*IX Congreso de Ingeniería de Organización*
*Gijón, 8 y 9 de septiembre de 2005*

# Un algoritmo *branch-and-bound* doble para resolver el problema flow-shop con bloqueos

**Ramón Companys Pascual[1], Manuel Mateo Doll[1]**

[1] Dpto. de Organización de Empresas. Escuela Politécnica Superior de Ingeniería Industrial de Barcelona, 08028 Barcelona. ramon.companys@upc.edu, manel.mateo@upc.edu.

## Resumen

*Se resuelve el problema de flow-shop sin espacio de almacenaje intermedio entre máquinas con una función objetivo consistente en minimizar el tiempo total de realización de todas las piezas. Se limita la consideración a soluciones consistentes en una única permutación. Se usa un algoritmo branch-and-bound doble, adecuado para el cálculo paralelo, y heurísticas auxiliares para alcanzar una buena solución inicial. El procedimiento heurístico se divide en dos pasos: la obtención de una permutación y la aplicación de un procedimiento de mejora local. La mejora se aplica sobre la heurística NEH. Ya que el problema de flow-shop con stocks intermedios es una relajación del problema sin ellos, los procedimientos desarrollados para este problema se pueden usar en ambos. Por ejemplo, se puede adaptar la acotación de Nabeshima o Lageweg. Además ambos problemas gozan de la propiedad de reversibilidad. Así, el un procedimiento branch and bound se puede aplicar a las instancias directa e inversa a la vez. El algoritmo de branch-and-bound LOMPEN se diseñó inicialmente para ejemplares de flow-shop con buffers y se ha adaptado fácilmente para ejemplares de flow-shop sin buffers. El algoritmo ha demostrado su potencia al determinar soluciones óptimas inéditas en ejemplares de la colección de Taillard.*

**Palabras clave:** programación, flow-shop con bloqueos, algoritmos branch-and-bound.

## 1. Introducción

Este trabajo considera el problema de programación en flujo regular sin espacio de almacenamiento entre máquinas. Dado que las $n$ piezas deben ser procesadas en las $m$ máquinas en idéntico orden, ya que la ausencia de espacio impide variaciones del orden inicial, este problema se reduce a determinar una permutación de las $n$ piezas. Se conoce los tiempos de operación de cada pieza $i$ en cada máquina $j$, $p_{j.i}$. La carencia de espacios de almacenaje entre etapas impide también que existan colas intermedias de piezas a la espera de su siguiente operación. Si una pieza $i$ finaliza su operación en una máquina $j$ y la siguiente máquina, $j+1$, aún está ocupada por la pieza anterior, la pieza $i$ finalizada tiene que permanecer en la máquina $j$ bloqueándola.

La función objetivo considerada en este trabajo será la minimización del tiempo total de finalización de todos los trabajos, $C_{max}$. Según la notación de Lawler *et al*. (1993) estos problemas se denominan $Fm|block|C_{max}$.

El problema de flujo regular ha sido uno de los temas que más intensamente se ha venido investigando en programación desde la publicación de Johnson (1954) sobre el problema $Fm|prmu|C_{max}$. Si el número de máquinas es dos, es fácil alcanzar una secuencia óptima en tiempo $O(n \log n)$ usando el algoritmo de Johnson. Para $m \geq 3$ se ha demostrado que el

problema es NP-hard (Garey et al. 1976). Lomnicki (1965) e Ignall y Schrage (1965) publicaron, casi simultáneamente, los primeros algoritmos de *branch-and-bound* para el problema Fm| *prmu* |$C_{max}$. Desde entonces se ha publicado otros algoritmos de *branch-and-bound*: Potts (1980), Companys (1999), …

La complejidad del problema Fm| *prmu* |$C_{max}$ ha producido el desarrollo de procedimientos heurísticos. Algunas de las heurísticas son Palmer (1965), Companys (1966), Campbell *et al*. (1970), Gupta (1971), Dannenbring (1977), Nawaz *et al*. (1983), etc. A esta lista se pueden añadir gran número de aplicaciones de los procedimientos de exploración local comunmente denominados metaheurísticas.

Recientemente Ladhari y Haouari (2004) presentaron un algoritmo de *branch-and-bound* que construye las permutaciones colocando alternativamente piezas en ambos extremos. Utilizan un subproblema de 2-máquinas como procedimiento de acotación, que proporciona una cota inferior muy ajustada, pero no se puede calcular en tiempo polinómico. El algoritmo de Ladhari utiliza la heurística NEH para obtener un valor como mejor solución inicial, o, para instancias más duras, una heurística de búsqueda basada en un original esquema de *branch-and-bound* (Haouari y Ladhari, 2003).

## 2. Descripción del problema

Nos limitamos a la situación en que piezas y máquinas están disponibles en el instante inicial 0, y se considera, como hemos dicho, la misma permutación de piezas en todas las máquinas. El tiempo de operación de la pieza $i$ ($i$ = 1, 2, ..., $n$) sobre la máquina $j$ ($j$ = 1, 2, ..., $m$) es $p_{j,i.}$

Sea [$k$] la pieza situada en la posición $k$ de una permutación de $n$ piezas. Sea $e_{j,k}$ el instante en que la pieza [$k$] entra en la máquina $j$ procedente de la máquina $j$−1 y $f_{j,k}$ el instante en que la pieza [$k$] abandona la máquina $j$ para dirigirse a la $j$+1 (en este caso, no existe cola intermedia). Considerando las características precedentes y asumiendo $p_{j,i} > 0$, se obtienen las siguientes ecuaciones para el caso Fm| *prmu* |$C_{max}$:

$$e_{j,k} = \max \{ f_{j,k-1} , f_{j-1,k} \} \qquad j = 1, 2, \ldots, m ; k = 1, 2, \ldots, n \qquad (1)$$

$$f_{j,k} = \max \{ e_{j,k} + p_{j,[k]} , f_{j+1,k-1} \} \qquad j = 1, 2, \ldots, m ; k = 1, 2, \ldots, n \qquad (2)$$

Con la misma notación la ecuación 2 para el problema Fm| *prmu* |$C_{max}$ sería:

$$f_{j,k} = e_{j,k} + p_{j,[k]} \qquad j=1,2,\ldots,m ; k=1,2,\ldots,n \qquad (2')$$

En consecuencia el problema Fm| *prmu* |$C_{max}$ puede verse como una relajación del problema Fm| *block* |$C_{max}$.

Para $k$ = 1, $j$ = 1 y $j$ = $m$, se supone que $f_{j,0}$, $f_{0,k}$ y $f_{m+1,k}$ se han definido de manera adecuada. Dada una permutación específica [$k$] ($k$ = 1 ,2, ..., $n$) y considerando $f_{j,0}$ = 0 para cualquier máquina $j$, y $f_{0,k} = f_{m+1,k} = 0$ para toda pieza $k$, pueden obtenerse los valores $e_{j,k}$ y $f_{j,k}$ mediante las ecuaciones (1) y (2). El tiempo de finalización de la última pieza es, por tanto, $F_{max} = C_{max} = f_{m,n}$. El problema se puede formular como: "dados $n$, $m$ y los tiempos $p_{j,i}$ encontrar una permutación de las $n$ piezas que minimice $C_{max}$".

Hall y Sriskandarajah (1996) publicaron un estado del arte sobre problemas de *flow-shop* con

bloqueos y sin esperas (*no-wait*). Si el número de máquinas es dos, Reddi y Ramamoorthy (1972) demostraron que existe un algoritmo polinomial, que proporciona una solución exacta. El problema F2 | *block* | $C_{max}$ se puede reducir a un problema de viajante de comercio (TSP, *Travelling Salesman Problem*) con $n$+1 ciudades (0, 1, 2, ..., $n$), siendo la distancia entre las ciudades $h$ e $i$ ($h \neq i$):

$$d_{h,i} = \left| \; p_{2,h} - p_{1,i} \; \right| \qquad h \, , \, i = 0, 1, 2, ..., n$$

Se asume que $p_{1,0} = p_{2,0} = 0$. La secuencia de ciudades en un camino óptimo conduce a la permutación óptima para el problema original. Existe un algoritmo en tiempo polinómico para resolver este problema, propuesto por Gilmore y Gomory (1964). Este algoritmo es O($n$ log $n$), según Hall y Sriskandarajah (1996).

Hall y Sriskandarajah (1996) muestran que el problema Fm| *block* | $C_{max}$ para $m \geq 3$ máquinas es NP-hard. McCormick *et al*. (1989) propusieron algunas heurísticas basadas en un problema de flujo máximo. Leisten (1990) comparó 11 heurísticas usando 90 formatos de stock intermedio (con $m$=2 y $m$=3) y llegó a la conclusión que la heurística NEH (Nawaz *et al.*, 1983), inicialmente propuesta para stocks ilimitados, se comportaba mejor que las otras 10.

Dado un conjunto de datos ($n$, $m$ y $p_{j,i}$), el objetivo es aplicar varias heurísticas (en especial, NEH+) para obtener una permutación inicial y más tarde utilizarla como punto de partida de un *branch-and-bound* doble, LOMPEN, considerando instancias del problema Fm| *block* | $C_{max}$.

## 3. El algoritmo LOMPEN para Fm½*block*½$C_{max}$

### 3.1. La propiedad de reversibilidad

La propiedad de reversibilidad fue establecida inicialmente en los problemas Fm| *prmu* | $C_{max}$. Dada una instancia *I*, que llamaremos instancia directa, con tiempos unitarios de operación $p_{j,i}$, se puede determinar otra instancia asociada *I'*, que se llamará instancia inversa, cuyos tiempos de operación $p'_{j,i}$:

$$p'_{j,i} = p_{m-j+1,i} \qquad j = 1, 2, ..., m \; ; \qquad i = 1, 2, ..., n$$

El valor $C_{max}$ en *I* para una permutación *S* es idéntico al hallado para la permutación inversa *S'* en la instancia *I'*. Por tanto, el tiempo de finalización mínimo es el mismo para *I* e *I'*, y las permutaciones asociadas son inversas entre sí. No importa resolver *I* o *I'*. Aplicando un algoritmo de *branch-and-bound*, por ejemplo el de Lomnicki (1965), a veces se resuelve más rápidamente con la instancia inversa que con la directa.

La propiedad puede extenderse, con idénticas características, a los problemas Fm| *block* | $C_{max}$.

### 3.2. La heurística NEH+

La heurística NEH+ es la suma de la heurística NEH y un procedimiento de mejora local (NEDM-RCT). La heurística NEH, propuesta por Nawaz *et. al.* (1983) para los problemas Fm| *prmu* | $C_{max}$, se podría resumir así:

- Paso 1: ordenar las *n* piezas por orden decreciente de suma de sus duraciones.
- Paso 2: tomar las dos primeras piezas y programarlas para minimizar el tiempo total de la subsecuencia como si el problema fuera Fm| *prmu* |$C_{max}$ con 2 piezas.
- Paso 3: *k* = 3, …, *n*, insertar la pieza *k* en la posición en que en la secuencia parcial, entre las *k* posibles, minimice el tiempo total para Fm| *prmu* |$C_{max}$ con k piezas; para deshacer empates, se optar por la secuencia que menor tiempo muerto total.

Si se aplica la heurística NEH a una instancia *I* se obtiene una permutación $S_1$, y un tiempo de liberación del taller $C_{max}(S_1)$. Si se aplica la heurística NEH a la instancia inversa *I'*, la permutación $S_2$' que se obtiene no será, por lo general, la inversa de $S_1$, y $C_{max}(S_2')$ puede diferir de $C_{max}(S_1)$. Se llamará NEH2 a la heurística consistente en aplicar NEH a ambas instancias, directa e inversa, reteniendo la mejor solución alcanzada. NEH2 es más eficiente que NEH pero requiere más tiempo de cálculo.

En la heurística NEH, los pasos 2 y 3 se puede aplicar a una secuencia inicial de piezas diferente del orden indicado en el paso 1, como en Watson *et al*. (1999) y Ronconi (2004). En el paso 1 de NEH, los empates de tiempos de finalización parciales se deshacen considerando la suma de tiempos muertos y tiempos de bloqueo en las máquinas.

La heurística NEH+ realiza una búsqueda local usando como mejor solución inicial la que proporciona NEH. El vecindario de la solución se define como el conjunto de $n\cdot(n-1)/2$ permutaciones obtenidas intercambiando dos posiciones cualesquiera en la permutación en curso. El vecindario se explora según un orden *a priori* fijo. Cuando un vecino es mejor que la solución en curso, se convierte en la nueva solución en curso y la exploración sigue en un nuevo vecindario.

Si ningún vecino mejora la solución en curso, se realiza una nueva exploración del vecindario teniendo en cuenta las permutaciones con igual $C_{max}$ que la solución en curso. Estas se convierten en nueva soluciones en curso sólo con una cierta probabilidad (por ejemplo, 0,5). El número de diferentes soluciones en curso aceptadas sin mejora de $C_{max}$ está limitado (por ejemplo, el límite puede ser 40). Se llama a este procedimiento de búsqueda local NEDM-RCT (*non-exhaustive descent method with random consideration of ties*).

Se llama NEH2+ a la heurística consistente en aplicar NEH+ a ambas instancias, directa e inversa, reteniendo la mejor solución alcanzada.

Las heurísticas NEH, NEH2, NEH+ y NEH2+ son adaptables al problema Fm| *block* |$C_{max}$. Basta substituir en todos los pasos y en el cálculo de $C_{max}$, las expresiones (2') por las (2).

### 3.3. El algoritmo LOMPEN

Companys (1993, 1999) desarrolló un algoritmo (LOMPEN, LOMnicki PENdular) basado en la propiedad de reversibilidad para el problema Fm| *prmu* |$C_{max}$. LOMPEN consiste en aplicar simultáneamente dos exploraciones *branch-and-bound* a las instancias *I* e *I'*. Dado que la propiedad de reversibilidad es extensible a los problemas Fm| *block* |$C_{max}$, también en ellos es aplicable la idea del LOMPEN. Respecto al algoritmo original en Companys (1999), no hay diferencias en la definición de vértices, la ramificación, la eliminación de vértices y la estrategia de exploración. Describimos primero la exploración aplicada a cada instancia y después las relaciones e intercambios de datos entre ambos procesos.

*Definición de vértices*: En el nivel $r$ ($r$ = 0, 1, .., $n$−1) un vértice corresponde a una secuencia inicial parcial $s$ de $r$ piezas. A nivel 0 sólo existe un vértice σ que está vacío.

*Ramificación*: Cada vértice se representa mediante $s$, permutación de $r$ piezas ($r < n$). $\boldsymbol{J}$ representa el conjunto de piezas que se encuentran en $s$ y $\bar{\boldsymbol{J}}$ el conjunto de aquellas aún no secuenciadas. Un sucesor inmediato de $s$ será $si$ con $i \in \bar{\boldsymbol{J}}$ para cualquier pieza $i$ factible en la posición $r$+1 (para aclarar el calificativo factible, véase la Regla 2).

*Cotas inferiores*: Asociado al vértice $s$, existe una cota inferior sobre el tiempo de total para cualquier secuencia que empiece con la subsecuencia inicial $s$. Lageweg et al. (1978), incluyendo ideas de Nabeshima (1967), propusieron la *cota dos-máquinas* que se construye como suma de tres términos, dadas dos máquinas $j$ y $k$ ($1 \le j \le k \le m$):

a) El instante $f_{j,r}(s)$, en que la máquina $j$ queda libre después de la piezas que forman parte de la permutación parcial inicial $s$.
b) Una cota del tiempo consumido por la última pieza de $\bar{\boldsymbol{J}}$ después de abandonar la máquina $k$.
c) Una cota del tiempo necesario para el subproblema definido por todas las operaciones de las piezas de $\bar{\boldsymbol{J}}$ en las máquinas entre $j$ y $k$, ambas incluidas.

Cada par ($j$,$k$) proporciona una cota y la cota inferior de $s$, que se notará por $b(s)$, se escoge como el máximo de las cotas dos-máquinas calculadas. En el vértice $s$, con $r$ piezas secuenciadas, los valores $f_{j,t}(s)$ ($t$ = $r$+1, ..., $n$; $j$ = 1, 2, ..., $m$) están acotados por valores $D_{j,t}$. Para la máquina $j$ = 1: $D_{1,r+1} = f_{2,r}(s)$; $D_{1,r+2} = f_{3,r}(s)$; …; $D_{1,r+m-1} = f_{m,r}(s)$. Y, en general, para el resto de las máquinas $j$ = 2, .., $m$−1: $D_{j,r+s} = f_{j+s,r}(s)$ con $r+s \le n$; $j \le m-s$.

Estas expresiones permiten la cota de los tiempos de bloqueo en las máquinas producido por las últimas $n$–$r$ piezas comparando los valores $D_{j,t}$ con las duraciones acumuladas de estas $n$–$r$ piezas, de manera parecida a como hicieron Ronconi y Armentano (2001).

Para una máquina $j$, los valores $p_{j,i}$ ($i \in \bar{\boldsymbol{J}}$) se ordenan de manera decreciente y se llaman $\bar{p}_{j,s}$ con $s$= 1, 2, ..., $n$–$r$, siendo $\bar{P}_{j,s} = \sum_{t=1}^{s} \bar{p}_{j,t}$. $BT_j(s)$, cota inferior del tiempo de bloqueo en la máquina $j$, se calcula para las operaciones en piezas de $\bar{\boldsymbol{J}}$:

$$bt_s = bt_{s-1} + \max \{ 0 ; D_{j,r+s} - (f_{j,r}(s) + \bar{P}_{j,s} + bt_{s-1}) \} \quad s = 1, 2, .., u \text{ con } bt_0 = 0$$

$$BT_j(s) = bt_u \qquad \text{siendo } u = \min \{ n-r , m-j \}$$

El término (c) se calcula de la siguiente forma. Si $k = j$, es $\sum_{i \in \bar{\boldsymbol{J}}} p_{j,i} + BT_j(s)$. Para $k = j$+1, se puede estimar siguiendo la sugerencia de Reddi y Ramamoorthy (1972) como extensión de las cotas de Nabeshima (1967), resolviendo el problema TSP asociado con $n$–$r$+1 ciudades. Las distancias se basan en las duraciones en las máquinas $j$ y $j$+1 respecto a las $n$–$r$ piezas del conjunto $\bar{\boldsymbol{J}}$ y una ciudad adicional 0 con duraciones: $p_{1,0} = 0$; $p_{2,0} = f_{j+1,r}(s)$

– $f_{j,r}(s)$. Para $k > j+1$, se puede obtener relajando el subproblema. Se usa como relajación el problema F2 | $l_i$,*prmu* | $C_{max}$ asociado.

*Mejor solución en curso*: Alguno autores prefieren llamarla "cota superior". Inicialmente es el valor de una buena solución que utiliza un método heurístico. En nuestro caso se alcanzaron buenos resultados usando la heurística NEH+. Cuando se llega a un vértice de nivel $n-1$, $\bar{J}$ es un conjunto con un solo elemento; si se añade el elemento restante queda completamente definida una permutación de n piezas. En el problema Fm| *prmu* | $C_{max}$, $b(s)$ es el valor de tiempo de liberación del taller para dicha permutación; en cambio, en el caso del problema Fm| *block* | $C_{max}$ esto puede no ser cierto y debe calcularse independientemente.

*Eliminación de vértices*: Cuando se ha generado todos aquellos vértices factibles descendientes de un vértice, el vértice padre se elimina. Asimismo, si la cota de un vértice es mayor o igual a la mejor solución en curso, se poda el vértice. Otra manera de eliminar un vértice es por dominancia, como se explica más adelante.

*Estrategia de búsqueda*: El número de vértices activos $N$ se considera para seleccionar qué vértice se va a ramificar. Se fijan dos números enteros $N_0$ y $N_1$ ($0 < N_0 < N_1$). Si $0 < N < N_0$, se selecciona un vértice activo con menor $b(s)$ en el nivel más bajo del árbol; si $N_0 \leq N < N_1$, se selecciona un vértice activo con menor $b(s)$, y se dilucidan los empates a favor del vértice en el nivel más alto; si $N_1 \leq N$, entonces se selecciona un vértice en el nivel más elevado, y se desempata escogiendo el vértice con menor $b(s)$.

### 3.4. Interrelación entre procesos de exploración

Ambos procesos de exploración comparten la misma cota y la mejor solución alcanzada, y otros datos deducidos de la siguiente cuatro reglas, diseñadas para mejorar el comportamiento del algoritmo:

*Regla 1*: En un momento dado de la evolución del algoritmo sea $\boldsymbol{T}$ el conjunto de los vértices abiertos (pendientes) de $I$ y $\boldsymbol{T'}$ el conjunto de los vértices abiertos de $I'$. La mejor cota de los vértices de $\boldsymbol{T}$ coincidirá con la mejor cota de los vértices de $\boldsymbol{T'}$, ya que en caso contrario sería necesario modificar la más pequeña de ambas.

Las primeras piezas en las secuencias $s(I)$ de los vértices de $\boldsymbol{T}$ suelen ser sólo un subconjunto (a veces reducido) de todas las piezas y estas piezas son las únicas que pueden ocupar la última posición de las secuencias de $\boldsymbol{T'}$ cuyo valor sea mejor al mejor hallado hasta el momento, lo cual puede permitir ajustar las cotas. Esta información puede servir para mejorar las cotas.

*Regla 2*: Si en $\boldsymbol{T}$ una pieza $i$ está situada antes de la posición $r+1$ en todos los vértices con cota $b(s) < b_0$, entonces la cota de todos los vértices de $\boldsymbol{T'}$ con $i$ en las primeras $n-r$ posiciones debe ser $b(s') \geq b_0$. Si $b_0$ es la mejor solución encontrada podemos eliminar (cerrar) todos los vértices de $\boldsymbol{T'}$ con la pieza $i$ en las $n-r$ primeras posiciones.

*Regla 3* (función SPLIT): El algoritmo LOMPEN puede interpretarse, considerando un instante de origen arbitrario, como la aplicación en paralelo de un conjunto de reglas a partir de una pareja inicial ($\boldsymbol{T}$, $\boldsymbol{T'}$). Sea la partición de $\boldsymbol{T}$ en dos conjuntos $\boldsymbol{T1}$ y $\boldsymbol{T2}$. El

algoritmo es equivalente a la aplicación en paralelo del mismo conjunto de reglas a las dos parejas iniciales (***T1***, ***T'***) y (***T2***, ***T'***). En esta meta-aplicación ambas aplicaciones compartirán la mejor solución alcanzada, pero no la mejor cota que será interna a cada aplicación individual. Eligiendo adecuadamente la partición puede conseguirse que, consecuencia de la regla 1 o la regla 2, alguna de las aplicaciones parciales llegue rápidamente a su fin, con lo que se habrá logrado, en definitiva, reducir la dimensión de los árboles a explorar.

*Regla 4*: Las mismas propiedades sirven también para ***T'*** versus ***T***.

Si se usan las reglas 1 y 2, sólo se puede eliminar vértices por dominancia en ***T*** o en ***T'***.

## 4. Experiencia computacional

El algoritmo LOMPEN se aplica sobre 9 colecciones de un millar de ejemplares cada una, con n = 13, 14, 15 y m = 3, 4, 5. La duración de operaciones es aleatoria siguiendo una distribución uniforme entre 1 y 25. Para cada ejemplar se calcula mediante LOMPEN una permutación óptima y su valor $C_{max}$ asociado, usando como solución inicial NEH+ (la suma de NEH y una exploración local). En una segunda fase, se ha evaluado una nueva versión de LOMPEN sobre 100 de los conocidos ejemplares de Taillard (1983), aquellos que combinan m = 5, 10, 20 y n = 20, 50, 100 y además los 10 ejemplares con m = 10 y n = 200.

La experiencia general fue diseñada para evaluar el impacto de la búsqueda local (NEDM-RCT) en la mejora de la solución inicial. La segunda experiencia ha pretendido como objetivo principal comprobar el comportamiento del algoritmo LOMPEN en ejemplares de mayores dimensiones y reconocidos por la comunidad científica.

### 4.1. Experiencia principal

LOMPEN estaba codificado en Visual BASIC e implementado en un Pentium IV 2.8 GHz PC con 512 Mb RAM. El algoritmo no usa en este caso la regla 3 (la función *split*) ni la eliminación por dominancia de vértices. Los valores $N_0$ y $N_1$, para la estrategia de búsqueda, son: $N_0 = 100$; $N_1 = 150$.

Para medir el impacto de NEDM-RCT, se evaluó la mejora en la solución NEH+ respecto a la solución NEH. Para ello, se determinó la discrepancia relativa en porcentaje para cada ejemplar antes y después de aplicar NEDM-RCT. La discrepancia relativa es igual a la diferencia entre el valor de la heurística y el valor óptimo (obtenido más tarde gracias a LOMPEN) dividido por el óptimo.

La tabla 1 presenta la discrepancia relativa media asociada a las heurísticas NEH, NEH+, NEH2 y NEH2+ en el caso $Fm|block|C_{max}$. El impacto de NEDM-RCT en el caso $Fm|block|C_{max}$ en la solución inicial de LOMPEN es importante. La mejora absoluta en media sobre la discrepancia relativa media producida por la búsqueda local es del 2% aproximadamente, tanto en NEH como en NEH2.

**Tabla 1**. Discrepancia relativa media de las soluciones dadas por las heurísticas (%) para Fm | *block* | $C_{max}$.

| *m* | *n* | NEH | NEH+ | NEH2 | NEH2+ |
|---|---|---|---|---|---|
| 3 | 13 | 4.065 | 1.846 | 3.444 | 1.382 |
| | 14 | 4.160 | 1.918 | 3.524 | 1.414 |
| | 15 | 4.471 | 2.096 | 3.818 | 1.573 |
| 4 | 13 | 4.289 | 2.211 | 3.671 | 1.681 |
| | 14 | 4.412 | 2.312 | 3.892 | 1.792 |
| | 15 | 4.917 | 2.496 | 4.262 | 1.893 |
| 5 | 13 | 4.152 | 2.239 | 3.666 | 1.733 |
| | 14 | 4.484 | 2.346 | 3.938 | 1.836 |
| | 15 | 4.782 | 2.501 | 3.051 | 0.827 |

Los tiempos computacionales de LOMPEN (incluyendo la obtención de la solución inicial mediante NEH+) en el caso Fm | *block* | $C_{max}$ se presentan en la tabla 2.

**Tabla 2**. Tiempo CPU medio y máximo (s/instancia) de LOMPEN (Fm | *block* | $C_{max}$).

| | | *tiempo CPU medio* | | | *tiempo CPU máximo* | | |
|---|---|---|---|---|---|---|---|
| | | m=3 | m=4 | m=5 | m=3 | m=4 | m=5 |
| n | 13 | 0.705 | 2.026 | 3.639 | 207 | 42 | 67 |
| | 14 | 3.348 | 7.441 | 18.984 | 1128 | 180 | 372 |
| | 15 | 7.772 | 28.321 | 70.737 | 787 | 879 | 1494 |

Cuanto mayor es el número de máquinas, la calidad de las soluciones de la heurística empeora, y asimismo el tiempo consumido por un algoritmo LOMPEN crece exponencialmente. Curiosamente, el ejemplar #952 para m = 5 y n = 13 consume sólo 3,24 s. en Fm | *block* | $C_{max}$, cuando era la más difícil de resolver en Fm | *prmu* | $C_{max}$ con 931,2 s. de tiempo. De los 9000 ejemplares tratados en la versión Fm | *block* | $C_{max}$ sólo 3 consumen más de 931,2 segundos. El ejemplar 104 de m = 5 y n = 15 consume 1494 segundos de tiempo CPU. En cambio, este ejemplar en el caso Fm | *prmu* | $C_{max}$ se resuelve en menos de 1 segundo.

### 4.2. Experiencia complementaria

Esta experiencia tenía como objetivo comprobar el comportamiento del algoritmo LOMPEN en ejemplares de grandes dimensiones del problema Fm | *block* | $C_{max}$. El código utilizado, fue realizado por Alemán (2004) en C usando Dev-C++, compilado con GCC y se llevó a cabo en un Pentium IV 2.8 GHz PC con 512 Mb RAM sobre KNOPPIX/Debian GNU/Linux 3.4. Esta versión de LOMPEN obtiene inicialmente 10 soluciones aplicando 5 heurísticas directas, como Palmer, trapecios (Companys, 1966), Gupta, NEH y PF (McCormick *et al.*, 1989) a las secuencias directa e inversa, y luego se aplica una mejora local sobre éstas. La mejora de las 10 soluciones restantes sirve como mejor solución. LOMPEN usa también la función *split*, y prueba de enlazar subsecuencias de ambos árboles. Los valores $N_0$ y $N_1$ están en función del número de piezas, $n$: $N_0 = 2 \cdot n$; $N_1 = 2 \cdot n + 100$.

Los datos utilizados son los ejemplares de Taillard (1993), a pesar que algunos autores los critiquen como Watson *et al* (1999). LOMPEN no alcanza ninguna solución óptima que se pueda validar en un tiempo razonable en los ejemplares de Taillard. Inicialmente se usó LOMPEN como una heurística, limitando el tiempo CPU a 20 minutos, en 100 ejemplares de

Taillard (n = 20, 50, 100, 200; m = 5, 10, 20). En la tabla 3 se muestran las soluciones. Los encabezamientos tienen el siguiente significado: *instancia*: nombre del ejemplar; *FBSV*: *final best solution value* de LOMPEN (mejor solución Fm|*block*|$C_{max}$).

En general, FBSV, la mejor solución final, no es óptima, aunque en 7 ejemplares marcados con * se ha comprobado que sí lo es. Para confirmarlo, se ha usado los códigos de ambas experiencias alimentados con la mejor solución conocida como solución inicial, con tiempo CPU limitado a 12 horas. Siguiendo en nuestras comparaciones podemos afirmar que los ejemplares *tail003* y *tail004* son igualmente duros cuando se tratan considerando el caso Fm|*prmu*|$C_{max}$, ya que se alcanza la solución óptima y su confirmación en un tiempo del orden de 1 s. Sin embargo en el caso Fm|*block*|$C_{max}$ parece que *tail003* es más duro de resolver que *tail004*.

## 5. Conclusiones

En este trabajo se muestra el impacto de la búsqueda local en la heurística NEH, cuyo impacto se puede extender a otras heurísticas. Se define un procedimiento de mejora por búsqueda local que tiene en cuenta empates (NEDM-RCT). Otra mejora consiste en aplicar las heurísticas a las instancias directa e inversa.

Se muestra la aplicación de LOMPEN al problema Fm|*block*|$C_{max}$, y como se adapta la cota 2-máquinas a este problema y las relaciones entre las exploraciones directa e inversa. LOMPEN se comporta de manera diferente para las mismas instancias en Fm|*prmu*|$C_{max}$ y en Fm|*block*|$C_{max}$. Como causa, se podría apuntar el hecho que el problema Fm|*block*|$C_{max}$ es más duro que el Fm|*prmu*|$C_{max}$, o bien que los procedimientos conocidos son menos eficientes en este caso.

También se incluye la aplicación de LOMPEN a ejemplares de gran dimensión, conocidos como Taillard *benchmarks*. La experiencia computacional principal ya había sugerido el potencial de LOMPEN para ejemplares grandes. LOMPEN ha resuelto por primera vez 7 ejemplares: tail001, tail002, tail004, tail005, tail006, tail007, tail009 de the Fm|*block*|$C_{max}$. Un ejemplar con los mismos datos numéricos a veces cuesta de resolver considerando el caso Fm|*prmu*|$C_{max}$ y en cambio no como Fm|*block*|$C_{max}$, y al contrario.

El algoritmo LOMPEN está pensado para aplicarse en dos procedimientos *branch and bound* paralelos con intercambio de datos entre ellos, con lo cual LOMPEN es apto para la paralelización. Aunque se han realizado algunos experimentos al respecto es necesario un estudio más profundo para llegar a conclusiones generales.

**Agradecimientos**

Esta investigación ha sido financiada por el proyecto DPI2001-2169 (MCYT).

**Tabla 3**.Soluciones obtenidas con LOMPEN como heurística, tiempo máximo CPU = 20 min (Fm|*block*|$C_{max}$).

| *n ´ m* | *instancia* | *FBSV* | *n ´ m* | *instancia* | *FBSV* | *n ´ m* | *instancia* | *FBSV* |
|---|---|---|---|---|---|---|---|---|
| | tail001 | 1374* | | tail011 | 1701 | | tail021 | 2436 |
| | tail002 | 1408* | | tail012 | 1833 | | tail022 | 2236 |
| | tail003 | 1280 | | tail013 | 1659 | | tail023 | 2479 |
| | tail004 | 1448* | | tail014 | 1535 | | tail024 | 2348 |
| 20 × 5 | tail005 | 1341* | 20 × 10 | tail015 | 1617 | 20 × 20 | tail025 | 2439 |
| | tail006 | 1363* | | tail016 | 1590 | | tail026 | 2389 |
| | tail007 | 1381* | | tail017 | 1622 | | tail027 | 2390 |
| | tail008 | 1379 | | tail018 | 1731 | | tail028 | 2328 |
| | tail009 | 1373* | | tail019 | 1747 | | tail029 | 2363 |
| | tail010 | 1283 | | tail020 | 1782 | | tail030 | 2324 |
| | tail031 | 3024 | | tail041 | 3710 | | tail051 | 4574 |
| | tail032 | 3234 | | tail042 | 3559 | | tail052 | 4357 |
| | tail033 | 3055 | | tail043 | 3556 | | tail053 | 4358 |
| | tail034 | 3145 | | tail044 | 3728 | | tail054 | 4449 |
| 50 × 5 | tail035 | 3189 | 50 × 10 | tail045 | 3696 | 50 × 20 | tail055 | 4346 |
| | tail036 | 3209 | | tail046 | 3662 | | tail056 | 4379 |
| | tail037 | 3071 | | tail047 | 3751 | | tail057 | 4391 |
| | tail038 | 3101 | | tail048 | 3634 | | tail058 | 4415 |
| | tail039 | 2940 | | tail049 | 3603 | | tail059 | 4412 |
| | tail040 | 3163 | | tail050 | 3700 | | tail060 | 4508 |
| | tail061 | 6222 | | tail071 | 7119 | | tail081 | 7969 |
| | tail062 | 6107 | | tail072 | 6881 | | tail082 | 7986 |
| | tail063 | 5986 | | tail073 | 7009 | | tail083 | 7968 |
| | tail064 | 5827 | | tail074 | 7272 | | tail084 | 7952 |
| 100 × 5 | tail065 | 6073 | 100 × 10 | tail075 | 6944 | 100 × 20 | tail085 | 7980 |
| | tail066 | 5916 | | tail076 | 6743 | | tail086 | 8033 |
| | tail067 | 6077 | | tail077 | 6877 | | tail087 | 8086 |
| | tail068 | 5984 | | tail078 | 6955 | | tail088 | 8127 |
| | tail069 | 6214 | | tail079 | 7167 | | tail089 | 8021 |
| | tail070 | 6199 | | tail080 | 7093 | | tail090 | 8066 |
| | | | | tail091 | 13543 | | | |
| | | | | tail092 | 13484 | | | |
| | | | | tail093 | 13623 | | | |
| | | | | tail094 | 13464 | | | |
| | | | 200 × 10 | tail095 | 13498 | | | |
| | | | | tail096 | 13299 | | | |
| | | | | tail097 | 13699 | | | |
| | | | | tail098 | 13643 | | | |
| | | | | tail099 | 13458 | | | |
| | | | | tail100 | 13579 | | | |

## Referencias


Alemán, A., 2004. *Estudio y Aplicación del Algoritmo Lomnicki Pendular al Problema Fm|block|Fmax*, PFC, ETSEIB-UPC

Campbell, H. G., Dudek, R. A., Smith, M. L., 1.970. “A heuristic algorithm for the n job m machine sequencing problem”, *Management Science* **16** (10), pp. 630-637
Companys, R., 1966. "Métodos heurísticos en la resolución del problema del taller mecánico", *Estudios Empresariales* **5** (2), pp. 3-14.
Companys, R., 1993. “Algoritmo de Lomnicki pendular”, *Notas sobre el problema del taller mecánico*, D.I.T. 93/12, ETSEIB-UPC, pp. 51-58.
Companys, R., 1999. “Note on an improved branch-and-bound algorithm to solve n/m/P/Fmax problems”, *TOP* **7** (1), pp. 25-31.
Dannenbring, D. G., 1977. “An evaluation of flow-shop sequencing heuristics”, *Management Science* **23**, pp. 1.174-1.182
Garey, M. R., Johnson, D. S., Sethi, R., 1976. “The complexity of flow shop and job shop scheduling”, *Mathematics of Operations Research* **1**, pp. 117-129
Gilmore, P. C., Gomory, R. E., 1964. “Sequencing a one state-variable machine: a solvable case of the traveling salesman problem”, *Operations Research* **12**, pp. 655-665
Gupta, J. N. D., 1971. “A functional heuristic algorithm for the flow-shop scheduling problem”, *Operational Research Quarterly* **22** (1)
Hall, N. G., Sriskandarajah, C., 1996. “A survey of machine scheduling problems with blocking and no wait in process”, *Operations Research* **44** (3), pp. 510-525.
Haouari, M., Ladhari, T., 2003. “A branch-and-bound-based local search method for the flow shop problem” *Journal of the Operational Research Society* **54**, pp. 1076-1084
Ignall, E., Schrage, L. E., 1965. “Application of the branch-and-bound algorithm to some flow shop problems”, *Operations Research* **13**, pp. 400-412
Johnson, S. M., 1954. "Optimal two- and three-stage production schedules with set-up time included", *Naval Research Logistics Quarterly* **1** (1), pp. 61-68.
Ladhari, T., Haouari, M., 2005. “A computational study of the permutation flow shop problem based on a tight lower bound”, *Computers & Operations Research* **32**, pp. 1831-1847.
Lageweg, B. J., Lenstra, J. K., Rinnooy Kan, A. H. G., 1978. “A general bounding scheme for the permutation flow-shop problem”, *Operations Research* **26** (1), pp. 53-67
Lawler, E. L., Lenstra, J. K., Rinnooy Kan, A. H. G., Shmoys, D. B., 1993. “Sequencing and scheduling: algorithms and complexity”, *in* Graves, C. G., Rinnooy Kan, A. H. G., Zipkin, P. (eds) *Handbooks in Operations Research and Management Science, Vol. 4: Logistics of Production and Inventory*, North-Holland, Amsterdam, pp. 445-522
Leisten, R., 1990. “Flowshop sequencing problems with limited buffer storage”, *Int. J. Prod. Res.* **28** (11) 2085-2100
Lomnicki, Z. A., 1965. “A branch and bound algorithm for the exact solution of three-machine scheduling problems”, *Operational Research Quarterly* **16** (1), pp. 89-100.
McCormick, S. T., Pinedo, M. L., Shenker, S., Wolf, B., 1989. “Sequencing in an Assembly Line with Blocking to Minimize Cycle Time”, *Operations Research* **37**, pp. 925-935.
Nabeshima, I., 1967. “On the bound of makespans and its application to m machine scheduling problem”, *Journal of the Operations Research Society of Japan* **9**, pp. 98-136.
Nawaz, M., Enscore E. E., Ham, I., 1983. “A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem”, *OMEGA* **11**, pp. 91-95.
Palmer, D. S., 1965 “Sequencing jobs through a multi-stage process in the minimum total time – A quick method of obtaining a near optimum” *Operational Research Quarterly* **16** (1), pp. 101-107
Potts, C. N., 1980. “An adaptive branching rule for the permutation flow-shop problem”, *European Journal of Operational Research* **5**, pp. 19-25
Reddi, S. S., Ramamoorthy, C. V., 1972. “On the flow-shop sequencing problem with no wait in process”, *Operational Research Quarterly* **23** (3), pp. 323-331

Ronconi, D. P., 2004 “A note on constructive heuristics for the flowshop problem with blocking” *Int. J. Production Economics* **87** pp. 39-48
Ronconi, D. P., Armentano, V. A., 2001. “Lower bounding schemes for flowshops with blocking in-process”, *Journal of the Operational Research Society* **52**, pp. 1289-1297
Taillard, E., 1993. “Benchmarks for basic scheduling problems”, *European Journal of Operational Research* **44**, pp. 375-382
Watson, J. P., Barbulescu, L., Howe, A. E., Whitney, L. D., 1999. “Algorithm performance and problem structure for flow-shop scheduling” *Proceedings 16th National Conference on AI.*