Minimum Makespan Scheduling and Scheduling on Identical Machines

Algoritmos de aproximación, PTAS y FPTAS

Rodrigo Arévalo Gaytán Fausto David Hernández Jasso

Universidad Nacional Autónoma de México

2023-06-21

Contenido

Contenido

Descripción del problema

- Sean $J_1, J_2, \ldots, J_n$ trabajos cuyos tiempos de procesamiento son $p_1, p_2, \ldots, p_n$ respectivamente y un entero positivo m *(el número de máquinas)*, encontrar un asignación de los n trabajos a las m máquinas tal que el tiempo que se tardan en completar éstos trabajos en las máquinas **(que es llamado makespan)** es mínimo.

Entendiendo el enunciado

- Las m máquinas son idénticas entre sí, es decir, tienen el mismo poder de cómputo.
- Los trabajos no pueden ser divididos entre las máquinas.
- Definimos al conjunto A_j como el conjunto de índices de los trabajos asignados a la máquina j con $1 \leq j \leq m$.

Entendiendo el enunciado

- Definimos a T_j como $\sum_{i \in A_j} p_i$, notemos que T_j el tiempo total de procesamiento que toma ejecutar todos los trabajos asignados a la máquina j.
- Lo que debemos de encontrar es una asignación de los trabajos a las máquinas de tal manera que el **makespan** sea mínimo, es decir buscamos minimizar **max** T_j, para toda $1 \leq j \leq m$.

Algoritmo 2 aproximado

Sean $J_1, J_2, \ldots, J_n$ trabajos

- Ordenamos los n trabajos de manera arbitraria. Es decir, los trabajos después de ordenarlos serán:

$$J_{\sigma(1)}, J_{\sigma(2)}, \ldots, J_{\sigma(n)}$$

  donde σ es una permutación del conjunto $\{1, 2, \ldots, n\}$.
- Ir ejecutando los n trabajos en las m máquinas en el nuevo orden de la siguiente manera:
  - Sea $J_{\sigma(i)}$ el siguiente trabajo a ejecutar con el orden obtenido en el paso anterior, $J_{\sigma(i)}$ se ejecutará en el procesador que tenga la menor carga de trabajo en ese momento.

Cota inferior

El algoritmo descrito anteriormente fue diseñado tomando en cuenta las siguientes **cotas inferiores** en la solución óptima, **OPT**.

- El tiempo de procesamiento promedio de todas las máquinas el cuál es: $\frac{1}{m}\sum_{i=1}^{n} p_i$.
- El tiempo de procesamiento más largo correspondiente a algún trabajo, es decir: $\mathbf{max}\{p_i\}$.

Así sea **LB** la cota inferior que considera ambos casos, y la definimos como sigue:

$$\mathbf{LB} = \mathbf{max}\left\{\frac{1}{m}\sum_{i=1}^{n} p_i, \mathbf{max}\{p_i\}\right\}$$

Teorema

- El algoritmo descrito anteriormente es un **algoritmo 2 aproximado**.

Demostración

- Sea T el **makespan óptimo**, entonces notamos que T cumple con las siguientes desigualdades:

$$T \geq \max\{p_i\}$$

$$T \geq \frac{1}{m}\sum_{j=1}^{m} T_j = \frac{1}{m}\sum_{i=1}^{n} p_i$$

- Es muy sencillo argumentar el porqué ambas desigualdades son verdaderas.

Demostración

Veamos $T \geq \mathbf{max}\{p_i\}$

- Por definición $T = \mathbf{max}\ T_j$, para alguna $1 \leq j \leq m$.
- De nuevo por definición tenemos que $T = \mathbf{max} \sum_{i \in A_j} p_i$ para alguna $1 \leq j \leq m$.

Demostración

Tenemos dos casos:

- $\mathbf{max}\{p_i\}$ forma parte de los sumandos de T, así puede ocurrir lo siguiente:
  - T solamente tiene un sumando, así se tiene que se cumple $T \geq \mathbf{max}\{p_i\}$, ya que en particular $T = \mathbf{max}\{p_i\}$.
  - T tiene más sumandos además de $\mathbf{max}\{p_i\}$ así se tiene que se cumple $T \geq \mathbf{max}\{p_i\}$, ya que en particular $T > \mathbf{max}\{p_i\}$
- $\mathbf{max}\{p_i\}$ no forma parte de los sumandos de T, así p_i forma parte de alguna T_k tal que $T \geq T_k$, así obtenemos que $T \geq \mathbf{max}\{p_i\}$.

Demostración

Veamos $T \geq \frac{1}{m} \sum_j^m T_j$

- Es inmediata ésta desigualdad.

Demostración

- Sea M_i la última máquina que completa los trabajos que se le asignaron a través del algoritmo y sea j el índice del último trabajo ejecutado sobre M_i.
- Entonces por construcción el máximo tiempo de procesamiento está en M_i, es decir, $T_i \geq T_k$ para toda $1 \leq k \leq m$ con $k \neq i$.

Demostración

- Se deduce fácilmente que la máquina M_i tiene la menor carga de trabajo al momento de asignar al trabajo J_j a ejecutarse en dicha máquina *(definición del algoritmo)*.
- Consecuentemente la máquina M_i tiene un tiempo de procesamiento más pequeño que el promedio.

Demostración

- Lo que hemos dicho es que:

$$T_i - p_j \leq \frac{1}{m} \sum_{k=1}^{n} p_k$$

- $T_i - p_j$ es el momento en el cuál se asignó la ejecución del trabajo J_j en la máquina M_i.
- Es trivial ver que:

$$p_j \leq \mathbf{max}\{p_k\}$$

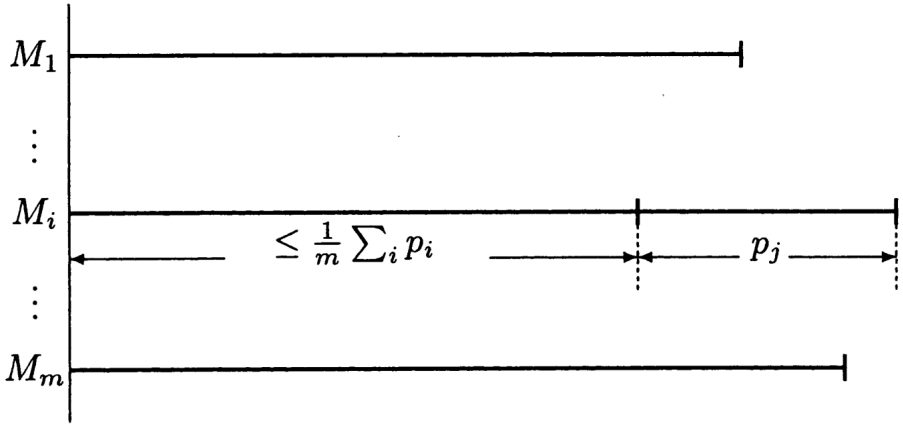
Demostración

- Así

$$\begin{aligned} T_i &= (T_i - p_j) + p_j \\ &\leq \frac{1}{m}\sum_{k=1}^{n} p_k + \mathbf{max}\{p_k\} \\ &\leq T + T = 2 \cdot T \end{aligned}$$

- Por lo tanto el algoritmo es un **algoritmo** 2 **aproximado**.

Demostración



Ejemplo

- $n = m^2 + 1$
- Los trabajos son:

$$J_1, J_2, \ldots, J_{m^2-1}, J_{m^2}, J_{m^2+1}$$

- $p_i = 1$ para toda $i \in \{1, 2, \ldots, m^2 - 1, m^2\}$
- $p_{m^2+1} = m$
- El **makespan óptimo** es $m + 1$.
- El **makespan obtenido por el algoritmo** es $2m$.

Ejemplo

Makespan óptimo

- Notemos que el mejor de los casos sería que una máquina ejecute el trabajo J_{m^2+1} con un tiempo de duración de m unidades de tiempo.
- Así los otros m^2 trabajos restantes tienen un tiempo de duración de una unidad de tiempo.
- Mientras que alguna máquina i con $1 \leq i \leq m$ ejecuta el trabajo J_{m^2+1} las restantes $m-1$ máquinas ejecutan cada una un trabajo con un tiempo de duración de una unidad de tiempo durante m unidades de tiempo *(que es lo que dura el trabajo J_{m^2+1})*.

Ejemplo

Makespan óptimo

- Al finalizar el tiempo m los trabajos que se han ejecutados están dados por la siguiente expresión:

$$m(m-1)+1 = m^2 - m + 1$$

- A los trabajos que nos faltan por ejecutar están dados por la expresión:

$$m^2 + 1 - (m^2 - m + 1) = m^2 + 1 - m^2 + m - 1$$
$$m^2 - m^2 + 1 - 1 + m = m$$

- En éste momento contamos con m máquinas disponibles para los m trabajos así en todos los casos tenemos que $T_i = m + 1$ para toda $1 \leq i \leq m$. Por lo tanto el **makespan óptimo** es $m + 1$.

Ejemplo

Makespan dado por el algoritmo

- Notemos que en el peor de los casos el algoritmo nos proporciona el siguiente orden para los trabajos:

$$J_{\sigma(1)}, J_{\sigma(2)}, \ldots, J_{\sigma(m^2)}, J_{\sigma(m^2+1)}$$

con σ una permutación del conjunto $\{1, 2, \ldots, m^2, m^2+1\}$ y $\sigma\left(m^2+1\right) = m^2+1$.

- Dado lo anterior, el último trabajo en ejecutarse es el trabajo J_{m^2+1} cuyo tiempo de duración es de m unidades de tiempo.

Ejemplo

Makespan dado por el algoritmo

- Por la descripción del algoritmo en las primeras m unidades de tiempo las m máquinas ejecutan en cada unidad de tiempo un trabajo cuya duración es de una unidad de tiempo así, los trabajos ejecutados hasta el tiempo m están dados por la expresión:

$$m(m) = m^2$$

- Al finalizar el tiempo m, alguna de las m máquinas ejecutará el trabajo J_{m^2+1} dándonos así que el **makespan** dado por el algoritmo es de $2m$.

¿Es un PTAS?

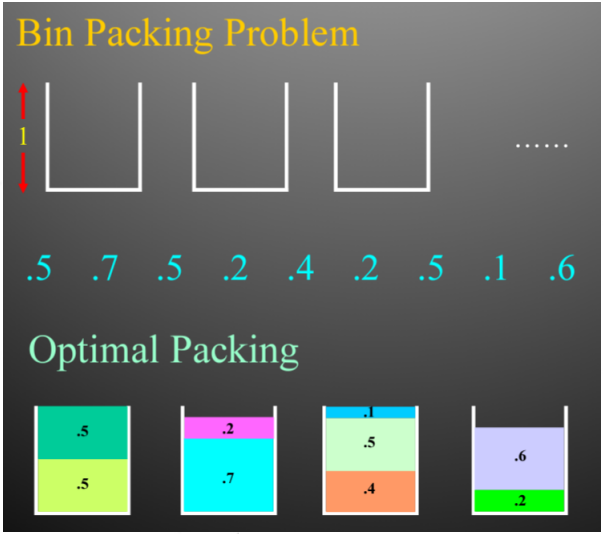
- No es un **PTAS**(*Polynomial Time Approximation Scheme*).

Bin Packing

Descripción del problema

- Dados n objetos con tamaños $a_1, a_2, \ldots, a_n$ tal que $0 < a_i \leq 1$ para todo $1 \leq i \leq n$.
- Encontrar un empaquetamiento en contenedores de tamaño 1 tal que el número de contenedores usado es el mínimo.

Bin Packing


Bin Packing Problem
1
.....
.5 .7 .5 .2 .4 .2 .5 .1 .6
Optimal Packing
.5
.5
.2
.7
.1
.5
.4
.6
.2

Bin Packing

Relación con minimum makespan scheduling

- El problema **minimum makespan scheduling** está íntimamente relacionado con el problema de **bin packing** ya que existe una caledarización *(scheduling)* con un **makespan** de tiempo t sí y solamente sí n objetos de tamaños $p_1, p_2, \ldots, p_n$ pueden ser empacados en m contenedores de tamaño t cada uno.
- Es decir, existe una reducción del problema **minimum makespan scheduling** al problema de **bin packing**, ya que:
  - De n trabajos pasamos a n objetos.
  - De los tiempos de duración $p_1, p_2, \ldots, p_n$ correspondientes a los n trabajos pasamos a los tamaños $p_1, p_2, \ldots, p_n$ de los n objetos.
  - De m máquinas para ejecutar los trabajos pasamos a m contenedores para guardar los objetos.
  - Del **makespan** el cual es t pasamos a que cada contenedor tenga tamaño t.

Descripción del esquema de aproximación

Notación

- Denotaremos al conjunto de los tamaños de los n objetos, los cuales son $p_1, p_2, \ldots, p_n$ con la letra I.
- **bins** (I, t) es el mínimo número de contenedores de tamaño t que se necesitan para empacar los n objetos.
- El **makespan** mínimo está dado por $\textbf{min}\{\ t\ :\ \textbf{bins}\,(I, t)\ \leq\ m\}$

Descripción del esquema de aproximación

- Hacemos la observación de que LB es una cota inferior y $2 \cdot LB$ es una cota superior del **makespan** mínimo.
- Lo que se hará es determinar el **makespan** mínimo por búsqueda binaria en el intervalo $[LB, 2 \cdot LB]$.
- El problema se puede resolver en tiempo polinomial sí el conjunto de los tamaños de los objetos es finito y su cardinalidad es fija.

Descripción del esquema de aproximación

Bin packing con un número fijo de tamaños de los objetos.

- Sea k el número fijo que denota los distintos tamaños de los objetos y supondremos que los contenedores tiene capacidad de 1.
- Ordenamos el tamaño de los objetos.
- Ahora un ejemplar genérico del problema de **bin packing** puede ser descrito de la siguiente manera:
  - Una $k-$tupla definida como $(i_1, i_2, \ldots, i_k)$ donde cada i_j con $1 \leq j \leq k$ especifica el número de objetos que tienen un tamaño que está en el conjunto I, es decir, $(i_1, i_2, \ldots, i_k)$ especifica el número de de objetos de cada tamaño.
- **BINS** $(i_1, i_2, \ldots, i_k)$ denota el número mínimo de contenedores que son necesitados para empacar éstos objetos.

Descripción del esquema de aproximación

Algoritmo de programación dinámica

- Sea $(n_1, n_2, \ldots, n_k)$ un ejemplar del problema de **bin packing con número fijo de tamaños de los objetos** tal que $\sum_{i=1}^{k} n_i = n$
- Calcularemos el conjunto $\mathcal{Q}$ cuyos elementos serán todas las $k-$tuplas $(q_1, q_2, \ldots, q_k)$ tales que:
  - $\textbf{BINS}(q_1, q_2, \ldots, q_k) = 1$
  - y además

$$0 \leq q_i \leq n_i$$
$$1 \leq i \leq k$$

Descripción del esquema de aproximación

- $\mathcal{Q}$ contiene a lo más $O\left(n^k\right)$ elementos, ésto es muy sencillo de ver.
- Denotamos como w_i al peso de cada uno de los n_i objetos.
- Así el peso total de los n_i objetos está dado por $w_i \cdot n_i$ para toda $1 \leq i \leq k$.
- Sí ocurre que:

$$\left(\sum_{i=1}^{n} w_i \cdot n_i\right) \leq 1$$

- Entonces cualquier $k-$tupla $(q_1, q_2, \ldots, q_k)$ tales que:

$$0 \leq q_i \leq n_i$$
$$1 \leq i \leq k$$

cumplirá que $\textbf{BINS}\left(q_1, q_2, \ldots, q_k\right) = 1$

- Así el número de tuplas está dado por:

$$(n_1 + 1) \cdot (n_2 + 1) \cdots (n_{k-1} + 1) \cdot (n_k + 1)$$

Descripción del esquema de aproximación

- Es $n_i + 1$ para toda $1 \leq i \leq k$ ya que tenemos que contar la opción de poner el número 0.
- Tenemos que

$$n_i \leq n$$

para toda $1 \leq i \leq k$.

- Trivialmente tenemos que:

$$n_i + 1 \leq n + 1$$

para toda $1 \leq i \leq k$.

- Así

$$(n_1 + 1) \cdot (n_2 + 1) \cdots (n_{k-1} + 1) \cdot (n_k + 1) \leq (n + 1) \cdot (n + 1) \cdots (n + 1) \cdot (n + 1)$$
$$(n_1 + 1) \cdot (n_2 + 1) \cdots (n_{k-1} + 1) \cdot (n_k + 1) \leq (n + 1)^k$$

de ahí la complejidad en tiempo.

Descripción del esquema de aproximación

- Ahora calcular todas las **BINS** $(i_1, i_2, \ldots, i_k)$ para cada $k-$tupla $(i_1, i_2, \ldots, i_k) \in \mathcal{Q}'$
- El conjunto $\mathcal{Q}'$ está definido como sigue:

$$\mathcal{Q}' = \{0, \ldots, n_1\} \times \{0, \cdots, n_2\} \times \cdots \times \{0, \ldots, n_k\}$$

- Hacemos la observación de que **BINS** $(q) = 1$ para todo $q \in \mathcal{Q}$.

Descripción del esquema de aproximación

- El cálculo se hará de manera recursiva como sigue:

$$\textbf{BINS}\left(i_1, i_2, \ldots, i_k\right) = 1 + \min_{q \in \mathcal{Q}} \textbf{BINS}\left(i_1 - q_1, i_2 - q_2, \ldots, i_k - q_k\right)$$

- Calcular cada $\textbf{BINS}\left(i_1, i_2, \ldots, i_k\right)$ nos toma tiempo $O\left(n^k\right)$ ya que de nuevo en el peor de los casos el conjunto $\mathcal{Q}$ tiene $O\left(n^k\right)$ elementos.
- Por lo tanto calcular todos los $\textbf{BINS}\left(i_1, i_2, \ldots, i_k\right)$ nos toma tiempo $O\left(n^{2k}\right)$.
- Al calcular todos los $\textbf{BINS}\left(i_1, i_2, \ldots, i_k\right)$, podemos determinar $\textbf{BINS}\left(n_1, n_2, \ldots, n_k\right)$.

Descripción del esquema de aproximación

- Podemos aceptar algunos error al calcular **minimum makespan scheduling**. Sólo hay dos formas de generar un error las cuales son:
  - Redondear los tamaños de los objetos, para así tener un número acotado de los distintos tamaños de los objetos.
  - Terminar la búsqueda binaria.

Core Algorithm

- Sea ε el parámetro de error y sea t tal que $t \in [LB, 2 \cdot LB]$.
- **objeto pequeño** Un objeto es pequeño sí su tamaño es a lo más $\varepsilon \cdot t$.
- Redondearemos los objetos que no son pequeños de la siguiente manera:
  - Cada $p_j \in [t \cdot \varepsilon (1+\varepsilon)^i, t \cdot \varepsilon (1+\varepsilon)^{i+1}]$, entonces definimos p'_j como $t\varepsilon(1+\varepsilon)^i$.
- A lo más habrá k diferentes tamaños, donde $k = \lceil \log_{1+\varepsilon} \frac{1}{\varepsilon} \rceil$.
- Determinamos el empaquetamiento óptimo para los objetos que hemos redondeado su peso con el algoritmo de **programación dinámica** que dimos anteriormente, en contenedores de tamaño t.

Core Algorithm

- Como al redondear podemos reducir el tamaño de cada objeto por un factor de a lo más $(1+\varepsilon)$ *(Ya que en el peor de los casos* $p_j = t \cdot \varepsilon\,(1+\varepsilon)^{i+1}$*) se convierte a* $p'_j = t \cdot \varepsilon\,(1+\varepsilon)^{i}$ y el factor está dado por $\frac{p_j}{p'_j}$.
- Así al considerar los tamaños originales de los objetos, el empaquetamiento obtenido por el **algoritmo de programación dinámica** es válido para contenedores de tamaño $t\,(1+\varepsilon)$.
- Ahora empacamos los objetos pequeños de manera *greedy* en los espacios que sobran en los contenedores.
- Denotamos $\alpha\,(I, t, \varepsilon)$ como al número de contenedores usado por **Core Algorithm**.

Core Algorithm

- Se cumple que $\alpha(I, t, \varepsilon) \leq \textbf{bins}(I, t)$ ya que tenemos dos casos:
  1. Sí **Core Algorithm** no utiliza nuevos contenedores para empacar los objetos pequeños entonces es trivial ver que se cumple la desigualdad.
  2. Sí **Core Algorithm** utiliza nuevos contenedores entonces todos los contenedores con excepción posiblemente del último, están llenos al menos en un tamaño t entonces al menos el empacamiento dentro de contenedores de tamaño t debió usar al menos $\alpha(I, t, \varepsilon)$ contenedores, por lo tanto se sigue cumpliendo la desigualdad.
- Por lo anterior obtenemos que:

$$\text{mín}\{\ t\ :\ \alpha(I, t, \varepsilon) \leq m\} \leq OPT$$

Core Algorithm

- Sí mín$\{\ t\ :\ \alpha(I, t, \varepsilon) \leq m\}$ puede ser determinado sin error adicional durante la **búsqueda binaria**, entonces podemos usar **Core Algorithm** para obtener un **makespan** de $(1 + \varepsilon) \cdot OPT$ *(debido al tamaño de los contenedores)*.

Algoritmo A

- Si $\alpha(I, LB, \varepsilon) \leq m$, entonces usamos el empaquetamiento dado por **Core Algorithm** para $t = LB$.
- Si $\alpha(I, LB, \varepsilon) > m$, entonces hacemos una **búsqueda binaria** para encontrar un intervalo $[T', T]$ tal que $[T', T] \subseteq [LB, 2 \cdot LB]$ con $T - T' \leq \varepsilon \cdot LB$, tal que $\alpha(I, T', \varepsilon) > m$ y $\alpha(I, T, \varepsilon) \leq m$. Entonces regresamos el empaquetamiento proporcionado por **Core Algorithm** para $t = T$.

Teorema

- El algoritmo A, tal que para toda $\varepsilon > 0$, A encuentra una **caledarización** *(schedule)* con un **makespan** de a lo más $(1 + 3 \cdot \varepsilon) \cdot OPT$. La **complejidad en tiempo** del algoritmo A es $O\left(n^{2k} \cdot \lceil \log_2 \frac{1}{\varepsilon} \rceil\right)$, donde $k = \lceil \log_{1+\varepsilon} \frac{1}{\varepsilon} \rceil$.

Análisis del Algoritmo A

- La **búsqueda binaria** utiliza a lo más $\lceil \log_2 \frac{1}{\varepsilon} \rceil$ pasos, ya que la **complejidad en tiempo** del algoritmo A es una hipótesis del teorema anterior.
- Si $\alpha(I, LB, \varepsilon) \leq m$ entonces por las observaciones hechas tenemos que:

$$\begin{aligned} \alpha(I, LB, \varepsilon) &\leq (1+\varepsilon) \cdot LB \\ &\leq (1+\varepsilon) \cdot OPT \\ &\leq (1+3 \cdot \varepsilon) \cdot OPT \end{aligned}$$

Análisis del Algoritmo A

- Si $\alpha(I, LB, \varepsilon) > m$ entonces tenemos que se cumple:
  - $m < \alpha(I, T', \varepsilon) \leq \textbf{bins}(I, T')$
  - $T' \leq OPT$
  - y además:

$$
\begin{aligned}
T &\leq T' + \varepsilon \cdot LB \\
T &\leq OPT + \varepsilon \cdot LB \\
T &\leq OPT + \varepsilon \cdot OPT \\
T &\leq (1 + \varepsilon) \cdot OPT
\end{aligned}
$$

  - Ya que **Core Algorithm** para $T = t$ regresa una **calendarización** *(schedule)* con un **makespan** de a lo más $(1 + \varepsilon) \cdot T$, el **makespan** de la **calendarización** cumple lo siguiente:

$$
\begin{aligned}
(1 + \varepsilon) \cdot T &\leq (1 + \varepsilon)^2 \cdot OPT \\
&\leq (1 + 3 \cdot \varepsilon) \cdot OPT
\end{aligned}
$$

Contenido

Notación de algoritmos de scheduling

Trabajos

- Número: n
- Índice: j
- Features:
  - tiempo de procesamiento: pj ó pij
  - tiempo de liberación: rj
  - tiempo limite: dj
  - peso: wj

Notación de algoritmos de scheduling

Máquinas

- Número: m
- Índice: i
- Ambientes:
  - 1 máquina: 1
  - Máquinas paralelas: P,P$_m$
  - Máquinas relacionadas: Q,Q$_m$
  - Máquinas no relacionadas: R,R$_m$

Notación de algoritmos de scheduling

Restricciones

- Que los trabajos tienen tiempo de liberación: rj
- Que los trabajos pueden adelantarse (preemption): pmtn
- Restricciones de precedencia: prec
- Las máquinas pueden dejar de funcionar: bkdwn
- Tamaño límite del buffer: block

Notación de algoritmos de scheduling

Objetivos

- Tiempo para completar los trabajos para cada trabajo: C_j
- Por retraso: $L_j = C_j$ - d_j
- Por tardanza: $T_j = \max L_j$
- Y otros.

Ejemplos:
P||Cmax – máquinas paralelas idénticas, minimizar la longitud de calendarización.
1|prec, pmtn|$\Sigma w_j C_j$ - Una máquina, restricciones de precedencia y adelantamiento, minimizar la suma de los pesos de los tiempos de completud.

P | pmtn | C_{max}

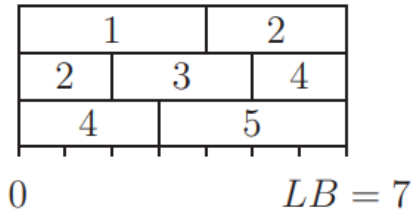
Un límite inferior para este problema es:

$$LB := max\{max_i pi, (\sum_{i=1}^{n} pi)/m\}.$$

Una calendarización que tenga este límite puede ser construida en tiempo $O(n)$: llenar las máquinas sucesivamente, programar los trabajos en cualquier orden y dividir trabajos en dos partes siempre que se cumpla el límite de tiempo anterior. Programar la segunda parte de un trabajo adelantado en la siguiente máquina en el tiempo cero.
Debido al hecho de que $p_i \leq LB$ para todo i, las dos partes del trabajo no se traslapan.

$m = 3$

i	1	2	3	4	5
p_i	4	5	3	5	4



Scheduling on Unrelated Parallel Machines

Dado un conjunto J de trabajos, un conjunto M de maquinas, y por cada $j \in J$ y $i \in M$, tenemos $p_{ij} \in \mathbf{Z}^+$, que es el tiempo que se tarda en procesar el trabajo j en la maquina i, el problema es calendarizar los trabajos en las maquinas para minimizar el makespan, es decir el tiempo de procesamiento máximo de cualquier maquina. En el libro se denota el numero de trabajos como n y el numero de maquinas como m.

Nota: decimos "no relacionadas"porque no hemos supuesto ninguna relación entre los tiempos de procesamiento de un trabajo en diferentes maquinas. si cada trabajo tiene el mismo tiempo de procesamiento, sea p_j dicho tiempo, en cada una de las maquinas, entonces decimos que las maquinas son *identicas*.

Scheduling on Unrelated Parallel Machines

Para el 2-algoritmo para el problema de la programación en máquinas paralelas no relacionadas aplicaremos la técnica de poda paramétrica, junto con el redondeo de programación lineal, para obtener el algoritmo.

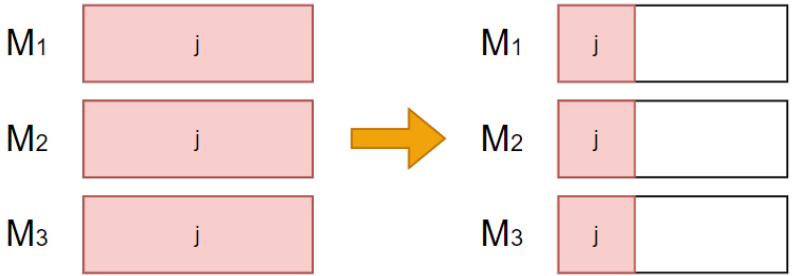
Poda paramétrica en una configuración de Programación Lineal

En este programa x_{ij} es una variable indicadora que nos dice si el trabajo j está programado(calendarizado) en la máquina i. El objetivo es minimizar t, el makespan. El primer conjunto de restricciones nos asegura que cada trabajo es calendarizado en una de las máquinas, y el segundo conjunto asegura que cada maquina tiene un tiempo de procesamiento de a lo más t.

$$
\begin{aligned}
&\texttt{minimizar } t \\
&\texttt{sujeto a } \sum_{i \in M} x_{ij} = 1, && j \in J \\
&\sum_{j \in J} x_{ij} p_{ij} \leq t, && i \in M \\
&x_{ij} \in \{0,1\}, && i \in M, j \in J
\end{aligned}
$$

Ejemplo

Supongamos que tenemos un solo trabajo, cuyo tiempo de procesamiento es m en cada m maquina. El minimo makespan es m. Sin embargo, la solución óptima para la relajación lineal es calendarizar el trabajo exactamente a $1/m$ en cada maquina, lo que conduce a un valor de 1, y dando una brecha de integralidad de m.



Poda paramétrica en una configuración de Programación Lineal

Este ejemplo ocupa una ventaja injusta que se le da por la relajación lineal. El programa entero asigna 0 a x_{ij} de forma automática si $p_{ij} > t$. Pero visto de otra forma, a la relajación lineal se le tiene permitido asignar estas variables a valores que no sean 0, y de ahí obtener la solución más "barata". Esta situación podría ser rectificada si pudiéramos añadir la siguiente restricción a la relajación lineal:

$$\forall i \in M, j \in J : \texttt{if } p_{ij} > t \texttt{ then } x_{ij} = 0.$$

Poda paramétrica en una configuración de Programación Lineal

La restricción anterior no es una restricción lineal.

Entonces para afrontar esta dificultad se usa la técnica de poda paramétrica. El parámetro será $T \in \mathbf{Z}^+$, que sera ell candidato para un limite inferior para el makespan optimo. El parámetro nos permitirá podar todos los pares máquina-trabajo tales que $p_{ij} > T$. Se define $S_T = \{(i,j)|p_{ij} \leq T\}$. Se define una familia de programas lineales $LP(T)$, una para cada valor el parámetro $T \in \mathbf{Z}^+$. $LP(T)$ usa las variables x_{ij} solo para $(i,j) \in S_T$, y pregunta si hay una calendarización fraccionada y factible del makespan $\leq T$ usando las posibilidades restringidas.

$$\sum_{i:(i,j)\in S_T} x_{ij} = 1, j \in J$$

$$\sum_{j:(i,j)\in S_T} x_{ij} p_{ij} \leq T, i \in M$$

$$x_{ij} \geq 0, (i,j) \in S_T$$

Propiedades de solución de punto extremo

Por medio de una búsqueda binaria, encontramos el valor más pequeño de T tal que $LP(T)$ tiene una solución factible. Digamos que T^* es este valor, T^* es un limite inferior en OPT(el costo de una solución fraccionada óptima). El algoritmo redondea una solución de punto extremo a $LP(T*)$ para encontrar una calendarización con makespan $\leq 2T^*$.

Propiedades de solución de punto extremo

Lema 17.3: Cualquier solución de punto extremo para $LP(T)$ tiene a lo más $n + m$ variables diferentes de 0.

Demostración: Sea $r = |S_T|$ el número de variables en que $LP(T)$ está definido. Recordemos que una solución factible para $LP(T)$ es una solución de punto extremo si y sólo si corresponde a establecer r restricciones linealmente independientes de $LP(T)$ para la igualdad.

De estas r restricciones linealmente independientes, al menos $r - (n + m)$ deben ser elegidas a partir del tercer conjunto de restricciones (de la forma $x_{ij} \geq 0$). Las variables correspondientes se establecen con 0. Entonces, cualquier solución de punto extremo tiene a lo más $n + m$ variables distintas de 0.

Propiedades de solución de punto extremo

Sea x una solución de punto extremo para $LP(T)$. Diremos que el trabajo j es establecido de manera integral en x si esta asignado de manera entera a una maquina. De otro modo diremos que j está establecido de manera fraccionada.

Propiedades de solución de punto extremo

Corolario 17.4: Cualquier solución de punto extremo para $LP(T)$ debe establecer al menos $n - m$ trabajos de forma integral.

Demostración: Sea x una solución de punto extremo para $LP(T)$, y sea α y β el número de trabajos que son establecidos de manera integral y fraccionada respectivamente. Cada trabajo fraccionado está asignado al menos a 2 maquinas y así resulta en al menos 2 entradas en x diferentes de 0. Así tenemos

$$\alpha + \beta = n\alpha + 2\beta \leq n + m$$

Entonces tenemos que $\beta \leq m$ y $\alpha \geq n - m$

Propiedades de solución de punto extremo

Correspondiente a una solución de punto extremo x a $LP(T)$, definimos $G = (J, M, E)$ como la gráfica bipartita en el conjunto de vértices $J \cup M$ tal que $(j, i) \in E$ si y solo si $X_{ij} \neq 0$. Sea $F \subset J$ el conjunto de trabajos que se establecen de manera fraccionada en x, y sea H la subgrafica de G inducida en el conjunto de vértices $F \cup M$. Claramente,(i, j) es una arista en H si y solo si $0 < X_{ij} < 1$. Un emparejamiento en H será perfecto si empareja todos los trabajos $j \in F$. El procedimiento de redondeo utiliza el hecho que la gráfica H tiene un emparejamiento perfecto.

Algoritmo de aproximación

1. Por medio de una búsqueda binaria entre el intervalo $[\alpha/m, \alpha]$, encontramos el valor más pequeño de $T \in \mathbf{Z}^+$ para el cual $LP(T)$ tiene una solución factible. Le asignamos este valor a T^*.
2. Buscamos una solución para el punto extremo, sea x esta solución para $LP(T^*)$.
3. Asignamos todos los trabajos agregados a las maquinas como con x.
4. Construimos la gráfica H y encontramos un emparejamiento perfecto M en ella.
5. Asignamos trabajos establecidos fraccionadamente a las máquinas acorde al emparejamiento de M.

Propiedades adicionales de solución de punto extremo

Una gráfica conexa con su conjunto V de vértices es un pseudo-arbol si contiene a lo más $|V|$ aristas. Como la gráfica es conexa tiene que tener al menos $|V| - 1$ aristas. Entonces tenemos que la gráfica o es un árbol o un árbol con una arista extra. En este último caso sabemos que la gráfica tiene un solo ciclo. Digamos que una gráfica es un pseudo-bosque si cada uno de los componentes conexos de la gráfica son pseudo-arboles.

Propiedades adicionales de solución de punto extremo

Lemas

17.6 La gráfica G es un pseudo-bosque.

17.7 La gráfica H tiene un emparejamiento perfecto.

Teorema

17.8 El algoritmo de aproximación nos garantiza un factor 2 para el problema de calendarizar en maquinar paralelas no relacionadas.

Lema: La gráfica G es un pseudo-bosque

Demostración:
En esta demostración mostramos que el número de aristas en cada componente conexo de G está acotado por el número de vértices que hay en el. Entonces cada componente conexo es un pseudo-árbol.

Consideramos un componente conexo G_C. Restringimos $LP(T)$ y x a los trabajos y maquinas de G_C, para obtener $LP_C(T)$ y x_C. Sea $x_{\overline{C}}$ el resto de x.

Una observación importante es que x_C debe ser una solución de punto extremo para $LP_C(T)$.

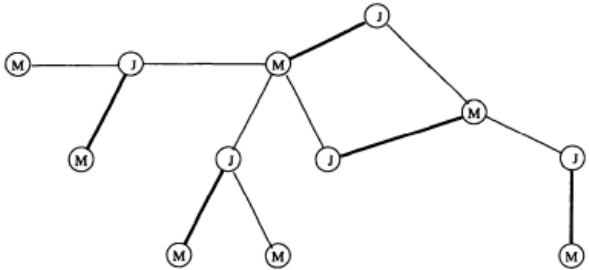
Propiedades adicionales de solución de punto extremo

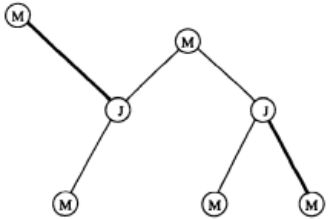
Supongamos que este no es el caso. Entonces x_C es una combinación convexa de 2 soluciones factibles para $LP_C(T)$. Cada una de estas, junto con $x_{\overline{C}}$, forman una solución factible para $LP(T)$. Así x es una combinación convexa de 2 soluciones factibles para $LP(T)$. Esto nos lleva a una contradicción.

Aplicando el lema 17.3 (Cualquier solución de punto extremo para $LP(T)$ tiene a lo más $n + m$ variables diferentes de 0), tenemos que G_C es un pseudo-árbol.

Lema: H tiene emparejamiento perfecto

Cada trabajo que se coloca en x tiene exactamente una arista incidente en G. Eliminamos estos trabajos, junto con sus aristas incidentes, de G. La gráfica que nos queda es H. Ya que se eliminaron un mismo número de aristas y de vértices, H igual es un pseudo-bosque.


M
J
M
J
M
J
M
J
J
M
M
M


M
M
J
J
M
M
M

En H, cada trabajo tiene un grado de a lo menos 2. Entonces, todas las hojas en H deben ser maquinas. Seguimos emparejando una hoja con el trabajo en el que la hoja incide, y eliminamos a ambos de la gráfica. En cada paso las hojas deben ser maquinas, al final nos quedaran ciclos pares, ya que empezamos con una gráfica bipartita. Emparejamos aristas de forma alternada en cada ciclo. Esto nos da un emparejamiento perfecto en H.

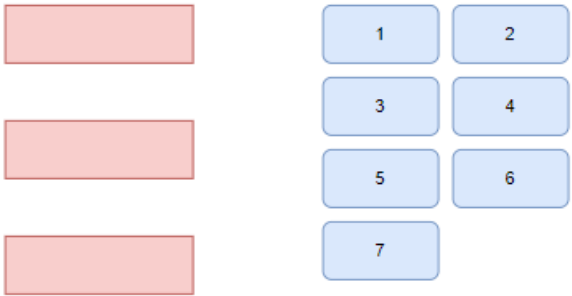
Garantizado factor 2 para algoritmo de aproximación

Demostración:

Claramente $T^* \leq$OPT, ya que $LP(OPT)$ tiene una solución factible. La solución de punto extremo, x, para $LP(T^*)$ tiene un makespan fraccionado $\leq T^*$. Por lo tanto, la restricción de x para establecer trabajos integralmente tiene un makespan $\leq T^*$. Cada arista (i, j) de H satisface $p_{ij} \leq T$. El emparejamiento perfecto encontrado en H calendariza a lo más un trabajo de más en cada maquina. Por lo tanto, el makespan total es $\leq 2T^* \leq 2 \cdot OPT$. El algoritmo corre en tiempo polinomial.

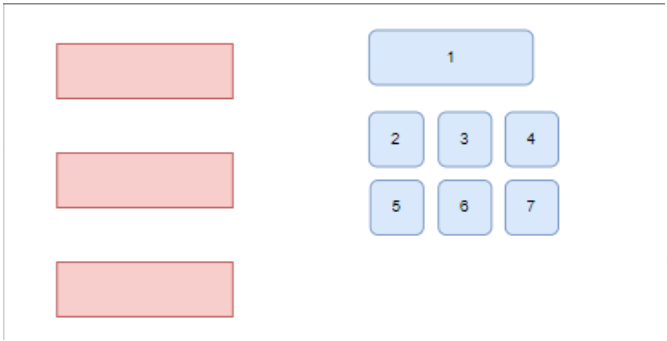
Ejemplo

Sean $m^2 - m + 1$ los trabajos que necesitan ser calendarizados en m maquinas.



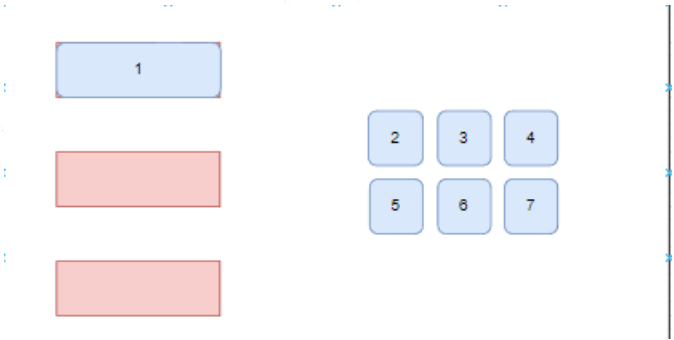
Ejemplo

El primer trabajo tiene un tiempo de procesamiento de m en todas las maquinas, y el resto de los trabajos tienen tiempo de procesamiento unitario en cada máquina.



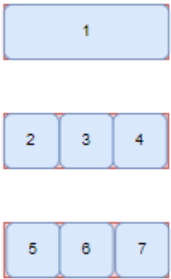
Ejemplo

El calendarizado optimo asigna el primer trabajo a una máquina



Ejemplo

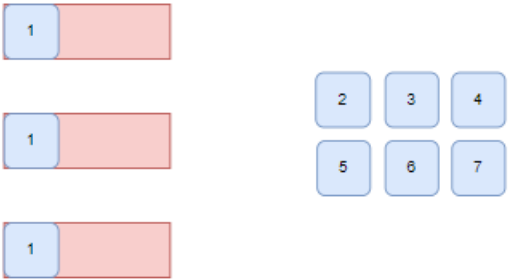
y a los m trabajos restantes a cada $m-1$ máquinas restantes.



El makespan es m. Es fácil ver que $LP(T)$ no tiene una solución factible para $T < m$.

Ejemplo

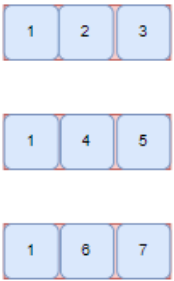
Ahora suponemos que la siguiente solución de punto extremo para $LP(T)$ es elegida. Asigna $1/m$ al primer trabajo



y $m-1$ a los demás trabajos a cada una de las m maquinas.
El redondeo producirá una calendarización con un makespan de $2m-1$.

Ejemplo

y $m-1$ a los demás trabajos a cada una de las m maquinas.



El redondeo producirá una calendarización con un makespan de $2m-1$.

Referencias I

[1] Vazirani, V. V. Approximation Algorithms *Springer. Berlin (2003), 79-83, 139-144.*