ALGORÍTMO DE BÚSQUEDA DE ENTORNO VARIABLE PARA MINIMIZAR EL *MAKESPAN* EN UN SISTEMA *FLOW SHOP* CON MÁQUINAS DE PROCESAMIENTO POR LOTES

RESUMEN

En este documento se presenta una propuesta de solución para un problema de programación de producción en un sistema de fabricación en línea o *flow shop*, en el cual el objetivo es minimizar el tiempo necesario para fabricar un conjunto de trabajos, de tal manera que indirectamente se maximice la utilización de las máquinas. Cada trabajo pendiente de ser programado está descrito por un tiempo de procesamiento en cada máquina y un valor de peso o tamaño. El sistema de producción objeto de estudio está compuesto por máquinas de procesamiento por lotes (MPL); es decir, máquinas que pueden procesar múltiples trabajos simultáneamente (i.e., lotes), siempre y cuando la suma de los tamaños de estos lotes no excedan la capacidad de la máquina, ya sea en términos de peso o de tamaño. En este caso el tiempo de procesamiento de un lote está dado por el tiempo máximo de procesamiento de los trabajos que conforman el lote. Dado que este problema es del tipo NP-completo se propone una heurística de búsqueda de entorno variable para resolver el problema. Los experimentos computacionales encontraron que con un tiempo de cómputo de 30 minutos por instancia se pueden encontrar resultados similares y en algunos casos mejores que los encontrados al resolver un modelo de programación lineal entera implementado en un *solver* comercial con tiempos de hasta 6 horas de procesamiento.

Palabras Clave: Heurística, búsqueda de entorno variable, máquinas de procesamiento por lotes, programación de producción.

A VARIABLE NEIGHBORHOOD SEARCH ALGORITHM TO MINIMIZE THE MAKESPAN IN A FLOW SHOP OF BATCH PROCESSING MACHINES

ABSTRACT

This work presents a solution approach for the flow shop scheduling problem with the objective of minimizing the time required to complete the set of jobs (makespan minimization) so that the utilization of machines is maximized. Each job is represented by a processing time and a value of size or weight. The manufacturing system studied in this work is comprised of batch processing machines; this is, machines that can process several jobs simultaneously as batches, given that the total size of the jobs in the batch does not exceed the capacity of the machine in terms of either size or weight. For this work, the processing time of a batch is given by the largest processing time among the jobs in the batch. As the problem is NP-complete, a variable neighborhood search (VNS) approach is proposed to solve the problem. The computational experiments conducted determined that within 30 minutes of computational time, the VNS approach is able to find similar and sometime better solutions than those found after solving a mixed integer linear formulation using a commercial solver.

Keywords: Heuristic, variable neighborhood search, batch processing machines, scheduling, flow shop scheduling problem.

INTRODUCCIÓN

Las MPL o máquinas de procesamiento por lotes se caracterizan porque pueden procesar múltiples trabajos simultáneamente. Las MPL son comunes en procesos de pintura en los cuales múltiples piezas se procesan en una cabina de pintura de manera simultánea, o en tratamientos térmicos de piezas metálicas, en los cuales un conjunto de piezas se procesa simultáneamente en un horno. En este artículo se considera un sistema de fabricación en línea, también conocido como *flow-shop*, en el cual se cuenta con una MPL en cada etapa del proceso. El objetivo de esta investigación es proponer un método de programación de producción que minimice la medida de desempeño conocida en la literatura como el *makespan*, que consiste en el tiempo necesario para procesar la totalidad de los trabajos. Al minimizar esta métrica se maximiza la utilización de estos equipos (i.e., las MPL), lo cual es consistente con una estrategia de minimización de costos.

Más formalmente, el problema de programación de producción abordado en esta investigación consiste en programar un conjunto J de n trabajos en un conjunto M de m máquinas, todas del tipo MPL, dispuestas en una configuración de producción de tipo *flow-shop*, cada una con una capacidad máxima S. Cada trabajo $j \in J$ está descrito por el tiempo mínimo que debe permanecer el trabajo en cada máquina $m \in M$ (p_{jm}); y el tamaño del trabajo (s_j). El objetivo consiste en conformar lotes de los trabajos, sin que se viole la capacidad de las máquinas, de manera que se puedan terminar todos los trabajos en el menor tiempo posible (i.e. *makespan*). Bajo el supuesto de que los trabajos pueden permanecer en la máquina más tiempo del mínimo establecido en cada caso sin que esto perjudique la calidad de este, el tiempo de procesamiento de un lote en una máquina está dado por el mayor tiempo de procesamiento entre los trabajos que componen el lote. Para explicar mejor el problema se presenta el ejemplo descrito en la Tabla 1, el cual consiste en una instancia con 10 trabajos y 3 máquinas.

p_{jm}	m=1	m=2	m=3	s_j
j=1	27	30	14	8
j=2	9	11	30	4
j=3	13	1	17	4
j=4	1	24	20	6
j=5	10	7	26	5
j=6	20	24	16	1
j=7	24	12	28	4
j=8	7	4	18	2
j=9	22	18	2	3
j=10	12	21	17	9

Tabla1. Tiempos de procesamiento y tamaños en una instancia con 10 trabajos

Para resolver este problema es importante reconocer que existen dos posibles estrategias de agrupación de los trabajos para formar los lotes. Una primera

estrategia consiste en definir lotes fijos, es decir, lotes que no cambian en la medida en que los trabajos son procesados por las diferentes máquinas. Bajo esta estrategia, una posible solución es la que se presenta en la Tabla 2. El diagrama de Gantt de la Figura 1 presenta el *makespan* en que se incurre para fabricar los 10 trabajos del ejemplo, el cual en este caso es de 131 unidades de tiempo.

Lote	Trabajos	Tiempo de finalización del lote
L1	[3, 5]	46
L2	[4]	66
L3	[2,7,8]	96
L4	[10]	113
L5	[1, 6]	129
L6	[9]	131

Tabla 2. Estrategia de lotes fijos.

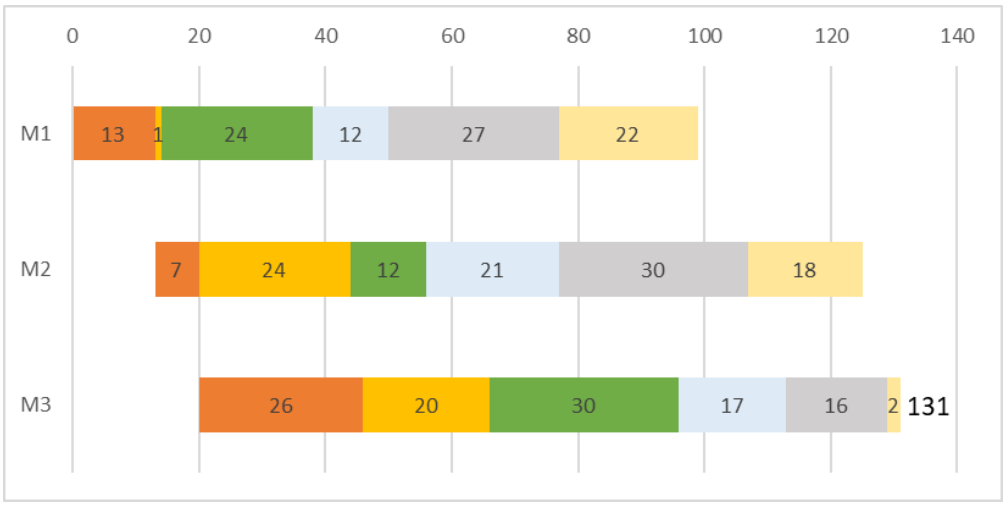

Figura 1. Diagrama de Gantt para la estrategia lotes fijos.

Una segunda estrategia de agrupación de los trabajos en lotes consiste en permitir reagrupar los trabajos en la medida en que los lotes son procesados por las diferentes máquinas. A esta estrategia la denominamos de ahora en adelante la estrategia de lotes variables. La Tabla 3 presenta una posible agrupación de los trabajos bajo esta segunda estrategia, y la Figura 2 presenta el diagrama de Gantt correspondiente a esta solución. En este caso el *makespan* es de 128 unidades de tiempo.

	Composición de los lotes		
Lote	**Máquina 1**	**Máquina 2**	**Máquina 3**
L1	[2,5]	[5]	[5]
L2	[4]	[2,4]	[3,4]
L3	[3,7,8]	[3]	[2,7,8]
L4	[10]	[7,8]	[10]
L5	[1,6]	[10]	[1,6]
L6	[9]	[1,6]	[9]
L7		[9]	

Tabla 3. Estrategia de lotes variables.

Tiempo de finalización del lote			
Lotes	**M1**	**M2**	**M3**
L1	10	17	43
L2	11	41	63
L3	35	42	93
L4	47	54	110
L5	74	75	126
L6	96	105	128
L7		123	

Tabla 4. Tiempo de procesamiento por lotes estrategia de lotes variables.

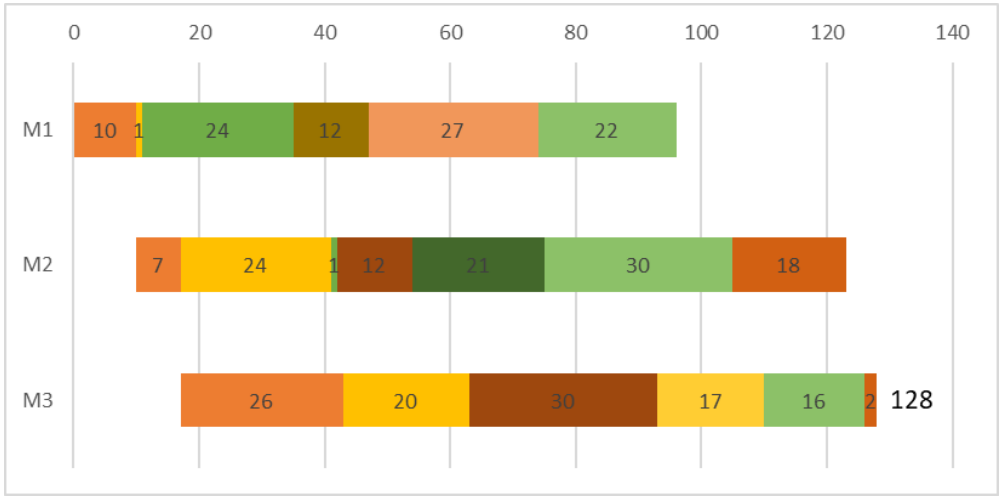

Figura 2. Diagrama de Gantt para la estrategia de lotes variables.

Con este trabajo se busca establecer si la adopción de una estrategia de solución heurística como la búsqueda de entorno variable es competitiva en términos de la calidad de las soluciones encontradas y del costo computacional asociado. Así mismo se busca establecer si la estrategia de conformar lotes variables permite mejorar las soluciones encontradas con relación a la estrategia de lotes fijos.

REVISIÓN DE LITERATURA

La revisión de literatura que se presenta a continuación está limitada a problema de programación de producción en sistemas de tipo *flow shop* en los cuales se involucre al menos una máquina de procesamiento por lotes. Adicionalmente en la revisión solo se consideraron trabajos que abordan el problema objeto de esta investigación bajo los siguientes dos supuestos:

1. El tiempo de procesamiento de un lote está determinado por el tiempo máximo de procesamiento de los trabajos que componen el lote.
2. Cada trabajo tiene un tamaño asociado, y la suma de los tamaños de los trabajos que conforman el lote no puede superar la capacidad de la máquina.

Hasta donde se pudo constatar, el primer trabajo que aborda un problema con las características descritas es la referencia [1]. En este trabajo los autores proponen una formulación lineal entera mixta para un problema con dos máquinas de procesamiento por lotes con el objetivo del minimizar el *makespan*. Kashan y Karimi [2] extendieron el trabajo anterior a un número arbitrario de máquinas, y propusieron algunas mejoras en la formulación matemática con el fin de mejorar el costo computacional necesario para encontrar la solución óptima. Debido a que estos trabajos solo permiten resolver instancias relativamente pequeñas, los autores de la referencia [3] propusieron dos heurísticos constructivos y dos implementaciones del metaheurístico conocido como recocido simulado (i.e., *simulated annealing*) para minimizar el *makespan* en un sistema con dos máquinas. En la misma dirección, los autores de la referencia [4] proponen un heurístico constructivo y una implementación de recocido simulado para resolver instancias de hasta 100 trabajos en sistemas con dos máquinas. En el trabajo de Manjeshwar et. al. [5] se abordó una vez más el problema de dos máquinas con el objetivo de minimizar el *makespan*. En este trabajo se propuso un algoritmo genético que, de acuerdo con los experimentos computacionales publicados, encuentra en promedio mejores soluciones que el método propuesto en [4]. La referencia [6] propone extender el problema para incluir tiempos de liberación de los trabajos. En este trabajo se considera un sistema con dos máquinas, pero con una variación en relación con los trabajos anteriormente mencionados. El tiempo de procesamiento de un lote en la segunda máquina se calcula como la suma de los tiempos de proceso de los trabajos en el lote. Bajo estas premisas los autores proponen una formulación lineal entera mixta para el problema, y un algoritmo de optimización PSO (i.e., *particle swarm optimization*). En la referencia [7] se extiende el trabajo publicado en [6] considerando sistemas con un número arbitrario de máquinas. En este trabajo se

proponen tres heurísticos constructivos, así como un algoritmo de optimización PSO.

La variante del problema conocida en la literatura especializada como *no-wait* fue abordada en la referencia [8]. En este caso no se permite que un trabajo espere para ser procesado; es decir, siempre que se termine el procesamiento de un lote, la máquina siguiente en el proceso debe estar disponible, de manera que no se generen esperas para los trabajos. Para resolver esta variante del problema se consideraron sistemas con dos máquinas y se propuso una implementación de la metaheurística conocida como GRASP (i.e., *greedy randomized adaptive search procedure*). Una solución basada en computación evolutiva para el problema de dos máquinas con tiempos de liberación arbitrarios se propuso en [9]. Los resultados obtenidos se comparan con las soluciones obtenidas luego de implementar un modelo de programación entera mixta en un *solver* comercial.

La referencia [10] considera el problema de dos máquinas con tiempos de liberación arbitrarios. En este trabajo los autores proponen una formulación entera mixta mejorada y un heurístico constructivo para minimizar el *makespan*.

Más recientemente, los autores de la referencia [11] consideran por primera vez la posibilidad de que los lotes puedan ser modificados antes de iniciar el proceso en las máquinas. Como método de solución proponen una formulación lineal entera mixta y un algoritmo PSO. Por último, la referencia [12] también propone un algoritmo PSO, pero esta vez para minimizar la medida de desempeño conocida como *earliness-tardiness*, en la cual se consideran fechas de entrega arbitrarias para cada trabajo, y se penaliza tanto si el trabajo se termina tarde (i.e., *tardiness*), como si se termina anticipadamente (i.e., *earliness*).

Hasta donde se pudo constatar en esta revisión de literatura, la única referencia en la cual se considera la estrategia de lotes variables es la referencia [11]. En este trabajo se propone una alternativa al método propuesto en el citado trabajo.

SOLUCIÓN PROPUESTA

Para resolver un problema de optimización como el presentado anteriormente, una opción válida es el algoritmo denominado búsqueda de entorno variable (BEV) o *variable neighbourhood search* (VNS). Esta es una metaheurística que realiza un cambio sistemático de vecindarios dentro de un proceso de búsqueda local, teniendo en cuenta que un óptimo local para un vecindario no necesariamente es un óptimo local para otro vecindario. La BEV realiza un cambio de vecindario cuando encuentra en un óptimo local, con el fin de mejorar la solución del problema. Para la implementación de una BEV se parte de un conjunto $N = \{N_1, N_2, \dots, N_K\}$ de K vecindarios seleccionados previamente.

En este trabajo es utilizada una variante de la BEV denominada búsqueda de entorno variable descendente (BEVD), en la cual los vecindarios se exploran de

manera determinística (sin aleatoriedad). En este caso en particular se implementaron dos mecanismos de exploración de los vecindarios, denominados *First Improve* (FI) y *Best Improve* (BI). En una búsqueda *FI*, se exploran los vecindarios hasta que se encuentre la primera mejora en la solución, la cual se adopta como la nueva mejor solución, y se inicia nuevamente el proceso de búsqueda. En una búsqueda BI el vecindario se explora exhaustivamente; es decir, se evalúan todas las soluciones en el vecindario; para luego adoptar como nueva solución la mejor solución encontrada. En ambos casos, cuando el algoritmo se quede en un óptimo local, se procede a cambiar de vecindario dentro de los K vecindarios establecidos previamente.

Para inicializar el algoritmo es necesario contar con una solución inicial, a partir de la cual se pueda desplegar el proceso de búsqueda descrito anteriormente. En este caso la solución inicial se construye de la siguiente manera:

Paso 1: Ordenar los trabajos en orden decreciente del tiempo total de procesamiento; es decir, de la suma de los tiempos de proceso sobre todas las máquinas.

Paso 2: Agrupar los trabajos en lotes teniendo en cuenta el tamaño de cada trabajo y la capacidad máxima de las máquinas.

Paso 3: Programar los lotes en las máquinas usando la heurística NEH.

Según la heurística NEH propuesta en [13], los lotes en un sistema *flow shop* se programan de la siguiente manera:

Paso 1: Calcule el tiempo total de procesamiento de cada trabajo; es decir, la suma de los tiempos de procesamiento del trabajo en todas las máquinas.
Paso 2: Construya una lista L con los trabajos en orden no creciente del tiempo total de procesamiento calculado en el paso 1.
Paso 3: Considere los dos primeros trabajos en la lista (i.e., j y k), y prográmelos en el *flow shop* usando las dos posibles secuencias parciales (i.e., j→k y k→j). Seleccione la secuencia S de menor *makespan*.
Paso 4: Elimines los trabajos j y k de la lista.
Paso 5: Considere el siguiente trabajo en la lista (i.e., k). Inserte el trabajo k en todas las posibles posiciones en S. Actualice S con la secuencia resultante de menor valor de *makespan*. Elimine el trabajo k de la lista L.
Paso 6: Si no hay más trabajos en L, terminar; de lo contrario ir al paso 5.

A continuación, se presenta un ejemplo con el fin de ilustrar cómo se construye la solución inicial propuesta. Para este propósito se utiliza la instancia de 10 trabajos y 5 máquinas y una capacidad máxima por maquina de 10, presentada en la Tabla 5.

p_{jm}	m=1	m=2	m=3	m4	m5	Total
j=1	8	2	7	1	7	25
j=2	5	9	8	10	1	33
j=3	1	2	5	7	6	21
j=4	7	3	8	1	6	25
j=5	5	8	8	10	1	32
j=6	6	3	4	8	4	25
j=7	4	2	8	4	2	20
j=8	9	3	10	1	8	31
j=9	9	2	3	7	3	24
j=10	1	8	3	7	1	20

s_j
2
6
3
4
4
4
5
3
3
4

Tabla 5. Tiempos de procesamiento para el ejemplo.

En la Tabla 6 se presentan los trabajos en orden descendente del tiempo total de procesamiento de cada trabajo, así como los trabajos agrupados por lotes.

p_{jm}	m=1	m=2	m=3	m4	m5	Total
j=2	5	9	8	10	1	33
j=5	5	8	8	10	1	32
j=8	9	3	10	1	8	31
j=1	8	2	7	1	7	25
j=4	7	3	8	1	6	25
j=6	6	3	4	8	4	25
j=9	9	2	3	7	3	24
j=3	1	4	5	7	6	23
j=7	4	2	8	4	2	20
j=10	1	8	3	7	1	20

s_j	Lote (b_i)
6	1
4	1
3	2
2	2
4	2
4	3
3	3
3	3
5	4
4	4

Tabla 6. Ejemplo solución inicial - Agrupar.

Finalmente, en la Tabla 7 se presentan los tiempos de procesamiento de los lotes obtenidos en el paso anterior. Con esta información se da inicio al proceso de generar una secuencia de los lotes siguiendo el heurístico NEH. La solución inicial una vez implementado este heurístico se presenta en la Figura 3.

	m1	m2	m3	m4	m5
b1	5	9	8	10	1
b2	9	3	10	1	8
b3	9	4	5	8	6
b4	4	8	8	7	2

Tabla 7. Ejemplo solución inicial – Lotes NEH.

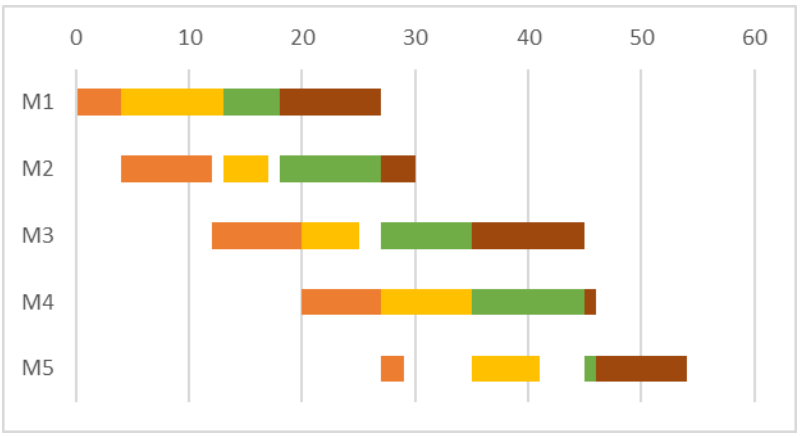

Figura 3. Diagrama de Gantt Mejor makespan NEH.

Vecindarios propuestos

Para resolver este problema se proponen dos tipos de vecindarios: los simples y los compuestos. Los vecindarios simples son aquellos donde se encuentra una solución realizando un solo movimiento del sobre la solución original. Los vecindarios compuestos son aquellos donde la solución se obtiene después de realizar dos o más movimientos hasta encontrar una nueva solución. Para moverse a través de estos vecindarios a continuación se explicará de qué manera se harán los movimientos para encontrar los diferentes tipos de vecindarios, de tal manera que se busque una nueva solución a partir de los siguientes tipos de movimientos:

Movimiento de inserción simple (MIS)

Teniendo una solución inicial a partir de alguno modelo, que para este caso es la NEH, se extrae el trabajo del lote K y se inserta en otro lote, este movimiento no debe violar la restricción de capacidad de la máquina; es decir, la inserción simple (IS) puede crear un conjunto de soluciones con cada movimiento realizado siempre y cuando no se violen las capacidades de la máquina m al generar el grupo de lotes K.

En la Figura 4 se ilustra un movimiento de inserción, donde el trabajo número 4 se inserta en el lote número 1 proveniente del lote número 2. Este movimiento genera una reducción del makespan de 17 a 14, esto como consecuencia de la inserción realizada sin violar la capacidad de la máquina.

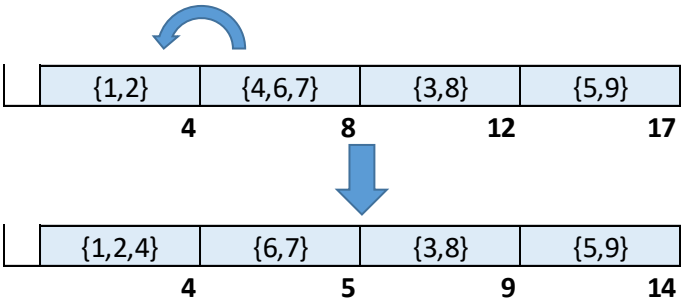

Figura 4. Ejemplo inserción simple (IS).

Movimiento de intercambio simple (MXS)

Como en el ejemplo anterior, teniendo en cuenta un grupo de lotes K, el MXS consiste en tomar dos trabajos de diferentes lotes en intercambiarlos entre sí, siempre y cuando este intercambio no viole la capacidad de la máquina m. Estos movimientos pueden crear un conjunto de soluciones dentro del grupo de lotes K. En la Figura 5 se ilustra un movimiento de intercambio, donde el trabajo número 4 que pertenece al lote número 2, se intercambia por el trabajo número 1 del lote número 1. Este movimiento genera una reducción del *makespan* de 17 a 15, esto como consecuencia del intercambio realizado sin violar la capacidad de la máquina.

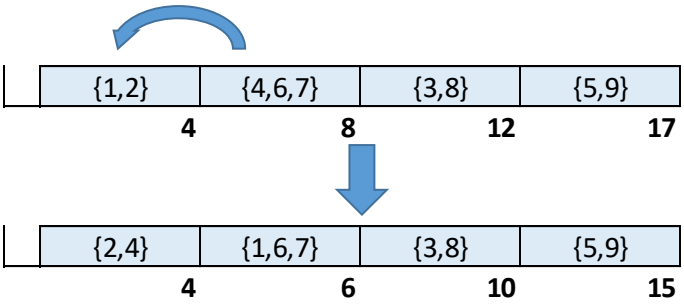

Figura 5 Ejemplo intercambio simple (XS).

Toda la información anteriormente definida nos permite obtener 4 diferentes tipos de rutas a seguir desde nuestra solución inicial. Estas rutas son mejor conocidas bajo el modelo de trabajo como mecanismos de búsqueda local. Estos 4 posibles vecindarios serían los siguientes: 1. IS-FI, 2. IS-BI, 3. XS-FI, 4. XS-BI. Si se combinasen estos 4 posibles vecindarios se generan 16 posibles nuevos vecindarios, lo que genera un efecto exponencial de crecimiento de los posibles vecindarios a buscar.

Dado que el número de lotes iniciales se crean bajo una heurística pero esta no es la única conjugación de lotes posible, se hace imperativo tener la posibilidad de crear y al mismo tiempo deshacer los lotes, de tal manera que no se restrinja la búsqueda de una solución en los lotes creados inicialmente, ya que dentro de estos

no necesariamente se puede encontrar un óptimo, solo un óptimo local para este número de lotes. Como consecuencia de todo lo anteriormente mencionado y con la intención de mejorar la calidad de la respuesta de nuestro modelo, se crea un paso adicional que consiste en la generación de nuevos lotes. Este movimiento a su vez puede destruir un lote reconfigurando el número de lotes K. Para poder hacer lo anterior se asume que siempre existe un lote vacío entre los lotes ya definidos de tal manera que se genere un movimiento de inserción que permita ubicar un trabajo en el lote que se encuentra vacío. En la Figura 6 se presenta un ejemplo de la modificación de un lote y como consecuencia de este movimiento la eliminación de uno ya existente (los espacios representan lotes vacíos).

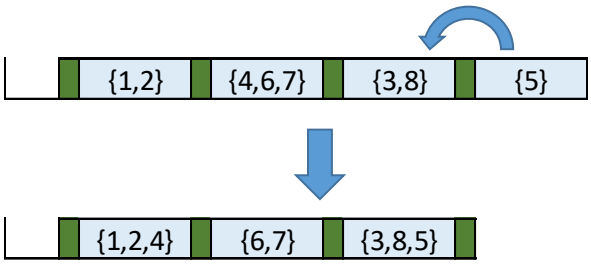

Figura 6. Movimiento de inserción con eliminación de lotes.

En la Figura 7 se observa la creación de un nuevo lote a partir del uso de los lotes vacíos que se encuentran entre los lotes ya definidos.

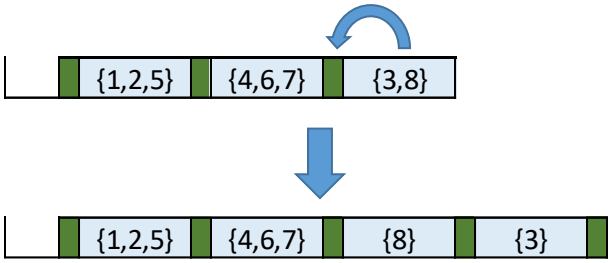

Figura 7. Ejemplo de inserción con creación de lote.

Heurística BEVD propuesta

Para la exploración de los vecindarios se definieron 20 vecindarios que corresponde a las combinaciones posibles entre Inserción Simple (IS) First Improve (FI) y Best Improve (BI), así mismo el Intercambio Simple Firsrt Improve (FI) y Best Improve (BI). Al final estos 20 vecindarios están compuestos por 4 simples y 3 compuestos, el listado de los vecindarios a explorar se muestra en la Tabla 8.

Vecindario	Identificación
1	ISFI
2	ISBI
3	IXFI
4	IXBI
5	ISBI - ISBI
6	ISFI - ISFI
7	ISBI - ISFI
8	ISFI - ISBI
9	ISFI - IXFI
10	ISBI - IXBI
11	ISFI - IXBI
12	ISBI - IXFI
13	IXBI - IXBI
14	IXFI - IXFI
15	IXBI - IXFI
16	IXFI - IXBI
17	IXFI - ISBI
18	IXFI - ISFI
19	IXBI - ISBI
20	IXBI - ISFI

Tabla 8. Listado de vecindarios propuestos

Durante el proceso de exploración de vecindarios se realizan los movimientos los cuales siguen una secuencia creciente, empezando por el lote número 1 al lote máximo según el número de lotes K.

Teniendo en cuenta lo anterior, el recorrer y evaluar cada uno de los diferentes estados se convierte en una tarea extensa, y de alto consumo de recursos computacionales; motivo por el cual se incluyó el componente de aleatoriedad en los modelos, con la finalidad de crear cambios de estados más contundentes siguiendo los patrones de movimiento y agrupamiento que establece la heurística originalmente. Adicional a esto, permite al algoritmo no seguir siempre el mismo patrón de búsqueda, lo que limita la posibilidad de quedarse en los mismos óptimos locales cada iteración que realice, Cabe resaltar que, al momento de incluir el componente de aleatoriedad, se realizó bajo los mismos parámetros de movimiento que permite la heurística y los modelos planteados.

Es por esto por lo que en el modelo de intercambio simple se agregó el componente aleatorio al momento de insertar el lote, lo que ayuda a ampliar las posibilidades y no recurrir al mismo camino al momento de explorar los vecindarios.

Con el objetivo de encontrar la mejor solución posible el algoritmo tiene una regla de terminación de 1800 segundos.

EXPERIMENTOS COMPUTACIONALES

Con el objetivo de evaluar el desempeño de la heurística anteriormente definida, se utilizaron 60 diferentes instancias, donde se encuentra 3 escenarios de 10, 15 y 20 trabajos, para todos los casos se consideraron 3 máquinas.

Para efectos comparativos del rendimiento de la heurística se realiza contra en un modelo de programación entera mixta (MPEM) propuesto en [2], ejecutado en un software comercial de optimización. En total de los 60 problemas fueron resueltos a través de este software obteniendo una mejor solución entera (MSE), el comparativo se realiza contra la mejor solución de la heurística encontrada (MSH). Al final se realiza un comparativo de ambos métodos (%Δ), es decir, la diferencia porcentual entre la MSE y la MSH calculada a través de la siguiente ecuación.

$$\%\Delta = \frac{MSE - MSH}{MSE}\ x\ 100\%$$

El modelo desarrollado en el software de optimización fue ejecutado hasta encontrar el valor óptimo o hasta un máximo de 6 horas, el modelo evaluado a través de la heurística se ejecutó por 1800 segundos.

Acorde a los datos obtenidos, a pesar de que el software de optimización logra comúnmente mejores resultados, también se observa que los tiempos de procesamiento que se requieren son mayores para los casos con 20 trabajos.

Por otro lado, la siguiente tabla muestra un resumen de la eficiencia de la heurística vs el software de optimización. Puede observarse como esta diferencia se va ampliando a medida que aumentan los trabajos.

	n=10	n=15	n=20
S=5	0,00%	0,00%	4,58%
	0,00%	-1,48%	2,42%
	0,00%	0,00%	5,06%
	0,00%	0,00%	2,14%
	3,31%	0,00%	4,27%
	5,41%	2,14%	2,68%
	0,00%	0,00%	4,88%
	0,00%	0,00%	3,03%
	7,45%	0,00%	5,14%
	-0,93%	0,00%	3,13%
	1,52%	**0,07%**	**3,73%**

	n=10	n=15	n=20
S=10	0,00%	0,00%	1,39%
	0,00%	1,11%	5,60%
	0,00%	0,00%	14,39%
	0,00%	0,00%	10,41%
	-0,76%	0,55%	7,00%
	0,00%	0,48%	4,85%
	0,00%	1,95%	3,76%
	8,84%	19,00%	0,00%
	0,00%	1,82%	2,80%
	-1,36%	0,00%	8,66%
	0,67%	**2,49%**	**5,89%**

Tabla 9. %Δ variación eficiencia modelos por número de trabajos.

Por último, se analiza las variaciones que se tienen entre los resultados de la heurística propuesta y el software de optimización para cada una de las instancias. Como se podrá observar la variación crece a medida que el número de trabajos crece.

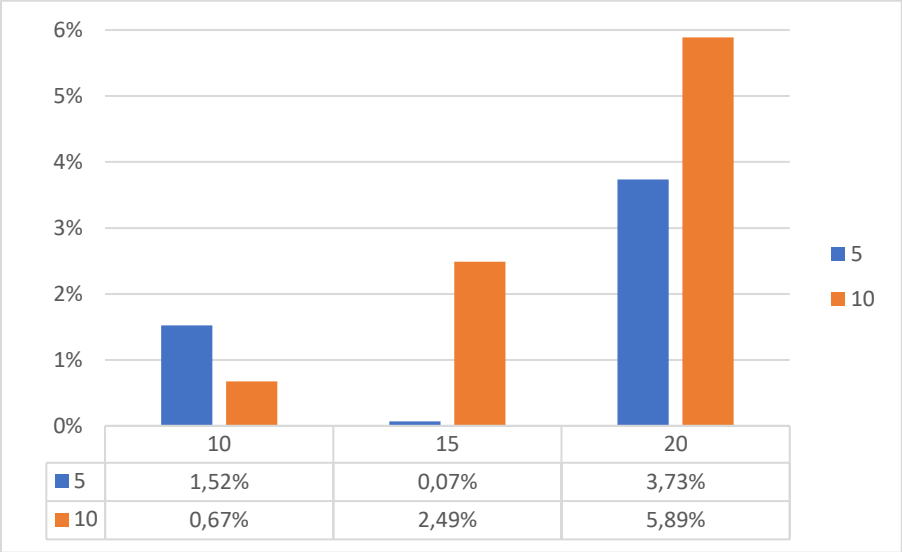

Figura 8. % Diferencia porcentual.

Como se puede observar en la grafica anterior, a medida que crecen los trabajos la la diferencia porcentual de la heurística baja en comparación de los resultados obtenidos por el software de optimización, esto puede deberse en gran parte a que los tiempos de procesamiento se mantuvieron iguales para la heurística, lo cual no permitía al modelo revisar de manera exhaustiva más vecindarios posibles. Sin embargo, la diferencia estuvo cerca del 5% global.

Es importante resaltar que durante el proceso se encontraron diferencias en valores negativos. Estas representan un *makespan* obtenido por la heurística mejor que el valor encontrado por el software de optimización comercial. Esto se debe a la capacidad de generar lotes libres y no fijos de tal manera que se pueden conseguir resultados mejores a los conseguidos a través de un modelo de lotes fijos, como se pudo observar en la introducción y como se puede observar en la siguiente gráfica.

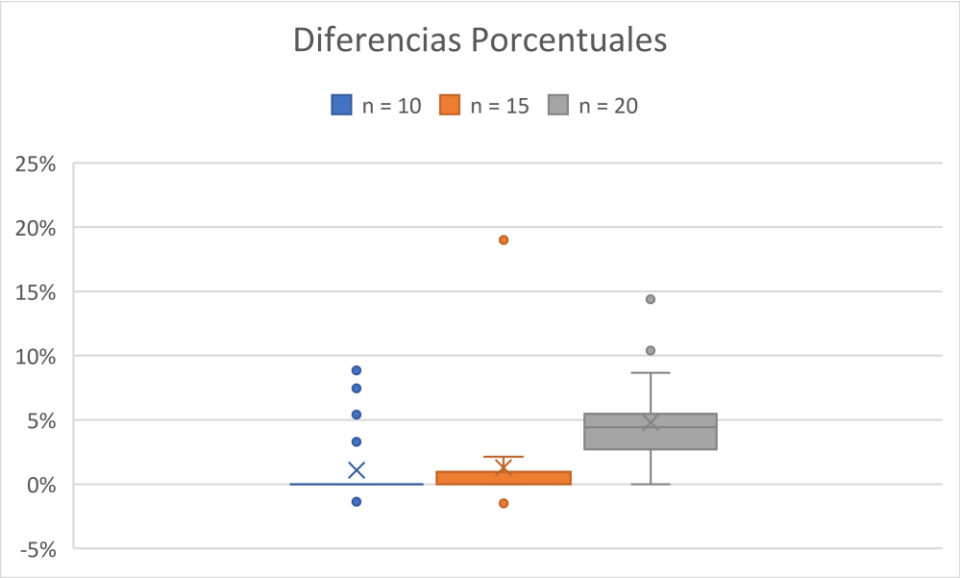

Figura 9. Diferencias porcentuales entre el software comercial y la BEV

CONCLUSIONES

En este documento se trabaja el problema de programación de producción de un sistema flow-shop a través de una búsqueda de entorno variables. Este tipo de problema ha venido llamando cada vez más la atención en la literatura dado los requerimientos de la industria hoy en día, particularmente la industria de semiconductores. Por lo anterior se hace necesario encontrar una manera económica en tiempos de procesamiento a través de heurísticas, las cual no necesariamente garanticen la solución óptima pero que si logran tener una solución de buena calidad en un tiempo razonable de procesamiento.

Los experimentos computacionales permiten concluir que el software comercial de optimización (XPRESS IVE 7.1®) logra mejores resultados que los conseguidos por la heurística, sin embargo la diferencia entre los modelos para 10 máquinas ronda entre 0.67% y 1.52%; lo cual se podría considerar poco significativo, ya que este último demora 30 minutos en conseguirse versus 6 horas de corrida del software de optimización, incluso según el tamaño de la compañía la diferencia entre 15 trabajos que ronda entre 0.07% y 2.49% se podría considerar poco significativa, pero cuando llegamos al caso de 20 trabajos vemos que estas diferencias se elevan considerablemente hasta llegar a una diferencia promedio de 5.89%. En este último caso las diferencias son especialmente grandes, y quizás con mayores tiempos de procesamiento de la heurística podríamos conseguir mejores resultados incluso que los del software comercial como se logró conseguir durante las pruebas. Por todo lo anterior se puede concluir que el heurístico propuesto es aplicable en industrias no muy grandes que no tienen los recursos para invertir en software de optimización ni recursos de tiempo para procesamiento pero que requieren de soluciones de buena calidad y cercanas a las ofrecidas por este tipo de programas.

BIBLIOGRAFÍA

[1] P. Damodaran y K. Srihari, «Mixed integer formulation to minimize makespan in a flow shop with batch processing machines,» *Mathematical and Computer Modelling,* vol. 40, nº 13, pp. 1465-1472, 2004.

[2] A. H. Kashan y B. Karimi, «An improved mixed integer linear formulation and lower bounds for minimizing makespan on a flow shop with batch processing machines,» *The International Journal of Advanced Manufacturing Technology,* vol. 40, p. 582, 2009.

[3] H. S. Mirsanei, K. B. y F. Jolai, «Flow shop scheduling with two batch processing machines and nonidentical job sizes,» *The International Journal of Advanced Manufacturing Technology,* vol. 45, pp. 553-572, 2009.

[4] P. K. Manjeshwar, P. Damodaran y K. Srihari, «Minimizing makespan in a flow shop with two batch-processing machines using simulated annealing,» *Robotics and Computer-Integrated Manufacturing,* vol. 25, nº 3, pp. 667-679, 2009.

[5] P. K. Manjeshwar, P. Damodaran y K. Srihari, «Genetic algorithms for minimizing makespan in a flow shop with two capacitated batch processing machines,» *The International Journal of Advanced Manufacturing Technology,* vol. 55, nº 9-12, pp. 1171-1182, 2011.

[6] S. Muthuswamy, M. C. Vélez-Gallego, J. Maya y M. Rojas-Santiago, «Minimizing makespan in a two-machine no-wait flow shop with batch processing machines,» *The International Journal of Advanced Manufacturing Technology,* vol. 63, nº 1-4, p. 281–290, 2012.

[7] P. Damodaran, A. G. Rao y S. Mestry, «Particle swarm optimization for scheduling batch processing machines in a permutation flowshop,» *The International Journal of Advanced Manufacturing Technology,* vol. 64, nº 5-8, p. 989–1000, 2013.

[8] J. Maya, S. Muthuswamy, M. C. Vélez-Gallego y M. Rojas-Santiago, «Minimising makespan in a no-wait flow shop with two batch processing machines: a grasp algorithm,» *International Journal of Industrial and Systems Engineering,* vol. 17, nº 2, 2014.

[9] H. Chen, Z. Shengchao, X. Li y R. Xu, «A hybrid differential evolution algorithm for a two-stage flow shop on batch processing machines with arbitrary release times and blocking,» *International Journal of Production Research,* vol. 52, nº 19, pp. 5714-5734, 2014.

[10] J.-D. Huang, J.-J. Liu, Q.-X. Chen y N. Mao, «Minimizing Makespan in a Flow Shop with Two Batch Machines,» de *22nd International Conference on Industrial Engineering and Engineering Management*, 2015.

[11] H. Matin, N. Salmasi y O. Shahvari, «Makespan minimization in flowshop batch processing problem with different batch compositions on machines,» *International Journal of Production Economics,* vol. 193, pp. 832-844, 2017.

[12] H. Mokhtari y A. Noroozi, «An efficient chaotic based PSO for earliness/tardiness optimization in a batch processing flow shop scheduling problem,» *Journal of Intelligent Manufacturing,* vol. 29, nº 5, p. 1063–1081, 2018.

[13] M. Nawaz, E. Enscore y I. Ham, «A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem,» *Omega,* vol. 11, nº 1, pp. 91-95, 1983.