DESDE 2013
https://repository.uaeh.edu.mx/revistas/index.php/icbi/issue/archive
Pädi Boletín Científico de Ciencias Básicas e Ingenierías del ICBI

ISSN: 2007-6363

Publicación Semestral Pädi Vol. 14 No. Especial 3 (2026) 339-351

# Algoritmo basado en aprendizaje probabilístico para la optimización del makespan en el problema de programación en taller de flujo
# Algorithm based on probabilistic learning for makespan optimization in the flow shop scheduling problem

C. Acosta-Acosta [a], J. Medina-Marín [a], M.G. Serna-Díaz [b], G.E. Anaya-Fuentes [a], M.P. González-Trinidad [a,*]

[a] *Área Académica de Ingeniería y Arquitectura, Universidad Autónoma del Estado de Hidalgo, 42184, Pachuca, Hidalgo, México.*
[b] *Escuela Preparatoria No.1, Universidad Autónoma del Estado de Hidalgo, 42060, Pachuca, Hidalgo, México.*

**Resumen**

En este artículo se presenta la propuesta de un algoritmo metaheurístico basado en aprendizaje probabilístico adaptativo para la búsqueda de permutaciones con el propósito de obtener el valor mínimo del *makespan* en el problema de programación en taller de flujo (*Flow Shop Scheduling Problem*, FSSP). A diferencia de los métodos basados en poblaciones, como la optimización por enjambre de partículas o los algoritmos genéticos, el método propuesto emplea un mecanismo de aprendizaje probabilístico sobre las asignaciones en la posición de los trabajos, combinado con una perturbación adaptativa y una búsqueda local selectiva. La estructura del algoritmo se enfoca en una única solución, permitiendo generar soluciones competitivas mediante un mecanismo de aprendizaje probabilístico adaptativo. Los resultados experimentales muestran que el algoritmo propuesto alcanzó una eficiencia global del 96 % respecto a los mejores valores de *makespan* obtenidos, logró el valor óptimo en diversas instancias y mantuvo un desempeño competitivo frente a los algoritmos evaluados para la minimización del *makespan* en el *Flow Shop Scheduling Problem*.

*Palabras Clave:* FSSP, Algoritmos metaheurísticos, aprendizaje adaptativo.

**Abstract**

This article presents a metaheuristic algorithm based on adaptive probabilistic learning for permutation construction, aimed at minimizing the *makespan* in the *Flow Shop Scheduling Problem* (FSSP). Unlike population-based methods, such as Particle Swarm Optimization (PSO) and Genetic Algorithms (GA), the proposed method employs a probabilistic learning mechanism to model job-position assignments, combined with an adaptive perturbation strategy and a selective local search procedure. The algorithm follows a single-solution approach, enabling the generation of competitive solutions through the continuous update of probabilistic job-position assignments. Experimental results show that the proposed algorithm achieved an overall efficiency of 96% with respect to the best *makespan* values obtained, reached the optimal solution in several benchmark instances, and maintained competitive performance compared with the evaluated algorithms for *makespan* minimization in the *Flow Shop Scheduling Problem*.

*Keywords: FSSP, metaheuristic algorithms, adaptive learning.*

## 1. Introducción

La programación de la producción ha adquirido una creciente relevancia en diversos sectores industriales, tales como la manufactura, el transporte y las comunicaciones, debido a su impacto directo en la optimización de los tiempos de entrega, la eficiencia operativa y la utilización eficiente de los recursos (Silva Rodríguez et al., 2017).

De acuerdo con Chase y Jacobs (2010), este campo se enfoca en la asignación y secuenciación de recursos y

*Autor para la correspondencia: go380108@uaeh.edu.mx
**Correo electrónico**: ac440904@uaeh.edu.mx (Citlali Acosta-Acosta), jmedina@uaeh.edu.mx (Joselito Medina-Marín), maría_serna10342@uaeh.edu.mx (María Guadalupe Serna-Díaz) ganaya@uaeh.edu.mx (Gustavo Erick Anaya-Fuentes) go380108@uaeh.edu.mx (Monica Paola Gonzalez-Trinidad)

**Historial del manuscrito:** recibido el 04/06/2026, última versión-revisada recibida el 17/08/2026, aceptado el 07/09/2026, publicado el 06/11/2026. **DOI:** https://doi.org/10.29057/icbi.v14iEspecial3.18078

CC BY NC ND

actividades, con el objetivo de mejorar el desempeño de los sistemas productivos.

En este contexto, diversos autores han investigado y propuesto enfoques para abordar el problema de programación considerando recursos clave como maquinaria, materia prima, mano de obra y tareas. Estas contribuciones han permitido desarrollar metodologías orientadas a optimizar la asignación y secuenciación de los recursos productivos, mejorando indicadores de desempeño como el tiempo de procesamiento y la utilización de los recursos (González-Neira et al., 2017).

Entre los problemas de programación más estudiados se encuentra el Flow Shop Scheduling Problem (FSSP), debido a su alta relevancia en entornos manufactureros y a la complejidad que representa su resolución. En este sistema se consideran $n$ trabajos que deben ser procesados en $m$ máquinas siguiendo una secuencia fija y predeterminada, donde la decisión principal radica en determinar el orden óptimo de los trabajos para minimizar el tiempo total de ejecución o *makespan* (Ruiz y Maroto, 2005). Bajo este modelo, cada máquina solo puede atender un trabajo a la vez, el procesamiento no puede ser interrumpido y un trabajo no puede iniciar en la siguiente máquina sin haber finalizado completamente en la anterior (Ruiz y Maroto, 2005).

Gupta y Stafford (2006) afirman que el modelo de optimización ha cobrado mayor relevancia en las últimas décadas y los procesos productivos se han vuelto más complejos debido a la amplia aplicación de la automatización en la industria, por lo que, frecuentemente se busca la optimización del FSSP por medio de algoritmos, lo cual contribuye a una mejora de resultados, desempeñando un papel fundamental debido a la alta complejidad computacional que presentan. Por ello, las heurísticas y metaheurísticas permiten obtener soluciones cercanas al óptimo en tiempos razonables, optimizando indicadores clave como el *makespan*, los tiempos de espera y la utilización de recursos (Ruiz y Maroto, 2005). Estos enfoques destacan por su flexibilidad y capacidad de adaptación a condiciones reales de los sistemas productivos, lo que los convierte en herramientas esenciales para la toma de decisiones en la programación de la producción y la mejora de la eficiencia operativa (Babor et al., 2021).

Entre los métodos más representativos se encuentra la heurística NEH, propuesta como una estrategia eficiente de construcción de secuencias iniciales, así como enfoques metaheurísticos como Simulated Annealing (SA), Particle Swarm Optimization (PSO), Ant Colony Optimization (ACO), algoritmos genéticos (GA) e Iterated Greedy (IG), los cuales han demostrado obtener soluciones competitivas para la minimización del *makespan* en diferentes escenarios. En particular, los algoritmos basados en búsqueda iterativa han destacado por su capacidad para combinar mecanismos de destrucción y reconstrucción de soluciones con estrategias de mejora local. Sin embargo, diversos estudios coinciden en que el desempeño de estos enfoques depende de las características particulares de cada instancia, por lo que ningún algoritmo mantiene un comportamiento consistentemente superior para todos los tamaños y configuraciones del problema (Ruiz y Maroto, 2005).

Como una alternativa a estas limitaciones surgieron los Estimation of Distribution Algorithms (EDA), los cuales sustituyen los operadores tradicionales de recombinación y mutación por modelos probabilísticos capaces de aprender las características de las mejores soluciones encontradas durante el proceso evolutivo (Mühlenbein y Paaß, 1996). Gracias a esta estrategia, los EDA han demostrado ser una herramienta competitiva para resolver diversos problemas de optimización combinatoria. Sin embargo, en el contexto del FSSP, uno de los principales desafíos de estos algoritmos consiste en mantener un equilibrio adecuado entre la convergencia hacia regiones prometedoras y la preservación de la diversidad en el espacio de búsqueda. Puesto que el modelo probabilístico suele actualizarse únicamente a partir de la población seleccionada en cada iteración, se genera una concentración excesiva de probabilidades que limita la exploración y favorece la convergencia prematura (Lemtenneche et al., 2023). Por esta razón, resulta relevante desarrollar estrategias adaptativas que modifiquen dinámicamente la distribución de probabilidad durante el proceso evolutivo.

Con base en lo anterior, el presente trabajo propone un algoritmo de aprendizaje adaptativo basado en la estructura de población por capas de edad (ALPS, Hornby, 2006) para el FSSP. El método emplea una matriz de probabilidades para construir soluciones, incorpora un mecanismo de actualización incremental del modelo probabilístico mediante una tasa de aprendizaje, aplica una perturbación controlada directamente sobre la matriz de probabilidades para favorecer la exploración del espacio de búsqueda y complementa el proceso con una búsqueda local selectiva basada en un operador de inserción. La integración de estos mecanismos busca mejorar la calidad de las soluciones obtenidas para el FSSP manteniendo un equilibrio entre exploración y explotación durante el proceso de optimización.

El resto del artículo está estructurado de la siguiente manera. La Sección 2 presenta el estado del arte sobre las principales propuestas de solución para el problema Flow Shop que sobresalen en la literatura. En la Sección 3 se describe la formulación del problema FSSP considerado, mientras que en la Sección 4 se abordan los fundamentos de los algoritmos metaheurísticos utilizados como referencia. En la Sección 5 se detalla el algoritmo propuesto; posteriormente, en la Sección 6 se analizan los resultados experimentales obtenidos y, finalmente, en la Sección 7 se presentan las conclusiones.

## 2. Estado del arte

Johnson (1954) resolvió de manera definitiva la programación óptima para sistemas *Flow Shop* de dos máquinas y extensión a tres máquinas bajo condiciones específicas mediante una regla combinatoria basada en tiempos mínimos de procesamiento. El desafío de este hallazgo radicó en haber delimitado la frontera de la optimización exacta; a medida que los sistemas industriales reales añaden más estaciones y restricciones, el problema adquiere una complejidad exponencial, se ha demostrado que la secuenciación para tres o más máquinas es *NP-hard* (problemas cuya resolución óptima presenta una complejidad computacional elevada) (Garey et al., 1976). Esto marcó el punto de partida para el desarrollo de las metaheurísticas contemporáneas.

En este contexto surge el *Flow Shop Scheduling Problem* (FSSP), uno de los problemas clásicos de optimización combinatoria en la investigación de operaciones debido a su alta aplicabilidad industrial y a la elevada complejidad computacional que presenta. El objetivo del FSSP es determinar el orden en que un conjunto de trabajos debe procesarse en una serie de máquinas con el fin de optimizar un

criterio de desempeño, generalmente la minimización del tiempo total de finalización *(makespan)* (Najarro et al., 2018).

Ante la complejidad del FSSP, se han desarrollado métodos heurísticos capaces de generar soluciones cercanas al óptimo con tiempos computacionales reducidos, destacando los trabajos de Palmer (1965), Campbell et al. (1970), Gupta (1971) y Dannenbring (1977). No obstante, la contribución más influyente corresponde al algoritmo propuesto por Nawaz, Enscore y Ham (1983), conocido como *NEH*, el cual es considerado uno de los algoritmos constructivos más exitosos para el problema de Flow Shop. Su estrategia consiste en ordenar inicialmente los trabajos de acuerdo con sus tiempos totales de procesamiento e incorporarlos progresivamente en la posición que minimiza el *makespan* (Ruiz & Maroto, 2005).

Debido al incremento de escenarios de problemas industriales más complejos, la investigación evolucionó a la incorporación de metaheurísticas capaces de explorar espacios de búsqueda considerablemente mayores. Algoritmos como los *Genetic Algorithms* (AG) desarrollados por Holland (1975), *Simulated Annealing* (SA) de Kirkpatrick et al. (1983), *Ant Colony Optimization* (ACO) de Dorigo et al. (1996) y *Particle Swarm Optimization* (PSO) propuesto por Kennedy y Eberhart (1995) han sido estudiados para el FSSP, demostrando ser aptos para explorar espacios de búsqueda considerablemente mayores, obteniendo soluciones de alta calidad en tiempos razonables. Para evaluar y comparar de manera uniforme el rendimiento de estos desarrollos, Taillard (1993) propuso un conjunto de instancias de prueba que rápidamente se convirtió en el *benchmark* estándar para la evaluación comparativa de algoritmos destinados al FSSP.

Más adelante, Ruiz y Stützle (2007) demostraron que la incorporación de mecanismos de intensificación y diversificación mediante búsqueda local iterativa *Iterated Greedy* (IG) permite incrementar considerablemente el desempeño de las metaheurísticas para problemas de secuenciación de gran escala. Sus resultados evidenciaron que el equilibrio entre exploración y explotación constituye uno de los factores más importantes para obtener soluciones competitivas en problemas NP-hard como el FSSP.

Pese a los resultados obtenidos por los algoritmos y heurísticas consolidadas como IG, SA, PSO, ACO, AG y NEH, la literatura de optimización demuestra que no existe un método único capaz de obtener soluciones superiores en todos los escenarios del FSSP. Mientras que las heurísticas constructivas, como NEH, destacan por su eficiencia computacional y capacidad para generar soluciones iniciales de alta calidad, las metaheurísticas permiten explorar espacios de búsqueda más amplios mediante mecanismos de intensificación y diversificación. Por ello, el área se mantiene en constante evolución, promoviendo la integración de nuevas estrategias capaces de superar las limitaciones de los enfoques existentes —particularmente el estancamiento prematuro en óptimos locales—. En este sentido, el algoritmo *Age-Layered Population Structure* (ALPS), propuesto originalmente por Hornby (2006) para preservar la diversidad en sistemas evolutivos, surge como una alternativa prometedora. Dado que su aplicación en problemas de secuenciación de la producción como el FSSP permanece prácticamente inexplorada, este trabajo propone la aplicación de ALPS para la minimización del *makespan*, evaluando su desempeño frente a los métodos tradicionales y metaheurísticos de referencia de la literatura.

## 3. Flow Shop Scheduling Problem (FSSP)

El problema clásico de *Flow Shop Scheduling* (FSSP) tiene su origen en el trabajo seminal de Johnson (1954), quien formuló el problema de secuenciación para dos máquinas y propuso un algoritmo óptimo para minimizar el tiempo total de finalización (*makespan*). Este trabajo sentó las bases para el desarrollo de los modelos generales de Flow Shop y de los numerosos métodos exactos y metaheurísticos propuestos posteriormente.

El *Flow Shop Scheduling Problem* (FSSP) es uno de los problemas clásicos de programación de la producción y consiste en determinar la secuencia óptima de un conjunto de trabajos que deben procesarse sobre un conjunto de máquinas dispuestas en serie, con el objetivo de optimizar uno o más indicadores de desempeño, siendo la minimización del *makespan* uno de los criterios más utilizados (Rossit et al., 2018).

En este problema se dispone de un conjunto de m máquinas,

$$M = \{M_1, M_2, \ldots, M_m\} \quad (1)$$

y un conjunto de n trabajos,

$$J = \{J_1, J_2, \ldots, J_n\} \quad (2)$$

Cada trabajo debe procesarse exactamente una vez en cada máquina y todos los trabajos siguen la misma ruta de procesamiento; es decir, deben recorrer las máquinas en un orden fijo y preestablecido.

De acuerdo con Rossit et al. (2018), el Flow Shop Scheduling Problem puede clasificarse en dos variantes principales: el *Permutation Flow Shop Scheduling Problem* (PFSP) y el *Non-Permutation Flow Shop Scheduling Problem* (NPFSP). En el PFSP, todos los trabajos conservan la misma secuencia de procesamiento en cada una de las máquinas; es decir, el orden establecido en la primera máquina permanece invariable durante todo el proceso productivo. En contraste, en el NPFSP la secuencia de los trabajos puede cambiar entre máquinas, lo que incrementa el espacio de búsqueda y la complejidad del problema.

El criterio de optimización más utilizado consiste en minimizar el *makespan* ($C_{max}$) definido como el tiempo de finalización del último trabajo procesado en la última máquina. Matemáticamente, este criterio se expresa como:

$$C_{max} = max\{C_1, C_2, \ldots, C_n\} \quad (3)$$

donde Ci representa el tiempo de finalización del trabajo i.

El tamaño del espacio de búsqueda aumenta factorialmente con el número de trabajos, por lo que el problema adquiere una elevada complejidad computacional conforme crecen las dimensiones de la instancia. En particular, Garey et al. (1976) demostraron que la minimización del *makespan* en el *Flow Shop Scheduling Problem* con tres o más máquinas pertenece a la clase de problemas *NP-hard*. Como consecuencia, la aplicación de métodos exactos resulta poco práctica para resolver instancias de gran tamaño, lo que ha impulsado el desarrollo de heurísticas, metaheurísticas y algoritmos basados en aprendizaje probabilístico capaces de obtener soluciones de alta calidad en tiempos computacionales razonables (Pinedo, 2012; Rossit et al., 2018).

Para la formulación del problema se consideran los siguientes supuestos, ampliamente aceptados para el Flow Shop clásico (González-Neira et al., 2017):

- Cada máquina puede procesar solo un trabajo a la vez.
- Cada máquina está disponible continuamente y sin interrupciones.
- Cada trabajo puede ser procesado por una sola máquina a la vez.

- Los tiempos de procesamiento de las tareas en las diferentes máquinas son determinados y fijos.
- Una vez que se inicia el procesamiento de un trabajo, no puede interrumpirse.
- Las máquinas de la celda pueden estar ociosas durante el horizonte de planificación.

Con base en estas características, el algoritmo propuesto para resolver el *Flow Shop Scheduling Problem* se desarrolla siguiendo el modelo metodológico mostrado en la Figura 1, el cual resume las etapas empleadas para su implementación y evaluación.




Figura 1. *Modelo Metodológico para sistema de Flow Shop.*

Fuente*:* elaboración propia.

## 4. Algoritmos Heurísticos y Metaheurísticos

En su libro, Yang (2020) menciona que existen dos tipos de algoritmos estocásticos: los heurísticos y los metaheurísticos. Por un lado, la palabra heurística proviene de la raíz “encontrar” o “descubrir por prueba y error”. Estos métodos encuentran buenas soluciones rápidamente y a un bajo costo, pero no garantizan llegar a la solución óptima global. Son útiles cuando no se necesita la respuesta perfecta, sino una "suficientemente buena" y fácil de obtener.

Por otro lado, las metaheurísticas, cuyo significado es “más allá de la heurística” y son algoritmos de mayor nivel que superan las limitaciones de las heurísticas simples. Su principal fortaleza radica en la combinación equilibrada entre aleatoriedad e intensificación, coordinando mecanismos de exploración global y explotación local. Gracias a esta ventaja, las metaheurísticas pueden escapar de los óptimos locales, lo que las hace adecuadas para buscar soluciones eficientes a problemas de gran escala y complejos.

Estos enfoques han intensificado un sinfín de investigaciones y análisis. En la revisión de literatura realizada por Zaied et al. (2021) se analizaron 120 artículos, los cuales buscaban respuestas de minimización del *makespan* en *el Flow Shop Problem* y concluyeron que en el 81% de esas investigaciones utilizaron algoritmos metaheurísticos para su solución, concluyendo que estos algoritmos arrojan mejores resultados que los heurísticos y los métodos tradicionales. De manera similar, Liang et al. (2022) afirman que un enfoque de algoritmo metaheurístico híbrido como la combinación entre el NGA (Algoritmo Genético de Nicho) y el NEH (algoritmo utilizado para resolver problemas de flujo) han proporcionado soluciones más eficientes a problemas NP-duros. Por su parte, Umam et al. (2021) integraron el *Genetic Algorithm* (GA) con *Tabu Search* (TS) para la minimización del *makespan* en un problema del *flow shop*, en dicha aplicación, los autores demostraron que la metaheurística no solo reduce el *makespan* de forma más efectiva, sino que ofrece una capacidad de búsqueda global mucho más eficiente ante problemas de alta complejidad.

En este estudio se analizan y comparan algunos algoritmos de la literatura, los cuales abarcan desde la heurística tradicional *Nawaz-Enscore-Ham* (NEH) y la estrategia de destrucción-reconstrucción *Iterated Greedy* (IG), hasta las metaheurísticas como *Simulated Annealing* (SA), *Ant Colony Optimization* (ACO), *Particle Swarm Optimization* (PSO) y *Genetic Algorithm* (GA), sirviendo como fundamento para la evaluación del algoritmo propuesto, ALPS. A continuación, se describe el principio de funcionamiento y modelo matemático de cada uno de los algoritmos a comparar con el que se propone.

### *4.1. Particle Swarm Optimization (PSO)*

El algoritmo *Particle Swarm Optimization* (PSO), propuesto por Kennedy y Eberhart en 1995, es una heurística de búsqueda poblacional inspirada en el comportamiento social de ciertas especies. En este método, cada partícula representa una solución potencial caracterizada por su posición, velocidad y memoria. Este método pertenece a la categoría de algoritmos de búsqueda poblacional, donde cada individuo, denominado partícula, representa una solución potencial dentro del espacio de búsqueda (Abualigah, 2025).

Zeidan et al. (2024) resaltan que este algoritmo está inspirado en el comportamiento colectivo de una bandada de aves en su búsqueda por alimento.

Clerc (2006) señala que en este método, se toman elementos simples llamados partículas, las cuales se colocan en un entorno de búsqueda asociado a un problema o función, en ese contexto, cada partícula evalúa la función que tiene como objetivo en ese instante, finalmente, después de un tiempo buscando, cada una de ellas ajusta su trayectoria óptima en función a su experiencia y la de uno o más miembros del enjambre, donde se integran algunas perturbaciones aleatorias. Con el tiempo, el enjambre termina funcionando en un punto casi óptimo de la función establecida.

El PSO tiene la capacidad de equilibrar la exploración y la explotación del espacio de búsqueda lo que hace eficaz al enfrentar problemas complejos y NP-duros (Mansouri et al., 2023).

Para determinar la velocidad y la posición de "la partícula" tenemos la ecuación (1) y (2) (Zeidan et al., 2021):

$$vel_{ij} = vel_{ij} + c_1 * r_1 * (p_{ij} - part_{id}) + c_2 * r_2 * (p_{gj} - part_{id}) \quad (4)$$

$$part_{id} = part_{ij} + vel_{ij} \quad (5)$$

donde $vel_{ij}$ es la velocidad de la partícula i en la j-ésima dimensión, $c_1$ y $c_2$ son pesos, conocidos con el nombre de factores de aprendizaje individual y social, respectivamente aplicados a la influencia de las mejores posiciones obtenidas hasta el momento por la partícula i y la mejor partícula del *swarm* g; $r_1$ y $r_2$, son valores aleatorios distribuidos uniformemente en el intervalo. Luego de que la velocidad se actualiza usando la ecuación (1) se calcula la nueva posición

de la partícula i en su dimensión j-ésima a través de la ecuación (2). Este proceso se repite para cada dimensión de la partícula i y para todas las partículas de la población. En un trabajo posterior Kenedy y Eberhart introdujeron el concepto de factor de inercia $w$, cuya finalidad es controlar la cantidad de la velocidad previa de la partícula a ser mantenida en el ciclo actual. En consecuencia, la ecuación (1) se reformula como:

$$vel_{i,j} = w * vel_{i,j} + c_1 * r_1 * (p_{i,j} - part_{id}) + c_2 * r_2 * (p_{g,j} - part_{id}) \quad (6)$$

### 4.2. Simulated Annealing (SA)

*Simulated Annealing (SA),* en español recocido simulado, propuesto por Kirkpatrick et al. (1983) está inspirado en el proceso de recocido de materiales en metalurgia. Debido a su capacidad de exploración, es usado ampliamente para dar respuestas al FSSP tanto en permutaciones simples como complejas (Hajji et al., 2024).

De acuerdo con SA, el parámetro de temperatura T determina la posibilidad de aceptar una nueva solución durante la ejecución del proceso. Cuando la temperatura se eleva, existe una mayor probabilidad de que el algoritmo admita soluciones con un valor de costo o error superior. En cambio, conforme la $T$ disminuye, también se reduce la opción de aceptar soluciones menos favorables. Si el nuevo modelo presenta un valor objetivo con menor error que el actual, se acepta de manera automática. De no ser así, todavía puede ser considerado, aunque solo con una probabilidad determinada (Khah et al., 2024).

El algoritmo SA se representa por (Khah et al., 2024) (4)

$$\Delta E = E_{k+1} - E_k$$
$$si\ \Delta E \leq 0\ entonces; aceptado \quad (7)$$

$$Por\ otro\ lado,\ p = e^{\frac{\Delta E}{T}}\ and; r = U[0,1]$$
$$si\ p \geq r, entonces\ se\ acepta \quad (8)$$

Donde:

$p =$ probabilidad de aceptar la nueva solución.
$\Delta E =$ cambio en la función objetivo o costo entre la solución nueva y la actual.
$E_k =$ Valor objetivo de la solución actual
$E_{k+1} =$ valor objetivo de la nueva solución.
$T =$Temperatura del algoritmo, este parámetro controla que tan flexible es la búsqueda.

### 4.3. Ant Colony Optimization (ACO)

El enfoque de optimización *Ant Colony Optimization* (ACO) traducido al español como optimización por colonia de hormigas es un algoritmo probabilístico usado para hallar el camino óptimo en el gráfico (Yan et al., 2023).

El modelo ACO, propuesto por primera vez por Dorigo et al. (1996) se considera como un algoritmo de inteligencia de enjambre (inteligencia grupal), que emula el comportamiento de las hormigas. Se basa en la forma en la cual las hormigas organizan sus colonias y cooperan para encontrar rutas eficientes. Las hormigas son criaturas autoorganizadas, lo que les permite una flexibilidad de adaptación a variaciones en su ambiente, además, las hormigas utilizan una sustancia conocida como feromona para marcar su camino, entre más feromona dejen en un rastro, más probabilidad habrá de que sea recorrida por las hormigas, en base a ello, se asume que esta ruta es una de las más óptimas y cortas (Rostamzadeh et al., 2024).

Etapas del algoritmo ACO (Rostamzadeh et al., 2024):

1. Componentes de valoración. Se colocan todas las hormigas en un punto o en diferentes puntos.

2. Localización. Se lleva a cabo la inicialización para encontrar las feromonas, (el valor de la feromona debe ser un pequeño número entero, de tal modo que se evite que las hormigas se queden quietas y se muevan en la ubicación de inicio).

3. Buscar una solución. Se utiliza una función de probabilidad para explicar el camino. La probabilidad de llegar al punto $j$ desde el punto $i$ (probabilidad de transferencia) se calcula usando la ecuación (9):

$$P_{ij}\{\frac{\left(\tau_{ij}^{\alpha} * \eta_{ij}^{\beta}\right)}{\sum_{h \epsilon tabuK} \tau_{ij}^{\alpha} * \eta_{ij}^{\beta}}, si\ j \notin tabuK \quad (9)$$

Donde:

$T_{ij}$= cantidad de la sustancia de feromona (sentido de hormiga) en los puntos $(i, j)$

$\eta_{ij}$ =es la proporción de visibilidad (el ojo de la hormiga y es la heurística del problema) de los puntos $(i, j)$, se determina como $\eta_{ij} = 1/d_{ij}$, donde $d_{ij}$ es la distancia entre dos nodos $i$ y $j$.

$\alpha$ y $\beta$= parámetros compatibles y representan tareas sobresalientes del camino frente a la proporción de visibilidad al elegir el siguiente punto para el movimiento.

$tabuK$ =es la lista de nodos visitados en el camino actual y forma la memoria de la hormiga.

Si $\alpha = 0$, el algoritmo se vuelve codicioso, de modo que la selección del siguiente nodo no considera la cantidad de feromonas, y como resultado se prioriza la selección del camino más cercano. Si $\beta = 0$, el algoritmo considera únicamente la cantidad de feromonas sin tener en cuenta la longitud del camino.

### 4.4. Genetic Algorithm (AG)

Los Algoritmos Genéticos son algoritmos de búsqueda basados en la mecánica de selección natural y de la genética natural. Combinan la supervivencia del más apto entre estructuras de secuencias con un intercambio de información estructurado, aunque aleatorizado, para constituir así un algoritmo de búsqueda que tenga algo de las genialidades de las búsquedas humanas, inspirado en la teoría de la evolución de Charles Darwin, inventados por John Holland en 1975.

El algoritmo genético mantiene una población de individuos (se denomina individuo a un vector que tiene “n” componentes, donde “n” es el número de variables que se

desea conocer), entiéndese P(t), para una generación t, donde cada individuo representa una potencial solución al problema considerado. (González Palomino, 2016, p. 46)

Los AG trabajan a partir de estas equivalencias: (Corzo y Paredes, 2000)

- Gen: un conjunto finito de bits.
- Genotipo o cromosoma: el conjunto de los genes, es decir, todas las cadenas de bits.
- Mutación puntual: alteración en un bit de un gen al azar.
- Cruce: intercambio de bits entre las cadenas de bits.
- Función objetivo: entorno al que debe adaptarse los individuos para sobrevivir.
- Factor de mutación: fracción de individuos que sufren mutación.
- Factor de intercambio. Fracción de individuos que sufren intercambio de bits.

### *4.5 NEH*

De acuerdo con Kaushik et al., 2025, el algoritmo NEH desarrollado por Nawaz, Enscore y Ham (1983) es una heurística ampliamente utilizada por su simplicidad y eficiencia para determinar la secuencia de procesamiento que minimiza el tiempo total de procesamiento (*makespan*, $C_{max}$) en entornos de *flow shop*.

El procedimiento algorítmico del NEH citado en Pinedo (2022) es el siguiente:

1. Cálculo total de procesamiento: Se calcula el tiempo total de procesamiento $T_k$ Para cada trabajo $k\{1,2,\dots,n\}$ se calcula la suma de sus tiempos de procesamiento a través de las $m$ máquinas, de acuerdo a la fórmula (10)

$$T_k = \sum_{j=1}^{m} \quad p_{kj} \tag{10}$$

2. Ordenamiento descendente: Se ordena de forma descendente los $n$ trabajos, de mayor a menor según el valor de $T_k$ La lista resultante establece el orden de prioridad de inserción.
3. Construcción de la secuencia inicial: Se toman los dos primeros trabajos de la lista ordenada y se evalúan las dos permutaciones posibles entre ellos. Se selecciona la secuencia que genere el menor $C_{max}parcial$.
4. Inserción iterativa: Para $K = 3,\dots,n$, se toma el trabajo en la posición $k$ de la lista priorizada y se prueba en todas las $k$ posiciones posibles dentro de la secuencia parcial ya establecida (sin alterar el orden relativo de los trabajos previamente insertados). Se fija el trabajo en la posición que entregue el menor $C_{max}parcial$ y se repite el procedimiento hasta haber posicionado los $n$ trabajos.

### *4.6 Iterated Greedy (IG)*

El algoritmo *Iterated Greedy* (IG), introducido por Ruiz y Stützle (2007), se inspira en el principio de destrucción y reconstrucción de estructuras, combinando la simplicidad de las heurísticas constructivas con la capacidad de exploración de las metaheurísticas. Su principal aporte al problema de Flow Shop radica en su alta eficiencia computacional y su capacidad para generar soluciones de excelente calidad en tiempos reducidos.

Los pasos a seguir para la aplicación de este algoritmo son los siguientes:

1. Solución inicial: Se construye una secuencia inicial $\pi$ aplicando el algoritmo de Nawaz, Enscore y Ham (NEH, 1983).
2. Fase de destrucción: Se extraen al azar $d$ trabajos de la secuencia actual $\pi$ (donde $d$ es el parámetro de destrucción). Los trabajos extraídos forman una lista de trabajos removidos $D$, dejando una secuencia parcial $\pi_R$ con los $n - d$ trabajos restantes.
3. Fase de reconstrucción: Los trabajos de la lista $D$ uno a uno y se reinsertan en $\pi_R$ utilizando la lógica de evaluación e inserción del algoritmo NEH, probando todas las posiciones posibles para cada trabajo y seleccionando la que minimice el *makespan* $(C_max)$.
4. Búsqueda local (opcional): La secuencia reconstruida $\pi'$se somete a un procedimiento de mejora (como búsqueda local por inserción) para intentar optimizar aún más su valor de $C_max$.
5. Criterio de aceptación: Si $C_max\ (\pi') \leq C_max\ (\pi)$ (ecuación 9), la nueva solución se acepta directamente. De lo contrario, se acepta con una probabilidad (ecuación 12) para escapar de óptimos locales.

$$C_max\ (\pi') \leq C_max\ (\pi) \tag{11}$$

$$P\ = exp\ exp\ \left(-\frac{C_{max}(\pi')-C_{max}\ (\pi)}{T}\right) \tag{12}$$

6. Criterio de parada: Los pasos 2 al 5 se repiten iterativamente hasta cumplir la condición límite (tiempo de cómputo o número máximo de iteraciones sin mejora).

## 5. Algoritmo de aprendizaje probabilístico propuesto

Una solución en el FSSP es representada como una permutación del conjunto de trabajos:

$$\mu = (\mu_1, \mu_2, \dots, \mu_n) \tag{13}$$

Donde $\mu_i, i = 1,2,\dots,n$, representa a cada uno de los $n$ trabajos que serán procesados en la línea de producción.

Por otro lado, para proporcionar un aprendizaje probabilístico al algoritmo que busca minimizar el *makespan*, se hace uso de una matriz de probabilidad $M \in R^{n\times n}$, donde cada elemento $(i, j)$ de la matriz se le asigna la probabilidad de que el trabajo *j* sea asignado en la posición *i*.

$$M(i,j) = P(trabajo\ j\ se\ asigne\ a\ la\ posición\ i) \tag{14}$$

Para cada renglón de la matriz se debe satisfacer que la suma de las probabilidades sea igual a 1.

$$\sum_{j=1}^{n} \quad M(i,j) = 1 \tag{15}$$

Inicialmente, a cada uno de los valores de la matriz de probabilidad se le asigna una probabilidad de (1/n).

Para llevar a cabo la generación de las posibles soluciones al problema, estas se generan de forma secuencial, mediante la asignación aleatoria de los valores de $\mu$. Con la restricción de que no exista repetición en estos valores.

Después de evaluar una solución, la matriz de probabilidad se actualiza, con base en la siguiente ecuación:

$$M_{ij} \leftarrow (1-\alpha)M_{ij} + \alpha\ \delta(j=\mu_i) \quad (16)$$

Donde $\alpha$ es la tasa de aprendizaje y $\delta$ es la delta Kronecker.

Para evitar una convergencia prematura, se aplica un mecanismo de perturbación a la matriz de probabilidades, utilizando la distribución uniforme entre 0 y 1.

$$M \leftarrow M + \in \cdot u(0,1) \quad (17)$$

Además, un subconjunto de soluciones se somete a un refinamiento local, mediante el uso de un operador de inserción $\mu' = Insert(\mu, i, j)$, con lo cual se incrementa la capacidad de diversificar las soluciones.

La implementación de este algoritmo requirió el uso de 4 subrutinas y un código principal (Algoritmo 1). A continuación, se presenta el pseudocódigo de cada una de ellas.

**Algoritmo 1**

```
Algoritmo ALPS para la minimización del makespan en el FSSP
Entrada: Archivo CSV con los tiempos de procesamiento P de los trabajos en las máquinas.
Salida: Mejor valor de makespan (Cmax) encontrado
1. Leer la matriz de tiempos de procesamiento P desde el archivo CSV.
2. Determinar:
   nTrabajos ← número de filas de P
   nMáquinas ← número de columnas de P
3. Inicializar parámetros:
   maxIter ← 100
   ⟨ ← 0.20
   ® ← 0.10
   probBusquedaLocal ← 0.30
   reseteo ← 0
4. Crear matriz de probabilidades:
   probMatrix ← matriz(nTrabajos × nTrabajos) con valor 1/nTrabajos
5. Generar secuencia aleatoria inicial:
solucionActual←PermutaciónAleatoria(nTrabajos)
6. Evaluar la solución inicial:
   mejorSolucion ← solucionActual
   mejorValor ← CalcularMakespan(solucionActual, P)
7. Para iter ← 1 hasta maxIter hacer
   7.1 Generar una nueva solución:
       solucion ← GenerarSolucion(probMatrix)
   7.2 Evaluar la solución:
       valor ← CalcularMakespan(solucion, P)
   7.3 Si númeroAleatorio < probBusquedaLocal entonces
       solucion ← BusquedaLocalInsercion(solucion, P)
       valor ← CalcularMakespan(solucion, P)
       FinSi
   7.4 Si valor < mejorValor entonces
           mejorValor ← valor
           mejorSolucion ← solucion
       FinSi
   7.5 Para i ← 1 hasta nTrabajos hacer
           trabajo ← solucion(i)
           probMatrix(i,:) ← (1 − α) · probMatrix(i,:)
           probMatrix(i,trabajo) ← probMatrix(i,trabajo) + α
       FinPara
  7.6 Normalizar cada fila de probMatrix:
      probMatrix ← probMatrix / sumaFilas(probMatrix)
  7.7 Si númeroAleatorio < β entonces
        probMatrix ← probMatrix +
                0.05 × matrizAleatoria
        Normalizar(probMatrix)
      FinSi
  7.8 Registrar mejorValor en el historial
  7.9 Si iter > 1 y mejorValor = historial(iter−1) entonces
          reseteo ← reseteo + 1
      FinSi
  7.10 Si reseteo ≥ 5 entonces
          reseteo ← 0
          probMatrix←Resetea(probMatrix)
       FinSi
   FinPara
8. mejorMakespan ← mejorValor
9. Retornar mejorMakespan.
```

Las subrutinas adicionales se muestran a continuación.

**Algoritmo 2**

```
Procedimiento GenerarSolucion
Entrada: probMatrix.
Salida: solución
1. n ← número de filas de probMatrix
2. disponibles ← {1, 2, ..., n}
3. Para i ← 1 hasta n hacer
     3.1 Obtener probabilidades asociadas
         a los trabajos disponibles.
     3.2 Normalizar probabilidades.
     3.3 Generar un número aleatorio r.
    3.4 Seleccionar el primer trabajo cuya probabilidad acumulada sea ≥ r.
```

```
    3.5 Asignar el trabajo seleccionado a
la posición i de la solución.
    3.6 Eliminar dicho trabajo del
conjunto de disponibles.
   FinPara
4. Retornar solucion
```

**Algoritmo 3**

```
Procedimiento de búsqueda local por
inserción
Entrada: solucion, P
Salida: mejorSolucion
  1. mejorSolucion ← solucion
  2. mejorValor ←
CalcularMakespan(solucion, P)
  3. Para i ← 1 hasta longitud(solucion)
hacer
     Para j ← 1 hasta longitud(solucion)
hacer
     Si i ≠ j entonces
        3.1 Crear solución temporal.
        3.2 Extraer el trabajo ubicado
            en la posición i.
        3.3 Insertarlo en la posición j.
        3.4 Evaluar makespan.
        3.5 Si valor < mejorValor
            entonces
              mejorValor ← valor
              mejorSolucion ← temporal
            FinSi
      FinSi
    FinPara
   FinPara
4. Retornar mejorSolucion.
```

**Algoritmo 4**

```
Procedimiento para cálcular del makespan
Entrada: secuencia, P
Salida: Cmax
1. Crear matriz de tiempos de
finalización C.
2. Para cada trabajo i hacer
    Para cada máquina j hacer
      Si i = 1 y j = 1 entonces
        C(i,j) ← P(secuencia(i),j)
      SinoSi i = 1 entonces
        C(i,j) ← C(i,j−1) +
                 P(secuencia(i),j)
      SinoSi j = 1 entonces
        C(i,j) ← C(i−1,j)+
                   P(secuencia(i),j)
      Sino
        C(i,j) ← max(C(i−1,j), C(i,j−1))
                   + P(secuencia(i),j)
      FinSi
    FinPara
   FinPara
  3. Cmax ← C(nTrabajos,nMáquinas)
4. Retornar Cmax.
```

**Algoritmo 5**

```
Procedimiento de reseteo de la matriz de
probabilidades
Entrada: probMatrix
Salida: matrizReseteada
1. matrizReseteada ← probMatrix
2. Seleccionar aleatoriamente dos
columnas:
   c1 y c2
3. Intercambiar las columnas c1 y c2.
4. Retornar matrizReseteada.
```

## 6. Resultados

Para evaluar el desempeño del algoritmo propuesto (ALPS), se comparó cuantitativamente con seis algoritmos metaheurísticos y heurísticos de la literatura: *Genetic Algorithm* (GA), *Ant Colony Optimization* (ACO), *Particle Swarm Optimization* (PSO), *Simulated Annealing* (SA), la heurística NEH e *Iterated Greedy* (IG).

Todos los algoritmos de comparación (SA, PSO, GA, ACO, NEH e IG), así como el algoritmo propuesto (ALPS), fueron programados e implementados en el entorno de desarrollo MATLAB R2020b, con el propósito de garantizar un marco de codificación homogéneo y una comparación equitativa entre todos los métodos. Las pruebas experimentales se realizaron utilizando la misma plataforma de hardware: una MacBook Pro equipada con un procesador Intel Core i9 de 8 núcleos a 2.3 GHz y 32 GB de memoria RAM.

Para asegurar una evaluación justa entre las diferentes metaheurísticas, se estableció un criterio de parada uniforme de 20 iteraciones por ejecución. Dado el carácter estocástico de estos algoritmos, cada método se ejecutó de forma independiente 20 veces para cada una de las 20 instancias de prueba, registrando los resultados obtenidos en cada corrida. Las instancias empleadas corresponden al *benchmark* para el problema de programación de taller de flujo (FSSP) publicado por Akilelkamel (2024).

Posteriormente, se elaboraron 20 tablas, cada una correspondiente a una instancia, desde 3 máquinas × 10 trabajos hasta 10 máquinas × 100 trabajos. En cada tabla se registraron los resultados obtenidos en las 20 ejecuciones de cada algoritmo (ALPS, SA, ACO, PSO, NEH, GA e IG). A partir de dichas ejecuciones, se extrajeron los valores mínimos absolutos del *makespan* alcanzados por cada algoritmo como indicador del mejor desempeño obtenido, como se muestra en la Tabla 1. Esta tabla resume y compara los valores mínimos (*best makespan*) obtenidos mediante las distintas técnicas de solución.

**Tabla 1.** Comparativa de los valores mínimos de *makespan* obtenidos por los algoritmos SA, PSO, AG, ACO, NEH, IG y ALPS en los conjuntos de datos evaluados

| N | SA | PSO | AG | ACO | ALPS | NEH | IG |
|---|---|---|---|---|---|---|---|
| **3_10** | 126 | 126 | 126 | 126 | 126 | 126 | 126 |
| **3_20** | 218 | 218 | 218 | 218 | 218 | 218 | 218 |
| **3_30** | 321 | 321 | 321 | 327 | 321 | 321 | 321 |
| **3_50** | 535 | 534 | 534 | 534 | 534 | 534 | 534 |
| **3_100** | 1013 | 1013 | 1013 | 1013 | 1013 | 1012 | 1012 |
| **5_10** | 129 | 129 | 129 | 132 | 129 | 132 | 129 |
| **5_20** | 264 | 264 | 264 | 273 | 273 | 269 | 264 |
| **5_30** | 327 | 334 | 327 | 349 | 346 | 327 | 327 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **5_50** | 563 | 569 | 563 | 589 | 582 | 565 | 561 |
| **5_100** | 1085 | 1095 | 1086 | 1115 | 1109 | 1085 | 1085 |
| **8_10** | 187 | 187 | 187 | 190 | 187 | 197 | 187 |
| **8_20** | 289 | 304 | 288 | 317 | 313 | 303 | 284 |
| **8_30** | 389 | 397 | 389 | 409 | 413 | 390 | 389 |
| **8_50** | 579 | 604 | 581 | 630 | 621 | 575 | 571 |
| **8_100** | 1111 | 1162 | 1123 | 1187 | 1182 | 1099 | 1092 |
| **10_10** | 215 | 215 | 215 | 217 | 215 | 216 | 215 |
| **10_20** | 391 | 302 | 290 | 317 | 314 | 292 | 288 |
| **10_30** | 401 | 424 | 407 | 441 | 434 | 403 | 391 |
| **10_50** | 640 | 673 | 645 | 693 | 686 | 646 | 631 |
| **10_100** | 1156 | 1202 | 1161 | 1235 | 1239 | 1154 | 1129 |

Fuente: elaboración propia

Para analizar qué tan eficiente es ALPS frente los resultados de los 6 algoritmos mencionados, se encontró el valor mínimo global de cada instancia, y se aplicó la fórmula de eficiencia, dividiendo cada valor óptimo de cada instancia entre los valores mínimos óptimos obtenidos por ALPS, ver tabla 2.

**Tabla 2.** Comparación de la Eficiencia del Algoritmo ALPS respecto a los Valores Mínimos Óptimos obtenidos.

| ALPS | VALORES MINIMOS ÓPTIMOS | EFICIENCIA DE ALPS |
|---|---|---|
| **126** | 126 | 100% |
| **218** | 218 | 100% |
| **321** | 321 | 100% |
| **534** | 534 | 100% |
| **1013** | 1012 | 100% |
| **129** | 129 | 100% |
| **273** | 264 | 97% |
| **346** | 327 | 95% |
| **582** | 561 | 96% |
| **1109** | 1085 | 98% |
| **187** | 187 | 100% |
| **313** | 284 | 91% |
| **413** | 389 | 94% |
| **621** | 571 | 92% |
| **1182** | 1092 | 92% |
| **215** | 215 | 100% |
| **314** | 288 | 92% |
| **434** | 391 | 90% |
| **686** | 631 | 92% |
| **1239** | 1129 | 91% |
| **Eficiencia Global** | | 96% |

Fuente: elaboración propia.

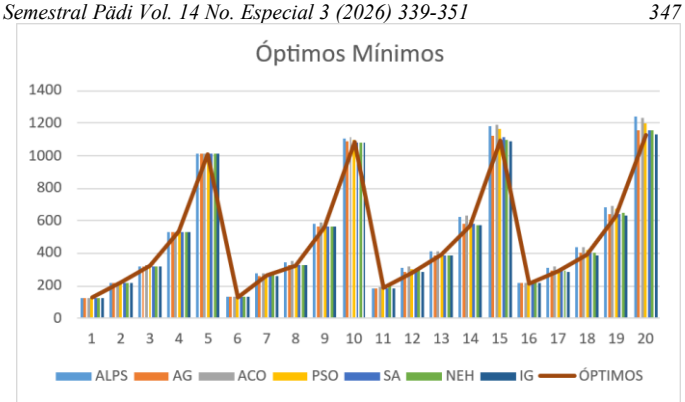



Figura 2. *Comparación de los valores mínimos obtenidos por los algoritmos SA, PSO, AG, ACO, ALPS, NEH e IG en 20 conjuntos de datos.*

Fuente: elaboración propia.

En la Figura 2 se presenta la comparación gráfica de los valores mínimos del *makespan* alcanzados por cada algoritmo en las 20 instancias evaluadas, contrastados con el mejor valor de referencia (línea continua). Se aprecia que, en las instancias iniciales de menor escala (escenarios 1 a 6), la mayoría de los algoritmos metaheurísticos convergen hacia el mismo valor mínimo. Sin embargo, a medida que se incrementa el número de máquinas y trabajos, la brecha entre los algoritmos se amplía. En este comportamiento, IG y SA se mantienen más cercanos al mejor valor de referencia, mientras que el algoritmo propuesto (ALPS) demuestra una trayectoria estable y cercana a dicha referencia, superando en consistencia a métodos poblacionales como la Optimización por Colonia de Hormigas (ACO).

En relación con los resultados obtenidos, ALPS mostró un desempeño competitivo al alcanzar una eficiencia global del 96 % respecto a los mejores valores registrados. En las instancias de menor complejidad (hasta 3 máquinas × 50 trabajos), el algoritmo alcanzó una eficiencia del 100 %, obteniendo el mismo valor mínimo de *makespan* que los mejores algoritmos evaluados. Conforme aumenta el tamaño de las instancias, particularmente el número de trabajos y la combinación de máquinas y tareas, la eficiencia de ALPS disminuye gradualmente, manteniéndose en un intervalo comprendido entre el 90 % y el 98 % respecto al mejor *makespan* obtenido. Estos resultados indican que, aunque el algoritmo conserva un desempeño competitivo en problemas de mayor dimensión, la diferencia respecto a los mejores algoritmos tiende a incrementarse conforme crece la complejidad del problema.

Por otra parte, la Tabla 1 muestra que IG obtuvo el mayor número de mejores soluciones entre los algoritmos comparados, consolidándose como el método con mejor desempeño global en términos de calidad de solución. Asimismo, SA y AG alcanzaron resultados iguales o superiores a los obtenidos por ALPS en la mayoría de las instancias de tamaño medio y grande. En contraste, ALPS presentó un comportamiento muy similar al de ACO, superándolo en once de las veinte instancias evaluadas y alcanzando el mismo valor mínimo de *makespan* en problemas como 3 × 30, 5 × 10, 8 × 10 y 10 × 10, lo que evidencia un desempeño competitivo frente a este algoritmo.

Aunado a lo anterior, se presentan los tiempos mínimos de procesamiento obtenidos por cada algoritmo durante las ejecuciones realizadas, ver tabla 3.

**Tabla 3.** Tiempos de Proceso en Segundos de los 20 conjuntos de datos ejecutados con Algoritmos Metaheurísticos.

| N | ALPS | AG | ACO | PSO | SA | NEH | IG |
|---|---|---|---|---|---|---|---|
| **3_10** | 0.01995 | 0.06813 | 0.08067 | 0.01478 | 0.01938 | 0.00014 | 0.00803 |
| **3_20** | 0.05306 | 0.07127 | 0.15586 | 0.01539 | 0.01789 | 0.00055 | 0.02628 |
| **3_30** | 0.11704 | 0.07528 | 0.23692 | 0.01735 | 0.02188 | 0.00106 | 0.05954 |
| **3_50** | 0.32449 | 0.08397 | 0.43045 | 0.02273 | 0.02243 | 0.00331 | 0.19718 |
| **3_100** | 1.56544 | 0.10097 | 1.01068 | 0.02638 | 0.02659 | 0.01595 | 1.24049 |
| **5_10** | 0.01935 | 0.06851 | 0.08122 | 0.01450 | 0.01699 | 0.00014 | 0.00832 |
| **5_20** | 0.05007 | 0.07232 | 0.15679 | 0.01609 | 0.01879 | 0.00059 | 0.03011 |
| **5_30** | 0.11682 | 0.07858 | 0.23950 | 0.01786 | 0.01985 | 0.00122 | 0.08097 |
| **5_50** | 0.37690 | 0.08664 | 0.43269 | 0.02099 | 0.02349 | 0.00382 | 0.29801 |
| **5_100** | 1.61819 | 0.10657 | 1.01785 | 0.02935 | 0.03114 | 0.01900 | 1.66247 |
| **8_10** | 0.01976 | 0.06918 | 0.08200 | 0.01513 | 0.01913 | 0.00013 | 0.00921 |
| **8_20** | 0.05101 | 0.07260 | 0.15820 | 0.01716 | 0.02181 | 0.00053 | 0.04094 |
| **8_30** | 0.14036 | 0.08024 | 0.24152 | 0.01932 | 0.02495 | 0.00130 | 0.11006 |
| **8_50** | 0.37027 | 0.09144 | 0.43449 | 0.02413 | 0.03395 | 0.00441 | 0.44831 |
| **8_100** | 2.09768 | 0.11552 | 1.02292 | 0.03500 | 0.04104 | 0.02443 | 2.93661 |
| **10_10** | 0.02066 | 0.06979 | 0.08226 | 0.01600 | 0.01895 | 0.00014 | 0.01014 |
| **10_20** | 0.05573 | 0.07524 | 0.15720 | 0.01807 | 0.02285 | 0.00067 | 0.04293 |
| **10_30** | 0.11784 | 0.08062 | 0.24489 | 0.02107 | 0.02618 | 0.00145 | 0.12747 |
| **10_50** | 0.36446 | 0.09305 | 0.43656 | 0.02933 | 0.03166 | 0.00484 | 0.47371 |
| **10_100** | 1.69665 | 0.12061 | 1.02437 | 0.04290 | 0.04716 | 0.02680 | 3.72318 |

Fuente*:* elaboración propia.

Para analizar la eficiencia de los tiempos (ETO) se dividió el tiempo más rápido obtenido de cada instancia (TO) entre el tiempo obtenido por ALPS, ver tabla 4.

**Tabla 4.** Eficiencia del Algoritmo ALPS respecto a los Tiempos Mínimos de Procesamiento obtenidos.

| ALPS | TO | ETO |
|---|---|---|
| **0.019947** | 0.000142 | 0.007119 |
| **0.053060** | 0.000548 | 0.010336 |
| **0.117044** | 0.001063 | 0.009084 |
| **0.324493** | 0.003315 | 0.010215 |
| **1.565442** | 0.015947 | 0.010187 |
| **0.019346** | 0.000135 | 0.006989 |
| **0.050066** | 0.000591 | 0.011795 |
| **0.116817** | 0.001216 | 0.010412 |
| **0.376901** | 0.003820 | 0.010135 |
| **1.618188** | 0.018997 | 0.011740 |
| **0.019762** | 0.000127 | 0.006441 |
| **0.051014** | 0.000532 | 0.010437 |
| **0.140360** | 0.001297 | 0.009238 |
| **0.370273** | 0.004408 | 0.011904 |
| **2.097682** | 0.024426 | 0.011644 |
| **0.020665** | 0.000141 | 0.006834 |
| **0.055726** | 0.000667 | 0.011970 |
| **0.117839** | 0.001450 | 0.012308 |
| **0.364456** | 0.004844 | 0.013291 |
| **1.696649** | 0.026800 | 0.015796 |

Fuente*:* elaboración propia.

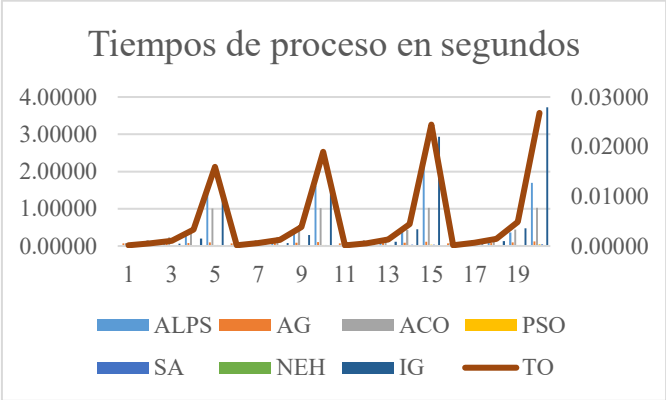



**Figura 3**. Comparación de los tiempos de procesamiento en segundos, obtenidos por los algoritmos SA, PSO, AG, ACO, ALPS, NEH e IG en los conjuntos de datos evaluados.

Fuente*:* elaboración propia.

Al analizar la eficiencia de tiempo de ALPS (ETO), como se muestra en la Tabla 4, se observa que su rendimiento relativo respecto al algoritmo más rápido (TO) se mantiene en un rango entre 0.64 % y 1.58 %. Esta tendencia se visualiza en la Figura 3, donde se ilustra la variación de los tiempos de ejecución a lo largo de los 20 escenarios evaluados. Se aprecian incrementos marcados en el tiempo de cómputo en las instancias de mayor dimensión, particularmente en aquellas con 100 trabajos (escenarios 5, 10, 15 y 20). La línea continua representa el tiempo mínimo de ejecución (TO) alcanzado por la técnica más rápida (NEH). Aunque métodos como NEH o PSO registran los tiempos más bajos, el algoritmo ALPS evidencia un comportamiento computacional estable e intermedio, ofreciendo un balance adecuado entre el tiempo de ejecución y la calidad del *makespan* obtenido.

Con el propósito de determinar si existían diferencias estadísticamente significativas entre los algoritmos evaluados, se aplicó el test no paramétrico de Friedman sobre los valores promedio de *makespan* obtenidos en las 20 instancias. Los resultados mostraron diferencias estadísticamente significativas entre los siete algoritmos, ($x_F^2(6) = 99.0194$), ($p<0.001$), por lo que se rechazó la hipótesis nula de igualdad

de desempeño. Este resultado fue confirmado mediante la corrección de Iman-Davenport, ($F_F = 6.114$), ($p<0.001$), ver tabla 5.

**Tabla 5.** Resultados del análisis post hoc de Friedman para la métrica de makespan

| Alg 1 | Alg2 | Δ rangos | \|z\| | p sin ajustar | p ajustado Holm |
|---|---|---|---|---|---|
| ALPS | IG | 4.900000 | 7.172866 | 0.000000 | 0.000000 |
| ACO | IG | 4.800000 | 7.026481 | 0.000000 | 0.000000 |
| PSO | IG | 4.000000 | 5.855400 | 0.000000 | 0.000000 |
| ALPS | NEH | 3.750000 | 5.489438 | 0.000000 | 0.000001 |
| ACO | NEH | 3.650000 | 5.343053 | 0.000000 | 0.000002 |
| ALPS | SA | 3.425000 | 5.013687 | 0.000001 | 0.000009 |
| ACO | SA | 3.325000 | 4.867302 | 0.000001 | 0.000017 |
| PSO | NEH | 2.850000 | 4.171973 | 0.000030 | 0.000423 |
| ALPS | AG | 2.675000 | 3.915799 | 0.000090 | 0.001171 |
| AG | ACO | -2.5750 | 3.769414 | 0.000164 | 0.001964 |
| PSO | SA | 2.525000 | 3.696222 | 0.000219 | 0.002407 |
| AG | IG | 2.225000 | 3.257066 | 0.001126 | 0.011257 |
| AG | PSO | -1.7750 | 2.598334 | 0.009368 | 0.084310 |
| SA | IG | 1.475000 | 2.159179 | 0.030836 | 0.246690 |
| NEH | IG | 1.150000 | 1.683428 | 0.092292 | 0.646046 |
| AG | NEH | 1.075000 | 1.573639 | 0.115571 | 0.693426 |
| ALPS | PSO | 0.900000 | 1.317465 | 0.187683 | 0.938414 |
| AG | SA | 0.750000 | 1.097888 | 0.272254 | 0.966266 |
| ACO | PSO | 0.800000 | 1.171080 | 0.241567 | 0.966266 |
| ALPS | ACO | 0.100000 | 0.146385 | 0.883617 | 1.000000 |
| SA | NEH | 0.325000 | 0.475751 | 0.634252 | 1.000000 |

Fuente: elaboración propia.

De acuerdo con los rangos promedio, IG presentó el mejor desempeño general, con un rango de 1.350, seguido de NEH con 2.500, SA con 2.825 y AG con 3.575. Por su parte, PSO, ACO y ALPS obtuvieron rangos promedio de 5.350, 6.150 y 6.250, respectivamente. Debido a que un rango menor representa un mejor desempeño en la minimización del *makespan*, estos resultados posicionan a IG como el algoritmo con mayor rendimiento global en las instancias analizadas, ver figura 3.

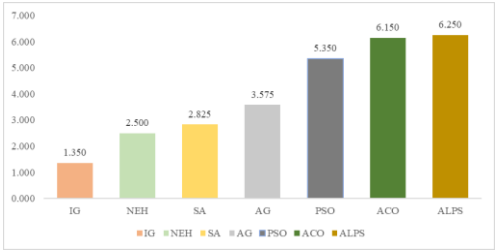



**Figura 4**. Diagrama de rangos promedio y diferencia crítica de Nemenyi

Fuente: elaboración propia.

Posteriormente, se aplicó un análisis post-hoc con corrección de Holm, tomando al algoritmo propuesto ALPS como método de referencia. Los resultados mostraron diferencias estadísticamente significativas entre ALPS e IG, NEH, SA y AG, cuyos rangos promedio fueron menores. En contraste, no se encontraron diferencias estadísticamente significativas entre ALPS y PSO ni entre ALPS y ACO. Por consiguiente, el desempeño promedio de ALPS puede considerarse estadísticamente similar al de PSO y ACO en el conjunto de instancias evaluadas, aunque inferior al observado para IG, NEH, SA y AG, ver tabla 6.

**Tabla 6.** Resultados del análisis ALPS frente a los demás algoritmos

| Algoritmo 1 | Algoritmo 2 | Diferencia de rangos |
|---|---|---|
| ALPS | IG | 4.900000 |
| ALPS | NEH | 3.750000 |
| ALPS | SA | 3.425000 |
| ALPS | AG | 2.675000 |
| ALPS | PSO | 0.900000 |
| ALPS | ACO | 0.100000 |

Fuente: elaboración propia.

## 7. Conclusiones

En este trabajo se presentó un algoritmo metaheurístico basado en aprendizaje probabilístico adaptativo para la resolución del *Flow Shop Scheduling Problem* (FSSP), el cual integra una matriz de probabilidades, un mecanismo de actualización incremental, una estrategia de perturbación adaptativa y un operador de inserción para el refinamiento local, con el propósito de construir secuencias de trabajos orientadas a la minimización del *makespan*.

La evaluación experimental se realizó mediante la comparación del algoritmo propuesto (ALPS) con seis métodos representativos de la literatura: *Simulated Annealing* (SA), *Particle Swarm Optimization* (PSO), Algoritmo Genético (AG), *Ant Colony Optimization* (ACO), *Nawaz-Enscore-Ham* (NEH) e *Iterated Greedy* (IG), utilizando veinte instancias del *benchmark* seleccionado y veinte ejecuciones independientes por instancia. Los resultados mostraron que ALPS alcanzó una eficiencia global del 96 % respecto a los mejores valores de *makespan* obtenidos y alcanzó el mejor valor registrado en diversas instancias de prueba. Asimismo, el análisis estadístico mediante el test de Friedman con comparación *post hoc* permitió establecer objetivamente el desempeño del algoritmo con respecto a los métodos comparados, respaldando los resultados obtenidos durante la experimentación.

En términos de calidad de solución, ALPS presentó un desempeño competitivo, alcanzando el mejor valor registrado en todas las instancias de menor complejidad y manteniendo resultados cercanos a los mejores valores obtenidos en instancias de mayor dimensión. Respecto al tiempo de ejecución, el algoritmo mostró un desempeño computacional intermedio, obteniendo tiempos inferiores a ACO e IG en la mayoría de las instancias evaluadas y manteniendo un desempeño competitivo en términos de calidad de solución y tiempo de ejecución. En conjunto, los resultados obtenidos muestran que el modelo de aprendizaje probabilístico adaptativo propuesto constituye una alternativa competitiva para la minimización del *makespan* en el *Flow Shop Scheduling Problem*.

Como trabajo futuro, se propone extender el algoritmo a variantes del problema, como el *Non-Permutation Flow Shop Scheduling Problem* (NPFSP), así como evaluar la incorporación de estrategias de búsqueda local con múltiples

vecindarios que permitan analizar su impacto en la calidad de las soluciones y el desempeño computacional del algoritmo.

## 8. Declaración sobre el uso de Inteligencia Artificial generativa y tecnologías asistidas por IA

Durante la preparación del presente manuscrito, los autores utilizaron ChatGPT (OpenAI, modelo GPT-5.6 Sol) para corrección gramatical y mejora de redacción. Todo el contenido generado o sugerido por la herramienta fue revisado y editado por los autores, quienes asumen plena responsabilidad por la exactitud, originalidad, integridad y rigor científico del contenido final del manuscrito.

## 9. Agradecimientos

Expresamos nuestro sincero agradecimiento a la Universidad Autónoma del Estado de Hidalgo y al Instituto de Ciencias Básicas e Ingeniería (ICBI) por brindarnos el espacio, los recursos e infraestructura necesarios para nuestra formación académica en la licenciatura en Ingeniería Industrial. Agradecemos de manera especial a nuestro mentor, el Dr. Joselito Medina Marín, por su valiosa cátedra, constante guía académica y rigurosa orientación a lo largo de nuestra formación. Asimismo, agradecemos a nuestros compañeros por el apoyo y las experiencias compartidas, y a nuestras familias por su comprensión, respaldo y acompañamiento constante a lo largo de nuestra formación académica.

## 10. Referencias

Abualigah, L. (2025). Particle swarm optimization: Advances, applications, and experimental insights. Computers, Materials & Continua, 82(2), 1539–1592. https://doi.org/10.32604/cmc.2025.060765

Akilelkamel. (2024). FSSP dataset [Data set]. GitHub. https://github.com/akilelkamel/fssp-dataset

Angarita Moreno, A. (2024). Minimización del tiempo máximo de terminación (*makespan*) en líneas de producción tipo flow shop donde los tiempos de procesamiento varían en función de la hora del día. [Tesis de maestría, Universidad de los Andes]. https://hdl.handle.net/1992/73549

Babor, M., Senge, J., Rosell, C. M., Rodrigo, D., & Hitzmann, B. (2021). Optimization of no-wait flowshop scheduling problem in bakery production with modified PSO, NEH and SA. Processes, 9(11), 2044. https://doi.org/10.3390/pr9112044

Campbell, H. G., Dudek, R. A., & Smith, M. L. (1970). A heuristic algorithm for the n job, m machine sequencing problem. Management Science, 16(10), B630–B637. https://doi.org/10.1287/mnsc.16.10.B630

Clerc, M. (2006). *Particle swarm optimization*. ISTE Ltd. https://doi.org/10.1002/9780470612163

Corzo-Suárez, R., & Paredes Olay, F. J. (2000). Algoritmos genéticos: la evolución como modelo matemático. SUMA: Revista sobre Enseñanza y Aprendizaje de las Matemáticas, 35, 15–20. https://funes.uniandes.edu.co/wp-content/uploads/tainacan-items/32454/1228418/015-020-1.pdf

Dannenbring, D. G. (1977). An evaluation of flow shop sequencing heuristics. Management Science, 23(11), 1174–1182. https://doi.org/10.1287/mnsc.23.11.1174

Dorigo, M., Maniezzo, V., & Colorni, A. (1996). Ant system: Optimization by a colony of cooperating agents. IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), 26(1), 29–41. https://doi.org/10.1109/3477.484436

Garey, M. R., Johnson, D. S., & Sethi, R. (1976). The complexity of flow-shop and job-shop scheduling. Mathematics of Operations Research, 1(2), 117–129. https://doi.org/10.1287/moor.1.2.117

González-Neira, E., Montoya-Torres, J., & Barrera, D. (2017). Flow-shop scheduling problem under uncertainties: Review and trends. *International Journal of Industrial Engineering Computations*, *8*(4), 399-426. 10.5267/j.ijiec.2017.2.001

González Palomino, G. (2016). Algoritmos genéticos. En Motores lineales de imanes permanentes: principios de funcionamiento y optimización (pp. 46). Programa Editorial Universidad Autónoma de Occidente.

Gupta, J. N. D. (1971). A functional heuristic algorithm for the flowshop scheduling problem. Operational Research Quarterly, 22(1), 39–47. https://doi.org/10.1057/jors.1971.6

Gupta, J. N. D., & Stafford, E. F., Jr. (2006). Flowshop scheduling research after five decades. European Journal of Operational Research, 169(3), 699–711. https://doi.org/10.1016/j.ejor.2005.02.001

Hajji, M. K., Hamlaoui, O., & Hadda, H. (2024). A simulated annealing metaheuristic approach to hybrid flow shop scheduling problem. Advances in Industrial and Manufacturing Engineering, 9, 100144. https://doi.org/10.1016/j.aime.2024.100144

Holland, J. H. (1975). *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence*. University of Michigan Press.

Hornby, G. S. (2006). The age-layered population structure for reducing the problem of premature convergence. In Proceedings of the 8th annual conference on Genetic and evolutionary computation (GECCO '06) (pp. 815–822). ACM. https://doi.org/10.1145/1143997.1144142

Johnson, S. M. (1954). Optimal two- and three-stage production schedules with setup times included. Naval Research Logistics Quarterly, 1(1), 61–68. https://doi.org/10.1002/nav.3800010110

Kaushik, P., Gupta, D., & Goel, S. (2025). Comparative study of B&B with heuristics NEH and CDS for bi stage flow shop scheduling problem under fuzzy environment. Journal of Mechanics of Continua and Mathematical Sciences, 20(2). https://doi.org/10.26782/jmcms.2025.02.00007

Kennedy, J., & Eberhart, R. (1995). Particle swarm optimization. In Proceedings of ICNN'95 - International Conference on Neural Networks (Vol. 4, pp. 1942–1948). IEEE. https://doi.org/10.1109/ICNN.1995.488968

Khah, N. K. F., Salehi, B., Kianoush, P., & Varkouhi, S. (2024). Estimating elastic properties of sediments by pseudo-wells generation utilizing simulated annealing optimization method. Results in Earth Sciences, 2, 100024. https://doi.org/10.1016/j.rines.2024.100024

Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing. Science, 220(4598), 671–680. https://doi.org/10.1126/science.220.4598.671

Lemtenneche, S., Bensayah, A., & Cheriet, A. (2023). An estimation of distribution algorithm for permutation flow-shop scheduling problem. Systems, 11(8), 389. https://doi.org/10.3390/systems11080389

Liang, Z., Zhong, P., Liu, M., Zhang, C., & Zhang, Z. (2022). A computational efficient optimization of flow shop scheduling problems. Scientific Reports, 12(1), 845. https://doi.org/10.1038/s41598-022-04887-8

Mansouri, M., Bahmani, Y., & Smadi, H. (2023). Optimization of the flow-shop scheduling problem under time constraints with PSO algorithm. Engineering Proceedings, 56(1), 220. https://doi.org/10.3390/asec2023-15410

Najarro, R., Lopez, R. J., Racines, R. E., & Puris, A. Y. (2018). Un algoritmo genético híbrido para optimizar un caso especial del problema de secuenciación de tareas. Revista Tecnológica ESPOL (RTE), 31(1), 39-52. https://rte.espol.edu.ec/index.php/tecnologica/es/article/view/646/386

Nawaz, M., Enscore, E. E., & Ham, I. (1983). A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. Omega, 11(1), 91–95. https://doi.org/10.1016/0305-0483(83)90088-9

Palmer, D. S. (1965). Sequencing jobs through a multi-stage process in the minimum total time—A quick method of obtaining a near optimum. Operational Research Quarterly, 16(1), 101–107. https://doi.org/10.1057/jors.1965.8

Pinedo, M. L. (2022). *Scheduling: Theory, Algorithms, and Systems (6th ed.).* Springer. https://doi.org/10.1007/978-3-031-05886-8

Rossit, D. A., Tohmé, F., & Frutos, M. (2018). *The non-permutation flow-shop scheduling problem: A literature review*. *Omega, 77*, 143–153. https://doi.org/10.1016/j.omega.2017.05.010

Rostamzadeh, R., Gholipour, A., Komari Alaei, M. R., Zavadskas, E. K., & Saparauskas, J. (2024). An attempt to resolve no-wait flow shop scheduling problems using hybrid ant colony and whale optimization algorithms. Economics and Management, 27(2), 108–124. https://doi.org/10.15240/tul/001/2024-2-007

Ruiz, R., & Maroto, C. (2005). A comprehensive review and evaluation of permutation flowshop heuristics. European Journal of Operational Research, 165(2), 479–494. https://doi.org/10.1016/j.ejor.2004.04.017

Ruiz, R., & Stützle, T. (2007). A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. European Journal of Operational Research, 177(3), 2033–2049. https://doi.org/10.1016/j.ejor.2005.12.009

Silva Rodríguez, J., Díaz Cárdena, C., & Galindo Carabalí, J. (2017). Herramientas cuantitativas para la planeación y programación de la producción: estado del arte. Ingeniería Industrial. Actualidad y Nuevas Tendencias, 5(18), 99–114. https://www.redalyc.org/articulo.oa?id=215052403008

Taillard, É. D. (1993). Benchmarks for basic scheduling problems. European Journal of Operational Research, 64(2), 278–285. https://doi.org/10.1016/0377-2217(93)90182-M

Umam, M. S., Mustafid, M., & Suryono, S. (2021). A hybrid genetic algorithm and tabu search for minimizing *makespan* in flow shop scheduling problem. Journal of King Saud University - Computer and Information Sciences, 34(9), 7459–7467. https://doi.org/10.1016/j.jksuci.2021.08.025

Yan, S., Zhang, G., Sun, J., & Zhang, W. (2023). An improved ant colony optimization for solving the flexible job shop scheduling problem with multiple time constraints. Mathematical Biosciences and Engineering, 20(4), 7519-7547. https://doi.org/10.3934/mbe.2023325

Yang, X. S. (2020). *Nature-inspired optimization algorithms (2nd ed.).* Academic Press.

Zaied, A. N. H., Ismail, M. M., & Mohamed, S. S. (2021). Permutation flow shop scheduling problem with *makespan* criterion: Literature review. Journal of Theoretical and Applied Information Technology, 99(4), 830-848. https://www.jatit.org/volumes/Vol99No4/6Vol99No4.pdf

Zeidan, M. A., Al-Abaji, M. A., & Ahmed, M. H. (2024). Improved particle swarm algorithm for permutation flow shop scheduling problems. Investigación Operacional, 42(2), 195-203. https://revistas.uh.cu/invoperacional/article/view/9124/7847