

## Tarea 2: Planificación de Trabajos

CC4102 - Diseño y Análisis de Algoritmos

Profesor:	Gonzalo Navarro
Auxiliar:	Teresa Bracamonte
Alumnos:	Cristián Carreño
	Sergio Maass
Fecha:	17 de Noviembre de 2013

# 1. Introducción

Los algoritmos online cumplen un rol muy importante al permitir resolver problemas a medida que su entrada se va recibiendo, o sea, sin conocer el total de los datos previamente. Pero además, son muy útiles para aproximar soluciones a problemas difíciles en tiempo razonablemente corto.

Diversos problemas de optimización pertenecientes a la clase  $NP$  admiten soluciones aproximadas mediante algoritmos online  $p$ -competitivos. Un algoritmo de estas características permite resolver instancias arbitrarias del problema en tiempo polinomial, garantizando que la solución estará dentro de un radio  $p$  del óptimo. Esto es muy importante, ya que si  $P \neq NP$  -lo que parece ser el caso- no es posible encontrar una solución óptima en tiempo polinomial.

En este informe se presentarán dos versiones del problema de planificación de trabajos ( $NP$  - *completo*<sup>1</sup>) y se diseñará un algoritmo online 2-competitivo para cada una. Se demostrará formalmente la competitividad de cada algoritmo y luego se comprobará experimentalmente.

## 2. Trabajos de una etapa

El problema consiste en un conjunto de  $n$  trabajos que deben asignarse a  $m$  máquinas idénticas. Cada trabajo  $j$  requiere una cantidad de tiempo  $T_j$  para ser finalizado. Además, cada trabajo  $j$  sólo puede ser asignado a una máquina  $A_j$ , y debe ejecutarse por completo y sin interrupciones en esta.

Sea  $A$  una asignación de trabajos a máquinas, y sea el *makespan* definido por:  $makespan(A) = \max_i \sum_{A_j=i} T_j$ . El objetivo del problema es determinar la asignación  $A$  que minimiza el *makespan*.

### 2.1. Algoritmo 2-competitivo propuesto

El algoritmo consiste simplemente en asignar cada trabajo  $j$  a la máquina que posea la menor carga. En particular, para cada  $j$ :

1. Obtener la máquina  $M_i$  de menor carga  $L_i$ .
2. Asignar  $j$  a  $M_i$
3.  $L_i := L_i + T_j$

La complejidad del algoritmo depende del paso 1. Usando un *heap* podemos obtener la máquina de menor carga en tiempo  $O(1)$  y luego reinsertarla en la estructura con su nueva carga en tiempo  $O(\log m)$ . Por lo tanto, para  $n$  trabajos tenemos un tiempo  $O(n \log m)$ .

---

<sup>1</sup>Garey, M.R. (1976). "The Complexity of Flowshop and Jobshop Scheduling"

## 2.2. Demostración de competitividad

Sea  $L_i$  la carga de la máquina con mayor tiempo asignado, y sea  $j$  su último trabajo. La carga de la máquina antes de esa asignación era  $L_i - T_j$ . Debido a que el algoritmo asigna un trabajo a la máquina de menor carga tenemos que  $L_i - T_j \leq L_k, \forall k \in [1, m]$ .

Luego, sumando sobre todas las máquinas obtenemos

$$\begin{aligned} L_i - T_j &\leq \frac{1}{m} \sum_{k=1}^m L_k \\ &= \frac{1}{m} \sum_{k=1}^n T_k \\ &\leq L^* \end{aligned} \tag{1}$$

Donde  $L^*$  corresponde al *makespan* del caso óptimo.

Finalmente, considerando (1) y el hecho de que  $T_j \leq L^*$ , ya que el trabajo más largo igual debe ser realizado por alguna máquina en el caso óptimo, tenemos que

$$\begin{aligned} L_i &= L_i - T_j + T_j \\ &\leq L^* + T_j \\ &\leq 2L^* \end{aligned} \tag{2}$$

Con lo que se demuestra que el *makespan* obtenido por el algoritmo online está acotado por el doble del *makespan* en el caso óptimo. O sea, el algoritmo es 2-competitivo.

## 2.3. Comprobación empírica de competitividad

Para comprobar empíricamente la competitividad del algoritmo propuesto, éste se implementó en Java junto a un algoritmo que encuentra el *makespan* óptimo para este problema. Luego se hicieron pruebas con ambos algoritmos, usando los mismos datos, y se compararon los *makespan* obtenidos.

### 2.3.1. Implementación del algoritmo online

El algoritmo online se implementó tal cual fue descrito en la sección 2.1. Se utilizó la clase *PriorityQueue* de *java.utils*, la cuál está implementada con un *heap*, para almacenar las máquinas ordenadas según carga. De este modo, se pueden planificar  $n$  tareas en  $m$  máquinas en tiempo  $O(n \log m)$ .

### 2.3.2. Implementación del algoritmo óptimo

Para obtener la asignación de una lista  $L$  de  $n$  trabajos a  $m$  máquinas que produce un menor *makespan* se procedió a generar todas las posibles asignaciones que pueden entregar distinto *makespan*. Para esto se utilizó un arreglo  $A$  de tamaño  $n$  donde cada

índice  $i \in [0, n - 1]$  representa un trabajo, y cada valor  $A_i$  representa la máquina que se le ha asignado al trabajo  $i$ . Además se utilizó un objeto  $G$  encargado de generar asignaciones para  $A$  a través de su método  $G.nextAssignment()$ , y de señalar mediante un método  $G.hasNext()$  si siguen habiendo asignaciones para generar. El algoritmo funciona del siguiente modo:

- Se comienza con  $A$  con todas sus entradas en cero.
- Se inicializa el entero  $minMakespan := Integer.MAX\_INT$
- Mientras  $G.hasNext()$  entregue verdadero:
  - Se asigna  $A' := G.nextAssignment()$ .
  - Se prueba la asignación  $A'$  con la lista  $L$  y  $m$  máquinas y se calcula su  $makespan$ .
  - Si  $makespan(A') < minMakespan$ :
    - Se copia  $A'$  en  $A$ .
    - $minMakespan := makespan(A')$ .
- Ahora  $A$  contiene la asignación que minimiza el makespan.

El objeto  $G$  mantiene un arreglo  $B$  interno que va modificando para producir cada asignación.  $B$  se inicializa con todas sus entradas en 0, representando que todos los trabajos son asignados a la máquina 0.  $G.nextAssignment()$  opera del siguiente modo:

- Se itera desde  $i = n - 1$  hasta  $i = 0$ :
  - Si  $i = 0$ , desde ahora  $G.hasNext()$  siempre retornará falso. Se sale del ciclo.
  - Si  $B_i + 1 < m$ , se asigna  $B_i := B_i + 1$  y se sale del ciclo.
  - En otro caso, se asigna  $B_i = 0$  y se continúa en el ciclo.
- Se retorna  $B$ , el cual representa una nueva asignación.

De esta forma se verifican todas las asignaciones posibles que comienzan asignándole el trabajo 0 a la máquina 0. No es necesario verificar otros casos ya que asignarle el primer trabajo a otra máquina no reducirá el makespan en ningún caso.

### 2.3.3. Experimentos

Para comprobar la competitividad del algoritmo online se procedió a comparar los makespan que entrega con los que entrega el algoritmo óptimo para la misma lista de trabajos y número de máquinas. Para esto se generaron listas con trabajos de duración aleatoria entre 1 y 100, y se obtuvieron los makespan para las asignaciones encontradas por ambos algoritmos.

### 2.3.4. Resultados

Se probó con listas de tamaño 8, 10, 12, 14 y 16, y 4 máquinas. Para cada tamaño de lista se produjeron 400 listas y se compararon los makespan obtenidos por ambos algoritmos.

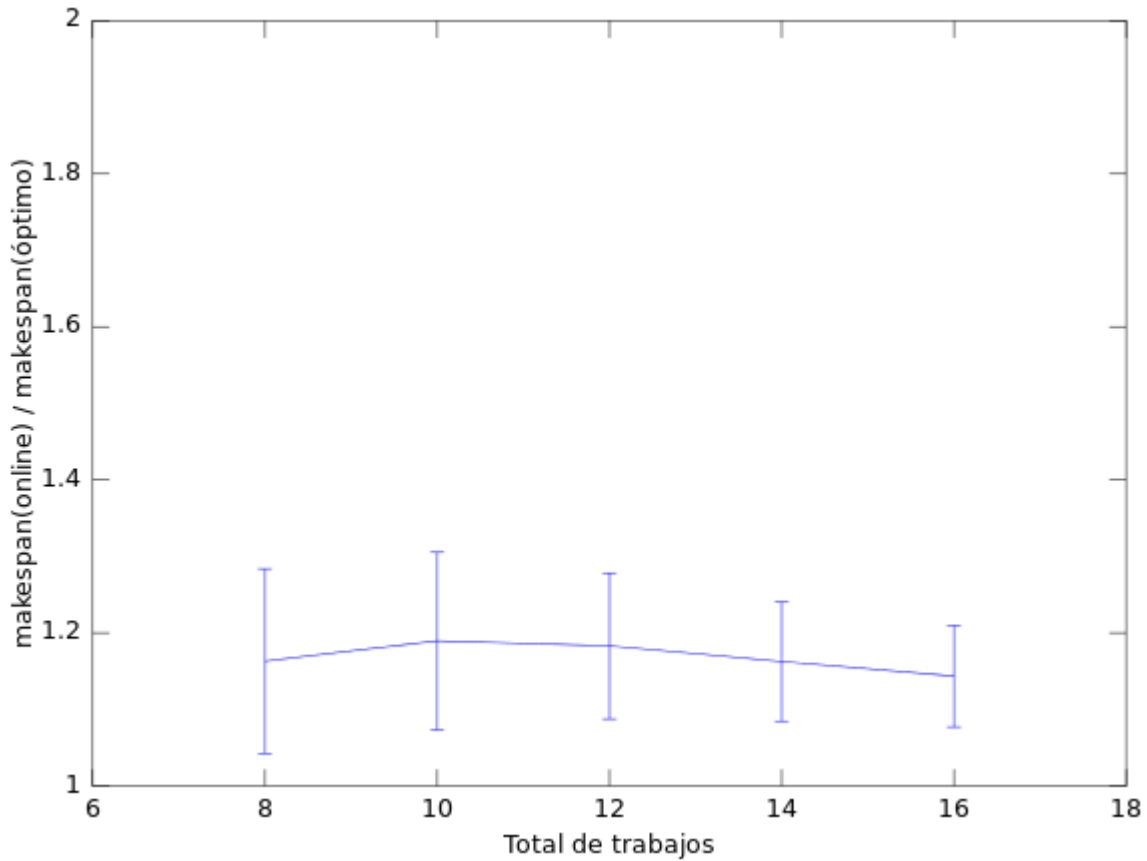


Figura 1: Comparación de *makespan* entre algoritmo online y algoritmo óptimo en función del número de trabajos. Cada punto representa la media entre 400 datos, y las barras son la desviación estándar.

### 3. Trabajos con múltiples etapas

#### 3.1. Algoritmo 2-competitivo propuesto

#### 3.2. Demostración de competitividad

#### 3.3. Comprobación empírica de competitividad

##### 3.3.1. Implementación del algoritmo online

##### 3.3.2. Implementación del algoritmo óptimo

##### 3.3.3. Experimentos

##### 3.3.4. Resultados

### 4. Conclusiones