Seguimiento

Alfonso Fierro López

Semestre 2023-2

1. Definiciones

Dados n trabajos de un conjunto $\mathcal{J}$, se pretende crear un algoritmo de ordenamiento sobre $\mathcal{J}$ tal que el tiempo requerido para procesar todos los trabajos por las m máquinas de un conjunto $\mathcal{M}$ sea mínimo.

Para cada trabajo $j \in \mathcal{J}$ se calcula el tiempo de procesamiento mínimo P_j que requeriría para atravesar todas las máquinas. Si p_{jm} representa el tiempo que toma procesar el trabajo j en la máquina m entonces P_j sigue la siguiente definición.

$$P_j = \sum_{k \in M} p_{jk}$$

Cuando existen varios trabajos con un mismo tiempo de procesamiento mínimo, se dice que un trabajo j domina a un trabajo i o $j \prec i$ si al comparar los tiempos de procesamiento ocurre que

$$(\min_k p_{jk} < p_{ik}) < (\min_k p_{ik} < p_{jk})$$

Un elemento mínimo de un conjunto U en que utilice esta relación de orden estará denotado como

$$\overset{*}{\min} U$$

Una vez calculados todos los tiempos de procesamiento mínimo se construye con ellos el conjunto $\mathcal{P}_1$. Decimos que el trabajo con primer prioridad para procesarse $j^{(1)}$ corresponde al subíndice j de los elementos $P_j \in \{\min_{P_j} \mathcal{P}_1\}$, tal que este también tenga el menor tiempo de trabajo en la máquina con enumeración más baja. Usando las definiciones se puede afirmar que

$$j^{(1)} = \overset{*}{\min_j}\{P_j \in \{\min_{P_j \in P} \mathcal{P}_1\}\}$$

Se construye el conjunto $\mathcal{P}_2 = \mathcal{P}_1 - P_{j^{(1)}}$. El trabajo que debería tener segunda prioridad $j^{(2)}$ para procesarse tiene la siguiente definición.

$$j^{(2)} = \overset{*}{\min_j}\{P_j \in \{\min_{P_j \in P} \mathcal{P}_2\}\}$$

De forma general, se deben construir n conjuntos para poder asignar a cada trabajo una prioridad w. Esto se realiza para poder ordenar los trabajos según su tiempo de procesamiento mínimo.

$$\mathcal{P}_k = \mathcal{P}_{k-1} - P_{j^{(k-1)}} \quad k > 1 \tag{1}$$

$$j^{(k)} = \overset{*}{\min_j}\{P_j \in \{\min_{P_j \in P} \mathcal{P}_k\}\} \tag{2}$$

Se asume que todo trabajo debe ser procesado en orden desde la máquina 1 hasta la máquina m. Lo anterior no debe confundirse con el orden en que deben procesarse los trabajos, esto último será referido como la *posición* del trabajo y se señalará como un subíndice. Así, $j^{(w)}_{(v)}$ se lee como el trabajo j con posición v y prioridad w.

El trabajo con *prioridad* 1 también tendrá inicialmente la *posición* 1. El tiempo requerido para terminar en la máquina m el trabajo en la posición 1 y prioridad x $\tau^{(x)}_{(1)m}$, es igual a la suma del tiempo de todos sus procesos hasta la máquina m, esto es

$$\tau^{(x)}_{(1)m} = \sum_{i=1}^{m} p_{j^{(x)}_{(1)} i}$$

El tiempo requerido para terminar en la máquina m el trabajo en la posición 2 y prioridad arbitraria y $\tau^{(y)}_{(2)m}$ puede definirse como sigue

$$\tau^{(y)}_{(2)m} = \text{máx}\{\tau^{(y)}_{(2)m-1}, \tau^{x}_{(1)m}\} + p_{j^{(y)}_{(2)}m}$$

esto es, el máximo entre el tiempo de finalización en la máquina m del trabajo $j^{x}_{(1)}$ y el tiempo de finalización en la máquina $m-1$ del trabajo $j^{(y)}_{(2)}$, más el tiempo que le toma al mismo trabajo en la máquina m. En general, para un trabajo en la posición $k > 1$ y con prioridad arbitraria, el tiempo requerido para terminar en la máquina z puede definirse como sigue

$$\tau^{(x)}_{(k)z} = \text{máx}\{\tau^{(x)}_{(k)z-1}, \tau^{(y)}_{(k-1)m}\} + p_{j^{(y)}_{(2)}m}$$

De esta forma, el tiempo de finalización de todos los trabajos corresponde con el tiempo de finalización del trabajo en la posición m, esto permite deducir que el propósito del algoritmo es minimizar $\tau^{x}_{(m)m}$.

2. Descripción del algoritmo

El algoritmo de minimización para $\tau^{x}_{(m)m}$ está compuesto por las siguientes etapas.

1. Se deben ordenar por prioridad todos los elementos en el conjunto de trabajos.
2. Se debe asignar la posición 1 al elemento con prioridad 1.
3. Se calcula el tiempo que toma terminar el trabajo 1.
4. El elemento de prioridad 1 es descartado del conjunto de trabajos.
5. Se introduce en la posición más alta el siguiente elemento con prioridad más baja.
6. El nuevo elemento introducido se descarta del conjunto de trabajos.
7. Se calcula y memoriza como $\mathcal{T}$ el tiempo que toma terminar el trabajo en la última posición.
8. Se memoriza como $\mathcal{C}$ la configuración de trabajos que producen el resultado anterior .
9. El nuevo trabajo introducido intercambia posiciones con el trabajo en la posición anterior.
10. Se calculan y memorizan como $\mathcal{T}_N$ y $\mathcal{C}_n$ el tiempo y la configuración que toma terminar el trabajo en la última posición.
11. Si $\mathcal{T}_N$ es menor que $\mathcal{T}$, se dice que ahora $\mathcal{T} = \mathcal{T}_N$ y $\mathcal{C} = \mathcal{C}_N$.
12. Se repiten los pasos 9,10 y 11 hasta que el trabajo introducido esté en la posición 1.
13. Se fija a $\mathcal{C}$ como la configuración sobre la que se tendrán que adicionar nuevos trabajos.
14. Se repite desde el paso 5 hasta el paso 13 hasta que el conjunto de trabajos quede vacío.

El último valor de $\mathcal{C}$ y $\mathcal{T}$ sirven como una primer aproximación para el valor mínimo de $\tau^{x}_{(m)m}$.

3. Pseudocódigo

Se presenta el pseudocódigo para el algoritmo descrito.

```
\begin{algorithm}
    \caption{Pseudocódigo para ordenamiento de trabajos}\label{alg: OrdenTrabajos}
    \begin{algorithmic}
        \Require J,M \Comment{Requerir conjunto de trabajos y máquinas}
        \State Jr = SORT J \Comment{Ordenar trabajos en una lista}
        \State $C$ \gets [] \Comment{Crear lista con el orden de procesamiento de los trabajos}
        \State $T$ \gets $\infty$
        \While{J not empty}
            \State Cn \gets []
            \State Tn \gets $\infty$
            \State Entrada \gets pop(J)
            \State C.append(Entrada)
            \ForAll{Position i in C}
                \State Entrada \gets Position i \Comment{Cambiar posición de la entrada}
                \State Cn \gets C with Entrada in i
                \State Tn \gets Tiempo final de trabajo
                \If{Tn < T}
                    \State T \gets Tn
                    \State C \gets Cn
                \EndIf
            \EndFor
        \EndWhile
        \State \Return C, T
    \end{algorithmic}
```

Figura 1: Pseudocódigo propuesto para el ordenamiento de trabajos