# Algoritmos metaheurísticos híbridos para el problema de máquinas de procesamiento por lotes

Ana María Cortés[a,1], Juan Carlos Rivera[a,2,*]

[a] *Grupo de Investigación en Modelado Matemático*
*Departamento de Ciencias Matemáticas, Escuela de Ciencias, Universidad EAFIT*
*Medellín, Colombia*

**Resumen**

Una máquina de procesamiento por lotes (BPM por sus siglas en inglés), se caracteriza porque en ella es posible procesar múltiples trabajos simultáneamente. Este tipo de máquinas son comunes en procesos industriales como el recubrimiento electrolítico, tratamientos térmicos y hornos de secado. El BPM consiste en agrupar un conjunto trabajos en lotes para ser procesados en una máquina con una capacidad limitada, de manera que el tiempo necesario para la fabricación de todos los trabajos (makespan) sea mínimo. Cada trabajo se caracteriza por su tiempo de liberación, su tiempo mínimo de procesamiento y su tamaño. El BPM es un problema NP-Hard, razón por la que usualmente se aborda mediante metaheurísticos. En este trabajo se adaptan técnicas como el algoritmo de los ahorros, NEH, Large Neighborhood Search (LNS) y Split (routing-first cluster-second) para resolver el BPM. El desempeño de los algoritmos es evaluado utilizando instancias conocidas de la literatura cuyos tamaños varían entre 10 y 100 trabajos. Los algoritmos propuestos mejoran algunas de las mejores soluciones conocidas en la literatura.

*Palablas clave:* Optimización combinatoria, BPM, Algoritmos metaheurísticos, Algoritmo de los ahorros, NEH, LNS, Split.

## 1. Introducción

El problema de máquinas de procesamiento por lotes (BPM) es un problema típicamente conocido como un problema de optimización combinatoria. Una BPM es un tipo de máquina que puede procesar varios trabajos simultáneamente en un lote. Este tipo de máquinas son comunes en la industria manufacturera y en la industria metalúrgica en procesos de tratamientos térmicos, recubrimiento electrolítico, hornos de secado y procesos de pintura.

La búsqueda de metodologías de solución del BPM es considerado imperante a nivel práctico, teniendo en cuenta que generalmente dichas máquinas tienen un elevado costo; convirtiéndolas en un recurso restrictivo, es decir, una programación óptima de las BPM impacta directamente en la capacidad de producción real y en la eficiencia de los procesos. Desde el punto de vista teórico, dada su pertenencia a la clase NP-Hard de problemas, no existen algoritmos exactos que garanticen obtener soluciones óptimas en tiempo polinomial; por lo tanto es importante proponer algoritmos eficientes que obtengan soluciones cercanas al óptimo.

El problema consiste en programar un conjunto $J = \{1, ..., n\}$ de $n$ trabajos en una BPM con capacidad máxima $B$. Cada trabajo $j \in J$ es descrito por un tiempo de liberación $r_j$, un tiempo mínimo de procesamiento $p_j$ y un peso o tamaño $s_j$. Los trabajos pueden ser agrupados en lotes de tal manera que la suma de los pesos $w_j$ de los trabajos que los componen no superen la capacidad de la máquina $B$. El

*Corresponding author
*Email addresses:* acortesz@eafit.edu.co (Ana María Cortés), jrivera6@eafit.edu.co (Juan Carlos Rivera)
[1] Estudiante de Maestría en Matemáticas Aplicadas.
[2] Director de tesis.

tiempo de liberación y el tiempo de procesamiento de un lote se calcula como el máximo de los tiempos de liberación y el máximo de los tiempos de procesamiento de los trabajos que lo componen, respectivamente. De lo anterior, se entiende que los trabajos pueden ser procesados en un lote con tiempo de procesamiento mayor sin afectar el producto final. El objetivo es encontrar un programa $T$, i.e. tiempo de inicio y finalización para cada trabajo, de manera que el tiempo necesario para la fabricación de los $n$ trabajos ($C_{max}$) sea mínimo (Cheng et al. [3]).

Existen diversas variaciones fundamentales del problema, las cuales dependen principalmente de la consideración (o no) de los tiempos de liberación y los tamaños arbitrarios de los trabajos. Según lo anterior, y siguiendo la notación introducida por Graham *et al.* [11], éste problema puede ser dividido en las siguientes cuatro categorías principales:

- $1|batch, B|C_{max}$: No considera tiempos de liberación ni tamaños diferentes para los trabajos a programar, por lo que la máquina puede procesar hasta $B$ trabajos simultáneamente. (Ver Chandru et al. [2] y Boudhar [1])
- $1|batch, r_j, B|C_{max}$: Incluye tiempos de liberación pero no considera tamaños diferentes de los trabajos. (Ver Sung y Choung [27] y Poon y Zhang [20])
- $1|batch, \sum_j s_j \leq B|C_{max}$: Considera tamaños arbitrarios de los trabajos pero no tiempos de liberación. Dupont y Dhaenens-Flipo [10] y Damodaran et al. [8] estudian este caso.
- $1|batch, r_j, \sum_j s_j \leq B|C_{max}$: Considera tanto tiempos de liberación como tamaños arbitrarios para los trabajos.

En este artículo, nos enfocamos en el caso en el que se tienen en cuenta tanto diferentes tiempos de liberación como tamaños arbitrarios para cada uno de los trabajos ($1|batch, r_j, \sum_j s_j \leq B|C_{max}$). Este caso en particular ha sido uno de los menos estudiados en la literatura.

Uzsoy [29] demuestra que el problema $1|batch, \sum_j s_j \leq B|C_{max}$ es un problema NP-hard. De lo anterior se puede deducir que el problema $1|batch, r_j, \sum_j s_j \leq B|C_{max}$ también lo es. Por esta razón, para su solución es frecuente el uso de diversas metodologías heurísticas.

Li et al. [12] abordan el problema general con tiempos de liberación y tamaños arbitrarios, y desarrollan un algoritmo de aproximación $(2+\epsilon)$, donde $\epsilon$ puede ser arbitrariamente pequeño. Por su parte, Melouk et al. [17] proponen una solución a través de un algoritmo de Recocido Simulado (SA). Los tiempos de ejecución y las soluciones de SA son comparadas con los resultados de CPLEX, usando el modelo matemático propuesto por ellos, obteniendo mejores resultados en un tiempo de ejecución más corto.

Chou et al. [6] dividen el problema en dos fases. En la primera fase se usa un algoritmo genético (GA) para definir la secuencia de los trabajos. En la segunda fase los trabajos son agrupados en lotes por medio del algoritmo heurístico denomnado *First-Fit Longest Processing Time* (FFLPT).

Chou [4] utiliza programación dinámica en combinación con un algoritmo genético (GA + DP) para solucionar específicamente el problema de la programación de un horno de quemado. El enfoque propuesto se enmarca en el GA en el que se utiliza el algoritmo de programación dinámica para evaluar cada cromosoma y se aplican los operadores de GA para generar una variedad de secuencias de trabajos. Las soluciones son comparadas con los resultados reportados en Chou et al. [6], superándolos en todas las instancias.

Velez-Gallego et al. [30] proponen una extensión del heurístico SKP (*Succcessive Knapsack Problem*), introducido por Dupont y Ghazvini [9], como solución constructiva del problema $1|batch, \sum s_j \leq B|C_{max}$, y comparan los resultados con los del (GA+DP) [4].

Xu et al. [31] proponen un algoritmo metaheurístico de colonia de hormigas (ACO) en el cual, para hacer el algoritmo más eficiente y efectivo, introducen una estrategia de lista de candidatos que restringe el número de posibles candidatos elegibles a ser considerados en cada paso de construcción y un nuevo método para construir información heurística.

Adicionalmente a los estudios anteriormente mencionados, en la literatura se encuentran otras variantes del problema en las cuales se consideran familias de trabajos (los trabajos que pertenecen a la misma familia comparten el tiempo de procesamiento), tiempos de entrega o diferentes funciones objetivo (ver López Jiménez y Vélez Gallego [14], Mathirajan et al. [16] y Chou y Wang [5]).

En lo sucesivo, este artículo esta organizado de la siguiente manera: En la Sección 2 proponemos algunas modificaciones al modelo de *programación lineal entera mixta* presentado por Trindade et al. [28]. En la Sección 3 presentamos los diferentes algoritmos heurísticos y metaheurísticos propuestos. En la Sección 4 se resumen los resultados computacionales comparando las diferentes formulaciones y algoritmos propuestos con los resultados de la literatura. En la Sección 5 se presentan las conclusiones y futuras direcciones de investigación.

## 2. Formulación matemática

El BPM puede ser formulado matemáticamente como un modelo de *programación lineal entera mixta* (MILP). Diferentes modelos han sido propuestos en Velez-Gallego et al. [30], Xu et al. [31] y Li [13].

La siguiente formulación es presentada por Trindade et al. [28]. En dicha formulación se introducen restricciones de ruptura de simetrías, con las cuales se reduce el tamaño de la región factible. Dicho modelo asume que los trabajos están organizados en orden no decreciente de sus tiempos de liberación ($r_1 \leq r_2 \leq r_3 \leq \cdots \leq r_n$).

En dicha formulación matemática se asume la existencia de un conjunto $K = \{1, .., n\}$ de lotes de producción. El número de lotes es igual al número de trabajos. Se puede notar que, en el peor de los casos, cada lote puede ser compuesto por solo un trabajo.

Las variables de decisión binarias $x_{ik}$ toman el valor uno ($x_{ik} = 1$) si el trabajo $i \in J$ es procesado en el lote $k \in K$ y cero ($x_{ik} = 0$) en caso contrario. Para cada lote $k \in K$ se definen además las variables $P_k$ y $T_k$ las cuales indican el tiempo de procesamiento y el tiempo de inicio del lote, respectivamente.

Adicionalmente, Trindade et al. [28] demuestran que, dado un programa $T = \{T_1, T_2, ..., T_m\}$ y dos lotes $k$ y $k' \in K$ con los tiempos de inicio $T_k$ y $T_{k'}$, el orden de los lotes $T_k \leq T_{k'}$ que satisface $R_k \leq R_{k'}$ es óptimo, donde $R_k$ es el tiempo de liberación del lote $k$.

De lo anterior se puede deducir que, dado que los trabajos están organizados en orden no decreciente de su tiempo de libración, se pueden configurar los lotes de manera que todo lote $k \in K$ esté conformado por trabajos con tiempo de liberación inferior o igual a $r_k \; \forall \; k \in J$. Dado que, en general, habrán más lotes de los necesarios para resolver el problema, existirán lotes vacíos (sin trabajos asignados) cuyo tiempo de procesamiento es cero.

Los anteriores resultados son usados por Trindade et al. [28] en su formulación matemática descrita por las ecuaciones (1) a (10).

$$\min \; C_{max} = T_n + P_n \tag{1}$$

$$\text{s.a.} \sum_{\substack{k \in K \\ k \geq j}} x_{jk} = 1, \qquad \forall \; j \in J \tag{2}$$

$$\sum_{\substack{j \in J \\ j \leq k}} s_j \cdot x_{jk} \leq B \cdot x_{kk}, \qquad \forall \; k \in K \tag{3}$$

$$x_{jk} \leq x_{kk}, \qquad \forall \; j \in J, \; k \in K : j \leq k \tag{4}$$

$$P_k \geq p_j \cdot x_{jk}, \qquad \forall \; j \in J, \; k \in K : j \leq k \tag{5}$$

$$T_k \geq r_k \cdot x_{kk}, \qquad \forall \; k \in K \tag{6}$$

$$T_k \geq T_{k-1} + P_{k-1}, \qquad \forall \; k \in K : k > 1 \tag{7}$$

$$x_{jk} \in \{0, 1\}, \qquad \forall \; j \in J, \; k \in K : j \leq k \tag{8}$$

$$P_k \geq 0, \qquad \forall \; k \in K \tag{9}$$

$$T_k \geq 0, \qquad \forall \; k \in K \tag{10}$$

La función objetivo (1) minimiza el makespan o tiempo requerido para finalizar el procesamiento del último lote ($C_{max}$). Las restricciones (2) determinan que cada trabajo sea asignado a un solo lote, tal que

$k \geq j$. Las ecuaciones (3) representan las restricciones de capacidad, donde los lotes no pueden exceder la capacidad de la máquina. Adicionalmente estas ecuaciones garantizan que todo lote $k \in K$ esté compuesto por trabajos con índice inferior a $k$ ($j \leq k$). Por otro lado, si un lote $k \in K$ no es vacío, entonces el trabajo $k \in J$ es procesado en dicho lote. Las restricciones (4) son redundantes con las restricciones (3), pero son incluidas para fortalecer la relajación lineal del modelo. Las restricciones (5) limitan el mínimo tiempo de procesamiento de cada lote $k \in K$. Las restricciones (6) y (7) determinan el tiempo en el que inicia el procesamiento de cada lote $k \in K$. Las restricciones (6) limitan dicho tiempo al máximo tiempo de liberación de los trabajos que componen el lote respectivo. Por otro lado, las restricciones (7) limitan el el tiempo de inicio de un lote al tiempo de finalización del lote anterior. Finalmente, las restricciones (8) a (10) definen el dominio de las variables de decisión.

De acuerdo con Trindade et al. [28], computacionalmente este modelo es el mejor conocido en la literatura actualmente.

A continuación se proponen dos nuevas ecuaciones que modifican el modelo propuesto en [28] basadas en la idea de fortalecer la relajación lineal del modelo y romper algunas simetrías aún existentes. La función objetivo (11) reemplaza a la ecuación (1) mientras que las restricciones (12) son desigualdades válidas adicionales.

$$\min \; F = T_n + P_n + \frac{\sum_{j \in J} j \cdot \left( \sum_{k \in K} k \cdot x_{jk} \right)}{n^3} \tag{11}$$

$$T_k \geq T_{k-1} + p_j \cdot x_{j,k-1}, \qquad \forall \; j \in J, \; k \in K : k > 1, \; j < k \tag{12}$$

La función objetivo (11) adiciona el término $\sum_{j \in J} j \cdot \left( \sum_{k \in K} k \cdot x_{jk} \right) \Big/ n^3$ al makespan $C_{max}$, el cual toma valores en el intervalo $(0, 1)$ y permite al modelo diferenciar múltiples soluciones con igual $C_{max}$ y preferir aquellas en las cuales los trabajos con menor índice se procesan en los lotes con menor índice posible. Por otro lado, las desigualdades válidas (12), aunque son redundantes con las restricciones (7), fortalecen la relajación lineal del modelo. Se puede notar que, cuando los tiempos de liberación y los tiempos de procesamiento de cada trabajo son enteros, el $C_{max}$ tiene valores enteros y la nueva función objetivo garantiza que la solución óptima al modelo descrito por las ecuaciones (2) a (12) aún corresponde al $C_{max}$ óptimo.

En la Sección 4 se presenta una comparación entre los resultados obtenidos con cada modelo matemático.

## 3. Enfoques heurísticos

En esta sección se describen los algoritmos heurísticos (y metaheurísticos) propuestos para resolver el BPM. Entre los algoritmos propuestos se encuentran seis algoritmos constructivos y cinco variantes del metaheurístico LNS (Large Neighborhood Search). Adicionalmente se presenta una adaptación del algoritmo *split* (utilizado en problemas de ruteo de vehículos).

Con el objetivo de ejemplificar algunos procedimientos, en la Tabla 1 se presenta un ejemplo de BPM con $n = 6$ trabajos, cada uno con su peso $s$, tiempo de procesamiento $p$ y tiempo de liberación $r$. La capacidad de la maquina es $S = 12$.

Tabla 1: Ejemplo de BPM con $n = 6$ y $B = 12$

| **j** | **s** | **p** | **r** |
|---|---|---|---|
| 1 | 3 | 4 | 14 |
| 2 | 2 | 2 | 5 |
| 3 | 2 | 1 | 12 |
| 4 | 7 | 2 | 0 |
| 5 | 5 | 4 | 12 |
| 6 | 4 | 1 | 8 |

### *3.1. Algoritmo de los ahorros*

El algoritmo de los ahorros, fue propuesto por primera vez por Clarke y Wright [7] como una solución al problema de ruteo de vehículos (VRP). En dicho problema, dadas dos rutas determinadas, se define el ahorro, en terminos de distancia, como la diferencia entre el costo de realizar estas dos rutas por separado y costo al unirlas en una sola ruta. Como punto de partida, el algoritmo asume que cada nodo del grafo es visitado por un vehículo diferente.

Nosotros adaptamos el concepto de ahorro al BPM. El procedimiento inicia con $n$ lotes, cada uno compuesto por un trabajo. Cuando dos lotes se pueden procesar simultáneamente, es decir la suma del tamaño de ambos lotes no supera la capacidad de la máquina, se calcula el ahorro de acuerdo al tiempo de procesamiento total que se obtiene al procesar dos trabajos por separado o al procesarlos en un mismo lote. A continuación se selecciona un par de lotes cuyo ahorro sea lo mayor posible, se crea un nuevo lote producto de la unión de los lotes seleccionados y se actualizan los lotes disponibles y sus ahorros asociados. El Algoritmo 1 describe el procedimiento general. En dicho algoritmo, en las líneas 2 y 7 se calculan los ahorros de acuerdo a los procedimientos descritos en las Secciones 3.1.1 y 3.1.2; aquí se descartan los lotes infactibles y aquellos con ahorro menor o igual a cero.

**Algoritmo 1** – Ahorros

**Require:** $r, p, s, B, J$
1: $\Omega \leftarrow J$
2: $A \leftarrow Ahorros(\Omega, r, p, s, B)$
3: **while** $A \neq \varnothing$ **do**
4: $\quad (u, v) \leftarrow \max(A)$
5: $\quad k \leftarrow u \oplus v$
6: $\quad \Omega \leftarrow (\Omega \cup k) \setminus \{u, v\}$
7: $\quad A \leftarrow Actualizar(A, \Omega, r, p, s, B)$
8: **end while**
9: **return** $\Omega$

En nuestra adaptación, proponemos dos metodologías diferentes para calcular los ahorros $A_{u,v}$ entre dos lotes $u$ y $v$, las cuales se exponen a continuación.

#### *3.1.1. Método Ahorros 1 ($A_1$)*

Dados dos lotes, $u$ y $v$, con tiempos de procesamiento $P_u$ y $P_v$, y tiempos de liberación $R_u$ y $R_v$, se calcula el ahorro como la diferencia entre el mínimo tiempo de procesamiento al procesar los lotes por separado y el tiempo de procesamiento de los lotes unidos en un único lote. Note que dados un par de lotes, $u$ y $v$, existen dos programas válidos diferentes: programar $T_u < T_v$ y $T_u > T_v$. Las equaciones (13) a (16) permiten calcular los ahorros de acuerdo al procedimiento anterior, donde $P_{u,v}$ y $P_{v,u}$ son los tiempos totales de procesamiento al realizar primero el lote $u$ y luego el $v$ y viceversa, $P_{uv}$ es el tiempo de procesamiento al realizar los dos lotes en un mismo lote nuevo, y $A_{uv}$ es el ahorro generado por el nuevo lote.

$$P_{u,v} = \max((R_u + P_u, R_v) + P_v \tag{13}$$
$$P_{v,u} = \max((R_v + P_v, R_u) + P_u \tag{14}$$
$$P_{uv} = \max(R_u + R_v) + \max(P_u, P_v) \tag{15}$$
$$A_{u,v} = \min(P_{u,v}, P_{v,u}) - P_{uv} \tag{16}$$

Note que este procedimiento puede generar ahorros negativos. Se puede verificar que el ahorro entre los trabajos 2 y 3 del ejemplo en la Tabla 1 es -1.

*3.1.2. Método Ahorros 2 ($A_2$)*

Dados dos lotes, $u$ y $v$, con tiempos de procesamiento $P_u$ y $P_v$, y tiempos de inicio $T_u$ y $T_v$ (considerando tiempos de liberación) de modo que $T_u < T_v$, este método calcula el ahorro utilizando la ecuación (17).

$$A_{uv} = \begin{cases} R_u + P_u, & \text{Si } P_u > P_v \\ P_u, & \text{Si } P_u \leq P_v \end{cases} \tag{17}$$

Los métodos para calcular los ahorros, $A_1$ y $A_2$, calculan los ahorros para todas las combinaciones posibles de lotes en la línea 2 del Algoritmo 1. Luego de actualizar los lotes disponibles en la línea 6, se actualizan los ahorros eliminando todos los ahorros relativos a los lotes $u$ y $v$, y calculando los ahorros para el nuevo lote $k$.

*3.2. Algoritmo Split*

Este algoritmo, al igual que el de los ahorros, es una adaptación del algoritmo *Split* propuesto por primera vez por Prins [21] para el VRP, también conocido como *routing-first cluster-second*. Una descripción más detallada del algoritmo y sus aplicaciones puede ser encontrada en la revisión de literatura en Prins et al. [22].

En la versión tradicional del algoritmo, dada una sequencia de nodos, llamada "giant tour" o solución TSP (*Travelling Salesman Problem*), se obtiene la solución óptima al problema de determinar la inserción de las visitas al depósito, considerando las restricciones del problema, tales como las restricciones de capacidad y autonomía de los vehículos. El procedimiento se basa en la transformación del VRP original en un problema del camino más corto.

Análogmente, para el BPM, se transforma una secuencia de trabajos en un conjunto de lotes a ser procesados mediante la transformación del problema en un problema del camino más corto de la siguiente manera.

Sea un problema de BPM y una secuencia completa de trabajos $\sigma = (\sigma_0, \sigma_1, ..., \sigma_n)$. Se define un problema del camino más corto sobre un grafo dirigido $G = (J', E)$. El conjunto de nodos o trabajos $J' = J \cup 0$ contiene un nodo inicial (0) ficticio y $n$ trabajos indexados desde 1 hasta $n$. Cada trabajo $j \in J$ tiene un tamaño $s_j$, un tiempo de procesamiento $p_j$ y un tiempo de liberación $r_j$. El conjunto $E$ contiene los arcos $(i, j)$ de modo que $i < j$ y $\sum_{k=i+1}^{j} s_{\sigma_k} \leq B$. En el algoritmo *Split* para problemas de ruteo cada arco en $E$ corresponde a una ruta; en nuestro grafo cada arco $(i, j) \in E$ corresponde a un lote de producción compuesto por los trabajos $(\sigma_{i+1}, ..., \sigma_j)$.

En la Figura 1 se representa de manera gráfica el subproblema de camino más corto resultante con la secuencia de trabajos $\sigma = (0, 1, 2, 3, 4, 5, 6)$. Note que el grafo siempre inicia con el nodo 0, el cual es un trabajo ficticio usado para representar el nodo inicial del grafo del camino más corto. El nodo destino siempre es el nodo final de la secuencia $\sigma$. Cada arco $(i, j)$ representa un lote potencial y está representado

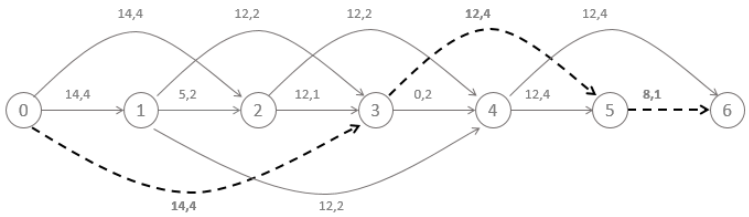


Figura 1: Ejemplo del problema de camino más corto como subproblema del BPM con $\sigma = (1, 2, 3, 4, 5, 6)$

por su tiempo de liberación y su tiempo de procesamiento $(R_{ij}, P_{ij})$. Por ejemplo, el arco (0,2) representa un lote que contiene los trabajos 1 y 2, con tiempo de liberación (calculado como el máximo tiempo de liberación de los trabajos 1 y 2) $R_{02} = \max\{14, 5\} = 14$, y tiempo de procesamiento (calculado como el máximo tiempo de procesamiento de los trabajos 1 y 2) $P_{02} = \max\{4, 2\} = 4$.

El algoritmo *Split* propuesto en este artículo, y representado por el Algoritmo 2, halla la solución óptima al subproblema del camino más corto representado por el grafo $G$ descrito anteriormente por medio de una adaptación al Algoritmo de Belmann-Ford. Dicho algoritmo explora cada nodo $i \in \sigma$, expande todos los arcos factibles $(i, j)$ $\forall\ j \in \sigma,\ j > i$, tales que $\sum_{k=i+1}^{j} s_{\sigma_k} \leq B$, y evalua el costo de alcanzar el nodo $j$, $V_j$, como $V_j = \min\{V_j, R_{ij} + P_{ij}, V_i + P_{ij}\}$. El algoritmo inicia con $V_j = \infty\ \forall j > 0$ y $V_0 = 0$. La ruta con menor valor $V_{\sigma_n}$ es aquella que representa el conjunto de lotes que obtiene el menor makespan $(C_{max})$.

El vector $I$ en el Algoritmo 2 permite identificar los trabajos pertenecientes a cada lote. Iniciando en la posición $n$, se sabe que el último lote termina con el trabajo en la posición $n$ de la secuencia $Seq$, e inicia en la posición $I_n + 1$ (note que $I_n$ indica el nodo inicial del arco). Así un lote que termine en la posición $i$ estará compuesto por todos los trabajos entre la posición $I_i + 1$ e $i$, $(Seq_{i+1}, Seq_{i+2}, ..., i-1, i)$.

En el ejemplo de la Figura 1, los arcos punteados y en negrita representa la solución óptima, resultando así una solución en la cual se tienen tres lotes ($\{1, 2, 3\}$, $\{4, 5\}$ y $\{6\}$), cuyo tiempo total de procesamiento (*makespan*) es 20.

Hasta donde tenemos conocimiento, esta es la primera aplicación o adaptación del algoritmo *Split* para problemas que no pertenecen a la familia de los problemas de ruteo de vehículos, VRP.

Este procedimiento puede ser utilizado sobre cualquier secuencia de trabajos obteniendo una solución factible. En particular, las soluciones obtenidas con los algoritmos *Ahorros 1* y *Ahorros 2* se pueden convertir en secuencias equivalentes. En dichas secuencias el algoritmo *Split* siempre obtiene una solución igual o mejor.

En la Sección 4 se comparan las soluciones obtenidas con los algoritmos de los ahorros y la obtenida luego de aplicar split a estas soluciones.

### *3.3. Algoritmo NEH/Split*

El algoritmo *NEH* fue propuesto por Nawaz et al. [18] para resolver el Flow-Shop Scheduling Problem (FSP). Actualmente es uno de los mejores procedimientos para resolver el Permutation Flow-Shop Scheduling Problem (PFSP) y el Blocking Flow-Shop Scheduling Problem (BFSP) [23].

Dicho algoritmo inicia ordenando los trabajos de acuerdo a la regla LPT (Longest Processing Time). A continuación se toman los dos primeros trabajos de la lista ordenada y se programan de manera que el makespan sea mínimo. Luego para $k$ desde 3 hasta $n$ se inserta el $k$-ésimo trabajo en la posición que genere el menor makespan posible.

En el algoritmo adaptado para BPM, *NEH/Split*, la evaluación del makespan es realizada utilizando el Algoritmo *Split* descrito en la Sección 3.2. El procedimiento puede ser resumido en el Algoritmo 3.

Una variante adicional consiste en invertir el orden en que se añaden los trabajos a la secuencia $\pi$. Esta versión inicia con el nodo $n$ y en cada iteración $k$ añade el nodo $n - k$ a la secuencia. En la Sección 4 este procedimiento es denominado *NEH/Split* $\times$ *2* e incluye la inserción de nodos en ambos sentidos.

**Algoritmo 2** – Split

**Require:** Seq,r,p,s,B,n.

```
for i = 1 to n + 1 do
    V_i ← ∞
end for
V_1 ← 0
for i = 1 to n + 1 do
    j ← 1
    L ← 0, C ← 0, P ← 0, R ← 0
    while j < n + 1 and L + s_{Seq_{j+1}} ≤ B do
        j ← j + 1
        L ← L + s_{Seq_j}
        if P < p_{Seq_j} then
            P ← p_{Seq_j}
        end if
        if R < r_{Seq_j} then
            R ← r_{Seq_j}
        end if
        if R > V_i then
            C ← R + P
        else
            C ← V_i + P
        end if
        if V_j > C then
            V_j ← C
            I_j ← i
        end if
    end while
end for
return V_n, I
```

### *3.4. Large Neighborhood Search (LNS)*

El algoritmo Large Neighborhood Search (LNS) fue propuesto por Shaw [26]. Este método hace parte de la familia de metaheurísticos conocidos como *Very Large Scale Neighborhood search* (VLSN). La idea principal del LNS es que un vecindario amplio permite al heurístico navegar por el espacio de soluciones fácilmente.

En el metaheurístico LNS, se mejora gradualmente una solución inicial alternando destrucciones y reparaciones de la misma. Dichas destrucciones y reparaciones se definen por un método de destrucción (*Destroy*) y uno de reparación (*Repair*). El método de destrucción destruye parte de la solución actual; libera algunas variables y deja las otras congeladas en sus valores actuales. Con frecuencia este método incluye aleatoriedad de tal forma que la parte de la solución que se destruye en cada iteración del método sea diferente. El método de reparación reconstruye la solución destruida.

El LNS adaptativo (ALNS) utiliza múltiples operadores de destrucción y reparación [25]. En cada iteración se selecciona un método de destrucción y uno de reparación. La probabilidad de seleccionar cada método se asigna de manera dinámica de acuerdo a la historia de la búsqueda.

Nuestro LNS propuesto considera el uso de múltiples operadores de destrucción y reparación. Sin embargo, las probabilidades de selección son fijas durantes todo la ejecución del mismo.

El algoritmo 4 presenta un pseudocódigo del LNS propuesto en este artículo. En dicho pseudocódigo, la variable $T$ representa la solución inicial (o actual), $T^b$ es la mejor solución encontrada durante la búsqueda, y $T^t$ representa una solución temporal que puede ser descartada o promovida (aceptada) como solución

**Algoritmo 3** – NEH

**Require:** J,r,p,s,B,n.

1: $O \leftarrow LPT(J, p)$
2: $\pi \leftarrow O_1$
3: $k \leftarrow 2$
4: **while** $k \leq n$ **do**
5: $\quad C^b_{max} \leftarrow \infty$
6: $\quad$ **for** $i = 1$ **to** $k$ **do**
7: $\quad\quad \pi^t \leftarrow Insertar(\pi, O_k, i)$
8: $\quad\quad$ **if** $C_{max}(\pi^t) < C^b_{max}$ **then**
9: $\quad\quad\quad C^b_{max} \leftarrow C_{max}(\pi^t)$
10: $\quad\quad\quad \pi^b \leftarrow \pi^t$
11: $\quad\quad$ **end if**
12: $\quad$ **end for**
13: $\quad \pi \leftarrow \pi^b$
14: $\quad k \leftarrow k + 1$
15: **end while**
16: **return** $\pi, C^b_{max}$

actual.

Las funciones $d(\cdot)$ y $r(\cdot)$ son los métodos de destrucción y reparación respectivamente, de tal forma que $d(T)$ retorna una copia de una solución $T$ parcialmente destruida, y $r(d(T))$ devuelve una solución factible creada a partir de la solución destruida.

En la línea 1 se inicializa la mejor solución. En la línea 3, se aplican los métodos de destrucción y de reparación para obtener una nueva solución $T^t$. En la línea 5 se evalúa la nueva solución y se determina si esta solución debe convertirse en la nueva solución actual (línea 6) o si debe ser rechazada por medio de la función *Aceptar*. En la línea 8 se comprueba si la nueva solución es mejor que la mejor solución conocida ($c(T)$ denota el valor de la función objetivo dada la solución $T$). En la línea 9 se actualiza la mejor solución. En la línea 12 se devuelve la mejor solución encontrada.

Nuestro LNS toma como solución inicial la solución obtenida por el Algoritmo *Ahorros 1* (ver sección 3.1.1). El criterio de parada seleccionado es el número total de iteraciones a realizar.

El método de destrucción es un elemento crucial del LNS. De acuerdo con Pisinger y Ropke [19], lo más importante al aplicar este método es la selección del grado de destrucción, pues si se destruye una pequeña parte de la solución, entonces el algoritmo puede tener problemas para explorar el espacio de búsqueda y se

**Algoritmo 4** – LNS$^+$

**Require:** $T$

$T^b = T$
**while** Criterio de parada = falso **do**
$\quad T^t = r(d(T))$
$\quad T^t = Split(T^t)$
$\quad$ **if** $Aceptar(T^t, T)$ **then**
$\quad\quad T = T^t$
$\quad$ **end if**
$\quad$ **if** $c(T^t) < c(T^b)$ **then**
$\quad\quad T^b = T^t$
$\quad$ **end if**
**end while**
**return** $T^b$

perdería el efecto de vecindarios amplios; y si se destruye una parte muy grande de la solución, el algoritmo podría tomarse mucho tiempo o, dependiendo de cómo se repara la solución, producir soluciones de baja calidad.

Se consideran, de forma probabilista, cinco posibles operadores de destrucción, cada una con la misma probabilidad de ser seleccionada. A continuación, se realiza una breve descripción de dichos operadores.

- *Destrucción del trabajo mas pesado (de acuerdo a $s_j$):* en este operador se examinan los trabajos dentro de un lote, eliminando de la solución el trabajo $j$ con mayor peso o tamaño, $s_j$.
- *Destrucción del trabajo con mayor tiempo de procesamiento (de acuerdo a $p_j$):* en este operador se examinan los trabajos dentro de un lote, eliminando de la solución el trabajo con mayor tiempo de procesamiento.
- *Destrucción del trabajo con mayor volumen:* en este operador se examinan los trabajos dentro de un lote, eliminando de la solución el trabajo con mayor volumen. Aquí, nosotros definimos el volumen de un trabajo como la multiplicación entre el tamaño y el tiempo de procesamiento ($s_j \cdot p_j$).
- *Destrucción aleatoria:* Se elimina aleatoriamente un trabajo.
- *Destrucción aleatoria de un lote:* se eliminan aleatoriamente todos los trabajos de un lote.

En los operadores de destrucción descritos anteriormente, el grado de destrucción es directamente proporcional a la proporción de trabajos eliminados de una solución.

Los operadores de *Reparación* son procedimientos orientados a optimizar las variables liberadas por los operadores de *Destrucción*. Aquí hemos implementado diferentes procedimientos constructivos que insertan uno a uno los trabajos eliminados por los procedimientos de destrucción. En este artículo se consideran tres funciones de *Reparación* las cuales se describen a continuación:

- *Menor desperdicio:* se insertan los trabajos en el lote en el cual quede menos espacio libre después de ser insertado el trabajo.
- *Primer posible:* se insertan los trabajos en el primer lote que sea factible la inserción.
- *Mejor inserción:* se insertan los trabajos en el lote que menos afecte negativamente la función objetivo.

Una vez aplicado el operador de reparación se obtiene una solución completa y factible $T^t$. En nuestra versión del LNS (denominada LNS$^+$), se utiliza el algoritmo *Split* para intentar mejorar dicha solución. En la Sección 4 se comparan los resultados de dicho algoritmo con el LNS en el cual no se aplica el Algoritmo *Split*.

Dado que los operadores de reparación insertan los trabajos uno a uno a partir de un vector que contine los trabajos eliminados, se consideró reordenar dicho vector antes de aplicar los operadores de reparación. Se utilizaron tres formas de ordenar:

- Órden decreciente de acuerdo a su tamaño ($s_j$)
- Órden creciente de acuerdo a su tiempo de liberación ($r_j$).
- Órden decreciente de acuerdo a su tiempo de procesamieto ($p_j$).

Debido a que no se encontraron diferencias significativas en la calidad de la solución, dichos procedimientos de ordenamiento no son considerados en los reportes de resultados. Por otro lado, estos procedimientos tenían un efecto negativo en el tiempo de cómputo.

La función *Aceptar* puede ser implementada de diversas formas. La opción más sencilla es la propuesta originalmente en Shaw [26], la cual consiste en aceptar únicamente las soluciones de mejora. Esta estrategia es la utilizado en nuestra implementación. Otros estudios como Ropke y Pisinger [25] y Ribeiro y Laporte [24] proponen funciones de aceptación probabilísticas similares a la usada en el metaheurístico *Recocido Simulado* (SA – Simulated Annealing).

### *3.5. LNS Iterado (ILNS)*

En esta sección proponemos una nueva variante del Algoritmo LNS. El *ILNS* (*Iterated Large Neighborhood Search* o *LNS Iterado*) es un híbrido entre los algoritmos LNS e ILS (*Iterated Local Search* [15]).

La principal diferencia con el LNS (Algoritmo 4) es la presencia del operador *Perturbar*, el cual permite añadir modificaciones aleatorias a la solución incluso si la función objetivo no mejora.

**Algoritmo 5** – ILNS

**Require:** $T$
1: $T \leftarrow Split(T)$
2: $T^b \leftarrow LNS^+(T)$
3: **while** Criterio de parada = false **do**
4: $\quad T \leftarrow Perturbar(T^b)$
5: $\quad T^t \leftarrow LNS^+(T)$
6: $\quad$ **if** $c(T^t) < c(T^b)$ **then**
7: $\quad\quad T^b = T^t$
8: $\quad$ **end if**
9: **end while**
10: **return** $T^b$

En nuestra version del algoritmo ILNS, el operador *Perturbar* interviene la solución generada por el algoritmo split, de tal forma que esta solución es destruida y reparada alaetoriamente, es decir, los trabajos eliminados se ingresan de nuevo a la solución en una posicion aleatoria, siempre y cuando se cumpla con la restricción de capacidad. El grado de destrucción es un número fijo de trabajos.

### *3.6. LNS basado en NEH y Split*

En esta sección se propone una nueva implementación del metaheurístico LNS. A diferencia del LNS descrito en la Sección 3.4, este algoritmo siempre opera sobre una secuencia de trabajos (sin tener en cuenta los lotes ni tiempos de procesamiento de los trabajos).

Se usa un solo operador de *Destrucción*, el cual elimina trabajos aleatoriamente (*LNS/NEH 1*). De manera alternativa, se puede usar un procedimiento de exploración que evalue sistemáticamente todas las alternativas posibles de eliminación de trabajos (*LNS/NEH 2*). Para la *Reparación* de la solución se utiliza una variación del algoritmo NEH descrito en la Sección 3.3, en la cual no se inicia con una secuencia de dos nodos sino con la solución parcial obtenida del procedimiento de destrucción y se analiza la inserción de los nodos eliminados.

## 4. Resultados

En esta sección de comparan los resultados obtenidos por los diferentes métodos propuestos entre ellos y con los resultados reportados en la literatura.

Para una mejor comprensión de los mismos, se han dividido en las siguientes subsecciones: En la Subsección 4.1 se describen las instancias de prueba utilizadas para comparar los resultados. En la Subsección 4.2 se compara el desempeño de las modificaciones propuestas en la formulación matemática. La Subsección 4.3 compara los resultados obtenidos con los diferentes métodos constructivos. Finalmente, la Subsección 4.4 presenta los resultados obtenidos con las diferentes variaciones del metaheurístico LNS.

Todas las pruebas fueron realizadas en un computador con procesador Intel core-i7 (2.60GHz) y 16 GB de RAM. Los modelos matemáticos fueron programados en Gurobi 8.0.1. y los algoritmos heurísticos en Python 2.7.

### *4.1. Instancias*

Las instancias de prueba utilizadas en este artículo son las mismas consideradas en Xu et al. [31], las cuales fueron generadas aleatoriamente de acuerdo a la metodología propuesta en Chou et al. [6].

Dichas instancias tienen en cuenta diferentes tamaños del problema ($n = 10,\ 20,\ 50$ y $100$). El tiempo de procesamiento $p_j$ se obtiene a partir de una distribución uniforme en el intervalo $[8, 48]$. El tiempo de liberación $r_j$ es generado a partir de una distribución uniforme en $[0, LB]$, donde $LB$ es una cota inferior propuesta por Xu et al. [31], se consideran dos tipos de problema según los tamaños de los trabajos $s_j$, los cuales pueden ser pequeños o grandes de acuerdo a una distribución uniforme en los intervalos $[1, 30]$ o $[15, 35]$ respectivamente. La capacidad de la máquina es igual para todos los casos ($B = 40$).

### *4.2. Métodos Exactos*

En esta sección se comparan los resultados obtenidos con la formulación matemática propuesta por Trindade et al. [28] y la formulación obtenida con las modificaciones propuestas utilizado las ecuaciones (11) y (12) (ver Sección 2). Aquí, ambas formulaciones son ejecutadas en el mismo computador usando Gurobi 8.0.1.

En la Tabla 2 se resumen los resultados obtenidos. Las dos primeras columnas indican el tipo de problema: tamaño ($n$) e intervalo del tamaño de los trabajos (Tipo). Los intervalos para la generación de los tamaños de los trabajos $s$ se clasifican con los valores 1 (intervalo $[1, 30]$) y 2 (intervalo $[15, 35]$). Para cada tipo de instancia, los resultados mostrados son el promedio de 10 instancias diferentes. Las columnas 3 y 5 presentan el Gap porcentual promedio obtenido por Gurobi para el modelo propuesto por Trindade et al. [28] ($M_1$) y el modelo con los cambios propuestos en este artículo ($M_2$), respectivamente. Las columnas 4 y 6 presentan los tiempos de cómputo (en segundos) para $M_1$ y $M_2$, respectivamente.

Se puede observar que ambas formulaciones obtienen una solución óptima para todos los problemas de prueba, obteniendo un Gap de 0%. En cuanto al tiempo de cómputo, las ecuaciones propuestas en la Sección 2 mejoran el tiempo promedio en 9 segundos, 6.45%. Con base en los tiempos de cómputo, se puede observar que las instancias de Tipo 1 tienen un grado de complejidad más alto que las de Tipo 2: El modelo $M_2$ pasa de 0.11 segundos en promedio para las instancias de tipo 2 a 260.49 segundos en promedio para las instancias de tipo 1.

### *4.3. Métodos Constructivos*

En esta sección se comparan los resultados obtenidos con los diferentes métodos constructivos propuestos: *Ahorros 1*, *Ahorros 2*, *Ahorros 1 + Split*, *Ahorros 2 + Split*, *NEH/Split*.

En la Tabla 3 se resumen los resultados obtenidos. Al igual que en la Tabla 2, las dos primeras columnas indican el tipo de problema. Para cada método constructivo se indica el Gap porcentual promedio con respecto a la solución óptima y el tiempo de cómputo promedio en segundos.

Tabla 2: Resumen de resultados con modelos matemáticos

| Instancia | | $M_1$ | | $M_2$ | |
|---|---|---|---|---|---|
| $n$ | Tipo | Gap | Tiempo | Gap | Tiempo |
| 10 | 1 | 0.00 | 0.01 | 0.00 | 0.01 |
| 10 | 2 | 0.00 | 0.00 | 0.00 | 0.00 |
| 20 | 1 | 0.00 | 0.06 | 0.00 | 0.09 |
| 20 | 2 | 0.00 | 0.00 | 0.00 | 0.00 |
| 50 | 1 | 0.00 | 378.80 | 0.00 | 368.85 |
| 50 | 2 | 0.00 | 0.15 | 0.00 | 0.07 |
| 100 | 1 | 0.00 | 736.84 | 0.00 | 673.00 |
| 100 | 2 | 0.00 | 0.31 | 0.00 | 0.36 |
| Promedio | | 0.00 | 139.52 | 0.00 | 130.30 |

Tabla 3: Resumen de resultados con algoritmos constructivos

| Instancia | | $A_1$ | | $A_2$ | | $A_1 + Split$ | | $A_2 + Split$ | | NEH/*Split* | | NEH/*Split*×2 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $n$ | Tipo | Gap | Tiempo | Gap | Tiempo | Gap | Tiempo | Gap | Tiempo | Gap | Tiempo | Gap | Tiempo |
| 10 | 1 | 3.87 | 0.00 | 7.46 | 0.00 | 2.35 | 0.00 | 7.29 | 0.00 | 1.93 | 0.00 | 0.88 | 0.00 |
| 10 | 2 | 1.73 | 0.00 | 1.90 | 0.00 | 1.07 | 0.00 | 1.90 | 0.00 | 1.40 | 0.00 | 0.80 | 0.00 |
| 20 | 1 | 6.84 | 0.00 | 11.88 | 0.00 | 3.91 | 0.01 | 10.13 | 0.01 | 3.29 | 0.01 | 1.07 | 0.00 |
| 20 | 2 | 1.46 | 0.00 | 2.60 | 0.00 | 1.46 | 0.00 | 1.96 | 0.00 | 1.83 | 0.00 | 1.04 | 0.00 |
| 50 | 1 | 12.44 | 0.24 | 11.57 | 0.25 | 9.48 | 0.24 | 11.04 | 0.25 | 5.32 | 0.03 | 4.30 | 0.03 |
| *50* | 2 | 2.90 | 0.04 | 3.75 | 0.04 | 2.72 | 0.04 | 3.27 | 0.04 | 2.21 | 0.04 | 1.94 | 0.04 |
| 100 | 1 | 10.66 | 6.81 | 8.42 | 7.52 | 8.57 | 6.81 | 7.48 | 7.58 | 3.76 | 0.15 | 3.61 | 0.14 |
| 100 | 2 | 2.41 | 1.05 | 2.35 | 1.07 | 2.34 | 1.05 | 2.20 | 1.08 | 1.22 | 0.32 | 1.12 | 0.33 |
| Promedio | | 5.29 | 1.02 | 6.24 | 1.11 | 3.99 | 1.02 | 5.66 | 1.12 | 2.62 | 0.07 | 1.85 | 0.07 |

Con respecto a los métodos basados en los ahorros, $A_1$ y $A_2$, se puede observar que el primer método obtiene en promedio menor Gap y menor tiempo promedio. El tiempo promedio es inferior a 7 segundos para todos los tipos de instancias y el Gap es inferior a 12.44%. Sin embargo, para las instancias de 50 y 100 trabajos, en los casos en los que $s_j$ es pequeño, el algoritmo $A_2$ mejora el Gap promedio con respecto al algoritmo $A_1$.

Con respecto al algoritmo Split, éste permite mejorar el Gap promedio en todos los tipos de instancias. El tiempo de cómputo no se modifica significativamente, mostrando la gran eficiencia del algoritmo. El mejor resultado, considerando tiempo y calidad, es el obtenido por el método $A_1$ + *Split*.

Con relación a los algoritmos basados en el método *NEH*, su desempeño es superior a los anteriores tanto en Gap como en tiempo de cómputo. El algoritmo NEH/*Split*×2 es el algoritmo constructivo con el mejor Gap y mejor tiempo de cómputo. El Gap es inferior a 4.30% para todos los tipos de instancias.

### *4.4. Variantes de LNS*

En esta sección se presentan los resultados obtenidos por medio de los diferentes algoritmos basados en el metaheurístico LNS.

En la Tabla 4 se resumen los resultados obtenidos. Las dos primeras columnas hacen referencia al tipo de problema (de la misma forma que en la Tabla 2). Para cada algoritmo implementado, se presenta el Gap porcentual promedio con respecto a la solución óptima y el tiempo de cómputo promedio en segundos.

La fase de destrucción de los algoritmos LNS, LNS$^+$ e ILNS tienen un grado de destrucción que depende del tamaño del problema, de tal forma que se destruyen el valor máximo entre 5 trabajos y el 20% del

Tabla 4: Resumen de resultados de las variantes del LNS

| Instancia | | LNS | | LNS$^+$ | | ILNS | | LNS/NEH 1 | | LNS/NEH 2 | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $n$ | Tipo | Gap | Tiempo | Gap | Tiempo | Gap | Tiempo | Gap | Tiempo | Gap | Tiempo |
| 10 | 1 | 0.00 | 0.34 | 0.00 | 0.89 | 0.00 | 0.89 | 0.00 | 0.04 | 0.00 | 0.02 |
| 10 | 2 | 0.00 | 0.21 | 0.00 | 0.95 | 0.00 | 0.95 | 0.00 | 0.03 | 0.00 | 0.02 |
| 20 | 1 | 2.53 | 0.38 | 0.42 | 1.52 | 0.03 | 1.75 | 0.48 | 0.09 | 0.00 | 0.12 |
| 20 | 2 | 0.17 | 0.28 | 0.15 | 1.89 | 0.00 | 2.08 | 0.54 | 0.08 | 0.06 | 0.12 |
| 50 | 1 | 8.64 | 1.20 | 6.89 | 5.11 | 4.94 | 5.34 | 3.44 | 0.37 | 1.02 | 9.67 |
| 50 | 2 | 1.33 | 0.60 | 1.18 | 6.32 | 0.66 | 6.56 | 1.11 | 0.52 | 0.09 | 10.53 |
| 100 | 1 | 8.57 | 11.38 | 7.88 | 23.84 | 6.28 | 24.78 | 3.46 | 1.01 | 1.00 | 122.02 |
| 100 | 2 | 2.11 | 3.05 | 1.76 | 21.87 | 1.16 | 23.18 | 1.09 | 2.37 | 0.17 | 233.43 |
| Promedio | | 2.92 | 2.18 | 2.29 | 7.80 | 1.63 | 8.19 | 1.26 | 0.56 | 0.29 | 46.99 |

número de trabajos del problema ($\max\{5, 0.2 \cdot n\}$). El criterio de parada usado en todos los algoritmos es de 100 iteraciones.

El metahuerístico LNS mejora los resultados obtenidos por su solución inicial (algoritmo de los ahorros $A_1$) con un tiempo de cómputo adicional aproximado de 1.16 segundos.

De manera similar, el metaheurístico denominado LNS$^+$ logra mejorar los resultados obtenidos por su solución inicial y por el LNS. Ambos algoritmos logran encontrar todas las soluciones óptimas (Gap = 0%) para las instancias con $n = 10$. El tiempo de cómputo del algoritmo LNS$^+$ es 167% superior al del LNS. Sin embargo, el Gap promedio de ambos métodos es aún superior al del algoritmo NEH/*Split*×2.

El algoritmo ILNS mejora el Gap obtenido con el algoritmo NEH/*Split*×2. Además de las instancias con $n = 10$, éste halla las soluciones óptimas para todos los problemas con $n = 20$ de tipo 2.

Para el algoritmo LNS/NEH 1, debido al espacio de solución que utiliza, el grado de destrucción es un número de trabajos predeterminado. Los resultados en la Tabla 4 son obtenidos eliminando 3 trabajos en cada operación de destrucción. Dicho algoritmo permite alcanzar un Gap promedio de 1.26% en 0.56 segundos.

Finalmente, el algoritmo LNS/NEH 2 es el que obtiene el menor Gap promedio. Aunque el tiempo de cómputo es el mayor, 233 segundos (4 minutos aproximadamente) es aún bajo para efectos prácticos. Para las instancias pequeñas, $n = 10$ y $n = 20$, los algoritmos LNS/NEH 1 y LNS/NEH 2 presentan los menores tiempos de cómputo. El algoritmo LNS/NEH 1 presenta el menor tiempo de cómputo en promedio.

De manera similar que en la Tabla 2, con los algoritmos heurísticos constructivos y de búsqueda, las instancias de tipo 1 tienden a ser más difíciles presentando un mayor Gap.

### *4.5. Comparación con resultados de la literatura*

En las Tablas 5 a 8 se comparan los resultados obtenidos por los algoritmos propuestos con los resultados reportados por Xu et al. [31]. En [31] solo se publican los resultados para 5 instancias de cada tipo identificadas como J$m$St-$i$, donde $m$ identifica el número de trabajos de la instancia, $t$ identifica el rango de generación del tamaño de cada trabajo e $i$ es un contador. Los valores de $m$ varían entre 1 y 4 para identificar los valores de $n \in \{10, 20, 50, 100\}$, respectivamente, $t$ toma los valores 1 o 2, e $i$ varía entre 1 y 5; i.e. J3S2-1 representa la instancia $i = 1$ con $n = 50$ trabajos de tipo 2.

En las Tablas 5, 6, 7 y 8 se comparan los resultados obtenidos con los métodos basados en LNS con el algoritmo de colonias de hormigas (ACO, *Ant Colony Optimization*), de Xu et al. [31], para las instancias con $n = 10, 20, 50$ y 100, respectivamente.

En la Tabla 5 se puede observar que todos los métodos obtienen las soluciones óptimas para las 10 instancias reportadas. Los algoritmos LNS/NEH 1 y LNS/NEH 2 reportan los menores tiempos de cómputo.

Tabla 5: Comparación de resultados de los métodos basados en LNS y [31] para instancias con $n = 10$

| Instancia | ACO [31] | | LNS | | LNS$^+$ | | ILNS | | LNS/NEH 1 | | LNS/NEH 2 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Gap | Tiempo | Gap | Tiempo | Gap | Tiempo | Gap | Tiempo | Gap | Tiempo | Gap | Tiempo |
| J1S1-1 | 0.00 | 0.01 | 0.00 | 0.39 | 0.00 | 0.80 | 0.00 | 0.84 | 0.00 | 0.05 | 0.00 | 0.08 |
| J1S1-2 | 0.00 | 0.23 | 0.00 | 0.24 | 0.00 | 0.77 | 0.00 | 0.91 | 0.00 | 0.09 | 0.00 | 0.03 |
| J1S1-3 | 0.00 | 0.01 | 0.00 | 0.30 | 0.00 | 0.74 | 0.00 | 0.89 | 0.00 | 0.03 | 0.00 | 0.01 |
| J1S1-4 | 0.00 | 0.01 | 0.00 | 0.25 | 0.00 | 0.76 | 0.00 | 0.83 | 0.00 | 0.02 | 0.00 | 0.01 |
| J1S1-5 | 0.00 | 0.23 | 0.00 | 0.29 | 0.00 | 0.72 | 0.00 | 0.92 | 0.00 | 0.03 | 0.00 | 0.01 |
| Promedio | 0.00 | 0.10 | 0.00 | 0.29 | 0.00 | 0.76 | 0.00 | 0.88 | 0.00 | 0.04 | 0.00 | 0.03 |
| J1S2-1 | 0.00 | 0.20 | 0.00 | 0.24 | 0.00 | 0.76 | 0.00 | 0.88 | 0.00 | 0.04 | 0.00 | 0.06 |
| J1S2-2 | 0.00 | 0.00 | 0.00 | 0.16 | 0.00 | 0.81 | 0.00 | 1.00 | 0.00 | 0.03 | 0.00 | 0.01 |
| J1S2-3 | 0.00 | 0.22 | 0.00 | 0.25 | 0.00 | 0.85 | 0.00 | 0.88 | 0.00 | 0.03 | 0.00 | 0.01 |
| J1S2-4 | 0.00 | 0.20 | 0.00 | 0.16 | 0.00 | 0.78 | 0.00 | 1.06 | 0.00 | 0.03 | 0.00 | 0.02 |
| J1S2-5 | 0.00 | 0.20 | 0.00 | 0.15 | 0.00 | 0.81 | 0.00 | 1.01 | 0.00 | 0.03 | 0.00 | 0.01 |
| Promedio | 0.00 | 0.16 | 0.00 | 0.19 | 0.00 | 0.80 | 0.00 | 0.96 | 0.00 | 0.03 | 0.00 | 0.02 |

Tabla 6: Comparación de resultados de los métodos basados en LNS y [31] para instancias con $n = 20$

| Instancia | ACO [31] | | LNS | | LNS$^+$ | | ILNS | | LNS/NEH 1 | | LNS/NEH 2 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Gap | Tiempo | Gap | Tiempo | Gap | Tiempo | Gap | Tiempo | Gap | Tiempo | Gap | Tiempo |
| J2S1-1 | 0.00 | 0.78 | 4.75 | 0.32 | 0.00 | 1.53 | 0.00 | 1.58 | 0.00 | 0.09 | 0.00 | 0.10 |
| J2S1-2 | 0.00 | 0.81 | 3.77 | 0.32 | 0.00 | 1.27 | 0.00 | 1.95 | 0.00 | 0.07 | 0.00 | 0.07 |
| J2S1-3 | 1.39 | 0.85 | 1.39 | 0.27 | 0.69 | 1.25 | 0.00 | 1.59 | 0.69 | 0.08 | 0.00 | 0.19 |
| J2S1-4 | 2.60 | 1.11 | 5.19 | 0.31 | 2.08 | 1.32 | 0.00 | 1.65 | 0.00 | 0.07 | 0.00 | 0.08 |
| J2S1-5 | 0.00 | 0.09 | 3.24 | 0.27 | 1.47 | 1.33 | 0.29 | 1.65 | 0.29 | 0.07 | 0.00 | 0.14 |
| Promedio | 0.80 | 0.73 | 3.67 | 0.29 | 0.85 | 1.34 | 0.06 | 1.68 | 0.20 | 0.07 | 0.00 | 0.12 |
| J2S2-1 | 0.00 | 0.40 | 0.60 | 0.24 | 0.00 | 1.45 | 0.00 | 1.70 | 0.00 | 0.13 | 0.60 | 0.09 |
| J2S2-2 | 0.00 | 0.27 | 0.00 | 0.15 | 0.00 | 1.72 | 0.00 | 2.01 | 0.00 | 0.09 | 0.00 | 0.23 |
| J2S2-3 | 0.00 | 0.30 | 0.16 | 0.23 | 0.00 | 1.54 | 0.00 | 2.41 | 0.00 | 0.07 | 0.00 | 0.09 |
| J2S2-4 | 0.00 | 0.27 | 0.00 | 0.25 | 0.00 | 1.73 | 0.00 | 1.92 | 0.00 | 0.07 | 0.00 | 0.08 |
| J2S2-5 | 0.00 | 0.28 | 0.00 | 0.19 | 0.00 | 1.59 | 0.00 | 2.08 | 0.00 | 0.08 | 0.00 | 0.10 |
| Promedio | 0.00 | 0.30 | 0.15 | 0.21 | 0.00 | 1.60 | 0.00 | 2.02 | 0.00 | 0.09 | 0.12 | 0.12 |

Tabla 7: Comparación de resultados de los métodos basados en LNS y [31] para instancias con $n = 50$

| Instancia | ACO [31] | | LNS | | LNS$^+$ | | ILNS | | LNS/NEH 1 | | LNS/NEH 2 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Gap | Tiempo | Gap | Tiempo | Gap | Tiempo | Gap | Tiempo | Gap | Tiempo | Gap | Tiempo |
| J3S1-1 | 1.96 | 5.13 | 13.25 | 1.06 | 8.53 | 4.26 | 5.76 | 5.62 | 4.49 | 0.33 | 0.81 | 8.35 |
| J3S1-2 | 8.84 | 6.16 | 13.83 | 1.01 | 15.18 | 4.32 | 12.98 | 5.11 | 8.16 | 0.31 | 3.40 | 16.33 |
| J3S1-3 | 4.65 | 4.94 | 10.36 | 0.93 | 6.66 | 4.53 | 6.13 | 5.47 | 5.07 | 0.29 | 0.74 | 12.65 |
| J3S1-4 | 5.35 | 6.78 | 10.82 | 1.11 | 7.30 | 4.27 | 5.87 | 6.36 | 1.96 | 0.28 | 0.39 | 5.63 |
| J3S1-5 | 3.78 | 9.13 | 3.65 | 1.11 | 3.65 | 4.39 | 1.89 | 5.07 | 1.89 | 0.25 | 1.89 | 1.37 |
| Promedio | 4.92 | 6.43 | 10.38 | 1.05 | 8.26 | 4.36 | 6.53 | 5.53 | 4.32 | 0.29 | 1.45 | 8.87 |
| J3S2-1 | 0.00 | 2.24 | 1.18 | 0.44 | 0.21 | 5.52 | 0.00 | 6.33 | 1.04 | 0.33 | 0.00 | 4.45 |
| J3S2-2 | 0.48 | 4.00 | 3.53 | 0.56 | 4.97 | 5.33 | 3.21 | 6.51 | 3.21 | 0.39 | 0.80 | 17.50 |
| J3S2-3 | 0.00 | 2.36 | 0.44 | 0.56 | 0.07 | 5.11 | 0.29 | 6.44 | 0.80 | 0.31 | 0.00 | 9.55 |
| J3S2-4 | 0.00 | 1.90 | 0.00 | 0.46 | 0.00 | 5.52 | 0.00 | 6.66 | 0.00 | 0.31 | 0.00 | 1.34 |
| J3S2-5 | 0.24 | 2.53 | 1.43 | 0.52 | 0.00 | 5.41 | 0.00 | 6.76 | 0.00 | 0.30 | 0.00 | 2.95 |
| Promedio | 0.14 | 2.61 | 1.32 | 0.51 | 1.05 | 5.38 | 0.70 | 6.54 | 1.01 | 0.33 | 0.16 | 7.16 |

Tabla 8: Comparación de resultados de los métodos basados en LNS y [31] para instancias con $n = 100$

| Instancia | ACO [31] | | LNS | | LNS$^+$ | | ILNS | | LNS/NEH 1 | | LNS/NEH 2 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Gap | Tiempo | Gap | Tiempo | Gap | Tiempo | Gap | Tiempo | Gap | Tiempo | Gap | Tiempo |
| J4S1-1 | 3.40 | 20.61 | 12.20 | 9.27 | 10.44 | 20.04 | 9.94 | 26.51 | 2.08 | 22.78 | 1.19 | 148.06 |
| J4S1-2 | 6.78 | 28.11 | 8.00 | 8.70 | 7.23 | 19.61 | 5.05 | 24.57 | 0.00 | 38.65 | 0.00 | 65.53 |
| J4S1-3 | 9.00 | 26.38 | 9.15 | 9.85 | 8.83 | 20.05 | 7.38 | 26.12 | 2.62 | 111.32 | 1.60 | 270.60 |
| J4S1-4 | 9.66 | 22.61 | 19.19 | 9.59 | 18.12 | 20.13 | 15.64 | 24.02 | 6.11 | 78.31 | 3.49 | 216.39 |
| J4S1-5 | 3.99 | 21.16 | 8.28 | 9.61 | 7.64 | 20.02 | 5.56 | 23.26 | 1.45 | 69.32 | 1.22 | 111.26 |
| Promedio | 6.57 | 23.77 | 11.36 | 9.41 | 10.45 | 19.97 | 8.71 | 24.89 | 2.45 | 64.08 | 1.50 | 162.37 |
| J4S2-1 | 2.40 | 13.89 | 5.38 | 2.13 | 6.58 | 17.50 | 4.49 | 20.21 | 1.24 | 136.27 | 1.24 | 300.23 |
| J4S2-2 | 0.00 | 6.31 | 0.00 | 2.23 | 0.00 | 19.15 | 0.00 | 23.19 | 0.00 | 4.36 | 0.00 | 7.25 |
| J4S2-3 | 0.00 | 7.64 | 0.00 | 2.80 | 0.00 | 19.41 | 0.00 | 22.76 | 0.00 | 5.44 | 0.00 | 5.61 |
| J4S2-4 | 0.04 | 12.31 | 2.68 | 2.41 | 2.09 | 17.47 | 1.22 | 21.86 | 0.00 | 104.65 | 0.00 | 121.36 |
| J4S2-5 | 0.61 | 13.38 | 2.19 | 3.25 | 1.88 | 17.64 | 0.84 | 22.37 | 0.00 | 43.34 | 0.00 | 67.62 |
| Promedio | 0.61 | 10.71 | 2.05 | 2.57 | 2.11 | 18.23 | 1.31 | 22.08 | 0.25 | 58.81 | 0.25 | 100.41 |

De manera similar, de la Tabla 6, los algoritmos LNS/NEH 1 y LNS/NEH 2 obtienen los menores tiempos de cómputo para instancias con $n = 20$. El algoritmo LNS/NEH obtiene el menor Gap promedio, seguido por el ILNS.

Para las instancias con $n = 50$, ver Tabla 7, el Gap promedio continua siendo menor. Por otro lado, el tiempo de cómputo del método LNS/NEH 2 se incrementa significativamente. Los tiempos de cómputo de los algoritmos ACO y LNS/NEH 1 no presentan diferencias significativas.

Finalmente, el tiempo de cómputo para las instancias con $n = 100$ trabajos es significativamente superior con todos los métodos propuestos en este artículo en comparación con ACO. Por otro lado, los Gap promedio de los métodos LNS/NEH 1 y LNS/NEH 2 son 62% y 75% menores respectivamente.

Los algoritmos LNS/NEH 1 y LNS/NEH 2 también obtienen el mayor número de soluciones óptimas, 29 y 28 respectivamente, en comparación con las 23 soluciones óptmas obtenidas por ACO en [31].

## 5. Conclusiones

En este artículo se estudia el problema conocido como *batch processing machine* – BPM. El problema consiste en agrupar un conjunto de trabajos en lotes y asignar tiempos de inicio y finalización a cada lote de modo que el makespan ($C_{max}$) sea mínimo.

Se han propuesto algunas modificaciones al modelo de *programación lineal entera mixta* presentado por Trindade et al. [28]. En dichas modificaciones se adiciona un nuevo término en la función objetivo, permitiendo al modelo diferenciar múltiples soluciones con igual $C_{max}$ y preferir aquellas en las cuales los trabajos con menor índice se procesan en los lotes con menor índice posible. Adicionalmente, se incluyen nuevas restricciones que fortalecen la relajación lineal del modelo. Con estas modificaciones se logra una mejora en el tiempo promedio de cómputo del modelo.

Adicionalmente, se proponen seis algoritmos heurísticos constructivos y cinco algoritmos metaheurísticos basados en LNS. Los algoritmos constructivos están basados en adaptaciones de los algoritmos de los *ahorros* [7] y del método *split* [22], originalmente diseñados para resolver problemas de ruteo de vehículos – VRP. El método *split* es aplicado en combinación con el procedimiento de los *ahorros* y una adaptación del algoritmo *NEH* [18]. El algoritmo denominado NEH/Split$\times$2 es el que presenta las mejores propiedades en términos de Gap y tiempo de cómputo promedio.

Los métodos basados en LNS permiten mejorar las soluciones obtenidas por los métodos constructivos. Los métodos LNS/NEH 1 y LNS/NEH 2 presentan los mejores resultados en términos del Gap promedio. El tiempo de cómputo también es menor que el de ACO publicado en [31] en las intancias más pequeñas ($n = 10, 20$ y 50).

Tanto con los métodos exactos como con los procedimientos heurísticos y metaheurísticos, se encuentra una diferencia significativa en los desempeños de los algoritmos para las instancias de tipo 1 y de tipo 2. Las instancias de tipo 1, las cuales presentan tamaños de los trabajos más pequeños, son más difíciles de resolver.

Como direcciones de investigación futura se propone mejorar la eficiencia de los algoritmos para disminuir el tiempo de cómputo. Se deben generar nuevas instancias de prueba para evaluar los algoritmos heurísticos con instancias más grandes. Por otro lado, las estrategias de solución propuestas pueden ser aplicadas a otros problemas de *scheduling*; en particular, pueden ser aplicadas a problemas de procesamiento por lotes con múltiples máquinas.

## Referencias


[1] Mourad Boudhar. Scheduling on a batch processing machine with split compatibility graphs. *Journal of Mathematical Modelling and Algorithms*, 4(4):391–407, 2005. doi: 10.1007/s10852-005-3083-z.

[2] Vijaya Chandru, Chung-Yee Lee, y Reha Uzsoy. Minimizing total completion time on a batch processing machine with job families. *Operations Research Letters*, 13(2):61–65, 1993. doi: 10.1016/0167-6377(93)90030-K.

[3] T.C.E. Cheng, C.T. Ng, J.J. Yuan, y Z.H. Liu. Single machine scheduling to minimize total weighted tardiness. *European Journal of Operational Research*, 165(2):423–443, 2005. doi: 10.1016/j.ejor.2004.04.013.

[4] Fuh Der Chou. A joint GA+DP approach for single burn-in oven scheduling problems with makespan criterion. *International Journal of Advanced Manufacturing Technology*, 35(5-6):587–595, 2007. doi: 10.1007/s00170-006-0738-5.

[5] Fuh-Der Chou y Hui-Mei Wang. Scheduling for a single semiconductor batch-processing machine to minimize total weighted tardiness. *Journal of the Chinese Institute of Industrial Engineers*, 25(2):136–147, 2008. doi: 10.1080/10170660809509079.

[6] Fuh Der Chou, Pei Chann Chang, y Hui Mei Wang. A hybrid genetic algorithm to minimize makespan for the single batch machine dynamic scheduling problem. *International Journal of Advanced Manufacturing Technology*, 31(3-4):350–359, 2006. doi: 10.1007/s00170-005-0194-7.

[7] G Clarke y J W Wright. Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, 12(4):568–581, 1964. doi: 10.1287/opre.12.4.568.

[8] Purushothaman Damodaran, Praveen Kumar Manjeshwar, y Krishnaswami Srihari. Minimizing makespan on a batch-processing machine with non-identical job sizes using genetic algorithms. *International Journal of Production Economics*, 103(2):882–891, 2006. doi: 10.1016/j.ijpe.2006.02.010.

[9] L. Dupont y F.J. Ghazvini. Minimizing makespan on a single batch processing machine with non-identical job sizes. *European Journal of Automation*, 32:431–440, 1998.

[10] Lionel Dupont y Clarisse Dhaenens-Flipo. Minimizing the makespan on a batch machine with non-identical job sizes: An exact procedure. *Computers & Operations Research*, 29(7):807–819, 2002.

[11] R.L. Graham, E.L. Lawler, J.K. Lenstra, y A.H.G.Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. In P.L. Hammer, E.L. Johnson, y B.H. Korte, editors, *Discrete Optimization II*, volume 5 of *Annals of Discrete Mathematics*, pages 287–326. Elsevier, 1979. doi: 10.1016/S0167-5060(08)70356-X.

[12] Shuguang Li, Guojun Li, Xiaoli Wang, y Qiming Liu. Minimizing makespan on a single batching machine with release times and non-identical job sizes. *Operations Research Letters*, 33(2):157–164, 2005. doi: 10.1016/j.orl.2004.04.009.

[13] Xiaolin Li. Scheduling batch processing machine using max – min ant system algorithm improve by a local search method. *Hindawi*, 2018. doi: 10.1155/2018/3124182.

[14] Juan Diego López Jiménez y Mario César Vélez Gallego. Algoritmo de búsqueda de entorno variable para minimizar la tardanza total ponderada en una máquina de procesamiento por lotes. *Revista Ingeniería Industrial*, 10(1):5–18, 2011.

[15] Helena R. Lourenço, Olivier C. Martin, y Thomas Stützle. *Iterated Local Search: Framework and Applications*, pages 363–397. Springer US, 2010.

[16] Muthu Mathirajan, V. Bhargav, y V. Ramachandran. Minimizing total weighted tardiness on a batch-processing machine with non-agreeable release times and due dates. *The International Journal of Advanced Manufacturing Technology*, 48 (9):1133–1148, 2010. doi: 10.1007/s00170-009-2342-y.

[17] Sharif Melouk, Purushothaman Damodaran, y Ping Yu Chang. Minimizing makespan for single machine batch processing with non-identical job sizes using simulated annealing. *International Journal of Production Economics*, 87(2):141–147, 2004. doi: 10.1016/S0925-5273(03)00092-6.

[18] Muhammad Nawaz, E Emory Enscore, y Inyong Ham. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega*, 11(1):91–95, 1983. doi: 10.1016/0305-0483(83)90088-9.

[19] David Pisinger y Stefan Ropke. *Large Neighborhood Search*, pages 399–419. Springer US, 2010.

[20] Chung Keung Poon y Pixing Zhang. Minimizing makespan in batch machine scheduling. In Gerhard Goos, Juris Hartmanis, Jan van Leeuwen, D. T. Lee, y Shang-Hua Teng, editors, *Algorithms and Computation*, pages 386–397. Springer Berlin Heidelberg, 2000.

[21] Christian Prins. A simple and effective evolutionary algorithm for the vehicle routing problem. *Computers & Operations Research*, 31:1985–2002, 2004. doi: 10.1016/S0305-0548(03)00158-8.

[22] Christian Prins, Philippe Lacomme, y Caroline Prodhon. Order-first split-second methods for vehicle routing problems: A review. *Transportation Research Part C*, 40:179–200, 2014. doi: 10.1016/j.trc.2014.01.011.

[23] Imma Ribas y Manel Mateo. Improvement tools for neh based heuristics on permutation and blocking flow shop scheduling problems. In Bruno Vallespir y Thècle Alix, editors, *Advances in Production Management Systems. New Challenges, New Approaches*, pages 33–40, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[24] Glaydston Mattos Ribeiro y Gilbert Laporte. An adaptive large neighborhood search heuristic for the cumulative capacitated vehicle routing problem. *Computers & Operations Research*, 39(3):728–735, 2012. doi: 10.1016/j.cor.2011.05.005.

[25] Stefan Ropke y David Pisinger. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science*, 40(4):455–472, 2006. doi: 10.1287/trsc.1050.0135.

[26] Paul Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In *Proceedings of the 4th International Conference on Principles and Practice of Constraint Programming*, pages 417–431. Springer-Verlag, 1998.

[27] C.S. Sung y Y.I. Choung. Minimizing makespan on a single burn-in oven in semiconductor manufacturing. *European Journal of Operational Research*, 120(3):559–574, 2000. doi: 10.1016/S0377-2217(98)00391-9.

[28] Renan Spencer Trindade, Olinto César, Bassi De Araújo, Marcia Helena Costa, y Felipe Martins Müller. Modelling and symmetry breaking in scheduling problems on batch processing machines. *International Journal of Production Research*, 7543:1–18, 2018. doi: 10.1080/00207543.2018.1424371.

[29] R. Uzsoy. Scheduling a batch processing machine with non-identical job sizes. *International Journal of Production Research*, 38(10):2173–2184, 2000. doi: 10.1080/00207540050028034.

[30] Mario Velez-Gallego, Purushothaman Damodaran, y Manuel Rodríguez. Makespan minimization on a single batch processing machine with unequal job ready times. *International Journal of Industrial Engineering: Theory, Applications and Practice*, 18(10), 2011.

[31] Rui Xu, Huaping Chen, y Xueping Li. Makespan minimization on single batch-processing machine via ant colony optimization. *Computers & Operations Research*, 39(3):582–593, 2012. doi: 10.1016/j.cor.2011.05.011.