Material de apoyo Teórica VI

Temario

Seguimos trabajando con problemas combinatorios:

- Problema de la mochila
- Problema de scheduling (calendarización o secuenciamiento de tareas)

Problemas combinatorios

- Son aquellos en los cuales se desea determinar combinaciones óptimas.
- Se caracterizan por tener un número finito de soluciones factibles.
- Generalmente este número es muy grande.

Problema de la mochila

Introducción

El problema de la mochila es un problema simple de entender: hay una persona que tiene una mochila con una cierta capacidad y tiene que elegir que elementos pondrá en ella. Cada uno de los elementos tiene un peso y aporta un beneficio. El objetivo de la persona es elegir los elementos que le permitan maximizar el beneficio sin excederse de la capacidad permitida.

A la vez es un problema complejo, si por complejidad nos referimos a la computacional. Un problema se cataloga como inherentemente difícil si su solución requiere de una cantidad significativa de recursos computacionales, sin importar el algoritmo utilizado[1]. El problema de la mochila forma parte de una lista histórica de problemas NP Completos elaborada por Richard Karp en 1972[2].

En el caso del problema de la mochila, si contáramos con 4 productos, para saber cuál es la mejor solución podríamos probar las 2^4=16 posibilidades. El 2 se desprende del hecho de que cada decisión es incluir o no al producto y el 4 de la cantidad de productos. 16 posibilidades es un número manejable, sin embargo, si la cantidad de elementos por ejemplo ascendiera a 20, tendríamos que analizar nada más y nada menos que 220=1048576 posibilidades

Formulación lineal

Una de las técnicas matemáticas que se puede utilizar para la resolución de este problema es la programación lineal.

Definiendo a

- *c* como la capacidad de la mochila
- *pi* como el beneficio unitario obtenido por ingresar el producto *i* en la
- mochila
- *wi* como el peso del producto *i.*
- *n* como la cantidad de productos
- *c, pi* y *wi* como valores enteros y positivos

se puede plantear el modelo como:

$$\sum_{i=1}^{n} w_i x_i \leq c \qquad \text{(Capacidad)}$$

$$max \rightarrow z = \sum_{i=1}^{n} p_i x_i \qquad \text{(Funcional)}$$

No obstante, el problema que tiene esta técnica es que no siempre se puede resolver el problema debido a la complejidad matemática del problema. En esas ocasiones, es necesario recurrir a heurísticas. Una heurística es un procedimiento que en la mayoría de las ocasiones nos permite obtener una buena solución pero que no necesariamente es la óptima.

Variantes

Existen distintas variaciones[3] que se han realizado al problema de la mochila estándar. A continuación se presenta una breve introducción a algunas de ellas.

Acotado:

En esta variante, en vez de contar con solamente un elemento de cada tipo, se cuenta con una cantidad limitada y conocida de cada uno de los elementos (que no necesariamente es la misma). De cada objeto i hay disponible una cantidad bi, así que agregamos otra constante en el modelo.

Las variables Xi ya no son binarias, son enteras y representan la cantidad de objetos de tipo i que hay en la mochila.

Es decir, el modelo pasa a ser:

$$x_i \leq b_i \qquad (\forall i = 1..n)$$

$$\sum_{i=1}^{n} w_i x_i \leq c$$

$$max \rightarrow z = \sum_{i=1}^{n} p_i x_i$$

No acotado:

El problema de la mochila no acotado es una instancia particular del problema de la mochila acotado en el cual cada *bi* = 1

Suma de subconjuntos:

Existe una variante en la cual el beneficio es igual al peso para cada uno de los elementos. Esto no implica que el beneficio (o peso) sea el mismo para cada uno de los elementos

$$\sum_{i=1}^{n} w_i x_i \leq c$$

$$max \rightarrow z = \sum_{i=1}^{n} w_i x_i$$

Problema del cambio:

Hay un caso particular que aparece cuando se exige cumplir exactamente con la restricción de capacidad. En este caso además se plantea que *pi=1* ya que se

intenta reconstruir la situación de un cajero que debe maximizar o minimizar la cantidad de monedas que entrega de cambio.

$$\sum_{i=1}^{n} w_i x_i = c$$

$$max \rightarrow z = \sum_{i=1}^{n} x_i$$

Múltiples mochilas:

Otra de las variantes conocidas surge de la generalización del problema estándar, cuando en vez de tener un solo contenedor (mochila), se tienen varios.

$$\sum_{j=1}^{m} x_{ij} \leq 1 \qquad (\forall\ i)$$

$$\sum_{i=1}^{n} w_i x_{ij} \leq c_j \qquad (\forall\ j)$$

$$max \rightarrow z = \sum_{i=1}^{m} \sum_{i=1}^{n} p_i x_{ij}$$

[1] Teoría de la complejidad computacional. Wikipedia

[2] KARP, Richard. Reducibility Among Combinatorial Problems. 1972

[3] MARTELLO, S. and TOTH, P. Knapsack Problems: Algorithms and Computer Implementation. 1990

Calendarización (Scheduling)

Introducción

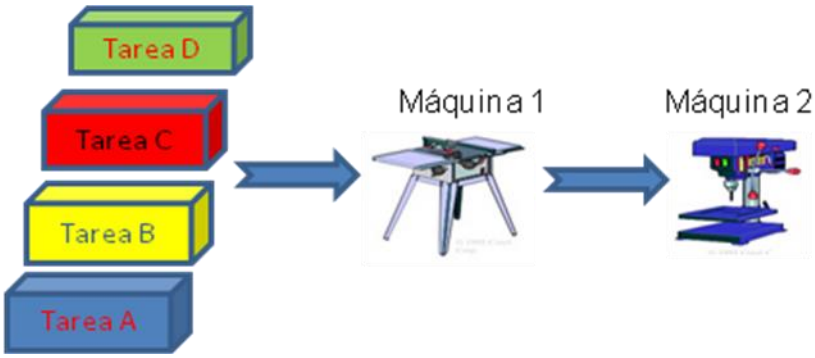
El estudio formal de los problemas de calendarización se remonta a la década del 50'. En ese entonces, industrias de diferente índole intentaban resolver el problema de cómo llevar adelante la realización de múltiples tareas teniendo recursos (máquinas) limitados. Oliver Wight identifica dos conceptos clave en lo que refiere a los problemas de calendarización orientados a la producción: prioridades y capacidades. En definitiva, qué deberá hacerse primero y quién debería hacerlo. La respuesta a estas dos preguntas podía marcar la diferencia entre una empresa competitiva y una que no lo era.
El hecho de que sea un problema al que se le encontraban tantas aplicaciones prácticas, impulsó su estudio y la aplicación de distintas técnicas para resolverlo. En la década del 70' se demostró que ciertos problemas de calendarización, pertenecen al grupo de problemas NP-Hard. Debido al costo (y en algunos casos incluso la imposibilidad) de obtener soluciones óptimas, se prepararon heurísticas que aunque no conseguían la solución óptima, encontraban soluciones suficientemente buenas para lo que se necesitaba.

Ejemplo de problema de calendarización

Supongamos la existencia de N tareas a realizar, cada una con un tiempo de procesamiento que depende de la tarea. Cada una de las tareas se puede realizar en cualquiera de las M máquinas. No existen restricciones en lo que respecta a la precedencia de las tareas. El objetivo sería definir en qué momento se debe procesar cada tarea y en que máquina, para poder completar todas las tareas lo antes posible. Es decir, que la tarea que finaliza último, termine lo antes posible. Una de las técnicas matemáticas que se puede utilizar para la resolución de este problema es la programación lineal.

Un ejemplo práctico

Trabajaremos con un caso en el cual tenemos cuatro tareas (A, B, C y D) que tienen que pasar primero por la máquina 1 y luego por la máquina 2.



Los tiempos de procesamiento de cada tarea en las máquinas son:

Tarea	Tiempo en máquina 1	Tiempo en máquina 2
A	3 minutos	2 minutos
B	6 minutos	8 minutos
C	5 minutos	6 minutos
D	7 minutos	4 minutos

Cuando se empieza a procesar una tarea en una máquina no se interrumpe (sigue hasta que se termine)

Queremos tardar lo menos posible en terminar todas las tareas

¿Y si lo planteamos así?:

Tiempomáquina1 = 3 + 6 + 5 + 7 = 21 minutos
Tiempomáquina2 = 2 + 8 + 6 + 4 = 20 minutos

¡En 21 minutos terminamos!

Pero no estamos asegurando que una tarea se procese en la máquina 2 cuando ya se haya terminado de procesar en la máquina 1

Necesitamos saber en qué momento empieza cada tarea en cada máquina (y en qué momento termina, para que pueda pasar a la otra máquina):

Iij: Minuto en que empieza la tarea i en la máquina j
Fij: Minuto en el cual finaliza la tarea i en la máquina j
(como las tareas no se interrumpen Fij = Iij + Tiempodelatareaienmáqj)

Por ejemplo, para la tarea A:
FA1= IA1 + 3
FA2 = IA2 + 2
Y así para las demás tareas (B, C y D)
Para que una tarea no empiece en la máquina 2 antes de haber finalizado su procesamiento en la máquina 1 podemos hacer:

Por ejemplo, para la tarea A:
FA1 ≤ IA2
Y así para las demás tareas (B, C y D)

Para saber cuál es la tarea que termina más tarde podemos definir:

FINAL: Minuto en el cual finaliza la última tarea

FA2 ≤ FINAL

FB2 ≤ FINAL

FC2 ≤ FINAL

FD2 ≤ FINAL

Y la función objetivo es MIN FINAL

Resumiendo, nuestro modelo sería:

Fi1= Ii1 + Tiempoienmáq1
Fi2 = Ii2 + Tiempoienmáq2
Para i = A, B, C, D

Fi1 ≤ Ii2
Para i = A, B, C, D

FA2 ≤ FINAL; FB2 ≤ FINAL; FC2 ≤ FINAL; FD2 ≤ FINAL

MIN FINAL

Vamos a resolverlo con software y vemos cuál es la solución óptima:

```
MIN FINAL

ST
FA1 - IA1 = 3
FA2 - IA2 = 2
FB1 - IB1 = 6
FB2 - IB2 = 8
FC1 - IC1 = 5
FC2 - IC2 = 6
FD1 - ID1 = 7
FD2 - ID2 = 4
FA1 - IA2 < 0
FB1 - IB2 < 0
FC1 - IC2 < 0
FD1 - ID2 < 0
FA2 - FINAL < 0
FB2 - FINAL < 0
FC2 - FINAL < 0
FD2 - FINAL < 0

END
```

```
LP OPTIMUM FOUND AT STEP       5

        OBJECTIVE FUNCTION VALUE

        1)       14.00000

 VARIABLE          VALUE
    FINAL        14.000000
      FA1         3.000000
      IA1         0.000000
      FA2         5.000000
      IA2         3.000000
      FB1         6.000000
      IB1         0.000000
      FB2        14.000000
      IB2         6.000000
      FC1         5.000000
      IC1         0.000000
      FC2        14.000000
      IC2         8.000000
      FD1         7.000000
      ID1         0.000000
      FD2        14.000000
      ID2        10.000000
```

¡INICIA TODAS LAS TAREAS AL MISMO TIEMPO EN LA MÁQUINA 1!

¿Cómo podemos hacer para que no haga dos tareas al mismo tiempo en la misma máquina?

Por ejemplo, en la máquina 1, o hace la tarea A antes que la B, o hace la tarea B antes que la A

$FA1 \leq IB1$

ó

$FB1 \leq IA1$

¿Cómo hacemos para que de las dos restricciones el modelo anule una de las dos y deje la otra activa? Lo vimos hace dos semanas

$$FA1 \leq IB1 + M\ YanuloAB$$

$$FB1 \leq IA1 + M\ YanuloBA$$

$$YanuloAB + YanuloBA = 1$$

Entonces tenemos que agregar en el modelo el siguiente grupo de restricciones:

Para todas las tareas en la máquina 1:
$Fi1 \leq Ik1 + M\ Yanuloik$
$Fk1 \leq Ii1 + M\ Yanuloki$
$Yanuloik + Yanuloki = 1$
Para todo par de tareas i k

Y lo mismo en la máquina 2:
$Fi2 \leq Ik2 + M\ Yanulo2ik$
$Fk2 \leq Ii2 + M\ Yanulo2ki$
$Yanulo2ik + Yanulo2ki = 1$
Para todo par de tareas i k

El modelo, en el software, quedaría así (recordemos que como no existe el signo de mayor o menor estricto, cuando ponemos menor el software interpreta que es menor o igual):

```
MIN FINAL

ST
FA1 - IA1 = 3
FA2 - IA2 = 2
FB1 - IB1 = 6
FB2 - IB2 = 8
FC1 - IC1 = 5
FC2 - IC2 = 6
FD1 - ID1 = 7
FD2 - ID2 = 4
FA1 - IA2 < 0
FB1 - IB2 < 0
FC1 - IC2 < 0
FD1 - ID2 < 0
FA2 - FINAL < 0
FB2 - FINAL < 0
FC2 - FINAL < 0
FD2 - FINAL < 0
```

```
FA1 - IB1 - 30 Y1AB < 0
FB1 - IA1 - 30 Y1BA < 0
Y1AB + Y1BA = 1
FA1 - IC1 - 30 Y1AC < 0
FC1 - IA1 - 30 Y1CA < 0
Y1AC + Y1CA = 1
FA1 - ID1 - 30 Y1AD < 0
FD1 - IA1 - 30 Y1DA < 0
Y1AD + Y1DA = 1
FB1 - IC1 - 30 Y1BC < 0
FC1 - IB1 - 30 Y1CB < 0
Y1BC + Y1CB = 1
FB1 - ID1 - 30 Y1BD < 0
FD1 - IB1 - 30 Y1DB < 0
Y1BD + Y1DB = 1
FC1 - ID1 - 30 Y1CD < 0
FD1 - IC1 - 30 Y1DC < 0
Y1CD + Y1DC = 1
FA2 - IB2 - 30 Y2AB < 0
FB2 - IA2 - 30 Y2BA < 0
Y2AB + Y2BA = 1
FA2 - IC2 - 30 Y2AC < 0
FC2 - IA2 - 30 Y2CA < 0
Y2AC + Y2CA = 1
FA2 - ID2 - 30 Y2AD < 0
FD2 - IA2 - 30 Y2DA < 0
Y2AD + Y2DA = 1
FB2 - IC2 - 30 Y2BC < 0
FC2 - IB2 - 30 Y2CB < 0
Y2BC + Y2CB = 1
FB2 - ID2 - 30 Y2BD < 0
FD2 - IB2 - 30 Y2DB < 0
Y2BD + Y2DB = 1
FC2 - ID2 - 30 Y2CD < 0
FD2 - IC2 - 30 Y2DC < 0
Y2CD + Y2DC = 1
END
```

```
INT Y1AB
INT Y1BA
INT Y1AC
INT Y1CA
INT Y1AD
INT Y1DA
INT Y1BC
INT Y1CB
INT Y1BD
INT Y1DB
INT Y1CD
INT Y1DC
INT Y2AB
INT Y2BA
INT Y2AC
INT Y2CA
INT Y2AD
INT Y2DA
INT Y2BC
INT Y2CB
INT Y2BD
INT Y2DB
INT Y2CD
INT Y2DC
```

Y ahora lo vamos a volver a resolver para ver si ahora nos coloca las tareas en las máquinas de manera que no se superpongan dos tareas en la misma máquina:

```
      OBJECTIVE FUNCTION VALUE

      1)       25.00000

 VARIABLE         VALUE
     Y1AB          1.000000
     Y1BA          0.000000
     Y1AC          1.000000
     Y1CA          0.000000
     Y1AD          0.000000
     Y1DA          1.000000
     Y1BC          1.000000
     Y1CB          0.000000
     Y1BD          0.000000
     Y1DB          1.000000
     Y1CD          0.000000
     Y1DC          1.000000
     Y2AB          1.000000
     Y2BA          0.000000
     Y2AC          1.000000
     Y2CA          0.000000
     Y2AD          0.000000
     Y2DA          1.000000
     Y2BC          1.000000
     Y2CB          0.000000
     Y2BD          0.000000
     Y2DB          1.000000
     Y2CD          0.000000
     Y2DC          1.000000
    FINAL         25.000000
      FA1         14.000000
      IA1         11.000000
      FA2         21.000000
      IA2         19.000000
      FB1         11.000000
      IB1          5.000000
      FB2         19.000000
      IB2         11.000000
      FC1          5.000000
      IC1          0.000000
      FC2         11.000000
      IC2          5.000000
      FD1         21.000000
      ID1         14.000000
      FD2         25.000000
      ID2         21.000000
```

```
FINAL      25.000000
  FA1      14.000000
  IA1      11.000000
  FA2      21.000000
  IA2      19.000000
  FB1      11.000000
  IB1       5.000000
  FB2      19.000000
  IB2      11.000000
  FC1       5.000000
  IC1       0.000000
  FC2      11.000000
  IC2       5.000000
  FD1      21.000000
  ID1      14.000000
  FD2      25.000000
  ID2      21.000000
```

Entonces no superpone y termina en 25 minutos.

Notación

Tan grande es el conjunto de problemas de calendarización, que incluso se desarrolló una notación para identificarlos. Los problemas de calendarización se pueden especificar de la forma

$$\alpha \mid \beta \mid \gamma$$

El término **α** representa el entorno de las máquinas. Por ejemplo, si existiera una sola máquina se indicaría con un ***1***. Si existieran 3 máquinas idénticas, se marcaría ***P3***.

El segundo término, **β**, indica las características de las tareas. Si el trabajo en una tarea se pudiera interrumpir para que pueda ser continuado más adelante, agregaríamos ***pmtn*** en este término. Si existieran precedencias también serían detalladas en este campo. Es necesario mencionar que las características no son excluyentes. Es decir, podría armarse por ejemplo un problema de calendarización con precedencias y con interrupción de tareas.

Por último, en el término **γ** se incluye la función objetivo a resolver. La función objetivo no necesariamente será completar la última tarea lo antes posible (denominada ***Cmax***). Constituyen otros ejemplos de funciones objetivo:

- Minimizar la finalización de todas las tareas ponderadas por un peso.
- Si se estableciera una fecha de finalización esperada para cada tarea, podría minimizarse la demora o la cantidad de tareas

La notación correspondiente al problema identificado en la sección 1 es:

Pm | | Cmax

Aplicaciones

Ya se describió que la industria fue la que impulsó el estudio de los problemas de calendarización, aplicándolo principalmente para el caso en el cual los recursos eran máquinas. No obstante, con el tiempo el problema fue extendiéndose a nuevas áreas. El libro Handbook of Scheduling compila varias de estas aplicaciones[1].

Algunos de los ejemplos son:

- Asignación de horarios: El problema de asignar horarios a docentes que dan clases en un organismo educativo puede ser considerado como un problema de calendarización. Además de la complejidad intrínseca

asociada al aspecto matemático del problema, existen temas que tienen que ver con el hecho de que los recursos, no son máquinas sino que son humanos. No siempre se podrá satisfacer a todos los docentes con los horarios que ellos soliciten. En consecuencia, el hecho de que puede ser necesario llevar adelante una especie de negociación es un factor que se debe tener en cuenta.

- Industria aérea: Varios de los problemas que tienen que resolver las aerolíneas podrían catalogarse dentro del mundo de scheduling. Lo que distingue a esta industria es que muchas de las decisiones que debe tomar están interrelacionadas entre sí. Tiene que definir vuelos para satisfacer la demanda de pasajeros y asignar tripulaciones a estos vuelos, siempre teniendo en consideración las regulaciones legales y otras restricciones existentes. A diferencia del estudio de calendarización en máquinas donde se pudo por ejemplo definir una notación, el estudio en la industria de las aerolíneas esta menos estandarizado debido a la imposibilidad de generar un modelo común a estos problemas.

Referencias Scheduling

1. LEUNG, Joseph et al. Handbook of Scheduling. 2004.
2. HERRMANNN, Jeffrey et al. Handbook of Production Scheduling. 2006.
3. GRAHAM, Ronald et al. Optimization and approximation in deterministic sequencing and scheduling. 1979.
4. SOTSKOV, Yu y SHAKHLEVICH N.V. NP-hardness of shop-scheduling problems with three jobs. 1989.

¿Qué nos queda de esta clase?

- Seguimos trabajando con problemas combinatorios que vamos a encontrar en la Guía 3 (enteras)
  - Mochila / Knapsack
  - Secuenciamiento de tareas (Scheduling)