ISSN 0717-9103
ISSN Online 0718-8307
Ingeniería Industrial-Año. 10 Nº1: 5-18, 2011

ALGORÍTMO DE BÚSQUEDA DE ENTORNO VARIABLE PARA MINIMIZAR LA TARDANZA TOTAL PONDERADA EN UNA MÁQUINA DE PROCESAMIENTO POR LOTES

A VARIABLE NEIGHBORHOOD SEARCH ALGORITHM TO MINIMIZE THE TOTAL WEIGHTED TARDINESS ON A BATCH PROCESSING MACHINE

Mario César Vélez Gallego[1] , Juan Diego López Jiménez[1]
[1]PDepartamento de Ingeniería de Producción, Universidad EAFIT, Medellín, Colombia

RESUMEN

En este artículo se presentan los resultados de una investigación desarrollada para resolver el problema de programación de producción, en un sistema en el cual el objetivo principal es cumplir con las fechas de entrega. Cada trabajo está descrito por su tiempo de proceso, su fecha de liberación, su fecha de entrega, su importancia relativa con respecto a los otros trabajos y su peso. La máquina de procesamiento por lotes (MPL) puede procesar múltiples trabajos simultáneamente, siempre y cuando el peso total de éstos no exceda la capacidad máxima de la máquina. El tiempo de proceso del lote es el tiempo máximo de proceso de los trabajos que lo componen. De la misma forma, el tiempo de liberación de un lote es el tiempo máximo de liberación de los trabajos que lo componen. Teniendo en cuenta que el problema es NP–Completo, en este artículo se propone un heurístico de búsqueda de entorno variable para minimizar la tardanza total ponderada en una MPL. Los experimentos computacionales realizados para establecer la calidad de las soluciones encontradas demostraron que, en un tiempo de cómputo restringido a un máximo de 30 minutos, el heurístico propuesto encuentra soluciones considerablemente mejores que las encontradas mediante la implementación de un modelo de programación entera mixta, disponible en la literatura e implementado en un software comercial de optimización.

Palabras clave: Heurística, programación de producción, máquina de procesamiento por lotes, búsqueda de entorno variable.

ABSTRACT

This paper presents the results of a research project conducted to solve a production scheduling problem observed in a system in which meeting customer due dates is the primary objective. Each job to be scheduled is defined by its processing time, ready time, due date, weight and size. The Batch Processing Machine (BPM) can process several jobs simultaneously as a batch as long as its capacity is not violated. The processing time of a batch is the largest processing time among the jobs in the batch, and the batch ready time is the largest ready time among the jobs in the batch. Given that the problem is NP-hard we propose a Variable Neighborhood Search (VNS) heuristic to minimize the total weighted tardiness on a single BPM. The computational experiments conducted to assess the quality of the solutions found show that in a computational time restricted to a maximum of 30 minutes, the proposed heuristic finds solutions considerably better than the solutions obtained after implementing a mixed integer programming model available in the literature in a commercial solver.

Keywords: Heuristic, production scheduling, batch processing machine, variable neighborhood search

Autor para correspondencia: marvelez@eafit.edu.co
Recibido: 31.05.2011 Aceptado: 11.08.2011

INTRODUCCIÓN

Las máquinas de procesamiento por lotes (MPL) se caracterizan porque en ellas es posible procesar múltiples trabajos simultáneamente. Las MPL son comunes en procesos de recubrimiento electrolítico, tratamientos térmicos, hornos de secado y procesos de pintura, entre otros. Este artículo está motivado en el caso particular de un proceso de recubrimiento electrolítico (e.g. estañado), en el cual un conjunto de piezas de hierro fundido se sumergen en un tanque para recibir un recubrimiento de estaño por electrodeposición. Todas las piezas que conforman un lote, entran y salen del tanque al mismo tiempo. Debido a que en este proceso se recubren con estaño piezas de diversos tamaños y formas, cada pieza tiene especificado un tiempo mínimo de permanencia en el tanque, suficiente para que sobre su superficie se deposite la cantidad de estaño especificada por el cliente. Si una pieza permanece en el tanque un tiempo mayor al mínimo especificado, al final del proceso tendrá una cantidad de estaño mayor a la requerida, pero no presentará detrimento alguno en su calidad. Por el contrario, si una pieza permanece menos tiempo del especificado, se considera fuera de especificación, ya que la capa de estaño depositada tendrá un espesor inferior al necesario. Por esta razón, el tiempo de permanencia de un lote de piezas en el tanque de estañado lo determina la pieza que requiera mayor tiempo de permanencia.

En el sistema de producción que originó este proyecto, las piezas de hierro llegan durante el día a un sitio de almacenamiento temporal frente al tanque de estañado, donde esperan para ser agrupadas en lotes y procesadas. Cada pieza tiene estipulada una fecha de entrega previamente pactada con el cliente, y un valor de importancia relativa o prioridad, determinado por el programador de producción. El instante de llegada de una pieza a este sitio de almacenamiento temporal se denomina tiempo de liberación, y está determinado por la finalización del proceso anterior. Por este motivo, la fecha de liberación de un lote lo determina la fecha de liberación de la última pieza en llegar; esto es, por la mayor fecha de liberación de las piezas que conforman el lote. Adicionalmente, el tanque de estañado tiene una capacidad máxima dada en kilogramos, lo que obliga a que el peso total de los lotes formados no pueda sobrepasar esta capacidad. Al ser el estañado el proceso que más tiempo consume dentro del sistema de producción, su programación tiene un impacto directo en el nivel de servicio al cliente, que para este caso se mide en términos del cumplimiento de las fechas de entrega pactadas con él. El objetivo de este artículo consiste en proponer un algoritmo de programación de producción para el tanque de estañado, de manera que se minimicen los retrasos en las entregas a los clientes.

Más formalmente, el problema de programación de producción, objeto de este artículo, consiste en desarrollar un algoritmo para programar un conjunto J de n trabajos (piezas) en una MPL con capacidad S de tal manera que la tardanza total ponderada (TTP) sea mínima. Cada trabajo $j \in J$ está descrito por un tiempo mínimo de proceso p_j, un tiempo de liberación r_j, una fecha de entrega d_j, un tamaño q_j y un valor de importancia relativa respecto a los otros trabajos w_j. De acuerdo con la terminología tradicionalmente usada en la literatura (Pinedo, 2008), la tardanza del trabajo j (T_j) se define como $T_j=max\{0, C_j - d_j\}$ donde C_j es tiempo de terminación del trabajo j; esto es, la magnitud de la diferencia entre la fecha de entrega pactada con el cliente y la fecha efectiva de terminación del trabajo cuando esta diferencia es positiva. La tardanza ponderada del trabajo j es igual a la tardanza de j multiplicada por la importancia relativa de j (w_jT_j); y la TTP para un conjunto de trabajos (i.e. J) es la suma de las tardanzas ponderadas de éstos ($\Sigma_j w_j T_j$). Para ilustrar mejor el problema se presenta el siguiente ejemplo. La tabla 1, contiene los datos correspondientes a cinco trabajos que deben ser programados en una MPL con capacidad igual a 40. Para este ejemplo la figura 1 presenta un diagrama de Gantt con una solución arbitraria del problema conformada por tres lotes con una TTP igual a 68. Cada rectángulo en el diagrama representa un lote y los números dentro del rectángulo representan los trabajos que lo componen. La longitud y ubicación de cada rectángulo representan respectivamente el

ISSN 0717-9103
ISSN Online 0718-8307

tiempo de procesamiento y el tiempo de liberación del lote.

Tabla 1. Ejemplo de una instancia de 5 trabajos

j	p_j	r_j	d_j	q_j	w_j
1	9	12	30	16	2
2	14	7	28	25	5
3	5	4	25	19	1
4	11	9	32	8	4
5	12	2	26	13	2

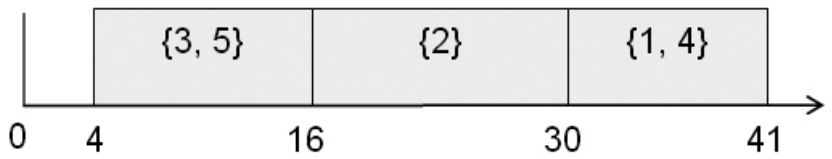

Figura 1. Diagrama de Gantt de una solución arbitraria para el ejemplo de la tabla 1

ESTADO DEL ARTE

Dependiendo de la forma en que se calcule el tiempo de proceso de un lote, se encuentran en la literatura dos tipos de máquinas de procesamiento por lotes: máquinas en paralelo y máquinas en serie. Una máquina de lotes en paralelo, también conocida como *p–batch,* es aquella en la cual el tiempo de proceso del lote está determinado por el tiempo máximo de proceso de las partes que conforman el lote, mientras que una máquina de lotes en serie o *s–batch* es aquella en la cual el tiempo de proceso del lote es igual a la suma de los tiempos de proceso de las partes que conforman el lote (Brucker, 2007). Según la notación clásica de programación de producción introducida en (Graham, *et al.,* 1979) y extendida en (Brucker, 2007) para el caso de máquinas de procesamiento por lotes, el problema en estudio se denota como $1|p\text{–}batch, r_j, q_j, |\Sigma_j w_j T_j$.

La referencia (Brucker, *et al.,* 1998) es el primer trabajo de que se tenga noticia que aborda un problema similar. En este trabajo se aborda el problema bajo el supuesto de que todos los trabajos tienen la misma fecha de liberación y el mismo tamaño. Para este problema simplificado, los autores proporcionan resultados teóricos sobre la complejidad computacional para algunos casos particulares, y proponen una formulación de programación dinámica para resolver el problema general. Para el mismo problema (Pérez *et al.,* 2005) proponen una estrategia de dos fases, en la cual se forman los lotes en la primera fase, para luego programarlos en la máquina en la segunda fase. En este trabajo se proponen dos heurísticos para la primera fase y cuatro para la segunda. En las referencias (Kurz & Mason, 2008; Tangudu & Kurz, 2006), los autores consideran el problema bajo el supuesto de fechas de liberación diferentes de cero y familias de trabajos incompatibles; esto es, que cada trabajo pertenece a una familia y todos en un lote deben pertenecer a la misma familia. En (Tangudu & Kurz, 2006) se propone un algoritmo de ramificación y acotamiento (i.e *branch & bound*), y se resuelven óptimamente instancias de hasta 32 trabajos, mientras que en (Kurz & Mason, 2008) se propone un método de solución

de dos fases, similar al propuesto en (Perez *et al*., 2005) acompañado de un heurístico de búsqueda local. Finalmente, en (Mathirajan *et al*., 2010) se aborda el problema sin considerar familias de productos, bajo los supuestos de tiempos de liberación y tamaños arbitrarios de los trabajos. Para este problema, los autores lo formulan como un modelo de programación entera mixta, y proponen cuatro heurísticos constructivos y un heurístico de recocido simulado para su solución. En este trabajo, los autores evalúan instancias con 10, 12, 20, 50 y 100 trabajos, y los resultados de estos métodos de solución son evaluados y comparados a través de un estimador estadístico de la solución óptima, obtenido a partir de un procedimiento propuesto en (Rardin & Uzsoy, 2001).

Hasta donde se pudo constatar en la revisión de la literatura, la única referencia que se refiere al mismo problema que se aborda en este artículo es (Mathirajan *et al*., 2010). Desafortunadamente, una comparación entre el algoritmo propuesto en el presente artículo y el método de solución propuesto en la citada referencia no es posible, debido a que (1) el conjunto de instancias de prueba utilizadas en el mencionado trabajo no está disponible, (2) los estimadores por punto de las soluciones óptimas no se proporcionan en la referencia, y (3) estos estimadores fueron construidos usando soluciones generadas aleatoriamente y, por lo tanto, son variables aleatorias.

SOLUCIÓN PROPUESTA

La búsqueda en entorno variable (BEV) es un heurístico de mejoramiento introducido en (Mladenovic & Hansen, 1997), en el cual se explota el hecho de que un óptimo local con respecto a un determinado vecindario puede no ser un óptimo local para otro vecindario. Basados en esta observación, la BEV cambia de vecindario cuando durante la exploración de éste el algoritmo se queda atrapado en un óptimo local, buscando que bajo el nuevo vecindario la solución actual no sea un óptimo local y el proceso de mejoramiento de la solución pueda continuar. Dependiendo de la forma en que se tome la decisión de pasar de un vecindario a otro, se configuran tres variantes básicas de la BEV: La BEV Descendiente (BEVD), la BEV Reducida (BEVR) y la BEV Básica (BEVB). Las tres variantes parten del conjunto $N=\{N_1, N_2, \ldots, N_K\}$ de K vecindarios seleccionados *a priori*. En la variante BEVD los vecindarios se exploran de manera determinista (i.e. sin aleatoriedad), a través de un procedimiento de búsqueda local que puede ser de dos tipos: *First Improve* (FI) o *Best Improve* (BI). En una búsqueda local FI, el vecindario se explora hasta que se encuentra la primera solución vecina mejor que la solución inicial. Una vez encontrada ésta, se adopta como la nueva mejor solución y el proceso de exploración del vecindario se inicia nuevamente. En una búsqueda BI, se adopta como nueva solución la mejor solución encontrada luego de explorar exhaustivamente el vecindario. La decisión de cambiar de un vecindario al siguiente bajo la variante BEVD ocurre cuando el procedimiento de búsqueda local queda atrapado en un óptimo local. El algoritmo termina cuando explora los K vecindarios consecutivamente, sin mejorar la solución. En la variante BEVR, en vez de explorar exhaustivamente los vecindarios, éstos se muestrean aleatoriamente. El criterio de parada en este caso puede variar, ya sea porque el algoritmo evalúe una cantidad de soluciones vecinas sin mejorar la solución actual, porque el algoritmo alcance un tiempo de ejecución máximo, o por una combinación de las anteriores. Por último, la variante BEVB es similar a la variante BEVR, en la cual se aplica un procedimiento de búsqueda local cada vez que se muestrea aleatoriamente una solución vecina a la solución actual. Una explicación detallada del heurístico BEV y sus diferentes variantes puede encontrarse en las referencias (Hansen *et al*., 2009; Hansen *et al*., 2010), mientras que algunas aplicaciones recientes de BEV aplicadas a resolver problemas de programación de producción pueden encontrarse (Amiri *et al.,* 2010; Liao & Cheng, 2007; Roshanaei *et al.,* 2009; Sevkli & Aydin, 2006; Zandieh & Adibi, 2010).

El método de solución propuesto para resolver el problema en estudio es un heurístico BEVD,

que parte de una solución inicial para luego mejorarla por medio de la exploración sistemática de múltiples vecindarios. Al finalizar el proceso de búsqueda el algoritmo reporta la mejor solución encontrada y el tiempo que tardó en encontrarla. Para mejorar el desempeño del algoritmo se propone ejecutarlo múltiples veces, cada una con una solución inicial diferente, y almacenar la mejor solución encontrada por el algoritmo durante las múltiples ejecuciones. El resto de esta sección está estructurada de la siguiente manera: primero se describe la forma como se construyen las soluciones iniciales, luego se describen los diferentes vecindarios que explora el heurístico y, por último, se describe el algoritmo propuesto.

Soluciones Iniciales

Para la construcción de las soluciones iniciales se usaron las siguientes reglas de despacho:

1. *First–In–First–Out* (FIFO): Los trabajos se ordenan en el orden en que llegan al sistema.
2. *Earliest Due Date (EDD):* Los trabajos se ordenan de acuerdo con su fecha de entrega (de menor a mayor).
3. *Earliest Ready Time (ERT):* Los trabajos se ordenan de acuerdo con su fecha de liberación (de menor a mayor).
4. *Weighted Shortest Processing Time (WSPT):* Para cada trabajo $j \in J$ se calcula el índice w_j / p_j y se ordenan de acuerdo con éste índice de mayor a menor.
5. *Weighted Longest Processing Time (WLPT):* Para cada trabajo $j \in J$ se calcula el índice $w_j \times p_j$ y se ordenan de acuerdo con éste índice de mayor a menor.
6. *Largest Job Size (LJS):* Los trabajos se ordenan de acuerdo con su tamaño (de mayor a menor).
7. *Smallest Job Size (SJS)* Los trabajos se ordenan de acuerdo con su tamaño (de menor a mayor).
8. *Apparent Tardiness Cost* (ATC): Para cada trabajo $j \in J$ se calcula el índice $I_j(t)$ que se calcula como en la ecuación (1) y se programa primero el trabajo con el mayor índice.

El índice es función del instante de tiempo *t* en el cual la máquina se encuentra disponible. Bajo esta regla los trabajos se programan uno a la vez, calculando para cada caso el índice $I_j(t)$ se con el valor de *t* actualizado.

$$I_j(t) = \frac{w_j}{p_j} \exp\left(-\frac{\max(d_j - p_j - t, 0)}{K\bar{p}}\right) \quad (1)$$

Una descripción detallada de estas reglas puede encontrarse en las referencias (Mathirajan *et al.,* 2010; Pinedo, 2008). Cada regla proporciona un criterio para ordenar los trabajos, con los cuales, una vez ordenados, se forman lotes por medio del heurístico conocido como *Online First Fit*. El algoritmo utilizado para construir las soluciones iniciales se ilustra a continuación.

Pseudocódigo para la generación de soluciones iniciales

```
Inicio
    Ordenar los trabajos según la regla de despacho seleccionada
    Abrir un lote vacío
    Mientras falten trabajos por asignar
        Si el lote abierto tiene capacidad suficiente entonces
            Asignar al lote abierto el primer trabajo en la lista
            Eliminar el primer trabajo de la lista
        Si no
            Cerrar el lote abierto y abrir un nuevo lote vacío
        Fin Si
    Fin Mientras
Fin
```

Vecindarios Propuestos

Para este artículo se definen dos clases de vecindarios: los simples y los compuestos. Un vecindario simple es aquel en el cual las soluciones vecinas se alcanzan por medio de un solo movimiento efectuado sobre la solución original, mientras que un vecindario compuesto es aquel en el cual la solución vecina se obtiene tras efectuar sobre la solución original dos o más movimientos consecutivos. A continuación se describen los movimientos que dan origen a los vecindarios simples y compuestos usados en este artículo.

Movimiento de Inserción Simple (MIS)

Sea *X* una solución factible al problema en estudio. El MIS consiste en extraer un trabajo de un lote en X e insertarlo en un lote diferente, siempre y cuando con este movimiento no se viole la capacidad de la máquina. En consecuencia, el vecindario de Inserción Simple (IS) está conformado por el conjunto de soluciones factibles que se pueden construir tras efectuar un MIS en X. En la figura 2 se ilustra, para el ejemplo de la tabla 1, un movimiento de inserción simple en el cual el trabajo 4, asignado al tercer lote en la secuencia, es sacado de este lote e insertado en el segundo lote de la secuencia. Como se puede apreciar en este ejemplo, el tiempo de terminación del tercer lote cambia de 41 a 39 como consecuencia de esta inserción.

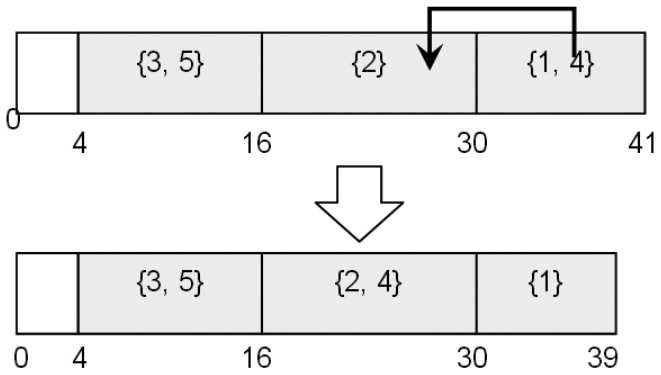

Figura 2. Ejemplo de un movimiento de inserción simple (IS)

Movimiento de Intercambio Simple (MXS)
Sea *X* una solución factible al problema en estudio. El MXS consiste en tomar dos trabajos asignados a dos lotes diferentes en X e intercambiarlos de lote, siempre y cuando con este movimiento no se viole la capacidad de la máquina. El vecindario de Intercambio Simple (XS) está conformado por el conjunto de soluciones que se pueden construir tras efectuar un MXS

en X. En la figura 3 se ilustra para el ejemplo de la tabla 1 un MXS en el cual el trabajo 4 asignado al segundo lote en la secuencia es intercambiado con el trabajo 1, asignado al tercer lote en la secuencia. Como se puede apreciar en este ejemplo el tiempo de terminación del tercer lote cambia de 39 a 41 como consecuencia de este intercambio.

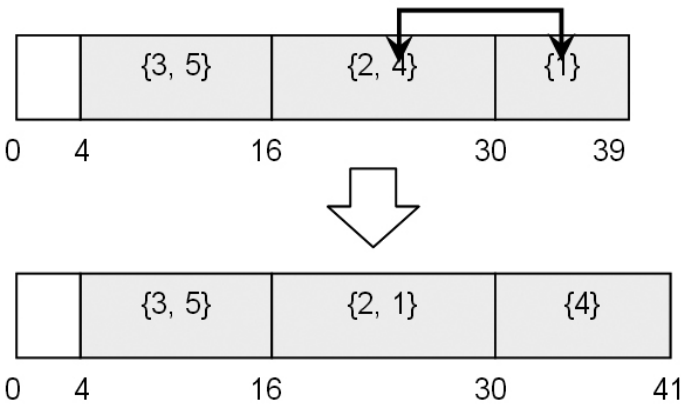

Figura 3. Ejemplo de un movimiento de intercambio simple (IS)

Al combinar los dos vecindarios simples propuestos con los dos mecanismos de búsqueda local antes descritos (i.e. FI y BI) se da origen a cuatro posibles vecindarios: (1) IS–FI, (2) IS–BI, (3) XS–FI, (4) XS–BI. Las combinaciones de estos cuatro vecindarios simples dan origen a 16 vecindarios compuestos, cada uno con dos movimientos simples consecutivos. La tabla 2 presenta los 16 vecindarios compuestos que pueden construirse.

Tabla 2. Denominación de los vecindarios compuestos

	Movimiento 2			
Movimiento 1	**IS–FI**	**IS–BI**	**XS–FI**	**XS–BI**
IS–FI	IS–FI+IS–FI	IS–FI+IS–BI	IS–FI+XS–FI	IS–FI+XS–BI
IS–BI	IS–BI+IS–FI	IS–BI+IS–BI	IS–BI+XS–FI	IS–BI+XS–BI
XS–FI	XS–FI+IS–FI	XS–FI+IS–BI	XS–FI+XS–FI	XS–FI+XS–BI
XS–BI	XS–BI+IS–FI	XS–BI+IS–BI	XS–BI+XS–FI	XS–BI+XS–BI

Es importante anotar que, para obtener un mejor desempeño, los mecanismos de exploración de vecindarios deben tener la posibilidad de crear y eliminar lotes. De no ser así, la búsqueda de mejores soluciones quedará restringida a soluciones con el mismo número de lotes de la solución inicial, lo cual no es necesariamente conveniente debido a la presencia de fechas de liberación arbitrarias para los trabajos. En esta investigación, la posibilidad de crear y eliminar lotes está incluida en los movimientos de inserción. La eliminación de un lote se da cuando se extrae de un lote el único trabajo en el lote, con el objetivo de insertarlo en otro. En este caso, el lote de donde se extrae el trabajo queda vacío y se elimina. Para la creación de lotes se asume que al inicio y al fin de la secuencia, así como entre cada par de lotes consecutivos,

en la misma existe un lote vacío imaginario. Así, al insertar un trabajo en uno de estos lotes imaginarios se crea un lote nuevo. La figura 4 presenta un ejemplo de eliminación de lote a través de un movimiento de inserción, en el cual el lote de origen (con un solo trabajo) es eliminado una vez efectuada la inserción. En la figura 5 se presenta un ejemplo de la creación de un lote a través de un movimiento de inserción, en el cual el lote destino es un lote vacío.

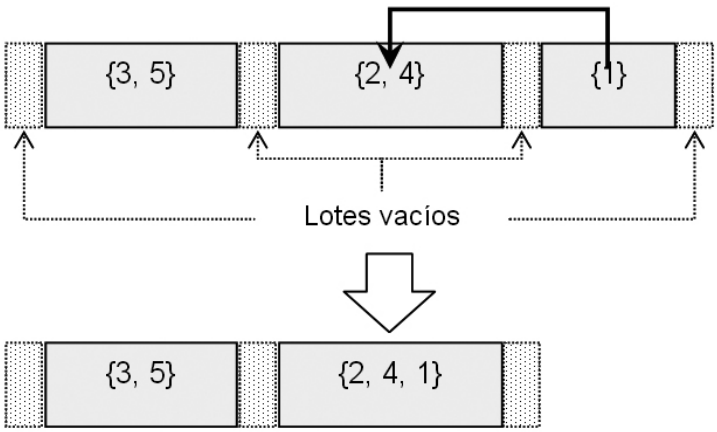

Figura 4. Ejemplo de un movimiento de inserción en el que se elimina un lote

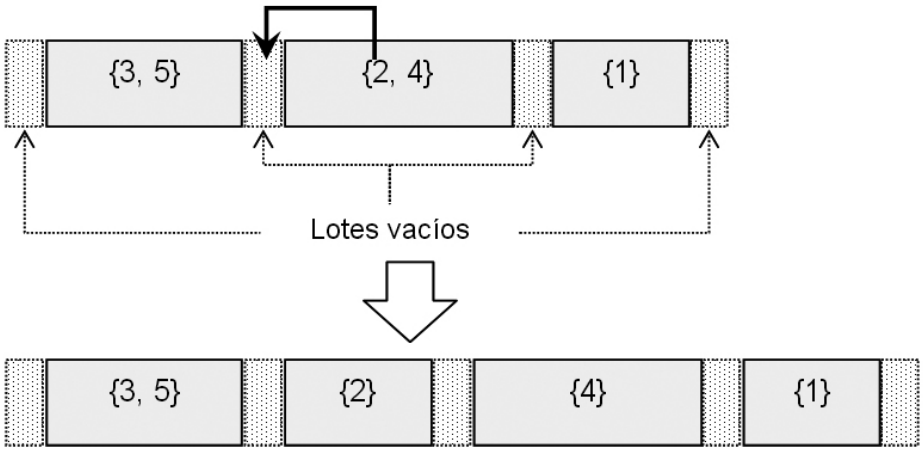

Figura 5. Ejemplo de un movimiento de inserción en el que se crea un lote

Heurístico BEVD Propuesto

Con el fin de buscar un balance adecuado entre el costo computacional y la calidad de las soluciones encontradas por el algoritmo, se llevó a cabo una etapa de experimentación preliminar para determinar algunos de los elementos críticos en el desempeño del algoritmo: (1) la cantidad de vecindarios a explorar, (2) cuáles vecindarios explorar en caso de no poder explorar los 20 vecindarios diseñados (cuatro simples y 16 compuestos), (3) el orden en que se deben explorar los vecindarios seleccionados y (4) el criterio de parada del algoritmo. Con respecto a la cantidad de vecindarios a explorar, se pudo establecer que al explorar siete vecindarios se logra un balance adecuado entre el costo computacional y calidad de las soluciones. Con relación a la selección de los siete vecindarios a explorar y al orden en que éstos deben ser explorados, durante la etapa de experimentación preliminar se seleccionó la configuración de mejor desempeño. La configuración seleccionada se presenta en la tabla 3.

Tabla 3. Vecindarios seleccionados

Vecindario	Identificación
1	IS–FI
2	XS–BI
3	IS–BI
4	XS–FI
5	IS–BI+XS–BI
6	XS–BI+XS–BI
7	IS–BI+IS–BI

Con respecto a la condición de terminación, en esta implementación se agregó un criterio de parada adicional, consistente en un tiempo máximo de ejecución, que se estableció en media hora (1800 s). En consecuencia, el algoritmo termina porque (1) no es posible mejorar la solución luego de explorar todos los vecindarios considerados o (2) porque se cumple el tiempo máximo de ejecución. A continuación se presenta el pseudocódigo de la implementación de BEVD usada para este artículo.

Definiciones

- Sea R el conjunto de reglas de despacho a utilizar para construir la solución inicial. R = {FIFO, EDD, ERT, WSPT, WLPT, LJS, SJS, ATC}.
- Sea X←SolucionInicial(r, I) una función que devuelve en X la solución inicial encontrada para la instancia I usando la regla de despacho r ∈ R.
- Sea N = {N_1, N_2,...,N_k} el conjunto de vecindarios explorados
- Sea X←N_j(X) una función que devuelve en X la mejor solución encontrada luego de ejecutar un procedimiento de búsqueda local que explora el vecindario N_j partiendo de la solución X.
- Sea Y←TWT(X) una función que devuelve en Y la tardanza total ponderada de la solución X.
- Sea X* la mejor solución encontrada.

Pseudocodigo del heurístico BEVD

Inicio
Leer datos de la instancia I
TWT* ← ∞
Mientras el tiempo de ejecución sea menor al tiempo máximo **hacer**:
Para cada r ∈ R:
Hacer X ← SolucionInicial(r, I)
Para cada j ∈ N:
Hacer X ← N_j(X)
Fin

```
        Fin
        Si TWT(X) < TWT* entonces
            X* ← X
        Fin
    Fin
Fin
```

EXPERIMENTOS COMPUTACIONALES

Para evaluar el desempeño computacional del heurístico propuesto se utilizó un conjunto de instancias generadas aleatoriamente. Para cada instancia, los tiempos de proceso y los tamaños de los trabajos fueron generados a partir de distribuciones uniformes discretas en los intervalos [1, *MaxP*] y [1, *MaxS*], respectivamente. Una vez generados los tiempos de proceso para cada trabajo en la instancia, las fechas de liberación fueron generadas a partir de una distribución uniforme discreta en el intervalo [1, RZ], donde Z es la suma de los tiempos de proceso de los trabajos en la instancia y R es un valor entre cero y uno que determina la tan dispersión de las fechas de entrega en el horizonte de planeación. Las fechas de entrega se generaron de manera que fueran fechas factibles. Así, la fecha de entrega del trabajo j (d_j) se generó en el intervalo [r_j+p_j, Q_j], donde Q_j es el *makespan* o tiempo total de fabricación encontrado por medio del heurístico PSKP propuesto en (Damodaran & Velez-Gallego, 2010). Por último, los pesos o importancia de los trabajos se generaron a partir de una variable aleatoria discreta en el intervalo [1, 10]. Se consideraron instancias con 10, 20 y 50 trabajos, y en cada caso se generaron cinco instancias independientes para cada combinación de *MaxP*, *MaxS* y *R,* para un total de 180 instancias. Los valores considerados para *n*, *MaxP*, *MaxS* y *R* fueron los siguientes:

- *n*: {10, 20, 50}
- *MaxP*: {10, 30}
- *MaxS*: {5, 20}
- *R*: {0.05, 0.10, 0.50}

La capacidad de la máquina se consideró constante e igual a 20.

En una primera etapa de experimentación el modelo de programación entera mixta (MPEM) propuesto en (Mathirajan *et al.,* 2010) se implementó en un software comercial de optimización. El MPEM implementado fue ejecutado para cada una de las 180 instancias generadas, y se le asignó un tiempo máximo de cómputo por instancia de media hora. Si el software comercial encuentra la solución óptima dentro del tiempo asignado, se reporta el valor óptimo de la función objetivo y el tiempo de cómputo. Si por el contrario, al cabo de media hora de búsqueda el software comercial no ha terminado la búsqueda de la solución óptima, se detiene la ejecución del modelo para la instancia correspondiente y se reporta la mejor solución entera (MSE) y la mejor cota inferior o mejor relajación lineal (MRL) encontrada. Para cada instancia se calculó la brecha de optimalidad (%Δ), es decir, la diferencia porcentual entre la MSE y la MRL calculada como en la ecuación (2).

$$\%\Delta = \frac{\mathrm{MSE} - \mathrm{MRL}}{\mathrm{MSE}} \times 100\% \qquad (2)$$

En la Tabla 4 presenta un resumen de los resultados obtenidos. Para cada tamaño de instancia evaluado se muestra (1) el número de soluciones óptimas encontradas por el software de optimización durante media hora de cómputo y (2) la brecha de optimalidad promedio. En la mencionada tabla se puede apreciar cómo a medida que crece el tamaño de las instancias, la brecha de optimalidad promedio crece hasta hacerse igual al 100% en instancias con 50

trabajos; y el número de instancias resueltas óptimamente decrece hasta hacerse igual a cero.

Tabla 4. Resumen de resultados del modelo matemático

n	Total de Instancias	Soluciones Óptimas	%Δ Promedio
10	60	60	0.0%
20	60	7	79. 7%
50	60	0	100.0%

En una segunda etapa de experimentación se ejecutó el heurístico BEVD para cada una de las instancias generadas. En cada caso se reportó el valor de la función objetivo y el tiempo de cómputo. Para evaluar la calidad de las soluciones encontradas por el heurístico BEVD se calculó, para cada instancia, la desviación porcentual (%θ) entre la solución encontrada por el heurístico BEVD en un máximo de media hora, y la MSE encontrada por el software comercial de optimización, también en un máximo de media hora. La desviación porcentual se calculó de acuerdo con la ecuación (3).

$$\%\theta = \begin{cases} \dfrac{\text{BEVD} - \text{MSE}}{\text{MSE}} \times 100\% & \text{si MSE} > 0 \\ \text{BEVD} \times 100\% & \text{si MSE} = 0 \end{cases} \quad (3)$$

La figura 6 presenta la desviación porcentual promedio para cada tamaño de instancia evaluado, discriminada por el valor de R, que representa la dispersión de las fechas de liberación en las instancias generadas. En la mencionada figura se puede apreciar cómo, en instancias con 20 trabajos o más, las soluciones encontradas por el heurístico BEVD son mejores a las encontradas por el software comercial de optimización. Asímismo, la magnitud de la diferencia porcentual crece con el tamaño de la instancia.

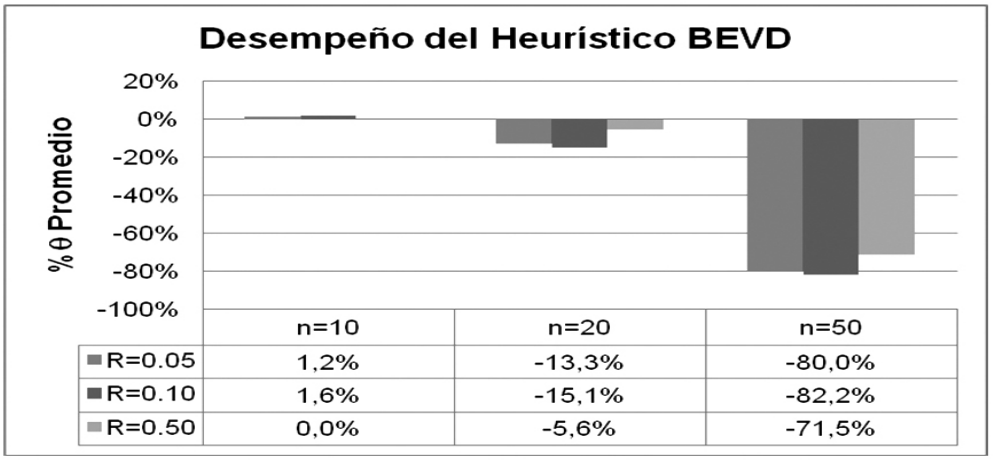

Figura 6. Desviación porcentual promedio entre el heurístico BEVD y la MSE

Por otro lado, la figura 7 presenta el tiempo computacional promedio en segundos reportado por ambos métodos de solución. Puede observarse cómo, para las instancias de 10 y 20 trabajos, el heurístico BEVD es consistentemente más rápido que el software comercial de optimización. En las instancias con 50 trabajos, el tiempo computacional promedio es igual al tiempo máximo de cómputo (1800 s) para ambos métodos de solución, debido a que la ejecución fue interrumpida tras alcanzar el tiempo máximo de cómputo.

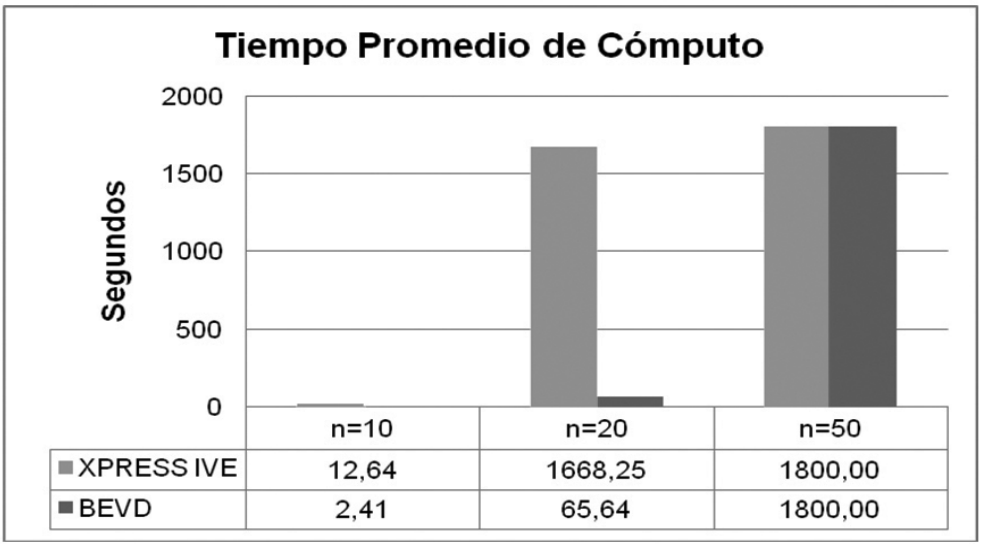

Figura 7. Desviación porcentual promedio entre el heurístico BEVD y la MSE

Por último, se analiza el impacto de cada una de las reglas de despacho utilizadas para obtener la solución inicial de la que parte el heurístico BEVD. Sea Z(r,i) una variable indicadora (binaria) igual a uno si la regla r condujo al heurístico BEVD a la mejor solución encontrada para la instancia i. El porcentaje de participación ($\%P_r$) para la regla r se calculó de acuerdo con la ecuación (4). En la figura 8 se presenta el valor obtenido de $\%P_r$ para cada regla. Nótese que, para una misma instancia, la mejor solución puede encontrarse partiendo de dos o más soluciones iniciales diferentes.

$$\%P_r = \frac{\sum_{i=1}^{180} Z(r,i)}{180} \times 100\% \quad (4)$$

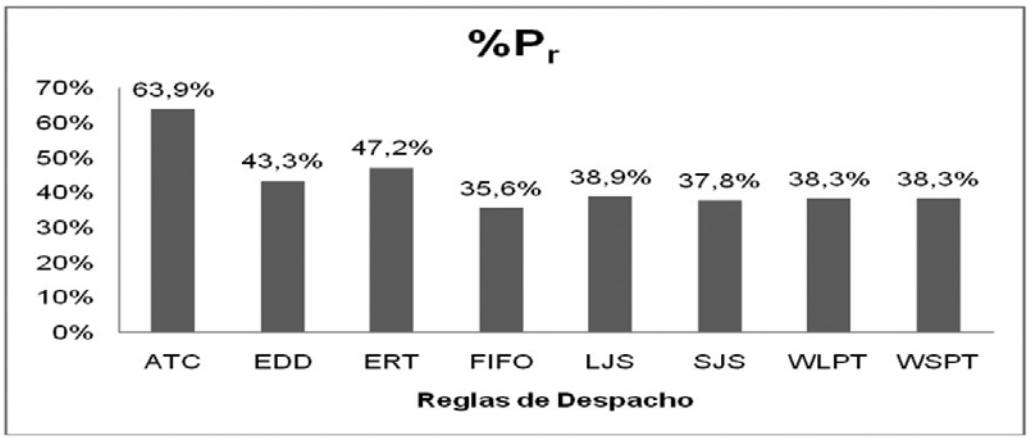

Figura 8. Participación de las reglas de despacho en la mejor solución encontrada

El software comercial de optimización en el cual se implementó el modelo matemático fue XPRESS IVE 7.1®, mientras que el heurístico BEVD fue codificado y compilado en Visual Basic 6.0 ®. Ambos métodos de solución fueron ejecutados en una computadora personal equipada con un procesador Intel ® Core 2 Duo de 2.2 GHZ y con 2 GB de memoria RAM.

CONCLUSIONES

El problema de programar la producción en una máquina de procesamiento por lotes del tipo *p–batch,* con el objetivo de minimizar la tardanza total ponderada, es un problema de optimización combinatoria computacionalmente complejo, del tipo NP—duro, que ha recibido poca atención en la literatura. Debido al alto costo computacional requerido por las aplicaciones comerciales de optimización orientadas a encontrar la solución óptima para este problema, es necesario recurrir a soluciones de aproximación o heurísticas, que aunque no garantizan la solución óptima permiten encontrar soluciones de buena calidad en un tiempo computacional razonable.

De los experimentos computacionales realizados se puede concluir que, en instancias pequeñas con hasta 10 trabajos, un software comercial de optimización (XPRESS IVE 7.1 ®) se desempeña mejor que el heurístico BEVD en cuanto a la calidad de las soluciones encontradas. A partir de 20 trabajos, el heurístico BEVD propuesto encuentra soluciones mejores a las encontradas por el software de optimización, como se puede apreciar en la figura 6. A medida que el número de trabajos en la instancia crece, el desempeño del heurístico BEVD mejora con respecto al software de optimización, de tal manera que al evaluar instancias con 50 trabajos el heurístico propuesto encontró soluciones entre el 71.5% y el 82.2% mejores en promedio, como se aprecia en la figura 6. Por todo lo anterior es posible concluir que el heurístico propuesto es efectivo en la solución del problema en estudio, y puede ser usado en aplicaciones industriales.

BIBLIOGRAFÍA

Amiri, M., Zandieh, M., Yazdani, M., & Bagheri, A. (2010). A variable neighbourhood search algorithm for the flexible job-shop scheduling problem. *International Journal of Production Research*, 48(19), 5671-5689.

Brucker, P. (2007). Batching Problems. In: Brucker, *Scheduling Algorithms,* (5th ed., pp. 267-277). Springer.

Brucker, P., Gladky, A., Hoogeveen, H., Kovalyov, M. Y., Potts, C. N., Tautenhahn, T., Van de Valde, S. (1998). Scheduling a batching machine. *Journal of Scheduling*, 1(1), 31-54.

Damodaran, P., & Vélez-Gallego, M. C. (2010). Heuristics for makespan minimization on parallel batch processing machines with unequal job ready times. *International Journal of Advanced Manufacturing Technology*, 49(9-12), 1119-1128.

Graham, R. L., Lawler, E. L., Lenstra, J. K., & Rinnooy Kan, A. H. G. (1979). Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics*, 5, 287-326.

Hansen, P., Mladenovic, N. Brimberg, J, Pérez Moreno, J.A. (2010). Variable Neighborhood Search. In M. Gendreau & J.-Y Potvin (Eds.), Handbook of Metaheuristics International Series in Operations Research & Management Science, vol 146, (pp. 61-86). Springer.

Hansen, P., Mladenović, N., & Moreno Pérez, J. A. (2009). Variable neighbourhood search: methods and applications. *Annals of Operations Research*, 175(1), 367-407.

Kurz, M. E., & Mason, S. J. (2008). Minimizing total weighted tardiness on a batch-processing machine with incompatible job families and job ready times. *International Journal of Production Research*, 46(1), 131-151.

Liao, C.J., & Cheng, C.C. (2007). A variable neighborhood search for minimizing single machine weighted earliness and tardiness with common due date. *Computers & Operations Research*, 52(4), 404-413.

Mathirajan, M., Bhargav, V., & Ramachandran, V. (2010). Minimizing total weighted tardiness on a batch-processing machine with non-agreeable release times and due dates. *International Journal of Advanced Manufacturing Technology*, 4, 1133-1148.

Mladenovic, N., & Hansen, P. (1997). Variable Neighborhood Search. *Computers & Operations Research*, 24(11), 1097-1100.

Perez, I. C., Fowler, J. W., & Carlyle, W. M. (2005). Minimizing total weighted tardiness on a single batch process machine with incompatible job families. *Computers & Operations Research*, 32, 327 - 341.

Pinedo, M. L. (2008). Scheduling: Theory, Algorithms and Systems (3rd ed.). New York. NY: Springer.

Rardin, R. L., & Uzsoy, R. (2001). Experimental Evaluation of Heuristic Optimization Algorithms: A Tutorial. *Journal of Heuristics*, 7, 261-304.

Roshanaei, V., Naderi, B., Jolai, F., & Khalili, M. (2009). A variable neighborhood search for job shop scheduling with set-up times to minimize makespan. *Future Generation Computer Systems*, 25(6), 654-661.

Sevkli, M., & Aydin, M. E. (2006). A Variable Neighbourhood Search Algorithm for Job Shop Scheduling Problems. In J. Gottlieb & G. R. Raidl (Eds.), *Evolutionary Computation in Combinatorial Optimization* (Vol. 3906, pp. 261-271). Berlin, Heidelberg: Springer.

Tangudu, S. K., & Kurz, M. E. (2006). A branch and bound algorithm to minimise total weighted tardiness on a single batch processing machine with ready times and incompatible job families. *Production Planning & Control October*, 17(7), 728-741.

Zandieh, M., & Adibi, M. A. (2010). Dynamic job shop scheduling using variable neighbourhood search. *International Journal of Production Research*, 48(8), 2449-2458.