Pontificia Universidad Javeriana Cali
Facultad de Ingeniería y Ciencias

Diseño de una Mateheurística para Resolver el Flexible Job Shop Problem with Sequence-Dependent Setup Times

Autor: Jhoan Arley Duque Otabo

Director: Daniel Morillo-Torres, Ph.D.

Cali, Colombia
agosto, 2026

Abstract

El *Flexible Job Shop Scheduling Problem* (FJSSP) es un referente clásico de la optimización combinatoria por su capacidad para modelar entornos industriales con máquinas o recursos polivalentes y complejidad en la secuenciación de las actividades operacionales. En este trabajo se aborda una variante del FJSSP que incorpora*Sequence-Dependent Setup Times* (SDST), una característica fundamental para representar de manera más realista los procesos productivos, pero que incrementa significativamente la complejidad computacional del problema. Para resolver esta variante, se propone una Mateheurística que integra un Algoritmo Genético (GA), encargado de la búsqueda global del espacio de soluciones, con un modelo de Programación Lineal Entera Mixta (MILP) incorporado como operador de diversidad. Este operador resuelve subproblemas de asignación y secuenciación mediante una estrategia de división en lotes de operaciones, permitiendo mejorar la calidad de las soluciones generadas por el GA. La eficacia del enfoque se evalúa mediante dos estrategias de integración GA–MILP. La primera incorpora un operador MILP independiente, en el que cada lote se optimiza de forma aislada. La segunda utiliza un operador MILP dependiente, que incorpora restricciones derivadas de los lotes previamente optimizados para orientar la búsqueda del GA hacia regiones más prometedoras del espacio de soluciones. Los resultados computacionales obtenidos sobre un conjunto de 276 instancias adaptadas muestran que la Mateheurística propuesta obtiene mejor desempeño dentro de las alternativas evaluadas y bajo las condiciones experimentales definidas al modelo MILP como al GA ejecutados de forma independiente, especialmente en instancias de mayor complejidad, caracterizadas por altos niveles de flexibilidad.

Palabras Clave: Flexible Job Shop Scheduling, Mateheurística, Sequence Dependent Setup Times, Algoritmo Genético, Programación Lineal Entera Mixta.

The *Flexible Job Shop Scheduling Problem* (FJSSP) is a classical reference in combinatorial optimization due to its ability to model industrial environments with multifunctional machines or resources and complex sequencing of operational activities. This work addresses a variant of the FJSSP that incorporates *Sequence-Dependent Setup Times* (SDST), a key feature for more realistically representing production processes, but one that significantly increases the computational complexity of the problem.To address this variant, a matheuristic is proposed that integrates a Genetic Algorithm (GA), responsible for the global exploration of the solution space, with a Programación Lineal Entera Mixta (MILP) model incorporated as a diversity operator. This operator solves assignment and sequencing subproblems through a batching strategy of operations, allowing improvements in the quality of the solutions generated by the GA. The effectiveness of the approach is evaluated through two GA–MILP integration strategies. The first incorporates an independent MILP operator, in which each batch is optimized separately. The second uses a dependent MILP operator that incorporates constraints derived from previously optimized batches, guiding the GA search toward more promising regions of the solution space. The computational results obtained on a set of 276 adapted instances suggest that, under the experimental conditions considered, the pro-

posed matheuristic achieves better performance than both the standalone MILP model and the standalone GA, particularly on more complex instances characterized by high flexibility levels..

Keywords: Flexible Job Shop Scheduling, Matheuristic, Sequence Dependent Setup Times, Genetic Algorithm, Mixed-Integer Linear Programming.

Contents

1. Introducción

La programación de la producción puede definirse formalmente como el proceso de toma de decisiones que asigna operaciones a los diferentes recursos disponibles (máquinas, equipos y personal) para fabricar un conjunto de productos, determinando los períodos de tiempo en que dichas operaciones deben iniciarse (Framinan et al., 2014). Esta área constituye una actividad clave en la planificación de la mayoría de los sistemas productivos y logísticos (Pinedo, 2022). En particular, una programación adecuada permite un uso inteligente de recursos escasos y de la capacidad de producción, determinando en gran medida el rendimiento operativo y la eficiencia global del sistema (Ghasemi et al., 2024). Por ello, garantizar una programación óptima es un objetivo central para las organizaciones, aunque en muchos sistemas alcanzarla resulta sumamente complejo. Dentro de estos problemas destaca el denominado Job Shop Scheduling Problem (JSSP), un referente clásico de la optimización combinatoria (Xiong et al., 2022) que modela entornos reales tan diversos como los talleres metalmecánicos, la fabricación de semiconductores o el flujo de pacientes en un hospital (Dauzère-Pérès et al., 2024). En este problema, un conjunto de J trabajos (también llamados actividades o jobs), que consisten a su vez en un conjunto determinado de H_j operaciones cada uno, deben ser procesados por un conjunto de I máquinas o recursos. Cada conjunto de operaciones de un trabajo tiene asignado un orden de procesamiento específico en las máquinas, diferente entre trabajos, y el objetivo es secuenciar las actividades bajo la optimización de un determinado criterio, generalmente relacionado con el tiempo de finalización del proyecto. Este problema se encuentra en la categoría NP-hard, debido a su naturaleza combinatoria (Xiong et al., 2022). Es decir, no es viable computacionalmente encontrar una solución óptima a medida que el problema crece en tamaño. Requiriendo la adopción de métodos que encuentren buenas soluciones en una cantidad de tiempo práctico, en lugar de garantizar la obtención de una solución óptima.

Una de las variantes del JSSP más relevantes es el denominado Flexible Job Shop Scheduling Problem (FJSSP), generalmente presente en sistemas de fabricación flexible (Dauzère-Pérès et al., 2024; Li et al., 2022). Su principal diferencia es que algunas (o todas) las operaciones pueden procesarse en más de una máquina; específicamente, cada operación de una actividad tiene un conjunto de máquinas donde puede ejecutarse. Conceptualmente, esto es similar a tener distintos modos de ejecución cuando las máquinas en las que se pueden realizar las operaciones son compartidas por otras operaciones del sistema. Esto implica una mayor dificultad para resolver el problema debido a que no solo se debe secuenciar las operaciones de las actividades, sino elegir el modo de ejecución adecuado (asignación máquina-operación). Debido a esta complejidad y aplicabilidad en problemas reales, la literatura en relación al sistema FJSSP se ha incrementado sostenidamente en los últimos años, como se observa en la Fig. 1.

La flexibilidad del FJSSP ha dado lugar a variantes que pueden considerar distintas restricciones para abordar un entorno aún más realista de producción. Algunos trabajos incorporan tiempos de procesamiento inciertos, tiempos de transporte, interrupciones en las máquinas o tiempos de configuración. Estas restricciones han sido abordadas de forma más limitada en comparación con el FJSSP clásico, debido a que la literatura ha seguido mayoritariamente un enfoque *bottom-up*, es decir, en lugar de desarrollar marcos de resolución generales, la mayoría de los trabajos se concentran en resolver aplicaciones industriales muy

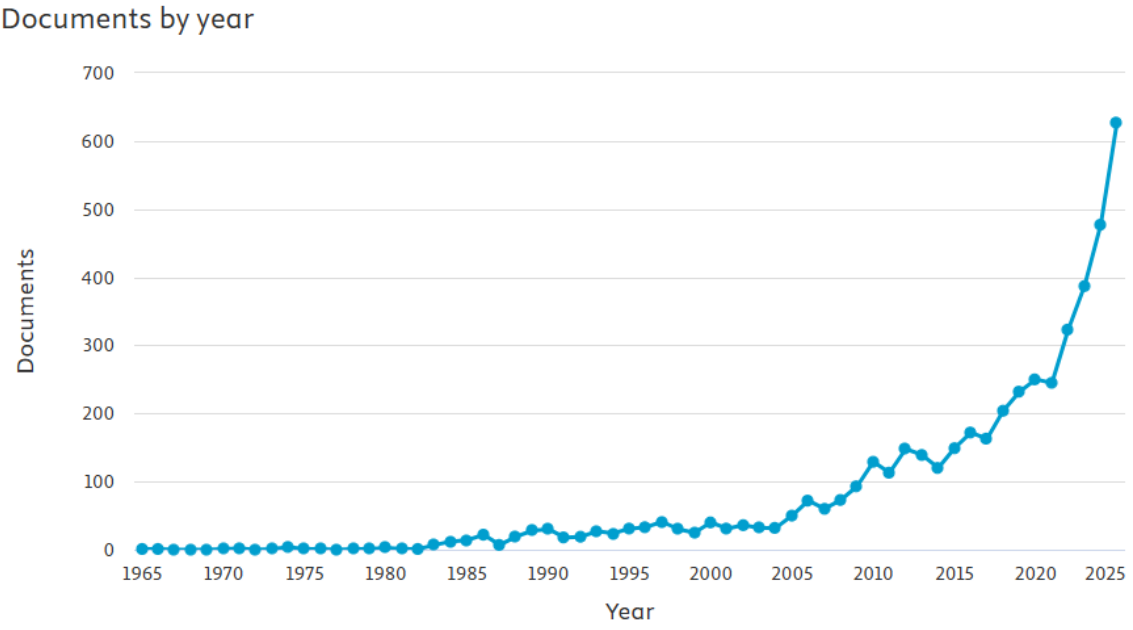


Figura 1: Publicaciones relacionadas con el FJSSP (basado en un análisis de 4 000 artículos en Scopus)

específicas, diseñando soluciones a la medida de cada caso particular. Como resultado, el campo se ha fragmentado en una gran cantidad de variantes aisladas, cada una con sus propias restricciones y criterios (Dauzère-Pérès et al., 2024; Li et al., 2022).

Una de las características adicionales a destacar es la inclusión de *Sequence-Dependent Setup Times* (SDST), implicando un tiempo de ajuste o preparación entre el cambio de un trabajo a otro, programados en la misma máquina (Shen et al., 2018); a este problema se le conoce como el FJSP-SDST. Si bien estos tiempos suelen considerarse dentro de la propia duración de la actividad, esto no es posible cuando los tiempos dependen explícitamente de la secuencia de programación. La investigación que incorpora este tipo de restricciones demuestra que una modelación más precisa de los tiempos de configuración puede generar ahorros significativos y mejoras en el desempeño de la programación de la producción. Además, se consideran como uno de los aspectos más difíciles de los problemas de programación, tanto por su impacto en la calidad de los programas como por la complejidad que añaden a los algoritmos de resolución (Allahverdi & Allahverdi, 2026; Allahverdi et al., 2008).

La aplicabilidad de los SDST se presenta en sectores con alta variedad de productos. Por ejemplo, en la manufactura de cerámicas, donde el cambio de moldes para distintos formatos puede consumir turnos completos de trabajo (Framinan et al., 2014). De igual forma, en el ensamblaje de tarjetas de circuito impreso (PCB) y en la industria textil, donde se ha reportado que los SDST pueden representar entre el 20 % y el 50 % de la capacidad de producción perdida (Allahverdi, 2015) . En adición, casos documentados reportaron grandes beneficios económicos al reducir el tiempo de configuración. Por ejemplo, en una planta de montaje de placas de circuito impreso, por ejemplo,una reducción del 80 % de estos tiempos

generó un ahorro de 1,8 millones de dólares (Trovinger & Bohn, 2005). De manera similar, Loveland et al. (2007) abordaron en Dell Inc. el problema de programación de la producción con el objetivo de minimizar el tiempo de configuración, logrando hasta un 35 % más de volumen de producción y un ahorro superior al millón de dólares. Estos hechos motivan el desarrollo de nuevas investigaciones en torno al FJSP-SDST.

Entre las metodologías más usadas para resolver el FJSSP, se encuentran los algoritmos poblacionales, específicamente los Algoritmos Genéticos (GA) (Amjad et al., 2018; Dauzère-Pérès et al., 2024; Gao et al., 2019). Estos algoritmos se caracterizan por su capacidad para explorar amplios espacios de búsqueda, dirigiéndose hacia subespacios promisorios. A pesar de sus ventajas, los GA presentan un desafío crítico: tienden a converger prematuramente, es decir, pierden diversidad poblacional demasiado rápido y quedan atrapados en óptimos locales antes de explorar adecuadamente el espacio de búsqueda (Liu et al., 2021). Este inconveniente surge debido a la rápida aparición y dominancia de poblaciones de individuos superiores, que guían al algoritmo hacia una convergencia temprana antes de explorar eficientemente el espacio de búsqueda. En consecuencia, se han llevado a cabo diversos estudios centrados en mejorar los GAs para el FJSSP, destacando la utilización de algoritmos híbridos que combinan el GA con otras técnicas (Liu et al., 2021; Meng et al., 2023; Sun et al., 2023). En este trabajo se aborda el FJSSP-SDST mediante una Mateheurística que emplea un GA, un Programación Lineal Entera Mixta (MILP) como operador de diversidad para mejorar la precisión y la eficiencia del algoritmo. Este enfoque híbrido busca aprovechar las fortalezas de ambos métodos, combinando la capacidad exploratoria del GA y la capacidad de optimización exacta del MILP, con el objetivo de obtener soluciones más precisas y eficientes para el problema FJSSP.

El documento se organiza de la siguiente manera. En la sección 2 se describe formalmente el FJSSP-SDST, incluyendo sus supuestos, un ejemplo ilustrativo y la formulación matemática mediante un modelo MILP. La sección 3 presenta una revisión de la literatura, abordando la complejidad del problema, los trabajos que incorporan los SDST y los métodos de solución propuestos en los últimos años, con énfasis en los algoritmos genéticos híbridos y las Mateheurísticas. En la sección 4 se detalla la Mateheurística propuesta, describiendo la codificación y decodificación de las soluciones, los operadores del algoritmo genético y el operador MILP por lotes con sus dos estrategias de integración. La sección 5 presenta los resultados computacionales, incluyendo la generación de los casos de prueba, la parametrización del GA mediante el método de Taguchi, la selección del tamaño de lote y la comparación entre los métodos propuestos. Finalmente, en la sección 6 se exponen las conclusiones de la investigación y las líneas de trabajo futuro.

1.1. Objetivos de la Investigación

1.1.1. Objetivo General

Diseñar y evaluar una Mateheurística basada en un algoritmo genético que combine un modelo matemático exacto para resolver el FJSSP-SDST, con el objetivo de disminuir el tiempo total de ejecución de un proyecto (Makespan).

1.1.2. Objetivos Específicos

- Identificar los principales componentes de diseño y ejecución de un algoritmo genético, mediante la revisión de la literatura científica.
- Desarrollar un modelo matemático que permita integrarse como un operador decodificador de un algoritmo genético, al solucionar el problema de asignación del FJSSP-SDST.
- Desarrollar e implementar una Mateheurística que dé solución al problema FJSSP-SDST, integrando como base un algoritmo genético.
- Validar el método de solución propuesto mediante la medición de indicadores de desempeño en bases de datos disponibles de la literatura del FJSSP.
- Comparar el desempeño de la Mateheurística frente al algoritmo genético puro y el MILP, sobre instancias de la literatura adaptadas con tiempos de configuración, evaluando tanto la calidad de las soluciones como el tiempo de cómputo.

1.2. Alcance y limitaciones

El alcance de esta investigación se limita al estudio del FJSSP-SDST con el objetivo de minimizar el makespan de un proyecto, considerando casos de prueba con datos discretos en los que cada operación puede ser procesada en un conjunto de máquinas. Para abordar este problema, se propone el diseño de una metaheurística que consiste en la integración de un GA y un modelo MILP. Este enfoque constituye el aporte novedoso de este estudio, ya que integra el MILP dentro del GA mediante una estrategia por lotes de operaciones, dotando de diversidad a la población. Además, los hiperparámetros del GA se calibran a través de un diseño de experimentos de Taguchi. La contribución central analiza el efecto de incorporar el MILP al GA mediante dos estrategias por lotes (entendiendo por lote la división del problema en n partes, cada una con un conjunto de operaciones): en la primera, las soluciones de los lotes se fijan progresivamente, mientras que en la segunda, los lotes se resuelven de manera independiente y luego se integran consolidando la solución completa. Para la evaluación, se emplean casos de prueba ampliamente utilizados en la literatura del FJSSP que no consideran los SDST, por lo que dichos tiempos son generados y adaptados para este estudio; los resultados se comparan en términos del makespan y del MRE, tomando como referencia el *Lower Bound* de cada caso de prueba en su forma original de FJSSP. El problema abordado es aplicable a la programación de la producción en industrias metalmecánicas, farmacéuticas, alimentarias y de manufactura flexible.

Por otra parte, las limitaciones se asocian a que el estudio se centra exclusivamente en la minimización del makespan y no considera objetivos adicionales como la tardanza total, los costos de producción, la utilización de recursos, el consumo energético, la robustez frente a la incertidumbre, entre otros. Los datos utilizados fueron generados artificialmente mediante distribuciones aleatorias, por lo cual podrían no representar el comportamiento en entornos reales de producción. El desempeño de los algoritmos propuestos depende de la calibración de sus hiperparámetros, la cual fue realizada para el conjunto de datos artificiales abordado en este estudio y que podría ser distinta en entornos reales de producción. Finalmente, el modelo MILP, al aplicarse a un conjunto de operaciones (lotes), no garantiza la optimalidad

global del caso de prueba, al igual que los algoritmos GA y la mateheurística. También debe considerarse que los resultados pueden verse influenciados por las características particulares de cada caso de prueba y por la capacidad computacional utilizada.

1.3. Contribuciones de la investigación

Las contribuciones de la tesis se resumen en orden de importancia a continuación:

- Se diseñó un método de solución novedoso para el FJSSP-SDST que combina un Algoritmo Genético (GA) con un modelo MILP por lotes, dando lugar a una Mateheurística. La integración del componente exacto se basa en dividir la secuencia global en lotes de operaciones de los cromosomas del GA (subproblemas), de manera que el modelo pueda resolver cada conjunto en tiempos de cómputo reducidos, aprovechando tanto la información del individuo que brinda el GA como la optimización sobre un subproblema que otorga el modelo lineal. Bajo este enfoque se consideran dos estrategias de integración. En la primera, las soluciones de los lotes ya resueltos se fijan añadiendo restricciones al modelo, de modo que cada lote solucionado se incorpora de manera permanente y se procede iterativamente hasta completar la solución. En la segunda, cada lote se resuelve de forma independiente y posteriormente se integran los resultados parciales para reconstruir la programación completa.
- Se desarrolló un algoritmo genético en el que se seleccionaron y evaluaron los métodos más eficientes para los operadores de selección, cruce y mutación, identificando cuáles se adaptan mejor a los casos de prueba del FJSSP-SDST con baja, media y alta flexibilidad. Los componentes restantes, como la generación de la población inicial y el operador de reemplazo, se definieron en función de su integración posterior con la Mateheurística.
- Creación de nuevos casos de prueba para el FJSSP-SDST a partir de los conjuntos más utilizados en la literatura del FJSSP, a los cuales se les incorporan los SDST mediante una distribución aleatoria uniforme que garantiza el cumplimiento de la desigualdad triangular, condición que asegura la consistencia de los SDST.

2. Descripción del Problema

Formalmente, el FJSP-SDST consiste en realizar un conjunto de trabajos o actividades $J = \{0, \ldots, j, \ldots, N\}$, cada uno compuesto por un conjunto de operaciones $H_j = \{1, \ldots, h, \ldots, O_j\}$, que deben realizarse en un conjunto de máquinas $I = \{1, \ldots, M'\}$ según un orden específico, es decir, una operación O_j solo puede programarse si la operación o_{j-1} ha sido programada. Estas operaciones pueden ser ejecutadas en uno de los diferentes modos disponibles; estos se refieren a que una operación puede ejecutarse en diferentes máquinas. Adicionalmente, se consideran los tiempos de configuración (setup time) al cambiar de un trabajo a otro en una determinada máquina; este tiempo es dependiente de la actividad anteriormente programada (la secuencia); por tanto, existe un tiempo de preparación distinto para cada par de trabajos que se ejecutan consecutivamente. Este tiempo se denota como S_{ijl}, que representa el tiempo de preparación requerido al programar primero la actividad j y luego la actividad l en la máquina i. El objetivo es asignar cada operación a una máquina

de forma que minimice el tiempo de procesamiento de todo el proyecto, también llamado en la literatura como *makespan*.

El FJSP-SDST considera los siguientes supuestos:

- Se considera que no existen tiempos de configuración dependientes de la secuencia entre operaciones h_j, sino únicamente entre trabajos (Saidi-Mehrabad & Fattahi, 2007).
- Los trabajos son independientes entre sí, al igual que las máquinas.
- Una máquina puede procesar como máximo una operación a la vez y, por tanto, un solo trabajo.
- Los SDST son conocidos y constantes.
- Los tiempos de procesamiento pueden variar o no según el modo de ejecución.
- Cada una de las operaciones h_j debe ser ejecutada sin interrupciones; es decir, una vez que se inicia, debe ser ejecutada completamente.
- Todas las máquinas están disponibles desde el principio de la programación.
- Se permite la recirculación, es decir, que un trabajo puede requerir una máquina más de una vez.

Un aspecto a considerar en este problema es el nivel de flexibilidad; este está determinado por el promedio del número de modos por operación de las actividades. Cuando este promedio es igual al número total de máquinas, se tiene flexibilidad total, lo que significa que todas las operaciones pueden ejecutarse en cualquier máquina. En caso contrario, se tiene flexibilidad parcial. A mayor flexibilidad, mayor nivel de complejidad para realizar tanto la secuenciación como la selección del modo apropiado de ejecución para cada operación.

Un ejemplo del FJSP-SDST se muestra en la Tabla 1, donde un proyecto está compuesto por tres trabajos y dos máquinas (M1, M2). Cada trabajo j se compone de dos operaciones h, las cuales deben ser programadas para que el trabajo se considere finalizado. Las operaciones h pueden tener uno o varios modos de ejecución. Las columnas indican la operación y las filas los trabajos. Cada operación está asociada con un conjunto de máquinas que pueden ejecutarla con un tiempo de procesamiento específico (indicado en paréntesis). Así, por ejemplo, la operación $h = 2$ del trabajo $j = 3$ que puede realizarse en la máquina 1 (M1) con un tiempo de procesamiento de 10 unidades o en la máquina 2 (M2) con un tiempo de 9 unidades. En el ejemplo, casi todas las operaciones tienen dos modos de ejecución, es decir, dos máquinas donde pueden ejecutarse; sin embargo, la operación 2 del trabajo 1 cuenta con un solo modo. Al calcular el promedio total de modos por operación se obtiene un nivel de flexibilidad de $(2 + 1 + 2 + 2 + 2 + 2)/6 = 1{,}8$, lo que corresponde a flexibilidad parcial. Si dicha operación también tuviera dos modos, el promedio sería igual al número total de máquinas (2), y el caso se clasificaría como de flexibilidad total. La Tabla 2 muestra los SDST para cada máquina. Cada tiempo, cumple con la desigualdad triangular: para cada pareja J_l, J_k de trabajos se cumple que $S_{ilj} \leq S_{ilk} + S_{ikj}, \forall k \in J, i \in I$. Por ejemplo, el conjunto de trabajos $(3, 2)$ para la máquina 1 y el trabajo intermedio 1 cumple que $S_{1,3,2} \leq S_{1,3,1} + S_{1,1,2}$ $(4 \leq 3 + 2)$.

Tabla 1: Tiempos y modos de procesamiento.

	Operaciones	
Trabajos	1	2
1	M1(7)	M2(9)
	M2(7)	
2	M1(8)	M1(7)
	M2(7)	M2(11)
3	M1(7)	M1(10)
	M2(5)	M2(9)

Tabla 2: Tiempos de preparación.

	M1			M2		
Trabajos	1	2	3	1	2	3
1	0	2	3	0	2	4
2	3	0	5	5	0	3
3	3	4	0	5	3	0

Siguiendo la notación J_{jh}, que representa la ejecución de la operación $h \in H_j$ del trabajo $j \in J$, la solución óptima de este ejemplo se muestra a continuación. La secuencia para la máquina 1 es $J_{1,1}, J_{2,1}, J_{2,2}$, y para la máquina 2 es $J_{3,1}, J_{3,2}, J_{1,2}$. Esta solución se muestra en el diagrama de Gantt en la Fig. 2.

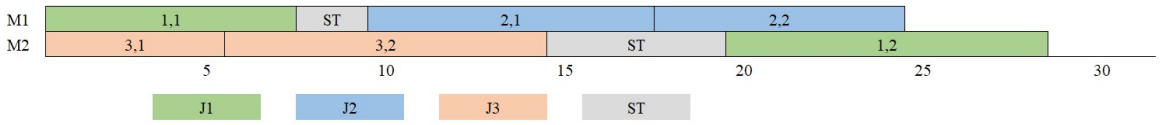


Figura 2: Diagrama de Gantt de la solución óptima para un FJSSP-SDST.

2.1. Formulación MILP del FJSP-SDTS

A continuación, se presenta la formulación matemática mediante un modelo MILP adaptado de Choi y Choi (2002) para el FJSP-SDTS. Inicialmente se presentan los conjuntos, los parámetros, las variables de decisión y finalmente el modelo completo.

Conjuntos

- $J := \{1, \ldots, N\} \cup \{0\}$: conjunto de N trabajos o jobs, se incluye el job ficticio 0 para indicar el inicio en cada una de las máquinas. Todos los tiempos de configuración y procesamiento relacionados con el job ficticio se establecen en cero.
- $H_j := \{1, \ldots, O_j\}$: conjunto de O_j operaciones para cada job $j \in J$.
- $I := \{1, \ldots, M'\}$: conjunto de M' máquinas disponibles.

Parámetros

A partir de aquí se usará la notación R_{jh} para hacer referencia a la operación $h \in H_j$ del job $j \in J$.

- S_{ijk}: tiempo de preparación de la máquina $i \in I$ al programar el job $j \in J$ y luego el $k \in J$, de forma que no existe ninguna otra actividad entre las dos.
- P_{ijh}: Tiempo de procesamiento de la operación R_{jh} en la máquina $i \in I$.

- $a_{ijh}: \begin{cases} 1, & \text{si la operación } R_{jh} \text{ puede realizarse en la máquina } i \in I \\ 0, & \text{de lo contrario} \end{cases}$
- M: un número suficientemente grande.

Variables de decisión

- $Y_{ijh}: \begin{cases} 1, & \text{si la operación } R_{jh} \text{ es realizada en la máquina } i \in I \\ 0, & \text{de lo contrario} \end{cases}$
- $X_{ijhkl}: \begin{cases} 1, & \text{si la operación } R_{jh} \text{ (del trabajo } j) \text{ se programa justo antes de la} \\ & \text{operación } R_{kl} \text{ (del trabajo } k) \text{ en la máquina } i \in I \\ 0, & \text{de lo contrario} \end{cases}$
- T_{jh}: tiempo de inicio de la operación R_{jh}.
- F_{jh}: tiempo de finalización de la operación R_{jh}.
- C_{max} tiempo de finalización de todo el proyecto (makespan).

Función objetivo

$$\text{Minimize } z = C_{max} \tag{1}$$

Sujeto a

$$F_{j(O_j)} \leq C_{max}, \qquad \forall j \in J \tag{2}$$

$$T_{0h} = 0, \qquad \forall h \in H_0 \tag{3}$$

$$T_{jh} + Y_{ijh} * P_{ijh} \leq F_{jh}, \qquad \forall i \in I, \forall j \in J, \forall h \in H_j \tag{4}$$

$$F_{j(h)} \leq T_{j(h+1)}, \qquad \forall j \in J, \forall h \in H_j \setminus \{O_j\} \tag{5}$$

$$\sum_{i \in I} Y_{ijh} = 1, \qquad \forall j \in J \setminus \{0\}, \forall h \in H_j \tag{6}$$

$$Y_{ijh} \leq a_{ijh}, \qquad \forall i \in I, \forall j \in J, \forall h \in H_j \tag{7}$$

$$\sum_{j \in J} \sum_{h \in H_j} X_{ijhkl} = Y_{ikl}, \qquad \forall i \in I, \forall k \in J \setminus \{0\}, \forall l \in H_k \tag{8}$$

$$\sum_{k \in J} \sum_{l \in H_k} X_{ijhkl} \leq Y_{ijh} \qquad \forall i \in I, \forall j \in J, \forall h \in H_j \tag{9}$$

$$X_{ijhjh} = 0, \qquad \forall i \in I, \forall j \in J, \forall h \in H_j \tag{10}$$

$$F_{jh} + S_{ijk} \leq T_{kl} + (1 - X_{ijhkl}) * M, \qquad \forall j \in J, k \in J \setminus \{0\}, \forall h \in H_j, \forall l \in H_k, \forall i \in I \tag{11}$$

$$F_{jh} \leq T_{jh} + P_{ijh} + (1 - Y_{ijh}) * M, \qquad \forall i \in I, \forall j \in J, \forall h \in H_j \tag{12}$$

$$X_{ijhkl} \in \{0, 1\}, \qquad \forall j, k \in J, \forall h \in H_j, \forall l \in H_k, \forall i \in I \tag{13}$$

$$Y_{ijh} \in \{0, 1\}, \qquad \forall i \in I, \forall j \in J, \forall h \in H_j \tag{14}$$

$$T_{jh} \geq 0, \qquad \forall j \in J, \forall h \in H_j \tag{15}$$

$$F_{jh} \geq 0, \qquad \forall j \in J, \forall h \in H_j \tag{16}$$

$$C_{max} \geq 0 \tag{17}$$

La función objetivo se representa en la ecuación (1), que busca la minimización del makespan. Las restricciones (2) determinan el Makespan del sistema, considerando el tiempo de finalización de la última operación de cada job. Las restricciones (3) establecen el tiempo de inicio del trabajo ficticio. Las restricciones (4) garantizan que el tiempo de finalización de una operación R_{jh} sea en un instante superior a su tiempo de procesamiento más su tiempo de inicio. Por otra parte, las restricciones (5) garantizan la dinámica de la secuencia entre operaciones de una actividad (precedencia entre operaciones de un mismo job). Las restricciones (6) y (7) garantizan la ejecución de un solo modo de la operación R_{jh}; a su vez, validan que el modo esté permitido. Las restricciones (8) y (9) determinan que solo debe existir una operación R_{jh} que preceda a la operación R_{kl}. Las restricciones (10) garantizan que ninguna operación pueda ser predecesora de sí misma. Por su parte, las restricciones (11) establecen que cuando se programen de forma consecutiva (inmediata o no) dos operaciones R_{jh} y luego R_{kl}, la operación R_{kl} no podrá iniciar hasta que la operación R_{jh} haya finalizado, incluyendo el tiempo de preparación. Las restricciones (12) obligan a que el tiempo de finalización de una operación corresponda a la suma del tiempo de inicio y su tiempo de procesamiento. Finalmente, las restricciones (13) a (17) corresponden a las definiciones de dominio de las variables.

3. Revisión de Literatura

En esta sección se presenta una revisión breve de la literatura sobre el FJSSP considerando tiempos de alistamiento dependientes de la secuencia (SDST). En particular, se analizan los métodos de solución más eficientes propuestos en los últimos años y se identifican sus principales características, ventajas y limitaciones. Para una revisión más exhaustiva del tema, se remite al lector a Amjad et al. (2018), Dauzère-Pérès et al. (2024) y Gao et al. (2019).

3.1. Complejidad del FJSSP

EL FJSSP fue introducido por primera vez por Brucker y Schlie (1990), quienes consideraron máquinas polivalentes equipadas con diferentes herramientas. Aplicaron un algoritmo polinomial para resolver solo dos trabajos. No obstante, dentro de sus recomendaciones, indicaban que era necesario el desarrollo de métodos heurísticos para resolverlo con un mayor número de trabajos y encontrar soluciones pseudo-óptimas. Esto se debe a que el FJSSP, al ser una variante del clásico JSSP, está catalogado en el conjunto de complejidad computacional NP-Hard; lo que implica que, hasta ahora, el tiempo de cómputo necesario para resolverlo crece de forma exponencial con el número de trabajos (Framinan et al., 2014). La complejidad real del FJSSP radica en que a la secuenciación se añade la asignación de máquinas: mientras que en el JSSP clásico solo se determina el orden de procesamiento de los trabajos, en el FJSSP es necesario decidir simultáneamente en qué máquina se procesa cada operación y en qué orden. El espacio de búsqueda resultante combina el crecimiento factorial de las secuencias ($n!$) con el crecimiento exponencial de las asignaciones (m^n), de modo que los métodos exactos no pueden abordar instancias de la industria de más de 15 trabajos con un nivel de flexibilidad alto (Framinan et al., 2014).

3.2. FJSS con SDST

Una de las variantes más relevantes es aquella que considera los tiempos de preparación dependientes de la secuencia (SDST). La importancia de esta característica fue reconocida mucho antes de la aparición del propio FJSSP (Panwalkar et al., 1973). Los autores observaron que una parte significativa de los trabajos requiere tiempos de configuración y que excluirlos de la programación puede disminuir la ventaja competitiva de una compañía. Esta hipótesis se ha reforzado con el tiempo; en su cuarta revisión exhaustiva de problemas de programación con tiempos de preparación, Allahverdi y Allahverdi (2026) muestran que, en diversos entornos de manufactura, los tiempos de alistamiento pueden representar entre el 20 % y el 50 % de la capacidad perdida. Lo que implica que su planificación es crítica para integrar adecuadamente la planeación de procesos y la programación de la producción; no tomarlos en cuenta conduce a cronogramas poco realistas y a un uso ineficiente de recursos, incluso en sistemas de manufactura distribuida y en servicios como la salud.

En la literatura sobre programación de operaciones, los tiempos de configuración han sido abordados de manera recurrente en problemas clásicos de secuenciación, particularmente en entornos de producción tipo flow shop y job shop. Sin embargo, su incorporación en el contexto del FJSSP ha sido limitada. A partir de una revisión bibliográfica en Scopus para el periodo 2 000 a 2 025, se identificaron 1 230 publicaciones relacionadas con tiempos de configuración

en estos tres contextos. De este total, aproximadamente el 38,9 % corresponde a estudios en FSP, el 50,1 % a job shop scheduling y solo el 11,0 % al FJSSP. Esta menor proporción muestra que la literatura sobre FJSSP ha tendido a concentrarse en la flexibilidad de asignación y secuenciación, mientras que la integración explícita de tiempos de configuración ha permanecido comparativamente menos desarrollada; por tanto, su incorporación constituye una línea pertinente de investigación hacia modelos de programación más realistas.

Desde el punto de vista de la modelación, los SDST se clasifican habitualmente en dos tipos: unidos y separados (Defersha & Chen, 2010). Los SDST unidos indican que el tiempo de configuración solo puede realizarse una vez que el trabajo llega a la máquina, mientras que los SDST separados permiten que, si la máquina está disponible, pueda empezar su tiempo de configuración sin que el trabajo haya llegado. La elección entre uno u otro depende de los requisitos reales de la compañía abordada. Sin embargo, las revisiones exhaustivas de problemas de programación con tiempos de configuración indican que, en la literatura de programación de tareas, estos tiempos se modelan mayoritariamente como separados; de hecho, cuando no se especifica el tipo de tiempo, suele asumirse por defecto que se trata del separado (Allahverdi & Allahverdi, 2026; Allahverdi et al., 2008). En coherencia con esta práctica y con las características del caso de estudio, en este trabajo se consideran únicamente SDST separados.

3.3. Métodos de solución para el FJSSP-SDST

De manera general, Sergienko et al. (2009) distinguen tres familias de técnicas de optimización para problemas combinatorios según el tipo de solución que producen: algoritmos exactos, aproximados y heurísticos. Los primeros garantizan en tiempo finito el retorno de una solución óptima o la prueba de insatisfactibilidad; los segundos devuelven soluciones alternativas cuya calidad puede acotarse mediante estimaciones de precisión *a priori* o *a posteriori*; y los terceros operan sin garantías formales de exactitud, pero suelen ser la única alternativa práctica para abordar problemas complejos dentro de tiempos razonables. Esta clasificación proporciona el marco conceptual a partir del cual se organiza la gran diversidad de algoritmos propuestos en la literatura.

Sobre esta base, los mismos autores identifican siete grandes grupos de técnicas: (1) algoritmos secuenciales o constructivos, que generan soluciones paso a paso siguiendo reglas simples y suelen ser muy rápidos, aunque con calidad limitada; (2) métodos de búsqueda local determinista, que mejoran iterativamente una solución inicial explorando su vecindario inmediato según la función objetivo; (3) métodos de búsqueda local estocástica, que introducen aleatoriedad en la aceptación de soluciones para escapar de óptimos locales (por ejemplo, *simulated annealing*); (4) métodos de *swarm intelligence* (SI), inspirados en el comportamiento colectivo de sistemas naturales como colonias de hormigas o enjambres de partículas; (5) algoritmos evolutivos (EA), que emplean principios de evolución biológica como selección, cruce y mutación, siendo el algoritmo genético (GA) su exponente más representativo; (6) métodos de exploración o *scanning methods*, basados en poblaciones que orientan la dirección de búsqueda a partir de múltiples alternativas en cada iteración; y (7) métodos especiales, que utilizan o adaptan enfoques exactos (como *branch-and-bound* o programación dinámica) para explotar estructuras particulares del problema.

Para resolver problemas pequeños del FJSSP-SDST, por ejemplo, instancias con alre-

dedor de seis trabajos y dos máquinas, los métodos exactos como MILP pueden encontrar la solución óptima en tiempos razonables (cortos). Sin embargo, a medida que el número de trabajos y máquinas aumenta, la complejidad exponencial del problema hace que estos enfoques se vuelvan ineficientes, lo que justifica el uso de algoritmos que buscan soluciones pseudo-óptimas. En la revisión de Gao et al. (2019), se observa que las técnicas aproximadas más empleadas pertenecen a las categorías (4) SI y (5) EA. Más del 60 % de las publicaciones revisadas se basan en estos enfoques bio-inspirados, y el GA se identifica como el algoritmo individual más utilizado para abordar el FJSSP, hecho que ha motivado revisiones de literatura específicas sobre el uso de GA en el FJSSP (Amjad et al., 2018; Huang et al., 2022) y sobre su aplicación en problemas de programación en general (Ying et al., 2026).

Desde el punto de vista metodológico, las principales fortalezas del GA radican en su elevada capacidad exploratoria y su robustez frente a espacios de búsqueda de gran escala y alta complejidad, produciendo resultados competitivos en instancias de referencia del FJSSP (Amjad et al., 2018; Huang et al., 2022). Además, la simplicidad relativa de sus operadores básicos (selección, cruce y mutación), unida a su carácter poblacional, convierte al GA en un marco idóneo para la hibridación con técnicas de búsqueda local, como *tabu search* o *simulated annealing*, con el fin de intensificar la búsqueda en regiones prometedoras del espacio de soluciones (Gao et al., 2019). No obstante, la literatura también destaca limitaciones importantes: los GA son propensos a sufrir convergencia prematura, quedando atrapados en óptimos locales cuando la diversidad de la población disminuye demasiado rápido (Huang et al., 2022). Estas fortalezas y debilidades justifican el interés por esquemas híbridos en los que el GA actúa como motor de búsqueda global.

3.3.1. GA híbridos para el FJSSP-SDST

Tutumlu y Saraç (2023) estudian el FJSSP con *job-splitting*, en el que, además de la asignación y secuenciación habituales, debe determinarse cuántos sublotes se genera para cada trabajo y cuál es el tamaño de cada uno, con el objetivo de minimizar el makespan. Para este problema, inicialmente formulan un MILP que decide simultáneamente la ruta de cada trabajo, la asignación de operaciones a máquinas y la descomposición en sublotes sin imponer límites prefijados al número ni al tamaño de estos, resolviéndolo para instancias de tamaño pequeño. No obstante, para problemas más grandes el MILP resulta inviable por lo que diseñaron un algoritmo genético híbrido (HGA) en el que el GA actúa como motor de búsqueda global sobre una codificación que representa asignaciones y secuencias, mientras que un algoritmo de búsqueda local (LSA) especializado se integra en lugar del operador de mutación o aplicado sobre soluciones élite para ajustar de forma inteligente los tamaños de los sublotes y explorar en profundidad vecindades prometedoras. En la evaluación computacional, los autores comparan el HGA con un GA clásico y con el modelo MILP en dos escenarios (con y sin *job-splitting*), mostrando, por un lado, que permitir la división de trabajos reduce de manera significativa el makespan respecto al caso sin división (12.14 % y 8.57 % para instancias de 12 y 24 operaciones respectivamente). Por otro, el algoritmo híbrido obtiene soluciones de mejor calidad y más estables que el GA puro en instancias medianas y grandes donde el MILP no consigue soluciones factibles dentro del límite de tiempo, el Hibrído obtiene mejoras en el makespan con respecto al GA en 13.53 % con un tamaño de 24 operaciones, de 22.53 % en un tamaño de 50 operaciones y de 17.62 % en problemas de gran escala entre 150 a 200 operaciones. Estos resultados muestran la relevancia de los enfoques

híbridos basados en GA para variantes realistas del FJSSP, al evidenciar que la combinación entre la exploración global proporcionada por el GA y la explotación local de un procedimiento de búsqueda especializado sobre estructuras como los sublotes ofrece ventajas claras frente a utilizar únicamente un GA estándar o un modelo exacto aislado.

Xie et al. (2023) estudian un distributed flexible job shop scheduling problem (DFJSP), extensión del FJSSP en la que los trabajos deben asignarse a varias plantas geográficamente distribuidas y, dentro de cada planta, programarse en máquinas alternativas, con el objetivo de minimizar el makespan global del sistema. integra de forma conjunta la asignación de trabajos a fábricas, la selección de máquinas elegibles y la secuenciación de operaciones en cada recurso. Para abordarlo, los autores proponen un algoritmo híbrido de búsqueda tabú genético (HGTSA) que combina la capacidad de exploración global de un GA con la explotación local de una búsqueda tabú focalizada en las fábricas críticas; el esquema de codificación representa simultáneamente la asignación de trabajos a fábricas, la ruta máquina–operación y el orden de procesamiento, y se acompaña de operadores de cruce y mutación diseñados para preservar estructuras de solución prometedoras mientras se mantiene la diversidad poblacional. Sobre las soluciones generadas por el GA, la fase de TS se aplica selectivamente a la fábrica crítica identificada en cada iteración (aquella con mayor carga o contribución al makespan). En la evaluación computacional, realizada sobre 69 instancias de referencia para DFJSP, el HGTSA se compara con tres algoritmos híbridos que ya tienen de base al GA, logrando actualizar 13 límites superiores previamente reportados en la literatura y obteniendo mejores soluciones en 10 de las 23 instancias con dos fábricas, mientras que en los casos con cuatro fábricas alcanza los mejores valores conocidos o los límites inferiores en todas las pruebas consideradas. Estos resultados ponen de manifiesto que la combinación de exploración global mediante GA y explotación local mediante TS sobre fábricas críticas no solo mejora la calidad de las soluciones respecto a algoritmos puramente genéticos o puramente de búsqueda local, sino que también ofrece una mayor robustez y eficiencia computacional en escenarios distribuidos de escala industrial.

Wang y Zhu (2021) abordan un FJSSP extendido con SDST y tiempos de retardo entre operaciones (FJSP-SDST-LT), proponiendo un GA híbrido con una búsqueda tabú (HGA-TS) para minimizar el *makespan*. En su esquema, la búsqueda tabú se aplica a cada descendiente inmediatamente después del operador de cruce y antes de pasar a la siguiente iteración del ciclo evolutivo; esta búsqueda intensiva se define sobre cuatro estructuras de vecindad centradas en la ruta crítica, entendida como el camino de mayor duración en la representación en grafo de la solución y, por tanto, como el conjunto de operaciones que determina directamente el valor del *makespan*, de igual forma incorpora un mecanismo de diversificación que se activa tras varias iteraciones sin mejora, el algoritmo modifica aleatoriamente la posición de operaciones no críticas y de operaciones pertenecientes a rutas cuya duración iguala el *makespan*, alterando la estructura del cronograma y permitiendo explorar regiones alternativas del espacio de búsqueda. Si la falta de mejora persiste, se aplica un reinicio mediante la generación de una nueva solución aleatoria. Así, mientras la búsqueda tabú explota de manera intensiva las configuraciones prometedoras identificadas por el GA, los mecanismos de diversificación y reinicio preservan la capacidad de exploración del método híbrido. Los autores adaptan dos bancos de prueba clásicos de FJSSP (Brandimarte, 1993; Dauzère-pérès & Paulli, 1997) a un entorno con SDST y tiempos de retardo, obteniendo dos conjuntos de instancias: el primero con 10 casos (10 a 20 trabajos, $6 - 15$ máquinas) y el segundo con 18

casos (10 a 20 trabajos, $5 - 10$ máquinas). Comparan el HGA-TS frente a un GA puro, dos variantes de búsqueda tabú (TS1 y TS2) y un modelo exacto MILP resuelto con CPLEX, fijando criterios de parada distintos para cada método: el GA y las TS disponen de hasta 10 000 iteraciones (con reinicios periódicos en el caso de TS), el HGA-TS se detiene a las 100 iteraciones o tras 20 iteraciones consecutivas sin mejora, y el MILP tiene un límite de 4 horas por instancia. Los resultados muestran que en ningún caso el GA puro ni las TS superan al híbrido. Frente al MILP, el HGA-TS mejora los límites superiores conocidos en promedio en un 9,7 % en el primer conjunto y en un 15,8 %. Por otra parte, en términos computaciones el HGA-TS es más costoso que una iteración de un GA puro debido a la aplicación de la búsqueda tabú, el algoritmo alcanza soluciones de mayor calidad con un límite de apenas 100 iteraciones, frente a las 10 000 iteraciones asignadas a los componentes ejecutados por separado. Por tanto, los resultados no deben interpretarse únicamente como una reducción directa del tiempo de ejecución por iteración, sino como una mayor eficiencia en la relación entre esfuerzo de búsqueda y calidad de solución: la exploración poblacional del GA permite identificar regiones prometedoras, mientras que la TS explota de forma dirigida la estructura crítica de cada calendario. Este estudio evidencia así que la hibridación entre GA y TS puede proporcionar soluciones de mejor calidad que las obtenidas por ambos métodos de manera aislada.

En conjunto, los trabajos de Wang y Zhu (2021), Tutumlu y Saraç (2023) y Xie et al. (2023) evidencian que los algoritmos genéticos híbridos constituyen una alternativa efectiva para resolver variantes del FJSSP. En los tres estudios, el GA desempeña el papel de mecanismo de exploración global, generando y combinando soluciones que representan decisiones de asignación, secuenciación y, según el problema, distribución de trabajos entre fábricas o partición en sublotes. Mostrando los resultados que la búsqueda evolutiva por sí sola no resulta suficiente para explotar eficazmente las estructuras particulares de cada variante. Por lo tanto, la evidencia indica que la principal ventaja de los híbridos no reside necesariamente en reducir el costo de cada iteración, de hecho puede aumentar al incorporar procedimientos de búsqueda local, sino en mejorar la eficiencia de la búsqueda en términos de calidad de solución alcanzada por unidad de esfuerzo computacional, mediante el equilibrio entre diversificación global e intensificación sobre una búsqueda local relevante en cada iteración. No obstante, estos enfoques se fundamentan principalmente en la combinación del GA con procedimientos heurísticos o metaheurísticos de mejora local, lo cual constituye una oportunidad para explorar otros mecanismos de hibridización que incorporen información estructural adicional o procedimientos de optimización más especializados para el FJSSP-SDST.

3.3.2. Mateheuristica para el FJSSP-SDST

Tras realizar la revisión de literatura sobre el Flexible Job Shop y la aplicación de mateheurísticas, se identificaron seis artículos en las bases de datos Scopus y Web of Science que integran explícitamente ambos conceptos. En estos estudios, las mateheurísticas se presentan como una alternativa para abordar la complejidad del problema, al combinar la capacidad de exploración de los algortimos metaheurísticas con el refinamiento proporcionado por métodos exactos, como la programación lineal entera mixta o la programación por restricciones. En general, el componente metaheurístico explora el espacio global de soluciones mediante algoritmos genéticos, algoritmos meméticos o procedimientos de búsqueda de vecindad variable. Posteriormente, el modelo matemático se aplica sobre soluciones parciales o individuos

prometedores para mejorar decisiones específicas, entre ellas la asignación de máquinas, la secuenciación de operaciones y el dimensionamiento de sublotes. Este último corresponde a una característica estructural del problema, pues determina la fragmentación física de los lotes durante la ejecución de las operaciones y no debe confundirse con las estrategias de búsqueda empleadas por el algoritmo. De esta manera, la integración de ambos enfoques evita resolver el modelo exacto sobre la totalidad del problema, reduce el esfuerzo computacional y conserva la capacidad de mejorar localmente la calidad de las soluciones.

Cheng et al. (2026) abordan el *Flexible Job Shop Scheduling Problem with Discrete Sequence Flexibility* (FJSPDS) (Cheng et al., 2026). Para esta variante se propone el algoritmo MILEA (*Matheuristic and Imitation Learning-driven Evolutionary Algorithm*), el cual combina un algoritmo evolutivo con modelos de programación por restricciones (*Constraint Programming*, CP). El algoritmo evolutivo genera y modifica soluciones mediante operadores de cruce y mutación, mientras que el aprendizaje por imitación selecciona de forma adaptativa los operadores de búsqueda local. El componente CP se utiliza tanto durante la inicialización como en el refinamiento de individuos destacados, permitiendo optimizar de manera más precisa la secuencia de las operaciones y la selección de máquinas. La propuesta fue evaluada en 110 instancias de referencia. La mateheurística MILEA obtuvo 52 nuevas mejores soluciones conocidas frente a métodos MILP aislados y otros algortimos metaheurísticos híbridos. Estos resultados sugieren que el CP resulta particularmente adecuada para instancias de pequeña y mediana escala porque puede encontrar la solución perfecta rápidamente utilizando la lógica de factibilidad. Para problemas de gran escala, su potencia se utiliza como un "motor de refinamiento"dentro de algoritmos híbridos como MILEA para asegurar que las soluciones no solo sean rápidas, sino matemáticamente precisas.

El siguiente artículo aborda al *Flexible Job Shop Scheduling Problem with Lot-streaming and Machine Reconfigurations* (FJSP-LSMR). Fan et al. (2023) en esta variante consideran lotes que pueden dividirse en sublotes, permitiendo el solapamiento entre operaciones consecutivas, así como máquinas reconfigurables que requieren configuraciones o módulos auxiliares para ejecutar determinadas operaciones. Para este problema se desarrolló la mateheurística MH-VNS, que combina un algoritmo genético con una estrategia de búsqueda de vecindad variable (VNS). La metaheurística se encarga de explorar decisiones de asignación, secuenciación y configuración, mientras que los modelos MILP denominados LSO1 y LSO2 se emplean como mecanismos de búsqueda local para optimizar los tamaños y, en algunos casos, el número de sublotes. La propuesta se comparó con un algoritmo genético convencional, una versión sin optimización matemática de lotes y otras configuraciones parciales del enfoque híbrido. En 80 instancias extendidas del conjunto Fdata, la MH-VNS obtuvo la mejor solución en 69 casos. Adicionalmente, en un caso industrial compuesto por 20 trabajos y 47 máquinas, la propuesta redujo la tardanza ponderada total desde 1220 hasta 709. Estos resultados muestran que la optimización explícita del dimensionamiento de lotes complementa de manera efectiva la exploración realizada por el algoritmo genético.

Zhao et al. (2026) abordaron el problema *Lot-streaming Flexible Job Shop Scheduling Problem with Variable Sublots and Intermingling* (LSFJSP-VI) introduce una mayor complejidad al permitir que los tamaños de los sublotes varíen y que sublotes pertenecientes a operaciones diferentes se entrelacen en una misma máquina. Para esta variante se propuso MH-MA, una mateheurística basada en un algoritmo memético. Su componente metaheurístico utiliza una

codificación de longitud variable y operadores especializados de cruce y mutación, adecuados para representar procesos de división y fusión de sublotes. El componente matemático incorpora cuatro estrategias MILP, denominadas LSO-S, LSO-LS-1, LSO-LS-2 y LSO-SLS, aplicadas sobre segmentos seleccionados de los cromosomas. Esta estrategia evita resolver el modelo completo en cada iteración y permite concentrar el esfuerzo de optimización en decisiones críticas de secuenciación y dimensionamiento. En 30 instancias derivadas de los conjuntos Fdata y Brandimarte, MH-MA superó a otros algoritmos metaheurísticos híbridos con un mayor *makespan*. En aplicaciones industriales, las mejoras en el *makespan* se ubicaron entre 2,66 % y 11,22 %. En consecuencia, la combinación de un algoritmo memético con subproblemas MILP resulta apropiada en entornos donde la flexibilidad de los sublotes amplía considerablemente el espacio de búsqueda.

Fan, Zhang, Yang et al. (2024), abordó una mateheurística aplicadas a contextos dinámicos mediante el *Flexible Job Shop Rescheduling Problem with Lot-streaming and Machine Reconfigurations* (FJRP-LSMR). Esta variante considera eventos tales como fallas de máquina o la llegada de trabajos urgentes, los cuales obligan a modificar un cronograma previamente establecido. La propuesta MHRLS (*Matheuristic with Re-Lot-Sizing Strategies*) combina un algoritmo genético y una búsqueda de vecindad variable con un modelo MILP denominado LSORLS. La componente metaheurística reorganiza las decisiones de programación luego de la perturbación, mientras que el modelo MILP ajusta los tamaños de los sublotes que aún no han sido procesados. El uso de estrategias de *re-lot-sizing* completo o parcial permite ampliar las alternativas de recuperación sin alterar las operaciones ya ejecutadas. La propuesta se comparó con algoritmos genéticos tradicionales y con variantes que no incorporaban el reajuste de lotes. En 80 instancias con escenarios de fallas e inserción de trabajos urgentes, MHRLS alcanzó la mejor solución en 68 casos. En un escenario industrial con 150 sublotes pendientes, redujo la tardanza ponderada total de 3735, obtenida mediante un algoritmo genético convencional, a 1392. Además, el componente MILP requirió entre 0,05 y 0,15 segundos en promedio, lo cual evidencia su potencial para apoyar decisiones de reprogramación en horizontes de tiempo reducidos.

Cheng et al. (2025) abordaron el *Flexible Job Shop Scheduling Problem with Sequence-Dependent Setup Times* (FJSP-SDST), la complejidad se incrementa porque los tiempos de preparación dependen de la operación procesada inmediatamente antes en cada máquina. Para esta variante se desarrolló MFLA-MH (*Meta-Feedback Learning-Assisted Matheuristic*), una propuesta que combina una búsqueda de vecindad variable colaborativa con mecanismos de aprendizaje de meta-retroalimentación. La parte metaheurística identifica y prioriza operadores de búsqueda según su desempeño previo, mientras que la parte matemática se fundamenta en un conjunto de modelos colaborativos de programación por restricciones, incluyendo un modelo maestro y modelos especializados para la secuenciación de operaciones y la selección de máquinas. La propuesta fue evaluada frente a otros algortimos metaheurísticos en 20 instancias reales de un taller de producción de componentes estructurales para maquinaria de carbón. MFLA-MH obtuvo el mejor *makespan*, tanto en el mejor resultado como en el promedio, para las 20 instancias analizadas. Estos resultados indican que el aprendizaje adaptativo puede mejorar la selección de mecanismos de búsqueda en problemas donde las decisiones de secuenciación tienen efectos directos sobre los tiempos de preparación.

Fan, Zhang, Tian et al. (2024) finalmente, abordaron el *Flexible Job Shop Scheduling Pro-*

blem with Variable Lot Sizes (FJSP-VLS) considera que las operaciones de un mismo trabajo pueden utilizar planes de división de lotes distintos. Para este problema se propone MHER (*Matheuristic based on Early Release policy*), que integra un algoritmo genético, búsqueda de vecindad variable y un modelo MILP simplificado denominado LSOER. El aporte central de esta propuesta es la política de liberación temprana (*Early Release*, ER), la cual determina el instante en que un sublote puede iniciar la siguiente operación sin esperar la finalización completa de todos los sublotes precedentes. La metaheurística explora configuraciones de secuenciación, asignación y división de lotes, mientras que el MILP mejora periódicamente el tamaño de los sublotes en las soluciones más prometedoras. La propuesta fue comparada con versiones del algoritmo que no incluían la política ER ni la búsqueda local MILP. Sobre 40 instancias extendidas de Fdata, distribuidas en los grupos A–D, MHER obtuvo 38 mejores soluciones y 39 mejores resultados promedio. Además, el componente matemático mejoró las soluciones iniciales de la metaheurística entre 20 %y30 % en promedio, con tiempos de resolución generalmente inferiores a 0,30 segundos. En un ejemplo reportado, la política de liberación temprana redujo el *makespan* de 86 a 71 unidades de tiempo, al eliminar esperas que no eran necesarias entre operaciones consecutivas.

En conjunto, la evidencia revisada indica que las mateheurísticas son especialmente útiles cuando el FJSP incorpora características que incrementan su complejidad, tales como la transmisión de lotes, el uso de sublotes de tamaño variable, la reconfiguración de máquinas, los SDST, la flexibilidad en las relaciones de precedencia y la ocurrencia de eventos dinámicos. En estos contextos, las metaheurísticas aportan diversidad y capacidad de exploración global, mientras que los MILP y CP permiten refinar decisiones locales de alto impacto. La principal contribución de esta integración no se limita a la mejora del valor de la función objetivo, sino que radica en la posibilidad de identificar subconjuntos de decisiones que pueden ser tratados mediante procedimientos exactos sin resolver el problema completo de forma integrada. Por ejemplo, los modelos MILP se han empleado principalmente para ajustar los tamaños de lote o sublotes, mientras que los modelos CP han mostrado ventajas en el tratamiento de restricciones complejas de secuenciación, precedencia y asignación. Así, la descomposición entre decisiones globales y locales permite obtener soluciones de alta calidad sin asumir el costo computacional asociado a la resolución exacta del problema completo.

No obstante, la efectividad de una mateheurística depende de decisiones de diseño tales como la representación de las soluciones, la selección de los individuos que serán refinados, la frecuencia con la que se activa el componente matemático y el límite de tiempo asignado al solucionador. A partir de la revisión realizada, no se identificaron mateheurísticas orientadas específicamente al FJSP-SDST que integren un GA con un modelo MILP aplicado sobre lotes de operaciones dentro de la estrategia algoritmica, distinto a ser una caraterítica del problema. En consecuencia, este trabajo propone una mateheurística que combina explícitamente ambos enfoques. La propuesta busca aprovechar la capacidad del GA para explorar de manera eficiente espacios de solución de gran escala y, a su vez, compensar sus limitaciones en la intensificación de la búsqueda local mediante un modelo MILP que optimiza subproblemas de asignación de máquinas y secuenciación sobre lotes de operaciones.

La integración se implementa como un operador de diversidad orientada a mejores resultados. Preservando la diversidad de la población que aporta el GA y e implementando el mecanismo mediante un decodificador por lotes basado en MILP. Este último recibe como

entrada la secuencia de operaciones generada por el GA y la divide en lotes de tamaño reducido. Para cada lote, el modelo MILP determina la asignación de máquinas que optimiza localmente el Scheduling y, cuando resulta conveniente, ajusta el orden de las operaciones pertenecientes al mismo lote. De esta manera, el GA conserva su función como mecanismo de exploración global sobre la población, mientras que el MILP actúa como un operador de mejora local intensiva integrado al proceso de decodificación. Los demás componentes de la mateheurística, incluidos los operadores de selección, cruce y mutación, se fundamentan en estrategias empleadas por algoritmos genéticos con resultados competitivos en la literatura para problemas de programación flexible de talleres (Amjad et al., 2018; Dauzère-Pérès et al., 2024; Gao et al., 2019).

4. Metodología

La metodología propuesta para abordar el FJSSP-SDST se basa en el diseño de una Mateheurística, usando un Algoritmo Genético (GA) como elemento central de la búsqueda, el pseudocódigo se presenta en el Algoritmo 1. El GA incorpora los operadores convencionales de generación de población inicial, selección, cruce, mutación y reemplazo para explorar el espacio de búsqueda. Tras generar la población inicial, se calcula el fitness de cada individuo mediante el decodificador genético, paso que se repite para los individuos hijos en cada generación. En este algoritmo, un individuo representa una solución factible al problema. La Mateheurística recibe como datos de entrada los casos de prueba descritos en la sección 5.1, y su salida corresponde al mejor individuo encontrado al finalizar el proceso evolutivo.

A pesar de los buenos resultados reconocidos por la literatura en este tipo de algoritmos evolutivos para resolver el FJSSP, el rendimiento de los GA han presentado limitaciones significativas, como la baja precisión y la convergencia prematura. Con el fin de mitigar esta problemática, en este trabajo se incorpora un modelo MILP como operador adicional a los genéticos, su función principal es diversificar las poblaciones generadas por el GA, contribuyendo a mejorar la exploración del espacio de soluciones. Esta incorporación se realiza cada cierto número de generaciones, controlada por el parámetro *NumGenMILP*, y se aplica sobre un porcentaje de la población determinado por *PerMILP*.

Por otra parte, los operadores convencionales del GA se evalúan de forma independiente utilizando diversos métodos, con el fin de identificar cuáles ofrecen mejor desempeño. Para ello, se ejecuta exclusivamente el GA sin incluir el operador MILP, manteniendo constantes los métodos de todos los operadores excepto el que se está evaluando, lo que permite aislar su influencia en el rendimiento del algoritmo. Se seleccionaron tres casos de prueba de tamaño mediano (HurinkEdata26, HurinkRdata24 y HurinkVdata28), cada uno con 15 trabajos, 10 máquinas y 150 operaciones, con niveles de flexibilidad baja, media y alta (1,1, 2,0 y 4,8, respectivamente). Las pruebas se realizan con un máximo de 500 generaciones para evaluar el nivel de convergencia de cada método en los distintos contextos de flexibilidad. En la sección 5) se presentan todos los resultados.

Para describir la propuesta, esta sección se divide en tres partes: la subsección 4.1 describe en detalle la codificación y decodificación usada en el GA, la subsección 4.2 describe los operadores genéticos implementados, y la subsección 4.3 presenta el operador exacto basado en un MILP, diseñado como componente de diversificación para el GA.

Algorithm 1 Pseudocódigo de la Matheurística

```
1: Input: Instancia de prueba SDST-FJSSP
2: vIndividuos = InicializarPoblación( popSize)
3: DecodificadorGenéticoFitness(vIndividuos)
4: for x in NumGeneraciones do
5:     vParejasPadres = Selección( NumParejas, vIndividuos)
6:     vIndividuosHijos = Cruce(vParejasPadres, vIndividuos)
7:     Mutación (vIndividuosHijos ,mutProb)
8:     if x Mod NumGenMILP == 0 then
9:         OperadorMILP(vIndividuos, PerMILP)
10:    end if
11:    DecodificadorGenéticoFitness(vIndividuosHijos)
12:    vIndividuos = Reemplazo(vIndividuos, vIndividuosHijos, repPerc)
13: end for
14: Return: Mejor Individuo
```

4.1. Codificación y decodificación de las soluciones

La codificación de las soluciones es la representación de una solución en el algoritmo, esta interactúa con todos los operadores del algoritmo e influye directamente en su desempeño (Gao et al., 2006), en este caso particular, se requiere una representación que sea fácilmente interpretable tanto para el GA como para el MILP. Por ello, se adopta el método de codificación basado en dos vectores propuesto por Gao et al. (2008): un vector para la secuenciación de operaciones y otro para la asignación de máquinas.

En la Tabla 3 se presenta un ejemplo ilustrativo de ambos vectores, utilizando una solución del problema de la Fig. 2. El primer vector indica la secuencia de operaciones, donde la h-ésima aparición de un trabajo j corresponde a su operación h. Así, el vector se descompone como $(R_{11}, R_{31}, R_{32}, R_{21}, R_{22}, R_{12})$, donde R_{jh} representa la operación h del job j. El segundo vector indica la posición de la máquina dentro del conjunto de modos posibles de cada operación; es decir, los valores mostrados no representan las máquinas directamente, sino el modo seleccionado. Por ejemplo, la operación R_{12} aparece en la posición 6 del vector de secuencia, y su valor correspondiente en el vector de modos es 1, lo que indica que se selecciona el primer modo disponible, que corresponde a la máquina 2. Además de ofrecer una representación clara de la solución, esta codificación facilita la integración del MILP dentro del GA, como de detalla la subsección 4.3.

Tabla 3: Ejemplo de la codificación basada en dos vectores usada en el GA.

Vector de secuencia	1	3	3	2	2	1
Vector de modos	1	2	2	1	1	1

Estas soluciones representan los cromosomas dentro del algoritmo genético y requieren ser decodificadas para obtener su función de aptitud, esta determina la calidad de cada solución

y es una medida de comparación. La decodificación puede realizarse de diversas maneras, ya que pueden surgir períodos de inactividad entre las operaciones. Pero es esencial que la decodificación genere soluciones dentro del espacio de Programación Activa, puesto que garantiza contener al menos una de las soluciones óptimas (Pinedo, 2022).

La decodificación se realiza mediante el enfoque basado en prioridades propuesto por Gao et al. (2008), que traduce los cromosomas en programas activos. En este método, las operaciones R_{jh} se programan siguiendo el orden establecido por el vector de secuencia, con la máquina indicada por el vector de modos. Para cada operación, se analizan de izquierda a derecha los intervalos de inactividad en la máquina correspondiente: si existe un intervalo disponible que cumple con las restricciones de precedencia y de los tiempos de configuración, la operación se programa en dicho intervalo; de lo contrario, se asigna al final de la máquina considerada. Este proceso garantiza la generación de programas activos y contribuye a la obtención de soluciones factibles.

4.2. Algoritmo Genético

Un algoritmo genético se clasifica dentro de la categoría de algoritmos evolutivos y se inspira en el proceso de evolución genética natural, fundamentado en la regla de supervivencia del más apto, propuesto por Holland (1975). La efectividad del GA reside en su capacidad para explorar eficientemente espacios de búsqueda amplios y complejos, lo que le permite encontrar buenas soluciones para problemas de alta dimensionalidad como el FJSSP (Amjad et al., 2018). Su proceso se organiza en torno a operadores fundamentales: la creación de la población inicial, la selección, el cruce, la mutación y el reeplazo.

4.2.1. Población Inicial

Se genera una población inicial compuesta por varios individuos, cada uno representado mediante dos vectores codificados: el vector de secuencia y el vector de modos. El vector de secuencia establece el orden en que se consideran las operaciones, mientras que el vector de modos define la alternativa de procesamiento seleccionada para cada operación. Ambos vectores se construyen de manera aleatoria a partir de una distribución uniforme, respetando las configuraciones factibles del problema. Para generar el vector de secuencia, en cada posición se selecciona una operación dentro del conjunto de operaciones disponibles. Una vez una operación ha sido asignada, se actualiza dicho conjunto para evitar que se exceda el número de operaciones correspondiente a cada trabajo. Por ejemplo, si el trabajo 1 cuenta con dos operaciones y ambas ya fueron incluidas en el vector de secuencia, dicho trabajo se elimina del conjunto de candidatos disponibles para las posiciones restantes. Por su parte, el vector de modos se genera seleccionando aleatoriamente, De igual forma mediante una distribución uniforme, uno de los modos factibles para cada operación. Por ejemplo, si la primera operación del trabajo 1, puede procesarse en cuatro máquinas alternativas, el modo asignado se selecciona entre los valores de 1 a 4.

Este procedimiento permite obtener una población inicial diversa, imparcial y factible. El tamaño de la población se define como un parámetro de entrada del algoritmo y determina la cantidad de individuos generados. Este valor se mantiene constante durante todo el proceso evolutivo.

4.2.2. Operador de Selección

El operador de selección produce como salida un vector de parejas, donde cada pareja representa los dos padres necesarios para generar un nuevo individuo. Se implementaron el método del torneo, y la combinación del método de torneo con ruleta, dos de los métodos más ampliamente utilizados en la literatura del FJSSP Amjad et al. (2018). Los algoritmos 2 y 3 muestran el pseudocódigo correspondiente. El número de parejas es equivalente a la mitad del tamaño de la población, dado que cada pareja genera dos individuos, lo que produce una población de descendientes del mismo tamaño que la población inicial.

Para el método de torneo se selecciona aleatoriamente un subgrupo de individuos según el tamaño del torneo, y de este se elige aquel con el menor makespan (más apto), obteniendo así el primer padre. Para el segundo padre, se repite el mismo proceso con la restricción de que el primer padre seleccionado no forme parte del nuevo subgrupo, lo que garantiza que una pareja no contenga al mismo individuo y proporciona diversidad en la selección. El método de torneo y ruleta, primero, se determina aleatoriamente el tamaño del torneo, que puede abarcar hasta la totalidad de la población. Con dicho tamaño se conforma un subgrupo de individuos extraídos de la población actual. Sobre este subgrupo se aplica el método de la ruleta. Donde se asigna a cada individuo z una probabilidad proporcional a su aptitud, calculada como el inverso multiplicativo del makespan: $Pr[z] = \frac{1}{vTournament[z].Makespan}$, de modo que las soluciones con menor makespan obtienen mayor probabilidad de selección. A continuación, estas probabilidades se normalizan y se acumulan, generando para cada individuo un rango dentro del intervalo $[0, 1]$. Este proceso se ejecuta dos veces para obtener ambos padres; tras seleccionar el primer padre, este se elimina del subgrupo y se re-escalan las probabilidades acumuladas antes de seleccionar el segundo. Cabe destacar que el tamaño del torneo varía aleatoriamente en cada iteración, lo que contribuye a la diversidad en la formación de parejas.

Algorithm 2 Pseudocódigo del Método de Selección 1: Torneo

```
1:  Input: vIndividuos
2:  for Np in NumParejas do
3:      for t=1 to 2 do
4:          vTorneo[t] = SelecciónAleatoria(vIndividuos, TamTorneo)
5:          OrdenarPorMakespanAscendente(vTorneo[t])
6:          Padre[t] = vTorneo[t][0]
7:      end for
8:      vParejasPadres = GuardarPareja(Padre[1], Padre[2])
9:  end for
10: Return: vParejasPadres
```

Las pruebas para el método 1 se realizaron con un tamaño de torneo fijo de 4, mientras que en el método 2 el tamaño del torneo se genera aleatoriamente en cada iteración, por lo que no requiere ser especificado como parámetro. Los resultados obtenidos, mostrados en las Figuras 3a, 3b y 3c, indican que el método de selección 2 tiene mejores resultados, además de presentar una convergencia más gradual. El método 1 tiende a converger prematuramente alrededor de las primeras 50 generaciones y posteriormente se estanca sin lograr mejoras significativas. Estos resultados sugieren que el método de selección 2 es más robusto ante

Algorithm 3 Pseudocódigo del Método de Selección 2: Torneo y Ruleta

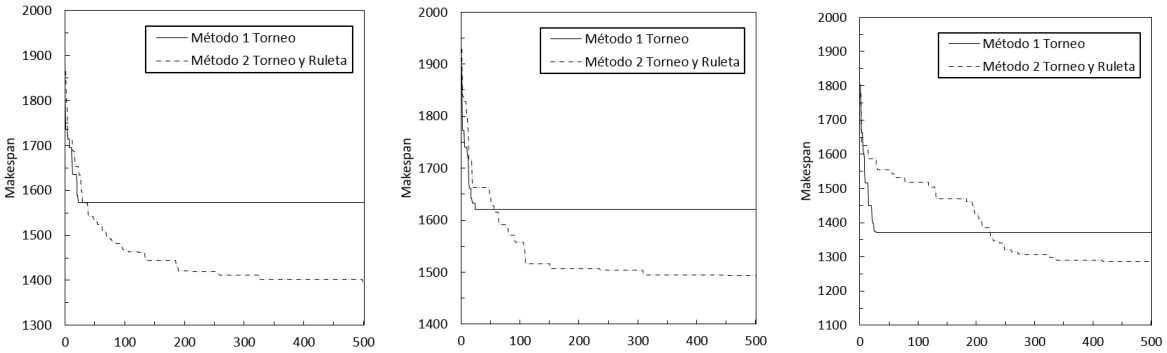
Input: vIndividuos
for Np in NumParejas **do**
    TamTorneo = **EnteroAleatorio**(2, vIndividuos.Size - 1)
    vTorneo = **SelecciónAleatoria**(vIndividuos, TamTorneo)
    for z=1 to TamTorneo **do**
        $Pr[z] = \frac{1}{vTorneo[z].Makespan}$
    end for
    for z=1 to TamTorneo **do**
        $PrAcum[z] = PrAcum[z-1] + \frac{Pr[z]}{\sum Pr[z]}$
    end for
    for t=1 to 2 **do**
        $R[t]$ = RandomReal(0,1)
        for z in TamTorneo **do**
            if $PrAcum[z-1] < R[t] \leq PrAcum[z]$ **then**
                Padre[t] = vTorneo[z]
                Break
            end if
        end for
    end for
    vParejasPadres = **GuardarPareja**(Padre[1], Padre[2])
end for
Return: vParejasPadres

diferentes niveles de flexibilidad, contribuyendo a un mejor rendimiento global del algoritmo genético.

4.2.3. Operador de Cruce

Este operador es el encargado de generar la población de hijos a partir de las parejas formadas en el operador de selección. Recibe como entrada las parejas seleccionadas y produce como salida una población de hijos con el mismo tamaño que la población inicial, dado que cada pareja genera dos hijos.

Al igual que en el operador de selección, no se ha establecido un método de cruce que sea consistentemente mejor que otro en todos los casos, por esto se implementaron los dos métodos más utilizados con el fin de identificar el más conveniente para el problema abordado. El primer método, denominado Preservación de Precedencia (PPX), respeta el orden absoluto de los genes en los cromosomas parentales (Ripon et al., 2011). El procedimiento recorre las pajeras generando dos hijos. Mediante un aleatorio binario se determina la procedencia de cada gen: tomando un valor de 0 cuando el gen se herede del Padre 1; y un valor de 1 cuando se herede Padre 2 (Algoritmo 4). Pero se debe hacer un proceso de bloqueo genética para evitar genes duplicados, que pueden ocasionar operaciones no existentes. Las figuras 4 y 5 muestra un ejemplo de este proceso. Las celdas verdes representan las operaciones disponibles para ser asignadas al hijo en cada iteración; las celdas amarillas identifican los genes del Padre 1, las celdas azules los genes del Padre 2 y las celdas grises las operaciones descartadas debido



(a) Flexibilidad Baja HurinkEdata26 **(b)** Flexibilidad Mediana HurinkRdata24 **(c)** Flexibilidad Alta HurinkVdata28

Figura 3: Convergencia métodos de Selección para diferentes niveles de flexibilidad.

al bloqueo genético, ya que habían sido previamente heredadas del otro padre. En la primera parte de la figura, las dos primeras posiciones de *vCrossover* tienen valor 0, por lo que se asignan al hijo los genes R_{11} y R_{12} del Padre 1. Al transferir estos genes, se bloquean en el Padre 2 para evitar duplicaciones. En la segunda parte, *vCrossover* indica tres genes del Padre 2, por lo que se seleccionan las tres primeras operaciones disponibles (no bloqueadas) de su vector de secuencia: R_{41}, R_{31} y R_{21}. Nótese que R_{12} no se selecciona a pesar de su posición en el vector, ya que se encuentra bloqueada por haber sido asignada previamente. Este proceso se repite iterativamente hasta completar la formación del nuevo hijo, como se muestra en la parte 3 de la figura, donde se evidencia la combinación de segmentos de secuencia de ambos padres. El vector de modos del hijo se asigna conforme al modo que posee cada operación en el padre correspondiente.

Algorithm 4 Pseudocódigo del Método de Cruce 1: PPX

```
1:  Input: vParejasPadres
2:  for p in vParejas do
3:      for a in NumActividades do
4:          vCruce[a] = EnteroAleatorio(0,1)
5:      end for
6:      for h in 2 do
7:          for a in NumActividades do
8:              if vCruce[a] == 0 then
9:                  AsignarGen(Padre1, Hijo[h], a)
10:             else
11:                 AsignaciónGen(Padre2, Hijo[h], a)
12:             end if
13:         end for
14:         vIndividuosHijos = GuardarHijo(Hijo[h])
15:     end for
16: end for
17: Return: vIndividuosHijos
```

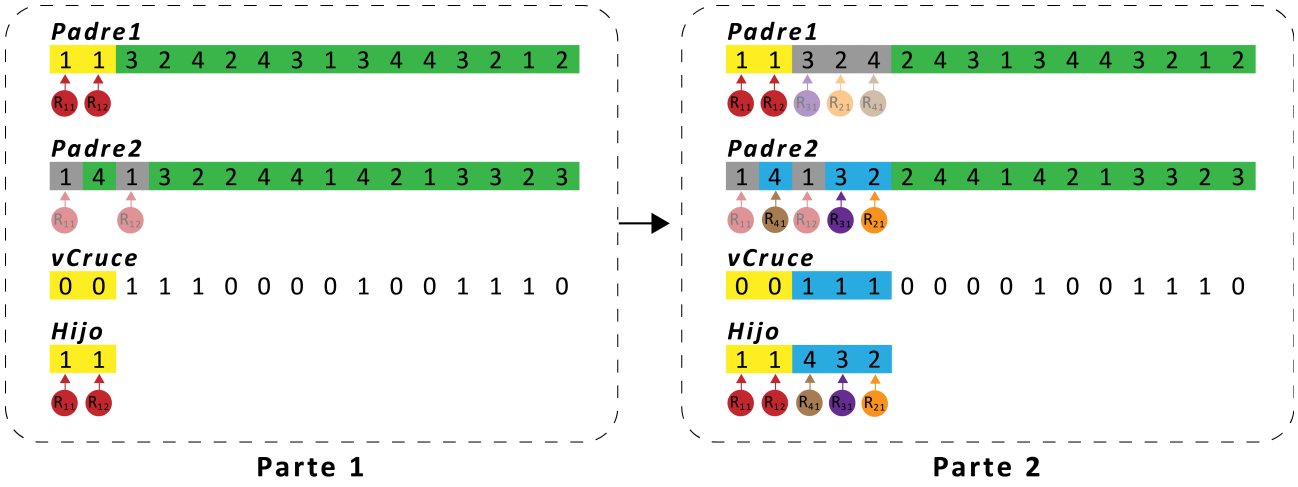


Figura 4: Ejemplo Operador PPX Parte 1 y 2

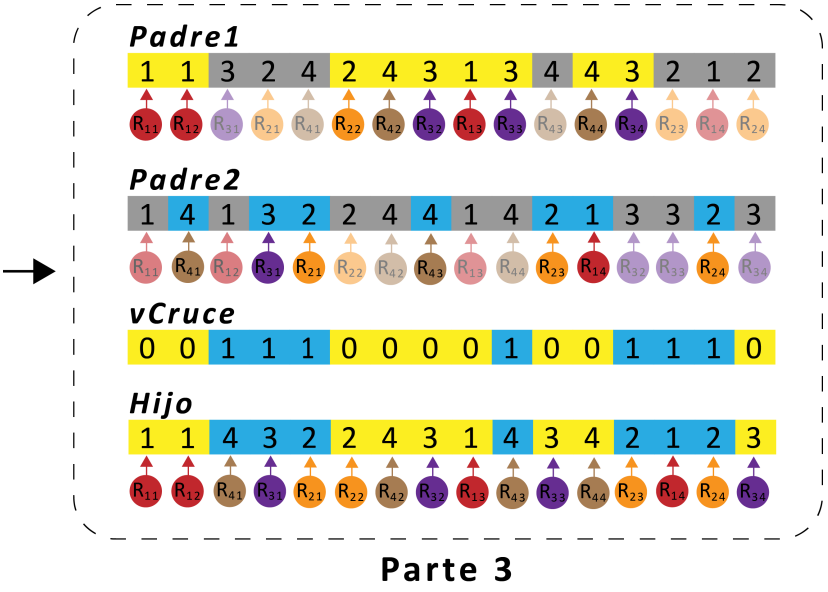


Figura 5: Ejemplo Operador PPX Parte 3

El segundo método emplea el cruce en un solo punto, detallado en el Algoritmo 5. En este caso se selecciona aleatoriamente un punto de corte dentro del vector de secuencia, restringido al intervalo entre el 25 % y el 75 % del total de operaciones, con el propósito de garantizar una transmisión significativa de genes de ambos padres y evitar que el hijo resulte idéntico a uno de ellos. A partir de este punto, los genes anteriores se heredan del Padre 1 y los restantes del Padre 2 siguiendo el proceso de bloqueo genético también. En la Fig. 6 se ilustra un ejemplo: el rango de selección abarca desde la operación 4 hasta la 12, y el punto aleatorio seleccionado es la posición 8, por lo que las primeras 8 posiciones del hijo provienen del Padre 1 y las restantes provienen de las posiciones no bloqueadas del Padre 2.

Finalmente, en las Figuras 7a,7b y 7c se comparan ambos métodos evaluando su conver-

Algorithm 5 Pseudocódigo del Método de Cruce 2

```
Input: vParejasPadres
for p in vParejasPadres do
    for h in 2 do
        R = EnteroAleatorio(25 % * NumActividades, 75 % * NumActividades)
        Hijo[h] = AsignarGenesPadre1(1, R)
        Hijo[h] = AsignarGenesPadre2(R + 1, NumActividades)
        vIndividuosHijos = GuardarHijo(Hijo[h])
    end for
end for
Return: vIndividuosHijos
```

gencia en los diferentes niveles de flexibilidad. Los resultados muestran que, si bien ambos métodos presentan una convergencia gradual, el método 1 logra alcanzar mejores resultados en los tres casos evaluados. Por lo tanto, se determina que el método 1 (PPX) será el aplicado en el algoritmo genético.

4.2.4. Operador de mutación

El operador de mutación diversifica la búsqueda de soluciones mediante la modificación de los genes de un individuo dada una probabilidad predefinida (*mutProb*). Entre los métodos más empleados para el problema abordado se encuentran el de intercambio y el aleatorio, como se menciona en Amjad et al. (2018). En este estudio se evalúan los dos métodos, los cuales se aplican cuando un individuo es seleccionado para mutación según una probabilidad definida.

El primer método, la mutación por intercambio, se describe en el Algoritmo 6. Se seleccionan aleatoriamente dos trabajos distintos J_1 y J_2, y para cada uno se escoge aleatoriamente una de sus operaciones (R_{11} y R_{22}) en el vector de secuencia del individuo y se intercambian. La selección se realiza por trabajos distintos y no por posiciones aleatorias del vector, ya que intercambiar dos operaciones del mismo trabajo no produciría ningún cambio en el individuo (considerando la decodificación). Tras el intercambio, no es posible violar las restricciones de precedencia entre las operaciones de un mismo trabajo, ya que el operador reorganiza automáticamente las operaciones de cada *job* involucrado. En particular, las operaciones se reordenan de acuerdo con su secuencia, preservando el orden relativo de las operaciones de cada trabajo (operación 1, operación 2, y así sucesivamente). Adicionalmente, el vector de modos del individuo mutado se regenera aleatoriamente. En la Fig. 8 se ilustra este proceso: las operaciones seleccionadas para el intercambio son R_{21} y R_{14}.Como consecuencia del intercambio, es necesario reorganizar las operaciones de los trabajos involucrados para no violar la precedencia, lo que afectó adicionalmente las posiciones de las operaciones R_{13}, R_{22} y R_{23}.

El segundo método consiste en desplazar las operaciones de un trabajo específico dentro del vector de secuencia, como se describe en el Algoritmo 7. Se selecciona aleatoriamente el trabajo que será desplazado, luego se eliminan temporalmente sus operaciones del vector de secuencia para facilitar la reinserción posterior. Luego, se determinan la dirección y la magnitud del desplazamiento: la variable L indica la dirección (0 hacia la derecha, 1 hacia la

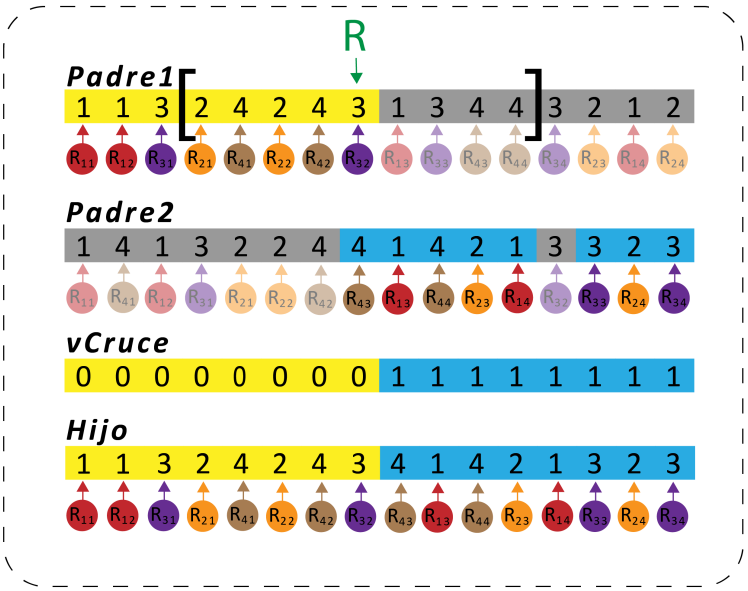


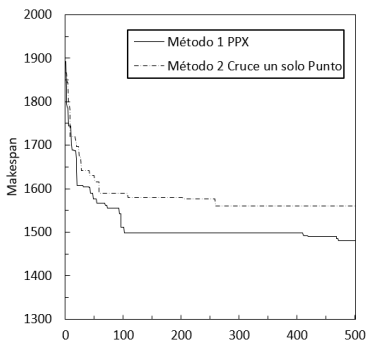
Figura 6: Ejemplo Operador Cruce de un solo punto

izquierda) y la variable C establece el número de posiciones del movimiento. Esta magnitud está acotada por las posiciones de la primera y última operación del trabajo seleccionado, de modo que el desplazamiento no viole la precedencia de sus operaciones. En la Fig. 9 se muestra un ejemplo de este operador: se selecciona el trabajo 2, la cantidad máxima de posiciones hacia la izquierda es 4 y hacia la derecha es 1. Superar estos límites provocaría que las operaciones del trabajo se reordenen internamente, alterando su precedencia. Finalmente, se realiza el desplazamiento insertando las operaciones del trabajo en sus nuevas posiciones. En este ejemplo desplazamiento del trabajo 2, es hacia la izquierda moviéndose 2 posiciones, este movimiento afecta también las posiciones de otras operaciones (R_{12}, R_{31}, R_{13}, R_{44}, R_{32} y R_{33}); sin embargo, el uso de la función de inserción simplifica el proceso al manejar estos reajustes de forma implícita, garantizando la coherencia del individuo mutado.

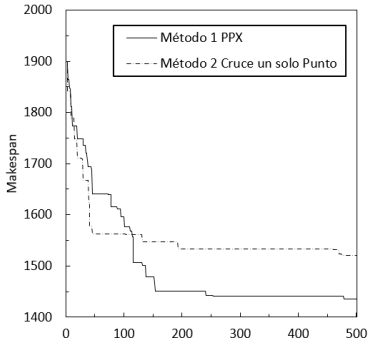
Al comparar ambos métodos, se observa en la Fig. 10 que el nivel de convergencia es muy similar en los tres casos de flexibilidad, incluso al aumentar el número de generaciones de 500 a 1 000 para ampliar el rango de comparación. En el caso de flexibilidad baja, los resultados son prácticamente idénticos; en flexibilidad media, el método 1 es ligeramente superior, y en flexibilidad alta, ocurre lo contrario. Por lo tanto, se determina que la elección entre uno u otro método no produce una diferencia significativa en la eficacia del algoritmo genético. No obstante, dado que el método 1 es más simple tanto algorítmica como computacionalmente, se selecciona el método de intercambio como operador de mutación.

4.2.5. Operador de Reemplazo

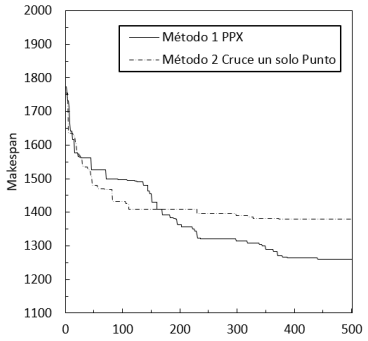
Cada generación de padres produce una nueva generación de hijos, lo que duplica temporalmente el tamaño de la población. Para mantener constante el tamaño de la misma en cada



(a) Flexibilidad Baja HurinkEdata26



(b) Flexibilidad Mediana HurinkRdata24



(c) Flexibilidad Alta HurinkVdata28

Figura 7: Convergencia métodos de Selección para diferentes niveles de flexibilidad.

Algorithm 6 Pseudocódigo del Método de Mutación 1: Intercambio

Input: vIndividuo a mutar
Secuencia = vIndividuo.Secuencia
J1, J2 = **MuestraAleatoria**(1, NumJobs)
Op1 = **EnteroAleatorio**(1, Job[J1].NumOperaciones)
Op2 = **EnteroAleatorio**(1, Job[J2].NumOperaciones)
Pos1 = **CalcularPosición**(J1, Op1)
Pos2 = **CalcularPosición**(J2, Op2)
vIndividuo.Secuencia[Pos1] = Secuencia[Pos2]
vIndividuo.Secuencia[Pos2] = Secuencia[Pos1]
ReordenarOperacionesTrabajo(vIndividuo)
ReasignarModosAleatoriamente(vIndividuo)
Return: vIndividuo mutado

generación y evitar un crecimiento acumulativo, se implementa un operador de reemplazo que selecciona los individuos que pasarán a la siguiente generación. En este trabajo se implementó un operador elitista. Se toma un porcentaje (*repPerc*) de los mejores individuos (padres más hijos) y se seleccionan directamente para la siguiente generación. Mientras que los restantes se seleccionan aleatoriamente de ambas poblaciones. De esta manera, se mantiene diversidad en las generaciones y, al mismo tiempo, se garantiza que los individuos más aptos propaguen sus genes, contribuyendo a la mejora progresiva de la población.

4.3. Operador MILP

Este operador corresponde a una de las principales contribuciones de la investigación. El operador tiene como objetivo de introducir variabilidad en la población y mejorar la aptitud de los individuos. La idea es integrar la búsqueda poblacional del GA pero mitigando la convergencia en óptimos locales e insertando diversidad mediante el uso de un modelo matemático exacto (detallado en la subsección 2.1). La inclusión de este modelo no puede ser directo debido a la naturaleza NP-hard del problema, por tanto se adaptó para que pueda ejecutarse en tiempos razonables, de forma que guíe la búsqueda e intervenga en la secuen-

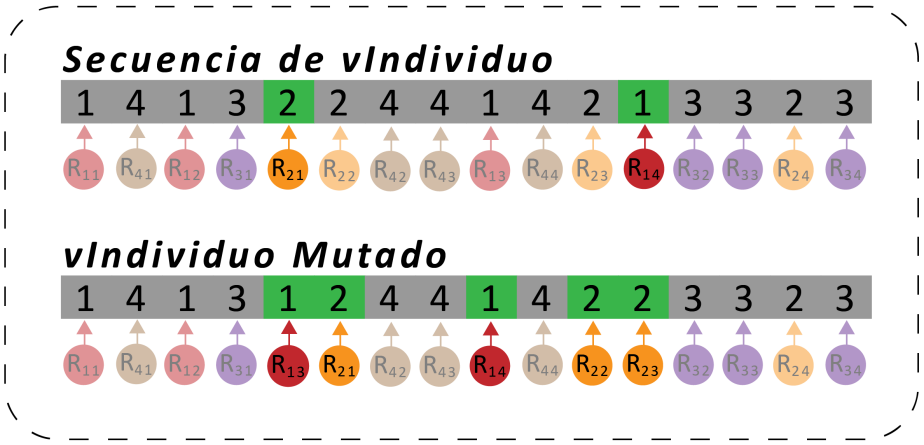


Figura 8: Ejemplo Operador Mutación de intercambio

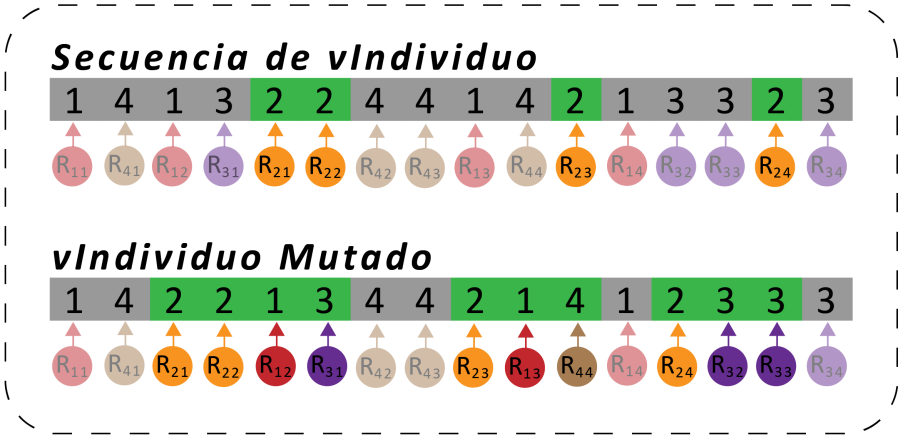


Figura 9: Ejemplo Operador Mutación Movimiento Operaciones de Job

ciación, pero de forma limitada para que puedan llevarse a cabo sus múltiples ejecuciones dependiendo del número de generaciones (*NumGenMILP*). Este procedimiento se detalla en la siguiente subsección. Así, de la población se selecciona aleatoriamente una cantidad de individuos equivalente al porcentaje indicado por el parámetro predefinido *PerMILP*, que serán aquellos modificados mediante el operador propuesto.

4.3.1. MILP por lotes

Para llevar resolver el modelo matemático generado a partir de los individuos del GA, el operador MILP divide el vector de secuencia (la codificación de las soluciones) en particiones de tamaño fijo según un parámetro de tamaño de lote predefinido. Por ejemplo, si se tiene un vector de secuencia con 12 operaciones y un tamaño de lote de 4, se obtienen 3 particiones (lotes). Para cada uno, se extraen únicamente los datos correspondientes a las operaciones contenidas en dicha partición, y se construye una instancia reducida del problema que será

Algorithm 7 Pseudocódigo del Método de Mutación 2: Intercambio de Job

```
Input: vIndividuo a Mutar
J = EnteroAleatorio(1, NumJobs)
vIndividuo = EliminarOperaciones(J)
L = EnteroAleatorio(0,1)
C = EnteroAleatorio(1, PosicionesPosiblesDesplazamiento)
if L == 0 then
    for a in vIndividuo.Actividades[J] do
        vIndividuo = InsertarOperación[a](PosiciónAnterior[J][a] + C)
    end for
else
    for a in vIndividuo.Actividades[J] do
        vIndividuo = InsertarOperación[a](PosiciónAnterior[J][a] - C)
    end for
end if
ReasignarModosAleatoriamente(vIndividuo)
Return: Mutado vIndividuo
```

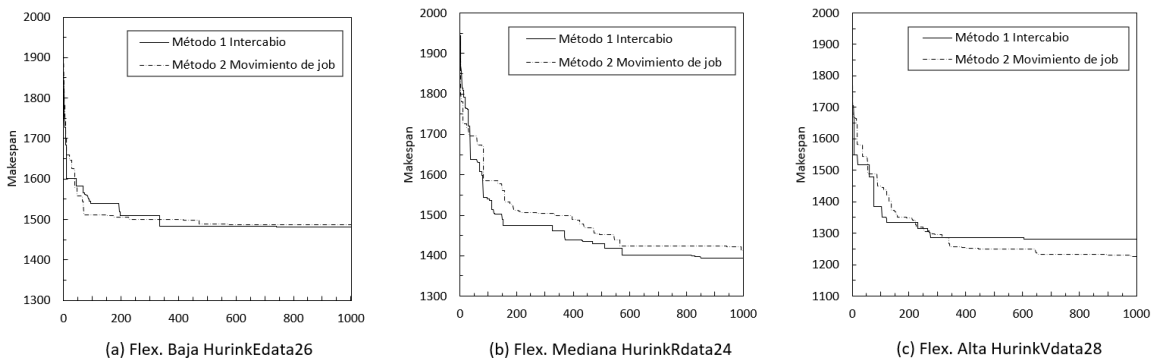


Figura 10: Gráfico de Convergencia métodos de Mutación

resuelta por el modelo MILP. Este proceso se repite secuencialmente hasta completar el total de operaciones. La relevancia de este enfoque radica en que la división depende directamente del vector de secuencia del individuo, por lo que individuos con secuencias distintas generan particiones diferentes a optimizar, lo que contribuye a la diversificación de las soluciones.

Los lotes de operaciones se resuelven aplicando dos veces el modelo matemático. En la primera ejecución, se utiliza el modelo estándar para obtener el makespan óptimo del lote correspondiente. En la segunda ejecución, se usa la función objetivo descrita en la ecuación (18), que busca minimizar la suma del tiempo de finalización de las actividades consideradas; y se modifica la restricción de la ecuación (2), reemplazando la variable C_{max} por el makespan obtenido en la primera ejecución. De esta manera, la segunda ejecución mantiene el makespan fijo, pero redistribuye las operaciones para que se programen lo más temprano posible, aprovechando de forma más efectiva las holguras entre actividades.

En la Fig. 11 se ilustra este proceso: la operación R_{42} se programa inicialmente en la máquina 3 durante la primera ejecución; sin embargo, en la segunda ejecución, esta operación se reasigna a la máquina 4, aprovechando un intervalo de inactividad disponible. Este ejemplo demuestra el beneficio de la doble ejecución al redistribuir las operaciones dentro del mismo makespan.

$$\text{Minimize} \sum_{j \in J} F_{j(O_j)} \tag{18}$$

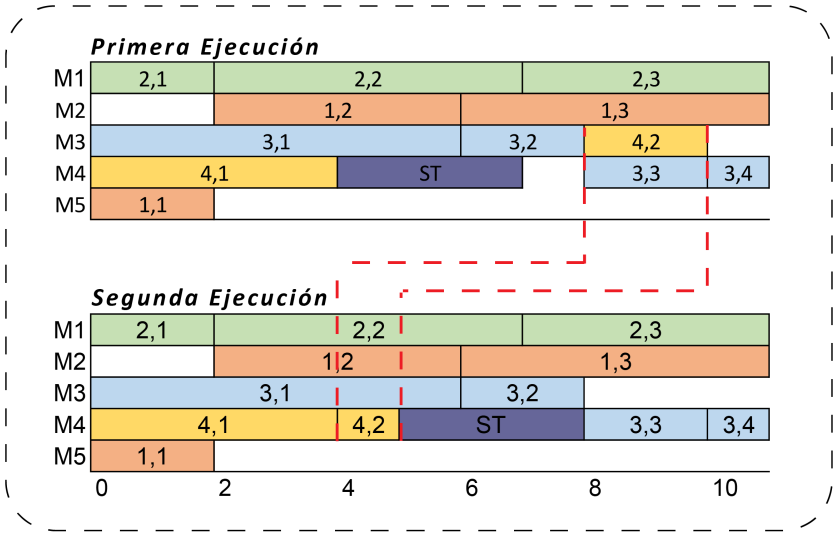


Figura 11: Diagrama de Gantt Modelo MILP Doble con el Caso de Prueba Kacem1

4.3.2. Tipo de Operadores MILP

La solución de cada lote requiere como datos de entrada la información de las operaciones contenidas en la partición en el orden indicado por el vector de secuencia del GA. La salida es la solución proporcionada por el modelo doble, codificada en los dos vectores de representación: el de secuencia y el de modos. Es importante señalar que, aunque el vector de secuencia forma parte de los datos de entrada, el orden de las operaciones puede cambiar durante la ejecución del modelo. El vector de modos, por su parte, es determinado completamente por la solución del modelo matemático, lo que permite obtener un individuo mejorado respecto al original.

La forma de ejecución de cada lote puede variar entre dos enfoques propuestos: una ejecución independiente, donde las soluciones de lotes anteriores no influyen en el lote actual, y una ejecución dependiente, donde se incorporan restricciones asociadas a los tiempos de inicio y finalización de operaciones resueltas en lotes previos. A continuación se describen ambos métodos.

4.3.3. Operador MILP Independiente

En el método independiente, cada lote se ejecuta sin considerar las soluciones de lotes anteriores. El aspecto clave de este enfoque reside en la consolidación de los resultados. Como se ilustra en la Fig. 12, los lotes ejecutados de manera independiente presentan sus tiempos ajustados desde el inicio del diagrama de Gantt. Al unificar los lotes, las operaciones del lote 2 se reprogramaran en su máquina asignada lo más temprano posible, respetando las restricciones de precedencia de operaciones y los tiempos de configuración asociados.

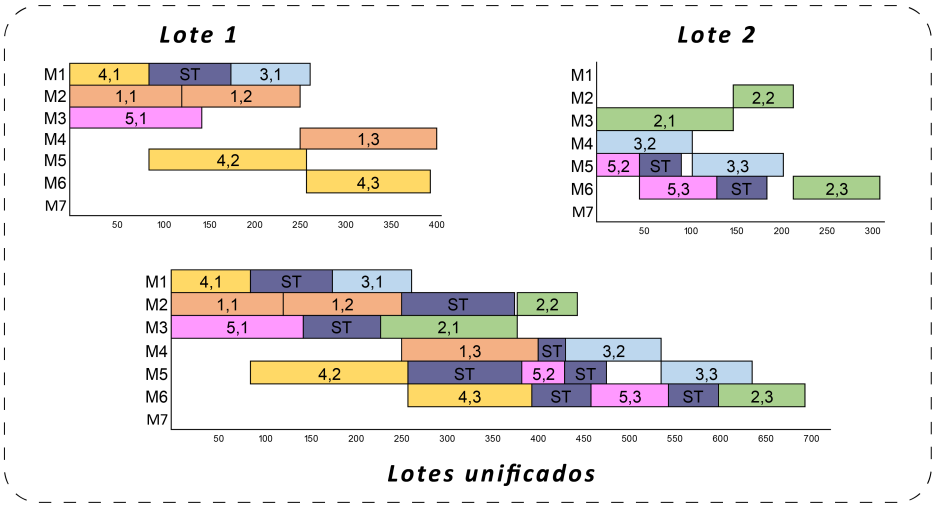


Figura 12: Diagrama de Gantt MILP Doble Independiente con el caso de Prueba Fattahi12

4.3.4. Operador MILP Dependiente

En el método dependiente, a diferencia del anterior, los tiempos de inicio (T_{jh}), finalización (F_{jh}) y la asignación de máquinas (Y_{ijh}) de las operaciones ya resueltas se fijan mediante restricciones adicionales en el modelo. De esta manera, al agregar las operaciones del siguiente lote, el número de variables de decisión se mantiene acotado al tamaño del lote, ya que las operaciones previas actúan como constantes. En la Fig. 13 se presenta un ejemplo de este método, donde las operaciones del lote 1 se encuentran fijadas y las operaciones del lote 2 se resuelven teniendo en cuenta la información del lote 1. Este proceso se implementa en cada lote del total obtenido de la secuencia de operaciones original del GA. De este modo, las programaciones correspondientes a los lotes previamente procesados se incorporan de manera acumulativa, induciendo una configuración específica de holguras disponibles para la programación del lote siguiente.

5. Resultados Computacionales

En esta sección se presentan los resultados computacionales de la implementación de la Mateheurística en diferentes casos de prueba. Todas las ejecuciones se llevaron a cabo utilizando un procesador Intel de 11a generación Core i5 con una velocidad de 2.70GHz y una memoria RAM de 16 GB a 2400MHz. Tanto el algoritmo genético como la mateheurística

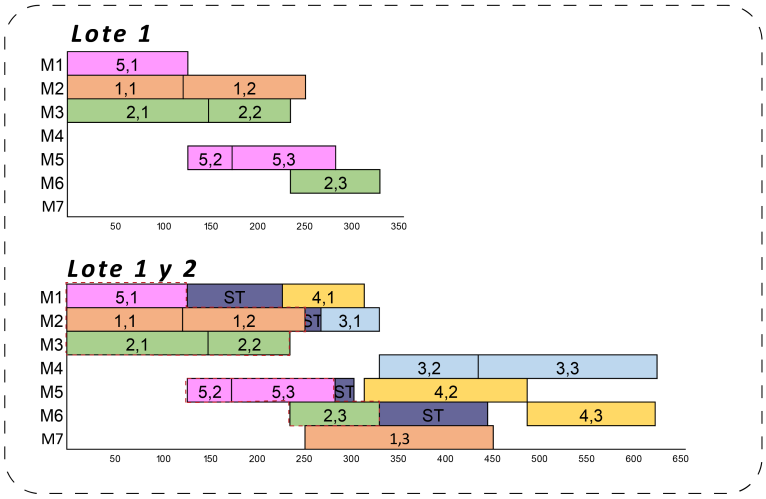


Figura 13: Diagrama de Gantt MILP Doble Dependiente con el caso de Prueba Fattahi12

fueron implementados en el lenguaje de programación C++, y para la ejecución del modelo matemático se empleó el lenguaje de modelado algebraico AMPL, con el solver CPLEX 12. Esta sección se divide en cuatro partes: en la subsección 5.1 se detalla la generación de casos de prueba; en la subsección 5.2 se muestra la parametrización del GA; la subección 5.3 detalla la paramerización del componente exacto de la Mateheurística; finalmente, en la subsección 5.4 se resumen los principales resultados.

5.1. Casos de prueba

Dentro de la literatura no se encuentra disponible un conjunto de casos de prueba estándar para el FJSSP-SDST, por lo que en este trabajo se generaron 276 problemas, con base el los casos de prueba encontrados para la generalización FJSSP, que se describe a continuación. El conjunto propuesto por Fattahi et al. (2007) suelen ser usados para evaluar métodos exactos, debido a que son problemas pequeños, desde dos trabajos que pueden tener dos o tres operaciones. Los casos de prueba de Kacem et al. (2002) consideran casos con flexibilidad total, es decir, que el número de modos posibles por operación es igual al número de máquinas. Además consideraron que cada modo tiene un tiempo de procesamiento distinto. Hurink et al. (1994) tomaron cinco conjunto de casos de prueba de la literatura del JSSP (Adams et al., 1988; Applegate & Cook, 1991; Carlier, 1978; Fisher & Thompson, 1963; Lawrence, 1984) y las adaptaron para el FJSSP realizando variaciones en el nivel de flexibilidad. Generaron tres variantes: iniciando con un nivel de flexibilidad bajo, es decir, pocas operaciones pueden ser asignadas a varias máquinas (Flexibilidad de 1,15), a este conjunto se le llamó *edata*; la segunda variante indica que varias operaciones pueden ser asignadas a varias máquinas (flexibilidad de 2,0), este conjunto se le llamó conjunto *rdata*; finalmente, la tercera variante, llamada *vdata*, permite que todas las operaciones puedan ser asignadas a varias máquinas (flexibilidad desde 2,5 hasta 5,0). Estos casos propuestos consideran que modos de una operación tienen el mismo tiempo de procesamiento.

Dauzère-pérès y Paulli (1997) generaron 18 casos de prueba con la propiedad distintiva

que el número de operaciones por trabajo siempre será mayor al número de máquinas. Indicando que, en cada una de las secuencias posibles para cada trabajo, siempre ocurrirá la recirculación. Chambers y Barnes (1996), utilizaron tres casos de prueba de la literatura del JSSP (la *mt10* de Fisher y Thompson (1963), y la *la24* y *la40* de Lawrence (1984)) para modificarlas con siete políticas de replicación de acuerdo a las máquinas que tenían mayor carga en los problemas originales. Estas instancias, al igual que las propuestas por Hurink et al. (1994), consideran que todos los modos tienen el mismo tiempo de procesamiento. Brandimarte (1993) propuso los casos de prueba con un mayor número de trabajos y máquinas, 30 y 15 respectivamente, además cada modo de una operación tiene tiempos de procesamientos distintos. En la Tabla 4 se resume los casos de prueba más relevantes para el FJSSP.

Tabla 4: Principales casos de prueba para el FJSSP.

Referencia	**Caso de Prueba**	**Tamaño (jobs x maquinas)**	**Adaptado de**
Brandimarte (1993)	mk1-mk15	(10 x 6) to (30 x 15)	Los datos fueron creados originalmente por el autor
Hurink et al. (1994)	abz5-abz9	(10 x 10) to (20 x 15)	Conjunto de datos original del JSSP propuesto por Adams et al. (1988)
	car1-car8	(7 x 7) to (14 x 4)	Conjunto de datos original del Flow Shop Problem propuesto por Carlier (1978)
	la01-la40	(10 x 5) to (30 x 10)	Conjunto de datos original del JSSP propuesto por Lawrence (1984)
	mt06, mt10 and mt20	(6 x 6), (10 x 10), (20 x 5)	Conjunto de datos original del JSSP propuesto por Fisher y Thompson (1963)
	orb1-orb10	(10 x 10)	Conjunto original de casos de prueba del JSSP propuesto por Applegate y Cook (1991)
Dauzère-pérès y Paulli (1997)	01a-18a	(10 x 5)	Introdujeron 18 Casos de prueba originales generadas para el Multiprocesor- JSSP
Chambers y Barnes (1996)	mt10x, mt10c1, mt10xx, mt10xy, mt10cc, mt10xxx, mt10xyz	(10 x 11), (10 x 12), (10 x 13)	Casos de prueba mt10 originales del JSSP propuesto por Fisher y Thompson (1963)
	setb4x, setb4c9, setb4xx, setb4xy, setb4cc, setb4xxx, setb4xyz	(15 x 11), (15 x 12), (15 x 13)	Casos de prueba la24 originales del JSSP propuesto por Lawrence (1984)
	seti5x, seti5c12, seti5xx, seti5xy, seti5cc, seti5xxx, set55xyz	(15 x 16), (15 x 17), (15 x 18)	Casos de prueba la40 originales del JSSP propuesto por Lawrence (1984)
Kacem et al. (2002)	kacem1-kacem4	(4 x 5) to (15 x 10)	Los datos fueron creados originalmente por el autor
Fattahi et al. (2007)	SFJS1 - SFJS10, MFJS1 - MFJS10	(2 x 2) to (12 x 8)	Los datos fueron creados originalmente por el autor

Como se mencionó anteriormente, ninguno de los casos de prueba tienen los SDST, así que fue necesario generarlos cumpliendo el teorema de la desigualdad triangular (Artigues & Feillet, 2008). Este implica que el SDST, denotado por S_{ijk} (tiempo de configuración de la máquina i al pasar del trabajo j al k), cumple para cada terna de distintos trabajos (j, k, x) la Ecuación (19), que establece que la suma de dos tiempos de configuración debe ser menor o igual al tiempo de configuración restante de la terna de trabajos. Como se muestra en la Fig. 14, donde los tiempos de configuración son representados como las longitudes de los lados del triángulo.

$$S_{ijx} \leq S_{ijk} + S_{ikx}, \qquad \forall (j \in J, x \in J) : j \neq x, \forall k \in J \setminus \{j, x\} \tag{19}$$

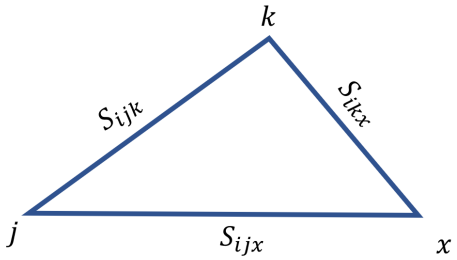


Figura 14: Representación gráfica de la desigualdad triangular

Para la generación de estos valores, cada SDST es generado aleatoriamente cumpliendo la desigualdad triangular para cada terna. Siendo el límite inferior de cero (no existe SDST), y el límite superior el promedio de los tiempos de procesamientos del caso de prueba. Estos límites se establecen, dado que por lo general, los SDST no suelen ser mayores a los tiempos de procesamiento y en ocasiones pueden ser muy pequeños o despreciables. Un ejemplo de esto lo podemos observar en el trabajo de Winklehner y Hauder (2022) donde estudian un caso real de la industria, donde el tiempo de procesamiento promedio de las operaciones es aproximadamente de 21 horas, mientras que los SDST son 0, 4, 8 y 15 horas.

Las instancias resultantes se estructuraron según el formato mostrado en la Fig. 15. Donde la primera línea indica el número de trabajos dado por N, seguido por el número de máquinas M y finalmente el parámetro V indica el nivel de flexibilidad del caso de prueba, entendido como, el número de modos promedio por operación. Las siguientes N líneas representan los trabajos, cuyo inicio en cada línea indica el número de actividades que tiene el trabajo asociado. En la siguiente columna está el número de modos de la operación, que vendrá acompañado con la pareja de la máquina y el tiempo de procesamiento. Los últimos datos muestran las matrices cuadradas $j \times j$ de los SDST. Estas se repiten de acuerdo con el número de máquinas M.

En total se consideraron 276 casos del FJSSP y todos ellos fueron adaptados para que incluyan los SDST. estas adaptaciones están disponibles en el repositorio https://github.com/dmorill/FJSSP_SDST_Instances.

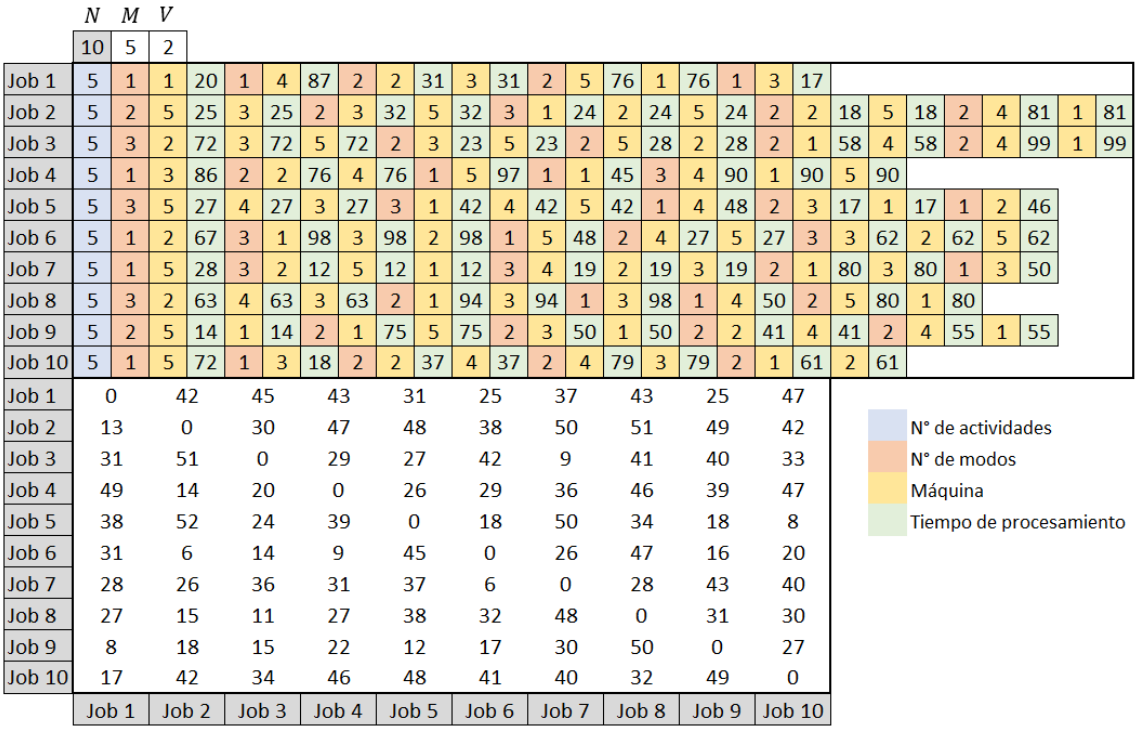


Figura 15: Formato de presentación de los casos de prueba para el FJSSP-SDST

5.2. Parametrización del GA

Los parámetros ajustables más relevantes de GA propuesto son: *mutProb* (probabilidad de mutación), *repPerc* (porcentaje de reemplazo directo) y *popSize* (tamaño de la población). Para determinar la mejor combinación de estos parámetros, se realizó un Diseño de Experimentos (DOE) como método sistemático de análisis (Kechagias et al., 2020). En particular, se utiliza el DOE factorial fraccional basado en matrices ortogonales de Taguchi, que permite identificar los factores más influyentes con un número reducido de experimentos en comparación con el DOE factorial completo. Este método ha sido ampliamente usado para la parametrización de metaheurísticas (Gnanavelbabu et al., 2021; Sahli et al., 2022; Zhao et al., 2023). Los niveles definidos para cada parámetro se detallan en la Tabla 5, con cuatro niveles por factor definidos dentro de rangos operativos razonables.

Tabla 5: Niveles DOE Taguchi

Parámetros	Nivel 1	Nivel 2	Nivel 3	Nivel 4
mutProb	5 %	10 %	15 %	20 %
repPerc	10 %	20 %	30 %	40 %
popSize	100	150	200	250

El método de Taguchi distingue entre factores controlables, cuyos niveles pueden establecerse deliberadamente, y factores de ruido, cuyos niveles no pueden controlarse (Maghsoodloo

et al., 2004). La evaluación se realiza mediante la relación señal-ruido (S/N), cuya formulación depende de la naturaleza de las característica de calidad: "menor es mejor", "mayor es mejor" o "el objetivo es mejor". La aplicación de este método permite identificar las configuraciones de los factores controlables que minimizan la influencia de los factores de ruido, promoviendo mejor rendimiento del algoritmo (Xia et al., 2019).

En este estudio se considereraron casos de prueba de rango de tamaños y niveles de flexibilidad amplio. En particular, se seleccionan 6 casos de prueba que abarcan el espectro completo de tamaños y grados de flexibilidad, como se muestra en la Tabla 6. Dichos casos incluyen desde 50 hasta 300 operaciones y niveles de flexibilidad entre 1,20 y 6,58, permitiendo que los valores seleccionados para *mutProb*, *repPerc* y *popSize* sean representativos tanto de instancias moderadas como de escenarios de alta complejidad.

Tabla 6: Casos de Prueba utilizados en el método Taguchi

Caso de Prueba	**Tamaño (Jobs x Maquinas x Actividades)**	**Flexibilidad**	**Tipo (Tamaño/Flexbilidad)**
HurinkEdata4	10 x 5 x 50	1.20	Pequeña / Baja
BrandimarteMk2	10 x 6 x 58	4.10	Pequeña / Alta
HurinkEdata24	15 x 10 x 150	1.15	Mediana / Baja
HurinkVdata24	15 x 10 x 150	4.77	Mediana / Alta
DPpaulli10a	15 x 8 x 293	1.24	Grande / Baja
HurinkVdata47	20 x 15 x 300	6.58	Grande / Alta

Posteriormente, siguiendo las matrices ortogonales de Taguchi y considerando el número de factores y niveles del algoritmo, se emplea el arreglo L_{16}, detallado en la Tabla 7. Los resultados obtenidos de estos experimentos, aplicados a los 6 casos de prueba seleccionados se presentan en la Tabla 8, ejecutando tres réplicas por cada combinación de caso de prueba y experimento.

Es importante señalar que la naturaleza del objetivo corresponde a la categoría "menor es mejor", ya que la función de aptitud del algoritmo genético está vinculada con la minimización del makespan. En este contexto, se aplica la ecuación (20) para calcular las relaciones S/N. Los resultados se presentan en la Tabla 9, donde se observa que el mayor valor de la relación S/N corresponde al experimento número 4. Este resultado se corrobora con la Fig. 16, en la que se identifica que los niveles más apropiados son: *mutProb* del 5 %, *repPerc* del 40 % y un tamaño de población de 250. Además, se observa que el parámetro *repPerc* tiene un impacto significativo en la calidad del algoritmo, ya que exhibe una variación más pronunciada en comparación con los demás parámetros.

$$S/N = -10 \cdot \log_{10}\left(\frac{1}{n}\sum_{i=1}^{n} Y_i^2\right) \tag{20}$$

Adicionalmente, se realiza un análisis clasificando los casos de prueba por tamaño, cuyos resultados se presentan en la Fig. 17. En este análisis persiste la tendencia que resalta el impacto del porcentaje de reemplazo en la variación de la relación S/N, coincidiendo con el valor del 40 % obtenido en el análisis completo para los casos de prueba medianos y grandes.

Tabla 7: Arreglo L26 Taguchi

N° Experimento	mutProb	repPerc	Tamaño Población
1	0.05	0.1	100
2	0.05	0.2	150
3	0.05	0.3	200
4	0.05	0.4	250
5	0.1	0.1	150
6	0.1	0.2	100
7	0.1	0.3	250
8	0.1	0.4	200
9	0.15	0.1	200
10	0.15	0.2	250
11	0.15	0.3	100
12	0.15	0.4	150
13	0.2	0.1	250
14	0.2	0.2	200
15	0.2	0.3	150
16	0.2	0.4	100

Para los casos pequeños, el análisis sugiere un porcentaje de reemplazo del 20 %. Se observan coincidencias similares para el tamaño de la población, con un valor de 250 para los casos medianos y grandes, y de 200 para los pequeños. Un resumen de los valores seleccionados por tamaño se presenta en la Tabla 10. Con base en estos resultados, se configuran los valores de los parámetros según el tamaño del caso de prueba, con el fin de mejorar el rendimiento del algoritmo genético.

5.3. Selección del tamaño de lote para el operador MILP

Como se mencionó en la sección 4.3, el operador MILP requiere definir un tamaño de lote específico, debido a dos principales razones: primero, no es viable procesar el caso de prueba completo, y segundo, el tamaño de lote es el mecanismo en el que el GA aporta información al operador MILP mediante el vector de secuencia, es otras palabras, es la principal manera de vincular el operador exacto en la metodología de búsqueda del GA. La división en lotes de operaciones también es una estrategia para obtener soluciones factibles en tiempos razonables. En consecuencia, la elección del tamaño de lote es un aspecto esencial del diseño de la Mateheurística.

En una primera etapa, se realiza un análisis comparativo entre el MILP independiente y el MILP dependiente. Para ello se selecciona el caso de prueba *HurinkRdata24*, de tamaño y nivel de flexibilidad medios, y se ejecutan pruebas incrementando el tamaño del lote de tres en tres. En cada configuración se registran el tiempo de cómputo total requerido para la unificación de todos los lotes y el makespan obtenido. Los resultados se presentan en la Fig. 18, donde el primer gráfico muestra el comportamiento del tiempo de cómputo y el segundo la evolución del makespan en función del tamaño de lote. La figura de tiempos se observa una diferencia marcada a favor del enfoque independiente. Al fijar las restricciones

Tabla 8: Resultados Experimentos de Taguchi

N°	HurinkEdata4			BrandimarteMk2			HurinkEdata24			HurinkVdata24			DPpaulli10a			HurinkVdata47		
1	884	893	903	53	50	52	1706	1699	1698	1519	1540	1565	4305	4222	4339	1113	1102	1114
2	878	874	879	47	46	46	1688	1663	1636	1433	1537	1482	4269	4352	4130	1079	1083	1078
3	874	878	869	48	46	49	1557	1614	1598	1384	1389	1407	4106	3978	4101	1030	997	1038
4	896	878	874	47	46	46	1526	1607	1542	1373	1341	1325	3964	4052	3962	1017	1025	958
5	896	884	894	50	49	49	1651	1720	1624	1602	1496	1536	4192	4257	4196	1099	1108	1087
6	879	874	882	51	48	48	1670	1616	1565	1470	1510	1469	4304	4270	4450	1051	1109	1085
7	882	881	878	46	44	47	1604	1578	1587	1319	1378	1362	3954	4100	4170	942	998	1002
8	892	882	898	47	50	46	1548	1588	1529	1354	1413	1342	3894	4060	4082	967	1014	964
9	874	880	878	46	49	50	1634	1595	1624	1504	1503	1518	4233	4218	4452	1062	1109	1082
10	878	874	874	44	47	44	1583	1575	1578	1422	1416	1396	4132	4063	4155	1003	1089	992
11	876	876	880	46	51	48	1666	1556	1581	1453	1459	1432	4344	4257	4220	1084	1035	1058
12	874	884	879	47	48	48	1616	1605	1576	1374	1364	1414	4058	4174	4164	1005	1023	1027
13	887	894	874	47	44	49	1621	1640	1637	1556	1520	1539	4334	4364	4241	1070	1110	1104
14	878	874	874	46	45	47	1631	1572	1608	1474	1480	1447	4275	4200	4009	1098	1026	1016
15	885	878	893	47	48	41	1622	1649	1648	1447	1409	1407	4005	4192	3918	1099	1029	1026
16	903	893	880	50	49	50	1558	1550	1565	1365	1454	1461	4161	4188	4215	1044	1071	1060

de las operaciones ya resueltas, el enfoque dependiente incrementa la carga del modelo en cada iteración, lo que se traduce en un mayor tiempo de procesamiento. Esta situación se aprecia con más detalle en la Fig. 19, donde se descompone el tiempo total en tiempo de cargue (tiempo que consume el proceso MILP excluyendo la resolución) y tiempo de solución (suma de los tiempos de resolución de cada lote). La prueba se realiza sobre el mismo caso de prueba *HurinkRdata24* con un tamaño de lote igual a 10, lo que implica la resolución de 15 lotes para completar la solución. Los resultados muestran que el tiempo de cargue del MILP dependiente crece de manera pronunciada a medida que se resuelve cada lote, mientras que en el MILP independiente el incremento es marginal.

Desde el punto de vista del tiempo de cómputo, el enfoque independiente presenta una ventaja considerable. Sin embargo, la calidad de las soluciones muestra un comportamiento contrario. Como se observa en la Fig. 18, el enfoque dependiente obtiene sistemáticamente mejores valores de makespan que el independiente para todos los tamaños de lote analizados, lo que implica una relación entre tiempo y calidad que debe considerarse en la configuración final de la Mateheurística.

Finalmente, se estudia el efecto del tamaño de lote en instancias de distinta escala, comprendidas entre 36 y 100 actividades, todas ellas con niveles elevados de flexibilidad, de manera que representan escenarios de máxima complejidad y, en consecuencia, de mayor demanda computacional. En cada caso se evalúa cómo varían simultáneamente el makespan y el tiempo total de cómputo al incrementar el tamaño del lote. Los resultados para el modelo dependiente y el modelo independiente se presentan en las figuras 20 y 21, respectivamente. En estas figuras se aprecia que, salvo en el caso de prueba pequeño, los tiempos de cómputo del enfoque dependiente son sistemáticamente superiores a los del enfoque independiente. No obstante, la forma de las curvas es similar en todas las instancias: para tamaños de lote muy pequeños el tiempo puede ser relativamente alto, luego tiende a disminuir conforme aumenta el tamaño de los lotes hasta alcanzar una zona intermedia en la que el tiempo es mínimo, y a partir de cierto punto vuelve a crecer de forma pronunciada. Por otro lado, la tendencia del makespan es, en general, decreciente a medida que aumenta el tamaño del lote.

Considerando conjuntamente ambas métricas, un tamaño de lote igual a 10 se identifica

Tabla 9: Resultados S/N y promedio por experimento

N° Exp.	mutProb	repPerc	Tam. Poblacion	Relaciones S/N	Promedio
1	0.05	0.1	100	-66.32	1597.61
2	0.05	0.2	150	-66.19	1566.67
3	0.05	0.3	200	-65.79	1497.94
4	0.05	0.4	250	-65.64	1471.06
5	0.1	0.1	150	-66.19	1577.22
6	0.1	0.2	100	-66.30	1575.06
7	0.1	0.3	250	-65.78	1487.33
8	0.1	0.4	200	-65.67	1476.11
9	0.15	0.1	200	-66.26	1572.83
10	0.15	0.2	250	-65.88	1509.17
11	0.15	0.3	100	-66.17	1551.22
12	0.15	0.4	150	-65.90	1510.00
13	0.2	0.1	250	-66.30	1585.06
14	0.2	0.2	200	-66.01	1533.33
15	0.2	0.3	150	-65.82	1513.5
16	0.2	0.4	100	-66.01	1528.72

Tabla 10: Parámetros Seleccionados por Tamaño de los Casos de Prueba

Clasificación	mutProb	repPerc	Tamaño Población
Análisis Completo	5 %	40 %	250
Casos Pequeños	15 %	20 %	200
Casos Medianos	15 %	40 %	250
Casos Grandes	5 %	40 %	250

como el punto en el que se obtiene una reducción importante del makespan antes de que el tiempo de cómputo comience a crecer de manera acelerada, tanto para el MILP independiente como para el dependiente. Con tamaños de lote menores, la reducción en tiempo no compensa la pérdida de calidad, mientras que con tamaños superiores la mejora en makespan es marginal y el tiempo de cómputo crece considerablemente. Por lo tanto, se adopta un tamaño de lote de 10 operaciones como configuración del operador MILP.

5.4. Comparación de los operadores MILP integrados en el GA

5.4.1. Mateheurística con el Operador MILP independiente

A continuación, se presenta un análisis comparativo de los casos de prueba mediante tres métodos: El GA puro con los parámetros optimizados, el modelo MILP descrito en la sección 2.1 con un tiempo límite de 10 minutos por instancia, y la Mateheurística propuesta basada en la división por lotes del MILP independiente. El GA y de la Mateheurística, se ejecutaron 3 veces y cada prueba se ejecutó con un máximo de 1000 generaciones, nivel en el cual ambos métodos suelen alcanzar la convergencia. El operador MILP se aplica cada 100 generaciones

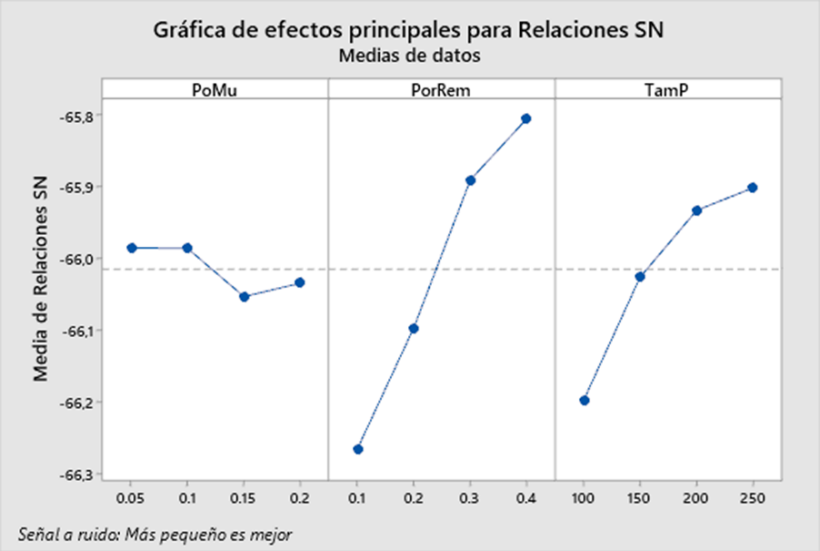
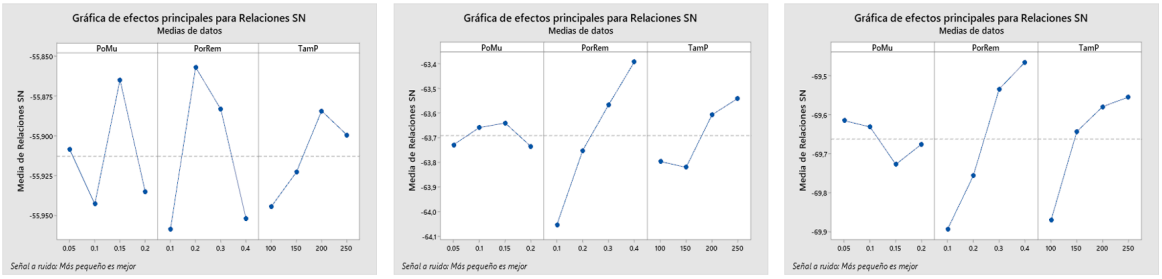


Figura 16: Medición relaciones S/N completo



(a) Casos de Prueba Pequeños

(a) Casos de Prueba Medianos

(a) Casos de Prueba Grandes

Figura 17: Medición relaciones S/N por Tamaño de los Casos de Prueba

sobre el 10 % de la población. Los resultados de makespan reportados corresponden al promedio de todas las ejecuciones y se evalúan mediante el indicador MRE (Mean Relative Error), cuya expresión se presenta en la ecuación (21). El MRE cuantifica la diferencia porcentual entre el makespan obtenido y la cota inferior (LB), debido a que el valor óptimo de las instancias es desconocido. Estas cotas inferiores se tomaron de Behnke y Geiger (2012), donde se establecen los límites para el FJSSP sin SDST; dado que el valor óptimo del problema con SDST es necesariamente igual o superior a dichos LB, resulta razonable adoptarlos como referencia válida para la evaluación del FJSSP-SDST.

$$MRE = \frac{C_{\text{máx}} - LB}{LB}, \quad \text{dónde } C_{\text{máx}} \text{ es le makespan obtenido y LB el Lower Bound.} \quad (21)$$

Los tres métodos se ejecutaron sobre las 276 instancias descritas en la sección 5.1, que cubren un amplio rango de tamaños y distintos niveles de flexibilidad. El análisis se realizó

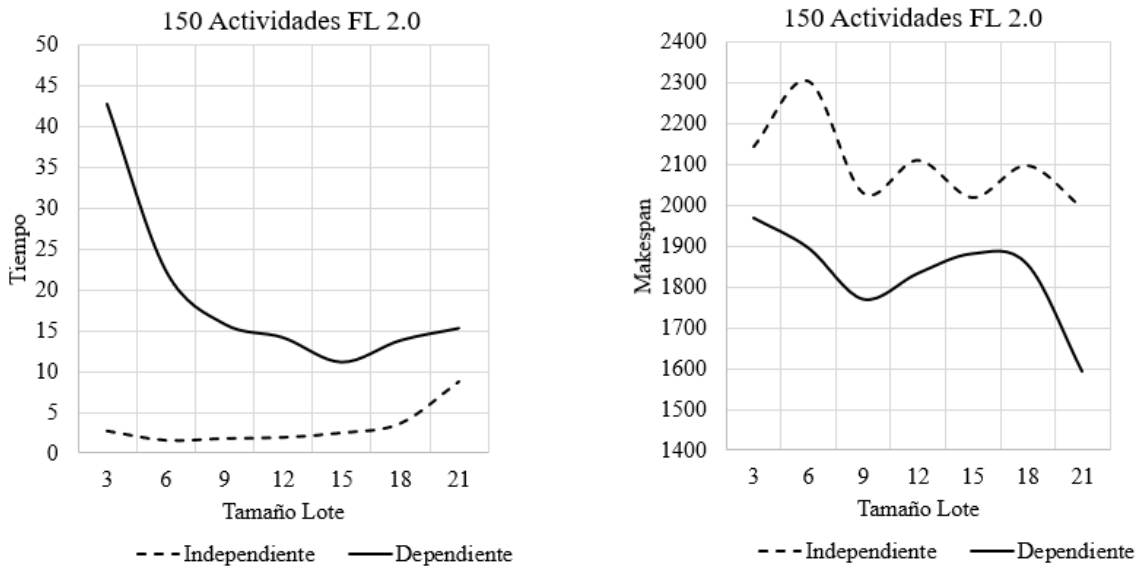


Figura 18: Tamaño de lote MILP Independiente vs Dependiente (Caso de Prueba HurinkRdata24

bajo dos criterios: (i) tamaño del problema, definido por el número total de actividades de cada instancia, y (ii) nivel de flexibilidad. En ambos casos se establecieron tres categorías (baja, media y alta), cuyos resultados promedio de MRE se presentan en las Tablas 11 y 12.

Tabla 11: Promedio MRE por tamaño de instancia

Total	Tamaño	Rango Actividades	MILP	GA	Independiente
164	Bajo	[0, 100]	172,8 %	74,8 %	72,4 %
74	Mediana	[101, 250]	355,6 %	74,7 %	73,6 %
38	Alta	[251, 387]	550,4 %	86,7 %	85,8 %

Tabla 12: Promedio MRE por nivel de flexibilidad

Total	Flex.	Rango de Flex.	MILP	GA	Independiente
180	Baja	[0 a 2,2]	171,4 %	77,5 %	77,3 %
45	Mediana	(2,2 a 4]	265,0 %	66,8 %	64,7 %
51	Alta	(4 a 10]	709,8 %	80,8 %	73,5 %

Dado que el GA y la Mateheurística son métodos estocásticos, además del valor promedio de MRE es relevante cuantificar la estabilidad de sus resultados entre ejecuciones. Con este fin, se calcula la desviación relativa del MRE para cada categoría de tamaño y flexibilidad, obtenida a partir de las tres corridas independientes por instancia. En el caso del MILP, al ser un modelo exacto determinista, no se reportan medidas de desviación, pues las soluciones no generarn una gran variación entre ejecuciones. Las Tablas 13 y 14 presentan las desviaciones de MRE por tamaño de instancia y nivel de flexibilidad, respectivamente.

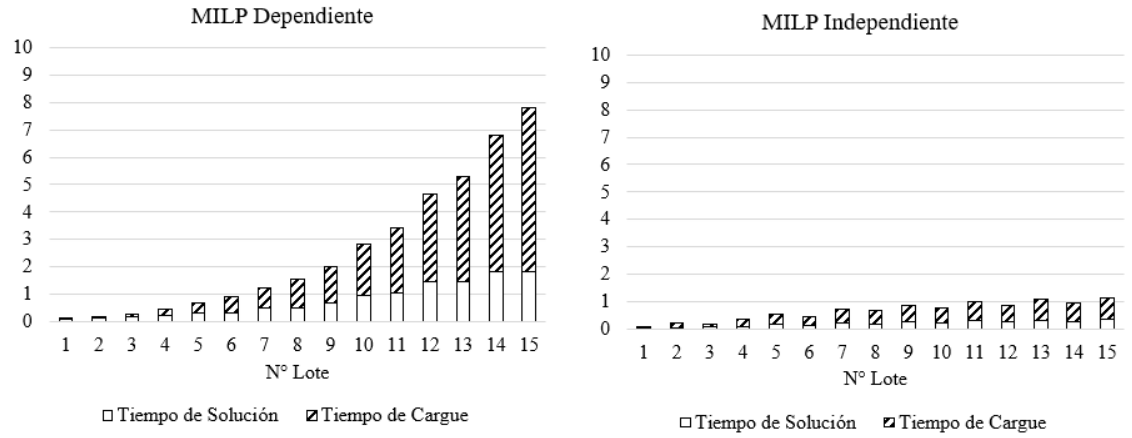


Figura 19: Tiempos de Cargue y Tiempos de Solución MILP Dependiente e Independiente

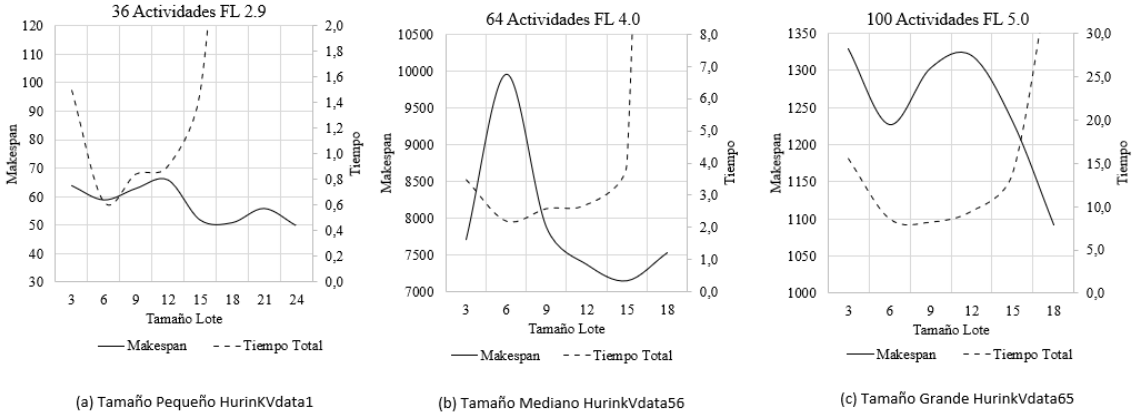


Figura 20: Gráficos Tamaño de lote Dependiente (Makespan vs Tiempo)

Las desviaciones observadas se mantienen en valores reducidos (entre aproximadamente 3 % y 5 %), lo que indica que tanto el GA como la Mateheurística producen resultados consistentes entre ejecuciones. Por tamaño de instancia, la variabilidad tiende incluso a disminuir ligeramente en problemas más grandes, lo que sugiere que el comportamiento promedio de ambos métodos es estable cuando se incrementa la dimensión del problema. Al analizar los promedios de MRE por tamaño, la Mateheurística obtiene sistemáticamente valores menores que el GA: en instancias de tamaño bajo la diferencia es de aproximadamente 2,5 %, en tamaño medio de 1,1 % y en tamaño alto de 0,9 % puntos. Estas mejoras son moderadas y de magnitud comparable a las desviaciones reportadas, por lo que puede concluirse que la Mateheurística presenta un desempeño ligeramente superior al GA en términos de MRE promedio, aunque la ventaja no es drástica en función del tamaño.

En términos de flexibilidad, la comparación resulta más reveladora. Para flexibilidad baja ambos métodos presentan prácticamente el mismo MRE promedio. En flexibilidad media, la

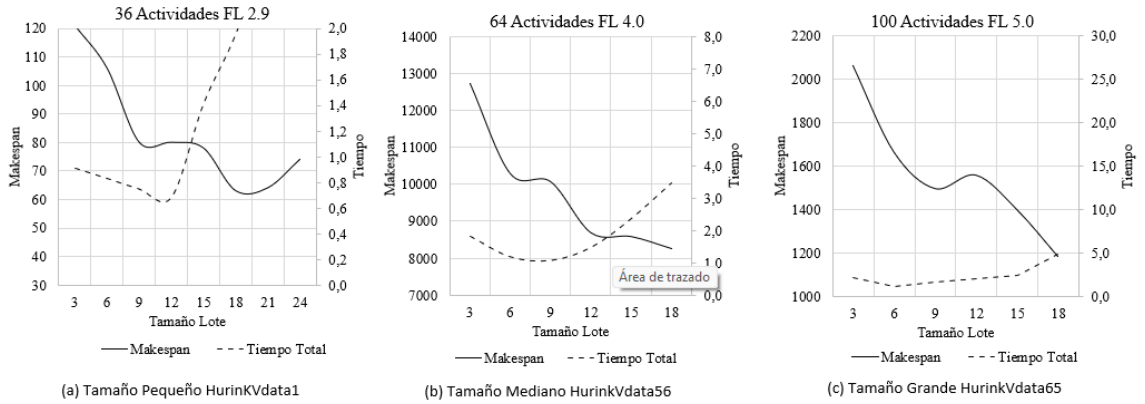


Figura 21: Gráficos Tamaño de lote Independiente (Makespan vs Tiempo)

Tabla 13: Desviación MRE por tamaño de instancia

Total	Tamaño	Rango Actividades	GA	Independiente
164	Bajo	[0, 100]	4,0 %	3,1 %
74	Mediana	[101, 250]	3,3 %	3,7 %
38	Alta	[251, 387]	2,7 %	3,2 %

Mateheurística mejora al GA en cerca de 2,1 %, mientras que en flexibilidad alta la diferencia se incrementa hasta 7,3 %. En este último caso, además, la desviación de la Mateheurística 3,5 % es claramente menor que la del GA 5,2 %, lo que indica que el operador MILP no solo mejora la calidad promedio de las soluciones en instancias altamente flexibles, sino que también las hace más estables entre ejecuciones. En conjunto, estos resultados permiten afirmar que la Mateheurística independiente presenta un desempeño superior al GA, especialmente en problemas de alta flexibilidad.

Como resultados, el MILP presenta los valores de MRE más elevados en todos los grupos analizados. Además, en 18 instancias no logra encontrar una solución factible dentro del límite de 10 minutos; estas instancias corresponden a tamaños grandes (entre 225 y 387 actividades) con flexibilidades elevadas (de 3 a $6,5$); estos valores se omiten en los cálculos del MRE del MILP al no contar con un valor de referencia. Su desempeño se deteriora a medida que aumenta tanto el tamaño del problema como el nivel de flexibilidad, pasando de valores moderados en las categorías bajas a valores considerablemente superiores en las categorías media y alta. Por su parte, el GA mantiene valores de MRE mucho menores y con una diferencia más contenida entre categorías. La Mateheurística independiente exhibe un comportamiento similar al GA, pero con los menores MRE en todos los casos.

En cuanto al tiempo de cómputo, en la Tabla 16 se presentan los promedios obtenidos. Para el MILP, los tiempos se mantienen alrededor de los 600 segundos, correspondientes al límite de 10 minutos establecido como criterio de parada, excepto en instancias de tamaño pequeño, donde en 22 de las 276 instancias el MILP encuentra la solución óptima en menos de 600 segundos. Estas instancias tienen menos de 36 actividades y, en su mayoría, flexibilidades inferiores a $2,6$; solo tres presentan flexibilidades altas (3, 7 y 10), correspondientes a los casos

Tabla 14: Desviación MRE por nivel de flexibilidad

Total	Flex.	Rango de Flex.	GA	Independiente
180	Baja	[0 a 2,2]	3,1 %	2,9 %
45	Mediana	(2,2 a 4]	3,6 %	4,5 %
51	Alta	(4 a 10]	5,2 %	3,5 %

de prueba de Kacem et al. (2002). Estas instancias se presentan en la Tabla 15, los resultados muestran que, si bien el MILP obtiene el resultado óptimo en las primeras tres instancias (de hasta 30 actividades), en el cuarto caso (56 actividades y flexibilidad de 10) no logra alcanzar el óptimo dentro del tiempo límite, y su makespan es ampliamente superado por el *GA* y, aún más, por la Mateheurística independiente. Los valores de MRE en este caso son del 625 % para el MILP, 233 % para el GA y 142 % para la Mateheurística. En términos de tiempo, el GA tarda 27 segundos y la Mateheurística independiente 87 segundos, resultando más eficientes que el MILP.

Tabla 15: Makespan de las instancias de Kacem

Instancias	*Jobs*	*Ma*	*H*	Flex.	LB	MILP	GA	Independiente
Kacem1	4	5	12	5,00	11	11	12	11
Kacem2	10	7	29	7,00	11	11	27	19
Kacem3	10	10	30	10,00	7	7	17	11
Kacem4	15	10	56	10,00	12	87	40	29

También se observa que el GA es considerablemente más rápido que el MILP en instancias de hasta 250 actividades, y solo un 24 % más lento en instancias de tamaño alto. Esto indica que el GA no solo encuentra mejores soluciones que el MILP, sino que además lo hace en menor tiempo en la mayoría de los casos. La Mateheurística independiente, por su parte, solo es más rápida que el MILP en tamaños pequeños; al compararla con el GA, sus tiempos resultan aproximadamente 3 veces mayores en promedio para todos los tamaños, debido al alto costo computacional del operador MILP incorporado en la Mateheurística. No obstante, aunque se otorgara al GA un tiempo equivalente al de la Mateheurística, no se obtendrían mejores resultados, ya que el GA no cuenta con mecanismos de búsqueda adicionales.

Tabla 16: Promedio de tiempos (Segundos)

Total Instancias	Tamaño	Rango Actividades	MILP	GA	Independiente
164	Bajo	[0, 100)]	528	58	339
74	Mediana	[101, 250]	602	301	989
38	Alta	[251, 387]	603	736	2233

5.4.2. Mateheurística con el Operador MILP Dependiente

En este apartado se realiza el análisis de la Mateheurística con el operador dependiente. Como se mostró en la sección 5.3, este operador presenta un crecimiento exponencial en el tiempo de carga cada vez que se procesa un nuevo lote, ya que requiere fijar las operacio-

nes resueltas en los lotes anteriores. Por esta razón, se opta por ejecutarlo únicamente en instancias con menos de 80 actividades, lo que permite obtener, para este subconjunto, tiempos y resultados comparables entre los métodos MILP, GA, Mateheurística independiente y Mateheurística dependiente.

Con el filtro descrito anteriormente se obtienen 71 instancias, que se analizan bajo los mismos criterios de tamaño y flexibilidad de la sección previa. No obstante, dado que se trata de un subconjunto distinto, es necesario recalcular los valores comparativos. Por otra parte, los rangos de tamaño se redefinen en tres categorías que cubran las 71 instancias. Los resultados se presentan en las Tablas 17 y 18.

Tabla 17: Promedio MRE por tamaño de instancia

Total	Tamaño	Rango Actividad	MILP	GA	Independiente	Dependiente
24	Bajo	[0, 36]	21,7 %	39,1 %	31,0 %	30,2 %
24	Mediana	[37, 57]	79,5 %	58,7 %	54,9 %	46,3 %
23	Alta	[58, 80]	77,5 %	53,3 %	51,5 %	46,0 %

Tabla 18: Promedio MRE por nivel de flexibilidad

Total	Flexibilidad	Rango de Flex.	MILP	GA	Independiente	Dependiente
39	Baja	(0 a 2,2)	44,7 %	39,8 %	39,8 %	37,5 %
26	Mediana	(2,2 a 4)	62,8 %	50,7 %	51,0 %	44,0 %
6	Alta	(4 a 10)	139,2 %	113,5 %	61,5 %	44,2 %

Respecto al análisis por tamaño, en el rango de instancias con menos de 36 actividades, el MILP logra encontrar el valor óptimo en 22 de las 24 instancias, por lo que su MRE promedio resulta el mejor entre los cuatro métodos. En este mismo rango, la Mateheurística dependiente e independiente obtienen un MRE alrededor de de 31 %, mejor que el GA (39,1 %), lo que indica que, aun cuando el MILP es competitivo en tamaños muy pequeños, el operador MILP aporta una mejora apreciable frente a los métodos puramente metaheurpisticos. A partir de 36 actividades, el MILP ya no es capaz de encontrar el valor óptimo dentro del tiempo límite, y en consecuencia su MRE promedio se deteriora significativamente (79,5 % y 77,5 % en tamaños medio y alto). En contraste, los demás métodos, en el rango entre 37 y 80 actividades, mantienen MRE promedio entre 46,0 % y 58,7 %, siendo la Mateheurística dependiente la que obtiene los menores valores en ambos casos (46,3 % frente a 58,7 % del GA y 54,9 % de la independiente en tamaño medio; 46,0 % frente a 53,3 % y 51,5 % en tamaño alto). Estas diferencias, de entre 5 y 12 puntos porcentuales de MRE, muestran que el operador MILP dependiente mejora de forma consistente la calidad promedio de las soluciones respecto al GA y a la Mateheurística independiente en problemas de mayor tamaño.

Al analizar por nivel de flexibilidad, se observa que tanto el MILP como el GA presentan un deterioro considerable del MRE promedio a medida que la flexibilidad aumenta, pasando de 44,7 % y 39,8 % en flexibilidad baja a 139,2 % y 113,5 % en flexibilidad alta. La Mateheurística independiente también empeora, pero con un incremento mucho menor (de 39,8 % a 61,5 %). En contraste, la Mateheurística dependiente mantiene MRE dentro de un rango mucho más acotado: 37,5 % en flexibilidad baja, 44,0 % en media y 44,2 % en alta. En flexi-

bilidad baja las diferencias frente al GA y a la Mateheurística independiente son moderadas (alrededor de (2,3 %), pero en flexibilidad media la mejora aumenta a aproximadamente 7 % (50,7 % y 51,0 % frente a 44,0 %), y en flexibilidad alta la Mateheurística dependiente supera ampliamente a todos los métodos, con una reducción de MRE de casi 69 % frente al GA y de alrededor de 17 % puntos frente a la Mateheurística independiente. Aunque el número de instancias con flexibilidad alta en este subconjunto es reducido (6 casos), la diferencia de MRE promedio es suficientemente grande para indicar que el operador dependiente resulta especialmente efectivo en escenarios de alta flexibilidad.

Al igual que en la sección anterior, los métodos abordados son estocásticos, por lo tanto es relevante cuantificar la estabilidad de sus resultados entre ejecuciones. Con este fin, se calcula la desviación relativa del MRE para cada categoría de tamaño y flexibilidad, obtenida a partir de las tres corridas independientes por instancia. Las Tablas 19 y 20 presentan las desviaciones de MRE por tamaño de instancia y nivel de flexibilidad, respectivamente.

Tabla 19: Desviación MRE por tamaño de instancia

Total	Tamaño	Rango Actividad	GA	Independiente	Dependiente
24	Bajo	[0, 36]	4,0 %	2,3 %	1,9 %
24	Mediana	[37, 57]	3,3 %	2,9 %	3,4 %
23	Alta	[58, 80]	3,1 %	3,2 %	3,0 %

Tabla 20: Desviación MRE por nivel de flexibilidad

Total	Flexibilidad	Rango de Flex.	GA	Independiente	Dependiente
39	Baja	(0 a 2,2)	2,0 %	2,0 %	3,0 %
26	Mediana	(2,2 a 4)	3,6 %	3,7 %	3,4 %
6	Alta	(4 a 10)	12,2 %	3,7 %	2,9 %

Las desviaciones observadas tienen un comportamiento similar al de la sección anterior, se mantienen en valores reducidos (entre aproximadamente 2 % y 5 % en la mayoría de las categorías), lo que indica que los tres métodos producen resultados consistentes entre ejecuciones. Por tamaño de instancia, la Mateheurística dependiente presenta la menor variabilidad en el rango de tamaño bajo (1,9 % frente a 4,0 % del GA y 2,3 % de la independiente), mientras que en tamaños medio y alto las desviaciones de los tres métodos son muy similares. En términos de flexibilidad, las diferencias en baja y media son moderadas, pero en flexibilidad alta la variabilidad del GA se incrementa de manera notable (12,2 %), mientras que la Mateheurística independiente y la dependiente mantienen desviaciones cercanas a 3,7 % y 2,9 %, respectivamente. Este comportamiento indica que, en instancias altamente flexibles, el operador MILP no solo alcanza MRE promedio significativamente menores, sino que además mantiene una estabilidad mucho mayor que el GA, reforzando la conclusión de que el operador MILP dependiente mejora de forma robusta la calidad y consistencia de las soluciones frente al GA en este subconjunto de problemas.

Con respecto a los tiempos promedio, los resultados se muestran en la Tabla 21. El GA es el que obtiene los tiempos considerablemente más bajos bajos, dado que no incorpora operadores de alto costo computacional. Sin embargo, tal como se mencionó al final de la

sección anterior, el GA carece de mecanismos de búsqueda adicionales, los cuales sí poseen los métodos independiente y dependiente. Aunque estos últimos incrementan considerablemente el tiempo de cómputo, es posible adoptar estrategias computacionales que mejoren su rendimiento en términos de tiempo, como la paralelización de la resolución de los lotes o mejorar el empaquetado de los lotes al ejecutar cada parte.

Tabla 21: Promedio de tiempos (Segundos)

Total Instancias	Tamaño	Rango Actividades	MILP	GA	Independiente	Dependiente
24	Bajo	(0, 36)	412	22	90	550
24	Mediana	(37, 57)	491	23	120	664
23	Alta	(58, 80)	302	16	185	512

En síntesis, el análisis de este subconjunto de 71 instancias muestra que la Mateheurística dependiente es el método con mejor calidad de solución, alcanzando los menores valores de MRE en prácticamente todos los rangos de tamaño y flexibilidad, con una ventaja que se acentúa en las instancias de mayor complejidad. Esta superioridad, sin embargo, tiene como contrapartida el mayor costo computacional entre los cuatro métodos evaluados, superando incluso los tiempos del MILP. Se evidencia así un compromiso entre calidad y tiempo de cómputo: el GA ofrece la mayor rapidez con soluciones aceptables, la Mateheurística independiente constituye un punto intermedio con mejoras a un costo moderado, y la dependiente maximiza la calidad a expensas del tiempo. Cabe señalar que estas conclusiones se limitan a instancias de hasta 80 actividades, por lo que extender el enfoque dependiente a instancias de mayor tamaño, mediante estrategias que mitiguen el crecimiento del tiempo de carga, se plantea como una línea de trabajo futuro.

6. Conclusiones

En esta investigación se abordó el *Flexible Job Shop Scheduling Problem* con tiempos de configuración dependientes de la secuencia (FJSSP-SDST) con el objetivo de minimizar el tiempo de ejecución de todas las actividades (*makespan*). Este problema, catalogado como NP-Hard, combina la dificultad de la secuenciación de operaciones con la asignación de máquinas, a lo que se suma la incorporación de los SDST, tiempos que la literatura reconoce como uno de los aspectos más complejos de los problemas de programación, tanto por su impacto en la calidad de los programas como por la complejidad que añaden a los métodos de solución. Si bien los algoritmos genéticos se han consolidado como una de las herramientas más utilizadas para el FJSSP, diversos estudios han evidenciado que son propensos a la convergencia prematura y a la pérdida de diversidad poblacional, lo que limita su capacidad para explorar adecuadamente el espacio de búsqueda en instancias de alta complejidad. A pesar de la relevancia práctica de los SDST en industrias con alta variedad de productos, la revisión de la literatura mostró que su integración explícita en el FJSSP ha sido comparativamente poco desarrollada y que el uso del MILP dentro de algoritmos evolutivos continúa siendo muy limitado, sin que se identificara una Mateheurística que integre un GA con un modelo MILP por lotes para abordar específicamente este problema.

La principal contribución de esta investigación es el diseño de una Mateheurística que combina un algoritmo genético con un modelo MILP, el vínculo entre metodologías se realiza

con la programación de actividades por lotes de operaciones que entrega el GA. El operador MILP propuesto consta de dos estrategias de integración: una independiente, donde cada lote se resuelve por separado y luego se consolida la solución, y una dependiente, donde las soluciones de los lotes anteriores se fijan mediante restricciones. Los resultados sobre 276 instancias mostraron que tanto el GA como la Mateheurística independiente superan ampliamente al MILP, cuya capacidad se limita a instancias de hasta 36 actividades. Entre los métodos propuestos, la Mateheurística independiente obtiene un mejor desempeño dentro de las alternativas evaluadas y bajo las condiciones experimentales definidas al GA especialmente en instancias de alta flexibilidad (de mayor complejidad), con una mejora del 7,3 % en el subconjunto que combina ambas características. Por su parte, la Mateheurística dependiente, evaluada en instancias de hasta 80 actividades, obtuvo los mejores valores de MRE, dentro de las condiciones experimentales definidas(datos artificiales y adaptados al FJSSP-SDST), en casi todos los rangos analizados, con mejoras de hasta el 69,2 % frente al GA y del 17,2 % frente al método independiente en flexibilidades altas, aunque a un costo computacional superior. Estas mejoras se obtuvieron con una estrategía de aplicación limitada del operador MILP: al ejecutarse cada 100 generaciones sobre el 10 % de la población, con un máximo de 1000 generaciones, es decir, el operador se aplicó únicamente 10 veces por ejecución y en cada una sobre una fracción reducida de los individuos. Si bien una aplicación más intensiva no garantiza necesariamente mejores resultados, estos valores conservadores dejan margen para explorar configuraciones con mayor participación del operador dentro del ciclo evolutivo del GA, ajustando su frecuencia o el porcentaje de individuos intervenidos, lo que podría potenciar el aporte del componente exacto sin comprometer el equilibrio de la búsqueda. Estos resultados evidencian un trade-off entre calidad y tiempo de cómputo. Adicionalmente, la investigación aporta un algoritmo genético calibrado mediante un diseño de experimentos de Taguchi, con operadores seleccionados a partir de evaluaciones comparativas en distintos niveles de flexibilidad, y un conjunto de 276 casos de prueba del FJSSP adaptados con SDST bajo el teorema de desigualdad triangular, disponibles públicamente para futuras investigaciones.

Como trabajo futuro, la línea más inmediata consiste en extender el operador dependiente a instancias de mayor tamaño, mitigando el crecimiento del tiempo de carga mediante estrategias como la mejora del empaquetado del modelo en cada ejecución, e incluso probar metodologías se solución basadas en descomposición. En el caso del método independiente, su rendimiento puede mejorarse mediante estrategias de paralelización de la resolución de los lotes, dado que estos no dependen entre sí. También resulta pertinente evaluar tamaños de lote adaptativos, que varíen según el tamaño y la flexibilidad de la instancia, en lugar del valor fijo adoptado en este estudio. Desde el punto de vista del problema, la Mateheurística propuesta puede extenderse a objetivos adicionales como la tardanza total o el consumo energético, así como a variantes con incertidumbre en los tiempos de procesamiento. Finalmente, la validación del método sobre casos industriales reales permitiría contrastar los resultados obtenidos con datos generados artificialmente y ajustar la calibración de los hiperparámetros a entornos de producción específicos.

Agradecimientos

Este trabajo contó con el apoyo de MinCiencias Colombia, bajo la subvención SGR BPIN 2022000100068.

Declaración de divulgación

Los autores no declararon ningún conflicto de intereses potencial.

Declaración de disponibilidad de datos

Las bases de datos de los casos de prueba de este estudio están disponibles en https://github.com/dmorill/FJSSP_SDST_Instances.

References

Adams, J., Balas, E., & Zawack, D. (1988). The Shifting Bottleneck Procedure for Job Shop Scheduling. *Management Science*, *34*(3), 391-401. https://doi.org/10.1287/mnsc.34.3.391

Allahverdi, A. (2015). The third comprehensive survey on scheduling problems with setup times/costs. *European Journal of Operational Research*, *246*(2), 345-378. https://doi.org/10.1016/j.ejor.2015.04.004

Allahverdi, A., & Allahverdi, M. (2026). The fourth comprehensive review of scheduling problems with setup times. *European Journal of Operational Research*. https://doi.org/https://doi.org/10.1016/j.ejor.2026.03.041

Allahverdi, A., Ng, C., Cheng, T., & Kovalyov, M. Y. (2008). A survey of scheduling problems with setup times or costs. *European Journal of Operational Research*, *187*(3), 985-1032. https://doi.org/10.1016/j.ejor.2006.06.060

Amjad, M. K., Butt, S. I., Kousar, R., Ahmad, R., Agha, M. H., Faping, Z., Anjum, N., & Asgher, U. (2018). Recent Research Trends in Genetic Algorithm Based Flexible Job Shop Scheduling Problems. *Mathematical Problems in Engineering*, *2018*. https://doi.org/10.1155/2018/9270802

Applegate, D., & Cook, W. (1991). A Computational Study of the Job-Shop Scheduling Problem. *ORSA Journal on Computing*, *3*(2), 149-156. https://doi.org/10.1287/ijoc.3.2.149

Artigues, C., & Feillet, D. (2008). A branch and bound method for the job-shop problem with sequence-dependent setup times. *Annals of Operations Research*, *159*(1), 135-159. https://doi.org/10.1007/s10479-007-0283-0

Behnke, D., & Geiger, M. J. (2012). *Test Instances for the Flexible Job Shop Scheduling Problem with Work Centers* (Working paper). Universitätsbibliothek der HSU/UniBw H. https://doi.org/10.24405/436

Brandimarte, P. (1993). Routing and scheduling in a flexible job shop by tabu search. *Annals of Operations Research*, *41*(3), 157-183. https://doi.org/10.1007/BF02023073

Brucker, P., & Schlie, R. (1990). Job-shop scheduling with multi-purpose machines. *Computing*, *45*(4), 369-375. https://doi.org/10.1007/BF02238804

Carlier, J. (1978). Scheduling with Disjunctive Constraints.; [ORDONNANCEMENTS A CONTRAINTES DISJONCTIVES.] *RAIRO Recherche Operationnelle*, *12*(4), 333-350. https://doi.org/10.1051/ro/1978120403331

Chambers, J. B., & Barnes, J. W. (1996). *Flexible Job Shop Scheduling by Tabu Search* (Technical Report N.º ORP96-09). Graduate Program in Operations Research y Industrial Engineering, The University of Texas at Austin. Austin, TX.

Cheng, W., Zhang, C., Meng, L., Ren, Y., & Ullah, S. (2025). Collaborative multi-CP model and meta-feedback learning-assisted matheuristic for solving the flexible job shop scheduling problem with sequence-dependent setup times. *Swarm and Evolutionary Computation*, *99*. https://doi.org/10.1016/j.swevo.2025.102173

Cheng, W., Zhang, C., Meng, L., Zhang, B., & Sang, H. (2026). A matheuristic and imitation learning-driven evolutionary algorithm for the flexible job shop scheduling benchmark problem with discrete operation sequence flexibility. *International Journal of Production Research*, *64*(9), 3564-3587. https://doi.org/10.1080/00207543.2025.2612155

Choi, I.-C., & Choi, D.-S. (2002). A local search algorithm for jobshop scheduling problems with alternative operations and sequence-dependent setups. *Computers & Industrial Engineering*, *42*(1), 43-58. https://doi.org/https://doi.org/10.1016/S0360-8352(02)00002-5

Dauzère-Pérès, S., Ding, J., Shen, L., & Tamssaouet, K. (2024). The flexible job shop scheduling problem: A review. *European Journal of Operational Research*, *314*(2), 409-432. https://doi.org/https://doi.org/10.1016/j.ejor.2023.05.017

Dauzère-pérès, S., & Paulli, J. (1997). An integrated approach for modeling and solving the general multiprocessor job-shop scheduling problem using tabu search. *Annals of Operations Research*, *70*, 281-306. https://doi.org/10.1023/a:1018930406487

Defersha, F. M., & Chen, M. (2010). A parallel genetic algorithm for a flexible job-shop scheduling problem with sequence dependent setups. *INTERNATIONAL JOURNAL OF ADVANCED MANUFACTURING TECHNOLOGY*, *49*(1-4), 263-279. https://doi.org/10.1007/s00170-009-2388-x

Fan, J., Zhang, C., Shen, W., & Gao, L. (2023). A matheuristic for flexible job shop scheduling problem with lot-streaming and machine reconfigurations. *International Journal of Production Research*, *61*(19), 6565-6588. https://doi.org/10.1080/00207543.2022.2135629

Fan, J., Zhang, C., Tian, S., Shen, W., & Gao, L. (2024). Flexible job-shop scheduling problem with variable lot-sizing: An early release policy-based matheuristic. *Computers and Industrial Engineering*, *193*. https://doi.org/10.1016/j.cie.2024.110290

Fan, J., Zhang, C., Yang, F., Shen, W., & Gao, L. (2024). A matheuristic with re-lot-sizing strategies for flexible job-shop rescheduling problem with lot-streaming and machine reconfigurations. *European Journal of Operational Research*, *319*(3), 747-762. https://doi.org/10.1016/j.ejor.2024.07.030

Fattahi, P., Saidi Mehrabad, M., & Jolai, F. (2007). Mathematical Modeling and Heuristic Approaches to Flexible Job Shop Scheduling Problems. *Journal of Intelligent Manufacturing*, *18*(3), 331-342. https://doi.org/10.1007/s10845-007-0026-8

Fisher, H., & Thompson, G. L. (1963). Probabilistic Learning Combinations of Local Job-Shop Scheduling Rules. En J. F. Muth & G. L. Thompson (Eds.), *Industrial Scheduling* (pp. 225-251). Prentice-Hall.

Framinan, J. M., Leisten, R., & Ruiz, R. (2014). *Manufacturing Scheduling Systems: An Integrated View on Models, Methods and Tools*. Springer. https://doi.org/10.1007/978-1-4471-6272-8

Gao, J., Gen, M., & Sun, L. (2006). Scheduling jobs and maintenances in flexible job shop with a hybrid genetic algorithm. *Journal of Intelligent Manufacturing*, *17*(4), 493-507. https://doi.org/10.1007/s10845-005-0021-x

Gao, J., Sun, L., & Gen, M. (2008). A hybrid genetic and variable neighborhood descent algorithm for flexible job shop scheduling problems. *Computers and Operations Research*, *35*(9), 2892-2907. https://doi.org/10.1016/j.cor.2007.01.001

Gao, K., Cao, Z., Zhang, L., Chen, Z., Han, Y., & Pan, Q. (2019). A review on swarm intelligence and evolutionary algorithms for solving flexible job shop scheduling problems. *IEEE/CAA Journal of Automatica Sinica*, *6*(4), 904-916. https://doi.org/10.1109/JAS.2019.1911540

Ghasemi, A., Farajzadeh, F., Heavey, C., Fowler, J., & Papadopoulos, C. T. (2024). Simulation optimization applied to production scheduling in the era of industry 4.0: A review and future roadmap. *Journal of Industrial Information Integration*, *39*, 100599. https://doi.org/10.1016/j.jii.2024.100599

Gnanavelbabu, A., Caldeira, R. H., & Vaidyanathan, T. (2021). A simulation-based modified backtracking search algorithm for multi-objective stochastic flexible job shop scheduling problem with worker flexibility. *Applied Soft Computing*, *113*, 107960. https://doi.org/https://doi.org/10.1016/j.asoc.2021.107960

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.

Huang, X., Chen, S., Zhou, T., & Sun, Y. (2022). Survey on genetic algorithms for solving flexible job-shop scheduling problem. *Jisuanji Jicheng Zhizao Xitong/Computer Integrated Manufacturing Systems, CIMS*, *28*(2), 536-551. https://doi.org/10.13196/j.cims.2022.02.018

Hurink, J., Jurisch, B., & Thole, M. (1994). Tabu search for the job-shop scheduling problem with multi-purpose machines. *OR Spektrum*, *15*(4), 205-215. https://doi.org/10.1007/BF01719451

Kacem, I., Hammadi, S., & Borne, P. (2002). Pareto-optimality approach for flexible job-shop scheduling problems: Hybridization of evolutionary algorithms and fuzzy logic. *Mathematics and Computers in Simulation*, *60*(3-5), 245-276. https://doi.org/10.1016/S0378-4754(02)00019-8

Kechagias, J. D., Aslani, K.-E., Fountas, N. A., Vaxevanidis, N. M., & Manolakos, D. E. (2020). A comparative investigation of Taguchi and full factorial design for machinability prediction in turning of a titanium alloy. *Measurement*, *151*, 107213. https://doi.org/https://doi.org/10.1016/j.measurement.2019.107213

Lawrence, S. (1984). *Resource Constrained Project Scheduling: An Experimental Investigation of Heuristic Scheduling Techniques* [Technical Report]. Graduate School of Industrial Administration, Carnegie-Mellon University.

Li, X., Guo, X., Tang, H., Wu, R., Wang, L., Pang, S., Liu, Z., Xu, W., & Li, X. (2022). Survey of integrated flexible job shop scheduling problems. *Computers and Industrial Engineering*, *174*. https://doi.org/10.1016/j.cie.2022.108786

Liu, Z., Wang, J., Zhang, C., Chu, H., Ding, G., & Zhang, L. (2021). A hybrid genetic-particle swarm algorithm based on multilevel neighbourhood structure for flexible job shop scheduling problem. *Computers and Operations Research*, *135*. https://doi.org/10.1016/j.cor.2021.105431

Loveland, J. L., Monkman, S. K., & Morrice, D. J. (2007). Dell uses a new production-scheduling algorithm to accommodate increased product variety. *Interfaces*, *37*(3), 209-219. https://doi.org/10.1287/inte.1060.0264

Maghsoodloo, S., Ozdemir, G., Jordan, V., & Huang, C.-H. (2004). Strengths and limitations of taguchi's contributions to quality, manufacturing, and process engineering. *Journal of Manufacturing Systems*, *23*(2), 73-126. https://doi.org/https://doi.org/10.1016/S0278-6125(05)00004-X

Meng, L., Cheng, W., Zhang, B., Zou, W., Fang, W., & Duan, P. (2023). An Improved Genetic Algorithm for Solving the Multi-AGV Flexible Job Shop Scheduling Problem. *Sensors*, *23*(8). https://doi.org/10.3390/s23083815

Panwalkar, S. S., Dudek, R. A., & Smith, M. L. (1973). Sequencing Research and the Industrial Scheduling Problem. En S. E. Elmaghraby (Ed.), *Symposium on the Theory of Scheduling and its Applications* (pp. 29-38). University of Texas at Austin.

Pinedo, M. L. (2022). *Scheduling: Theory, Algorithms, and Systems* (6.ª ed.). Springer. https://doi.org/10.1007/978-3-031-05921-6

Ripon, K. S. N., Siddique, N., & Torresen, J. (2011). Improved precedence preservation crossover for multi-objective job shop scheduling problem. *Evolving Systems*, *2*(2), 119-129. https://doi.org/10.1007/s12530-010-9022-x

Sahli, A., Behiri, W., Belmokhtar-Berraf, S., & Chu, C. (2022). An effective and robust genetic algorithm for urban freight transport scheduling using passenger rail network. *Computers and Industrial Engineering*, *173*. https://doi.org/10.1016/j.cie.2022.108645

Saidi-Mehrabad, M., & Fattahi, P. (2007). Flexible job shop scheduling with tabu search algorithms. *International Journal of Advanced Manufacturing Technology*, *32*(5-6), 563-570. https://doi.org/10.1007/s00170-005-0375-4

Sergienko, I., Hulianytskyi, L., & Sirenko, S. (2009). Classification of applied methods of combinatorial optimization. *Cybernetics and Systems Analysis*, *45*(5), 732-741. https://doi.org/10.1007/s10559-009-9134-0

Shen, L., Dauzere-Peres, S., & Neufeld, J. S. (2018). Solving the flexible job shop scheduling problem with sequence-dependen setup times. *European Journal of Operational Research*, *265*(2), 503-516. https://doi.org/10.1016/j.ejor.2017.08.021

Sun, K., Zheng, D., Song, H., Cheng, Z., Lang, X., Yuan, W., & Wang, J. (2023). Hybrid genetic algorithm with variable neighborhood search for flexible job shop scheduling problem in a machining system. *Expert Systems with Applications*, *215*, 119359. https://doi.org/https://doi.org/10.1016/j.eswa.2022.119359

Trovinger, S. C., & Bohn, R. E. (2005). Setup time reduction for electronics assembly: Combining simple (SMED) and IT-based methods. *Production and Operations Management*, *14*(2), 205-217. https://doi.org/10.1111/j.1937-5956.2005.tb00019.x

Tutumlu, B., & Saraç, T. (2023). A MIP model and a hybrid genetic algorithm for flexible job-shop scheduling problem with job-splitting. *Computers and Operations Research*, *155*. https://doi.org/10.1016/j.cor.2023.106222

Wang, Y., & Zhu, Q. (2021). A Hybrid Genetic Algorithm for Flexible Job Shop Scheduling Problem With Sequence-Dependent Setup Times and Job Lag Times. *IEEE ACCESS*, *9*, 104864-104873. https://doi.org/10.1109/ACCESS.2021.3096007

Winklehner, P., & Hauder, V. A. (2022). Flexible job-shop scheduling with release dates, deadlines and sequence dependent setup times: a real-world case [3rd International Conference

on Industry 4.0 and Smart Manufacturing]. *Procedia Computer Science*, *200*, 1654-1663. https://doi.org/https://doi.org/10.1016/j.procs.2022.01.366

Xia, X., Jiang, S., Zhou, N., Li, X., & Wang, L. (2019). Genetic algorithm hyper-parameter optimization using taguchi design for groundwater pollution source identification. *Water Science and Technology: Water Supply*, *19*(1), 137-146. https://doi.org/10.2166/ws.2018.059

Xie, J., Li, X., Gao, L., & Gui, L. (2023). A hybrid genetic tabu search algorithm for distributed flexible job shop scheduling problems. *Journal of Manufacturing Systems*, *71*, 82-94. https://doi.org/10.1016/j.jmsy.2023.09.002

Xiong, H., Shi, S., Ren, D., & Hu, J. (2022). A survey of job shop scheduling problem: The types and models. *Computers & Operations Research*, *142*, 105731. https://doi.org/https://doi.org/10.1016/j.cor.2022.105731

Ying, K.-C., Pourhejazy, P., & Chen, K.-Y. (2026). Revisiting the evolution of genetic algorithms in scheduling. *Applied Soft Computing*, *201*, 115602. https://doi.org/https://doi.org/10.1016/j.asoc.2026.115602

Zhao, Z., Zhou, H., Eun, J., & Zhao, L. (2026). A matheuristic for lot-streaming scheduling in a flexible job shop with variable sublots and intermingling settings [Cited by: 6]. *International Journal of Production Research*, *64*(7), 2397-2426. https://doi.org/10.1080/00207543.2025.2578700

Zhao, Z., Chen, X., An, Y., Li, Y., & Gao, K. (2023). A property-based hybrid genetic algorithm and tabu search for solving order acceptance and scheduling problem with trapezoidal penalty membership function. *Expert Systems with Applications*, *218*, 119598. https://doi.org/https://doi.org/10.1016/j.eswa.2023.119598