máquinas representantes, lo cual en el mejor caso nos entrega un makespan igual a $\frac{m}{(1-1/e+\delta)m} = \frac{e}{e-1} - \varepsilon$. Luego, es $\mathcal{NP}$-duro distinguir aquellas instancias que tienen makespan 1 y aquellas que tienen makespan $\frac{e}{e-1} - \varepsilon$, para todo $\varepsilon > 0$.

□

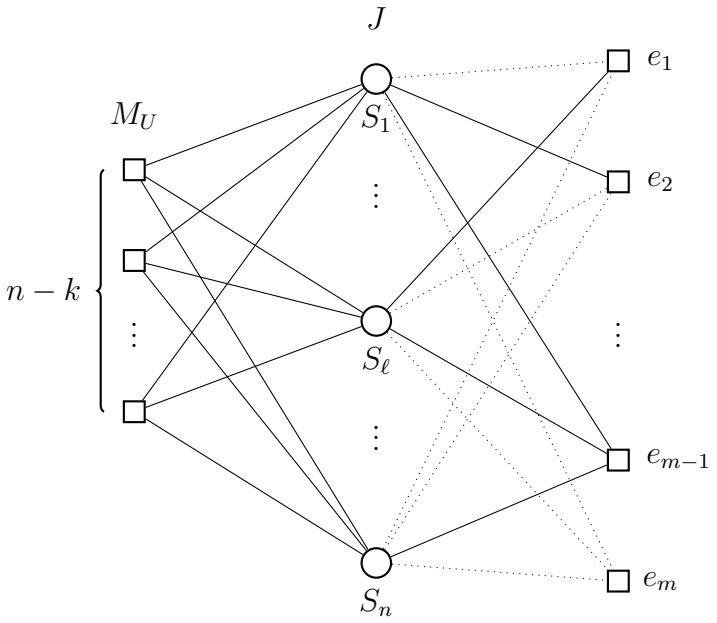

Figura 3.5: Gadget para la reducción desde Max-k-Cover.

3.7. PL de configuraciones

3.7.1. PL de configuraciones para las máquinas

Al igual que antes, consideremos el problema de decisión asociado donde cada máquina posee C^* unidades de tiempo disponibles para procesar trabajos. Una *configuración* B para una máquina corresponde a un vector $x^B \in [0,1]^J$ donde la coordenada x_j^B corresponde a la fracción del trabajo j que es asignada en esa máquina. Sea $\mathcal{B}_i$ el conjunto de configuraciones factibles para la máquina i, es decir, $B \in \mathcal{B}_i$ si y solo si $\sum_{j:x_j^B>0}(x_j^B p_{ij} + s_{ij}) \leq C^*$. Sea ρ_B una variable que indica si la configuración $B \in \bigcup_{i\in M} \mathcal{B}_i$ es usada por alguna máquina. Usando estas variables se tiene la siguiente relajación para el problema:

$$
\begin{aligned}
[\text{MCLP}]: \qquad \sum_{B\in\mathcal{B}_i} \rho_B &\leq 1 && \text{para todo } \ i \in M,\\
\sum_{i\in M}\sum_{B\in\mathcal{B}_i} x_j^B \rho_B &\geq 1 && \text{para todo } \ j \in J,\\
\rho_B &\geq 0 && \text{para todo } \ B \in \bigcup_{i\in M} \mathcal{B}_i.
\end{aligned}
$$

La primera restricción asegura que a lo más una configuración se asigna a cada máquina $i \in M$. Si para alguna máquina se tuviera que $s_{ij} > C^*$ para todo $j \in J$, entonces $\mathcal{B}_i = \emptyset$ y por lo tanto no se asignaría ninguna configuración a esa máquina. La segunda restricción asegura que cada trabajo es asignado completamente.

Este tipo de LP son una herramienta potente que permite muchas veces mejorar el gap de integralidad de las formulaciones estándar, como las usadas en las secciones anteriores. Sin embargo, el siguiente teorema muestra que en este caso no se mejora el gap de integralidad, y de hecho es igual al de $[\text{LST}_{\text{strong}}]$.

Teorema 3.18. *Para todo $\varepsilon > 0$, existe una instancia del problema $R|split,setup|C_{\text{máx}}$ tal que $opt/C^* \geq 1 + \phi - \varepsilon$, donde C^* es el menor racional tal que* [MCLP] *es factible.*

Demostración. Sea $\beta = \phi - 1$ y seleccionemos k, G, d enteros tales que $\left(1 - \frac{1}{k}\right)\frac{d}{G} \geq \beta$ y $\frac{G}{d} \geq \frac{1}{\beta} - \varepsilon$. Construimos la instancia I_k como sigue: se tienen k grupos disjuntos de G trabajos cada uno, $J_\ell = \{j_\ell^1, \ldots, j_\ell^G\}$, para $\ell \in \{1, \ldots, k\}$. Para cada $j \in \bigcup_{\ell=1}^k J_\ell$ se tiene una máquina hija $i_c(j)$. Para cada grupo $\ell \in \{1, \ldots, k\}$ existe una máquina padre $i_p(j_\ell^1), \cdots, i_p(j_\ell^G)$. Finalmente, existe un trabajo j^t, que corresponde a la raíz del árbol que representa la instancia (ver Figura 3.6). Los tiempos de procesamiento e instalación son los siguientes:

$$p_{ij} = \begin{cases} 0 & \text{para } j \in \bigcup_{\ell=1}^k J_\ell, i = i_p(j), \\ \frac{1}{1-\beta} & \text{para } j \in \bigcup_{\ell=1}^k J_\ell, i = i_c(j), \\ 0 & \text{para } j = j^t, i = i_p(j) \text{ para algún } j \in \bigcup_{\ell=1}^k J_\ell, \\ \infty & \text{en otro caso.} \end{cases}$$

$$s_{ij} = \begin{cases} \frac{1}{d} & \text{para } j \in \bigcup_{\ell=1}^k J_\ell, i = i_p(j), \\ 0 & \text{para } j \in \bigcup_{\ell=1}^k J_\ell, i = i_c(j), \\ 1 & \text{para } j = j^t, i = i_p(j') \text{ con } j' \in \bigcup_{\ell=1}^k J_\ell, \\ \infty & \text{en otro caso.} \end{cases}$$

□

Primero probaremos que toda solución óptima para la instancia posee makespan al menos $1 + \phi - \varepsilon$. Sea $i_p(j_\ell^i)$ la máquina que recibe el trabajo j^t en una solución óptima (este trabajo no se divide pues $p_{ij^t} = 0$ para todo i). Los trabajos $j_\ell^1, \ldots, j_\ell^G$ tampoco se dividen en una solución óptima. Luego, si alguno de estos trabajos es asignado a su máquina hija el makespan será $1/(1-\beta) = 1 + \phi$. En otro caso, $j_\ell^1, \ldots, j_\ell^G$ son asignados a la máquina i_p^ℓ, la cual tendrá una carga total igual a $1 + G/d \geq 1 + 1/\beta - \varepsilon = 1 + \phi - \varepsilon$.

Ahora veremos que [MCLP] es factible para $C^* = 1$. Dado que el tiempo de procesamiento de j^t es 0 y su tiempo de instalación es 1 es todas las máquinas padre, tenemos que la configuración B_t^i en donde el único trabajo asignado a i_p^i es j^t, el cual se asigna completo a esa máquina, es factible para todas las máquinas padre $i_p^1, \ldots, i_p^k$. Sea $\rho_{B_t^i} = 1/k$ para todo $i \in \{1, \ldots, k\}$. Notemos que esto asignará fraccionariamente el trabajo j^t sobre todas las máquinas padre, y en cada una de ellas habrá una fracción $1 - 1/k$ de espacio restante para asignar otros trabajos. Resta por asignar los trabajos en $\bigcup_{\ell=1}^k J_\ell$. Dado un trabajo $j \in \bigcup_{\ell=1}^k J_\ell$, la configuración que asigna una fracción $1 - \beta$ a su máquina hija es factible para dicha máquina, pues el tiempo de procesamiento para esa máquina $1/(1-\beta)$ y el tiempo de instalación es 0, es decir, la carga total sobre la máquina es igual a 1. Sea $\rho_{B_j} = 1$ para cada $j \in \bigcup_{\ell=1}^k J_\ell$, donde B_j es la configuración de la máquina $i_c(j)$ tal que $x_j^{B_j} = 1 - \beta$ y $x_r^{B_j} = 0$ para todo los otros trabajos $r \neq j$. Luego, resta por asignar una fracción β para cada trabajo en $\bigcup_{\ell=1}^k J_\ell$. Sea ℓ uno de los grupos de trabajos. Para asignar lo que resta, usaremos el espacio disponible para configuraciones en la máquina padre i_p^ℓ. Como los trabajos tienen tiempo de procesamiento 0 y tiempo de instalación $1/d$ en i_p^ℓ, existen $\binom{G}{d}$ maneras de formar una configuración en la cual se completen d trabajos

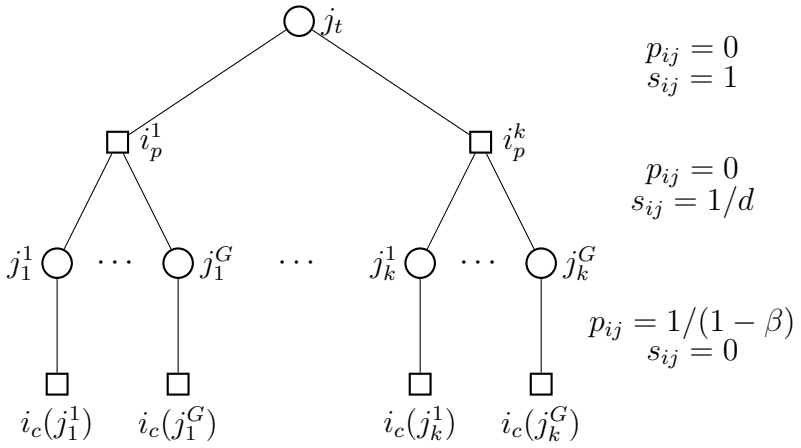

Figura 3.6: Ejemplo que muestra que el mejor factor posible para [MCLP] es $1 + \phi$.

del grupo ℓ. Llamemos $B_{i_p^\ell}$ tales configuraciones. Como el espacio restante en la máquina i_p^ℓ para asignar configuraciones es $1 - 1/k$, asignaremos $\rho_B = (1 - 1/k)/\binom{G}{d}$ para cada $B \in B_{i_p^\ell}$. Notemos que un trabajo $j \in J_\ell$ aparece en $\binom{G-1}{d-1}$ configuraciones $B_{i_p^\ell}$, y por lo tanto la fracción total asignada de j es:

$$\left(1 - \frac{1}{k}\right) \frac{\binom{G-1}{d-1}}{\binom{G}{d}} = \left(1 - \frac{1}{k}\right) \frac{d}{G} \geq \beta,$$

donde la última desigualdad es consecuencia de la elección hecha sobre los valores de d, k y G. Realizando este proceso sobre cada grupo se concluye con una asignación para [MCLP] factible para $C^* = 1$, y por lo tanto $\text{opt}(I_k)/C^*(I_k) \geq 1 + \phi - \varepsilon$.

3.7.2. PL de configuraciones para trabajos

Como se vio en la sección anterior, el PL de configuraciones para las máquinas tiene un gap de integralidad igual al de la relajación [$\text{LST}_{\text{strong}}$], lo que no permite usar [MCLP] para encontrar un algoritmo de aproximación con un factor menor a $1+\phi$. Es por ello que en vez de trabajar con las configuraciones sobre las máquinas, usaremos configuraciones de los trabajos. En este contexto, una configuración f para un trabajo j especifica la fracción de trabajo que es procesada en cada máquina. La configuración consiste en dos vectores $x^f \in [0,1]^M$, $y^f \in \{0,1\}^M$ tal que $\sum_{i \in M} x_i^f = 1$ e $y_i^f = 1$ si y solo si $x_i^f > 0$. La carga debido al trabajo j en la máquina i corresponde a $t_i^f = p_{ij} x_i^f + s_{ij} y_i^f$. Si $\mathcal{F}_j$ es el conjunto de configuraciones factibles para el trabajo j, entonces $f \in \mathcal{F}_j$ si y solo si $t_i^f \leq C^*$ para todo $i \in M$. Luego, cualquier programación factible se corresponde con alguna solución entera de

$$
\begin{aligned}
[\text{CLP}]: \qquad \sum_{f\in\mathcal{F}_j} \lambda_f &= 1 && \text{para todo } \ j \in J,\\
\sum_{j\in J}\sum_{f\in\mathcal{F}_j} \lambda_f t_i^f &\le C^* && \text{para todo } \ i \in M,\\
\lambda_f &\ge 0 && \text{para todo } \ f \in \bigcup_{j\in J}\mathcal{F}_j.
\end{aligned}
$$

Esta relajación posee infinitas variables, sin embargo, en el siguiente teorema se demuestra que podemos restringirnos a una clase especial de configuraciones $\mathcal{F}_j^*$ llamadas *maximales*: una configuración $f \in \mathcal{F}_j$ es *maximal* si existe a lo más una máquina $i \in M$ con $0 < x_i < x_{ij}^{\text{máx}} := (C^* - s_{ij})/p_{ij}$. Este conjunto de configuraciones es finito, pues depende de las máquinas en el soporte de la configuración y de aquella posible máquina que no alcanza el valor $x_{ij}^{\text{máx}}$.

Teorema 3.19. [CLP] *es factible si y solo si la restricción de* [CLP] *a las configuraciones maximales es factible.*

Demostración. Consideremos el dual [D] de [CLP]:

$$
\begin{aligned}
[\text{D}]: \quad \text{mín} \quad & C^*\sum_{i\in M}\delta_i + \sum_{j\in J}\mu_j,\\
\text{s.t.} \quad & \mu_j + \sum_{i\in M} t_i^f\delta_i \ge 0 && \text{para todo } j\in J, f\in\mathcal{F}_j,\\
& \delta_i \in \mathbb{R}_+,\ \mu_j \in \mathbb{R} && \text{para todo } i\in M, j\in J.
\end{aligned}
$$

Sabemos que [CLP] es factible si y solo si [D] es acotado [25, Teorema 2.2]. Vamos a probar que toda restricción de [D] está implicada por alguna restricción de [CLP] correspondiente a una configuración maximal. Esto implicaría que [D] es un poliedro. Sea $\delta \in \mathbb{R}_+^M$ y $\mu \in \mathbb{R}^J$ una solución infactible para [D]. Notemos que existe $f \in \mathcal{F}_j$ para algún j tal que $\mu_j + \sum_{i\in M} t_i^f\delta_i < 0$. Si f no es maximal, entonces existen al menos dos máquinas i, k tales que $0 < x_i < x_{ij}^{\text{máx}}$ y $0 < x_k < x_{kj}^{\text{máx}}$. Sin pérdida de generalidad, supongamos que $p_{ij}\delta_i \le p_{kj}\delta_k$ y consideremos la configuración f' que se obtiene al trasladar la mayor cantidad posible del trabajo j desde la máquina k a la máquina i:

$$
x_\ell^{f'} := \begin{cases} \text{mín}\{x_{ij}^{\text{máx}},\ x_i^f + x_k^f\} & \text{if } \ell = i,\\ \text{máx}\{0,\ x_k^f + x_i^f - x_{ij}^{\text{máx}}\} & \text{if } \ell = k,\\ x_\ell^f & \text{en otro caso.}\end{cases}
$$

Si $x_{ij}^{\text{máx}} > x_i^f + x_k^f$ entonces $x_i^{f'} = x_i^f + x_k^f$ y $x_k^{f'} = 0$. Luego, en este caso se tiene

$$\begin{aligned} t_i^{f'}\delta_i + t_k^{f'}\delta_k &= p_{ij}\delta_i x_i^{f'} + s_{ij}\delta_i y_i^{f'} + p_{kj}\delta_k x_k^{f'} + s_{kj}\delta_k y_k^{f'} \\ &\leq p_{ij}\delta_i \min\{x_{ij}^{\text{máx}}, x_i^f + x_k^f\} + s_{ij}\delta_i y_i^f \\ &\quad + p_{kj}\delta_k \max\{0, x_i^f + x_k^f - x_{ij}^{\text{máx}}\} + s_{kj}\delta_k y_k^f \\ &= p_{ij}\delta_i(x_i^f + x_k^f) + s_{ij}\delta_i y_i^f + s_{kj}\delta_k y_k^f \\ &= p_{ij}\delta_i x_i^f + s_{ij}\delta_i y_i^f + p_{ij}\delta_i x_k^f + s_{kj}\delta_k y_k^f \\ &= p_{ij}\delta_i x_i^f + s_{ij}\delta_i y_i^f + p_{kj}\delta_k x_k^f + s_{kj}\delta_k y_k^f \\ &= t_i^f\delta_i + t_k^f\delta_k. \end{aligned}$$

Si $x_{ij}^{\text{máx}} \leq x_i^f + x_k^f$ entonces $x_i^{f'} = x_{ij}^{\text{máx}}$ y $x_k^{f'} = x_i^f + x_k^f - x_{ij}^{\text{máx}}$, y por lo tanto

$$\begin{aligned} t_i^{f'}\delta_i + t_k^{f'}\delta_k &= p_{ij}\delta_i x_i^{f'} + s_{ij}\delta_i y_i^{f'} + p_{kj}\delta_k x_k^{f'} + s_{kj}\delta_k y_k^{f'} \\ &= p_{ij}\delta_i x_{ij}^{\text{máx}} + s_{ij}\delta_i y_i^f + p_{kj}\delta_k(x_i^f + x_k^f - x_{ij}^{\text{máx}}) + s_{kj}\delta_k y_k^f \\ &= t_i^f\delta_i + t_k^f\delta_k + p_{ij}\delta_i(x_{ij}^{\text{máx}} - x_i^f) + p_{kj}\delta_k(x_i^f - x_{ij}^{\text{máx}}) \\ &= t_i^f\delta_i + t_k^f\delta_k + (p_{ij}\delta_i - p_{kj}\delta_k)(x_{ij}^{\text{máx}} - x_i^f) \\ &\leq t_i^f\delta_i + t_k^f\delta_k. \end{aligned}$$

Luego, se tiene que

$$\begin{aligned} \mu_j + \sum_{r\in M} t_r^{f'}\delta_r &= \mu_j + t_i^{f'}\delta_i + t_k^{f'}\delta_k + \sum_{r\neq i,k} t_r^{f'}\delta_i \\ &= \mu_j + t_i^{f'}\delta_i + t_k^{f'}\delta_k + \sum_{r\neq i,k} t_r^f\delta_i \\ &\leq \mu_j + t_i^f\delta_i + t_k^f\delta_k + \sum_{r\neq i,k} t_r^f\delta_i \\ &= \mu_j + \sum_{r\in M} t_r^f\delta_r < 0. \end{aligned}$$

Si iteramos este proceso obtenemos una configuración que será maximal y tal que la desigualdad que define es violada por (μ, δ). Luego, [D] queda completamente descrito en base a las configuraciones maximales y en consecuencia [CLP] tiene una solución factible si y solo si la restricción de [CLP] a las configuraciones maximales posee una solución factible. □

El Teorema 3.19 permite restringir [CLP] sin pérdida de generalidad a un número finito de variables, sin embargo, este valor sigue siendo exponencial en el tamaño de la entrada. Con el objetivo de resolver el PL en tiempo polinomial, discretizaremos [CLP]. Dado $\varepsilon > 0$, el discretizar [CLP] (al que llamaremos $[\text{CLP}]^d$) nos permitirá tener lo siguiente: si [CLP] es factible, entonces $[\text{CLP}]^d$ es también factible, y si $[\text{CLP}]^d$ es factible entonces [CLP] es factible para configuraciones con makespan $(1+\varepsilon)C^*$.

Dado un trabajo j, el conjunto de configuraciones discretizadas, al que denotamos F_j^d, poseerá todas las configuraciones tales que x_i^f es múltiplo de ε/m, es decir, $f \in F_j^d$ si

$x_i^f = k_i \cdot \varepsilon/m$ con $k_i \in \{0, \ldots, \lfloor m/\varepsilon \rfloor\}$, e $y_i^f = 1$ si y solo si $x_i^f > 0$. También relajamos la restricción de asignación para el trabajo, es decir, aceptamos configuraciones tales que $\sum_i x_i^f \geq 1 - \varepsilon$. Luego, $[\text{CLP}]^d$ corresponde al siguiente LP:

$$
\begin{aligned}
[\text{CLP}]^d\text{:} \qquad \sum_{f \in \mathcal{F}_j^d} \lambda_f &= 1 && \text{para todo } j \in J,\\
\sum_{j \in J} \sum_{f \in \mathcal{F}_j} \lambda_f t_i^f &\leq C && \text{para todo } i \in M,\\
\lambda_f &\geq 0 && \text{para todo } f \in \bigcup_{j \in J} \mathcal{F}_j^d.
\end{aligned}
$$

Lema 3.20. *Si* [CLP] *es factible para* C^**, entonces* $[\text{CLP}]^d$ *es factible para* C^**. Además, si* $[\text{CLP}]^d$ *es factible para* C^**, entonces* [CLP] *es factible para* $(1+\varepsilon)C^*$*.*

Demostración. Dada una configuración $f \in \mathcal{F}_j$ definimos $\tilde{f}$ tal que $x_i^{\tilde{f}} = \left\lfloor \frac{x_i^f m}{\varepsilon} \right\rfloor \frac{\varepsilon}{m}$ e $y_i^{\tilde{f}} = y_i^f$ para toda $i \in M$. Nótese que

$$
\begin{aligned}
x_i^f - \frac{\varepsilon}{m} &= \left(\frac{x_i^f m}{\varepsilon} - 1 \right) \frac{\varepsilon}{m} \leq \left\lfloor \frac{x_i^f m}{\varepsilon} \right\rfloor \frac{\varepsilon}{m} = x_i^{\tilde{f}},\\
x_i^{\tilde{f}} &= \left\lfloor \frac{x_i^f m}{\varepsilon} \right\rfloor \frac{\varepsilon}{m} \leq \frac{x_i^f m}{\varepsilon} \cdot \frac{\varepsilon}{m} = x_i^f.
\end{aligned}
$$

Luego, tenemos que $\sum_i x_i^{\tilde{f}} \geq \sum_i \left(x_i^f - \frac{\varepsilon}{m} \right) \geq \sum_i x_i^f - \varepsilon \geq 1 - \varepsilon$, y por lo tanto $\tilde{f} \in \mathcal{F}_j^d$. Recíprocamente, dada una solución factible de $[\text{CLP}]^d$ podemos definir $x_i^f(\varepsilon) = x_i^{\tilde{f}}/(1-\varepsilon) = x_i^{\tilde{f}}(1 + \mathcal{O}(\varepsilon))$ para todas las configuraciones $f \in \mathcal{F}_j^d$, y por lo tanto esto nos entrega una solución factible con makespan $(1 + \mathcal{O}(\varepsilon))C$. □

El siguiente paso es probar que $[\text{CLP}]^d$ puede ser resuelto en tiempo polinomial. En el contexto de $[\text{CLP}]^d$, una configuración $f \in \mathcal{F}_j^d$ será maximal si y solo si existe a lo más una máquina $i \in M$ tal que $0 < x_i^f < x_{ij}^{\text{máx}^d} := \left\lfloor \frac{(C - s_{ij})m}{p_{ij}\varepsilon} \right\rfloor \frac{\varepsilon}{m}$.

Lema 3.21. *El programa lineal* $[\text{CLP}]^d$ *puede ser resuelto en tiempo polinomial. Además, si* $[\text{CLP}]^d$ *es factible entonces existe una solución tal que todas las configuraciones en el soporte son maximales.*

Demostración. Gracias al lema de Farkas, $[\text{CLP}]^d$ es factible si y solo si el siguiente PL es infactible:

$$
\begin{aligned}
0 &> C^* \sum_{i \in M} \delta_i + \sum_{j \in J} \mu_j,\\
0 &\leq \mu_j + \sum_{i \in M} t_i^f \delta_i && \text{para todo } j \in J, f \in \mathcal{F}_j^d,\\
\delta_i &\in \mathbb{R}_+,\ \mu_j \in \mathbb{R} && \text{para todo } i \in M, j \in J.
\end{aligned}
$$

Para determinar la factibilidad de este programa dual usamos la equivalencia entre separación y optimización [12]. Dada una solución (μ, δ), el problema de separación puede ser resuelto fijando $j \in J$ y resolviendo

$$[P_j] := \text{mín} \left\{ \sum_{i \in M} t_i^f \delta_i : f \in \mathcal{F}_j^d \right\}.$$

Al igual que en el Teorema 3.19 existe una solución óptima al problema que corresponde a una configuración maximal. Luego, el algoritmo para resolver este problema es el siguiente: adivinamos la máquina $i^* \in M$ tal que $0 < x_{i^*j} < x_{i^*j}^{\text{máx}^d}$ además de la fracción correspondiente a $x_{i^*}^f$. Recordemos que $x_{i^*}^f$ puede tomar valores en un conjunto de tamaño $\lfloor m/\varepsilon \rfloor + 1$, y por lo tanto este proceso se puede realizar en tiempo $\mathcal{O}(m(\lfloor m/\varepsilon \rfloor + 1)) = \mathcal{O}(m^2/\varepsilon)$, el cual es polinomial. Luego, dada una máquina i^* y $x_{i^*}^f$, el problema $[P_j]$ se reduce al siguiente problema:

$$[KC_{i^*}] = \min_{S \subseteq M \setminus \{i^*\}} \left\{ \sum_{i \in S} (x_{ij}^{\text{máx}^d} p_{ij} + s_{ij}) \delta_i : \sum_{i \in S} x_{ij}^{\text{máx}^d} \geq 1 - \varepsilon - x_{i^*}^f \right\}.$$

Este problema corresponde a una instancia de *Knapsack-Cover*, y por lo tanto puede ser resuelto en tiempo pseudo-polinomial. En este caso será polinomial en el tamaño del input, pues los valores $x_i^f, x_{ij}^{\text{máx}^d}$ son de la forma $k_i \cdot \varepsilon/m$ con $k_i \in \{0, \ldots, \lfloor m/\varepsilon \rfloor\}$. □

3.7.3. Proyección de [CLP] al espacio de asignaciones

Consideremos el siguiente PL,

$$[\text{CLP}_{\text{proj}}]: \quad \sum_{j \in J} (p_{ij} x_{ij} + s_{ij} y_{ij}) \leq C \qquad \text{para todo } \ i \in M, \tag{3.16}$$

$$\sum_{i \in M} (\alpha_i x_{ij} + \beta_i y_{ij}) \geq M(j, \alpha, \beta) \quad \text{para todo } \ j \in J, \ \alpha, \beta \in \mathbb{R}_+^M, \tag{3.17}$$

donde $M(j, \alpha, \beta) := \text{mín}\{\sum_{i \in M} (\alpha_i x_i^f + \beta_i y_i^f) : f \in \mathcal{F}_j\}$. Veremos que cualquier solución λ de [CLP] puede ser proyectada en el poliedro definido por $[\text{CLP}_{\text{proj}}]$ mediante la siguiente transformación:

$$(x_{ij}^\lambda, y_{ij}^\lambda) = \left(\sum_{f \in \mathcal{F}_j} \lambda_f x_i^f, \sum_{f \in \mathcal{F}_j} \lambda_f y_i^f \right).$$

Teorema 3.22. *Si* $\lambda \in [\text{CLP}]$ *entonces* $(x^\lambda, y^\lambda) \in [\text{CLP}_{\text{proj}}]$. *Y si* $(x, y) \in [\text{CLP}_{\text{proj}}]$, *entonces existe* $\lambda \in [\text{CLP}]$ *tal que* $(x^\lambda, y^\lambda) = (x, y)$.

Demostración. Sea $\lambda \in [\text{CLP}]$ y sean $\alpha, \beta \in \mathbb{R}_+^M$. Por definición de $M(j, \alpha, \beta)$ tenemos que para todo $j \in J$

$$\sum_{i \in M} \alpha_i x_{ij} + \beta_i y_{ij} = \sum_{i \in M} \left(\alpha_i \sum_{f \in \mathcal{F}_j} \lambda^f x_i^f + \beta_i \sum_{f \in \mathcal{F}_j} \lambda^f y_i^f \right) \geq M(j, \alpha, \beta).$$

Por otro lado, de la factibilidad de λ obtenemos que (x^λ, y^λ) satisface la otra restricción de [CLP_{proj}],

$$\sum_{j\in J}(p_{ij}x_{ij}+s_{ij}y_{ij}) = \sum_{j\in J}\sum_{f\in\mathcal{F}_j}\left(p_{ij}\lambda^f x_i^f + s_{ij}\lambda^f y_i^f\right) = \sum_{j\in J}\sum_{f\in\mathcal{F}_j}\lambda^f t_i^f \le C.$$

Consideremos ahora $(x,y)\in[\text{CLP}_{\text{proj}}]$. Tenemos que existe $\lambda\in[\text{CLP}]$ tal que $(x,y)=(x^\lambda,y^\lambda)$ si y solo si el siguiente LP es no vacío:

$$\begin{aligned}
\sum_{f\in\mathcal{F}_j} x_i^f\lambda_f &= x_{ij} && \text{para todo } i\in M,\ j\in J,\\
\sum_{f\in\mathcal{F}_j} y_i^f\lambda_f &= y_{ij} && \text{para todo } i\in M,\ j\in J,\\
\sum_{f\in\mathcal{F}_j}\lambda_f &= 1 && \text{para todo } j\in J,\\
\sum_{j\in J}\sum_{f\in\mathcal{F}_j} t_i^f\lambda_f &\le C && \text{para todo } i\in M,\\
\lambda_f &\ge 0 && \text{para todo } f\in\bigcup_{j\in J}\mathcal{F}_j.
\end{aligned}$$

Esto será cierto si y solo si el dual es acotado:

$$\begin{aligned}
\text{mín}\quad & \sum_{i\in M}\sum_{j\in J}(x_{ij}\alpha_{ij}+y_{ij}\beta_{ij}) + \sum_{j\in J}\mu_j + C\sum_{i\in M}\delta_i &&\\
\text{s.t.}\quad & \sum_{i\in M}\left(x_i^f\alpha_{ij}+y_i^f\beta_{ij}\right)+\mu_j+\sum_{i\in M}t_i^f\delta_i \ \ge\ 0 && \text{para todo } j\in J,\ f\in\mathcal{F}_j,\\
& \delta_i \ \ge\ 0 && \text{para todo } i\in M.
\end{aligned}$$

Usando las desigualdades 3.16 y 3.17 podemos acotar inferiormente la expresión que se minimiza en el dual,

$$\begin{aligned}
& \sum_{i\in M}\sum_{j\in J}(x_{ij}\alpha_{ij}+y_{ij}\beta_{ij}) + \sum_{j\in J}\mu_j + C\sum_{i\in M}\delta_i\\
\ge\ & \sum_{i\in M}\sum_{j\in J}\big(x_{ij}(\alpha_{ij}+p_{ij}\delta_i)+y_{ij}(\beta_{ij}+s_{ij}\delta_i)\big)+\sum_{j\in J}\mu_j\\
\ge\ & \sum_{j\in J}\Big(M\big(j,\ (\alpha_{ij}+p_{ij}\delta_i)_{i\in M},\ (\beta_{ij}+s_{ij}\delta_i)_{i\in M}\big)+\mu_j\Big).
\end{aligned}$$

Las restricciones del dual implican que la última expresión es no negativa, y por lo tanto el dual es acotado. □

Para concluir esta sección, veremos que al introducir una cierta clase de desigualdades de [CLP_{proj}] en [$\text{LST}_{\text{strong}}$] la colección de instancias que alcanzan el gap de esta última relajación ya no serán factibles. Dado un subconjunto de máquinas $S\subseteq M$, definimos $L(j,S) := \sum_{i\in M\setminus S}\max\{(C-s_{ij})/p_{ij},0\}$ como la mayor fracción de j que puede ser procesada en las máquinas que no están en S.

Lema 3.23. *Sea (x, y) solución en $[\mathrm{LST}_{\mathrm{strong}}]$ inducida por una programación al problema de decisión asociado a $R|split,setup|C_{\mathrm{máx}}$ con makespan a lo más C. Entonces el siguiente conjunto de desigualdades son válidas:*

$$\frac{\sum_{i\in S'} x_{ij}}{1 - L(j, S \cup S')} + \sum_{i\in S} y_{ij} \geq 1 \quad \textit{para todo } j \in J \textit{ y } S, S' \subseteq M \textit{ con } L(j, S \cup S') < 1.$$

Demostración. Tenemos que cualquier solución factible (x, y) debe procesar una fracción de al menos $1 - L(j, S \cup S')$ en las máquinas $S \cup S'$. Luego, si una máquina en S es usada para procesar j, el lado izquierdo de la desigualdad es mayor o igual 1, y entonces la desigualdad es válida. Si ninguna máquina en S es usada para procesar j entonces $\sum_{i\in S'} x_{ij} = \sum_{i\in S\cup S'} x_{ij} \geq 1 - L(j, S \cup S')$, y por lo tanto también la desigualdad es válida en este caso. □

Estas desigualdades se pueden ver como un tipo especial de las 3.17 en $[\mathrm{CLP}_{\mathrm{proj}}]$: basta tomar $\alpha_i = 1/(1 - L(j, S \cup S'))$ para $i \in S'$ y $\beta_i = 1$ para $i \in S$. Además, si consideramos la colección de instancias usadas en el Teorema 3.13, notemos que $L(j, \{i_p(j)\}) = C/p_{i_c(j)j} < 1$, y por lo tanto $y_{i_p(j)j} = 1$ para todo $j \in J \cup J'$ en cualquier solución factible de $[\mathrm{CLP}_{\mathrm{proj}}]$. Luego, si $C < 1 + \phi$ la relajación $[\mathrm{CLP}_{\mathrm{proj}}]$ será infactible para esta colección de instancias. El mismo argumento se sigue para la colección de instancias mostradas en el Teorema 3.18. A pesar que los resultados obtenidos entorno a la estructura del PL de configuraciones de trabajos son prometedores, no se sabe como redondear este PL. Determinar el gap de integralidad para esta relajación es una pregunta que permanece abierta tras este capítulo.

Bibliografía

[1] A. Allahverdi, C. Ng, T. Cheng, and M. Kovalyov. A survey of scheduling problems with setup times or costs. *European Journal of Operational Research*, 187:985–1032, 2008.

[2] S. Arora. Polynomial time approximation schemes for euclidean traveling salesman and other geometric problems. *Journal of the ACM*, 45:753–782, 1998.

[3] A. Asadpour, U. Feige, and A. Saberi. Santa claus meets hypergraph matchings. *ACM Transactions on Algorithms*, 8:24:1–24:9, 2012.

[4] K. Baker. *Introduction to Sequencing and Scheduling*. John Wiley & Sons, 1974.

[5] N. Bansal and M. Sviridenko. The santa claus problem. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing*, pages 31–40, 2006.

[6] B. Chen, Y. Ye, and J. Zhang. Lot-sizing scheduling with batch setup times. *Journal of Scheduling*, 9:299–310, 2006.

[7] T. Ebenlendr, M. Krčál, and J. Sgall. Graph balancing: A special case of scheduling unrelated parallel machines. *Algorithmica*, doi: 10.1007/s00453-012-9668-9, 2012.

[8] U. Feige. A threshold of $\ln n$ for approximating set cover. *Journal of the ACM*, 45:314–318, 1998.

[9] U. Feige. On allocations that maximize fairness. In *Proceedings of the 19th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 287–293, 2008.

[10] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., 1979.

[11] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. R. Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. volume 5 of *Annals of Discrete Mathematics*, pages 287–326. Elsevier, 1979.

[12] M. Grötschel, L. Lovász, and A. Schrijver. Geometric algorithms and combinatorial optimization, 1988.

[13] B. Haeupler, B. Saha, and A. Srinivasan. New constructive aspects of the Lovasz Local Lemma. *Journal of the ACM*, 58:28:1–28:28, 2011.

[14] O. H. Ibarra and C. E. Kim. Fast approximation algorithms for the knapsack and sum of subset problems. *Journal of the ACM*, 22:463–468, 1975.

[15] D.-W. Kim, D.-G. Na, and F. Frank Chen. Unrelated parallel machine scheduling with setup times and a total weighted tardiness objective. *Robotics and Computer-Integrated Manufacturing*, 19:173–181, 2003.

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

[17] J. K. Lenstra, D. B. Shmoys, and E. Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Mathematical Programming*, 46:259–271, 1990.

[18] Z. Liu and T. C. E. Cheng. Minimizing total completion time subject to job release dates and preemption penalties. *Journal of Scheduling*, 7:313–327, 2004.

[19] N. Megow and J. Verschae. Dual techniques for scheduling on a machine with varying speed. In *Automata, Languages, and Programming*, volume 7965 of *Lecture Notes in Computer Science*, pages 745–756. 2013.

[20] L. Polacek and O. Svensson. Quasi-polynomial local search for restricted max-min fair allocation. In *Automata, Languages, and Programming*, number 7391 in Lecture Notes in Computer Science, pages 726–737. 2012.

[21] C. N. Potts and L. N. V. Wassenhove. Integrating scheduling with batching and lot sizing: A review of algorithms and complexity. *The Journal of Operational Research Society*, 43:395–406, 1992.

[22] F. Schalekamp, R. Sitters, S. van der Ster, L. Stougie, V. Verdugo, and A. van Zuylen. Split scheduling with uniform setup times. *CoRR*, abs/1212.1754, 2012.

[23] P. Schuurman and G. Woeginger. Preemptive scheduling with job-dependent setup times. In *Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete algorithms*, pages 759–767, 1999.

[24] P. Serafini. Scheduling jobs on several machines with the job splitting property. *Operations Research*, 44:617–628, 1996.

[25] A. Shapiro. Semi-infinite programming, duality, discretization and optimality conditions. *Optimization*, 58:133–161, 2009.

[26] W. Smith. Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, 3:59–66, 1956.

[27] O. Svensson. Santa claus schedules jobs on unrelated machines. *SIAM Journal on Computing*, 41:1318–1341, 2012.

[28] S. van der Ster. The allocation of scarce resources in disaster relief, 2010. Tesis de Master en Operations Research, VU University Amsterdam.

[29] V. V. Vazirani. *Approximation algorithms*. Springer, 2001.

[30] J. Verschae and A. Wiese. On the configuration-LP for scheduling on unrelated machines. In *Proceedings of the 19th European Symposium on Algorithms*, pages 530–542, 2011.

[31] D. P. Williamson and D. B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, 2011.

[32] W. Xing and J. Zhang. Parallel machine scheduling with splitting jobs. *Discrete Applied Mathematics*, 103:259–269, 2000.