Minimización de la tardanza para el *flowshop flexible* con *setup* utilizando heurísticas constructivas y un algoritmo genético

Tardiness minimization for the flexible flowshop with setup using constructive heuristics and a genetic algorithm

Eduardo Salazar Hornig[1] Belén Figueroa Morales[1]

Recibido 24 de mayo de 2010, aceptado 9 de abril de 2012
Received: May 24, 2010 Accepted: April 9, 2012

RESUMEN

En este trabajo se considera el problema de programar *n* trabajos en un *flowshop flexible* de *k* etapas, con diferente número de máquinas idénticas por etapa considerando tiempos anticipatorios de preparación dependientes de la secuencia (SDST) y minimización de la tardanza. Se comparan los resultados de heurísticas constructivas y un algoritmo genético estándar. La evaluación de los métodos se realiza en forma experimental sobre un conjunto de problemas de prueba generados aleatoriamente. Los resultados muestran que el algoritmo genético supera a alguna de las heurísticas comparadas pero no a todas.

Palabras clave: *Flexible flowshop*, algoritmos genéticos, heurísticas constructivas, tiempos anticipatorios de preparación dependientes de la secuencia, búsqueda en vecindad.

ABSTRACT

This paper studied the problem of sequencing n jobs in a k-stages flexible flowshop with different number of parallel machines per stage with anticipatory sequence dependent setup times (SDST) and tardiness minimization. The performance of constructive heuristics and the genetic algorithm are compared. The evaluation of the methods is made experimentally over a set of randomly generated test problems. The results indicate that the genetic algorithm do not outperforms all of the compared heuristics.

Keywords: Flexible flowshop, genetic algorithms, constructive heuristics, anticipatory sequence dependent setup times, neighborhood search.

INTRODUCCIÓN

La programación de la producción se entiende como la asignación de los recursos productivos a la ejecución de tareas productivas en un horizonte de corto plazo. En un entorno de manufactura moderna la programación de la producción es una tarea compleja y difícil de resolver en forma eficiente. Actualmente, tanto la rapidez como el cumplimiento de entregas a clientes constituyen una ventaja estratégica para la competitividad de los sistemas productivos. En tal sentido, lograr secuencias de producción que minimicen atrasos en la entrega es de vital importancia. Asociado a lo anterior se presenta el problema de la reducción de lotes de producción, originando frecuentes preparaciones de las instalaciones incurriendo en costosos tiempos de preparación, cuya relevancia en la programación se manifiesta en una amplia gama de configuraciones productivas [1].

En un sistema de manufactura, la disposición de la maquinaria define el tipo de configuración productiva necesaria para realizar el proceso de transformación.

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

El *flowshop flexible* (FFS) es una generalización del problema clásico de *flowshop* de una máquina por etapa, en el que se dispone de máquinas en paralelo en cada etapa. Esta configuración se encuentra frecuentemente en la industria textil, cerámica y del plástico, entre otras. En adelante nos referiremos a *setup* como concepto equivalente al de tiempos (anticipatorios) de preparación dependientes de la secuencia.

Dado que el problema de programar un *flowshop flexible* con diferentes objetivos es un conocido problema NP-completo [2] en la literatura se han propuesto principalmente algoritmos heurísticos y metaheurísticos.

Si bien para el problema FFS con tiempos de *setup* y minimización del *makespan* existen numerosos trabajos en la literatura [3-7], no es así para el caso de la minimización de la tardanza o de alguna de sus variantes.

El FFS de dos etapas ha sido tratado a través de enfoques heurísticos óptimos para la minimización de la tardanza máxima con *setup* [8] combinando un modelo de programación lineal con un algoritmo genético; también se ha considerado la minimización del número de trabajos atrasados [9].

El problema de un FFS de múltiples etapas sin *setup* con minimización de la tardanza máxima se ha tratado a través de métodos heurísticos [10], como también se ha considerado la minimización del costo de prontitud y tardanza [11]. Otros objetivos, como la minimización de una función que pondera el *makespan* (lapso de tiempo en el que se procesa la totalidad de los trabajos) y el número de trabajos tardíos en un FFS con *setup* y máquinas paralelas no relacionadas, han sido tratados a través de heurísticas constructivas y de metaheurísticas como *simulated annealing*, *tabu search* y algoritmos genéticos incorporando búsqueda en vecindad [12].

DEFINICIÓN DEL PROBLEMA

El problema de programar un *flexible flowshop* consiste en resolver la programación de *n* trabajos en un sistema secuencial de *m* etapas, en las que pueden existir una o más máquinas en paralelo. La secuencia de proceso de los trabajos se inicia en la etapa uno y finaliza en la etapa *m*. Los supuestos para el problema FFS considerado en este trabajo son:

- Las máquinas en cada etapa son idénticas.
- Todo trabajo i (i = 1, … , n) se procesa en cada etapa k (k = 1, … , m) en sólo una máquina.
- Una máquina puede procesar sólo un trabajo a la vez.
- Todo trabajo i (i = 1, … , n) tiene asociado una fecha de entrega d_i (*due date*).
- El tiempo de proceso del trabajo i en la etapa k está dado por p_{ik} (i = 1, … , n; k = 1, … , m).
- Los tiempos de preparación (*setup*) son dependientes de la secuencia y de carácter anticipatorio. Para procesar el trabajo j después del trabajo i en la etapa k el tiempo de preparación está dado por s_{ijk} (i = 1, … , n; j = 1, … , n; k = 1, … , m), donde s_{iik} representa la preparación si el trabajo i es el primer trabajo en una máquina de la etapa k.
- El proceso de un trabajo en una máquina es sin interrupción (*nonpreemption*).
- Todas las máquinas están disponibles en el tiempo inicial y operan sin fallas en el horizonte de programación.
- El objetivo es minimizar la tardanza total T.

La tardanza total corresponde a la suma de las tardanzas individuales T_i de cada trabajo:

$$T = \sum_{i=1}^{n} T_i$$

con $T_i = \max\{C_i - d_i, 0\}$ donde C_i es el tiempo de finalización del trabajo i en el programa. T_i representa el atraso efectivo del trabajo i en el caso $T_i > 0$; en el caso $T_i = 0$ el trabajo i se concluye antes de su fecha de entrega. La situación ideal en la que todos los trabajos se entreguen a tiempo está reflejada por T = 0 (en este caso $T_i = 0$, $\forall$ i).

Que los tiempos de *setup* sean dependientes de la secuencia significa que el tiempo de *setup* antes de procesar un trabajo en una máquina va a depender del trabajo procesado inmediatamente antes en la misma máquina. El concepto de *setup* anticipatorio significa que la preparación de la máquina que procesará un trabajo puede comenzar antes de que éste haya finalizado su operación previa siempre que la máquina donde se procesará esté desocupada.

```
procedure AsignaciónTrabajos
  Definir ListaOrden de trabajos ordenados por un criterio.
  ListaTrabajos ← ListaOrden
  while (ListaTrabajos no vacía) do
    Asignar primer trabajo de ListaTrabajos a la máquina de la etapa 1 donde finaliza antes.
    Eliminar primer trabajo de ListaTrabajos.
  endwhile
  A medida que los trabajos terminan su proceso en una etapa, pasan a la siguiente donde son asignados a la máquina
  libre que los finaliza antes, si todas las máquinas de la etapa están ocupadas, el trabajo espera. Al desocuparse
  una máquina, de aquellos trabajos que esperan, se selecciona aquel de mayor prioridad definida por ListaOrden.
endprocedure
```

Figura 1. Pseudocódigo para el Procedimiento de Asignación de Trabajos para el FFS.

MÉTODOS HEURÍSTICOS DE SOLUCIÓN

En este trabajo se evalúan heurísticas definidas como extensiones de heurísticas basadas en reglas de despacho como EDD (*earliest due date*), *Slack* (holgura) y CR (*critical ratio*), una extensión de la heurística ATCS (*apparent tardiness cost with setup*), simulación Montecarlo (SM) y un algoritmo genético estándar (AG). Posteriormente se aplica una heurística de búsqueda en vecindad IP (*pairwise interchange*) para mejorar la solución obtenida por cada heurística.

Todas las heurísticas consideradas generan una secuencia de trabajos que actúa como una lista de prioridad controlando la asignación de trabajos a las máquinas. El procedimiento de asignación basado en este concepto se presenta en la Figura 1.

Para extender las heurísticas al FFS con *setup* se debe asociar un único tiempo de proceso a cada trabajo, el que se define como el tiempo estimado de ocupación de máquina pe_{ik} para cada trabajo i (i = 1, … ,n) en cada etapa k (k = 1, … ,m):

$$pe_{ik} = p_{ik} + se_{ik} \quad \text{con} \quad se_{ik} = \sum_{j=1}^{n} s_{jik} \Big/ \text{n}$$

donde el término se_{ik} es el promedio de los *setup* que el trabajo i puede tener en la etapa k. La estimación del tiempo total de proceso del trabajo i se define por:

$$pe_i = \sum_{k=1}^{m} pe_{ik}$$

La extensión de una heurística al problema FFS con *setup* mantiene su nombre original, pero en el contexto de este trabajo se entenderá que se trata de la heurística aplicada al FFS.

Los métodos heurísticos de solución utilizados en este trabajo se definen a continuación:

EDD: La heurística EDD construye una lista ordenando los trabajos de menor a mayor d_i. Luego se aplica el procedimiento *AsignaciónTrabajos* de la Figura 1.

Slack: La heurística *Slack* construye una lista ordenando los trabajos de menor a mayor valor de la holgura estimada inicialmente $d_i - pe_i$. Luego aplica el procedimiento de *AsignaciónTrabajos* de la Figura 1.

CR: La heurística CR construye una lista ordenando los trabajos de menor a mayor valor de la razón entre el tiempo restante a su compromiso de entrega y el tiempo restante de proceso. Se utiliza de manera estática, es decir, se determina al inicio de la programación construyendo una lista ordenando los trabajos de menor a mayor del ratio crítico d_i / pe_i (en el instante inicial 0 el tiempo restante para la entrega del trabajo i es (d_i – 0) y pe_i es la estimación del trabajo restante del trabajo). Luego aplica el procedimiento de *AsignaciónTrabajos* de la Figura 1.

ATCS: La extensión propuesta de la heurística ATCS utiliza las heurísticas ATCS de una máquina [13] y de máquinas paralelas idénticas [14] en la etapa cuello de botella del FFS, la que se define como la etapa con mayor carga (ocupación) por máquina. La carga para la etapa k (c_k) se obtiene como:

$$c_k = \sum_{i=1}^{n} pe_{ik} \Big/ m_k$$

con m_k igual al número de máquinas de la etapa k. La heurística se aplica usando el siguiente procedimiento: **1)** Determinar $k_{cb} = \arg[\min_{k=1,\ldots,m}\{c_k\}]$, la etapa cuello de botella y **2)** Aplicar la heurística ATCS para una máquina (máquinas paralelas) si $m_k = 1$ ($m_k > 1$), considerando los parámetros de tiempos de proceso y *setup* asociados a la etapa k_{cb} redefiniendo los *due dates* como d_i / m (i = 1, … , *n*).

SM: La simulación Montecarlo genera en forma aleatoria una muestra de N secuencias de n trabajos, evaluando cada una de ellas mediante el procedimiento *AsignaciónTrabajos* de la Figura 1, entregando como resultado la secuencia de la muestra con menor tardanza.

AG: El algoritmo genético (se explica en detalle en la sección siguiente) utiliza individuos cuya estructura de cromosoma corresponde a una secuencia de n trabajos. Así, la evaluación de un individuo es la tardanza que se obtiene en la programación aplicando el procedimiento *AsignaciónTrabajos* de la Figura 1.

ALGORITMO GENÉTICO

Los algoritmos genéticos fueron introducidos por Holland en 1975 modelando el proceso de adaptación de la evolución natural de una población de individuos a través de un algoritmo de búsqueda. Los algoritmos genéticos, que asocian el concepto de *individuo* a una solución factible de un problema, y el de *población* a un conjunto de *individuos*, han sido ampliamente aplicados en problemas de secuenciamiento [15]. Los individuos son evaluados a través de una función de aptitud denominada *fitness*, que corresponde a una medida de la calidad (salud) del *individuo* como solución del problema.

En este trabajo se utiliza la estructura cromosómica de un *individuo* definida como una secuencia de *n* trabajos (representación de permutación). La Figura 2 ilustra un individuo para un problema en el que se deben programar n = 6 trabajos. Los individuos son evaluados a través de la tardanza que se obtiene aplicando el procedimiento *AsignaciónTrabajos* de la Figura 1 a la secuencia de su cromosoma (esta secuencia corresponde a la *ListaOrden* del procedimiento), definiendo el *fitness* de un individuo como *fitness* = 1 / (c + T), donde c es una constante positiva que evita la división por 0 para un individuo con T = 0.

Individuo: 3 – 6 – 1 – 5 – 2 – 4

Figura 2. Ilustración de estructura cromosómica.

Debido a que en este trabajo se compara el algoritmo genético con extensiones de heurísticas que se definen en base a una lista que prioriza los trabajos de acuerdo a un determinado criterio (por ejemplo EDD prioriza los trabajos según la fecha de entrega más próxima), se optó por una estructura cromosómica que pueda ser considerada como una lista de priorización.

La Figura 3 muestra el pseudocódigo para una estructura general de un algoritmo genético. La población inicial de soluciones (P_0) se determina en forma aleatoria, y el proceso de selección durante el proceso evolutivo se realiza de acuerdo a una distribución de probabilidades proporcional a su *fitness* [15], esto es, los individuos de mayor *fitness* (menor tardanza) tienen una probabilidad más alta de ser seleccionados.

```
procedure Algoritmo genético
t ← 0
   inicializar P_t
   evaluar P_t
   while (t < N_g) do
      t ← t+1
      seleccionar padres de P_t-1
      formar población P_t
      evaluar P_t
   endwhile
endprocedure
```

Figura 3. Pseudocódigo de un algoritmo genético [15].

La formación de la población de la generación t (P_t), a partir de la población de la generación t – 1 (P_{t-1}), se realiza de acuerdo a procesos de cruzamiento de dos individuos (padres) seleccionados aleatoriamente sujetos a una probabilidad de cruzamiento (p_c), generando la descendencia (hijos). El propósito del cruzamiento es intercambiar información entre los cromosomas padres con el objetivo de generar mejores descendientes. Sobre la descendencia opera una mutación de acuerdo a una probabilidad de

mutación (p_m), siendo el objetivo de la mutación el de producir variabilidad en la población de descendientes. La evaluación de la población P_t evalúa a cada individuo determinando su *fitness*. El proceso evolutivo utiliza un *operador de cruzamiento* y un *operador de mutación*, y se realiza hasta que se evalúan N_g generaciones.

En este trabajo se utilizan los operadores genéticos *PMX* (*partially mapped crossover*) propuesto por Goldberg y Lingle en 1985 [15] como operador de cruzamiento y *swap* (o *exchange*) como operador de mutación. Estos operadores son operadores clásicos y frecuentemente utilizados en la literatura en problemas de secuenciación.

El operador *PMX* selecciona aleatoriamente una subsecuencia central de igual longitud en ambos padres, intercambiándola en la reproducción de dos descendientes (hijos), generando de esta forma un mapeo entre los trabajos de ambas subsecuencias. Las posiciones restantes de los hijos se completan con los trabajos aún no asignados en la misma posición del padre que no aportó la subsecuencia central, esto es, si el trabajo no se encuentra en la subsecuencia central del hijo permanece en la misma posición que tiene en el padre, en caso contrario, se reemplaza por el trabajo que está en la misma posición de la subsecuencia central traspasada al descendiente del otro padre. El operador *swap* intercambia dos trabajos de un individuo en forma aleatoria

La Figura 4 muestra el cruzamiento de dos individuos (padres) aplicando el operador de cruzamiento PMX, seleccionando en forma aleatoria la subsecuencia que incluye las posiciones 3 a 5 (ambas inclusive) en ambos padres en un problema de $n = 6$ trabajos (fila 1 de la Figura 4), las que son intercambiadas para generar dos hijos (fila 2 de la Figura 4). Así se establece el correspondiente mapeo entre el padre 1 y el hijo 1 (como también entre el padre 2 y el hijo 2) visualizado en las filas 1 y 2 de la Figura 4 mediante flechas de dos sentidos. Las restantes posiciones del hijo 1 (hijo 2) se completan con el correspondiente elemento de la misma posición en el padre 1 (padre 2), siempre que este elemento no pertenezca a la subsecuencia ya traspasada al hijo 1 (hijo 2) desde el padre 2 (padre 1). En el ejemplo de la Figura 4 el padre 1 traspasa el trabajo 4 en la misma posición al hijo 1, pero no los trabajos 5 y 2 ya que estos últimos están presentes en la subsecuencia traspasada al hijo 1; en forma análoga se explica el traspaso del trabajo 4 del padre 2 al hijo 2 (ver fila 3 de la Figura 4).

La evaluación de los algoritmos se realizó utilizando instancias generadas en forma aleatoria de acuerdo a distribuciones de tiempos de proceso y de *setup* utilizadas en la literatura [16]. Los tiempos de proceso de los trabajos se generaron de la distribución uniforme discreta entre 1 y 100 ($p_{ik} \sim UD[1,100]$). Los tiempos de preparación dependientes de la secuencia de los trabajos se generaron de la distribución uniforme discreta entre 0 y 9 ($s_{ij} \sim UD[0,9]$), esto es, se consideran tiempos de preparación para un entorno productivo donde éstos son esencialmente menores a los tiempos de proceso.

Los *due dates* de los trabajos se definieron en base a una adaptación del método propuesto en [16]:

$$d_i = \sum_{k=1}^{m} p_{ik} + \sum_{k=1}^{m} se_{ik} + (n-1)\cdot \overline{p}_i \cdot u \quad \text{con}$$

$$se_{ik} = \sum_{j=1}^{n} s_{jik} \Big/ \text{n} \quad \text{y} \quad \overline{p}_i = \sum_{k=1}^{m} p_{ik} \Big/ \text{m}$$

La definición del *due date* para un trabajo considera su tiempo total de proceso, una estimación del tiempo total de *setup* a través de la suma de los tiempos promedio de *setup* que puede tener un

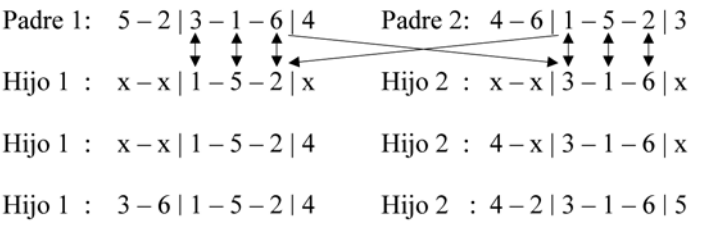

Figura 4. Operador de cruzamiento PMX.

trabajo en cada etapa, y una holgura que depende de la cantidad de trabajos en el sistema, el tiempo promedio de proceso por etapa del trabajo y de una componente aleatoria *u* como observación de una variable aleatoria U ~ U[0,1].

Se generaron 30 instancias de problemas de cinco etapas ($m = 5$) y 30 trabajos ($n = 30$). En cada etapa se consideró un número de máquinas generadas en forma aleatoria a partir de una distribución uniforme discreta entre 1 y 5, así el número de máquinas por etapa en cada instancia, como también la configuración de máquinas entre instancias, pueden ser diferentes.

Para evaluar el desempeño de los algoritmos se utilizó la tardanza total T como medida de desempeño (ver definición en la sección *Definición del Problema*). La tardanza entregada por cada método se compara en forma relativa respecto del intervalo entre el valor de la mínima tardanza (T_{min}) y de la máxima tardanza (T_{max}) observada en una instancia a través del indicador de diferencia relativa porcentual:

$$\%Dif = \frac{T^{Método} - T_{min}}{T_{max} - T_{min}} * 100$$

donde $T^{Método}$ es la tardanza obtenida por un método en una determinada instancia. De esta forma se obtiene una medida de la calidad relativa de las soluciones entregadas por los diferentes métodos.

Para la simulación Montecarlo se consideró una muestra de tamaño 100.000, el que se determinó como *trade off* entre el valor de tardanza y tiempo computacional.

Los parámetros de probabilidad de cruzamiento y de mutación y el tamaño de población del algoritmo genético (AG) se calibraron mediante experimentación. En un estudio preliminar se analizaron 10 instancias (distintas a las consideradas en el trabajo) con valores $p_c = 0.7$, 0.8 y 0.9; $p_m = 0.01$, 0.05 y 0.1; $N_p = 80$, 100 y 120, procesando 10 réplicas de 1.000 generaciones por instancia. Se observó la tardanza promedio para cada combinación de valores, obteniéndose el menor promedio para los valores $p_c = 0.9$, $p_m = 0.1$ y $N_p = 100$. Finalmente el número de generaciones se estableció en 700 para posibilitar una búsqueda exhaustiva. Se procesaron 30 réplicas para cada instancia, entregando como resultado del método la secuencia con menor tardanza.

La Tabla 1 resume la diferencia relativa porcentual promedio respecto de la mejor solución obtenida por los métodos, el porcentaje y el número de veces en que los métodos obtienen la mejor solución.

La Figura 5 presenta la diferencia relativa porcentual (eje vertical) de las soluciones para las 30 instancias consideradas (eje horizontal) comparando los métodos EDD y Slack, los métodos de mejor rendimiento promedio de acuerdo a lo indicado en la Tabla 1. El mejor rendimiento promedio de EDD respecto de Slack se refleja al observar que en la Figura 5 la curva de EDD tiende a estar bajo la curva de Slack. Análogamente, las Figuras 6 a 9 comparan el método EDD con AG, ATCS, SM y CR respectivamente, donde esta tendencia es más marcada aún.

La Tabla 1 cuantifica la apreciación gráfica respecto de la capacidad de los métodos para obtener buenas soluciones. La diferencia relativa porcentual promedio de la heurística EDD (4,74%) y Slack (7,84%), los más bajos entre todos los métodos, significa que en promedio las soluciones de estas dos heurísticas se acercan más a la mejor solución obtenida por los métodos analizados. En particular, EDD y Slack obtienen respectivamente en el 56,67% (17 sobre 30) y 53,33% (16 sobre 30) de los casos la mejor solución, y dado que sólo en seis instancias obtienen ambas la mejor solución, significa que entre ellas obtienen en el 90% de los casos (27 sobre 30) la mejor solución.

Cabe destacar que, si bien ATCS obtiene sólo en el 16,67% (5 sobre 30) de los casos la mejor solución, en tres de éstos es el único método que obtiene la mejor solución (comparar Figura 7 con las restantes figuras). De lo anterior se observa, por lo tanto, que entre EDD, Slack y ATCS obtienen el 100% de las mejores soluciones en la muestra de problemas analizada. Este resultado contrasta con la heurística AG, que teniendo un significativo mejor rendimiento promedio que ATCS (36,81% versus 53,75%), obtiene sólo en el 10% de los casos (3 sobre 30) la mejor solución, y en estos casos no es la única heurística que obtiene la mejor solución (ver Figura 6).

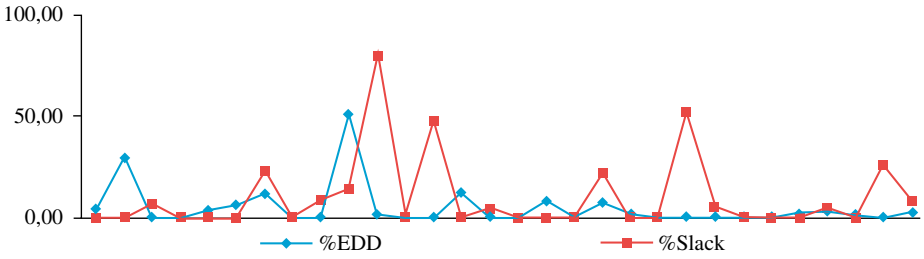

Figura 5. Diferencia relativa porcentual para EDD y Slack.

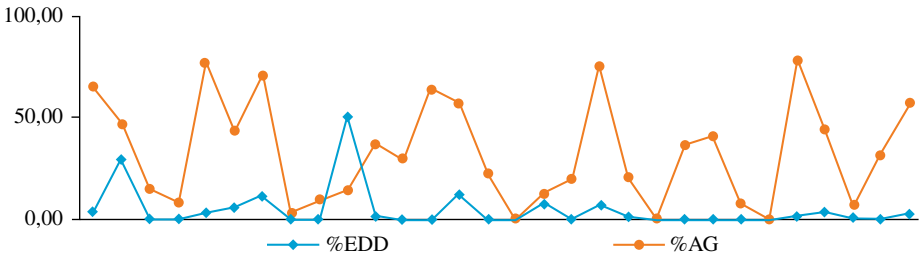

Figura 6. Diferencia relativa porcentual para EDD y AG.

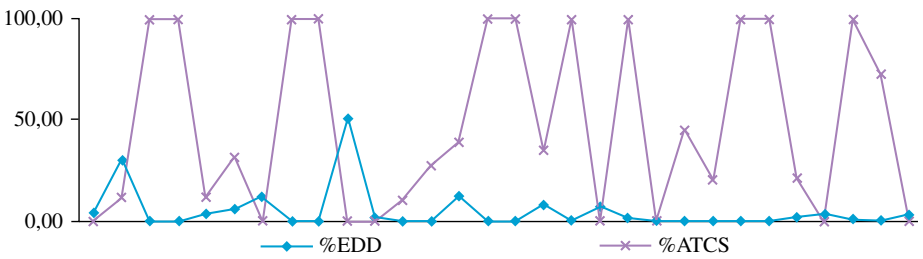

Figura 7. Diferencia relativa porcentual para EDD y ATCS.

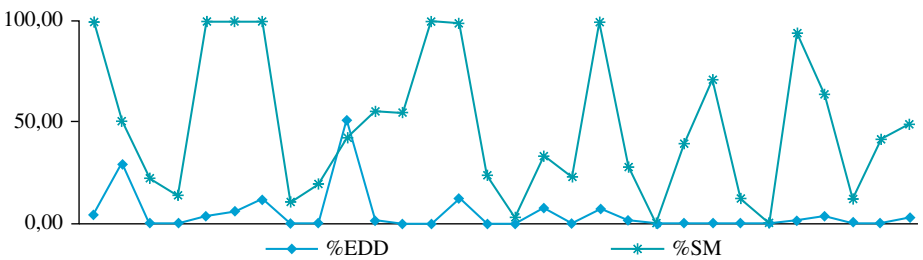

Figura 8. Diferencia relativa porcentual para EDD y SM.

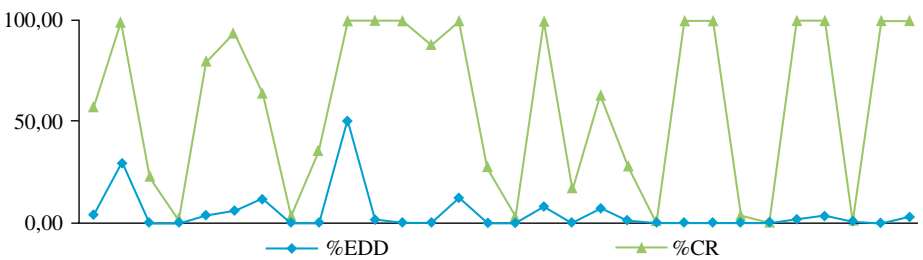

Figura 9. Diferencia relativa porcentual para EDD y CR.

Hay dos explicaciones para el bajo rendimiento de AG, el primero es que EDD y Slack, aunque simples, son métodos que hacen uso de información relevante al decidir la asignación de los trabajos en el procedimiento *AsignaTrabajos* cuando se construye el programa de producción, los *due dates*; en segundo lugar, AG, aunque es una metaheurística, es una versión básica de algoritmo genético que no incorpora conocimiento del problema en la definición de sus componentes actuando sólo como un sistema explorador del espacio de soluciones.

Con el objetivo de introducir una mejora se aplicó una búsqueda en vecindad IP a la solución obtenida por cada método. Se observó similar comportamiento relativo entre los métodos que para el caso sin la aplicación de la búsqueda en vecindad.

La Tabla 2 resume la diferencia relativa porcentual promedio respecto de la mejor solución para cada instancia, el porcentaje de veces en que los métodos obtienen la mejor solución y el porcentaje en que mejoran las soluciones aplicando la búsqueda en vecindad.

Los resultados que muestra la Tabla 2 resumen la capacidad media de los métodos para obtener buenas soluciones al aplicar una búsqueda en vecindad, es decir, si tienen capacidad de generar buenas soluciones iniciales (semillas) para métodos de busca en vecindad. En este caso la heurística EDD obtiene en el 56,67% de los casos (17 sobre 30) la mejor solución, mientras que Slack lo hace en el 53,33 de los casos (16 sobre 30). Dado que en este caso ambas obtienen en ocho instancias la mejor solución, en conjunto obtienen en el 83,33% de las instancias (25 sobre 30) la mejor solución; de las cinco instancias restantes AG (ATCS) obtiene en cuatro (1) de ellas como único método la mejor solución.

De la Tabla 2 se observa también la significativa mejora en la solución que produce la aplicación de la búsqueda en vecindad IP a la solución obtenida por todos los métodos, expresada como el promedio de la disminución porcentual observada en la tardanza total respecto de la tardanza total sin aplicar la búsqueda en vecindad calculada en aquellos problemas donde es posible mejorar (con tardanza positiva):

$$\%MejoraBV = \frac{T^{Método} - T_{BV}^{Método}}{T^{Método}} * 100$$

donde $T^{Método} > 0$ y $T_{BV}^{Método}$ corresponden a la tardanza total obtenida por un método en una determinada instancia sin y con aplicar búsqueda en vecindad a la solución obtenida por el método en la respectiva instancia.

El procesamiento se realizó en un computador Intel Core 2 Duo T7500 de 2.2 GHz de 2 GB de RAM, promedio de rutinas adaptadas del software SPS_Optimizer [17], herramienta diseñada para la programación de operaciones. El orden de magnitud del tiempo CPU para la ejecución de los algoritmos se presenta en la Tabla 3.

Tabla 1. Comparación de algoritmos.

Algoritmo	**EDD**	**Slack**	**AG**	**ATCS**	**SM**	**CR**
%Dif. Promedio	4,74	7,84	36,81	53,75	54,02	64,56
%Mejor Solución	56,67	53,33	10,00	16,67	6,67	6,67
nMejor Solución	17	16	3	5	2	2

Tabla 2. Comparación de algoritmos con búsqueda en vecindad.

Algoritmo	**EDD**	**Slack**	**AG**	**ATCS**	**SM**	**CR**
%Dif. Promedio	12,92	8,39	26,85	51,07	78,78	47,49
%Mejor Solución	56,67	53,33	26,67	16,67	6,67	13,33
nMejor Solución	17	16	8	5	2	4
%MejoraBV Promedio	30,38	26,88	45,78	42,68	27,4	47,35

Tabla 3. Tiempo CPU [s] de algoritmos.

Alg.	**EDD**	**Slack**	**AG**	**ATCS**	**SM**	**CR**
CPU	0,00	0,00	400,00	0,00	12,00	0,00

El tiempo CPU del método AG corresponde a la ejecución de 30 réplicas con 13,5 s/réplica. El tiempo CPU para la heurística de búsqueda en vecindad IP no superó los 0,20 s.

CONCLUSIONES

Se ha evaluado el comportamiento de un algoritmo genético estándar, comparándolo con la extensión de heurísticas como EDD (*earliest due date*), *Slack* (holgura), CR (*critical ratio*), ATCS (*apparent tardiness cost with setup*) y simulación Montecarlo (muestreo aleatorio). Posteriormente se aplicó a todos los algoritmos un método de búsqueda en vecindad IP (*pairwise interchange*) para mejorar la solución obtenida.

Si bien existen muchos operadores de cruzamiento y de mutación [18], en este trabajo se han utilizado los operadores clásicos *PMX (partially mapped crossover)*, posiblemente uno de los operadores de cruzamiento más utilizados [5], y el operador de mutación *swap (interchange)* para la minimización de la tardanza en un *flexible flowshop*, con tiempos de preparación anticipatorios dependientes de la secuencia. Por otro lado, también otras representaciones del cromosoma han sido utilizadas [7, 19]; en este trabajo se utilizó la representación de permutación frecuentemente utilizada para definir el cromosoma en problemas de secuenciación [5].

La experimentación mostró que las heurísticas EDD y *Slack* obtienen los mejores resultados tanto en términos de la diferencia porcentual respecto de la mejor solución, como en términos del porcentaje de veces que obtienen la mejor solución. Este resultado también se observa al aplicar la búsqueda en vecindad IP como heurística de mejora a las soluciones entregadas por los diferentes métodos.

Entre los métodos analizados las heurísticas EDD y Slack mostraron mejor rendimiento, mientras que la heurística AG muestra un rendimiento medio. A pesar de que AG muestra un rendimiento promedio significativamente mejor que ATCS, es esta última la que encuentra la mejor solución en instancias donde ni EDD ni Slack las obtienen. ATCS, CR y SM muestran un rendimiento promedio bajo; sin embargo, ATCS encontró en casos puntuales la mejor solución como único método, lo que nunca ocurrió con CR y SM. CR y SM tampoco reflejaron capacidad para obtener buenas soluciones ni para generar buenas soluciones iniciales al aplicar un método de búsqueda en vecindad. El bajo rendimiento de un método como SM era esperable, dado que se basa en un proceso de muestreo aleatorio que no incorpora criterios de optimización, sirviendo sólo como método de comparación.

Hay dos explicaciones para el bajo rendimiento de AG, primero, EDD y Slack, aunque simples, son métodos que hacen uso de información relevante (los *due dates*) al decidir la priorización de los trabajos, y segundo, AG es una versión básica de algoritmo genético que no incorpora conocimiento del problema en la definición de sus componentes actuando sólo como un sistema explorador del espacio de soluciones, pero que puede ser mejorado en trabajos futuros incorporando mecanismos que hagan uso de información del problema, en particular en la forma de generar la población inicial [5, 6, 20].

Independiente de que pueden explorarse otros métodos de resolución para el problema estudiado en este trabajo (en particular una mejora de AG), el uso conjunto de EDD, Slack y ATCS (esta última por su capacidad de encontrar buenas soluciones en instancias donde ni EDD ni Slack lo hacen) entrega una buena alternativa de solución para instancias reales, primero por su simplicidad de implementación y segundo, por sus tiempos computacionales no significativos.

Se destaca el significativo efecto de la aplicación de una búsqueda en vecindad IP a las soluciones entregadas por los métodos, logrando mejoras significativas con un esfuerzo computacional que no superó los 0.2 segundos. La búsqueda en vecindad IP produjo mejoras significativas en las soluciones entregadas por los métodos, en particular reforzó el mejor rendimiento de las heurísticas EDD y Slack, utilizando tiempos computacionales no significativos para el tamaño de los problemas analizados, y por otro lado mejoró el rendimiento de AG pasando a ser mejor complemento de EDD y Slack para encontrar buenas soluciones.

REFERENCIAS

[1] A. Allahverdi, C.T. Ng, T.C.E. Cheng and M. Kovalyov. "A survey of scheduling problems with setup times or costs". European Journal of Operational Research. Vol. 187, Issue 3, pp. 985-1032. 2008.

[2] J. Hoogeveen, J. Lenstra and B. Veltman. "Preemptive scheduling in a two-stage multiprocessor flow shop is NP-Hard". European Journal of Operational Research. Vol. 89, Issue 1, pp. 172-175. 1996.

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

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

[5] 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, Issue 2, pp. 358-378. 2009.

[6] R. Ruiz and C. Maroto. "A genetic algorithm for hybrid flowshops with sequence dependent setup times and machine eligibility". European Journal of Operational Research. Vol. 169, Issue 3, pp 781-800. 2006.

[7] S. Bertel and J. Billaut. "A genetic algorithm for an industrial multiprocessor flow shop scheduling problem with recirculation". European Journal of Operational Research. Vol. 159, Issue 3, pp. 651-662. 2004.

[8] H.T. Lin and C.J. Liao. "A case of study in a two-stage Flexible flow shop with setup time and dedicated machines". International Journal of Production Economics. Vol. 86, Issue 2, pp. 133-143. 2003.

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

[10] V. Botta-Genoulaz. "Hybrid flow shop scheduling with precedence constrains and time lags to minimize maximum lateness". International Journal of Production Economics. Vol. 64, Issue 1-3, pp. 101-111. 2000.

[11] M.B. Fakhrzad and M. Heydari. "A heuristic algorithm for Flexible flow shop production scheduling to minimize the sum of the earliness and tardiness costs". Journal of the Chinese Institute of Industrial Engineers. Vol. 25, Issue 2, pp. 105-115. 2008.

[12] 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 and Operations Research. Vol. 36, Issue 2, pp. 358-378. 2009.

[13] Y.H. Lee, Bhaskaran and M. Pinedo. "A heuristic to minimize the total weighted tardiness with sequence-dependent setups". IIE Transactions. Vol. 29, Issue 1, pp. 45-52. 1997.

[14] Y.H. Lee and M. Pinedo. "Scheduling jobs on parallel machines with sequence dependent setup times". European Journal of Operational Research. Vol. 100, Issue 3, pp. 464-474. 1997.

[15] Z. Michalewicz. "Genetic Algorithms+Data Structures=Evolution Programs". Third Edition. Springer. 1999.

[16] 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.

[17] E. Salazar. "Programación de Sistemas de Producción con SPS_Optimizer". Revista ICHIO. Vol. 1 Nº 2, pp. 33-46. ISSN 718–9605 (Digital). 2010.

[18] R. Ruiz and C. Maroto. "A genetic algorithm for hybrid flowshops with sequence dependent setup times and machine eligibility". European Journal of Operational Research. Vol. 169, Issue 3, pp. 781-800. 2006.

[19] D.C. Mattfeld and Ch. Bierwirth. "An efficient genetic algorithm for job shop scheduling with tardiness objectives". European Journal of Operational Research. Vol. 155, Issue 3, pp. 616-630. 2004.

[20] F. Sivrikaya Serifoglu and U. Ulusoy. "Multiprocessor task scheduling in multistage hybrid flow-shops: a genetic algorithm approach". Journal of the Operational Research Society. Vol. 55, Issue 5, pp. 504-512. 2004.