Aplicación de un algoritmo ACO al problema de *flowshop flexible* con tiempos de preparación dependientes de la secuencia y minimización de la tardanza total

An ant colony algorithm for scheduling flexible flowshop *with sequence dependent setup times and total tardiness minimization*

Eduardo Salazar Hornig[1]* Deisy Torres Pérez[1]

Recibido 23 de julio de 2014, aceptado 3 de diciembre de 2015
Received: July 23, 2014 Accepted: December 3, 2015

RESUMEN

Este estudio considera el problema de *flowshop flexible* con tiempos de *setup* anticipatorios dependientes de la secuencia y minimización de la tardanza total. Se propone un algoritmo de optimización de colonia de hormigas ACS *(Ant Colony System),* hibridizado con una búsqueda en vecindad de intercambio de pares, evaluado en un conjunto de problemas de prueba generados en estudios anteriores. Los resultados se comparan con otros métodos de solución, presentando el algoritmo propuesto mejores resultados.

Palabras clave: *Flowshop flexible*, colonia de hormigas, búsqueda en vecindad, intercambio de pares.

ABSTRACT

This study considers the flexible flowshop scheduling problem with anticipatory sequence dependent setup times and minimization of total tardiness. An ant colony optimization ACS (ant colony system) algorithm is proposed, hybridized with a pairwise interchange neighborhood search, which is evaluated on a set of test problems generated in previous studies, and compared with other solving methods. The proposed algorithm shows better results than those obtained by any other methods.

Keywords: Flexible flowshop, ant colony optimization, neighborhood search, pairwise interchange.

INTRODUCCIÓN

En los últimos años la programación de la producción ha tomado especial atención por quienes administran los procesos productivos. Lo anterior se debe principalmente al alto interés que existe por el aumento de la productividad y al alto nivel competitivo observado entre las empresas de un mismo rubro, las que se basan en estudios realizados con fines académicos para dar solución a los problemas que surgen en la gestión de la producción. Idealmente, buenos métodos de programación de producción deben ser simples, claros, fáciles de comprender, fáciles de llevar a cabo, flexibles y realistas. De acuerdo con lo anterior, el objetivo de la programación de producción es optimizar la utilización de los recursos de forma que se cumpla con los objetivos de producción [1].

La cantidad de máquinas disponibles para realizar las operaciones de un trabajo y su disposición en el sistema productivo determinan el tipo de configuración productiva que mejor se ajusta a las necesidades de producción.

El *flowshop flexible* (FFS) también conocido como *flowshop* híbrido (HFS) o *flowshop* con múltiples procesadores (FSMP) [2], es un tipo de configuración que se encuentra en todo tipo de

[1] Departamento de Ingeniería Industrial. Universidad de Concepción. Casilla 160-C. Concepción, Chile. E-mail: esalazar@udec.cl; deisytorres@udec.cl
* Autor de correspondencia

ambientes productivos del mundo real como lo es la industria electrónica, del papel, textil, entre otros.

Este problema de programación ha sido catalogado como NP-hard [3], incluso en el caso más simple donde existen dos centros de trabajo y dos máquinas idénticas en paralelo en cada centro. En este tipo de problemas no se puede obtener una solución óptima en bajos tiempos computacionales, razón por la que han surgido métodos heurísticos y metaheurísticos que pueden encontrar buenas soluciones cercanas al óptimo con un bajo tiempo computacional.

La mayoría de los estudios basados en la programación de un *flowshop flexible* buscan minimizar el *makespan*, dejando de lado otras medidas de desempeño como lo es la tardanza total [4]. El alto nivel competitivo al interior del sector industrial obliga a las compañías a adoptar técnicas que permitan obtener una programación que se adecue a las fechas establecidas con los clientes, es decir, buscan minimizar la tardanza total de entrega de los pedidos y así poder aumentar el nivel de satisfacción de quienes requieren de los bienes fabricados por las empresas.

Los trabajos realizados por [5-7] se basan en optimizar el *makespan*, además de otros criterios como el número de trabajos que se entregan en forma tardía en problemas de programación de *flowshop flexible*. Sin embargo, en los estudios como [2, 8-10] se presentan métodos de solución a este problema que buscan optimizar las medidas de desempeño que se relacionan con la tardanza.

Gran parte de las investigaciones que existen se basan en la aplicación de diferentes enfoques para la resolución de estos problemas, entre estos se pueden mencionar métodos analíticos [6], heurísticos y metaheurísticos, como la aplicación de algoritmos genéticos [9-10], simulado recocido y búsqueda tabú, entre otros [8].

En este trabajo se desarrolla un algoritmo de optimización de colonia de hormigas ACS, hibridizado con una búsqueda en vecindad, aplicado al problema FFS con *setup* anticipatorios dependientes de la secuencia, con minimización de la tardanza total. Los resultados se comparan con las reglas de despacho EDD (*earliest due date*), *Slack* (holgura), un algoritmo genético básico (AGB) [9] y un algoritmo genético mejorado (AGB_EDD y AGB_Slack) [10], el que introduce poblaciones iniciales generadas como vecindades de las soluciones generadas con las reglas de despacho EDD y Slack, respectivamente.

FORMULACIÓN DEL PROBLEMA

El problema de programación de un *flowshop flexible* consiste en programar un conjunto de n trabajos en un sistema de m ($m \geq 2$) centros de trabajo (etapas de producción en serie) de una o más máquinas paralelas idénticas dispuestas en paralelo. Cada trabajo tiene asociado una fecha de entrega (*due date*), un tiempo de proceso y un tiempo de preparación o *setup* en cada etapa del proceso productivo.

Dentro de las consideraciones que presenta el problema FFS se puede mencionar lo siguiente: el flujo de los trabajos es unidireccional; cada trabajo debe ser asignado para ser procesado en una máquina a la vez en cada uno de los m centros de trabajo; los tiempos de *setup* son anticipatorios, es decir, la preparación de la máquina puede iniciarse cuando el trabajo aun no ha terminado su proceso de la etapa anterior; una vez que el trabajo comienza a ser procesado en una máquina, la operación no puede ser interrumpida (*non preemption*); todos los trabajos son liberados en el tiempo cero (tiempo de referencia para el inicio de la programación); cada máquina está continuamente disponible desde el tiempo cero, pero no puede procesar más de un trabajo a la vez; las máquinas operan sin fallas durante el horizonte de programación; las operaciones de los trabajos no pueden ser adelantadas; los datos del problema son deterministas y conocidos de antemano.

En este trabajo se evalúa la tardanza total T, es decir, se busca obtener una programación factible que minimiza esta medida de desempeño global, la que se define en las ecuaciones (1).

$$T = \sum_{i=1}^{n} T_1 \tag{1}$$

$$T_i = \max\{0, C_i - d_i\} \tag{2}$$

La tardanza del trabajo i, definida en la ecuación (2), es el tiempo de retraso con respecto al *due date* (d_i), con el que se entrega el trabajo i. Este valor es positivo si el trabajo es entregado posterior a su

fecha de entrega y es cero en caso contrario. C_i es el tiempo de finalización del trabajo *i*.

ALGORITMO ACS PARA FFS

La optimización basada en colonia de hormigas (algoritmos ACO), es un enfoque para resolver problemas de optimización combinatoria que se inspira en el comportamiento de las hormigas para encontrar el camino más corto entre la fuente de alimento y el hormiguero. Desde 1991 este tipo de algoritmos se han convertido en un importante campo de investigación, pues los autores ha ido desarrollando modelos más sofisticados para resolver una gran variedad de problemas de optimización combinatoria [11].

Este enfoque fue propuesto inicialmente por Dorigo en 1992 [12], como un método para resolver el clásico problema del vendedor viajero (*traveling salesman problem*).

Los algoritmos ACO funcionan en general de la siguiente forma: *h* hormigas (artificiales) de la colonia se mueven de forma independiente y concurrente, a través de estados adyacentes del problema. Se utiliza una regla de transición para realizar este movimiento, la que se basa en la información local disponible en los arcos del grafo que representa el problema (los nodos representan las ciudades y los arcos los caminos que unen las ciudades). Para guiar la búsqueda, esta información contiene tanto la información heurística (η_{ij}) y los rastros de feromona (τ_{ij}) asociados a cada arco (i, j). Las hormigas depositan feromona al transitar por un arco mientras construyen una solución (actualización en línea paso a paso de los rastros de feromona). Después que las *h* hormigas generan una solución, se deposita una cantidad de feromona en los arcos de la mejor solución conocida (actualización en línea *a posteriori* de los rastros de feromona), de esta forma se guía la búsqueda de soluciones en el siguiente ciclo [13].

La Figura 1 muestra una estructura básica de los algoritmos ACO.

El primer paso corresponde a la inicialización de los valores de los parámetros: el rastro inicial de feromona, τ_0 (valor positivo pequeño, por lo general el mismo para todos los arcos), el número

```
ACO( )
{
    Definición de parámetros
    Inicialización de rastros de feromona

    while (criterio de término no satisfecho)
  {
       Generar hormigas y sus soluciones;
       Actualizar rastros de feromona;
       Aplicar acciones opcionales;
  }
}
```

Figura 1. Estructura básica de ACO [14].

de hormigas *h* de la colonia, y los pesos que definen la proporción en que afectarán la información heurística y los rastros de feromona en la regla de transición.

El procedimiento principal controla:

- La construcción de soluciones.
- La evaporación de los rastros de feromona.
- Las acciones opcionales.

Existen diversos modelos de optimización basados en colonias de hormigas, entre los cuales podemos mencionar: AS (*ant system*), ACS (*ant colony system*) y otras variantes como MMAS (*max-min ant system*) [15] y BWAS (*best-worst ant system*) [16]. Para efecto de este estudio la explicación se extiende a los primeros modelos mencionados, para mayor detalle de las otras variantes revisar [15].

En este trabajo se utiliza el algoritmo ACS (uno de las primeras variantes desarrolladas a partir de AS [17-18]), que asume una secuencia de trabajos como la representación de una solución. El programa de producción se obtiene asignando los trabajos, en el orden entregado por la secuencia, en la máquina de la primera etapa en la que finaliza antes. En las etapas 2 a *m*, los trabajos son asignados en cuanto se libera una máquina (en caso de empate prevalece la prioridad dada por la secuencia de trabajos de la solución).

El algoritmo ACS construye, por lo tanto, secuencias de trabajos que son evaluadas mediante el procedimiento de asignación descrito en el párrafo anterior.

Las soluciones en ACS se construyen de forma tal que en cada paso de construcción, una hormiga k en el nodo i escoge ir al nodo j_0 en base a una regla de transición pseudo aleatoria definida por medio de las ecuaciones (3) y (4).

$$j_o = \left\{ \begin{array}{cc} \arg\left[\max_{j \in Fk} \left[\tau_{ij} \cdot \eta_{ij}^{\beta} \right] \right] & si\ \mathrm{q} \le \mathrm{q}_0 \\ J & si\ \mathrm{q} > \mathrm{q}_0 \end{array} \right\} \quad (3)$$

En la ecuación (3), q es una observación (pseudo) aleatoria U(0,1) y J una v.a. discreta con distribución de probabilidades:

$$p_{ij} = \frac{\tau_{ij} \cdot \eta_{ij}^{\beta}}{\sum_{l \in F_k} \tau_{il} \cdot \eta_{il}^{\beta}} \text{ para } j \in F_k \quad (4)$$

F_k es el conjunto de nodos no asignados y β es un número real (parámetro) positivo que define la importancia relativa entre los rastros de feromona y la información heurística.

Cuando $q \le q_0$ se elige la mejor opción con respecto de la información heurística y los rastros de feromona (intensificación), de lo contrario, se realiza una exploración controlada a través de una variable aleatoria que selecciona el siguiente nodo en base a una distribución de probabilidades proporcional a la conveniencia de selección de cada nodo.

La regla de actualización en línea paso a paso, considera tanto la evaporación de feromona como la deposición de la misma. Debido a que la cantidad de feromona es muy pequeña, al aplicar esta regla disminuyen la feromona entre las conexiones recorridas por las hormigas. De esta forma se vuelven menos atractivos los recorridos que realizan un gran número de hormigas en la iteración actual, así no todas las hormigas siguen el mismo recorrido.

Se produce una actualización local y global según, respectivamente, las ecuaciones (5) y (6). La actualización local (actualización en línea) se produce cada vez que una hormiga transita por el arco (i, j)).

$$\tau_{ij} \leftarrow (1-\rho) * \tau_{ij} + \rho \cdot \tau_0 \quad (5)$$

El factor $\rho \in$ ([0,1] es el factor de evaporación de feromona y τ_0 es el nivel inicial de feromona en cada arco. La actualización global (actualización fuera de línea) produce una actualización de la feromona en los arcos de la mejor solución encontrada hasta el momento.

$$\tau_{ij} \leftarrow (1-\gamma) * \tau_{ij} + \gamma \cdot f_b \quad (6)$$

En la ecuación (6) f_b es una función dependiente de la calidad de la mejor solución.

En este trabajo se proponen dos variantes para definir la información heurística η_{ij} para el ACS aplicado al FFS:

– Información heurística basada en el atraso (*lateness*). Siendo el trabajo i el último trabajo asignado a la secuencia de trabajos, para todo trabajo j no asignado la información heurística se define como:

$$\eta_{ij} = \frac{L_j - L_{min} + 1}{L_{max} - L_{min} + 1} \quad (7)$$

– τ_j es el atraso del trabajo j, si este es asignado a continuación del trabajo i en la secuencia de trabajos. L_{min} y L_{max} corresponden, respectivamente, al mínimo y máximo atraso entre los atrasos de todos los trabajos j no asignados, respectivamente.

– Información heurística basada en la tardanza (*tardiness*). Siendo el trabajo i el último trabajo asignado en la secuencia de trabajos, para todo trabajo j no asignado la información heurística se define como:

$$\eta_{ij} = \frac{T_j - T_{min} + 1}{T_{max} - T_{min} + 1} \quad (8)$$

T_j es la tardanza del trabajo j, si éste es asignado a continuación del trabajo i en la secuencia de trabajos. T_{min} y T_{max} corresponden, respectivamente, a la mínima y máxima tardanza entre las tardanzas de todos los trabajos no asignados.

La inclusión del valor 1, tanto en el numerador como en el denominador de las ecuaciones (7) y (8), evita que la información heurística se indetermine en el caso de que todos los trabajos tengan, respectivamente, igual atraso o tardanza. Además, también garantiza que la información heurística sea estrictamente mayor a cero en todos casos.

El rastro inicial de feromona se calcula como $\tau_0 = \frac{1}{T_0}$, con T_0 igual al valor de la menor tardanza obtenida en una muestra aleatoria de 10 soluciones.

Para mejorar el rendimiento, el algoritmo ACS, este se hibridizó con una búsqueda en vecindad IP, aplicada en la solución generada por cada hormiga en todos los ciclos.

EXPERIMENTACIÓN

Para la experimentación se utilizó el conjunto de 30 instancias de cinco etapas generadas por [9] con las siguientes características:

- Número de trabajos: n = 30 trabajos
- Número de etapas: m = 5 etapas
- Número de máquinas por etapa: m_k ~ UD[1, 5]
- Tiempo de proceso: p_{ik}~ UD(1,100)
- Tiempos de *setup*: s_{ijk} ~ UD(0,9).

El *due date* es generado por [9] mediante la siguiente adaptación:

$$d_i = \sum_{k=1}^{m} p_{ik} + \sum_{k=1}^{m} \bar{s}_{ik} + (n-1) * \bar{p}_i * u \qquad (9)$$

En la ecuación (9) $\bar{p}_i = \sum_{k=1}^{m} p_{ik} / m$, $\bar{s}_{ik} = \sum_{j=1}^{n} s_{jik} / n$ y u es una observación de una v.a. U ~ U(0, 1). El *due date* (d_i) considera el tiempo total de proceso más la estimación del tiempo total de *setup*, y una holgura que depende de la cantidad de trabajos a procesar.

El algoritmo ACS se utilizó con valores de parámetros: h = 5, β = 1, $\gamma = \beta$, q_0 = 0,5. Para la evaluación de ACS (ACS_BV) utilizando 1.000 (100) ciclos, y se realizaron 10 réplicas para cada instancia.

La comparación de los métodos se realizó mediante el *índice de desviación relativa* (IDR) [19] definido en la ecuación (10), pues se trata de un problema de minimización de tardanza y se debe considerar el caso en que ésta sea cero.

$$IDR = \frac{T_{método} - T_b}{T_w - T_b} * 100\% \qquad (10)$$

En la ecuación (10), T_b y T_w corresponden, respectivamente, al mejor y peor valor de la tardanza obtenido por algún método para una determinada instancia, y $T_{método}$ corresponde al valor de la tardanza obtenido por un método en particular. Así, a menor valor de IDR, mejor es la calidad de la solución.

Para la evaluación de los métodos se utilizaron rutinas adaptadas del software *SPS_Optimizer* [20], herramienta diseñada para la programación de operaciones. La ejecución del software se realizó en un computador Pentium (R) Dual Core CPU T4300 de 2.10 GHz con 4 GB de RAM.

RESULTADOS

La Figura 2 muestra el índice de desviación relativa al comparar todos los algoritmos. El algoritmo AG_EDD obtiene en promedio la menor desviación relativa con 0,9%, lo que significa que este método, en promedio, genera en todas las instancias soluciones muy cercanas a la mejor solución generada entre todos los métodos. Le sigue el método AG_SLACK con 9,8% y el método ACS propuesto en este trabajo con 16,1%. Las heurísticas EDD y Slack obtienen en promedio una desviación relativa de 32,4% y 37,9%. El método AGB obtiene una desviación de 84,8%.

En la Figura 3 se muestran los resultados de tardanza total para las 30 instancias utilizando los diferentes métodos comparados. Al comparar los métodos se puede observar que las heurísticas EDD, Slack y AGB presentan un comportamiento similar, siendo los métodos AG_EDD y AG_Slack los que presentan un mejor desempeño respecto de la tardanza total, seguidos del algoritmo ACS. El algoritmo ACS en algunas instancias obtiene la mejor solución y es la misma que obtienen los métodos que mejoran el algoritmo genético (AG_EDD y AG_Slack).

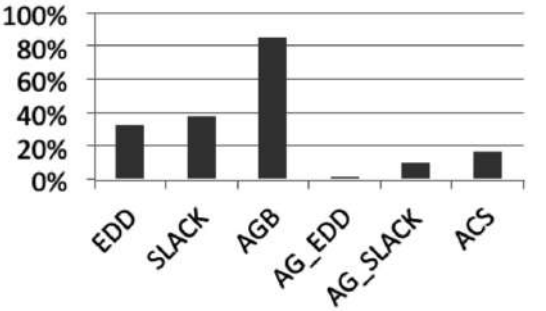

Figura 2. Índice de desviación relativa.

AG_EDD obtiene el mayor porcentaje de veces la mejor solución (83,3%) seguido de AG_SLACK y ACS que presentan el mismo porcentaje (40,0%), superando a las heurísticas EDD (23,3%) y Slack (33,3%). El AGB presenta el desempeño más bajo alcanzando un 13,3% de las veces la mejor solución. La Tabla 1 resume el índice de desviación relativa y el porcentaje de veces que cada algoritmo encuentra la mejor solución.

AG_EDD y AG_SLACK presentan un mejor rendimiento, obtienen los menores valores de la tardanza total, el mayor porcentaje de veces encuentran la mejor solución y muestran los valores más bajos del índice de desviación relativa. El algoritmo AGB, por otro lado, presenta el desempeño más bajo, ya que obtiene el menor porcentaje de veces la mejor solución y muestra un alto valor del índice de desviación relativa. Este método es seguido por el rendimiento que muestra EDD y Slack. El método ACS muestra un desempeño intermedio.

Con el objetivo de mejorar el rendimiento de los métodos, se aplicó un algoritmo de búsqueda en vecindad IP a cada uno de ellos.

Al hibridizar el método ACS con una búsqueda en vecindad IP, ACS_BV se observa que obtiene en promedio, la menor desviación relativa con un 0,0% (obtiene la mejor solución en el 100% de los casos), seguido de AG_EDD_BV con un 43,0% y AG_Slack con un 42,6%. Las heurísticas EDD_BV y Slack_BV obtienen una desviación relativa promedio de 21,0% y 31,2%, respectivamente. Nuevamente, el algoritmo AGB_BV presenta el desempeño más bajo con 81,5%. La Figura 4 muestra el índice de desviación relativa para todos los métodos con BV. La Tabla 2 resume el índice de desviación relativa y el porcentaje de veces que un método encuentra mejor solución. Respecto de este último, el algoritmo ACS_BV obtiene un porcentaje igual a 0,0%, los métodos EDD_BV, Slack_BV, AG_EDD_BV y AG_Slack_BV obtienen el 26,7%, 30,0%, 30,0% y 30,0% de las veces, respectivamente, seguidos por AGB_BV que encuentra la mejor solución en el 13,3% de las veces.

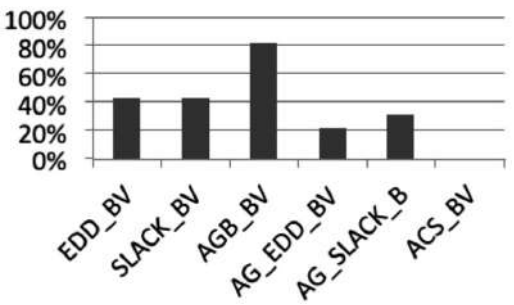

Figura 4. Índice de desviación relativa (con BV).

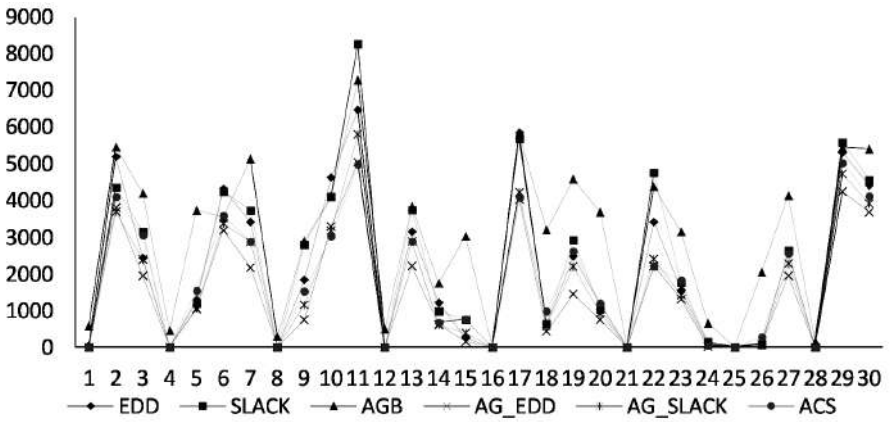

Figura 3. Comparación de métodos.

Tabla 1. Resumen comparación de algoritmos.

Método	EDD	SLACK	AGB	AG_EDD	AG_Slack	ACS
Índice de desviación relativa (%)	32,4	37,9	84,8	0,9	9,8	16,1
Mejor solución (%)	23,3	33,3	13,3	83,3	40,0	40,0

Tabla 2. Resumen comparación de algoritmos (con búsqueda en vecindad).

Método	EDD_BV	Slack_BV	AGB_BV	AG_EDD_BV	AG_Slack_BV	ACS_BV
Índice de desviación relativa (%)	43,0	42,6	81,5	21,4	31,2	0,0
Mejor solución (%)	26,7	30,0	13,3	30,0	30,0	100,0

ACS_BV es el que presenta el mejor rendimiento obteniendo bajos índices de desviación relativa y mayores porcentajes de veces que encuentran la mejor solución, AG_EDD_BV y AG_Slack_BV presentan un desempeño promedio y es el algoritmo AGB_BV el que obtiene un alto índice de desviación relativa y un bajo porcentaje de veces que encuentran la mejor solución. Las heurísticas EDD_BV, Slack_BV presentan un desempeño intermedio con una similar desviación relativa y porcentaje de veces que encuentran la mejor solución.

Al incorporar la búsqueda en vecindad IP al algoritmo ACS se observa que presenta mejores resultados que el resto de los métodos, pues en todas las instancias obtiene la mejor solución y además presenta el menor índice de desviación relativa, seguido de los métodos AG_EDD_BV y AG_Slack_BV.

Los tiempos CPU obtenidos con ACS presentan un orden de magnitud de 80 [s] por réplica. Al incorporar la búsqueda en vecindad IP a ACS los tiempos computacionales se incrementan al orden de 300 [s] por réplica.

CONCLUSIONES

En este trabajo se ha propuesto un algoritmo ACS para resolver el problema de programación de un *flowshop flexible* con máquinas paralelas idénticas en cada etapa, tiempos de preparación anticipatorios dependientes de la secuencia con el objetivo de minimizar la tardanza total.

El algortimo ACS se comparó con otros métodos heurísticos: las reglas de despacho EDD y Slack, un algoritmo genético básico y un algoritmo genético mejorado.

Al evaluar el desempeño del algoritmo ACS propuesto, este mostró un rendimiento más bajo que los algoritmos AG_EDD y AG_Slack; sin embargo, supera a las heurísticas EDD, Slack y al algoritmo AGB.

Al incorporar la búsqueda en vecindad IP en los métodos, los resultados muestran que el ACS_BV obtiene la mejor solución en todas las instancias evaluadas.

Se concluye que el algoritmo ACS propuesto no entrega siempre mejores resultados comparado con el algoritmo genético mejorado desarrollado en trabajos anteriores. Sin embargo, al hibridizar el algoritmo ACS con una búsqueda en vecindad, la mejora de ACS es significativa, superando el rendimiento de todos los métodos comparados, sin embargo; los tiempos computacionales se incrementan significativamente.

REFERENCIAS

[1] J.H. Heizer and B. Render. “Operations management”. Pearson/Prentice Hall. 2005.

[2] H-T. Lin and Ch-J. Liao. “A case study in a two-stage hybrid flow shop with setup time and dedicated machines”. International Journal of Production Economics. Vol. 86 Nº 2, pp. 133-143. 2003.

[3] R. Ruiz and J.A. Vázquez-Rodríguez. “The hybrid flow shop scheduling problem”. European Journal of Operational Research. Vol. 205 Nº 1, pp. 1-18. 2010.

[4] I. Ribas, R. Leisten and J.M. Framiñan. “Review and classification of hybrid flow shop scheduling problems from a production system and a solutions procedure perspective”. Computers & Operations Research. Vol. 37 Nº 8, pp. 1439-1454. 2010.

[5] D.-F. Shiau, S.-C. Cheng and Y.-M. Huang. “Proportionate flexible flow shop scheduling via a hybrid constructive genetic algorithm”. Expert Systems with Applications. Vol. 34 Nº 2, pp. 1133-1143. 2008.

[6] J. Jungwattanakit, M. Reodecha, P. Chaovalitwongse and F. Werner. “A comparison of scheduling algorithms for flexible flow shop problems with unrelated parallel machines, setup times, and dual

criteria". Computers & Operations Research. Vol. 36 Nº 2, pp. 358-378. 2009.

[7] R. Logendran, P. de Szoeke and F. Barnard. "Sequence-dependent group scheduling problems in flexible flow shops". International Journal of Production Economics. Vol. 102 Nº 1, pp. 66-86. 2006.

[8] B. Naderi, M. Zandieh, A. Khaleghi Ghoshe Balagh and V. Roshanaei. "An improved simulated annealing for hybrid flowshops with sequence-dependent setup and transportation times to minimize total completion time and total tardiness". Expert Systems with Applications. Vol. 36 Nº 6, pp. 9625-9633. 2009.

[9] E. Salazar y B. Figueroa. "Minimización de la tardanza para el flowshop flexible con setup utilizando heurísticas constructivas y un algoritmo genético". Ingeniare. Revista chilena de ingeniería. Vol. 20 Nº 1, pp. 89-98. 2012.

[10] E. Salazar y R. Sarzuri. "Algoritmo genético mejorado para la minimización de la tardanza total en un flowshop flexible con tiempos de preparación dependientes de la secuencia". Ingeniare. Revista chilena de ingeniería. Vol. 23 Nº 1, pp. 118-127. 2015.

[11] R.F. Tavares Neto and M. Godinho Filho. "Literature review regarding Ant Colony Optimization applied to scheduling problems: Guidelines for implementation and directions for future research". Engineering Applications of Artificial Intelligence. Vol. 26 Nº 1, pp. 150-161. 2013.

[12] M. Dorigo. "Optimization, Learning and Natural Algorithms (in Italian)". Ph.D. Thesis, Dipartimento de Elettronica, Politécnico de Milán, Italy. 1992.

[13] S. Alonso, O. Cordón, I. Fernández y F. Herrera. "La metaheurística de optimización basada en colonias de hormigas: modelos y nuevos enfoques". En Optimización inteligente: técnicas de inteligencia computacional para optimización, pp. 261-314. Universidad de Málaga. Málaga, España. 2004.

[14] M. Dorigo and G.D. Caro. "The ant colony optimization meta-heuristic". En New Ideas in Optimization, pp. 11-32. 1999.

[15] T. Stützle and H.H. Hoos. "MAX-MIN Ant System". Future Generation Computer Systems. Vol. 16 Nº 8, pp. 889-914. 2000.

[16] O. Cordón, I. Fernández de Viana and F. Herrera. "Analysis of the Best-Worst Ant System and Its Variants on the QAP". En Proceedings of the Third International Workshop on Ant Algorithms. London, UK, pp. 228-234. 2002.

[17] M. Dorigo, G. Di Caro and L.M. Gambardella. "Ant Algorithms for Discrete Optimization". Artificial Life. Vol. 5 Nº 2, pp. 137-172. 1999.

[18] M. Dorigo and L. M. Gambardella. "Ant colony system: a cooperative learning approach to the traveling salesman problem". IEEE Transactions on Evolutionary Computation. Vol. 1 Nº 1, pp. 53-66. 1997.

[19] B. Naderi, M. Zandieh and M.A.H.A. Shirazi. "Modeling and scheduling a case of flexible flowshops: Total weighted tardiness minimization". Computers & Industrial Engineering. Vol. 57 Nº 4, pp. 1258-1267. 2009.

[20] E. Salazar. "Programación de Sistemas de Producción con SPS Optimizer". Revista del Instituto Chileno de Investigación Operativa (ICHIO). Vol. 1 Nº 2, pp. 33-46. 2010.