3rd International Conference on Industrial Engineering and Industrial Management
XIII Congreso de Ingeniería de Organización
Barcelona-Terrassa, September 2nd-4th 2009

Talleres de Flujo Híbrido con Tiempos de Cambio Dependientes de la Secuencia. Algoritmos basados en Búsqueda Local Iterativa*

Rubén Ruiz

Grupo de Sistemas de Optimización Aplicada. Instituto Tecnológico de Informática de Valencia. Universidad Politécnica de Valencia. Camino de Vera s/n, 46022, Valencia. rruiz@eio.upv.es.

Keywords: Programación de la producción, Taller de flujo híbrido, tiempos de cambio dependientes de la secuencia, búsqueda local iterativa.

1. Introducción

La realidad productiva normalmente arroja problemas de tipo híbrido, donde ni tenemos talleres de flujo con una única máquina por etapa, ni una única etapa con máquinas paralelas. Lo más normal es encontrarnos con un conjunto de etapas productivas y varias máquinas disponibles por cada etapa. Adicionalmente, en un contexto productivo multi-producto, es necesario con frecuencia reconfigurar las máquinas después de la producción de cada producto. Este tiempo de reconfiguración (también llamado tiempo de cambio) suele ser dependiente de la secuencia de producción. La importancia de considerar los tiempos de cambio de forma explícita se detalla en Allahverdi et al. (1999) en Allahverdi et al. (2008) y en Allahverdi y Soroush (2008), por poner solo unos ejemplos.

De manera más concreta, en este trabajo estudiamos el taller de flujo híbrido y flexible con tiempos de cambio dependientes de la secuencia de producción con el objetivo de minimizar el tiempo máximo de finalización o makespan. Formalmente, definimos este taller de flujo híbrido y flexible como sigue: tenemos un conjunto N de trabajos, $N = \{1, \ldots, n\}$ a procesar en un conjunto M de etapas, $M = \{1, \ldots, m\}$. En cada etapa i, $i \in M$, tenemos un conjunto $M_i = \{1, \ldots, m_i\}$ de máquinas paralelas idénticas. Cada máquina en cada etapa es capaz de procesar todos los trabajos (no hay restricciones de elegibilidad). Cada trabajo se tiene que procesar todas las etapas en exactamente una máquina de entre las m_i máquinas paralelas idénticas disponibles en la etapa i. Sin embargo, los trabajos no tienen porqué visitar todas las etapas. Denotamos por F_j el conjunto de etapas que el trabajo j, $j \in N$ tiene que visitar. Obviamente, $1 \leq |F_j| \leq m$. Además, dado que estamos tratando con un taller de flujo híbrido, solo se permite saltar etapas, no visitarlas en cualquier orden. Por ejemplo, no es posible que un trabajo visite las etapas $\{1, 2, 3\}$ y otro las etapas $\{3, 2, 1\}$. En cambio, sí sería posible que un trabajo visitase las etapas $\{1, 3\}$ o incluso una única etapa.

El tiempo de proceso del trabajo j en la etapa i se denota por p_{ij}. Por último, S_{ijk} indica el tiempo de cambio entre los trabajos j y k, $k \in N$ en la etapa i. Como se puede ver, el tiempo de cambio depende de la secuencia de producción El criterio de optimización es la minimización del tiempo máximo de finalización, o makespan. C_j denota el instante en el que el trabajo j se termina en el taller, que es básicamente el instante de finalización en la máquina a la que se

* Este trabajo está parcialmente subvencionado por el Ministerio de Ciencia e Innovación, bajo los proyectos "OACS - Optimización Avanzada de la Cadena de Suministro" y "SMPA - Secuenciación Multiobjetivo Paralela Avanzada: Avances Teóricos y Prácticos" con referencias IAP-020100-2008-11 y DPI2008-03511/DPI, respectivamente.

asigna el trabajo en la última etapa que éste visita. El makespan es entonces $C_{max} = \max_{j \in N}\{C_j\}$. Vignier et al. (1999) Extendieron la notación de tres campos habitual en *scheduling* para los problemas híbridos. Teniendo esto en cuenta, el problema SDST/HFFS que trabajamos en este artículo se denotaría como $HFFSm_r\left(\left(FM^{(i)}\right)_{i=1}^{m}\right)/F_j S_{ijk}/C_{max}$.

Los talleres de flujo híbridos sencillos (sin tiempos de cambio y donde todos los trabajos visitan todas las etapas) se han revisado, como se ha comentado, en el trabajo de Vignier, et al. (1999) pero también en Linn y Zhang (1999). Más recientemente, Kis y Pesch (2005) y Quadt y Kuhn (2007) han publicado reviews actualizados. Algunos trabajos recientes son el de Haouari y Hidri (2008) donde se proponen cotas inferiores o las metaheurísticas presentadas por Jin et al. (2006).

El problema estudiado en este trabajo se ha estudiado, entre otros, por Ruiz y Maroto (2006), Ruiz et al. (2008) y Zandieh et al. (2006). Incluso en Ruiz y Maroto (2006) y en Ruiz, et al. (2008) se estudian problemas que se pueden reducir al SDST/HFFS que se estudia en este trabajo, pero no el mismo problema exactamente.

2. La importancia de la representación. Heurística MDDR

Un aspecto clave a la hora de diseñar los algoritmos es la representación de la solución. En el problema tratado en este artículo no solo tenemos la secuenciación de trabajos en cada máquina, sino de manera adicional, la asignación de trabajos a máquinas en cada etapa. Una representación muy indirecta da malos resultados y una representación exacta (donde todas las posibles soluciones o secuencias del problema pueden ser representadas) es costosa computacionalmente. En este trabajo, combinamos ambas posibilidades. Primero, usamos una permutación inicial de trabajos en la primera etapa. Vamos lanzando los trabajos a la primera etapa según ese orden de permutación y los asignamos a la máquina que antes pueda completarlos (no a la primera máquina disponible, como es común en la literatura). Este aspecto es muy importante, dado que en la literatura se suele utilizar la regla FAM "*First Available Machine*". Con la presencia de los tiempos de cambio, la primera máquina disponible, puede necesitar de mucho tiempo de cambio. Se propone entonces la regla ECT ("*Earliest Completion Time*"). Concretamente, cuando hay que decidir a qué máquina asignamos un trabajo dentro de una etapa, se asigna a la máquina que hace menor el tiempo de finalización siguiente: $C_{ij} = \max\{C_{i,j-1}, C_{i-1,j}\} + S_{i,j-1,j} + p_{ij}$ donde $C_{i,j-1}$ es el instante de finalización del anterior trabajo en la secuencia que estaba asignado a la misma máquina que el trabajo j en la etapa i. De manera similar, $C_{i-1,j}$ es el tiempo de finalización del trabajo j en la etapa anteriormente visitada.

Para la segunda etapa y siguientes, los trabajos ya no se lanzan según el orden de la permutación, sino en el orden en el que éstos van completándose en las etapas anteriores. A partir de estas ideas, presentamos una heurística, llamada MDDR, que básicamente se trata de una asignación voraz de trabajos a máquinas. MDDR no separa las decisiones de secuenciación de trabajos y asignación de máquinas. Empezamos por la etapa 1, todos los trabajos que visitan esta etapa se consideran, uno por uno. Se calcula el mínimo ECT para todos los trabajos y todas las máquinas paralelas en la etapa 1. El trabajo con el menor ECT global se secuencia y se asigna a la máquina que le ha dado el menor ECT. El procedimiento se repite con todos los trabajos restantes hasta que todos están ya asignados. Para el resto de etapas el proceso es el mismo. La siguiente figura muestra la heurística MDDR.

Procedure *MDDR_heuristic*
Tiempos de disponibilidad de todos los trabajos son cero inicialmente
For i := 1 **to** m **do**
N = todos los n trabajos
Tiempos de disponibilidad de los trabajos en la etapa i = tiempos de finalización en la etapa $i-1$
While $|N| > 0$ **do**
Calcular el ECT tras asignar todos los trabajos pendientes en N a todas las máquinas en la etapa i
Asignar el trabajo j a la máquina l que proporciona el menor ECT
Extraer trabajo j de N
Actualizar el tiempo de disponibilidad de la máquina l en la etapa i
endwhile
endfor

Figura 1. Heurística MDDR

MDDR es un método muy rápido. En cada etapa se consideran como mucho n trabajos (asumiendo que todos los trabajos visitan todas las etapas, que sería el peor caso posible). Cuando se secuencia el trabajo con menor ECT, se pasa a considerar los $n-1$ trabajos restantes. Entonces tenemos $n \cdot (n+1)/2$ pasos. En cada paso se calculan los ECT para todas las máquinas paralelas en la etapa. Este proceso se repite para todas las etapas. El resultado es que MDDR tiene una complejidad computacional de $O(n^2 \cdot \sum_{i=1}^{m} m_i)$. Realmente, después de calcular los ECT en una etapa, para los trabajos pendientes, solo hay que recalcular los ECT de la máquina a la cual hemos asignado el trabajo, dado que los ECT en las otras máquinas no varían. Luego en la práctica el método MDDR es muy veloz.

3. Metaheurística basada en búsqueda local iterativa ILS

La búsqueda local Iterativa o ILS, es, como su nombre indica, una sencilla metaheurística que aplica búsqueda local una y otra vez. Según Hoos y Stützle (2005), ILS es una de las metaheurísticas más sencillas que se pueden desarrollar y que aún permite escapar de óptimos locales. Según Ruiz y Maroto (2005), el ILS de Stützle (1998) es uno de los mejores métodos para el taller de flujo estándar. Por todo lo anterior, parece razonable aplicar ILS al problema que nos ocupa.

ILS comienza con una solución obtenida heurísticamente, preferiblemente a partir de una solución de alta calidad. Normalmente se lleva a cabo una búsqueda local sobre esta solución inicial antes de empezar el método. A partir de aquí arranca el bucle principal de ILS. Tres procedimientos principales se aplican de manera iterativa hasta que se cumple el criterio de terminación del algoritmo. El primero se suele llamar perturbación y consiste en cambiar la solución óptimo local actual para escapar del óptimo. El segundo operador es una búsqueda local para conseguir, a partir de la solución perturbada, un nuevo óptimo local. La idea es que, después de la perturbación y búsqueda local, se haya encontrado un nuevo óptimo local con mejor calidad que el anterior. En este momento se aplica el tercer y último operador, el de aceptación. Obviamente, si la nueva solución es mejor que la mejor solución encontrada hasta el momento, se acepta. En caso contrario se suele rechazar o aceptar probabilísticamente como en los métodos de tipo *Simulated Annealing*. Para más detalles se puede consultar el trabajo de Lourenço et al. (2003), entre otros.

El algoritmo ILS que proponemos comienza con una única solución y realiza una búsqueda en el espacio de soluciones iterando sobre los dos operadores de perturbación y búsqueda local anteriormente comentados. Aplicamos una sencilla búsqueda local sobre una solución

candidata *x*. Se trata de una búsqueda local tipo "first improvement", es decir, paramos tan pronto *x* se mejora. Definimos ahora en más detalle la búsqueda local.

4. Búsqueda local

El trabajo en la primera posición de la secuencia π, denotado por π_1, se recoloca en otra posición de la secuencia aleatoriamente escogida. Si esta nueva secuencia π' mejora el makespan, la solución actual π se reemplaza por π' y la búsqueda local termina. En caso contrario, la búsqueda prosigue con π_2. La búsqueda itera como mucho para todos los trabajos sin repetición. La motivación de esta búsqueda local tan simple es conseguir un método que no sea lento, dado que la evaluación de la función objetivo para este problema es muy costosa. La siguiente figura muestra el pseudo algoritmo de la búsqueda local.

```
Procedure Local_search
    i = 1
    While i ≤ n do
        x' = secuencia obtenida al reinsertar x_i en una posición aleatoria diferente
        if C_max(x') < C_max(x) then
            x = x'
            break
        endif
        i = i + 1
    endwhile
```

Figura 2. Pseudo algoritmo de la búsqueda local

5. Perturbación y aceptación de nuevas soluciones

Cuando sea que la búsqueda local exceda un número *no_change* de iteraciones sin mejora, se lanza el procedimiento de perturbación. De acuerdo con Lourenço et al. (2003), la perturbación debe ser lo suficientemente fuerte como para escapar del anterior óptimo local pero también no demasiado fuerte como para no convertir el ILS en una búsqueda aleatoria. Se trata pues de alcanzar un delicado balance dentro del método.

Proponemos una perturbación en forma de una colección de movimientos de inserción. Se seleccionan aleatoriamente un número *d* de trabajos, se extraen de la solución actual y se reinsertan en posiciones aleatorias diferentes a las originales. Adicionalmente, para eliminar algo de aleatoriedad en este proceso, se generan *mu_move* soluciones perturbadas, se evalúan todas ellas y se acepta la mejor de todas. Con esto conseguimos un operador de perturbación sesgado hacia buenas soluciones evitando una perturbación totalmente aleatoria.

En la siguiente figura se muestra el algoritmo ILS completo.

```
Procedure  Iterated_Local_Search
        counter = 0
        Inicializar x
        x_best = x
        while no se cumpla criterio parada do
                x' = Local search(x)                    % búsqueda local
                if f(x') < f(x) do
                        counter = 0
                        x = x'
                        if f(x) < f(x_best) do
                                x_best = x'
                        endif
                else
                        counter = counter +1
                        if counter > no_change do
                                counter = 0
                                for r := 1 to nu_move do      % perturbation
                                        s(r) = Perturbación(x)
                                endfor
                                                        x = s_best
                        endif
                endif
        endwhile
```

Figura 3. Algoritmo ILS completo

6. Inicialización

La investigación en scheduling arroja muchos estudios donde se pone de manifiesto la importancia de inicializar los algoritmos de tipo metaheurístico con una buena solución. Hoy en día casi todas las metaheurísticas propuestas en la literatura se inicializan con el resultado de heurísticas de alta calidad. De acuerdo al estudio de Ruiz y Maroto (2006), una adaptación de la conocida heurística NEH de Nawaz, et al.(1983), denotada por NEHH, es la que mejores resultados proporciona para un problema similar al SDST/HFS presentado en este trabajo. Por tanto, usaremos la NEHH para inicializar el método ILS propuesto.

Para más detalles sobre el método ILS desarrollado, hemos colgado en Internet una versión completa de este trabajo en http://www.upv.es/deioac/Investigacion/Naderi_Ruiz_HFFS.pdf.

7. Resultados computacionales

En esta sección pasamos a comentar los resultados obtenidos por la heurística MDDR y el método ILS propuestos.

8. Banco de pruebas

Generamos un banco de pruebas con las siguientes combinaciones de n y m, donde $n = \{20, 50, 80, 120\}$ y $m = \{2, 4, 8\}$. Los tiempos de proceso se generan a partir de una distribución uniforme en el rango $[1,99]$. Generamos cuatro combinaciones de tiempos de cambio, uniformemente distribuidos en los intervalos $[1,25]$, $[1,50]$, $[1,99]$ y $[1,125]$. Se consideran instancias donde el número de máquinas paralelas en cada etapa es de 2 y grupos donde tenemos entre 1 y 4 máquinas paralelas por etapa. La probabilidad de saltar una etapa

para cada trabajo (S_p) es de 0.10 y de 0.40. Como resultado, tenemos 192 combinaciones de n, m, m_i, S_{ijk}, y S_p. Para cada combinación se generan 5 instancias diferentes. Por tanto, el banco de pruebas contiene 960 problemas y está disponible en http://soa.iti.es.

9. Algoritmos comparados y condiciones del experimento

Proponemos comparar el algoritmo ILS propuesto (ILS-P) y la heurística MDDR con los siguientes métodos de la literatura:

- Las heurísticas "SPTCH", "FTMIH" y "g/2, g/2 Johnson's rule" de Kurz y Askin (2003).
- El "Random keys genetic algorithm RKGA" propuesto por Kurz y Askin (2004).
- La heurística "NEHH" y el algoritmo genético "GA_R" de Ruiz y Maroto (2006). Nótese como estos métodos se propusieron para problemas algo diferentes y algunas adaptaciones han sido necesarias.
- El algoritmo inmune "IA_Z" de Zandieh et al. (2006).

Todos estos métodos proporcionaron buenos resultados en sus trabajos originales. Por ejemplo, el GA_R de Ruiz y Maroto (2006) se comparó contra otros cuatro algoritmos genéticos, contra un simulated annealing, un tabu search y dos algoritmos de colonias de hormigas y aún así resultó ser el mejor método con diferencia. Lo que queremos decir con esto es que los métodos ILS-P y MDDR propuestos se están comparando con el estado del arte.

Todos los algoritmos se han implementado en MATLAB 7. A pesar de no ser muy eficiente, MATLAB permite una rápida implementación y testeo de los métodos. Todos los experimentos se han ejecutado en un PC con un procesador Intel Core 2 Duo corriendo a 2.0 GHz y con 1 GB de RAM. No se ha hecho uso de programación multi-núcleo o multi-hilo. Solo se ha utilizado un único núcleo del procesador.

Como medida de calidad, empleamos la desviación porcentual relativa con respecto a una buena solución conocida (RPD):

$$RPD = \frac{Alg_{sol} - Min_{sol}}{Min_{sol}} \cdot 100 \qquad (1)$$

Donde Alg_{sol} es el C_{max} obtenido por un algoritmo en una determinada instancia. Obviamente, se buscan valores bajos de RPD. Min_{sol} es la mejor solución conocida para la instancia. Todas las mejores soluciones están disponibles en http://soa.iti.es.

El criterio de parada para todos los algoritmos (para aquéllos que lo tienen) se fija a $n^2 \times m \times 1.5$ milisegundos de tiempo de CPU. Fijar el tiempo máximo en función del tamaño de la instancia permite analizar los resultados estadísticamente.

Como se puede observar, todos los algoritmos se han reimplementado en el mismo lenguaje, se han probado en el mismo ordenador y se han ejecutado con el mismo conjunto de instancias, comparado con las mismas soluciones de referencia y ejecutado durante el mismo tiempo de CPU. Por tanto, los resultados son total y completamente comparables. La siguiente tabla muestras los RPD medios, agrupados por tamaño de instancia ($n \times m$).

Tabla 4: Resultados de los métodos MDDR e ILS-P comparados con la literatura existente

Instancia	SPTCH	FTIMH	John.	NEHH	MDDR	ILS-P	RKGA	IA_Z	GA_R
20×2	39,38	*41,82*	29,29	8,03	14,43	**1,39**	6,93	4,39	2,61
20×4	29,96	*37,57*	21,79	10,16	15,15	**1,27**	3,89	4,25	2,94
20×8	24,35	*29,95*	18,83	10,21	14,44	**1,49**	3,09	4,35	3,50
50×2	34,18	*38,71*	30,30	7,20	10,24	**1,26**	9,01	4,82	2,51
50×4	26,60	*38,60*	26,60	8,93	10,53	**1,85**	5,85	4,24	4,03
50×8	24,68	*29,06*	19,30	9,49	8,97	**1,75**	3,79	4,28	3,20
80×2	35,65	*42,22*	31,66	6,13	5,84	**2,06**	8,28	7,19	2,15
80×4	27,59	*36,15*	25,85	9,37	7,49	**2,92**	7,39	6,70	4,64
80×8	28,07	*32,02*	23,92	12,20	8,18	5,46	8,53	5,95	**4,51**
120×2	36,03	*40,12*	34,27	6,66	5,19	**3,38**	10,58	7,06	3,42
120×4	33,92	*40,24*	30,79	11,25	**5,93**	6,74	12,04	8,96	5,97
120×8	28,74	*35,27*	25,97	14,21	6,80	9,25	11,69	7,82	**5,19**
Media	30,76	*36,81*	26,55	9,49	9,43	**3,24**	7,59	5,83	3,72

Como se puede observar, SPTCH, FTIMH y John producen desviaciones muy elevadas, en el mejor de los casos de 18,83%, con lo que no se pueden considerar buenos métodos. NEHH y MDDR están muy a la par. De los métodos metaheurísticos, ni IA_Z ni RKGA producen resultados competitivos, dado que son ampliamente superados por GA_R y ILS-P. En el caso de estos dos últimos, parece que ILS-P domina a GA_R en las instancias pequeñas, no así en las grandes.

Los tiempos medios de CPU para todos los métodos metaheurísticos (ILS-P, RKGA, IA_Z y GA_R) son de 40,85 segundos, yendo desde 1,2 segundos en las instancias de 20×2 hasta 172,8 segundos para las instancias de mayor tamaño (120×8). Las heurísticas SPTCH, FTIMH y John son prácticamente inmediatas. En cambio NEHH necesita 2,91 segundos de media (desde 0,05 en 20×2 hasta 14,59 en 120×8). MDDR es muy rápida, con 0,23 segundos de media y 1,03 segundos solo para las instancias más grandes de 120×8.

Por último, se ha llevado a cabo un diseño de experimentos y un ANOVA sobre la variable respuesta RPD y usando como factor el tipo de algoritmo. El resultado se muestra en la siguiente figura:

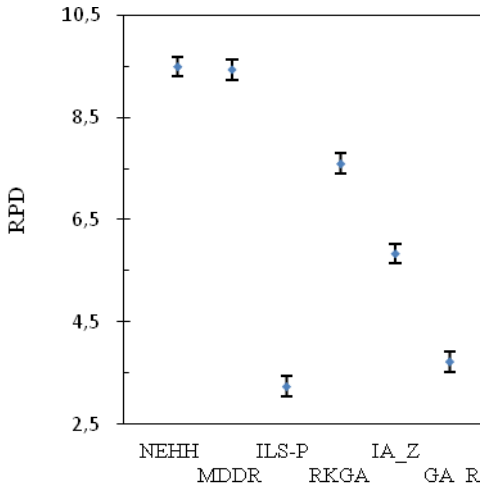

Figura 4. Medias e intervalos LSD para el resultado del ANOVA comparando los distintos algoritmos

Resulta interesante observar como de media MDDR es estadísticamente comparable a NEHH cuando resulta ser mucho más rápida. Como se puede observar, en promedio, ILS-P es estadísticamente mejor (aunque por poco) que GA_R. No obstante, y como se ha dicho antes, esto es dependiente del tamaño del problema. Si mostramos el gráfico de interacción entre *n* y el tipo de algoritmo tenemos el siguiente gráfico:

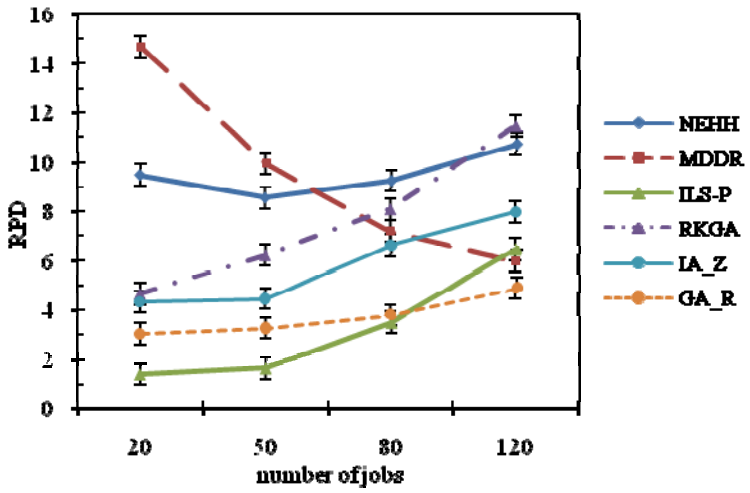

Figura 4. Medias e intervalos LSD para la interacción entre los algoritmos y *n*.

Se puede ver que ILS-P es peor que GA-R para *n*=120. Sorprendentemente, MDDR mejora radicalmente con el aumento del número de trabajos y para *n*=120 es competitivo con ILS-P.

10. Conclusiones

En este artículo se han propuesto dos algoritmos muy sencillos y muy competitivos para un problema de taller de flujo híbrido con tiempos de cambio de partida dependientes de la secuencia, máquinas paralelas idénticas en las etapas y saltado de etapas para los trabajos. El primer algoritmo propuesto es una rápida y eficiente regla de despacho dinámica, llamada MDDR, que ha demostrado ser muy competitiva, especialmente para problemas de gran tamaño. El segundo algoritmo propuesto está basado en la metodología del Iterated Local Search e incorpora una rápida búsqueda local y algunos otros elementos originales.

Los experimentos computacionales llevados a cabo indican que los dos métodos propuestos son muy competitivos a la par que sencillos y fáciles de implementar. Esperamos poder aplicar estos algoritmos a problemas con otros objetivos o a problemas de más utilidad en la práctica.

Referencias

Allahverdi. A.. Gupta. J. N. D. and Aldowaisan. T. (1999). A review of scheduling research involving setup considerations. Omega-International Journal of Management Science. 27 (2):219-239.

Allahverdi. A.. Ng. C. T.. Cheng. T. C. E. y Kovalyov. M. Y. (2008). A survey of scheduling problems with setup times or costs. European Journal of Operational Research. 187 (3):985-1032.

Allahverdi. A. y Soroush. H. M. (2008). The significance of reducing setup times/setup costs. European Journal of Operational Research. 187 (3):978-984.

Haouari. M. and Hidri. L. (2008). On the hybrid flowshop scheduling problem. International Journal of Production Economics. 113 (1):495-497.

Hoos. H. H. and Stützle. T. (2005). Stochastic local search: foundations and applications. Morgan Kaufmann Publishers. San Francisco. CA.

Jin. Z. H.. Yang. Z. and Ito. T. (2006). Metaheuristic algorithms for the multistage hybrid flowshop scheduling problem. International Journal of Production Economics. 100 (2):322-334.

Kis. T. and Pesch. E. (2005). A review of exact solution methods for the non-preemptive multiprocessor flowshop problem. European Journal of Operational Research. 164 (3):592-608.

Kurz, M. E. and Askin, R. G. (2003). Comparing scheduling rules for flexible flow lines, International Journal of Production Economics, 85 (3):371-388.

Kurz, M. E. and Askin, R. G. (2004). Scheduling flexible flow lines with sequence-dependent setup times, European Journal of Operational Research, 159 (1):66-82.

Linn. R. and Zhang. W. (1999). Hybrid flow shop scheduling: a survey. Computers & Industrial Engineering. 37 (1-2):57-61.

Lourenço. H. R.. Martin. O. C. and Stützle. T. (2003). Iterated Local Search. En Handbook of metaheuristics. F. Glover and G. A. Kochenberger. eds.. Kluwer Academic Publishers. Boston. pp. 321-353.

Nawaz. M.. Enscore. Jr. E. E. and Ham. I. (1983). A Heuristic algorithm for the *m*-machine. *n*-job flowshop sequencing problem. Omega-International Journal of Management Science. 11 (1):91-95.

Quadt. D. and Kuhn. H. (2007). A taxonomy of flexible flow line scheduling procedures. European Journal of Operational Research. 178 (3):686-698.

Ruiz. R. and Maroto. C. (2005). A comprehensive review and evaluation of permutation flowshop heuristics. European Journal of Operational Research. 165 (2):479-494.

Ruiz. R. y Maroto. C. (2006). A genetic algorithm for hybrid flowshops with sequence dependent setup times and machine eligibility. European Journal of Operational Research. 169 (3):781-800.

Ruiz. R.. Sivrikaya Serifoglu. F. y Urlings. T. (2008). Modeling realistic hybrid flexible flowshop scheduling problems. Computers & Operations Research. 35 (4):1151-1175.

Stützle. T. (1998). Applying iterated local search to the permutation flow shop problem. AIDA-98-04. FG Intellektik.TU Darmstadt.

Vignier. A.. Billaut. J.-C. and Proust. C. (1999). Les problèmes d'ordonnancement de type flow-shop hybride: État de l'art. RAIRO Recherche Operationnelle. 33 (2):117-183.

Zandieh. M.. Ghomi. S. M. T. F. y Husseini. S. M. M. (2006). An immune algorithm approach to hybrid flow shops scheduling with sequence-dependent setup times. Applied Mathematics and Computation. 180 (1):111-127.