Ingeniare. Revista Chilena de Ingeniería
ISSN: 0718-3291
facing@uta.cl
Universidad de Tarapacá
Chile

Salazar Hornig, Eduardo; Sarzuri Guarachi, René A.
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, núm. 1, enero, 2015, pp. 118-127
Universidad de Tarapacá
Arica, Chile

Algoritmo genético mejorado para la minimización de la tardanza total en un flowshop flexible con tiempos de preparación dependientes de la secuencia

Improved genetic algorithm for total tardiness minimization in a flexible flowshop with sequence-dependent setup times

Eduardo Salazar Hornig[1] René A. Sarzuri Guarachi[1]

Recibido 4 de abril de 2013, aceptado 17 de junio de 2014
Received: April 4, 2013 Accepted: June 17, 2014

RESUMEN

Este trabajo considera un entorno de producción de *flowshop flexible* con tiempos de preparación anticipatorios dependientes de la secuencia. Se presenta un algoritmo genético mejorado para minimizar la tardanza total. La generación de la población inicial se realiza utilizando vecindades de las heurísticas EDD (en inglés *Earliest Due Date*) y *Slack*, considerando además una búsqueda en vecindad IP para mejorar el rendimiento del algoritmo genético propuesto. Los resultados muestran que los algoritmos genéticos con población inicial generada como vecindad de EDD (AG_EDD) y Slack (AG_Slack) mejoran el rendimiento del algoritmo genético básico. El algoritmo AG_EDD muestra un mejor desempeño, característica que se mantiene al incorporar una búsqueda en vecindad IP.

Palabras clave: *Flowshop flexible*, algoritmos genéticos, heurísticas, búsqueda en vecindad, tardanza total, tiempos de preparación dependientes de la secuencia.

ABSTRACT

This paper considers a production environment of a flexible flowshop with anticipatory sequence-dependent setup times. A genetic algorithm improved to minimize total tardiness is presented. The generation of the initial population is performed using the heuristics EDD (Earliest Due Date) and Slack neighborhoods, also considering further search in IP neighborhood to improve the performance of the proposed genetic algorithm. The results show that the genetic algorithms with initial population generated as EDD (AG_EDD) and Slack (AG_Slack) neighborhood improve the performance of the basic genetic algorithm. The AG_EDD algorithm shows better performance, the feature that remains when incorporating IP neighborhood search.

Keywords: Flexible flowshop, genetic algorithms, heuristics, neighborhood search, total tardiness, sequence-dependent setup times.

INTRODUCCIÓN

En entornos cada vez más competitivos y de constante cambio en el mercado, se hace necesario el disponer de herramientas de planificación para obtener el máximo rendimiento de los sistemas productivos. Esto obliga a los sistemas productivos a adaptarse, reduciendo o aumentando el número de máquinas o etapas de producción, a fin de obtener la flexibilidad y poder cumplir con las demandas y compromisos con los clientes. La necesidad de adaptarse y responder a la demanda del mercado en forma rápida y eficiente da lugar a problemas de programación complejos [1].

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

Los problemas de programación de la producción pueden ser clasificados de acuerdo con su configuración productiva, o de acuerdo con el número de máquinas en el sistema.

El *flowshop flexible* (FFS) también llamado comúnmente *flowshop híbrido* (HFS), *línea de flujo flexible* (FFL) o *flowshop* con múltiples procesadores (FSMP) [2-3], es uno de los sistemas que permiten representar a muchos de los sistemas productivos existentes. Se han aplicado en la fabricación de partes electrónicas, industria de embalaje, fabricación de vidrio, industria textil, sector farmacéutico, en la producción de motores de aviones, semiconductores [4-5], como también su aplicación incluye la industria del papel [6] y la industria del acero [7].

El problema de programación de un FFS es un problema NP-hard [6, 8]. Además, los problemas de programación con tiempos de preparación dependientes de la secuencia (*sequence-dependent setup times* (SDST)) se encuentran entre los problemas más difíciles de resolver en programación de trabajos. El caso de una máquina con tiempos de *setup* dependientes de la secuencia es equivalente a un problema del vendedor viajero que es un conocido problema NP-hard [9].

El criterio de minimizar el *makespan* es un criterio frecuentemente utilizado en la programación de un *flowshop flexible* [3], mientras que no lo son otros criterios como la tardanza total.

Actualmente la minimización de la tardanza está tomando mayor interés de parte de los investigadores debido a su importancia en la vida real, que busca reducir la suma de todos los atrasos, o que los trabajos se entreguen a tiempo. Este criterio está relacionado con la satisfacción del tiempo de entrega comprometido con los clientes. La tardanza de un trabajo no solamente afecta el plan de producción del cliente, sino también daña el prestigio del fabricante [10].

En los trabajos [3, 5-6] se puede encontrar una amplia revisión y clasificación de problemas de programación de *flowshop flexible*, mientras que en los trabajos [11-12] se hace referencia al criterio de minimizar la tardanza. Para problemas con tiempos de *setup* dependientes de la secuencia con criterio de minimización de *makespan* ver [9, 13-14] y con criterio de minimización de tardanza ver [15-16].

En la literatura se encuentran investigaciones que evalúan diferentes heurísticas y metaheurísticas, como los algoritmos genéticos (AG) y métodos híbridos que combinan una o varias heurísticas para mejorar el rendimiento. La tendencia en el uso de los algoritmos genéticos es combinar con otras heurísticas para mejorar el desempeño de un AG básico [16-19].

Este estudio se basa en la investigación realizada por [15], donde se evalúa el rendimiento de un AG básico para el problema FFS con *setup* anticipatorios dependientes de la secuencia para el criterio de minimización de tardanza. A la vez se compara con las heurísticas EDD (*earliest due date*), *Slack* (holgura), una adaptación de la regla ATCS, SM (simulación montecarlo) y CR (*critical ratio*), obteniendo resultados interesantes en el que el algoritmo genético básico es superado por las heurísticas EDD y *Slack*, que son métodos heurísticos simples.

En este trabajo se diseña y evalúa un mecanismo que permite mejorar el rendimiento del algoritmo genético básico [15] para el problema de programación de un *flowshop flexible* con tiempos de preparación anticipatorios dependientes de la secuencia y minimización de la tardanza total. La población inicial se genera como vecindades de la solución obtenida con las reglas EDD y *Slack*, a diferencia del trabajo [15] donde se genera en forma aleatoria.

FORMULACIÓN DEL PROBLEMA

El problema de programar un *flowshop flexible* se describe como un conjunto de n trabajos que tienen que ser procesados en k etapas en serie ($k \geq 2$). Cada etapa puede tener más de una máquina idéntica en paralelo y todos los trabajos son procesados siguiendo el mismo flujo de la producción.

Las decisiones principales del FFS consisten en la asignación de trabajos a las máquinas y la secuenciación de los trabajos asignados a cada máquina en cada etapa del proceso productivo, optimizando uno o más criterios (objetivos).

Los supuestos considerados para el FFS en este trabajo son: todos los datos del problema son

conocidos de manera determinista, cada etapa tiene por lo menos una máquina y al menos en una etapa debe tener más de una máquina, las máquinas en cada etapa son iguales en capacidad y velocidad de procesamiento (máquinas idénticas). Los trabajos se procesan sin interrupción, cada trabajo se procesa en una sola máquina en cada etapa, pudiendo ser procesado por cualquiera de las máquinas. Todas las máquinas están disponibles en el tiempo inicial (sin averías o retraso por mantenimiento), los tiempos de transporte entre las etapas son no significativos. El tiempo de espera entre dos etapas no tiene límite, los tiempos de *setup* son dependientes de la secuencia y se consideran anticipatorios.

Este estudio se enfoca en obtener una programación factible que minimice la tardanza total T:

$$T = \sum_{i=1}^{n} T_i$$
$$T_i = \max\{0, C_i - d_i\}$$

T_i es la tardanza del trabajo *i* respecto de su fecha de entrega (d_i), que representa el retraso efectivo del trabajo *i*. Asume un valor positivo si el trabajo es entregado después de su fecha de entrega y cero en caso contrario. C_i es el tiempo de finalización del trabajo *i*.

ALGORITMO GENÉTICO

Los algoritmos genéticos (AG) han demostrado ser un método de optimización con algunas ventajas respecto de métodos convencionales [20]. Un AG mantiene un conjunto de soluciones potenciales en cada generación. Su popularidad se puede atribuir a la aplicación de problemas de optimización continuos y discretos, así como su naturaleza probabilística y menor probabilidad de quedar atrapado en óptimos locales que se presentan en muchos problemas prácticos de optimización [21].

Los algoritmos genéticos fueron introducidos por Holland (1975), como un paradigma que explica los procesos de adaptación de los sistemas naturales y la creación de nuevos sistemas artificiales que funcionan sobre bases similares en el mundo de las computadoras [22].

La terminología del algoritmo genético tiene una mezcla de términos naturales y artificiales. El conjunto de posibles soluciones se llama población. La población inicial evoluciona de generación en generación. Cada individuo de la población se llama cromosoma, y cada código presente en el cromosoma se conoce como gen (el cromosoma es una cadena de genes).

La Figura 1 muestra el esquema básico de un algoritmo genético básico.

```
Algoritmo Genético Básico()
{
 Generar población inicial
 Calcular fitness
 while (condición de término no satisfecha)
 {
  Cruzar y mutar
  Calcular fitness
  Seleccionar siguiente generación
 }
}
```

Figura 1. Esquema básico de AG [23].

Los algoritmos genéticos se caracterizan por la evolución de una población inicial (conjunto de soluciones) que lleva a mejores soluciones por medio de las iteraciones, permitiendo a los individuos más aptos reproducirse y dejar que los individuos menos aptos mueran.

La representación del cromosoma es el primer paso y uno de los aspectos más importantes de un AG. Algunas representaciones pueden llevar a soluciones buenas, mientras que otras no convergen o consumen demasiado tiempo de computación [22].

Para la programación de producción se han propuesto varios métodos como: codificación basado en los trabajos [24], codificación basada en las máquinas, y codificación basada en la operación [23]. En este estudio se utiliza la codificación basada en los trabajos, donde una posición (gen) de un cromosoma representa la posición en que se procesa el trabajo contenido en dicha posición, esto es, el cromosoma corresponde a una secuencia de trabajos que representa el orden en que son asignados (o priorizados) en una etapa. La Figura 2 muestra la representación de un cromosoma basado en los trabajos (o permutación de trabajos).

3	1	5	4	6	2

Figura 2. Representación del cromosoma.

El trabajo 3 es asignado en primer lugar, el trabajo 1 es asignado en segundo lugar, y así sucesivamente. Debido a que el correspondiente centro de trabajo (etapa) puede estar configurado por más de una máquina, y estas ya tener trabajos previamente asignados, el trabajo es asignado en aquella máquina donde finaliza primero.

Otros autores como en [9, 14, 25] utilizan una representación de clave aleatoria (*random keys genetic algorithms*) propuesta por Bean [26].

Para la generación de la población inicial [8, 15, 24-25, 27] utilizan una generación aleatoria, sin incluir algún conocimiento previo del conjunto de cromosomas [28]. Mientras en [14, 16, 29-30] utilizan heurísticas para la generación de la población inicial, introduciendo un conocimiento previo de un conjunto probable de buenas soluciones (cromosomas), el cual proporciona una forma de aceleración en la obtención de una buena solución. En este trabajo se utiliza la generación de la población inicial como vecindades de las soluciones generadas por las heurísticas EDD y *Slack* debido al buen desempeño observado respecto del algoritmo genético básico estudiado en [15].

Para la generación de la población inicial se definieron las siguientes estrategias: generación como vecindad de la solución entregada por la heurística EDD (AG_EDD), y generación como vecindad de la solución entregada por *Slack* (AG_Slack). Las vecindades generadas a partir de las heurísticas EDD y *Slack* se obtuvieron por medio de la vecindad *swap* (intercambio), sin incluir las respectivas soluciones entregadas por las heurísticas EDD y *Slack* en la población inicial de ambas estrategias.

La probabilidad de sobrevivencia de un individuo está determinada por su condición de aptitud (*fitness*), mediante la evolución los más aptos son seleccionados. Los individuos con mejor *fitness* se escogen con más frecuencia que los individuos con peores valores. La función *fitness* se asocia al objetivo de minimizar la tardanza total. Para la selección de los individuos de la actual generación que se cruzan y/o mutan se utiliza un esquema de selección probabilística basada en su aptitud (*fitness*).

Los operadores de cruce que combinan las características de dos padres para crear nuevos cromosomas (hijos) para la próxima generación tienden a aumentar la calidad de las poblaciones y la fuerza de convergencia. En este trabajo se emplean los operadores *partially mapped crossover* (PMX) y *order crossover* OX como operadores de cruzamiento.

El operador PMX selecciona dos puntos de cruce de manera aleatoria entre 1 y (n – 1) donde n es la longitud del cromosoma (número de trabajos). Las subcadenas de código genético definidos por los dos puntos de corte son llamadas *sección de mapeo*. Las dos subcadenas son intercambiadas para generar los nuevos hijos. Se determina la relación de mapeo correspondientes entre las dos secciones, y mediante el uso de esta relación, se legalizan los descendientes hijos. La Figura 3 ilustra el operador PMX.

El operador OX selecciona aleatoriamente dos puntos de cruce entre 1 y (n – 1). La cadena de códigos entre estos dos puntos de corte se copia en las mismas posiciones a la descendencia. Los códigos copiados son borrados del otro padre, heredando el descendiente el resto de los códigos, comenzando con la primera posición del segundo punto de cruzamiento. Después de cambiar los roles de los padres, el mismo procedimiento se aplica para producir el segundo hijo. En la Figura 4 se ilustra el operador OX.

El operador de mutación transforma levemente un individuo, evita la convergencia prematura y cambia la dirección de búsqueda. En este trabajo se utiliza el operador de mutación de intercambio (*swap*). Se eligen dos posiciones (diferentes) en forma aleatoria entre 1 y n, los genes de cada posición son intercambiados. La Figura 5 ilustra el operador de mutación *swap*.

Figura 3. Operador de cruzamiento PMX.

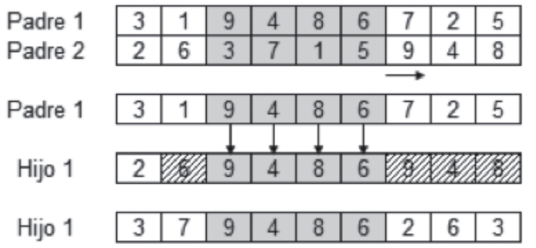

Figura 4. Operador de cruzamiento OX.

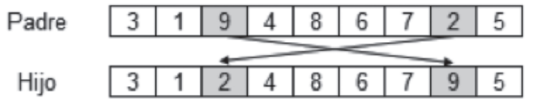

Figura 5. Operador de mutación Swap.

EXPERIMENTACIÓN

En una configuración FFS, las máquinas se organizan en serie, por etapas, donde en cada etapa las máquinas son idénticas. Los trabajos se procesan en solo una máquina en cada etapa. Para el problema de programación se consideran tiempos de *setup* anticipatorios dependientes de la secuencia. El objetivo es encontrar una secuencia de trabajos de manera que el procedimiento de asignación genere un programa de mínima tardanza total mediante un algoritmo genético mejorado. A modo de comparación, se evalúan las instancias generadas por [15].

Los datos que definen los problemas de evaluación están definidos de acuerdo con [15].

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

La fecha de entrega (*due date*) se asigna de acuerdo con la ecuación (adoptada de [31]):

$$d_i = \sum_{k=1}^{m} P_{ik} + \sum_{k=1}^{m} \overline{s}_{ik} + (n-1) * \overline{p}_i * u$$

donde $p_i = \sum_{k=1}^{m} P_{ik} / m$; $\overline{s}_{ik} = \sum_{j=1}^{n} S_{jik} / n$ y u es observación de 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 en el sistema.

Debido a que se utilizan los problemas propuestos por [15] y el propósito es comparar el rendimiento del AG básico con el AG mejorado propuesto en este trabajo, se utilizaron los parámetros del AG básico calibrados en [15].

- Tamaño de población: $N_p = 100$
- Número de generaciones: $N_g = 700$
- Probabilidad de cruzamiento: $P_c = 0{,}9$
- Probabilidad de mutación: $P_m = 0{,}1$
- Numero de réplicas: 30
- Criterio de detención: número de generaciones

Con el fin de mejorar la solución del algoritmo genético, se aplica un algoritmo de búsqueda en vecindad IP a los algoritmos AG_EDD y AG_Slack, que mejora la solución final obtenida por cada estrategia. Por lo tanto, la combinación con búsqueda en vecindad de las estrategias se denominará: AG_EDD_BV y AG_Slack_BV. Para efectos de comparación, las soluciones entregadas por las heurísticas AG_Básico, EDD y *Slack* son mejoradas por la búsqueda en vecindad IP.

La población inicial para el algoritmo AG_EDD se conforma en un 100% como vecindad de las heurísticas EDD, pero sin incluir la solución generada por EDD, esto con el objetivo de evaluar la capacidad del algoritmo de obtener por medio del proceso evolutivo una mejor solución que la semilla de la cual se generó la vecindad. Análogamente para el algoritmo AG_Slack.

En la literatura la medida de rendimiento más utilizada para comparar diferentes métodos es el porcentaje de desviación relativa RPD. Debido a que el RPD no se adapta para nuestro criterio de minimización de la tardanza (el valor óptimo puede ser 0), se utiliza el índice de desviación relativa DRI [32]:

$$DRI = \frac{Method_{sol} - Best_{sol}}{Worst_{sol} - Best_{sol}}$$

donde $Worst_{sol} - Best_{sol}$ es la diferencia entre la peor y mejor solución obtenidas entre todos los métodos. $Method_{sol}$ es la solución obtenida para un algoritmo en la instancia dada.

La evaluación de los problemas se realizó en un computador con procesador Intel Pentium dual –core de 2GB de Memoria (RAM), utilizando rutinas adaptadas del software SPS_Optimizer [33], herramienta diseñada para la programación de operaciones.

RESULTADOS

La Figura 6 muestra el índice de desviación relativa al comparar todos los algoritmos. El AG_EDD obtiene en promedio la menor desviación relativa de un 0,50%, lo que significa que este algoritmo genera la mejor solución en la mayoría de los casos. Le sigue la estrategia AG_Slack con un 9,82%. Las heurísticas EDD y *Slack* obtienen en promedio una desviación relativa de un 19,1% y 24,5%. El AG_Básico obtiene la mayor desviación de un 84,6%.

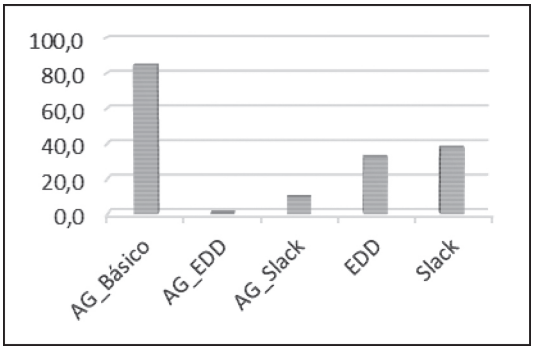

Figura 6. Índice de desviación relativa.

En las Figuras 7 a 10 se muestran los resultados de tardanza respecto de las 30 instancias utilizando los diferentes métodos comparados. Al comparar AG_EDD con AG_Slack (Figura 7), se observa que la curva de AG_EDD tiende a estar por debajo de AG_Slack con bajas diferencias en la tardanza total. Análogamente al comparar AG_EDD con EDD, Slack y AG_Básico, la tendencia de AG_EDD es más marcada, y las diferencias de tardanza de los métodos aumentan. Se observa que AG_EDD y AG_Slack obtienen los menores valores de la tardanza total superando a los demás métodos, y que AG_Básico obtiene los valores más elevados de tardanza total. También la Figura 11 muestra esta apreciación.

AG_EDD encuentra el mayor porcentaje de veces la mejor solución (48,1%) seguido de AG_Slack (20,4%) superando a las heurísticas EDD (11,1%) y Slack (14,8%). El AG_Básico en promedio presenta el de peor desempeño alcanzando un 5,6% 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 que encuentra la mejor solución y con valores bajos de desviación relativa. Mientras que AG_Básico presenta el peor desempeño, ya que obtiene el menor porcentaje de veces la mejor solución y muestra un valor alto de desviación relativa. Las heurísticas EDD y Slack muestran un desempeño intermedio. La Tabla 1 resume lo expuesto. El buen desempeño del algoritmo AG_EDD y AG_Slack se debe a que tanto EDD como Slack hacen uso de información relevante (tiempos de entrega) al realizar la priorización de los trabajos, lo que se traspasa a la población inicial de estos algoritmos. Por otro lado, AG_ Básico obtiene el peor desempeño debido a que no incorpora el conocimiento del problema, solo actúa en la exploración del espacio de soluciones.

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

Al integrar la búsqueda en vecindad en los métodos, se observa que AG_EDD_BV obtiene en promedio la menor desviación relativa de un 0,14%, seguido de AG_Slack_BV con un 17,33%. Las heurísticas EDD_BV y Slack_BV obtienen una desviación relativa promedio similar de un 29,28% y 30,20% respectivamente. AG_Básico_BV presenta el de peor desempeño obteniendo la mayor desviación

Tabla 1. Resumen comparación de algoritmos.

Método	AG_Básico	AG_EDD	AG_Slack	EDD	Slack
Índice de desviación relativa (%)	84,6	0,5	9,4	32,4	37,7
Mejor Solución (%)	5,6	48,1	20,4	11,1	14,8

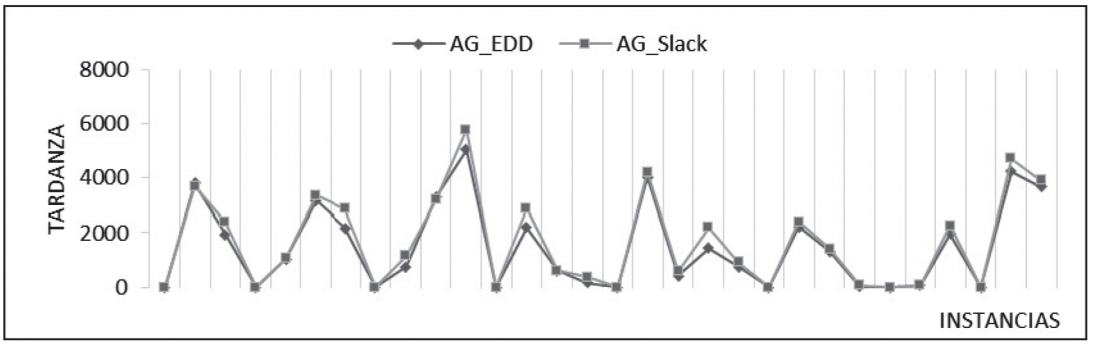

Figura 7. Comparación de AG_EDD y AG_Slack.

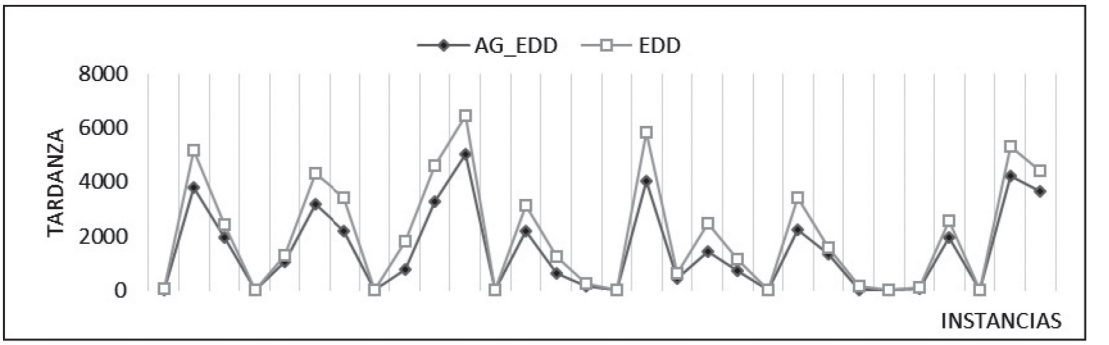

Figura 8. Comparación de AG_EDD y EDD.

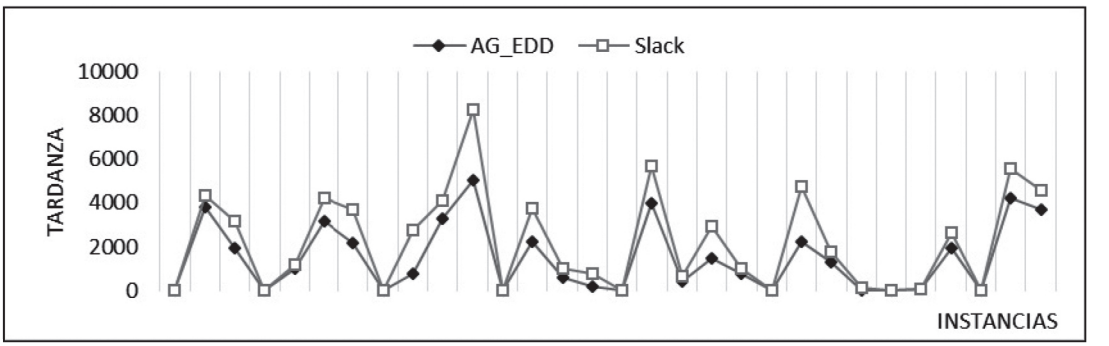

Figura 9. Comparación de AG_EDD y *Slack*.

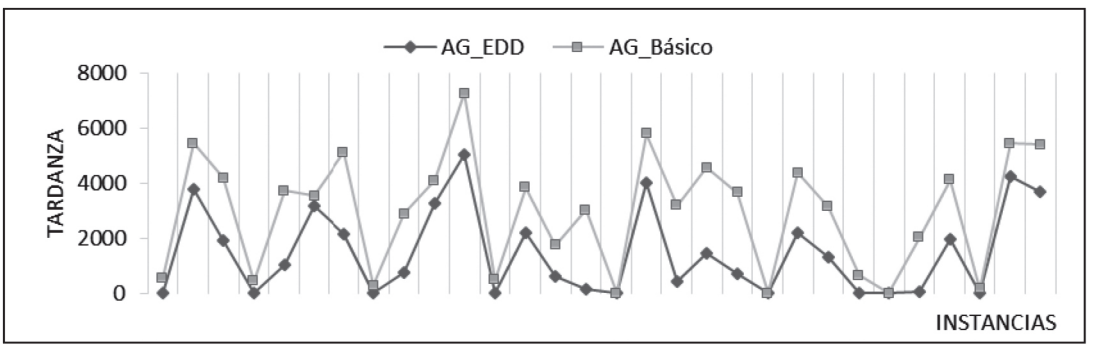

Figura 10. Comparación de AG_EDD y AG_Básico.

de un 71,5%. La Figura 11 muestra el índice de desviación relativa.

Respecto del porcentaje de veces que un método encuentra la mejor solución, AG_EDD_BV la obtiene en un mayor porcentaje igual a 46,7% seguido de AG_Slack_BV con el 18,3%. Las heurísticas EDD_BV y Slack_BV la obtienen el 13,3% y 15,0% de las veces respectivamente seguido por AG_Básico_BV que encuentra la mejor solución en el 6,7% de las veces. La Tabla 2 resume el índice de desviación relativa y el porcentaje de veces que un método encuentra mejor solución.

AG_EDD_BV seguido de AG_Slack_BV son los que presentan el mejor rendimiento obteniendo bajos índices de desviación relativa y mayores porcentajes de veces que encuentran la mejor solución, mientras que AG_Básico_BV presenta el peor desempeño, ya que obtiene un alto índice de desviación relativa y un bajo porcentaje de veces que encuentra la mejor solución. Las heurísticas EDD_BV y Slack_BV presentan un desempeño intermedio con una similar desviación relativa y porcentaje de veces que encuentran la mejor solución (ver Tabla 2).

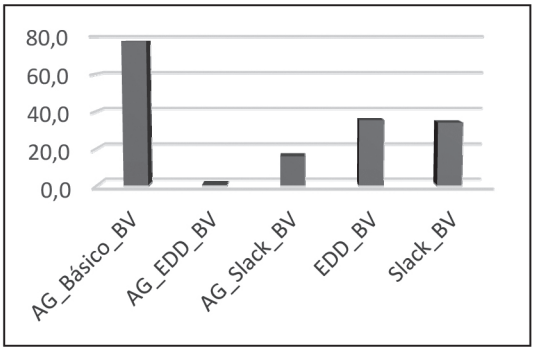

Figura 11. Índice de desviación relativa.

Al integrar la búsqueda en vecindad IP en los métodos, se observa que estos mantienen el mismo orden respecto del índice de desviación relativa que cuando no se aplica la búsqueda en vecindad. En ambos casos (con o sin búsqueda en vecindad), AG_EDD seguido AG_Slack presenta el mejor desempeño comparado con los demás métodos.

Los tiempos CPU obtenidos con AG_Básico, AG_EDD y AG_Slack presentan en promedio un tiempo del orden de 20 [s], mientras que EDD y *Slack* presentan un tiempo CPU no significativo. Al integrar la búsqueda en vecindad IP los tiempos computacionales se incrementan en el orden de 0,10 [s].

CONCLUSIONES

En este trabajo se ha considerado el problema de programación de un *flowshop flexible* con tiempos de preparación anticipatorios dependientes de la secuencia con el objetivo de minimización de la tardanza total.

En este estudio se ha propuesto una mejora de un algoritmo genético básico basada en la generación de la población inicial como una vecindad de las soluciones obtenidas por las heurísticas EDD (AG_EDD) y Slack (AG_Slack), a los que también se les aplicó una búsqueda en vecindad IP.

Al evaluar el desempeño de los algoritmos propuestos AG_EDD y AG_Slack muestran mejores rendimientos superando a las heurísticas EDD, Slack y AG_Básico. AG_EDD y AG_Slack muestran los menores valores de desviaciones relativas y de tardanza total con un mayor porcentaje de veces que encuentra la mejor solución.

Al integrar la búsqueda en vecindad en los métodos, los resultados muestran que el AG_EDD_BV y AG_Slack_BV obtienen los mejores resultados superando a las heurísticas EDD_BV, Slack_BV, y AG_Básico_BV con un mayor porcentaje de veces que encuentra mejor solución y con bajos índices de desviación relativa.

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

Método	**AG_Básico_ BV**	**AG_EDD_ BV**	**AG_Slack_ BV**	**EDD_ BV**	**Slack_ BV**
Índice de desviación relativa (%)	77,5	0,3	15,8	35,3	34,0
Mejor solución (%)	6,7	46,7	18,3	13,3	15,0

Entre ambas propuestas AG_EDD supera a AG_Slack. Cuando se integra la búsqueda en vecindad los resultados relativos entre los métodos se mantiene. La población inicial generada como vecindad de las soluciones entregadas por las heurísticas EDD y *Slack*, mejora el rendimiento del algoritmo genético básico, superando también al de las heurísticas EDD y *Slack*.

La integración del algoritmo de búsqueda en vecindad IP a las soluciones entregadas por los métodos mejora significativamente los valores de tardanza total, que se traduce en una reducción del índice de desviación relativa en la mayoría de los métodos (se tiende a reducir la diferencia entre la peor y mejor solución entre todos los métodos).

REFERENCIAS

[1] J.N.D. Gupta, K. Krüger, V. Lauff, F. Werner and Y.N. Sotskov. "Heuristics for hybrid flow shops with controllable processing times and assignable due dates". Computers and Operations Research. Vol. 29, Issue 10, pp. 1417-1439. 2002.

[2] C.-L. Chen and C.-L. Chen. "Bottleneck-based heuristics to minimize total tardiness for the flexible flow line with unrelated parallel machines". Computers & Industrial Engineering. Vol. 56, Issue 4, pp. 1393-1401. 2009.

[3] 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, Issue 8, pp. 1439-1454. 2010.

[4] J. Behnamian and M. Zandieh. "A discrete colonial competitive algorithm for hybrid flowshop scheduling to minimize earliness and quadratic tardiness penalties". Expert Systems with Applications. Vol. 38, Issue 12, pp. 14490-14498. 2011.

[5] D. Quadt and H. Kuhn. "A taxonomy of flexible flow line scheduling procedures". European Journal of Operational Research. Vol. 178, Issue 3, pp. 686-698. 2007.

[6] R. Ruiz and J.A. Vázquez-Rodríguez. "The hybrid flow shop scheduling problem". European Journal of Operational Research. Vol. 205, Issue 1, pp. 1-18. 2010.

[7] C.-J. Liao, E. Tjandradjaja, and T.-P. Chung. "An approach using particle swarm optimization and bottleneck heuristic to solve hybrid flow shop scheduling problem". Applied Soft Computing. Vol. 12, Issue 6, pp. 1755-1764. 2012.

[8] H. Khademi Zare and M.B. Fakhrzad. "Solving flexible flow-shop problem with a hybrid genetic algorithm and data mining: A fuzzy approach". Expert Systems with Applications. Vol. 38, Issue 6, pp. 7609-7615. 2011.

[9] M. Zandieh, S.M.T. Fatemi Ghomi and S. M. Moattar Husseini. "An immune algorithm approach to hybrid flow shops scheduling with sequence-dependent setup times". Applied Mathematics and Computation. Vol. 180, Issue 1, pp. 111-127. 2006.

[10] A. Hamidinia, S. Khakabimamaghani, M. M. Mazdeh and M. Jafari. "A genetic algorithm for minimizing total tardiness/earliness of weighted jobs in a batched delivery system". Computers & Industrial Engineering. Vol. 62, Issue 1, pp. 29-38. 2012.

[11] E. Vallada, R. Ruiz and G. Minella. "Minimising total tardiness in the m-machine flowshop problem: A review and evaluation of heuristics and metaheuristics". Computers and Operations Research. Vol. 35, Issue 4, pp. 1350-1373. 2008.

[12] H.-S. Choi and D.-H. Lee. "Scheduling algorithms to minimize the number of tardy jobs in two-stage hybrid flow shops". Computers and Industrial Engineering. Vol. 56, Issue 1, pp. 113-120. 2009.

[13] M.E. Kurz and R.G. Askin. "Comparing scheduling rules for flexible flow lines". International Journal of Production Economics. Vol. 85, Issue 3, pp. 371-388. 2003.

[14] M. Hekmatfar, S.M.T. Fatemi Ghomi and B. Karimi. "Two stage reentrant hybrid flow shop with setup times and the criterion of minimizing makespan". Applied Soft Computing. Vol. 11, Issue 8, pp. 4530-4539. 2011.

[15] E. Salazar and B.N. 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.

[16] K. Kianfar, S.M.T. Fatemi Ghomi and A. Oroojlooy Jadid. "Study of stochastic sequence-dependent flexible flow shop via developing a dispatching rule and a hybrid GA". Engineering Applications of Artificial Intelligence. Vol. 25, Issue 3, pp. 494-506. 2012.

[17] J.F. Goncalves, J.J. Magalhaes and M. Resende. "A hybrid Genetic Algorithm for the job shop scheduling problem". European Journal of Operational Research. Vol. 167, Issue 1, pp. 77-95. 2005.

[18] A.N. Haq, T.R. Ramanan, K.S. Shashikant and R. Sridharan. "A hybrid neural network-genetic algorithm approach for permutation flowshop scheduling. International Journal of Production Research. Vol. 48, Issue 14, pp. 4217-4231. 2010.

[19] E. Salazar. "Scheduling Multi-Stage Production Systems with Continuity Constraints - The Steelmaking and Continuous Casting System". Dissertation, RWTH Aachen University. 2013.

[20] O. Hasançebi and F. Erbatur. "Evaluation of crossover techniques in genetic algorithm based optimum structural design". Computers and Structures. Vol. 78, Issue 1-3, pp. 435-448. 2000.

[21] A.A. Javadi, R. Farmani and T.P. Tan. "A hybrid intelligent genetic algorithm". Advanced Engineering Informatics. Vol. 19, Issue 4, pp. 255-262. 2005.

[22] G. Renner and A. Ekárt. "Genetic algorithms in computer aided design". Computer-Aided Design. Vol. 35, Issue 8, pp. 709-726. 2003.

[23] J.-S. Chen, J.C.-H. Pan and C.-M. Lin. "A hybrid genetic algorithm for the re-entrant flow-shop scheduling problem". Expert Systems with Applications. Vol. 34, Issue 1, pp. 570-577. 2008.

[24] H.-M. Cho, S.-J. Bae, J. Kim and I.-J. Jeong. "Bi-objective scheduling for reentrant hybrid flow shop using Pareto genetic algorithm". Computers & Industrial Engineering. Vol. 61, Issue 3, pp. 529-541. 2011.

[25] M.E. Kurz and R.G. Askin. "Scheduling flexible flow lines with sequence-dependent setup times". European Journal of Operational Research. Vol. 159, Issue 1, pp. 66-82. 2004.

[26] J.C. Bean. "Genetic Algorithms and Random Keys for Sequencing and Optimization". ORSA Journal of Computing. Vol. 6, Issue 2, pp. 154-160. 1994.

[27] O. Engin, G. Ceran and M.K. Yilmaz. "An efficient genetic algorithm for hybrid flow shop scheduling with multiprocessor task problems". Applied Soft Computing. Vol. 11, Issue 3, pp. 3056-3065. 2011.

[28] A.W.M. Ng and B.J.C. Perera. "Selection of genetic algorithm operators for river water quality model calibration". Engineering Applications of Artificial Intelligence. Vol. 16, Issue 5-6, pp. 529-541. 2003.

[29] B. Akrami, B. Karimi and S.M. Moattar Hosseini. "Two metaheuristic methods for the common cycle economic lot sizing and scheduling in flexible flow shops with limited intermediate buffers: The finite horizon case". Applied Mathematics and Computation. Vol. 183, Issue 1, pp. 634-645. 2006.

[30] M.R. Amin-Naseri and M.A. Beheshti-Nia. "Hybrid flow shop scheduling with parallel batching". International Journal of Production Economics. Vol. 117, Issue 1, pp. 185-196. 2009.

[31] T. Eren and E. Güner. "A bicriteria flowshop scheduling problem with setup times". Applied Mathematics and Computation. Vol. 183, Issue 2, pp. 1292-1300. 2006.

[32] E. Vallada and R. Ruiz. "Genetic algorithms with path relinking for the minimum tardiness permutation flowshop problem". Omega. Vol. 38, Issue 1-2, pp. 57-67. 2010.

[33] E. Salazar. "Programación de Sistemas de Producción con SPS Optimizer". Revista del Instituto Chileno de Investigación Operativa (ICHIO). Vol. 1 N° 2, pp. 33-46. 2010.