Capítulo II

SCHEDULING

2.1. INTRODUCCIÓN

El scheduling es una tarea extremadamente difícil con una importante necesidad de cálculo [21]. Los problemas de scheduling se pueden identificar en distintas áreas de aplicación. Diversos ítems están sujetos a scheduling, tal como operaciones de producción en una industria de manufactura, procesamiento computacional en un sistema operativo, movimiento de camiones en transporte, etc. La gran importancia práctica convierte al scheduling en un área activa de investigación.

Los problemas de scheduling son problemas de optimización combinatoria. La función del scheduling es la asignación de recursos limitados a tareas a lo largo del tiempo. Tiene como finalidad la optimización de uno o más objetivos.

Los recursos y las tareas pueden tomar muchas formas. Los recursos pueden ser máquinas en un taller, pistas en un aeropuerto, ladrillos en una construcción, unidades de procesamiento en un ambiente computacional, etc. Como tareas se pueden tener operaciones de un proceso de producción, despegues y aterrizajes en un aeropuerto, etapas de un proyecto de construcción, ejecuciones de un programa de computación, etc. Cada tarea puede tener diferentes niveles de prioridad, tiempos de posibles inicios, etc. Los objetivos pueden tomar varias formas: uno posible es minimizar los tiempos de finalización de la última tarea, otro minimizar el número de tareas luego de una fecha de entrega acordada, etc.

Los problemas de scheduling en la práctica poseen estructuras de problemas más complejas, pero en situaciones reales pueden ser relevantes diferentes restricciones, tal como planes de procesamiento alternativos para la fabricación de un producto, estructuras de producción especializadas, etc.

El scheduling puede ser un problema difícil desde el punto de vista técnico como de implementación [115]. El tipo de dificultades encontradas en los aspectos técnicos son similares a las encontradas en otras ramas de optimización combinatoria y modelado

estocástico. Las dificultades encontradas desde el punto de vista de la implementación son de distintas clases y están relacionadas al modelado de problemas de scheduling de mundo real y la recuperación de información.

Analizar un problema de scheduling y desarrollar un procedimiento para tratar con el problema es sólo una parte. El procedimiento tiene que estar embebido en un sistema que habilite la aplicación del scheduler. El sistema de scheduling se tiene que incorporar en el sistema de información de la empresa u organización, lo cual puede ser una tarea considerable.

2.2. NOTACIÓN

En todos los problemas de scheduling considerados, el número de máquinas es finito. El número de jobs se identifica con *n* y el número de máquinas con *m*. Usualmente, el subíndice *j* hace referencia a un job, mientras el subíndice *i*, a una máquina. Si un job necesita varios pasos de procesamiento u operaciones, entonces el par (*i*,*j*) indica la operación (paso de procesamiento) del job *j* sobre la máquina *i*. Con cada job *j* se asocian los siguientes datos:

- ✓ *Tiempo de procesamiento (p_{ij}).* Representa el tiempo de procesamiento del job *j* sobre la máquina *i*. El subíndice *i* se omite si el tiempo de procesamiento del job *j* no depende de la máquina o si el job *j* sólo se procesa en una determinada máquina.
- ✓ *Fecha de release (r_j).* La fecha de *release* r_j del job *j* es la fecha de listo. Es el tiempo en el que el job *j* arriba al sistema, es decir, es el tiempo más temprano en el cual se puede iniciar su procesamiento.
- ✓ *Fecha de finalización (d_j).* La fecha de finalización d_j del job *j* representa el *committed shipping* (fecha de terminación, embarque o entrega), la fecha en la cual se promete el job al cliente. Se permite la finalización de un job después de su fecha de terminación, pero se penaliza. Cuando se debe cumplir con la fecha de terminación esto se denomina como un *deadline*.
- ✓ *Peso (w_j).* El peso w_j del job *j* es básicamente un factor de prioridad, indicando la importancia relativa del job *j* con respecto a otros jobs del sistema. Por ejemplo, este peso puede representar el costo actual de mantener el job en el sistema.

2.2.1. DESCRIPCIÓN DE UN PROBLEMA DE SCHEDULING

Un problema de scheduling se describe por $\alpha|\beta|\gamma$. El campo α describe el ambiente de máquina y contiene una única entrada. El campo β provee detalles de las características de procesamiento y restricciones; puede tener una única entrada, múltiples entradas o ninguna entrada. El campo γ identifica el objetivo a ser minimizado y usualmente contiene una única entrada.

2.2.1.1. CAMPO α

Los posibles ambientes de máquina que se pueden especificar en el campo α son [95]:

- ✓ *Máquina única* (1). El caso de máquina única es el ambiente de máquina más simple y es el caso especial de todos los demás ambientes de máquina.
- ✓ *Máquinas idénticas en paralelo* (P_m). Hay *m* máquinas idénticas en paralelo. El job *j* necesita una única operación y se puede procesar en cualquiera de las *m* máquinas o sobre alguna de un conjunto especificado. Si no se permite el procesamiento del job *j* sobre cualquier máquina o sobre sólo alguna perteneciente a un subconjunto dado (el subconjunto M_j), entonces en el campo β aparece la entrada M_j.
- ✓ *Máquinas en paralelo con diferentes velocidades* (Q_m)**.** Hay *m* máquinas en paralelo con diferentes velocidades; v_i indica la velocidad de la máquina *i*. El tiempo de permanencia p_{ij} del job *j* en la máquina *i* es p_j/v_i, asumiendo que sólo se procesa en la máquina *i*. Este ambiente también se lo conoce como máquinas *uniformes*. Si todas las máquinas tienen la misma velocidad, es decir $v_i = 1$ para todo *i* y $p_{ij} = p_j$, entonces este ambiente es idéntico al previo.
- ✓ *Máquinas no relacionadas en paralelo* (R_m). Este ambiente es una generalización del anterior. Hay *m* máquinas diferentes en paralelo. La máquina *i* puede procesar el job *j* a una velocidad v_{ij}. El tiempo de permanencia p_{ij} del job *j* sobre la máquina *i* es p_j/v_{ij}, asumiendo que sólo se procesa sobre la máquina *i*. Si la velocidad de las máquinas son independientes de los jobs, es decir $v_i = v_{ij}$ para todo *i* y *j*, entonces el ambiente es idéntico al anterior.
- ✓ *Flow shop* (*Fm*). Hay *m* máquinas en serie, cada job se procesa en cada una de ellas. Todos los jobs tienen la misma trayectoria, es decir, primero se procesan sobre la máquina 1, luego sobre la máquina 2, y así sucesivamente. Cuando un job deja de

usar una máquina se agrega a la cola de la próxima máquina. Usualmente, se asume que todas las colas trabajan bajo la disciplina *primero en entrar primero en salir (FIFO)*. Si se aplica la disciplina FIFO, al flow shop se lo denomina como flow shop con permutación y el campo β incluye la entrada *prmu.*

- ✓ *Flexible flow shop* (*FFs*). Es una generalización del flow shop y del ambiente de máquinas paralelas. En lugar de *m* máquinas en serie, hay *s* etapas en serie con una determinada cantidad de máquinas en paralelo en cada una de ellas. Cada job se procesa primero en la etapa 1, luego en la etapa 2, y así siguiendo. Cada etapa funciona como un banco de máquinas paralelas; en cada etapa el job *j* necesita sólo una máquina y cualquier máquina puede procesar cualquier job. Las colas entre etapas distintas trabajan bajo una disciplina FIFO.
- ✓ *Open Shop* (*Om*) . Hay *m* máquinas y *n* jobs. Cada job se debe procesar en cada una de las *m* máquinas. Sin embargo, algunas de esos tiempos de procesamiento puede ser cero. No hay restricción en relación al ruteo de cada job a través del ambiente de máquinas. El scheduler determina la ruta de cada job, diferentes jobs pueden tener distintas rutas.
- ✓ *Job shop* (*Jm*). Hay *m* máquinas y *n* jobs, cada job tiene predeterminada su ruta. Se hace una distinción entre job shops donde cada job puede visitar alguna máquina al menos una vez y aquellos donde un job puede visitar una máquina más de una vez. En el último caso, el campo β contiene la entrada *recrc* para indicar *recirculación*.

2.2.1.2.CAMPO β

Las restricciones de procesamiento especificadas en el campo β pueden incluir múltiples entradas. Las entradas posibles son:

- ✓ *Release time* (r_j). Si este símbolo está presente en el campo β, el job *j* no puede empezar su procesamiento antes de su release date r_j. Si r_j no aparece en el campo β, el procesamiento del job *j* puede comenzar en cualquier momento. En contraste con el release dates, los tiempos de entrega no se especifican en este campo. El tipo de función objetivo da información suficiente si se consideran tiempos de entrega o no.
- ✓ *Tiempos de setup dependientes de la secuencia* (s_{jk}). s_{jk} representa el tiempo de setup dependiente de la secuencia de los jobs *j* y *k*; s_{0k} indica el tiempo de setup para el job *k* si es el primero en la secuencia y s_{j0} hace referencia al tiempo de

clean-up luego del job *j* si éste es el último de la secuencia (tanto s_{0k} como s_{j0} pueden ser 0). Si el tiempo de setup entre los jobs *j* y *k* depende de la máquina, entonces se debe incluir el subíndice *i*, es decir s_{ijk}. Si s_{0k} no aparece en el campo β, se asume que todos los tiempos de setup son cero o independientes de la secuencia, en cuyo caso se pueden simplemente incorporar a los tiempos de procesamiento.

- ✓ *Restricciones de precedencia* (*prec*)**.** Las restricciones de precedencia pueden aparecer en un ambiente de máquina única o máquinas paralelas, exigiendo que uno o más jobs tengan que finalizar antes que otros comiencen su procesamiento. Hay varias formas de restricciones de precedencia. Si cada job tiene al menos un antecesor y un sucesor, las restricciones se denominan *cadenas*. Si cada job tiene al menos un sucesor, las restricciones se denominan *intree*. Si cada job tiene al menos un antecesor, las restricciones se denominan *outtree*. Si en el campo β no aparece *prec*, los jobs no están sujetos a restricciones de precedencia.
- ✓ *Permutaciones* (*prmu*)**.** Una restricción que puede aparecer en el ambiente flow shop es que la cola de cada máquina trabaje respetando la disciplina FIFO. Esto implica que el orden (o permutación) en la cual se asignan los jobs a la primer máquina se debe respetar en todas las demás.
- ✓ *Recirculación* (*recrc*). Puede ocurrir en job shop, cuando un job puede visitar más de una vez una máquina.

2.2.1.3.CAMPO γ

El objetivo es minimizar una función de tiempos de finalización de los jobs, los cuales dependen del schedule. El tiempo de finalización de la operación de un job *j* sobre la máquina *i* se denota como C_{ij}. El tiempo en el que el job *j* sale del sistema se indica como C_j. La función objetivo puede ser una función de tiempos de entrega. La *lateness* de un job *j* se define como:

$$L_j = C_j - d_j,$$

la cual es positiva cuando el job *j* finaliza tarde y negativo cuando se completa en forma temprana. La *tardiness* de un job *j* se define como:

$$T_j = \max(C_j - d_j, 0) = \max(L_j, 0)$$

La diferencia entre tardiness y lateness es que la primera nunca es negativa. La *unidad de penalidad* del job *j* se define como:

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

Lateness, tardiness y unidad de penalidad son las tres funciones de penalidad básicas relacionadas con tiempos de entrega.

Los siguientes son algunos ejemplos de funciones objetivos a ser minimizadas:

- ✓ *Makespan* (C_{max}). Se define como el max(C_1, C_2, …, C_n). Es equivalente al tiempo de finalización del último job en dejar el sistema. Un makespan mínimo usualmente implica una alta utilización de las máquinas.
- ✓ *Lateness máxima* (L_{max})**.** Se define como el max(L_1, L_2, …, L_n). Mide la peor violación de los tiempos de entrega.

2.2.2. EJEMPLOS

Los siguientes ejemplos ilustran la notación:

- ✓ $Jm \mid\mid C_{max}$, denota un problema de job shop con *m* máquinas. No hay recirculación, de modo que cada job puede visitar sólo una una vez cada máquina. El objetivo es minimizar el makespan.
- ✓ $Pm \mid prec \mid L_{max}$, denota un ambiente de máquinas paralelas idénticas con restricciones de precedencia entre los jobs y optimización del lateness máximo.

2.3. CLASES DE SCHEDULES

Se puede hacer una distinción entre una *secuencia*, un *schedule*, y una *política de scheduling*. Una secuencia es usualmente una permutación de un conjunto de jobs o un orden en el cual se deben procesar los jobs sobre una determinada máquina. Un schedule es una asignación de jobs a un ambiente de máquinas más complicados. El concepto de políticas de scheduling se usa frecuente en ambientes estocásticos: una política prescribe una acción apropiada para alguno de los estados en el que el sistema puede estar. En modelos determinísticos es importante las secuencias o schedules.

Hay distintas clases de schedules [72, 66]: *nondelay, activo y semiactivo*.

Un schedule factible es *nondelay* si ninguna máquina permanece ociosa mientras exista una operación disponible para su procesamiento.

Exigiendo que un schedule sea nondelay es equivalente a prohibir *unforced idleness*.

Ciertos procedimientos y algoritmos heurísticos se basan sobre la generación de schedules con propiedades especiales. Los schedules activos y semiactivos son importantes para la generación del schedule en procedimientos algorítmicos.

Un schedule factible es *activo* si no se puede adelantar la finalización de una operación por alteración de la secuencia de procesamiento sin que se retrase otra operación.

Un schedule nondelay es un schedule activo pero no es cierto lo inverso. El siguiente ejemplo describe un schedule que es activo pero no nondelay.

Ejemplo. Sea un problema de job shop con tres máquinas y dos jobs. El job 1 necesita una unidad de tiempo sobre la máquina 1 y tres sobre la máquina 2. El job 2 necesita dos unidades sobre la máquina 3 y tres sobre la máquina 2. La máquina 2 es la última en procesar ambos jobs. Sea el schedule que procesa el job 2 sobre la máquina 2 antes que el job 1, (figura 2.1). Este schedule es activo; invirtiendo la secuencia de los dos procesos sobre la máquina 2 se pospone el procesamiento del job 2. Sin embargo, el schedule no es un schedule nondelay. La máquina 2 permanece ociosa hasta el tiempo 2, mientras que hay un job disponible para procesamiento en el tiempo 1.

Un schedule factible se llama *semiactivo* si ninguna operación puede finalizar tempranamente sin alterar la secuencia de procesamiento de alguna de las máquinas.

Ejemplo. Sea un problema de job shop con tres máquinas y dos jobs [115]. Las rutas de los dos jobs son las mismas que en el ejemplo previo. El tiempo de procesamiento del job 1 sobre la máquina 1 y la máquina 2 es el mismo. El tiempo de procesamiento del job 2 sobre la máquina 2 y la máquina 3 es de dos. Sea el schedule bajo el cual el job 2 se procesa sobre la máquina 2 antes que el job1 (figura 2.2). Esto implica que el job 2 comienza su procesamiento sobre la máquina 2 en el tiempo dos y el job 1 comienza su procesamiento sobre la máquina 2 en el tiempo cuatro. Este schedule es semiactivo. Sin embargo, es no activo: el job 1 se puede procesar sobre la máquina 2 sin retrasar el procesamiento del job 2 sobre la misma máquina.

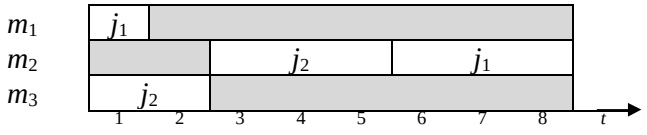


Figura 2.1 Schedule activo.

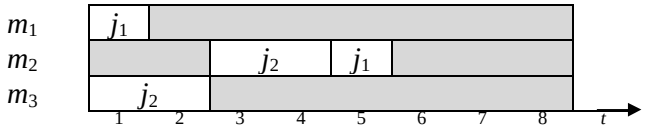


Figura 2.2 Schedule semiactivo.

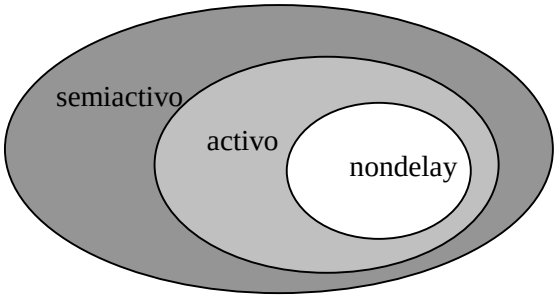


Figura 2.3 Relación entre las clases de schedules.

Se puede construir un ejemplo de schedule que no es semiactivo. Por ejemplo, posponer el comienzo del procesamiento del job 1 sobre la máquina 2 por una unidad de tiempo, es decir, la máquina 2 permanece ociosa por una unidad de tiempo entre el procesamiento del job 2 y del 1. Este schedule es no semiactivo.

La relación entre los distintos schedules se presenta en la figura 2.3. Un schedule óptimo está dentro del conjunto de schedules activos. Los schedules nondelay son más pequeños que los schedules activos, pero no hay garantía que el primero contenga el óptimo [18].

2.4. JERARQUÍA DE COMPLEJIDAD

Frecuentemente, un algoritmo para un problema de scheduling se puede aplicar a otros. Por ejemplo, $1||\sum C_j$ es un caso especial de $1||\sum w_j C_j$, y un procedimiento para $1||\sum w_j C_j$ se puede usar para $1||\sum C_j$ [115]. En terminología de complejidad se dice que $1||\sum C_j$ se reduce a $1||\sum w_j C_j$ y se indica como:

$$1||\sum C_j \propto 1||\sum w_j C_j$$

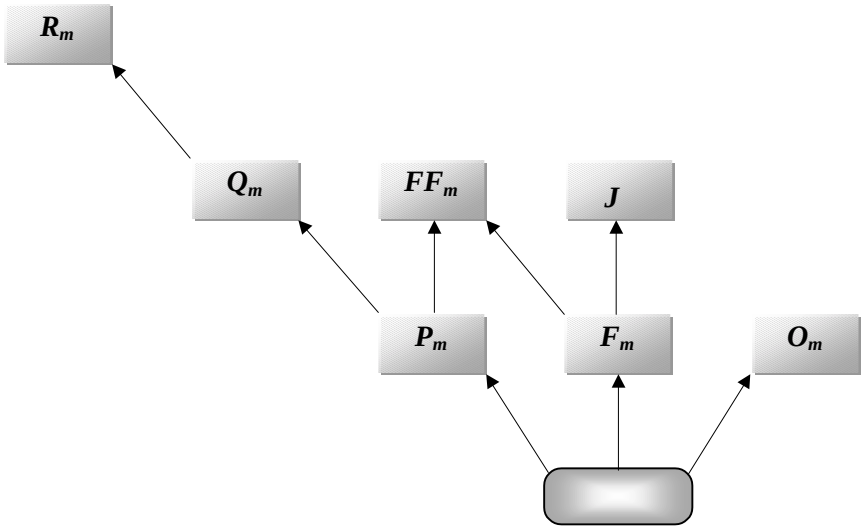
Basándose en este concepto, se puede establecer una cadena de reducciones. Por ejemplo,

$$1||\sum C_j \propto 1||\sum w_j C_j \propto P_m||\sum w_j C_j \propto Q_m|prec|\sum w_j C_j$$

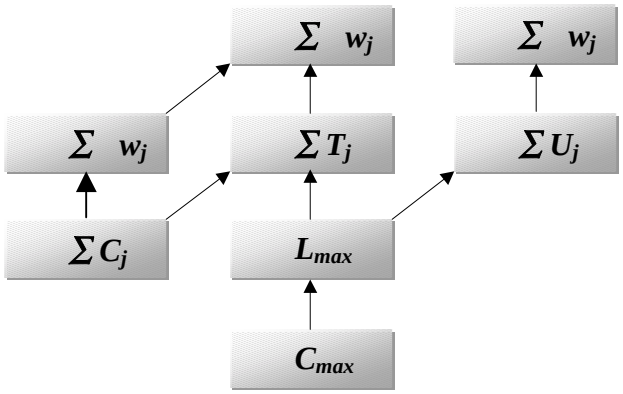
Hay varios problemas que no se pueden comparar entre sí. Por ejemplo, $P_m \parallel \sum w_j C_j$ y $J_m \parallel C_{max}$

Al comparar las complejidades de los diferentes problemas de scheduling, es de interés conocer cómo un cambio en un único elemento en la clasificación de un problema afecta su complejidad [117, 98, 93, 96]. En la figura 2.4, se muestra la jerarquía de complejidad de los problemas de scheduling determinístico [115].

Ha habido un gran desarrollado de investigación en scheduling determinístico para hallar un algoritmo eficaz que encuentre en tiempo polinomial una buena solución a los problemas de scheduling. Sin embargo, muchos problemas de scheduling no tienen algoritmos con tiempos polinomiales, por lo tanto esos problemas son problemas NP-duros.



(a) ambiente de máquina.



(b) funciones objetivo.

Figura 2.4 Jerarquía de complejidad de problemas de scheduling determinístico.

2.5. JOB SHOP SCHEDULING

Entre los ambientes de máquina presentados en la sección 2.1.1, este trabajo se enfoca en el problema de job shop scheduling clásico, el cual es uno de los problemas de scheduling más conocidos y el que con mayor frecuencia se encuentra dentro de un ambiente de producción. Se puede describir como sigue: hay *m* máquinas diferentes y *n* jobs diferentes a ser asignados. Cada job está compuesto por un conjunto de operaciones y el orden de las operaciones sobre las máquinas está preestablecido. Cada operación se caracteriza por la máquina requerida y el tiempo de procesamiento. Hay varias restricciones sobre jobs y máquinas:

- ✓ Un job no puede visitar una misma máquina dos veces.
- ✓ No hay restricciones de precedencia entre operaciones de distintos jobs.
- ✓ Las operaciones no se pueden interrumpir.
- ✓ Cada máquina puede procesar sólo un job a la vez.
- ✓ No se especifican ni *release times* (fecha de job listo a ser procesado)ni *due dates* (fecha de entrega).

El problema es determinar la secuencia de operaciones sobre las máquinas con el objetivo de minimizar el *makespan*, es decir, el tiempo necesario para completar todos los jobs.

El problema de job shop scheduling es uno de los problemas de optimización más duros. Debido a su dificultad para abordarlos, los procedimientos heurísticos son una alternativa muy atractiva. Los procedimientos heurísticos convencionales usan *reglas de prioridades*, es decir, una regla para elegir una operación desde un subconjunto especificado de operaciones aún no planificadas. Una de las técnicas más usadas para resolver este tipo de problema es la de búsqueda local, tal como simulated annealing, tabu search y algoritmos evolutivos. En el capítulo 7 se presentan distintas alternativas de algoritmos evolutivos para obtener soluciones para el problema de job shop scheduling.