Combinando Randomización Sesgada y Búsqueda Local Iterativa para Resolver Problemas de Flow-Shop

Juan, A.A., **Lourenço, H.R.,** Mateo, M., Grasas, A. and Agustín, A. (2012), Combinando Randomización Sesgada y Búsqueda Local Iterativa para Resolver Problemas de Flow-Shop. In Proceeding of the VIII Congreso Español sobre Metaheurísticas, Algoritmos Evolutivos y Bioinspirados, MAEB'2012 F. Rodriguez, B. Mélian, J.A. Moreno, J.M. Moreno (Eds.) Albacete, Spain, February 8-10, pp. 779-784.

ISBN 978-84-615-6931-1.

Combinando Randomización Sesgada y Búsqueda Local Iterativa para Resolver Problemas de Flow-Shop

Angel A. Juan[a], Helena R. Lourenço[b], Manuel Mateo[c], Alex Grasas[b], Alba Agustín[d]

Resumen— **A la hora de poder aplicar algoritmos teóricos a casos reales, no solo resulta conveniente que el algoritmo sea eficiente sino también que sea lo más comprensible posible y que no requiera de complejos procesos de parametrización. Siguiendo esta lógica, proponemos aquí un algoritmo híbrido que reune las características anteriores para resolver el problema del Flow-Shop (FSP). El algoritmo, que no require de parametrización alguna, combina estrategias de randomización con una Búsqueda Local Iterativa (ILS), logrando ser competitivo con otros conocidos algoritmos que se encuentran entre los más simples y eficientes para el FSP. Nuestro enfoque define (1) un nuevo operador para el proceso de perturbación ILS, (2) un nuevo criterio de aceptación basado en reglas simples y transparentes, y (3) un proceso de randomización sesgada de la solución inicial. Los resultados preliminares obtenidos con las instancias de Taillard permiten concluir que la solución propuesta puede ser una excelente alternativa en aplicaciones reales.**

Palabras clave— **Problema del Flow-Shop, Metaheurísticas, Algoritmos Randomizados, Iterated Local Search, GRASP**

I. Introducción y Bibliografía

El Problema del Flow-Shop (FSP) es un conocido problema de secuenciación consistente en: un conjunto J de k trabajos independientes tiene que ser procesado en un conjunto M de m máquinas independientes. Cada trabajo $j \in J$ requiere un tiempo de procesado determinado $p_{ij} \geq 0$ en cada máquina $i \in M$. Cada máquina puede realizar como máximo un solo trabajo a la vez, y se considera que todos los trabajos son procesados por las máquinas en el mismo orden. El objetivo más habitual es encontrar una secuencia para procesar los trabajos que minimice el instante máximo de finalización (o *makespan*), C_{max}. La Figura I ilustra este problema para el simple caso con $k = 3$ trabajos y $m = 3$ máquinas.

El FSP descrito se denota como $Fm|prmu|C_{max}$ y es un problema combinatorio con $k!$ posibles secuencias que, en general, es NP-completo [13]. Al igual que ha ocurrido con otros problemas combinatorios, numerosos enfoques distintos han sido desarrollados para solucionar el FSP. Estos enfoques varían desde los métodos exactos de optimización para resolver problemas de tamaño reducido, como la programación entera mixta o los algoritmos *branch-and-bound*, hasta el uso de heurísticas o metaheurísticas que proporcionan soluciones casi óptimas para problemas de medio o gran tamaño [15]. Nawaz et al. [10] introdujeron la heurística NEH, considerada la heurística simple con mejor rendimiento para el FSP. Esta heurística calcula el tiempo total de proceso de cada trabajo y crea una 'lista eficiente' de trabajos ordenándolos de forma decreciente. A cada paso, el primer trabajo de la lista es elegido para construir la solución, es decir, la regla de 'sentido común' es seleccionar primero aquellos trabajos con el tiempo total de procesado más alto. Una vez seleccionado, el trabajo es insertado en el conjunto ordenado de trabajos que configuran la solución actual. La posición exacta que el trabajo seleccionado ocupará viene determinada por el criterio de mínimo *makespan* siguiendo un movimiento de 'desplazamiento a la izquierda'. Esta heurística es el método aceptado en el FSP para obtener el límite superior para los mejores algoritmos *branch-and-bound* [7], [1]. Entre las distintas metaheurísticas utilizadas para resolver este tipo de problemas destacan *Simulated Annealing* [11], *Tabu Search* [18], [12], [9], *Genetic Algorithms*, y la Búsqueda Local Iterativa o *Iterated Local Search* (ILS). ILS es una poderosa metaheurística que proporciona un método fácil para mejorar el rendimiento de los algoritmos de búsqueda local, y ha sido ya aplicada con éxito al FSP [17], [16].

Muchos de estos métodos han alcanzado excelentes niveles de eficiencia, pero siguen utilizando varios parámetros que generalmente requieren procesos de ajuste no triviales y costosos en tiempo. La selección adecuada de estos parámetros y sus valores específicos tienen un impacto significativo en el rendimiento del algoritmo, es decir, su eficiencia tiende a ser bastante sensible a los valores asignados y, por tanto, son necesarias técnicas estadísticas como el diseño de experimentos para obtener algoritmos eficientes. Por otra parte, incluso cuando el proceso de configuración de los parámetros resulte exitoso, es frecuente encontrar 'números mágicos' dentro del algoritmo final. Dichos números son

[a]Estudios Informática, IN3-UOC, Rambla Poblenou, 156, 08018 Barcelona, España. E-mail: ajuanp@gmail.com.
[b]Dept. Economia i Empresa, Universitat Pompeu Fabra, R. Trias Fargas, 25-27, 08005 Barcelona, España. E-mails: helena.ramalhinho@upf.edu, alex.grasas@upf.edu.
[c]Dept. de Organización de Empresas, Universidad Politécnica de Cataluña, Av. Diagonal, 647, 08028 Barcelona, España. E-mail: manel.mateo@upc.edu.
[d]Dept. de Estadística e Investigación Operativa, Universidad Pública de Navarra, Campus Arrosadía, 647, 31006 Pamplona, España. E-mail: albamaria.agustin@unavarra.es.

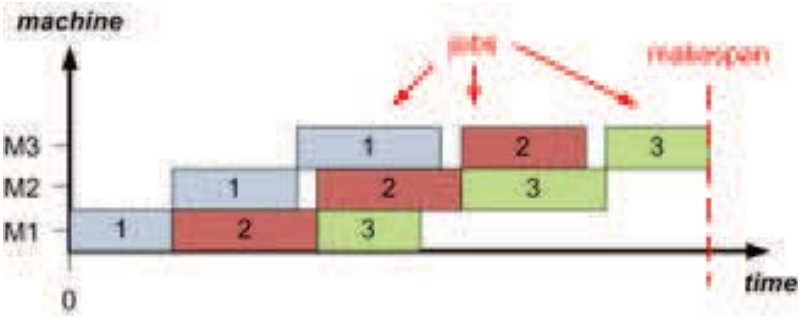

Fig. 1. Problema del Flow-Shop.

obtenidos mediante experimentación y corresponden a valores específicos de los parámetros que proporcionan el mejor rendimiento posible del algoritmo. Sin embargo, para el gestor que debe tomar decisiones operativas, y que no suele ser experto en metaheurísticas, le puede resultar difícil entender el significado real de estos valores, con lo que el algoritmo se convierte en una caja negra. Esta falta de transparencia reduce la credibilidad del método y, en consecuencia, hace el algoritmo menos interesante para aplicaciones reales [14].

A diferencia de los métodos expuestos, la principal contribución de este artículo es el diseño de un algoritmo eficiente, simple, y que no requiere de procesos de parametrización, para resolver el FSP. Como veremos con más detalle, nuestro algoritmo está basado en la metaheurística ILS, y no requiere de ningún proceso de parametrización ya que emplea reglas básicas de 'sentido común' para las etapas de búsqueda local, perturbación, y criterio de aceptación. En particular, se introduce en el proceso de perturbación un nuevo operador que combina un *swap* o intercambio con un movimiento clásico de 'desplazamiento a la izquierda', evitando así emplear 'números mágicos' y operadores complejos. Además, en lugar de usar un criterio de aceptación de tipo *Simulated Annealing* (SA), el cual funciona muy bien pero resulta poco transparente, se utiliza una regla nueva y más clara para esta etapa. Otra característica importante de nuestro enfoque es el proceso de randomización sesgada de la solución inicial, que emplea una distribución triangular sin parámetros para generar aleatoriamente distintas soluciones iniciales alternativas, todas ellas basadas en la heurística NEH. Así, se logra diversificar, de forma simple, la diversificación del punto inicial de la búsqueda local. Esta etapa de diversificación pretende evitar los puntos iniciales mal diseñados, y puede proporcionar mejoras cuando se aplican técnicas de computación paralela y distribuida.

II. El Algoritmo Randomizado

El algoritmo randomizado que proponemos en este artículo tiene algunas similitudes con las metaheurísticas Greedy Randomized Adaptive Search Procedure (GRASP) e ILS. El lector puede referirse a [3], [4] y [8], respectivamente, para revisiones recientes de estas dos metodologías. GRASP consiste básicamente en dos fases: (a) una primera fase constructiva que proporciona una buena solución inicial, y (b) una segunda fase que consiste en un procedimiento randomizado de búsqueda local para mejorar la solución inicial. Las dos fases son repetidas hasta que un cierto criterio de finalización es alcanzado, momento en el cual se proporciona la mejor solución obtenida en la búsqueda. Por su lado, ILS se basa en enfocar la búsqueda no en el espacio entero de soluciones, sino en un subespacio definido por las soluciones localmente óptimas. Para aplicar un algoritmo ILS a un problema de optimización hay que especificar con detenimiento las siguientes fases y criterios: generación de la solución inicial, búsqueda local, perturbación, y criterio de aceptación. En este artículo proponemos un método que combina ILS con técnicas de randomización similares (aunque distintas) a las que se usan en GRASP. La principal motivación para desarrollar un método que combina ILS con randomización sesgada para el FSP es proporcionar al algoritmo las siguientes propiedades deseables: *precisión*, *velocidad*, *simplicidad*, y *flexibilidad* [2].

El algoritmo randomizado que proponemos utiliza la metaheurística ILS como marco general (ver Procedimiento 1). Por tanto, integra los cuatro procesos descritos anteriormente. Vamos a explicar cada uno de ellos haciendo énfasis en los aspectos que diferencian nuestro algoritmo de anteriores algoritmos basados en ILS: el operador de perturbación, el criterio de aceptación y el proceso de randomización. El primer componente diferenciador está relacionado con la etapa de perturbación. Durante el proceso de perturbación se utiliza el operador '*intercambio mejorado*'. Éste es un operador muy sencillo, rápido y eficiente que básicamente hace lo siguiente: (a) selecciona al azar (usando una distribución uniforme) dos trabajos distintos de la solución actual, (b) intercambia ambos trabajos (es decir, intercambia sus posiciones en la permutación), y (c) aplica el clásico movimiento de 'desplazamiento a la izquierda' (originariamente propuesto en la heurística NEH) para ca-

da uno de los trabajos siguiendo un orden de izquierda a derecha. Este 'desplazamiento a la izquierda' se puede hacer más eficiente mediante las llamadas aceleraciones de Taillard [18].

El segundo componente diferenciador de nuestro algoritmo está en el criterio de aceptación. El algoritmo no utiliza un proceso tipo SA como la mayoría de algoritmos basados en ILS, sino una versión simplificada de un proceso tipo *Demon*. Este criterio está diseñado para contribuir, junto con el proceso de perturbación, a evitar los mínimos locales durante la ejecución del algoritmo. Para ello, el criterio establece simplemente los siguientes principios básicos: (a) cuando una solución recién generada mejora la solución base actual, la solución base es actualizada (mejorada) a esta nueva solución; asimismo, esta nueva solución es también comparada con la mejor solución encontrada para ver si ésta última debe ser también actualizada; y (b), aún cuando una solución recién generada sea peor que la solución base, la solución base será actualizada (deteriorada) a esta nueva solución siempre y cuando no haya más de un deterioro consecutivo y la degradación no exceda la última mejora. Nótese que al permitir que la solución base sea degradada hasta un cierto nivel, se reducen las probabilidades de que el algoritmo quede atrapado en un mínimo local. Una vez más, esto se logra con un conjunto muy simple de reglas básicas sin necesidad de ningún parámetro específico.

El tercer componente diferenciador en nuestro enfoque, y probablemente el más innovador, está relacionado con la solución inicial utilizada dentro del proceso ILS. Normalmente, esta solución inicial es la proporcionada por la heurística NEH, la cual produce una solución inicial relativamente buena en la mayoría de los casos. Usar la solución NEH en lugar de una solución generada aleatoriamente, suele servir para acelerar la convergencia del algoritmo. Sin embargo, parece razonable pensar que cuando se ejecutan múltiples series de la misma instancia, ya sea en secuencial o en paralelo, utilizar siempre el mismo punto de partida podría afectar a la convergencia en los casos en que la solución NEH sea relativamente 'pobre'. En este contexto, el término 'pobre' no necesariamente se refiere al valor de *makespan* de la solución, sino básicamente al número de movimientos o transformaciones que deben aplicarse a la solución inicial hasta llegar a una solución pseudo-óptima. Como estamos especialmente interesados en correr múltiples iteraciones para una instancia dada, hemos diseñado una forma de generar diferentes soluciones NEH randomizadas con propiedades similares. Para ello, usamos la distribución de probabilidad triangular decreciente, que es sesgada y sin parámetros, de manera similar a cómo Juan et al. [5] utilizan la distribución geométrica para obtener soluciones iniciales randomizadas para el Problema de Enrutamiento de Vehículos. La heurística NEH es un algoritmo iterativo que emplea una lista de trabajos ordenados por tiempo de terminación en todas las máquinas con el fin de construir una solución para el FSP. A cada paso de este proceso iterativo, la NEH selecciona el trabajo que encabeza la lista (aquel que presenta un mayor tiempo de procesado total) y lo añade a la solución parcial en aquella posición en que resulta un menor *makespan*. Como resultado, la NEH ofrece una solución determinista de 'sentido común', tratando de secuenciar los trabajos más exigentes en primer lugar. Nuestro método, en cambio, asigna una probabilidad de selección a cada trabajo en la lista. De acuerdo con nuestro diseño, esta probabilidad debe ser coherente con el tiempo total que cada trabajo necesita para ser procesado por todas las máquinas, es decir, los trabajos con un tiempo mayor tendrán más opciones de ser seleccionados de la lista que aquellos con menores tiempos. Finalmente, el proceso de selección debe hacerse sin introducir ningún parámetro. De lo contrario sería necesario realizar procesos de ajuste que suelen ser no triviales y requieren mucho tiempo. Para satisfacer todos estos requisitos, usamos una versión discreta de una distribución triangular durante el proceso de construcción de la solución: cada vez que un nuevo trabajo tiene que ser seleccionado de la lista, se utiliza una distribución triangular que asigna probabilidades que decrecen linealmente para cada trabajo elegible de acuerdo con sus correspondientes tiempos totales de proceso. De esta manera, los trabajos con mayor tiempo de proceso tienen más posibilidades de ser seleccionados de la lista primero, pero las probabilidades asignadas son variables y dependen de la distribución concreta seleccionada a cada paso. Iterando este procedimiento, se empieza un proceso de búsqueda aleatoria sesgada. Como consecuencia de ello, en la mayoría de los casos es posible obtener en tan solo unas pocas iteraciones (milisegundos para la mayoría de las instancias probadas) una solución randomizada cuyo *makespan* es casi igual o incluso mejor que la solución NEH original.

El proceso de *búsqueda local* que nuestro algoritmo utiliza es el mismo proceso simple e intuitivo usado en [16]. Por lo tanto, referimos al lector a dicho artículo para más detalles. Finalmente, y de acuerdo con los experimentos computacionales del apartado siguiente, un generador de números pseudo-aleatorios (RNG) rápido y de 'alta calidad' puede mejorar ligeramente el rendimiento de algunos algoritmos ILS. En particular, hemos observado que el uso de un 'buen' RNG [6], en lugar del RNG proporcionado originalmente por el propio lenguaje de programación, parece tener un impacto positivo en los resultados generales de los algoritmos con un criterio de aceptación tipo SA, donde se requiere un uso intensivo de números aleatorios uniformes en el intervalo (0, 1).

procedimiento 1 Algoritmo Randomizado

```
1:  sol_base ← randomizar[NEH] {DIVERSIFICACIÓN (randomización sesgada NEH)}
2:  sol_base ← busquedalocal[sol_base] {BÚSQUEDA LOCAL CLÁSICA}
3:  sol_mejor ← sol_base
4:  // BÚSQUEDA LOCAL ITERATIVA
5:  while criterio de finalización no se cumpla do
6:    sol_actual ← intercambio[sol_base] {PERTURBACIÓN}
7:    sol_actual ← busquedalocal[sol_actual] {BÚSQUEDA LOCAL CLÁSICA}
8:    delta ← coste[sol_actual] − coste[sol_base] {CRITERIO DE ACEPTACIÓN}
9:    // CASO A: MEJORA
10:   if delta < 0 then
11:     credito ← −delta
12:     sol_base ← sol_actual
13:     if coste[sol_base] < coste[sol_mejor] then
14:       sol_mejor ← sol_base
15:     end if
16:   end if
17:   // CASO B: DETERIORO
18:   if 0 < delta <= credito then
19:     credito ← 0
20:     sol_base ← sol_actual
21:   end if
22: end while
23: return sol_mejor
```

III. Estudio Computacional

En esta sección presentamos resultados preliminares que permiten comparar el rendimiento de nuestro algoritmo randomizado con otros algoritmos tipo ILS. El algoritmo randomizado fue implementado en Java. Para llevar a cabo los tests computacionales, ejecutados directamente en la plataforma Netbeans IDE para Java en Windows 7, se utilizó un Intel Xeon de 2.0 GHz y 4 GB RAM. Nuestro algoritmo randomizado fue comparado, en términos de la mejor solución encontrada tras 10 ejecuciones, con los siguientes algoritmos parametrizados (con parámetros D y T), también codificados en Java por los mismos programadores:

- El algoritmo ILS98-T04, propuesto en Stützle (1998), [17], usando $T = 0{,}4$.
- Los algoritmos IG-D4T04 y IG-D2T03, que representan dos parametrizaciones distintas ($D = 4$, $T = 0{,}4$ y $D = 2$, $T = 0{,}3$) del algoritmo propuesto en Ruiz y Stützle (2007), [16].
- El algoritmo RandIG-D4T04, que es nuestra propuesta para la versión randomizada del algoritmo IG-D4T04. Esta versión también incluye el uso del generador de números pseudo-aleatorios de alta calidad [6].

Para cada uno de los algoritmos, hemos diseñado y realizado tests con la misma máquina, mismo lenguaje de programación, mismo tiempo de ejecución, y mismo programador con las 120 instancias de Taillard [19]. Estas instancias, disponibles en http://mistic.heigvd.ch/taillard/default.htm, se agrupan en 12 conjuntos de 10 instancias cada una, combinando distinto número de trabajos y máquinas, esto es: grupo 20_5, grupo 20_10, grupo 20_20, grupo 50_5, grupo 50_10, grupo 50_20, grupo 100_5, grupo 100_10, grupo 100_20, grupo 200_10, grupo 200_20, y grupo 500_20. Así, para cada uno de los algoritmos considerados y cada instancia probada, se llevaron a cabo 10 iteraciones independientes (réplicas). Cada réplica fue ejecutada durante un tiempo máximo $t_{max} = 0{,}10\text{s} \times k \times m$, donde k es el número de trabajos, y m es el número de máquinas. Luego, para cada conjunto de réplicas se registró la mejor solución experimental encontrada (MEJOR-10). Además, también se obtuvo la mejor solución conocida (BKS) para cada instancia de la página web antes mencionada o de Zobolas et al. (2009), [20]. Hemos usado la métrica MEJOR-10 porque está fuertemente relacionada con el uso de capacidades multi-proceso de los actuales (y futuros) ordenadores. En efecto, distintas instancias de algoritmos randomizados como el que usamos pueden ser ejecutadas en paralelo sin incrementar el tiempo de reloj utilizado.

La Tabla I muestra, para cada algoritmo e instancia analizados, un resumen de los resultados experimentales cuando se considera el intervalo entre BKS y MEJOR-10 (la mejor solución encontrada en las 10 repeticiones). Si nos fijamos en los promedios de la tabla, podemos observar que todos los algoritmos ILS probados tienen un rendimiento bastante bueno

TABLA I
COMPARACIÓN ENTRE DISTINTOS ALGORITMOS ILS USANDO LA MÉTRICA MEJOR-10.

Instancias Taillard	Algoritmo Randomizado	Algoritmo ILS98-T04	Algoritmo IG-D4T04	Algoritmo IG-D2T03	Algoritmo RandIG-D4T04
Grupo 20_5	0,00 %	0,04 %	0,04 %	0,00 %	0,00 %
Grupo 20_10	0,00 %	0,00 %	0,00 %	0,04 %	0,00 %
Grupo 20_20	0,00 %	0,00 %	0,01 %	0,03 %	0,00 %
Grupo 50_5	0,00 %	0,00 %	0,00 %	0,00 %	0,00 %
Grupo 50_10	0,47 %	0,45 %	0,48 %	0,53 %	0,38 %
Grupo 50_20	0,71 %	0,74 %	0,66 %	1,00 %	0,62 %
Grupo 100_5	0,00 %	0,01 %	0,01 %	0,00 %	0,00 %
Grupo 100_10	0,10 %	0,07 %	0,07 %	0,10 %	0,10 %
Grupo 100_20	1,13 %	1,07 %	1,04 %	1,25 %	0,94 %
Grupo 200_10	0,09 %	0,11 %	0,08 %	0,14 %	0,08 %
Grupo 200_20	1,24 %	1,32 %	1,29 %	1,48 %	1,18 %
Grupo 500_20	0,63 %	0,67 %	0,63 %	0,71 %	0,62 %
PROMEDIOS	0,36 %	0,37 %	0,36 %	0,44 %	0,33 %

en promedio (teniendo en cuenta el tiempo máximo que cada instancia ha sido ejecutada). Sin embargo, parece que nuestra versión randomizada del algoritmo IG con parámetros óptimos, el RandIG-D4T04, es la que muestra el mejor rendimiento (intervalo medio = 0,33 %). Ligeramente peor que la versión randomizada, tenemos al IG-D4T04 óptimamente parametrizado y a nuestro algoritmo randomizado sin parámetros, con un rendimiento equivalente (intervalo medio = 0,36 %). A continuación, tenemos la parametrización ILS98-T04 (intervalo medio = 0,37 %). Y por último, lejos de los demás, tenemos el IG-D2T03 parametrizado de forma no óptima (intervalo medio = 0,44 %). Obsérvese que este último resultado parece implicar que el algoritmo IG ofrece cierto grado de sensibilidad con respecto a sus parámetros, es decir, su rendimiento puede reducirse considerablemente cuando se asignan valores no óptimos a los parámetros.

Un fenómeno interesante que hemos observado durante este estudio es que el uso de un RNG de 'alta calidad' parece tener un efecto positivo sólo en aquellos algoritmos ILS que incorporan un criterio de aceptación tipo SA. Sin embargo, el uso de un RNG de 'alta calidad' no parece haber tenido un impacto perceptible en los algoritmos ILS que no usan un proceso tipo SA. Nuestra hipótesis es que el proceso tipo SA es más sensible al uso de un RNG de 'alta calidad' porque emplea variables aleatorias continuas. Por el contrario, el resto de procesos aleatorios están básicamente asociados a variables discretas (selección de posiciones de trabajos) y, por tanto, no parecen ser demasiado sensibles con respecto a la calidad del RNG.

IV. Conclusiones

En este artículo se ha desarrollado un algoritmo eficiente, simple, y que no requiere de procesos de parametrización para resolver el Problema del Flow-Shop (FSP). El algoritmo combina técnicas de randomización con un marco de búsqueda local iterativa, siendo capaz de competir con otras metaheurísticas de tipo ILS. Estas otras metaheurísticas suelen utilizar una solución inicial determinista y contienen varios parámetros. Para el desarrollo de nuestro enfoque, hemos diseñado un nuevo operador de perturbación, un nuevo criterio de aceptación tipo *Demon*, y un proceso de randomización sesgada para la heurística NEH. Otra contribución de este trabajo es la incorporación del mencionado proceso de randomización a un algoritmo de tipo ILS que contiene un criterio de aceptación tipo SA. De acuerdo con los test computacionales realizados, el rendimiento de dicho algoritmo puede ser mejorado simplemente añadiendo nuestro proceso de randomización combinado con el uso de un generador pseudo-aleatorio de 'alta calidad'. Con todo, tanto nuestra versión randomizada del algoritmo ILS parametrizado, como el algoritmo sin parámetros propuesto, parecen ser excelentes alternativas para resolver el FSP en casos reales.

Agradecimientos

Este trabajo ha sido parcialmente financiado por el Ministerio Español de Ciencia e Innovación (proyectos TRA2010-21644-C03, ECO2009-11307, y DPI2007-61371), por el Departament Català d'Universitats, Recerca i Societat de la Informació (proyecto 2009 CTP 00007) y por la CYTED-IN3-HAROSA (CYTED2010-511RT0419, http://dpcs.uoc.edu).

Referencias

[1] Companys, R. and M. Mateo, 2007. *Different behaviour of a double branch-and-bound algorithm on $Fm|prmu|C_{max}$ and $Fm|block|C_{max}$ problems.* Computers & Operations Research, 34, 938-953.

[2] Cordeau, J.-F., Gendreau, M., Laporte, G., Potvin, J.-Y.,

and F. Semet, 2002.*A guide to vehicle routing heuristics.* Journal of the Operational Research Society 53, 512-522.

[3] Festa, P., and M.G.C. Resende, 2009a. *An annotated bibliography of GRASP – Part I: algorithms.* Int Trans Opl Res 16, 1-24.

[4] Festa, P., and M.G.C. Resende, 2009b. *An annotated bibliography of GRASP– Part II: applications.* Int Trans Opl Res 16, 131-172.

[5] Juan, A., Faulin, J., Ruiz, R., Barrios, B., and S. Caballe., 2010. *The SR-GCWS hybrid algorithm for solving the capacitated vehicle routing problem.* Applied Soft Computing 10(1), 215-224.

[6] L'Ecuyer, P. 2001. *Software for uniform random number generation: Distinguishing the good and the bad.* In Proceedings of the 2001 Winter Simulation Conference, 95-105. Pistacaway, NJ: IEEE Press

[7] Ladhari, T. and M. Haouari, 2005. *A computational study of the permutation flow shop problem based on a tight lower bound.* Computers & Operations Research, 32, 1831-1847.

[8] Lourenço H.R., Martin O. and T. Stützle., 2010. *Iterated Local Search: Framework and Applications.* In Handbook of Metaheuristics, 2nd. Edition. Vol.146. M. Gendreau and J.Y. Potvin (eds.), Kluwer Academic Publishers, International Series in Operations Research & Management Science, pp. 363-397.

[9] Moccellin, J.V., 1995. *A new heuristic method for the permutation flow-shop scheduling problem.* Journal of the Operational Research Society, 46, 883-886.

[10] Nawaz, M., Enscore, E.E., and I. Ham, 1983. *A heuristic algorithm for the m-machine, n-job flowshop sequencing problem.* OMEGA 11, 91-95.

[11] Osman, L., and C. Potts, 1989. *Simulated annealing for permutation flow-shop scheduling.* OMEGA 17(6), 551-557.

[12] Reeves, C.R., 1993. *Improving the efficiency of tabu search for machine scheduling problems.* Journal of the Operational Research Society 44(4), 375-382.

[13] Rinnooy Kan, A.H.G., 1976. *Machine Scheduling Problems: Classification, Complexity and Computations.* Springer.

[14] Robinson, S., 1997. *Simulation model verification and validation: increasing the users' confidence.* In Proceedings of the 1997 Winter Simulation Conference, 53-59.

[15] Ruiz, R., and C. Maroto, 2005. *A comprehensive review and evaluation of permutation flowshop heuristics.* European Journal of Operational Research 165, 479-494.

[16] Ruiz, R., and T. Stützle, 2007. *A simple and effective iterated greedy algorithm for the permutation flow-shop scheduling problem.* European Journal of Operational Research 177, 2033-2049.

[17] Stützle, T., 1998. *Applying Iterated Local Search to the Permutation Flow Shop Problem.* Available at: http://iridia.ulb.ac.be/∼stuetzle/publications/AIDA-98-04.pdf.

[18] Taillard, E., 1990. *Some efficient heuristic methods for the flow shop sequencing problem.* European Journal of Operational Research 47, 65-74.

[19] Taillard, E., 1993. *Benchmarks for basic scheduling problems.* European Journal of Operations Research 64, 278-285.

[20] Zobolas, C., Tarantilis, C., and G. Ioannou, 2009. *Minimizing makespan in permutation flow shop scheduling problems using a hybrid metaheuristic algorithm.* Computers & Operations Research 36, 1249-1267.