Solución al problema de “Flow-shop” permutado, minimizando los costos de inventario en proceso y penalización debido al retraso de lotes a través de un Algoritmo Genético.

Diego Alberto Rico Castillo

Hellen Geneth Rodríguez López

Trabajo de grado para optar el título de Ingeniero Industrial

Director:

Edwin Alberto Garavito Hernández

Magíster en Ingeniería Industrial

Codirector:

Laura Yeraldín Escobar Rodríguez

Ingeniera Industrial

Universidad Industrial de Santander

Facultad de Ingenierías Fisicomecánicas

Escuela de Estudios Industriales y Empresariales

Bucaramanga, Santander

2019

DEDICATORIA

A mi madre Rosa por apoyarme incondicionalmente y brindarme sus valiosos consejos;

A Diana ♥ por iluminar mi vida, por estar siempre a mi lado, por corregirme con sabiduría;

A mi padre Álvaro y hermanas, Laura y Sandra, por confiar siempre en mí;

A David, Sofi y Andrés, mi motor de vida.

Diego Alberto Rico Castillo.

A mis padres y familia, que desde el inicio de este viaje me apoyaron y acompañaron;

A mis docentes y compañeros, que hicieron parte del aprendizaje y crecimiento;

A mi pareja, por ser quien me animaba y fortalecía en los momentos más difíciles;

A cada una de las personas que de una u otra manera hicieron esto posible;

Y a los vengadores por salvar una vez más al universo.

Hellen Geneth Rodríguez López.

AGRADECIMIENTOS

A nuestro director Edwin Garavito y a nuestra amiga y codirectora Laura, por brindarnos la oportunidad de trabajar a su lado, por confiar en nosotros, por su paciencia, entrega y dedicación, por estar siempre dispuestos a ayudarnos.

A nuestras familias por su apoyo, comprensión y confianza depositada en nosotros, por esculpir la mejor versión de nosotros y enseñarnos el deber ser de las cosas.

A todos y cada uno de nuestros amigos y compañeros quienes de una u otra forma contribuyeron en nuestro desarrollo integral.

Tabla de contenido

Listado de tablas

Listado de Figuras

Lista de Apéndices

Nota: Los apéndices están adjuntos en el CD y puede visualizarlos en la base de datos de la biblioteca UIS.

Resumen

Título del proyecto[1]**:** "SOLUCIÓN AL PROBLEMA DE "FLOW-SHOP" PERMUTADO, MINIMIZANDO LOS COSTOS DE INVENTARIO EN PROCESO Y PENALIZACIÓN DEBIDO AL RETRASO DE LOTES A TRAVÉS DE UN ALGORITMO GENÉTICO".

Autores: Diego Alberto Rico Castillo, Hellen Geneth Rodríguez López[2]

Palabras clave: flow shop scheduling, work in process, tardiness, costs, genetic algorithm.

Descripción: Los problemas de programación en sistemas tipo "Flow-shop" pertenecen a una categoría esencial de los problemas de programación, en la que *n* trabajos que siguen un orden de procesamiento similar deben procesarse en *m* máquinas. La secuenciación apropiada del grupo de trabajos permite cumplir el objetivo planteado en la programación, que bien puede estar enfocado en los plazos de entrega, los costos o el número de unidades producidas. Considerando lo anterior, en la presente investigación se desarrolla un algoritmo genético (Genectic Algorithm, GA), con el fin de dar solución al problema de "Flow-Shop Scheduling" permutado, minimizando el costo total por unidad de tiempo de programación; se aborda el problema contemplando el costo de penalización por tardanza en la entrega, así como los costos por el mantenimiento de inventario y a su vez considerando los tiempos de alistamiento de las máquinas, basados en los planteamiento realizados por Mishra & Shrivastava (2018). Si bien los resultados experimentales, en términos generales, mostraron que los métodos de solución propuestos arrojaron soluciones acertadas al problema, se recomienda la realización de investigaciones adicionales, las cuales deben incluir otras variantes del problema, especialmente enfocados en aplicaciones del mundo real como, por ejemplo, la consideración de posibles fuentes de incertidumbre o la inclusión de nuevas restricciones.

---

[1] Trabajo de grado

[2] Facultad de Ingenierías Fisicomecánicas. Escuela de Estudios Industriales y Empresariales. Director: MSc. Edwin Alberto Garavito Hernández, Codirector: Ing. Laura Yeraldin Escobar Rodríguez.

Abstract

Title[3]**:** "A GENETIC ALGORITHM FOR PERMUTATION FLOW SHOP SCHEDULING TO MINIMIZE THE SUM OF INVENTORY HOLDING AND BATCH DELAY COSTS"

Authors: Diego Alberto Rico Castillo, Hellen Geneth Rodríguez López[4]

Keywords: flow shop scheduling, work in process, tardiness, costs, genetic algorithm.

Description: The scheduling problems in systems type "Flow-shop" belong to an essential category of programming problems, in which *n* works that follow a similar processing order must be processed in *m* machines. The appropriate sequencing of the group of works allows to get the objective set in the programming, which may well be focused on delivery times, costs or the number of units produced. Considering the above, in the present investigation a genetic algorithm is developed (Genectic Algorithm, GA); the objective is to obtain an optimum production schedule which minimizes the expected total cost per unit time of scheduling; The problem is addressed by considering an integrated cost model for flow shop scheduling in setting with penalties for tardiness in delivering customer orders, as well as costs for holding both delayed goods and WIP inventory considering the machine set up times, based on the approach made by Mishra & Shrivastava (2018). Although the experimental results, in general terms, showed that the proposed solution methods yielded correct solutions to the problem, it is recommended to carry out additional investigations, which should include other variants of the problem, especially focused on real-world applications such as, the consideration of possible sources of uncertainty or the inclusion of new restrictions.

[3] Bachelor Thesis

[4] Faculty of Physicomechanical Engineering, Industrial and Business School. Director: MSc. Edwin Alberto Garavito Hernández, Co-director: Laura Yeraldin Escobar Rodríguez

Introducción

Los sistemas productivos tipo “flow-shop” permutado (PFSP) son entornos de manufactura en los cuales un conjunto de trabajos, ordenados en una determinada secuencia, son procesados a través de una serie de estaciones o maquinas; este sistema de producción es común encontrarlo aplicado a muchos campos industriales como la industria electrónica, de papel, textil, entre otras; por consiguiente el problema de programación es relevante, debido a su sólida formación en ingeniería y relevancia practica para la industria (Yu, Semeraro, & Matta, 2018).

Ahora bien, el problema de la programación PFSP es un proceso de toma de decisiones mediante el cual se asignan tareas a recursos de producción disponibles en periodos de tiempo determinados, con el objetivo de optimizar uno o más objetivos, como por ejemplo el makespan y/o el rendimiento relacionados con la fecha de vencimiento, no obstante, para el presente trabajo se plantea minimizar un objetivo asociado a los costos.

El problema es conocido por ser NP-hard, de ahí que se hayan desarrollado muchos métodos heurísticos para proporcionar buenas soluciones en un tiempo razonablemente corto, en consecuencia se plantea el uso de un algoritmo GA para da solución al PFSP pues de acuerdo con Chen, Pan, & Lin (2008), el GA ha sido ampliamente utilizado para resolver problemas clásicos de FSP y ha tenido un buen desempeño. Adicionalmente, se emplea el uso de métodos exactos para comparar las soluciones optimas generadas por la técnica de programación exacta con las soluciones casi optimas proveídas por GA.

Ahora bien, cabe aclarar que en este documento se contempla un PFSP para el cual se pretende minimizar el costo de penalización por tardanza en la entrega de pedidos de los clientes, así como los costos por la tenencia de bienes demorados e inventario tomando en cuenta los tiempos de alistamiento de la máquina, conforme a sugerencias planteadas por Mishra & Shrivastava (2018).

En términos generales el documento se encuentra organizado de la siguiente manera: Los capítulos 1, 2 y 3, proporcionan las especificaciones generales que enmarcan el desarrollo del presente proyecto. En el capítulo 4, se presenta una breve revisión de la literatura relevante. En el capítulo 5, complementando el marco de referencia se presenta un marco teórico. En el capítulo 6 se puede apreciar la declaración en detalle del problema objeto de estudio.

Adicionalmente, el capítulo 7 explica el modelo matemático para la optimización de la suma de la retención de inventario y los costos de demora de lotes en el “flow-shop” permutado. A su vez, el capítulo 8 está dedicado al procedimiento y los pasos involucrados en la solución del problema por parte del algoritmo GA. En el capítulo 9 se aborda el problema mediante experimentación estadística se evalúa la estimación de los parámetros de estos algoritmos. El capítulo 10 muestra los resultados computacionales y las discusiones y, finalmente, el capítulo 11 ofrece conclusiones y extensiones futuras.

Finalmente, el cumplimiento de los objetivos planteados para la presente investigación se presenta la Tabla 1 como se puede apreciar a continuación:

Tabla 1.

Cumplimiento de Objetivos

Objetivos Específicos	***Cumplimiento***
Realizar una revisión de la literatura que aborde métodos y soluciones al problema de "Flow-Shop" permutado.	Capítulo 4, 5
Definir el modelo de programación matemática para el problema y solucionarlo haciendo uso de técnicas exactas.	Capítulo 6
Validar el modelo matemático a través de instancias adaptadas de la literatura.	Capítulo 7
Implementar el algoritmo genético en un software matemático y evaluar su eficiencia mediante la comparación con instancias halladas en la literatura o por medio de un análisis estadístico de los resultados.	Capítulo 8, 9
Elaborar un artículo de carácter publicable con base en los resultados obtenidos de la investigación realizada.	Apéndice E

1. Planteamiento del problema

La programación óptima de los trabajos tiene un impacto significativo en la eficiencia de las operaciones de los sistemas de producción en la actualidad; es un proceso de toma de decisiones en el que se asignan recursos limitados a tareas durante períodos de tiempo determinados para optimizar uno o más objetivos (Pinedo, 2012). Análogamente, en un sistema productivo tipo "flow-shop" donde el orden de procesamiento de los trabajos en las máquinas debe seguir siendo el mismo en cada estación, y las máquinas normalmente se disponen en serie, la correcta secuenciación de los trabajos juega un papel relevante en el adecuado flujo de producción y la reducción de costos.

La literatura considera una amplia variedad de problemas y objetivos de programación; un objetivo importante que no ha recibido mucha atención es el costo del mantenimiento del inventario en proceso (WIP). El nivel de inventario en proceso a menudo se considera como una de las medidas para la eficiencia de la producción; si bien es difícil operar líneas de producción sin un inventario de éste tipo, la mayoría de las compañías intentan minimizarlo (Yang, Society, & Engineering, 2018). Aunque, hay otro objetivo importante que también está recibiendo más atención en la literatura hoy en día debido a su aplicabilidad a las industrias, se denomina la entrega a tiempo de los trabajos.

En el entorno empresarial extremadamente competitivo de hoy, cumplir con las fechas de entrega juegan un papel crucial en los problemas de programación, entre otras cosas, debido a que las entregas tardías causan penalidades que las empresas deben asumir, en lo que radica la

importancia de la adecuada programación para reducir estos costos adicionales. Una forma de lograrlo es no solo mediante la entrega de todos los trabajos a tiempo, sino también con la satisfacción del cliente (Xiao, Yuan, Zhang, & Konak, 2015).

Por tal razón, en esta investigación se plantea un modelo de costo para la programación en un entorno "flow-shop" asociado con el establecimiento de multas por tardanzas en la entrega de los pedidos de los clientes, así como los costos de mantener tanto los bienes demorados como el inventario de WIP teniendo en cuenta los tiempos de configuración de la máquina.

Al considerar los factores anteriormente mencionados, se asegura que el problema propuesto se investigue bajo configuraciones que generalmente se observan en situaciones reales en la industria. Ahora bien, la complejidad de los problemas de programación de tareas es bien conocida, por ello se ha convertido en el problema más popular de programación de máquinas NP-hard con una extensa ingeniería (Mishra & Shrivastava, 2018); por tanto, se hace necesario el uso de meta-heurísticas que obtengan soluciones factibles en tiempos computacionales razonables.

Para el presente trabajo se plantea el uso de un Algoritmo Genético, método que salió a la luz a finales de los sesenta cuando John Holland, un investigador de la Universidad de Michigan se dio cuenta en primer lugar de la potencia de estos cálculos y de las oportunidades que ofrecían a diferentes campos de investigación. Los algoritmos genéticos son métodos adaptativos ampliamente usados para resolver problemas de búsqueda y optimización, son capaces de ir creando respuestas a partir de la evolución de soluciones base.

En cuanto a la pertinencia del presente trabajo, la consideración de los atributos para tener en cuenta obedece a sugerencias planteadas en la literatura por autores con una notoria trayectoria en el estudio del “flow shop scheduling”. En lo que respecta a futuros estudios del tema, según diversos autores, se deben encaminar a complementar el panorama investigativo conforme se incorporen atributos que acerquen el problema a cuestiones realistas.

Además, el desarrollo del estudio sirve como base para el planteamiento de nuevas investigaciones futuras por líneas similares y/o complementarias y también como marco de referencia metodológico para la obtención de buenos resultados en otras investigaciones.

2. Justificación del proyecto

La programación de la producción o también llamados programas de operaciones, desempeñan un papel de vital importancia en la planificación y operación de un sistema productivo; dichos programas son planes a corto plazo elaborados con el fin de implementar el plan maestro de producción. Éstos se centran en encontrar la mejor forma de usar la capacidad existente, tomando en cuenta las restricciones técnicas de producción (Krajewski, Ritzman, & Malhotra, 2009).

Adoptar un apropiado sistema para la programación de la producción tiene un impacto significativo en términos de la reducción de costos, del inventario en proceso y del adecuado flujo de producción; dicho de otro modo, si los programas de producción no se planean cuidadosamente, es posible que empiecen a formarse filas de espera por la acumulación de inventario, no haya un flujo continuo y, por consiguiente, se generen retrasos en las entregas; incumpliendo los plazos pactados con los clientes y con ello, el pago de penalizaciones por retrasos.

La programación puede resultar muy compleja dependiendo del tipo de configuración del sistema productivo (taller de trabajo, celda de manufactura, línea de ensamble o producción de flujo continuo o “flow-shop”) (Render & Heizer, 2009). En un sistema “flow-shop” se parte de la suposición de que existe un conjunto conocido de trabajos; la situación problema radica en la determinación de una secuencia apropiada de producción para el grupo de trabajos, a través de una serie de máquinas, satisfaciendo un conjunto de restricciones conforme a los objetivos planteados inicialmente en el modelo matemático.

Si bien la mayoría de la literatura sobre programación de sistemas tipo “flow-shop” ha enfatizado los criterios de rendimiento basados en el tiempo, como el tiempo medio de flujo, la tardanza y el “makespan”, el objetivo principal de la administración debería ser la maximización de la rentabilidad o en su defecto la minimización de los costos (Scudder & Hoffmann, 1987). Éstos deben considerarse explícitamente mientras se toman las decisiones de programación para aumentar la productividad, eliminar desperdicios, mejorar la utilización de los recursos y cumplir los plazos.

Actualmente hay varios tipos de costos asociados con los FSP. Los costos principales incluyen el costo de producción, el costo de mantenimiento del inventario, el costo de instalación, el costo de entregas tempranas, el costo de la tardanza, el costo de entrega, etc. (Bülbül, Kaminsky, & Yano, 2004).

El presente trabajo busca dar solución al problema del “Flow-Shop” Permutado minimizando el costo total por unidad de tiempo de programación. Se aborda el problema contemplando el costo de penalización por tardanza en la entrega de pedidos de los clientes, así como los costos por la tenencia de bienes demorados e inventario (WIP) considerando los tiempos de configuración o alistamiento de la máquina, teniendo en cuenta las sugerencias planteadas por Mishra & Shrivastava (2018).

De acuerdo con Mishra & Shrivastava (2018), el problema de secuenciación en entornos “flow-shop” se ha convertido en el problema más popular de programación de máquinas, su considerable dificultad para hallar soluciones óptimas en tiempos de respuesta razonables lo clasifican dentro

de la categoría NP-hard; por tal motivo, en el presente trabajo se propone la aplicación de una metaheurística para resolverlo. Se plantea un Algoritmo Genético (GA), el cual es un método adaptativo, generalmente usado en problemas de búsqueda y optimización de parámetros, basado en la reproducción sexual y en el principio de supervivencia del más apto (Fogel, 2005).

3. Objetivos

3.1. Objetivo General

Desarrollar una metaheurística basada en el algoritmo genético (Genectic Algorithm, GA) con el fin de dar solución al problema del “Flow Shop” permutado minimizando el costo total por unidad de tiempo de programación.

3.2. Objetivos Específicos

- Realizar una revisión de la literatura que aborde métodos y soluciones al problema de "Flow-Shop" permutado.
- Definir el modelo de programación matemática para el problema y solucionarlo haciendo uso de técnicas exactas.
- Validar el modelo matemático a través de instancias adaptadas de la literatura.
- Implementar el algoritmo genético en un software matemático y evaluar su eficiencia mediante la comparación con instancias halladas en la literatura o por medio de un análisis estadístico de los resultados.
- Elaborar un artículo de carácter publicable con base en los resultados obtenidos de la investigación realizada.

4. Revisión de Literatura

4.1. Antecedentes del PFSP

Los entornos de producción denominados tipo “flow-shop” toman relevancia investigativa desde los años cincuenta con la divulgación de la *Regla de Johnson*, la cual resuelve el problema de programación para dos máquinas con el objetivo de minimizar el tiempo requerido para finalizar *n* cantidad de trabajos en las dos máquinas, es decir, el “makespan” (Molina-Sánchez & González-Neira, 2016).

Desde entonces han sido propuestos numerosos enfoques, en su mayoría basados en el factor tiempo, se han desarrollado modelos y métodos bajo condiciones específicas del sistema productivo para generar programas apropiados como solución al problema de la programación en un entorno tipo “flow-shop”; no obstante, el primer documento de revisión exhaustivo sobre problemas de programación con costos de alistamiento, se realiza en 1999 y cubre unos 200 artículos, desde mediados de la década de 1960 hasta mediados de 1988 (Allahverdi, 2015).

Con el paso del tiempo varios tipos de costos han sido asociados con los problemas de programación en configuraciones tipo “flow-shop”; los costos principales incluyen el costo de producción, el costo de mantener inventario, el costo de alistamiento, el costo de anticipación, el costo de demora, el costo de entrega, entre otros (Mishra & Shrivastava, 2018).

El costo de anticipación emerge si un producto se completa antes de su fecha de entrega, este se encuentra relacionado con el costo de mantener inventario; por otra parte los costos de tardanza

se relacionan con las penalizaciones emergentes si la producción se completa después de la fecha pactada (Bauman & Józefowska, 2006).

Un enfoque importante que no ha recibido mucha atención es el costo del trabajo en proceso (WIP) asociado con el valor que se agrega o se resta durante el proceso de producción (Yang, 2009); Yang & Posner (2010) introducen por primera vez el problema de la programación de dos máquinas con el costo ponderado de WIP; posteriormente, Yang extiende el problema aún más al introducir otras variaciones del problema aumentando su grado de complejidad.

Ahora bien, buscando incorporar la mayor cantidad de parámetros reales de un sistema de manufactura, en la literatura se evidencian otros trabajos con combinaciones de factores asociados a costos; es por ello que Cao & Chen (2003) resuelven el problema de programación minimizando la suma ponderada del costo de producción y el costo incurrido por la entrega tardía del producto. Asimismo, Bülbül et al. (2004) determina la programación que minimiza la suma de los costos de anticipación, tardanza e inventario intermedio en todos los trabajos. Un último ejemplo en el cual el objetivo minimizar el costo total, que incluye los costos de producción, entrega e inventario fue desarrollado por Cheng, Leung, & Li (2017).

A continuación, se presentan los resultados de la revisión de literatura realizada en torno al "flow-shop scheduling" partiendo de la identificación de los objetivos planteados por los diferentes autores, tomando en cuenta la evolución de los atributos o parámetros para la formulación del modelo matemático y finalmente, el reconocimiento de los principales métodos de solución.

4.1.1. Objetivos planteados. En la literatura se encuentran numerosos estudios para los problemas de “flow-shop scheduling”. Los objetivos planteados por cada autor, en la mayoría de los casos, están asociados a factores como: en primera instancia, el tiempo, a partir del cual autores como Benkalai, Rebaine, Gagné, & Baptiste (2017) recientemente introducen una metaheurística para la minimización de “makespan” o tiempo de finalización de los trabajos; del mismo modo, Rathinam (2015) propone una metodología heurística basada en el árbol de decisión para descubrir la secuencia de trabajos para minimizar el tiempo de flujo total.

Otro ejemplo en el que se pretende resolver un problema de programación minimizando la tardanza ponderada total fue propuesto por Molina-Sánchez & González-Neira (2016); por otro lado, Logendran & Mehravaran (2013) realizan un estudio cuyo objetivo era alcanzar el máximo nivel de servicio, minimizando la suma del tiempo ponderado de finalización y la suma de las tardanzas ponderadas.

Como segundo factor se encuentra el *número de trabajos,* factor en el cual autores como Paul & Azeem (2010) basan sus estudios en la minimización del inventario en proceso de trabajo utilizando varios parámetros como la fecha de vencimiento, la tasa de ganancia y el costo a lo largo del tiempo. Adicionalmente, Abouei Ardakan, Hakimian, & Rezvan (2014) en su artículo considera el problema de la programación de dos máquinas con el objetivo de minimizar el número de trabajos tardíos; del mismo modo, Varmazyar & Salmasi (2012) en su investigación consideran tiempos de configuración dependientes de la secuencia y la minimización del número de trabajos tardíos como criterio.

Finalmente el factor *costo,* a partir del cual autores como Ciavotta, Meloni, & Pranzo (2013) resuelven el problema de secuenciar los lotes de manera rentable desde el punto de vista de minimizar el costo total de alistamiento pagado por los departamentos; en contraste, Cheng et al. (2017) consideran el problema de programar un conjunto de trabajos en una sola máquina de procesamiento por lotes con una capacidad fija minimizando el costo total, que incluye los costos de producción, entrega e inventario.

De modo similar, Yang (2009) considera dos nuevos problemas de programación de máquinas, en los que el objetivo era minimizar el costo total de WIP (trabajo en proceso); de forma semejante, Bülbül et al. (2004) dan solución al problema minimizando la suma de los costos asociados a penalizaciones por la tardanza en la entrega de los pedidos de los clientes, así como los costos por mantener inventario en proceso y terminado. Schaller & Valente (2013) consideran un objetivo que resume las penalizaciones en términos monetarios por anticipación y retraso en el procesamiento de un conjunto de trabajos en un "flow-shop".

Si bien los tres factores mencionados (tiempo, número de trabajos y costo) enmarcan de forma general los distintos objetivos propuestos, también es posible encontrar objetivos constituidos por combinaciones de estos los factores; por ejemplo, Hassanzadeh, Rasti-Barzoki, & Khosroshahi (2016) realizan un estudio multiobjetivo en que la primera función objetivo apunta a minimizar la tardanza total ponderada y la duración del proceso, y la segunda función objetivo apunta a minimizar la suma de la antigüedad ponderada total, el número total ponderado de trabajos tardíos, los costos de inventario y los costos totales de entrega; Yang & Posner (2010) por otra parte

estudian un nuevo "flow-shop" determinístico en que el objetivo era minimizar el tiempo total de finalización "makespan" y también minimizar el inventario WIP.

Otros investigadores como Mehravaran & Logendran (2012) consideran el problema de programación de taller de flujo con tiempos de configuración dependientes de la secuencia y un objetivo de bicriterio para minimizar el inventario de trabajo en proceso para el productor y maximizar el nivel de servicio de los clientes; por otro lado, Rasti-Barzoki, Hejazi, & Mazdeh (2013) dirigen su investigación con el objetivo de minimizar la suma del número total ponderado de trabajos tardíos y los costos de entrega.

4.1.2. Métodos de Solución. Para diferentes tipos de problemas se han desarrollado modelos y métodos para generar soluciones optimas, los numerosos métodos de optimización generalmente se dividen en dos grupos: en primera instancia, están los *métodos exactos de optimización*. El método más trivial de solución es el de evaluar todas las posibles soluciones factibles, es decir, *(n!)*m* soluciones siendo *n* el número de trabajos y *m* el número de máquinas, lo cual puede resultar verdaderamente extenso dependiendo del caso. En segundo lugar, encontramos los *métodos de solución no exactos,* basados en heurísticas y metaheurísticas, cuya ventaja es la reducción del tiempo de ejecución computacional comparado con el tiempo que emplearía un método de solución exacto, con la contrapartida de una posible reducción en la calidad de los resultados.

En un entorno tipo "flow-shop" determinístico, uno de los enfoques más ampliamente investigados como se ha expresado anteriormente es el de minimizar el "makespan", para obtener solución a dicho problema, diferentes heurísticas y metaheurísticas han sido propuestas por

investigadores en las últimas décadas, tales como: Algoritmo Genético (GA), Recocido Simulado (SA), Búsqueda Tabú (TS), Búsqueda de vecindad variable (VNS), Colonia Artificial de Abejas (ABC), Optimización de enjambre de partículas (PSO), Optimización de la colonia de hormigas (ACO), Red neuronal artificial (ANN), entre otras.

Debido a su solución múltiple, manejo de gran población y capacidades efectivas de búsqueda por vecindario, las meta-heurísticas, especialmente GA y SA, son las preferidas para resolver problemas difíciles de NP (Arora & Agarwal, 2016), sin embargo, recientemente muchos investigadores dieron solución a estos problemas desarrollando o adaptando nuevos métodos; por ejemplo Cao & Chen (2003) para resolver de manera eficiente el problema de programación de máquinas paralelas en un entorno "flow-shop", proponen un algoritmo heurístico que combina la búsqueda Tabú y el método de Johnson.

Del mismo modo basándose en TS Mehravaran & Logendran (2012), desarrollan una metaheurística de búsqueda que empleaba un concepto recientemente desarrollado conocido como búsqueda Tabú con perturbación progresiva integrada (TSEPP) para resolver en particular, problemas de tamaño industrial de manera eficiente. Por otro lado, Rasti-Barzoki et al. (2013) consideran las propiedades estructurales del problema para una sola máquina y los casos especiales del problema "flow-shop" de dos máquinas para configurar un nuevo algoritmo de ramificación y límite mejor conocido como Branch & Bound.

Años más tarde Hassanzadeh et al. (2016) desarrollan dos nuevos algoritmos meta-heurísticos para resolver el problema biobjetivo, el primer algoritmo es una PSO (optimización de enjambre

de partículas) multiobjetivo combinada con un operador de mutación heurística, función de membresía gaussiana y una secuencia caótica y el segundo es un algoritmo genético de clasificación no dominado II con una criterio heurístico para la generación de población inicial y un operador de cruce heurístico. En ese mismo año Molina-Sánchez & González-Neira (2016) proponen como una solución la metaheurística GRASP (Procedimiento de búsqueda adaptativa aleatoria codiciosa) que ha mostrado resultados competitivos para problemas combinatorios.

Otros autores emplearon un nuevo enfoque metaheurístico y evaluaron su desempeño para dar solución al problema de programación de un entorno “flow-shop”, tal es el caso de Benkalai et al. (2017) quienes adaptan una metaheurística conocida como Migrating Birds Optimization (MBO) para la minimización de “makespan”. Se presentan dos versiones del algoritmo, el primero es un MBO básico y el segundo presenta características adicionales. Un año más tarde Mishra & Shrivastava (2018), aplican dos nuevas técnicas de optimización metaheurística para optimizar la función objetivo: TLBO (optimización basada en enseñanza-aprendizaje) y Jaya y las compararon con respecto a dos algoritmos tradicionales: PSO (optimización de enjambre de partículas) y SA (recocido simulado). Paul & Azeem (2010) usan Lógica Difusa para obtener la minimización del inventario de trabajo en proceso en la programación de “flow-shops” híbridos.

4.2. Discusión y conclusiones

En términos generales, si bien los resultados experimentales mostraron que los métodos de solución propuestos superaron en una amplia variedad de configuraciones de problemas a los métodos comparativos, los autores argumentan que los métodos heurísticos ante la posibilidad de caer en un óptimo local deben ser mejorados y su desempeño debe ser estudiado; por tanto, se

considera necesaria la correcta calibración de aquellos parámetros que por naturaleza le proporcionan flexibilidad a la heurística.

Adicionalmente, otro aspecto sobre el que los autores concluyen es sobre la necesidad latente de investigaciones adicionales, la cuales deben incluir otras variantes del problema de la programación en un entorno “flow-shop”, especialmente inspirado en aplicaciones del mundo real como; por ejemplo, considerar las posibles fuentes de incertidumbre o la inclusión de nuevas restricciones. Dentro de las nuevas investigaciones también hay lugar para el desarrollo, la adaptación o aplicabilidad de otro tipo de heurísticas que demuestren ser más eficientes que las ya planteadas.

Basados en los resultados de su estudio computacional algunos autores demostraron que las soluciones metaheurísticas para los problemas proporcionan una ventaja de velocidad significativa sobre métodos alternativos y/o exactos y no comprometen la calidad de la solución en un alto grado; sin embargo, reconocen que su modelo tiene varias limitaciones.

5. Marco Teórico

5.1.“Flow-shop”

Se denomina Flow-shop” a una forma de configuración de las instalaciones de un sistema productivo en función del flujo de producción, se presenta cuando todos los trabajos siguen la misma ruta de procesamiento por lo que las máquinas normalmente se configuran en serie como se puede observar en la *Figura 1*. Algunas ventajas de una configuración tipo “flow-shop” son:

- Alto grado de automatización, procesos altamente estandarizados.
- Fabricación a grandes volúmenes, pueden ser productos discretos o no.
- Susceptible a optimizarse mediante balanceo de líneas.

Algunas desventajas de las tiendas de flujo son:

- La inversión inicial es alta debido a la maquinaria requerida en cada operación.
- Procesos poco flexibles, productos técnicamente homogéneos con poca variedad.
- Aumentar la capacidad a corto plazo resulta un gran desafío.

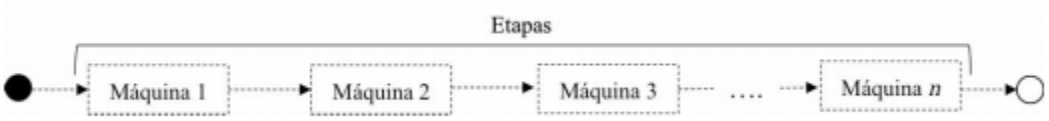


Figura 1. Configuración "Flow-shop" clásico.

5.2.Problema de “Flow-shop” (PFS)

La programación de los trabajos en un entorno tipo “Flow-shop” se conoce comúnmente como “flow-shop scheduling”. El problema radica en la determinación de la secuencia para un conjunto de *n* trabajos, tareas o elementos que logra el valor mínimo de la función objetivo, cuando todos

los trabajos se procesan en las *m* máquinas en el orden indicado por la secuencia. Las siguientes hipótesis adicionales se suelen asumir para el PFS:

- Cada trabajo puede procesarse como máximo en una máquina al mismo tiempo.
- Cada máquina *m* puede procesar solo un trabajo a la vez.
- No se permite ninguna preferencia, es decir, el procesamiento de un trabajo en una máquina determinada no se puede interrumpir.
- Todos los trabajos son independientes y están disponibles para su procesamiento en el momento 0.
- Los tiempos de configuración de los trabajos en las máquinas son insignificantes y, por lo tanto, pueden ignorarse.
- Las máquinas están continuamente disponibles.
- Se permite el inventario en proceso. Si la siguiente máquina en la secuencia que necesita un trabajo no está disponible, el trabajo puede esperar y unirse a la cola en esa máquina.

5.2.1. "Flow-Shop" permutado (PFS). En este caso las soluciones se ven limitadas por la secuencia de las tareas, lo que significa que el orden de procesamiento de los trabajos en las máquinas debe seguir siendo el mismo en cada máquina a través la línea de producción.

5.2.2. "Flow-Shop" sin permutación (NPFS). En esta variante, por el contrario, la secuencia de procesamiento puede cambiar de una máquina a otra por lo que mientras el enfoque PFS busca su solución óptima entre *n!* soluciones factibles, siendo n el número de trabajos, el enfoque NPFS tiene que considerar $n!^m$ posibilidades.

5.2.3. "Flow-Shop" sin espera. En esta variación del Flow-Shop básico no se permite que las tareas formen colas entre las máquinas. Este tipo de programación es necesaria cuando se requiere que las operaciones sigan una inmediatamente después de la otra debido a restricciones de producción, como durante la producción de molduras de acero o plástico.

5.2.4. "Flow-Shop" flexible/híbrido. Es aquella en la que existen máquinas en paralelo que pueden ser idénticas o no; las estaciones en paralelo reducen los tiempos de ciclos necesarios para una operación en una estación. De esta manera, puede darse el caso que una tarea pueda alcanzar a su predecesora.

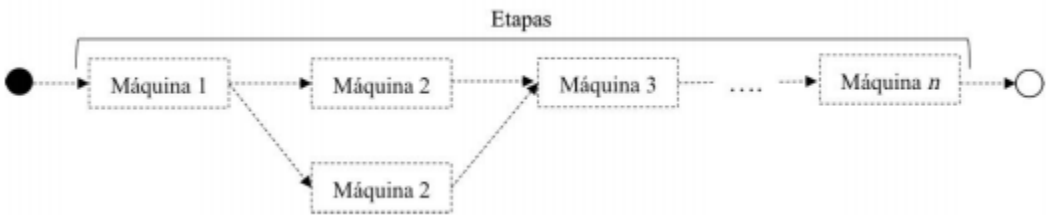


Figura 2. Configuración "Flow-shop" flexible

5.3. Optimización

La optimización matemática intenta dar respuesta a un tipo general de problemas donde se desea determinar la mejor solución entre un conjunto de elementos o soluciones factibles, tales que minimizan o maximizan una función objetivo. En términos generales, un problema de optimización hace referencia a un modelo matemático sujeto a restricciones de la forma mostrada a continuación en la *Figura 3*:

$$\begin{cases} \text{minimizar } f(x) \\ \text{sujeto a} \\ g_i(x) \le 0 \qquad i = 1,\ldots,m \\ x \in S \subset \Re^n. \end{cases}$$

Figura 3. Modelo Matemático general

5.3.1. Optimización combinatoria. En este tipo de problemas de optimización las variables de decisión son enteras, es decir, el espacio de soluciones está formado por ordenaciones o subconjuntos de números naturales. En este caso, se trata de hallar el mejor valor de entre un número finito o numerable de soluciones viables. Sin embargo, la enumeración de este conjunto resulta compleja, aún para problemas de tamaño moderado.

Usualmente se emplean métodos heurísticos para resolver instancias de problemas de optimización combinatoria, los cuales logran esto reduciendo el tamaño efectivo de la región de soluciones factibles y explorando el espacio de búsqueda eficientemente.

5.4. Complejidad Computacional

Los problemas de optimización pueden ser clasificados de acuerdo con la dificultad propia para resolverlos, basándose en los recursos necesarios y requeridos para establecer su grado de complejidad (Cruz Chavez, Moreno Bernal, & Peralta Abarca, 2014).

El modelo computacional de Turing clasifica los problemas por el grado de complejidad para resolverlos, a través de este modelo se han detectado problemas intratables, clasificados como NP “non-deterministic polynomial time”, que se piensa son imposibles de resolver en un tiempo

razonable cuando el número de variables que los componen es una cantidad extremadamente grande. Dentro de este grupo de problemas NP se encuentran dos subconjuntos de problemas como se observa a continuación en la *Figura 4*:

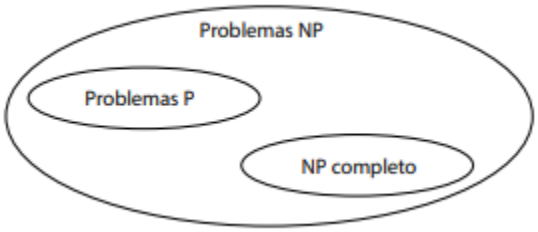


Figura 4. Clasificación de los problemas NP

5.4.1. Problemas P. Son considerados como el tipo de problemas relativamente sencillos para los que existen algoritmos eficientes o exactos, es decir, este tipo de problemas pueden ser resueltos mediante un algoritmo determinista con complejidad polinomial en tiempo polinómico.

5.4.2. Problemas NP-Complete. Estos problemas NP son aquellos cuya solución no se ha resuelto de manera exacta por medio de algoritmos deterministas, por el contrario, se tratan de resolver por medio de heurísticas computacionales acotados en tiempo polinomial, cuya solución deseable sea de complejidad polinomial.

Cabe resaltar que los algoritmos no determinísticos usados para los problemas NP y NP-completo, también pueden usarse en los problemas P, sin embargo, no es posible en caso contrario, que un algoritmo determinístico que resuelva un problema P en tiempo polinomial también pueda resolver un NP (H. Papadimitriou & Steiglitz, 1982).

5.4.3. Problemas NP-Hard. Un problema es NP-hard si es, al menos, tan difícil como un problema NP; muy a menudo los problemas de NP-hard requieren un tiempo exponencial para ser resueltos. Son problemas que implican, se abordan y se resuelven en un tiempo polinomial no-determinista. Los problemas NP-duros pueden ser del orden de problemas de decisión, problemas de búsqueda o bien problemas de optimización (Maldonado, 2013).

5.5. Programación Lineal Entera y Entera Mixta

Castillo, Conejo, & Pedregal (2002) definen un modelo de programación lineal entera como aquel donde las variables son números enteros no negativos. Cuando sólo es necesario que algunas de las variables sean enteras y el resto continuas, el modelo recibe el nombre de problema de Programación Lineal Entera Mixta. Esta clasificación incluye modelos que además de tener variables enteras no negativas y variables continuas, tienen también variables binarias (Hillier & Lieberman, 2001).

Los problemas de programación con enteros se formulan de la misma manera que los problemas de programación lineal, pero agregando la condición de que al menos alguna de las variables de decisión debe tomar valores enteros (Colina, 2011).

5.6. Métodos de Solución

Se han usado enfoques exactos y de aproximación para dar solución al PSFP, los primeros garantizan una solución óptima, siempre y cuando ésta exista; sin embargo, para instancias grandes el enfoque exacto podría resultar inviable debido a que el tiempo computacional crece en forma exponencial con el tamaño del problema, principal característica de los problemas NP-hard. Los

enfoques de aproximación son metodologías heurísticas o metaheurísticas que, si bien no garantizan la solución exacta, pueden dar muy buenas soluciones aproximadas en tiempos razonables.

5.6.1. Métodos Exactos. Como se mencionó anteriormente, en contraposición a los métodos heurísticos que se limitan a proporcionar una buena solución aproximada, los métodos exactos proporcionan la solución exacta del problema. La mayoría de los métodos de resolución exacta de un modelo de programación lineal entera se encuadran en alguno de los siguientes esquemas definidos por Rodes, (2011):

- *Branch and Bound.* La estrategia de un algoritmo Branch-and-Bound consiste en dividir el espacio de soluciones con el objetivo de facilitar la búsqueda del ´optimo. Al aplicar este concepto recursivamente, se genera un árbol cuya raíz corresponde al problema original y sus nodos tienen asociados subproblemas que resultan de la división en partes del espacio de búsqueda. Debido al tamaño que puede alcanzar el árbol, es esencial disponer de herramientas eficientes que permitan eliminar algunas de sus ramas.

- *Planos de Corte.* modificar el dominio de la relajación lineal asociada al problema entero hasta obtener una solución ´optima que satisfaga las condiciones de integridad requeridas. Para llevar a cabo esta operación, se utilizan una serie de desigualdades lineales, llamadas “planos de corte”, que, al ser agregadas a la formulación del problema, permiten eliminar soluciones fraccionarias de la relajación y conservar el conjunto de soluciones enteras del problema original.

- *Branch and Cut.* metodología mixta para resolver problemas binarios. Trabajaron con un algoritmo Branch and Bound, pero antes de comenzar la primera etapa de branching, aplicaron un algoritmo de Planos de Corte a la relajación lineal asociada a la raíz del árbol. De esta manera lograron mejorar la cota superior brindada por la relajación lineal y eso disminuyo el tamaño del árbol explorado.

- *Relajación Lagrangiana.* Es un método cuya idea se basa en descomponer un problema original restringido, en principio complejo de resolver, y reemplazarlo por otro problema que permita simplificar la resolución. Esto se logra incorporando aquellas restricciones que hacen compleja la resolución directa del problema a la función objetivo, donde cada una de éstas tendrá asociada un multiplicador de Lagrange que permitirá iterativamente penalizar el incumplimiento de estas al ser establecidos distintos valores para los multiplicadores.

5.6.2. Métodos Heurístico. Según Díaz et al., (1996) un método heurístico es un procedimiento para resolver un problema de optimización bien definido mediante una aproximación intuitiva, en la que la estructura del problema se utiliza de forma inteligente para obtener una buena solución. El principal problema que presentan los algoritmos heurísticos es su incapacidad para escapar de los óptimos locales como se muestra en la *Figura 5*.

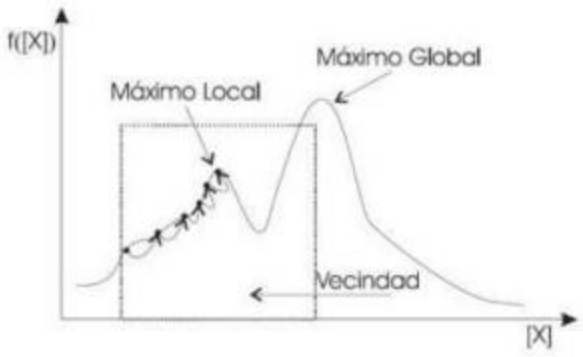


Figura 5. Encasillamiento de los algoritmos heurísticos en óptimos locales. Adaptado de Trabajo fin de máster titulado Algoritmos Heurísticos en Optimización

Existen muchos métodos heurísticos de naturaleza diferente, algunas de las categorías más amplias definidas por (Martí, 2003) son:

- *Métodos Constructivos.* Consisten en construir literalmente paso a paso una solución del problema. Usualmente son métodos deterministas y suelen estar basados en la mejor elección en cada iteración. Estos métodos han sido muy utilizados en problemas clásicos como el del viajante.

- *Métodos de descomposición.* El problema original se descompone en subproblemas más sencillos de resolver, teniendo en cuenta, aunque sea de manera general, que ambos pertenecen al mismo problema.

- *Métodos de Reducción.* Consiste en identificar propiedades que se cumplen mayoritariamente por las buenas soluciones e introducirlas como restricciones del problema. El objeto es restringir el espacio de soluciones simplificando el problema. El riesgo obvio es dejar fuera las soluciones óptimas del problema original.

- *Métodos Inductivos.* La idea de estos métodos es generalizar de versiones pequeñas o más sencillas al caso completo. Propiedades o técnicas identificadas en estos casos más fáciles de analizar pueden ser aplicadas al problema completo.

- *Métodos de Búsqueda Local.* A diferencia de los métodos anteriores, los procedimientos de búsqueda o mejora local comienzan con una solución del problema y la mejoran progresivamente. El procedimiento realiza en cada paso un movimiento de una solución a otra con mejor valor. El método finaliza cuando, para una solución, no existe ninguna solución accesible que la mejore.

5.7. Metaheurísticas

En los últimos años ha habido un crecimiento espectacular en el desarrollo de procedimientos heurísticos para resolver problemas de optimización, a continuación, se presentan algunos de los más comunes:

5.7.1. Búsqueda Tabú TS.

Cogollo Flórez (2017) define Es un procedimiento iterativo y heurístico para resolver problemas discretos de optimización combinatoria y de gran escala. Busca escapar de óptimos locales empleando algunas metodologías como el uso de memorias flexibles. Además, esta técnica impone y relaja las restricciones con el objetivo de explorar áreas prohibidas y de hacer cortes de la región factible, al tener en cuenta las restricciones que la limitan.

5.7.2. Colonia de Hormigas ACO.

Esta técnica metaheurística constituye un enfoque a la resolución de problemas de optimización combinatoria. Los algoritmos ACO son modelos inspirados en el comportamiento de colonias de hormigas reales: cómo son capaces de seguir la

ruta más corta entre la colonia y una fuente de abastecimiento, en un viaje de ida y regreso. Esto se debe al rastro de feromona que cada una de ellas va dejando a lo largo del camino. A medida que las hormigas siguen un rastro, van dejando su propia feromona, lo cual hace que la probabilidad de escogencia de un camino sea proporcional al número de hormigas que lo hayan escogido previamente. (Cogollo Flórez, 2017)

5.7.3. Recocido Simulado SA. Simulated annealing en inglés, se basa en principios de la termodinámica y el proceso de recocido del acero. Inicialmente, a temperaturas muy elevadas se produce una amalgama liquido en el que las partículas se configuran aleatoriamente. El estado sólido se caracteriza por tener una configuración concreta de mínima energía, el mínimo global. Para alcanzar esa configuración es necesario enfriar la amalgama lentamente ya que un enfriamiento brusco paralizaría el proceso y se llegaría a una configuración distinta de la buscada, un mínimo local distinto del mínimo global (Vidal Esmorís, 2013).

5.7.4. Búsqueda en Vecindario Variable VNS. Según Hansen, Mladenovic, & Moreno Pérez (2003) la VNS es una metaheurística para resolver problemas de optimización cuya idea básica es el cambio sistemático de entorno dentro de una búsqueda local. La VNS está basada en tres hechos simples: un mínimo local con una estructura de entornos no lo es necesariamente con otra; un mínimo global es mínimo local con todas las posibles estructuras de entornos; para muchos problemas, los mínimos locales con la misma o distinta estructura de entornos están relativamente cerca.

5.7.5. Algoritmo Genético. Es una técnica evolutiva de búsqueda desarrollada por el Dr. John Henry Holland, plasmada en su publicación “Adaptation in Natural and Artificial Systems” en el año 1975. La técnica se basa en la simulación de procesos biológicos donde la herencia genética de padres a hijos tras generaciones es su base fundamental.

Un algoritmo genético permite determinar cuáles individuos aptos para reproducción, y cuáles deben ser descartados; esto a través de parámetros y operadores genéticos establecidos que permitan llevar control y medición sobre los individuos seleccionados, de tal manera que se mantenga una población con genes aptos para heredar, sin dejar a un lado la diversidad entre los individuos seleccionados; lo cual garantiza una mejor representación de la realidad donde se abarquen los mejores individuos posibles en la población final.

Un algoritmo genético es ideal para problemas de optimización de magnitudes considerables. Mediante un proceso de iteraciones se busca mantener a través de generaciones una población final óptima que satisfaga las restricciones definidas para el caso. Para ello se debe definir la estructura del cromosoma, que dará representación a las posibles soluciones del problema, una función “fitness” que permita evaluar los individuos con respecto a la solución del problema, los operadores genéticos principales y a su vez los parámetros que van a dar estructura al problema de optimización.

- *Función “fitness”.* Es un elemento determinante dentro de un algoritmo genético, ya que ella establece en gran manera el proceso de evolución de los individuos. Mediante la función “fitness” se evalúan los valores de aptitud del cromosoma o individuo, y se determina si deben ser

elegidos para reproducción y creación de una nueva generación, o si por el contrario deben ser desechados. La estructura de la función será dependiente del problema de optimización a evaluar.

- *Población inicial.* Es un elemento crucial por determinar dentro de un algoritmo genético. De ser un tamaño pequeño con respecto a la magnitud del problema, se corre el riesgo de converger en corto tiempo a un óptimo local, y de ser un tamaño más grande que el ideal es más probable converger a un óptimo global, aunque la convergencia tome más tiempo. La población suele ser determinada al azar, probando diferentes niveles, sin embargo, algunos autores como Reeves (1998), quien utiliza el algoritmo NEH para generar la población inicial, obtuvo una solución de esta heurística y generó otras al azar. Chen (1995), por otro lado, utilizó una heurística desarrollada por Campbell, Dudek y Smith (CDS) (1970) para construir la población inicial.

- *Selección y reproducción*. El operador de selección y reproducción es el encargado de codificar al algoritmo para reproducir con mayor frecuencia a los individuos con mejores aptitudes, para mantener así la calidad de la población a través de las generaciones, imitando la herencia de los genes de la biología. De igual manera se debe tener en cuenta durante la selección el factor de diversidad, que permite seleccionar individuos con condiciones menos favorables para la reproducción, esto con el fin de evitar que el resultado converja en un óptimo local y asemejar la reproducción a la selección natural.

En concordancia con lo mencionado, existen diferentes métodos de selección; entre los más comunes se encuentran el método de selección por ruleta, en donde se asigna a cada cromosoma o individuo una sección de la ruleta proporcional a sus niveles de aptitud, y se procede a seleccionar

cromosomas de manera pseudoaletoria. Otro método usado con frecuencia es la selección por torneo, que consiste en enfrentar cromosomas escogidos al azar, seleccionando al cromosoma con mejor nivel de aptitud para la reproducción.

- *Cruce.* Es el operador que permite el intercambio de información durante la reproducción de los cromosomas, es decir, se acoplan características de ambos padres para dar lugar a una nueva generación que contenga la información genética de sus antecesores. El operador de cruce es el responsable de la reproducción, ya que es quien permite la formación y codifica la estructura de las nuevas generaciones.

- *Mutación.* La mutación permite tener un nivel determinado de diversidad, lo cual permite acercarnos más a la realidad de la biología genética y abarcar una población más amplia, evitando caer en óptimos locales. Durante la mutación se realiza un cambio aleatorio entre la estructura de los cromosomas en uno de los puntos de la estructura o en múltiples puntos. Ya que la mutación no es una característica común en el mundo real, igualmente en los algoritmos genéticos se debe considerar una probabilidad baja de mutación.

Una vez definidos cada uno de los operadores que componen el algoritmo GA, en la Figura 6 se observa el seudocódigo de funcionamiento para un algoritmo genético, en él se puede observar de forma más clara el rol que desempeña cada operador y como se interrelaciona con los demás.

```
Inicializar población actual aleatoriamente
MIENTRAS no se cumpla el criterio de terminación
        crear población temporal vacía
        SI elitismo: copiar en población temporal mejores individuos
        MIENTRAS población temporal no llena
        seleccionar padres
        cruzar padres con probabilidad Pc
        SI se ha producido el cruce
                mutar uno de los descendientes (prob. Pm)
                evaluar descendientes
                añadir descendientes a la población temporal
        SINO
                añadir padres a la población temporal
        FIN SI
        FIN MIENTRAS
        aumentar contador generaciones
        establecer como nueva población actual la población temporal
FIN MIENTRAS
```

Figura 6. Seudocódigo funcionamiento de un Algoritmo Genético. Adaptado de Dorado, Gestal, Rivero, Rabuñal, & Pazos (2010)

5.8. Análisis de Varianza ANOVA

El análisis de la varianza hace referencia al conjunto de técnicas estadísticas que permiten analizar cómo operan diversos factores, estudiados simultáneamente en un diseño factorial, sobre una variable respuesta. El objetivo principal de muchos experimentos consiste en determinar el efecto que tienen distintos niveles de algún factor, variable independiente y discreta, sobre alguna variable dependiente.

Condiciones:

- Cada muestra debe ser independiente de las otras.
- Cada muestra debe haber sido seleccionada al azar de la población de donde proviene.
- La población de donde provienen las muestras debe tener distribución normal.
- Las varianzas de cada población deben ser iguales.

Es común presentar el Análisis de Varianza en forma de tabla como se observa a continuación en la *Figura 7*:

FUENTE DE VARIACIÓN	SUMA DE CUADRADOS	GRADOS DE LIBERTAD	CUADRADO MEDIO	CONTRASTE
ENTRE TRATAMIENTOS (VE)	$S_T = \sum_i^k n_i \left(\bar{y}_{t_i} - \bar{y}\right)^2$	$\nu_T = k - 1$	$s_T^2 = \frac{S_T}{k-1}$	$\frac{s_T^2}{s_R^2}$
DENTRO DE TRATAMIENTOS (VNE)	$S_R = \sum_i^k \sum_j^{n_i} \left(y_{ij} - \bar{y}_i\right)^2$	$\nu_R = N - k$	$s_R^2 = \frac{S_R}{N-k}$	
TOTAL EN RELACIÓN A LA MEDIA GENERAL (VT)	$S_D = \sum_{i=1}^k \sum_{j=1}^{n_i} y_{ij}^2 - N\bar{y}^2$	$\nu_D = N - 1$	$s_D^2 = \frac{S_D}{N-1}$	

Figura 7. Análisis de Varianza de un Factor. Adaptado de: Herramientas Estadísticas - Comparación de más de dos muestras: Anova

Según Ruiz, (2009), la metodología para realizar el Análisis de Varianza puede resumirse como sigue a continuación:

- Fijar el nivel de significancia para el contraste, por ejemplo, α=95%.
- Establecer el contraste de hipótesis:

  ⇒ H_0: Los tratamientos son todos iguales: μ1=μ2=μ3=…=μk.

  ⇒ H_1: Alguno de los tratamientos es diferente.
- Calcular los estimadores S_R^2 y S_T^2
- Calcular el valor del estadístico S_T^2 / S_R^2

- Calcular el valor de $F_{k-1,\ n-k}$ para el nivel de significación prefijado. Si:

  $\Rightarrow S_T^2 / S_R^2 > F_{k-1,\ n-k}$ La diferencia entre los tratamientos es estadísticamente significativa con un nivel de significancia α.

  $\Rightarrow S_T^2 / S_R^2 < F_{k-1,\ n-k}$ La diferencia entre los tratamientos no es estadísticamente significativa con un nivel de significancia α.

6. Descripción del PFSP

Para resolver el problema bajo estudio, se toma como referencia el modelo matemático planteado por Mishra & Shrivastava (2018) quienes consideran un PFSP con un número *m* de máquinas organizadas en serie y *n* trabajos. Cada trabajo deberá recorrer las máquinas formando parte de una secuencia determinada, la cual deberá ser la misma de una máquina a otra.

Dado que se supone que la materia prima para todos los lotes es liberada al inicio del programa, se incurre en el costo de mantenimiento de inventario (unidad de costo/tiempo) mientras esté esperando en la cola antes de la máquina j, es decir, el tiempo de espera para el lote incluye el procesamiento y el tiempo de configuración de todos los lotes programados anteriormente y el tiempo de configuración del lote actual en la máquina.

En segundo lugar, durante el procesamiento de un lote, la materia prima se agota a una tasa constante y, por lo tanto, se incurre en el costo de mantenimiento de inventario para este período en función de la cantidad promedio de materia prima. Por simplicidad, para resolver el problema se parte de las siguientes suposiciones:

- La preferencia no está permitida, lo que significa que cada trabajo debe esperar la finalización de su trabajo anterior en la misma máquina.
- Los trabajos son independientes de la secuencia.
- Se considera que las máquinas funcionan continuamente con la misma eficiencia, es decir, sin falla o avería de las máquinas durante la producción.
- Las penalidades por anticipación a la fecha de entrega no se cuentan. En otras palabras, se consideran los costos de mantenimiento de inventario y los costos de penalización de trabajos tardíos.
- La materia prima para todos los trabajos se libera al inicio del programa.
- El costo de mantenimiento inventario de todos los trabajos es constante.
- Los costos de instalación son despreciados.

7. Modelo Matemático para el PFSP

A continuación, se presenta la formulación del modelo matemático para la solución al PFSP descrito en el apartado anterior, considerando los costos de inventario y los costos de penalización debidos a la demora del lote:

Conjuntos

i	Trabajos 1, 2, 3, …, n
j	Maquinas 1, 2, 3, …, m
k	Posición del trabajo en la secuencia 1, 2, 3, …, n

Parámetros

n	Número de trabajos
m	Número de máquinas
$p_{i,j}$	Tiempo de procesamiento del trabajo i en la máquina j (en minutos)
$s_{i,j}$	Tiempo de alistamiento para el lote i en la máquina j (en horas)
bs_i	Tamaño de lote para el trabajo i
t_f	Factor de retraso
ic	Costo de mantenimiento de inventario por trabajo por hora
pc_i	Costo de demora por unidad de tiempo, si el trabajo completa su procesamiento en la máquina final más allá de la fecha de entrega.

Variables

$E(CPUT)$	Costo total por unidad de tiempo de programación
IC_{bw}	Costo total por mantenimiento de inventario para los lotes esperando en la cola antes de las máquinas.
IC_{wp}	Costo total por mantenimiento de inventario de los lotes en procesamiento.
PC_{bd}	Costo total de penalización de lotes retrasados más allá de su fecha de entrega
$CT_{i,n,m}$	Tiempo de finalización del trabajo programado en la posición n en máquina m (makespan)
$CT_{i,k,j}$	Tiempo de finalización del trabajo programado en la posición k en la máquina j
$BW_{i,k,j}$	Tiempo de espera para el lote i programado en la k-ésima posición en la máquina j (en horas)

$P_{i,j}$ Tiempo de procesamiento total del lote i en la máquina j, incluida el tiempo de configuración (en horas).

DD_i Fecha de entrega del trabajo i (en horas).

$$Minimize\ E(CPUT)_s = \frac{(IC_{bw} + IC_{wp} + PC_{bd})}{CT_{i,n,m}} \quad (1)$$

$$P_{i,j} = \frac{bs_i * p_{i,j}}{60} + s_{i,j} \quad (2)$$

$$CT_{i,1,1} = P_{i,1,1} \quad (3)$$

$$CT_{i,k,1} = CT_{i,k-1,1} + P_{i,k,1} \quad (4)$$

$$CT_{i,1,j} = CT_{i,1,j-1} + P_{i,1,j} \quad (5)$$

$$CT_{i,k,j} = max\{(CT_{i,k-1,j} - CT_{i,k,j-1}, 0)\} + CT_{i,k,j-1} + P_{i,k,j} \quad (6)$$

$$BW_{i,1,j} = s_{i,j} \quad (7)$$

$$BW_{i,k,1} = s_{i,1} + CT_{i,k-1,1} \quad (8)$$

$$BW_{i,k,j} = s_{i,j} + max\{(CT_{i,k-1,j} - CT_{i,k,j-1}, 0)\} \quad (9)$$

$$DDi\ =\ n * \sum_{j=1}^{m} P_{ij} * (1 - t_f), 0 < t_f < 1 \quad (10)$$

$$IC_{bw} = ic * \left\{\sum_{i=1}^{n}\sum_{j=1}^{m} \theta_{i,k,j} * (bs_i * BW_{i,k,j})\right\} \quad (11)$$

$$IC_{wp} = ic * \left\{\sum_{i=1}^{n}\sum_{j=1}^{m} \frac{bs_i}{2} * p_{i,k,j} * \theta_{i,k,j}\right\} \quad (12)$$

$$\theta_{i,k,j} = \begin{cases} 1, & si\ CT_{i,k,j} > DD_i \\ 0, & DLC \end{cases} \quad (13)$$

$$PC_{bd} = \sum_{i=1}^{n} pc_i * max\{(CT_i - DD_i), 0\} \quad (14)$$

$$n, m, p_{i,j}, s_{i,j}, bs_i, P_{i,j}, CT_{i,k,j}, BW_{i,k,j}, DD_i, t_f, ic, IC_{bw}, IC_{wp}, pc_i, PC_{bd}, CT_{i,n,m}, E(CPUT)_s > 0 \quad (15)$$

El principal objetivo del PFSP es encontrar la secuencia de lotes que optimiza el modelo matemático, la función objetivo $E(CPUT)_s$ (1) contempla el costo total por unidad de tiempo de programación (makespan), es decir, expresa la relación entre la sumatoria de tres (3) costos y el "makespan", dichos costos son: el costo total por mantenimiento de inventario para los lotes esperando en la cola antes de las máquinas, el costo total por mantenimiento de inventario de los lotes en procesamiento y costo total de penalización de lotes retrasados más allá de su fecha de entrega.

La ecuación (2) permite determinar el tiempo de procesamiento total en horas de un lote en una máquina, este tiempo de procesamiento es resultado de la suma entre el tiempo de alistamiento "setup time" de la máquina y el tiempo de procesamiento del respectivo lote. Las restricciones (3) a (6) definen el proceso para el cálculo del "makespan": en (3) se establece que para un lote programado en la primera posición en la primera máquina el tiempo de finalización es igual al tiempo total de procesamiento del respectivo lote en la primera máquina.

La ecuación (4) efectúa el cálculo del tiempo de finalización en la primera máquina de los lotes programados en posiciones subsiguientes a la primera, el cual corresponde para determinado lote "makespan" acumulado en la posición anterior más el tiempo de procesamiento total del actual lote. Así mismo en (5) se determina el tiempo de finalización de un lote programado en la primera posición en las máquinas posteriores a la primera máquina, dicho tiempo corresponde al "makespan" acumulado del lote en la máquina anterior más el tiempo de procesamiento total de la actual máquina.

En ese sentido, por medio de la ecuación (6) se calcula el tiempo de finalización de un lote programado en una posición subsiguiente a la primera y en las máquinas posteriores a la primera máquina, permitiendo así el cálculo definitivo del "makespan" cuando se determine el tiempo de finalización del lote programado en la última posición y en la última máquina. Esta ecuación a su vez articula el proceso entre máquinas garantizando que una máquina no procese más de un lote a la vez, el tiempo de finalización tiene en cuenta el tiempo que deberá esperar determinado lote en caso de que la máquina subsiguiente se encuentre ocupada.

Las restricciones (7), (8) y (9) definen el proceso para el cálculo de las esperas entre máquinas para un lote cuya máquina siguiente se encuentre ocupada: en (7) se establece que para un lote programado en la primera posición el tiempo que espera es igual al tiempo de alistamiento "setup time" del lote en la respectiva máquina. La ecuación (8) calcula del tiempo de espera en la primera máquina de los lotes programados en posiciones siguientes a la primera posición, cuyo tiempo para determinado lote corresponde al "makespan" acumulado en la posición anterior más el tiempo de alistamiento del actual lote.

Por medio de la ecuación (9) se calcula el tiempo que espera un lote programado en una posición diferente a la primera en las máquinas posteriores a la primera máquina; tomando como base el tiempo de finalización del lote y el tiempo en que se libera la siguiente máquina se determina el tiempo que deberá esperar el lote cuando la máquina subsiguiente se encuentre ocupada. Por otro lado, en (10) se calcula la fecha de entrega estimada para los trabajos basándose en el número de trabajos y aplicando un factor de tardanza que agrega flexibilidad al tiempo de procesamiento.

Las ecuaciones (11) y (12) permiten calcular el costo por mantener inventario en espera de ser procesado por una máquina y el costo por mantener inventario en procesamiento; sin embargo, estos costos solo se harán efectivos cuando el tiempo de completamiento del trabajo sea mayor a la fecha de entrega del lote, como se puede apreciar en la ecuación (13). Adicional a estos costos se tiene el costo total de penalización, calculado mediante la ecuación (14), el cual se incurre cuando un lote se encuentra retrasado en su entrega, es decir, cuando el "makespan" se hace superior a la fecha de entrega. Finalmente, la ecuación (15) corresponde a la ecuación de no negatividad.

7.1. Ejemplo ilustrativo

Complementando la explicación anteriormente realizada, el siguiente ejemplo contempla el caso práctico formulado por Mishra & Shrivastava (2018), para cuatro (4) trabajos y tres (3) máquinas como se evidencia en la Tabla 2 en la cual se presentan los datos iniciales, los cuales corresponden al tiempo de procesamiento (p_{ij}), el tiempo de alistamiento (s_{ij}), el tamaño de lote (bs_i), la fecha de entrega (DD_i), el costo por mantener inventario (ic) y el costo por penalización de lotes retrasados (pc)

Tabla 2.

*Parámetros iniciales para la instancia 4*3.*

Trabajos	***Tiempo de procesamiento (min)***			***Tiempo de alistamiento (hrs)***			***Tamaño lote***	***Due date***	***Penalidad***
	M1	**M2**	**M3**	**M1**	**M2**	**M3**			
1	5	9	10	4	2	2	400	314	180
2	9	9	8	4	1	4	400	284	139
3	8	3	1	4	4	3	400	81	130
4	5	10	6	1	4	4	400	275	165

Adaptado de Mishra & Shrivastava (2018)

Por considerarse de un problema tipo combinatorio discreto, el número de soluciones factibles se reducen al número de permutaciones n!; sin embargo, de las 24 posibles soluciones solo se explicará el proceso a través de las ecuaciones para la secuencia 2-3-4-1. La ecuación (2) permite determinar el tiempo de procesamiento total en horas de un lote en una máquina como se puede observar en la Tabla 3.

Tabla 3.

Cálculo del tiempo de procesamiento total

Trabajos	***Tiempo total de procesamiento (hrs)***		
	M1	**M2**	**M3**
1	37,33	62	68,67
2	64	61	57,33
3	57,33	24	9,67
4	34,33	70,67	44

Adaptado de Mishra & Shrivastava (2018)

Como se puede observar en la *Figura 8*, en (3) el tiempo de finalización para el lote 2 corresponde al tiempo de procesamiento total de este lote en la primera máquina. Así mismo en la primera máquina, de acuerdo con la ecuación (4) el tiempo de finalización para el lote 3 se calcula a partir del tiempo de finalización del lote anterior más el tiempo de procesamiento total del lote 3; del mismo modo se realiza el cálculo del tiempo de finalización en la primera máquina para el lote 4 y 1.

$$CT_{i,1,1} = P_{i.1,1}$$

$$CT_{i,k,1} = CT_{i,k-1,1} + P_{i,k,1}$$

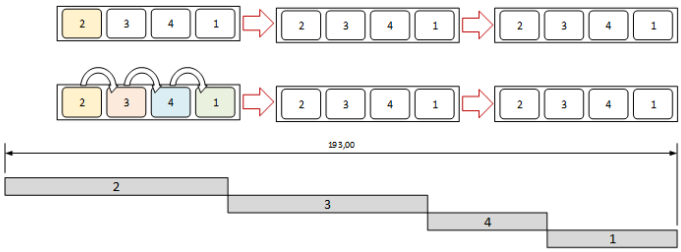


Figura 8. Representación gráfica ecuación (3) y (4) cálculo tiempo de finalización instancia 4*3

Continuando con el proceso y tomando como referencia la *Figura 9*, en (5) se establece que el tiempo de finalización en las máquinas 2 y 3 para el lote en la primera posición de la secuencia, es el resultado de la suma entre el tiempo de finalización en la máquina 1 y el tiempo de procesamiento total en la máquina 2; de igual forma ocurre para la máquina 3, se calcula mediante la suma del tiempo de finalización en la máquina anterior y el tiempo de procesamiento total en la máquina actual.

Por medio de la ecuación (6) se calculan el tiempo de finalización para los lotes 3, 4 y 1 en las máquinas faltantes (2 y 3); para el caso particular del lote 3 en la máquina 2, se deberá evaluar si al finalizar el lote 3 en la máquina 1 ya se encuentra disponible la máquina 2 o si por el contrario el lote 2 continua en procesamiento.

$$CT_{i,1,j} = CT_{i,1,j-1} + P_{i,1,j}$$

$$CT_{i,k,j} = max\{(CT_{i,k-1,j} - CT_{i,k,j-1}), 0)\} + P_{i,k,j}$$

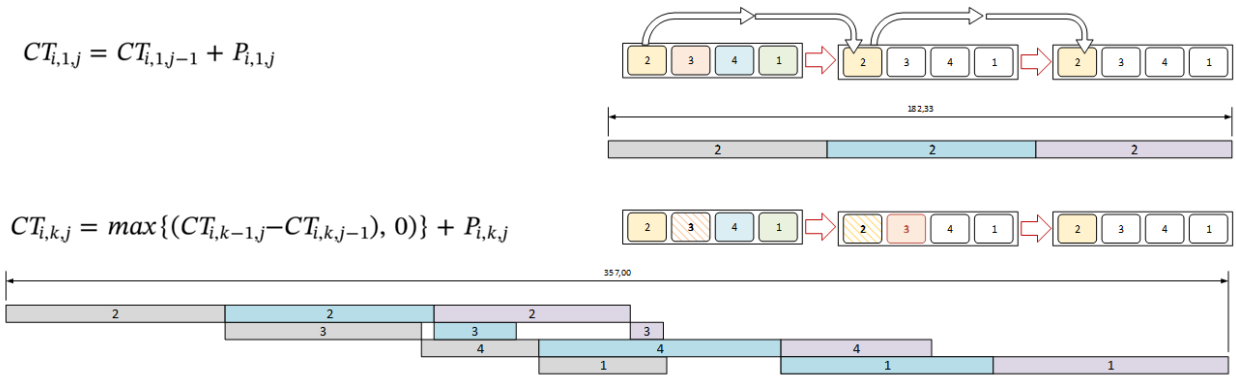
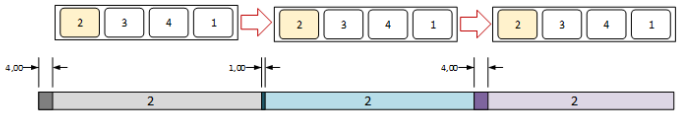


Figura 9. Representación gráfica ecuación (5) y (6) cálculo tiempo de finalización instancia 4*3

A partir del cálculo de los tiempos de finalización y por medio de las ecuaciones (7), (8) y (9) se determina el tiempo de espera para cada uno de los cuatro lotes en las tres máquinas, como se puede observar en la *Figura 10* y de acuerdo con la ecuación (7) el tiempo que espera el lote 2 programado en la primera posición de la secuencia corresponde al tiempo de alistamiento en cada una de las 3 máquinas. La ecuación (8) establece que en la primera máquina los lotes 3, 4 y 1 deberán esperar el procesamiento de sus predecesores, por ejemplo, al lote 4 le corresponde esperar el procesamiento del lote 2 y 3.

$$BW_{i,1,j} = s_{i,j}$$



$$BW_{i,k,1} = s_{i,1} + CT_{i,k-1,1}$$

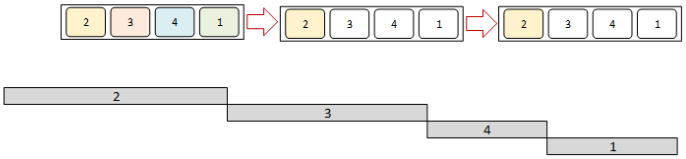


Figura 10. Representación gráfica ecuación (7) y (8) cálculo tiempo de espera instancia 4*3

Adicionalmente el tiempo de alistamiento que deberán esperar los lotes 3, 4 y 1 en las máquinas 2 y 3 se calcula mediante la ecuación (9) en la cual se determina el tiempo entre máquinas que deberá esperar en el caso de que al pasar a la máquina 2 y 3 se encuentren ocupadas. El cálculo del tiempo de espera se determina a partir del tiempo de finalización, por ejemplo, para el caso del lote 3 en la máquina 2 el tiempo de espera corresponde a la diferencia entre el tiempo de finalización de lote 2 en la máquina 2 y el tiempo de finalización del lote 3 en la máquina 1; si la diferencia es negativa, significa que el trabajo 2 ha finalizado lo suficientemente antes para que el lote 2 no tuviese que esperar.

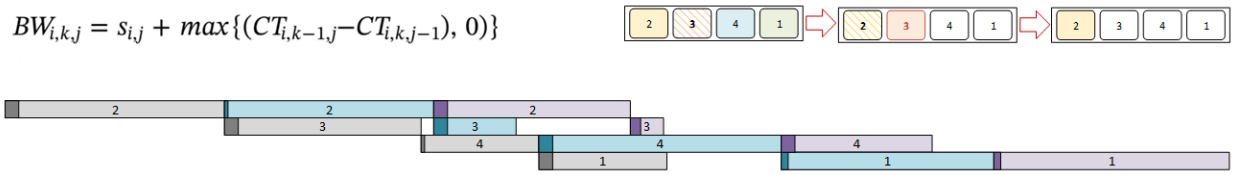


Figura 11. Representación gráfica ecuación (9) cálculo tiempo de finalización instancia 4*3

Una vez calculados los tiempos de finalización y los respectivos tiempos de espera para cada una de las 24 secuencias, por medio de las ecuaciones (11) y (12) se determinan el costo por mantener inventario en espera de ser procesado por una máquina y el costo por mantener inventario en procesamiento; sin embargo, como se mencionó en la explicación del modelo general estos costos solo se harán efectivos cuando el tiempo de finalización del trabajo sea mayor a la fecha de entrega del lote, como se puede apreciar en la ecuación (13).

Adicionalmente se determina el costo total de penalización mediante la ecuación (14), el cual se incurre cuando un lote se encuentra retrasado en su entrega, es decir, cuando el "makespan" se hace superior a la fecha de entrega. Ahora bien, conforme la función objetivo establece, para cada una se las soluciones factibles se calcula la relación entre la sumatoria de los tres (3) costos asociados a la caracterización del problema y el "makespan" para finalmente determinar la solución óptima.

8. Desarrollo e implementación

Después de explicar la forma en que opera el modelo matemático propuesto por Mishra & Shrivastava (2018), cabe aclarar que el modelo matemático no corresponde a un modelo de programación lineal entera "MILP", por el contrario describe el proceso sistemático para la

generación del total de las soluciones factibles, es decir, el modelo se basa en el método denominado fuerza bruta o búsqueda exhaustiva por lo que se considera pertinente la implementación en Matlab por su flexibilidad y usabilidad en relación con la programación.

8.1. Instancias para la validación del modelo matemático

Como es sabido, las metaheurísticas en la mayoría de los casos garantizan soluciones aproximadas, por lo tanto, para validar la efectividad y la precisión del GA, en primera oportunidad se comparan resultados con las soluciones exactas obtenidas por el método fuerza bruta para doce instancias de pequeño tamaño que van desde 8 trabajos 5 máquinas hasta 10 trabajos 20 máquinas.

No obstante, para problemas de tamaño mediano y grande, no se considera pertinente el uso del método fuerza bruta pues la obtención de los resultados emplearía un tiempo computacional no razonable, por el contrario, el GA provee soluciones aproximadas pero lo suficientemente acertadas para el problema objeto de estudio. En la Tabla 4, se presentan las instancias representativas a evaluar para problemas de tamaño pequeño, mediano y grande para dar solución al PFSP:

Tabla 4.

Instancias para el PFSP

Ítem	***Instancias problemas de pequeño tamaño***	***Instancias problemas de mediano-grande tamaño***
1.	8 x 5	20 x 5
2.	8 x 10	20 x 10
3.	8 x 15	20 x 20
4.	8 x 20	30 x 10
5.	9 x 5	30 x 15
6.	9 x 10	50 x 5
7.	9 x 15	50 x 10
8.	9 x 20	50 x 20

Continuación Tabla 4.
Instancias para el PFSP

9.	10 x 5	100 x 5
10.	10 x 10	100 x 10
11.	10 x 15	200 x 20
12.	10 x 20	500 x 20

Adaptado de: Mishra & Shrivastava (2018)

Adicionalmente en la Tabla 5, se presentan los parámetros iniciales de procesamiento que junto con las instancias anteriormente mencionadas conforman los escenarios a partir de los cuales los métodos de solución planteados, Fuerza bruta y Algoritmo Genético, proporcionan la solución al problema de secuenciación de trabajos en un entorno productivo tipo "Flow-shop".

Tabla 5.
Parámetros de procesamiento para la evaluación de E (CPUT)$_s$

Parámetros	***Notación***	***Tamaño***	***Valor***	***Distribución***
Tiempo de procesamiento	p_{ij}	n x m	rand[1,10]	Uniforme
Tiempo de alistamiento	s_{ij}	n x m	rand[1,5]	Uniforme
Tamaño de lote	bs_i	1 x 1	400	-
Costo por mantener inventario	ic	1 x 1	0.5	-
Factor de tardanza	t_f	1 x 1	0.9	-
Costo/penalización	pc_i	1 x n	rand[100,200]	Uniforme
Fecha de entrega	DD_i	1 x n	-	$DDi = n * \sum_{j=1}^{m} P_{ij} * (1 - t_f), 0 < t_f < 1$

Adaptado de: (Mishra & Shrivastava, 2018)

8.2.Desarrollo del modelo matemático en MATLAB

El modelo matemático, fuerza bruta, fue ejecutado en un computador con procesador Intel® Core™ i7-8700 CPU @ de 3,20 GHz, memoria RAM instalada de 8 GB y un sistema operativo Microsoft Windows 10 Pro-Versión 10.0 (Build 17763) de 64 bits. El software empleado fue

Matlab en la versión: 9.6.0.1099231 (R2019a) Update 1, con el solver Global Optimization Toolbox y el solver Optimization Toolbox.

La programación en Matlab del método fuerza bruta comprende tres etapas, las cuales se describen a continuación: en la *Figura 12* se observa la primera etapa, la cual inicia a partir de la definición del tamaño de problema *n x m;* el número de trabajos *n* permite determinar la cantidad de permutaciones que conforman el conjunto de soluciones factibles, seguido de esto se generan cada una de las soluciones factibles a manejar a través de las restricciones para el cálculo de $E(CPUT)_s$.

Si bien la presentación en pantalla de los resultados corresponde a la parte final del proceso es necesario aclarar que Matlab establece que las definiciones de funciones en un script deben aparecer al final del archivo por lo que es necesario mover todas las declaraciones y ubicarlas antes de la primera definición de la función local; es por ello por lo que esta etapa de programación contempla el script para la presentación de resultados.

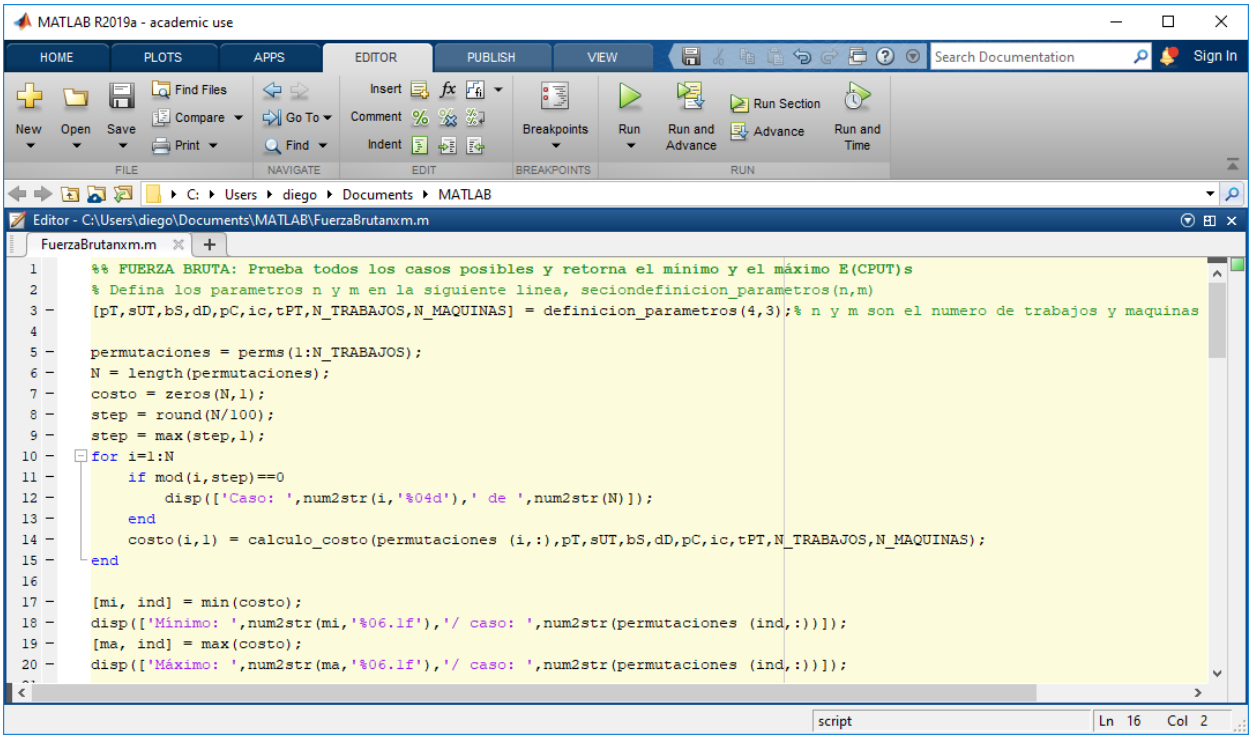


Figura 12. Programación en Matlab, sección definición del tamaño de problema y generación permutaciones

Cabe aclarar que el parámetro *n* no admite valores superiores a 11 trabajos, ya que como se puede observar en la Tabla 6 por el carácter exponencial del problema, un incremento de tan solo un (1) trabajo representaría un aumento porcentual del 1100%, lo que equivale a pasar de 39.916.800 a 479.001.600 soluciones aumentando considerable del tiempo computacional de procesamiento.

Tabla 6.

Conjunto de soluciones factibles se acuerdo al número de trabajos

N	***Numero de soluciones***	***Incremento porcentual***
10	3.628.800	900%
11	39.916.800	1000%
12	479.001.600	1100%
13	6.227.020.800	1200%
14	87.178.291.200	1300%
15	1.307.674.368.000	1400%

Continuando con la programación, la *Figura 13* hace referencia a la generación de parámetros de procesamiento iniciales conforme se establece en la Tabla 5; ahora bien, algunos de los parámetros de procesamiento iniciales tienen una distribución uniforme, sin embargo, estos son fijados reiniciando la semilla de generación de números aleatorios con el fin de realizar la respectiva comparación entre los dos (2) métodos propuestos, Fuerza Bruta y Algoritmo GA.

Adicionalmente, a partir del tiempo de procesamiento y alistamiento se calcula el tiempo de procesamiento total que junto con los demás parámetros de procesamiento iniciales conforman, para afectos prácticos de programación, la función de definición de parámetros.

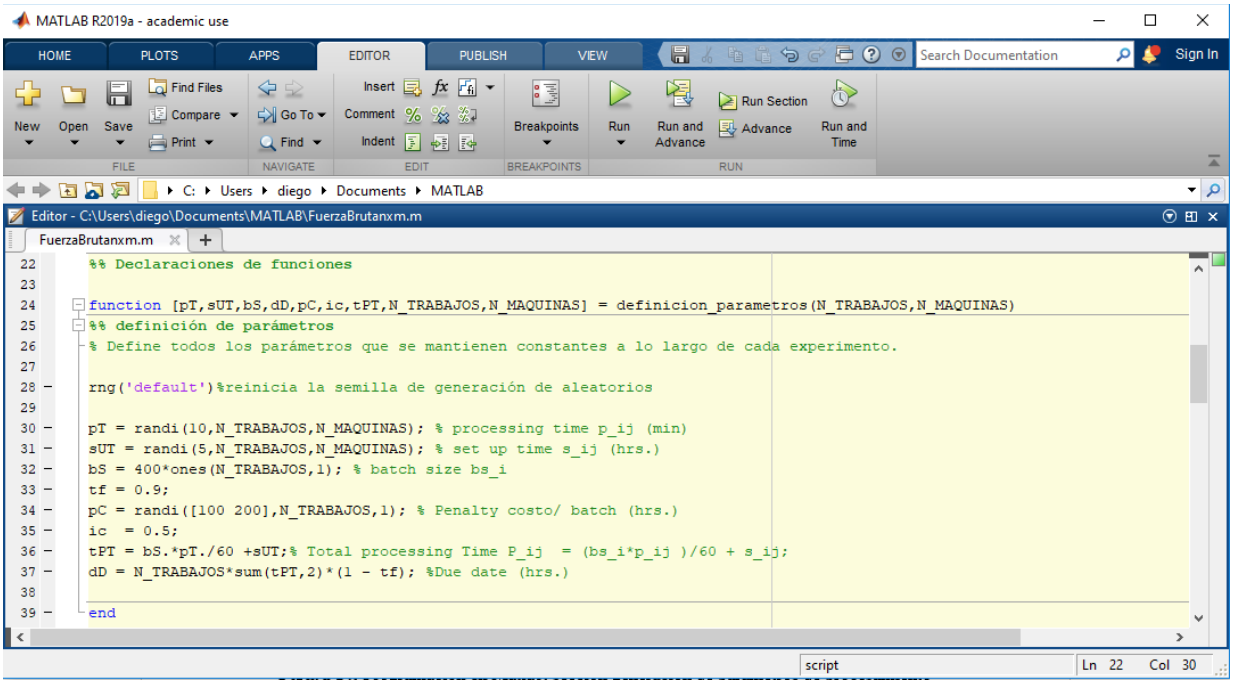


Figura 13. Programación en Matlab, sección generación de parámetros de procesamiento

De acuerdo con la última etapa, en la *Figura 14* se observa la definición de la función objetivo denominada en esta sección como función costo, que comprende los tres costos asociados a la caracterización del presente PFSP, que a su vez son calculados a partir del tiempo de finalización

del lote en cada una de las maquinas, el tiempo de espera en cola antes de una máquina, tiempo de procesamiento y la fecha de entrega del respetivo lote.

El cálculo del tiempo de finalización inicia con la generación de una matriz nula del tamaño necesario para el posterior almacenamiento de los valores producto de la programación de cada una de las restricciones definidas en el modelo matemático; del mismo modo ocurre para las demás variables que intervienen en el cálculo del costo.

Finalmente, los valores de $E(CPUT)_s$ para cada una de las soluciones factibles generadas son almacenados en un vector cero, previamente creado en la primera etapa, para posteriormente seleccionar la solución óptima y proceder con la presentación en pantalla de los resultados.

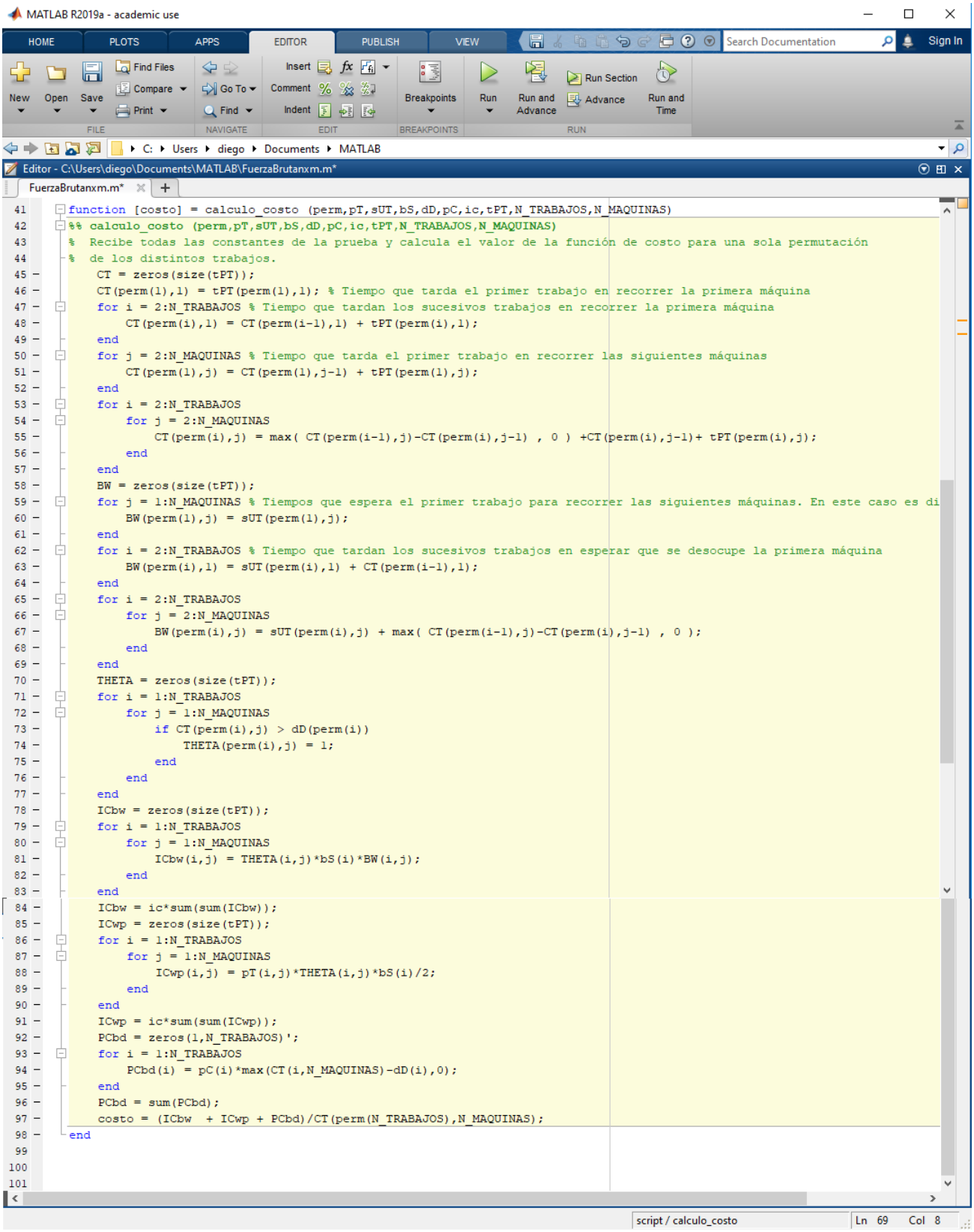


Figura 14. Programación en Matlab, sección restricciones del modelo matemáticos

8.3. Algoritmo Genético (GA)

A medida que se incrementa el tamaño del problema la resolución por métodos exactos de este tipo de problemas se complica considerablemente; sin embargo, trabajando con algoritmos genéticos debe tenerse en cuenta que no es necesario proporcionar un método de resolución, sino un método que permita determinar si una solución es buena o no, y en qué medida (Dorado et al., 2010); por tal motivo se desarrolló una metaheurística basada en el GA con el fin de dar solución al PFSP en instancias grandes, no obstante, el algoritmo también puede ser ejecutado para instancias pequeñas y proporcionar buenos resultados.

Un Algoritmo Genético opera sobre una población de individuos, cada uno de los cuales representa una posible solución al problema que se desea resolver; una generación se obtiene a partir de la anterior por medio de los operadores de reproducción. Inicialmente, se realiza la *selección* de los individuos que van a disponer de oportunidades de reproducirse; luego se genera una descendencia a partir del *cruce* del mismo número de individuos, generalmente dos (2), de la generación anterior.

Si desea optarse por una estrategia *elitista*, los mejores individuos de cada generación se copian siempre en la población temporal, para evitar su pérdida y así posteriormente una vez generados los nuevos individuos se realiza la *mutación* de un individuo, lo que provoca que alguno de sus genes, generalmente uno sólo, varíe su valor de forma aleatoria.

En Matlab, hay dos formas de especificar las opciones para el algoritmo genético, dependiendo de si está utilizando la aplicación de Optimización o llamando a las funciones “ga” o “gamultiobj” en la línea de comandos:

- Utilizando la aplicación de optimización (optimtool), se selecciona el solver de la lista desplegable y se ingresa los respectivos valores de las opciones en el campo de texto.
- Otra forma de hacerlo es la forma utilizada para el desarrollo del presente trabajo, la cual consiste en llamar a “ga” (en Global Optimization Toolbox) en la línea de comando y crear las opciones usando las funciones de optimización de la función, como se muestra en la *Figura 15*:

```
options = optimoptions('ga','Param1', value1, 'Param2', value2, ...);
% or
options = optimoptions('gamultiobj','Param1', value1, 'Param2', value2, ...);
```

Figura 15. Comando Algoritmo Genético en Matlab

8.3.1. Parámetros de entrada. Los parámetros iniciales para la ejecución de algoritmo fueron adaptados de Chen, Pan, & Lin (2008), dichos parámetros fueron propuestos con base en los resultados experimentales presentados en sus investigaciones relacionada con el PFSP. De acuerdo con la Tabla 7 para el algoritmo genético se establece un tamaño de la población de 50, 200 y 400 para problemas de tamaño pequeño, mediano y grande respectivamente, así mismo establece que el 80% de la población es cruzado y tan solo el 5% presenta mutación; por otro lado, un número de generaciones permitido de 100 hace referencia cantidad de generaciones limite sin encontrar una mejor solución.

Tabla 7.

Parámetros iniciales para la ejecución del algoritmo

Parámetro	***Tamaño del problema***		
	Pequeño	***Mediano***	***Grande***
Tamaño de la población	50	200	400
Probabilidad de cruce	0,8	0,8	0,8
Probabilidad de mutación	0,5	0,5	0,5
Numero de generaciones permitido	100	100	100

8.3.2. Instancias planteadas. El problema objeto de estudio es resuelto mediante la metaheurística propuesta en varios de los casos previamente utilizados en el modelo exacto, las instancias van desde 8 trabajos y 5 máquinas hasta 500 trabajos y 20 máquinas, agrupados en categorías de acuerdo el tamaño del problema; para problemas de tamaño pequeño las instancias van desde 8 trabajos 5 máquinas hasta 10 trabajos 20 máquinas y para problemas de tamaño mediano-grande las instancias van desde 20 trabajos 5 máquinas hasta 500 trabajos 20 máquinas como se pudo observar en la sección anterior mediante la Tabla 4.

8.3.3. Representación de la solución. Una solución hace referencia a un programa o secuencia de trabajos representado como un vector n-dimensional:

$$P_l = (\, p_{i,1},\, p_{i,2}, \ldots, p_{i,n}),$$

Donde:

$p_{i,l}$ denota el rango o la prioridad del trabajo i, en una secuencia l, n es el número total de trabajos y l ε L = {1, 2,… NP} donde NP es el tamaño de la población (Mishra & Shrivastava, 2018).

En otros términos, la solución corresponde a un vector fila que contiene una permutación de los enteros de 1 a *n*, por tanto, la longitud del vector depende del número de trabajos a programar para ser procesados. Por otro lado, la prioridad para determinado trabajo hace referencia a la posición

asignada dentro de la secuencia de programación, así aquellos trabajos asignados a las primeras posiciones minimizan el tiempo que espera determinado trabajo antes de una máquina, permitiendo el flujo continuo de procesamiento.

En la *Figura 16* se observa el vector solución para un problema ejemplo de tamaño *8 x m*, la solucion establece el orden en que deberan procesarse los trabajos, asi los trabajos número 8, 1, 3 corresponden al primer, segundo y tercer lanzamiento, y así sucesivamente hasta finalizar con el trabajo número 7.

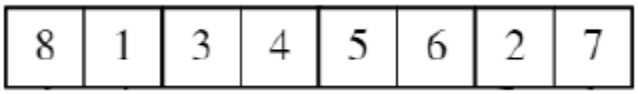


Figura 16. Ejemplo de representación de la solución

8.3.4. Función objetivo. Hay diferentes criterios empleados en la formulación de la función objetivo, entre los más populares de ellos están el "makespan" o tiempo de finalización y el tiempo medio de flujo, en esta sección del presente estudio se emplea la función objetivo del modelo matemático propuesto en la sección anterior, donde $E(CPUT)_s$ contempla el costo total por unidad de tiempo de programación (makespan), es decir, expresa el cociente de la sumatoria del costo total por mantenimiento de inventario para los lotes esperando en la cola antes de las máquinas, el costo total por mantenimiento de inventario de los lotes en procesamiento y costo total de penalización de lotes retrasados más allá de su fecha de entrega entre el "makespan".

8.3.5. Desarrollo del algoritmo. En un GA para cada espacio de solución, se encuentra la mejor solución y, al aplicar algunos operadores de GA, se exploran nuevos espacios de solución,

lo que conduce a mejores resultados. A continuación, se describe el desarrollo del algoritmo en función de los operadores que lo componen:

8.3.5.1. Generación de la población inicial. La población inicial juega un rol importante en la calidad de la solución del GA, por ello existen heurísticas en la literatura como una buena alternativa para construir la población inicial, sin embargo, la mayoría de los GA asumen que la población inicial se elige completamente al azar; por tal motivo el proceso de inicialización parte de una región factible de soluciones, las cuales son generadas aleatoriamente.

Los individuos de la población se generan aleatoriamente de acuerdo con las características anteriormente descritas en el numeral 8.3.3, principalmente que corresponda a un vector de fila que contenga una permutación aleatoria de los enteros de 1 a *n*. y además, bajo los parámetros presentados en la Tabla 7 donde se establece el tamaño de población en conformidad con el tamaño del problema *n x m*.

El número de soluciones iniciales que comprenden la población se denomina tamaño de la población "`PopulationSize`", el cual especifica cuántos individuos hay en cada generación; con un tamaño de población grande, la probabilidad de encontrar el óptimo global se hace mayor, no obstante, un tamaño de población grande también provoca que el algoritmo se ejecute más lentamente.

Por otro lado, es necesario definir el tipo de población "`PopulationType`", el cual especifica el tipo de entrada a la función de aptitud. Los tipos y sus restricciones son: Cadena de bits y Vector

doble. Cadena de bits "`cadena de bits`" se usa si los individuos en la población tienen componente binario, es decir, son 0 o 1, no obstante, acorde con el desarrollo de la presente investigación se usó Vector doble "`doubleVector`", el cual se usa si los individuos en la población tienen doble tipo, principalmente se utiliza para la programación de enteros mixtos y corresponde al valor predeterminado por Matlab.

8.3.5.2. Selección. El objetivo del procedimiento de selección es extraer de la población un grupo de padres para generar los descendientes, este conjunto también se conoce como el grupo de apareamiento. Para especificar el tipo de selección que utiliza el algoritmo se define en el campo Función de selección "`SelectionFcn`" en el panel Opciones de selección.

Existen diversos métodos de selección, sin embargo, no es posible demostrar que uno sea más efectivo que otro, por tal motivo se empleó la función de selección por defecto de "ga" en Matlab, denominada Estocástico uniforme "`selectionstochunif`", la cual establece una línea en la que cada padre corresponde a una sección de la línea de longitud proporcional a su valor escalado; el algoritmo se mueve a lo largo de la línea en pasos de igual tamaño por lo que en cada paso, el algoritmo asigna un padre de la sección en la que aterriza. El primer paso es un número aleatorio uniforme menor que el tamaño del paso (MathWorks, s.f).

8.3.5.3. Reproducción. Las opciones de reproducción especifican cómo el algoritmo genético crea hijos para la próxima generación. Estas son: el elitismo y el cruce

- *Elitismo.* El elitismo es un caso particular del operador de reproducción, consistente en copiar siempre al mejor o en su defecto a los mejores individuos de una generación en la siguiente generación. En Matlab, el recuento de élite "`EliteCount`" especifica el número de individuos que tienen la garantía de sobrevivir hasta la próxima generación, este debe ser un entero positivo menor o igual que el tamaño de la población o como el usado en este trabajo, un porcentaje de la población inicial.

- *Cruce.* Este mecanismo de diversificación permite que el algoritmo GA examine regiones no visitadas y genere soluciones que difieran de varias maneras significativas de las que se vieron anteriormente. En Matlab, La función de cruce "`CrossoverFcn`" especifica la función que realiza el cruce; se usó Scattered "`crossoverscattered`", la cual crea un vector binario aleatorio y selecciona los genes donde el vector es un 1 del primer padre, y los genes donde el vector es un 0 del segundo padre, y combina los genes para formar el hijo. Por ejemplo, si p1 y p2 son los padres.

Ejemplo:

```
p1 = [a b c d e f g h]
p2 = [1 2 3 4 5 6 7 8]
```

y el vector binario es [1 1 0 0 1 0 0 0], la función da como resultado el siguiente hijo:

```
child1 = [a b 3 4 e 6 7 8]
```

Adicionalmente fue necesario definir la fracción de cruce "`CrossoverFraction`" o probabilidad de cruce, la cual encarga de especificar la fracción de la próxima generación, distinta de los hijos de élite, que se reproduce mediante el cruce. Para el desarrollo de la presente investigación dicha

probabilidad es de 0.8. En la Figura 17 se explica de forma ilustrativa como sería el cruce de individuos conforme al vector solución planteado para en la sección 8.3.3.

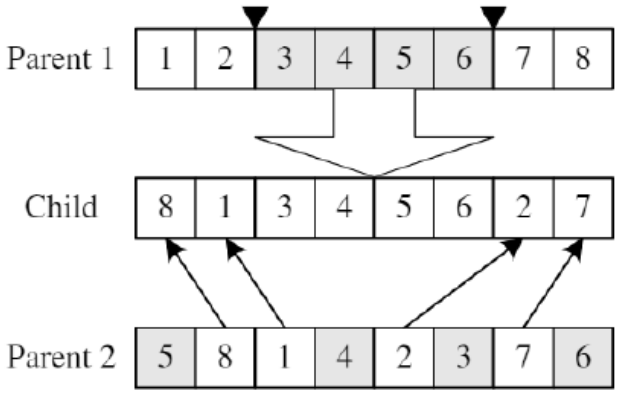


Figura 17. Ejemplo de cruce de dos puntos para el vector solución.

8.3.5.4. Mutación. Las opciones de mutación especifican cómo el algoritmo genético hace pequeños cambios aleatorios en los individuos de la población para crear hijos con mutación. La mutación proporciona diversidad genética y permite que el algoritmo genético busque un espacio más amplio. En Matlab, se definió La opción de mutación predeterminada, Gaussian "`mutationgaussian`" en el campo Función de mutación "`MutationFcn`", esta opción agrega un número aleatorio, o mutación, elegida de una distribución Gaussian, a cada entrada del vector hijo. Normalmente, la cantidad de mutación, que es proporcional a la desviación estándar de la distribución, disminuye con cada nueva generación (MathWorks, s.f.).

8.3.5.5. Criterio de parada. El algoritmo GA continúa procesando hasta que se logre el criterio de detención establecido por los criterios utilizados; este estudio utiliza un número fijo de generaciones como condición de terminación, el cual se define mediante la opción "`MaxGenerations`". Complementariamente, "`MaxStallGenerations`" conlleva a que el algoritmo

se detenga si el cambio relativo promedio en el mejor valor de la función objetivo no varía con el pasar de determinado número de generaciones.

Otra alternativa programada en el código como criterio de para es "`MaxTime`", mediante la cual el algoritmo se detiene después de ejecutarse después de determinados segundos; este límite se aplica después de cada iteración, por lo que "ga" puede exceder el límite cuando una iteración toma un tiempo sustancial. No obstante, si bien puede ser una herramienta práctica, no fue considerada en el presente estudio para así desarrollar de forma correcta el comparativo entre métodos de solución al PFSP.

8.3.6. Visualización del algoritmo GA. "ga" puede generar una o más funciones de trazado a través de un argumento en "`PlotFCN`". Esta característica es útil para visualizar el rendimiento del solucionador en tiempo de ejecución. En la *Figura 18*, se observan tres funciones de trazado: La primera función de trazado es "`gaplotstopping`", que traza el porcentaje de criterios de detención cumplidos, la segunda función de trazado es "`gaplotbestf`", la cual representa el mejor puntaje promedio de la población en cada generación y la tercera "`gaplotrange`" traza la media y el rango de las puntuaciones.

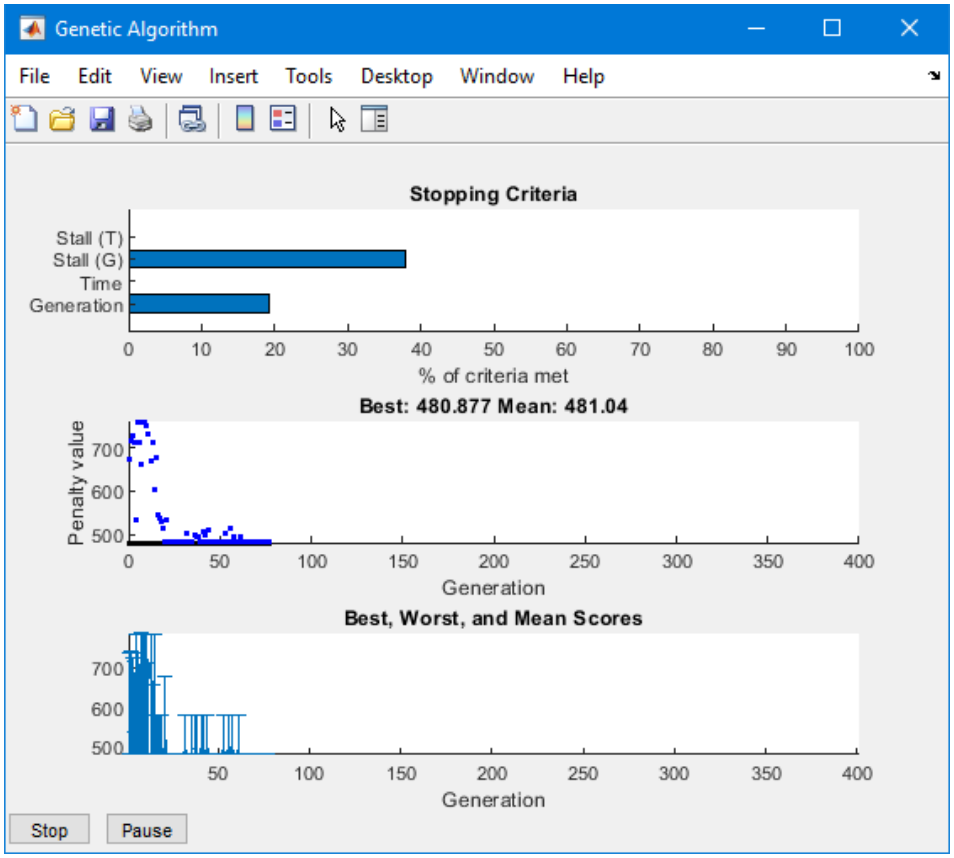


Figura 18. Visualización del algoritmo GA

Una vez definidos cada uno de los operadores que componen el algoritmo GA propuesto, a continuación, en la *Figura 18* puede apreciarse de forma más clara, el diagrama de flujo del proceso de ejecución del algoritmo GA, así como la las partes del proceso en la cual interviene cada operador y como se interrelaciona con los demás.

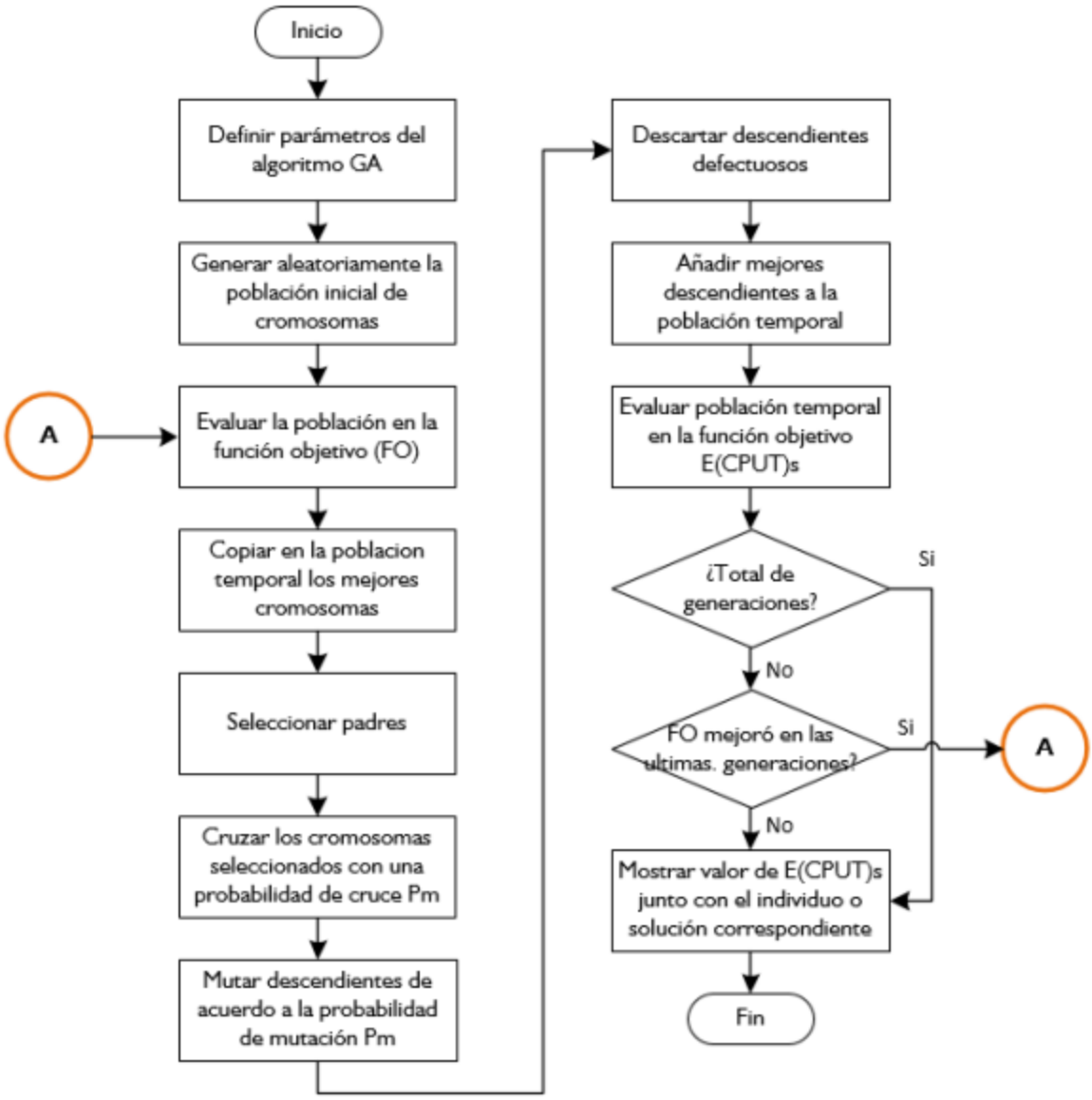


Figura 19. Diagrama de flujo ejecución del algoritmo GA

9. Diseño de Experimentos

Una vez desarrollada la metaheurística basada en el algoritmo GA se procede a comprobar su funcionalidad y consistencia haciendo uso de experimentos numéricos; un diseño de experimentos consiste en un cambio en las condiciones de operación de un sistema con el objetivo de medir el efecto del cambio sobre una o varias propiedades del producto, resultado o variable respuesta (Gutiérrez Pulido & De la Vara Salazar, 2008).

Minitab©[5] ofrece cinco tipos de diseños: diseños de cribado, diseños factoriales, diseños de superficie de respuesta, diseños de mezcla y diseños de Taguchi, no obstante, para el presente estudio se llevó a cabo un diseño factorial 2^3 con cinco réplicas*;* los factores considerados son el tamaño de la población, la probabilidad de cruce y el número máximo de generaciones permitido que junto con los respectivos niveles se muestran en la Tabla 8.

Tabla 8.
Configuración niveles por cada Factor

Factores		***Niveles***	
		Bajo	***Alto***
Tamaño de la población	(A)	50	300
Probabilidad de cruce	(B)	0,8	0,9
Numero de generaciones permitido	(C)	50	100

Asimismo, conforme al diseño factorial propuesto, en la Tabla 9 se muestran los tratamientos correspondientes para el diseño factorial 2^3; este diseño es aplicado a 5 instancias de diferente

[5] Minitab es un programa de computadora diseñado para ejecutar funciones estadísticas básicas y avanzadas.

tamaño, las cuales corresponden a una de las más grandes, una de las más pequeñas y tres intermedias de las instancias propuestas en la sección 8.1.8.1

Tabla 9.

Tratamientos para el diseño factorial 2^3

Etiqueta	***A***	***B***	***C***
-1	0	0	0
a	1	0	0
b	0	1	0
ab	1	1	0
c	0	0	1
ac	1	0	1
bc	0	1	1
abc	1	1	1

A partir del diseño experimental, se efectúa un análisis de varianza ANOVA, el cual permite identificar qué factores tienen un efecto significativo sobre la variable respuesta $E(CPUT)_s$ y cuál es el tratamiento con el que se obtienen los mejores resultados para esta variable.

9.1. Instancia N8-M5 problemas de tamaño pequeño

Con base en los factores y niveles establecidos anteriormente, se desarrolla el diseño de experimentos como se puede observar en la

Tabla 10, el cual está compuesto por tres réplicas de la función objetivo para cada uno de los tratamientos que lo componen.

Tabla 10.

Estructura para el diseño de experimentos Instancia N8-M5

Tratamientos	***Réplicas***		
	1	***2***	***3***
-1	755,18	756,96	704,9
a	739,29	751,8	731,35
b	826,24	742,11	749,95
ab	719,62	728,24	766,16
c	768,83	660,26	867,06
ac	715,3	711,99	690,85
bc	747,1	747,19	745,8
abc	776,5	660,26	769,38

Los datos son procesados haciendo uso del software Minitab© 19, el cual proporciona las herramientas necesarias para el análisis factorial como pruebas de normalidad, independencia, entre otras, además del análisis de varianza ANOVA.

Ahora bien, analizando los residuales mediante la *Figura 20*, es posible inferir que los residuos están distribuidos normalmente conforme al valor p obtenido a través de la Prueba de Kolmogorov-Smirnov. Asimismo, a partir del histograma de los residuos se observa que los datos son aparentemente simétricos, no obstante, incluyen valores atípicos. La gráfica de residuos vs. ajustes permite comprobar el supuesto de que los residuos tienen una varianza constante y finalmente la gráfica de residuos vs. orden permite comprobar el supuesto de que los residuos no están correlacionados entre sí, pues no es posible identificar patrón alguno.

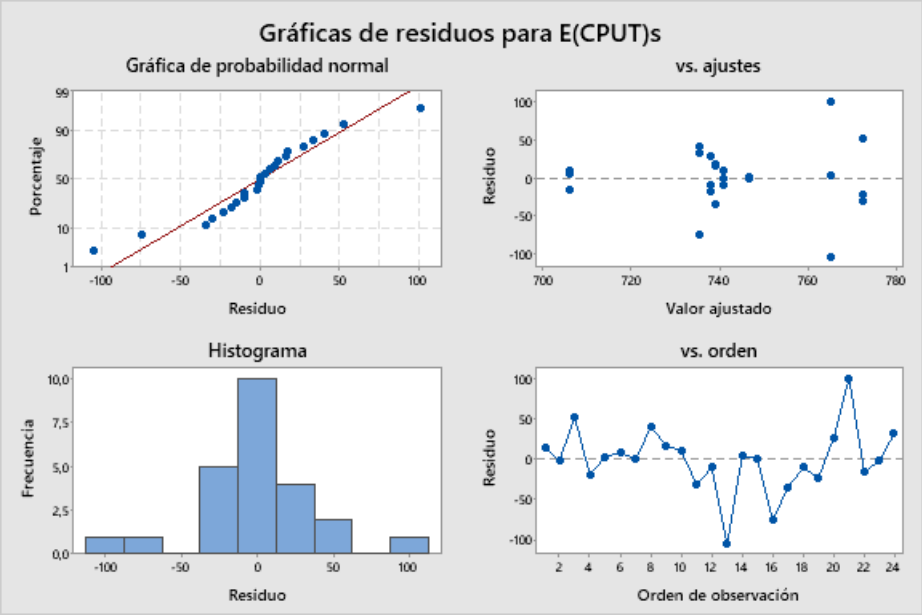


Figura 20. Gráfica de residuos para $E(CPUT)_s$. Instancia N8-M5

El análisis de varianza ANOVA, mostrado en la Tabla 11 permite identificar si alguno de los factores o combinaciones de factores tienen un efecto significativo sobre la variable respuesta $E(CPUT)_s$ y en qué medida lo hace, para ello el Valor-p debe ser menor o igual al nivel de significancia propuesto del 0,05.

Tabla 11.

Análisis de Varianza. Instancia N8-M5

Fuente	***GL***	***SC Ajust.***	***MC Ajust.***	***Valor F***	***Valor p***
Modelo	7	8609,9	1229,98	0,52	0,806
Lineal	3	5190,6	1730,21	0,73	0,547
Tamaño de la población	1	4025,9	4025,90	1,71	0,210
Probabilidad de cruce	1	648,8	648,75	0,27	0,607
Numero de generaciones	1	516,0	515,97	0,22	0,646
Interacciones de 2 términos	3	736,6	245,53	0,10	0,956
Tamaño de la población*Probabilidad de cruce	1	49,2	49,25	0,02	0,887
Tamaño de la población*N° de generaciones	1	532,8	532,80	0,23	0,641
Probabilidad de cruce*N° de generaciones	1	154,5	154,53	0,07	0,801
Interacciones de 3 términos	1	2682,7	2682,67	1,14	0,302

Continuación Tabla 11.
Análisis de Varianza. Instancia N8-M5

Fuente	***GL***	***SC Ajust.***	***MC Ajust.***	***Valor F***	***Valor p***
Tamaño de la población*Probabilidad de cruce*N° de generaciones	1	2682,7	2682,67	1,14	0,302
Error	16	37749,5	2359,35		
Total	23	46359,4			

De forma complementaria, en la *Figura 21* se observa que para instancias pequeñas ninguno de los factores en forma individual, ni sus interacciones de dos y tres términos tienen un efecto significativo en la variable respuesta $E(CPUT)_s$

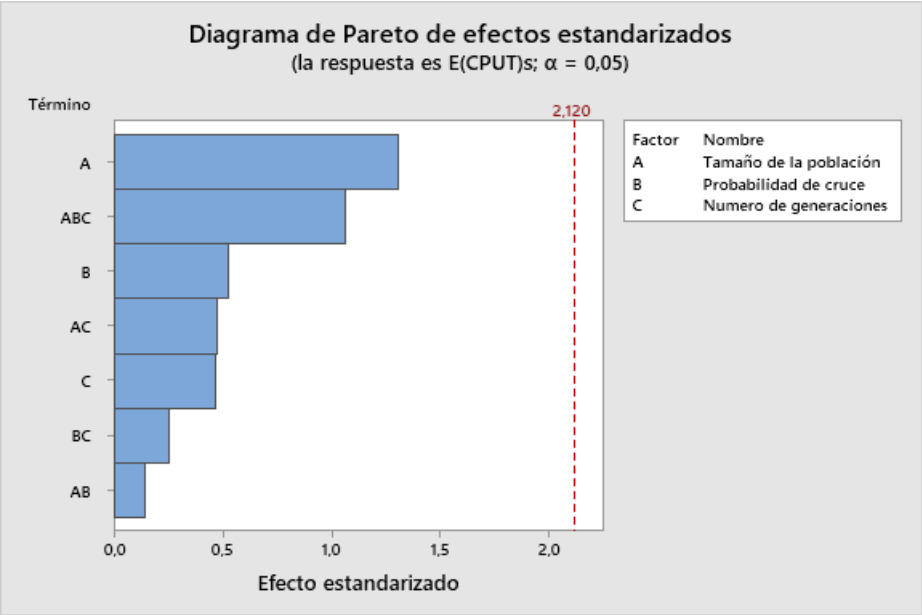


Figura 21. Diagrama de Pareto de efectos estandarizados. Instancia N8-M5

Cabe resaltar que el tamaño de la población corresponde al principal factor, en términos de efectos positivos aunque insignificantes sobre la variable respuesta $E(CPUT)_s$ como se puede apreciar en la *Figura 22*.

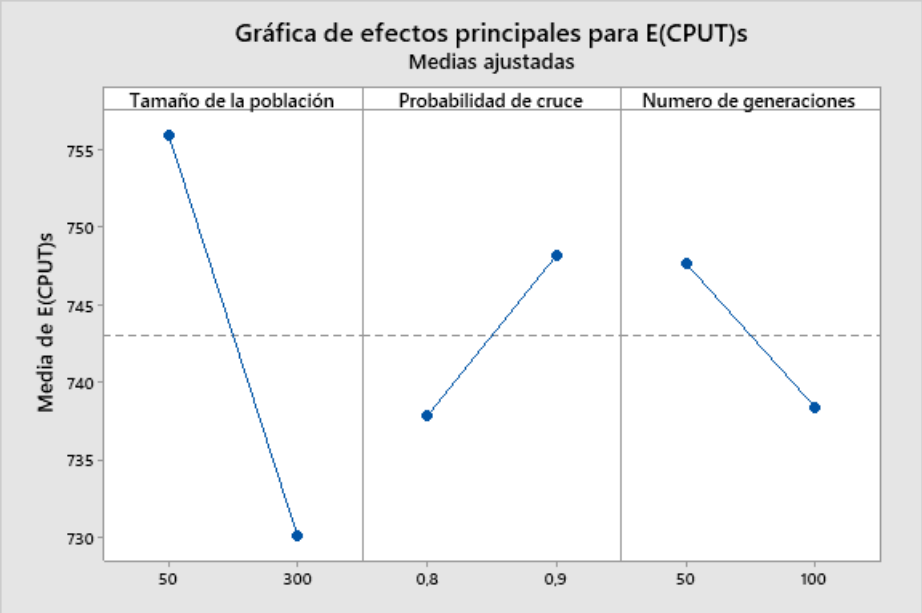


Figura 22. Gráfica de efectos principales para *E(CPUT)*$_S$. Instancia N8-M5

9.2. Instancia N10-M10 problemas de tamaño pequeño

Del mismo modo para la Instancia N10-M10, perteneciente a la categoría de problemas de tamaño pequeño, se plantea el diseño experimental mostrado a continuación en la Tabla 12:

Tabla 12.

Estructura para el diseño de experimentos Instancia N10-M10

Tratamientos	***Replicas***		
	1	***2***	***3***
-1	698,23	686,42	712,98
a	651,18	675,42	668,42
b	668,42	743,79	699,98
ab	654,91	691,42	657,05
c	616,79	673,84	619,16
ac	684,32	664,81	660,15
bc	685,42	721,22	682,81
abc	685,84	673,16	613,12

De acuerdo con *Figura 23*, es posible inferir que los residuos están distribuidos normalmente. Así, mismo que los datos son aparentemente simétricos con un sesgamiento a la izquierda, no obstante, incluyen valores atípicos. Adicionalmente se comprueba el supuesto de que los residuos tienen una varianza constante y que no están del todo correlacionados entre sí, salvo aquellos en lo que se presenta un patrón de descenso entre las observaciones 16 a la 20.

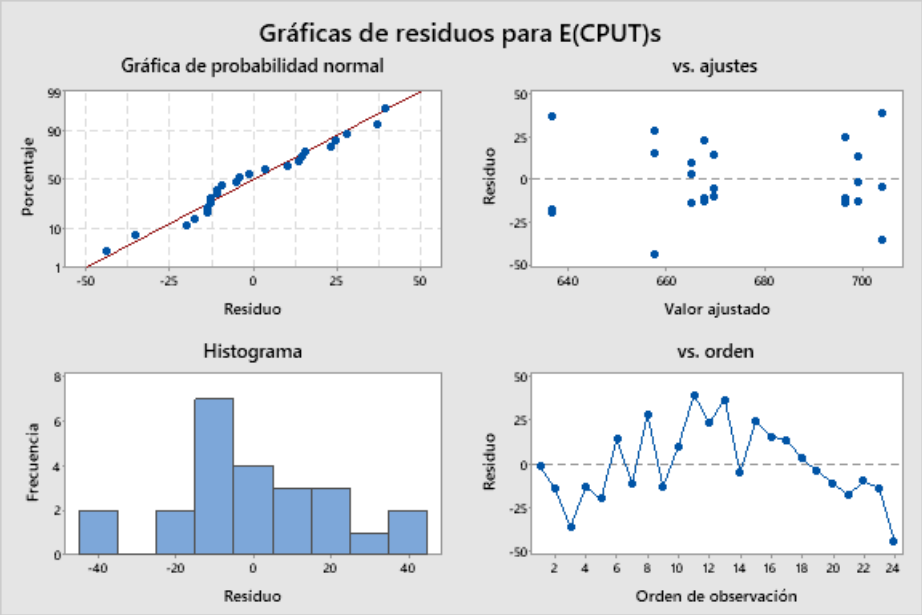


Figura 23. Gráfica de residuos para $E(CPUT)_s$. Instancia N10-M10

Por otro lado el análisis de varianza ANOVA, mostrado en la Tabla 13, complementado mediante la *Figura 24*. permite corroborar que, al igual que en la instancia N8-M5, ninguno de los factores en forma individual, ni sus interacciones de dos y tres términos tienen un efecto significativo en la variable respuesta $E(CPUT)_s$

Tabla 13.

Análisis de Varianza. Instancia N10-M10

Fuente	***GL***	***SC Ajust.***	***MC Ajust.***	***Valor F***	***Valor p***
Modelo	7	11566,2	1652,3	2,47	0,064
Lineal	3	5488,2	1829,4	2,73	0,078
Tamaño de la población	1	2190,0	2190,0	3,27	0,089
Probabilidad de cruce	1	1140,2	1140,2	1,70	0,211
Numero de generaciones	1	2158,0	2158,0	3,22	0,092
Interacciones de 2 términos	3	4229,6	1409,9	2,10	0,140
Tamaño de la población*Probabilidad de cruce	1	2072,4	2072,4	3,09	0,098
Tamaño de la población*N° de generaciones	1	1561,4	1561,4	2,33	0,146
Probabilidad de cruce*N° de generaciones	1	595,8	595,8	0,89	0,360
Interacciones de 3 términos	1	1848,4	1848,4	2,76	0,116
Tamaño de la población*Probabilidad de cruce*N° de generaciones	1	1848,4	1848,4	2,76	0,116
Error	16	10721,8	670,1		
Total	23	22288,0			

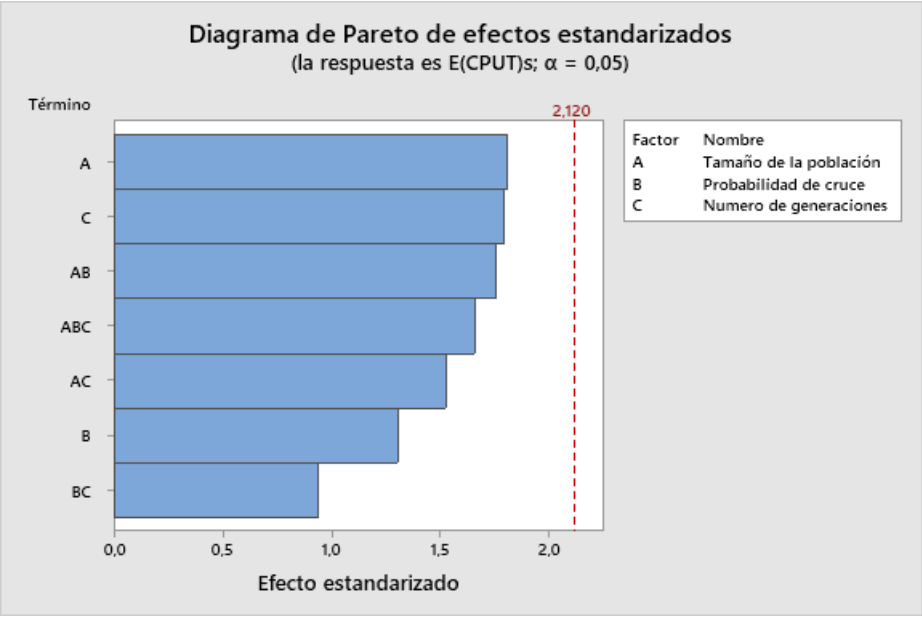


Figura 24. Diagrama de Pareto de efectos estandarizados. Instancia N10-M10

Adicionalmente, basados en la *Figura 25* se observa que conforme el tamaño del problema aumenta, el Factor Numero de generaciones empieza a tomar relevancia, aunque su efecto no sea considerado como significativo en la variable respuesta.

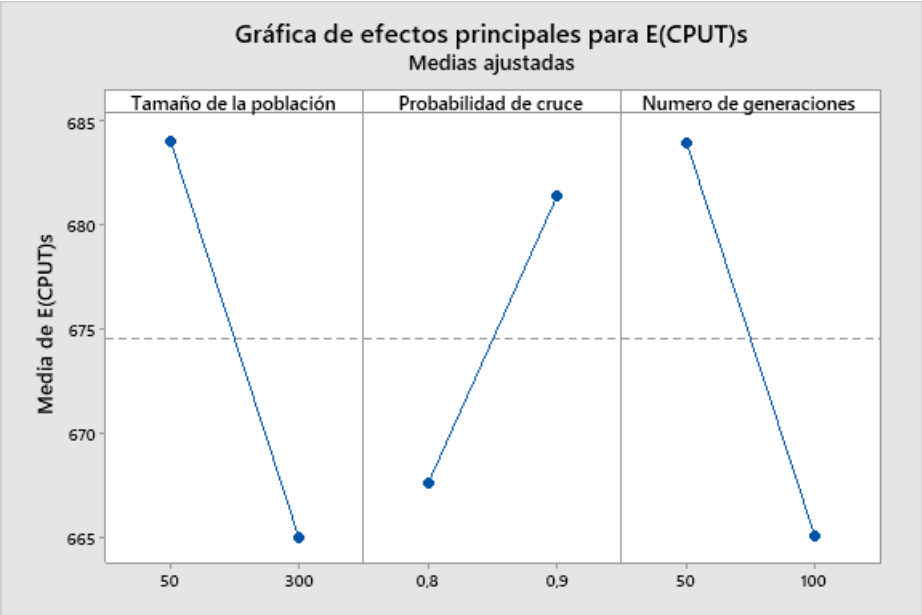


Figura 25. Gráfica de efectos principales para $E(CPUT)_s$. Instancia N10-M10

9.3. Instancia N20-M20 problemas de tamaño mediano

El diseño experimental mostrado a continuación en la Tabla 14 corresponde al empleado para la Instancia N20-M20, la cual pertenece a la categoría de los problemas de tamaño mediano

Tabla 14.
Estructura para el diseño de experimentos Instancia N20-M20

Tratamientos	***Replicas***		
	1	***2***	***3***
-1	168	151,58	181,41
a	137,15	124,58	139,06
b	178,34	156,93	145,99
ab	144,34	133,06	118,27
c	162,36	153,94	179,61

Continuación Tabla 14.
Estructura para el diseño de experimentos Instancia N20-M20

Tratamientos	***Replicas***		
	1	***2***	***3***
ac	127,66	107,59	123,87
bc	139,71	159,56	121,94
abc	95,92	146,73	110,95

A partir de los residuos y conforme a la *Figura 26*, es posible inferir que estos están distribuidos normalmente de acuerdo con la Prueba de normalidad realizada, donde el valor-p obtenido fue superior al nivel de significancia. Adicionalmente se observa que los datos son aparentemente simétricos con un sesgamiento a la izquierda e incluyen valores atípicos. Por otro lado, se comprueba el supuesto de que los residuos tienen una varianza constante y que algunos están correlacionados entre sí.

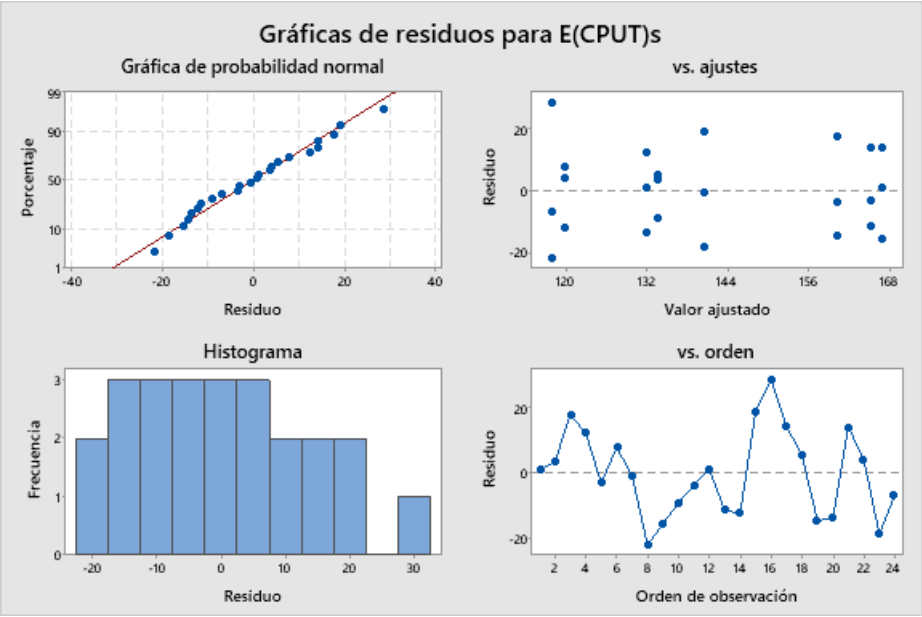


Figura 26. Gráfica de residuos para $E(CPUT)_s$. Instancia N20-M20

Contrariamente a las instancias para problemas de tamaño pequeño anteriormente analizadas, el análisis de varianza para la instancia analizada presentado en la Tabla 15, se identifica en la *Figura 27* que un nivel de confianza del 95% el principal factor que influye significativamente en la función objetivo es el tamaño de la población inicial con un valor-p inferior al nivel de significancia y cercano a cero.

Tabla 15.

Análisis de Varianza. Instancia N20-M20

Fuente	***GL***	***SC Ajust.***	***MC Ajust.***	***Valor F***	***Valor p***
Modelo	7	8285,9	1183,70	4,63	0,005
Lineal	3	7727,1	2575,70	10,07	0,001
Tamaño de la población	1	6343,7	6343,68	24,79	0,000
Probabilidad de cruce	1	460,0	459,99	1,80	0,199
Numero de generaciones	1	923,4	923,43	3,61	0,076
Interacciones de 2 términos	3	434,7	144,90	0,57	0,645
*Tamaño de la población*Probabilidad de cruce*	*1*	*292,5*	*292,53*	*1,14*	*0,301*
Tamaño de la población*N° de generaciones	1	14,4	14,43	0,06	0,815
Probabilidad de cruce*N° de generaciones	1	127,7	127,74	0,50	0,490
Interacciones de 3 términos	1	124,1	124,08	0,48	0,496
Tamaño de la población*Probabilidad de cruce*N° de generaciones	1	124,1	124,08	0,48	0,496
Error	16	4094,4	255,90		
Total	23	12380,3			

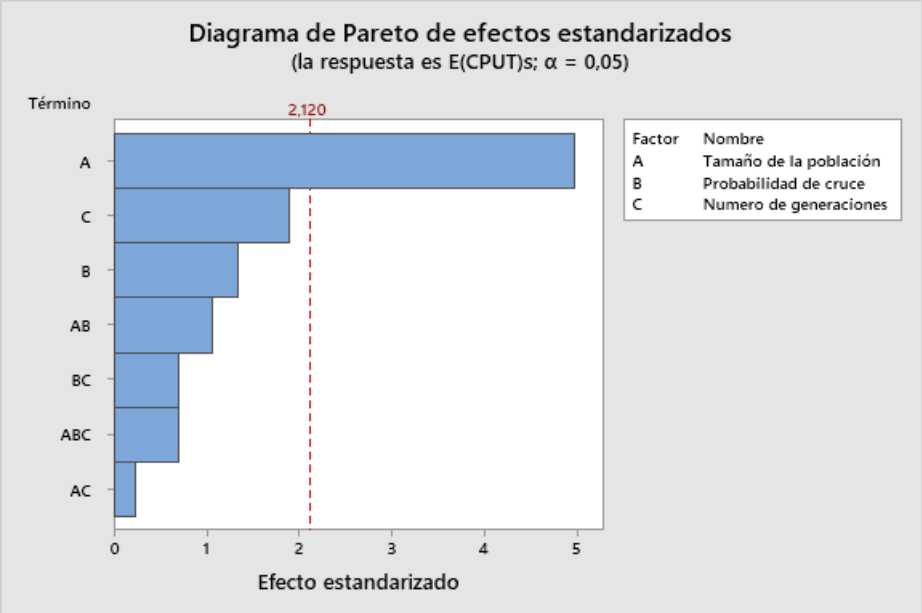


Figura 27. Diagrama de Pareto de efectos estandarizados. Instancia N20-M20

En la *Figura 28* se puede apreciar que el parámetro que más incidencia tiene sobre la variable respuesta es el tamaño de la población el cual, en el nivel alto ofrece una mejor solución, no obstante, la probabilidad de cruce al igual que el número de generaciones también ofrecen una mejor solución en el nivel alto.

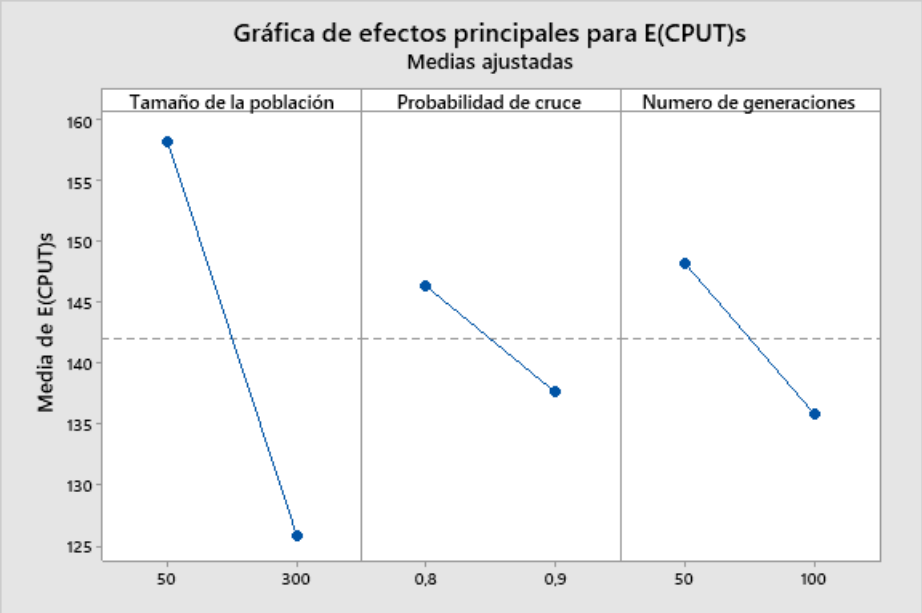


Figura 28. Gráfica de efectos principales para $E(CPUT)_s$. Instancia N20-M20

9.4. Instancia N50-M10 problemas de tamaño grande

A continuación en la Tabla 16, se muestra el diseño experimental empleado para la Instancia representativa de problemas de tamaño grande N50-M10:

Tabla 16.

Estructura para el diseño de experimentos Instancia N50-M10

Tratamientos	***Replicas***		
	1	***2***	***3***
-1	1304,62	1203,73	1083,66
a	1175,96	946,07	832,55
b	1050,38	1239,79	1275,75
ab	1071,62	1106,66	918,52
c	1351,47	1163,88	1111,73
ac	898,59	1100,21	1152,03
bc	1261,2	1300,15	1116,68
abc	1014,47	809,63	965,42

La *Figura 29* de por sí sola no permite dar cuenta si los datos están distribuidos normalmente, por lo que mediante la Prueba de normalidad Kolmogórov-Smirnov se determina el valor p (0,055) y se logra corroborar la normalidad de los residuos. Complementariamente, se aprecia que la distribución tiene un comportamiento bimodal. Por otro lado, se comprueba el supuesto de que los residuos tienen una varianza constante y que los residuos no están correlacionados entre sí, pues no es posible identificar patrón alguno.

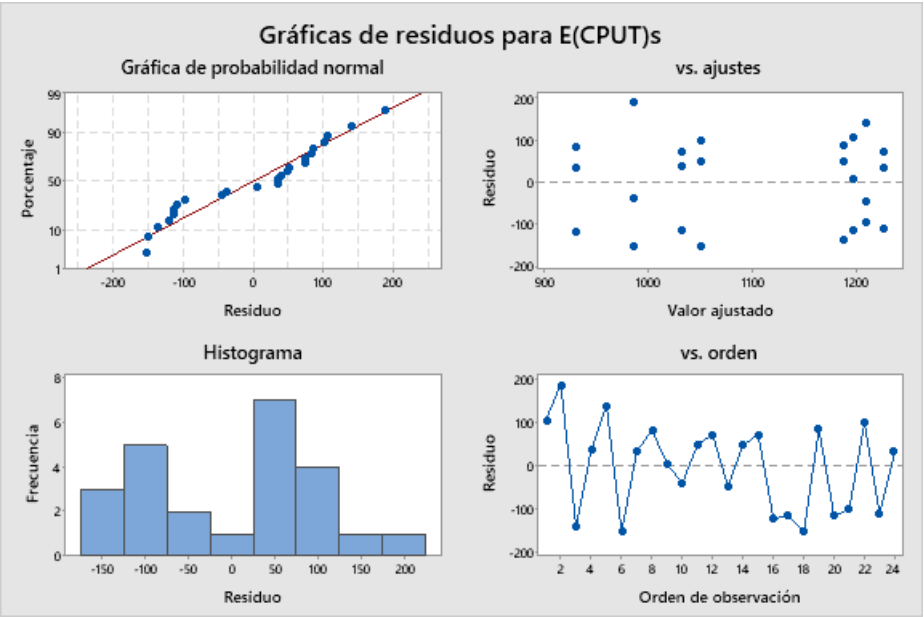


Figura 29. Gráfica de residuos para $E(CPUT)_s$. Instancia N50-M10

Por otro lado el análisis de varianza ANOVA, mostrado en la Tabla 17, complementado de manera gráfica mediante la *Figura 30* permite da cuenta que, de lejos, el principal factor cuyos efectos son altamente significativos en $E(CPUT)_s$ es el tamaño de la población inicial con un valor-p del 0,01.

Tabla 17.

Análisis de Varianza. Instancia N50-M10

Fuente	***GL***	***SC Ajust.***	***MC Ajust.***	***Valor F***	***Valor p***
Modelo	7	282981	40426	2,65	0,051
Lineal	3	256100	85367	5,59	0,008
Tamaño de la población	1	254474	254474	16,67	0,001
Probabilidad de cruce	1	1572	1572	0,10	0,752
Numero de generaciones	1	54	54	0,00	0,953
Interacciones de 2 términos	3	12837	4279	0,28	0,839
Tamaño de la población*Probabilidad de cruce	1	2480	2480	0,16	0,692
Tamaño de la población*N° de generaciones	1	2778	2778	0,18	0,675
Probabilidad de cruce*N° de generaciones	1	7579	7579	0,50	0,491
Interacciones de 3 términos	1	14044	14044	0,92	0,352
Tamaño de la población*Probabilidad de cruce*N° de generaciones	1	14044	14044	0,92	0,352
Error	16	244254	15266		
Total	23	527235			

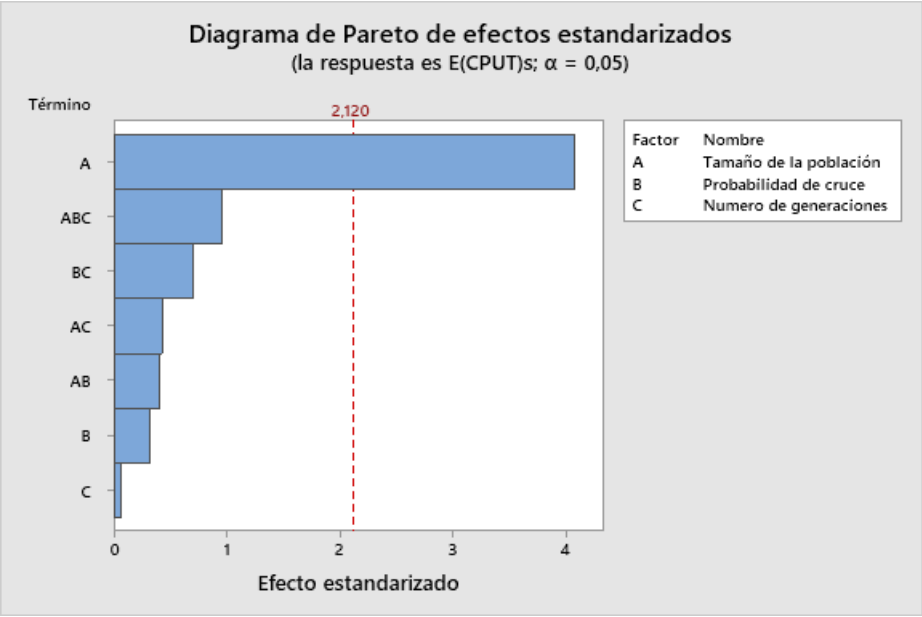


Figura 30. Diagrama de Pareto de efectos estandarizados. Instancia N50-M10

Adicionalmente, en la *Figura 31* se corrobora a través de comparación con los demás factores cuan influyente es el tamaño del a población en la variable respuesta cuando se encuentra en el nivel alto; por el contrario, aunque no tienen efectos significativos, la probabilidad de cruce al igual que el número de generaciones ofrecen soluciones promedio sin destacar algún nivel del diseño experimental.

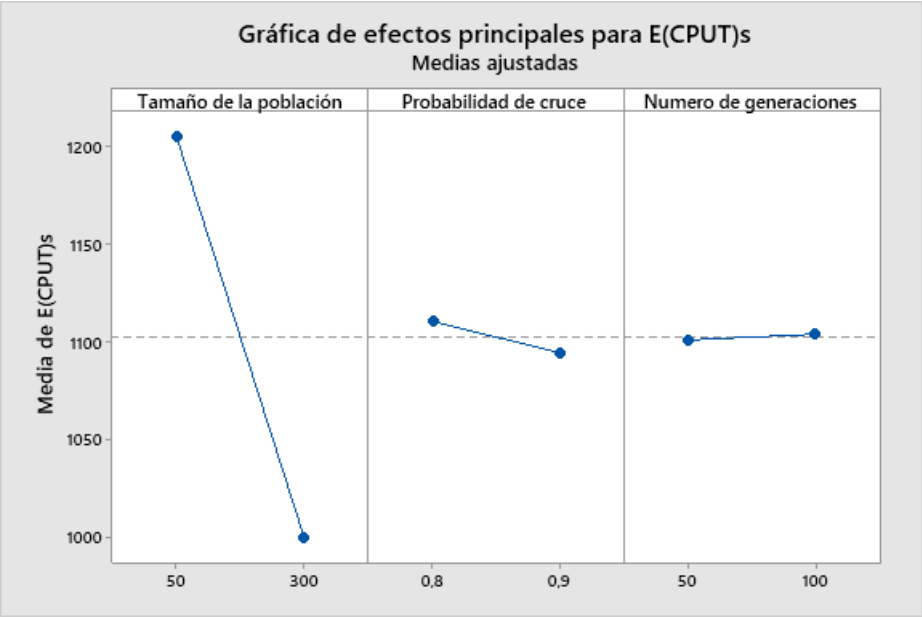


Figura 31. Gráfica de efectos principales para $E(CPUT)_s$. Instancia N50-M10

9.5. Instancia N100-M10 problemas de tamaño grande

Por último, se experimenta con una de las instancias de mayor tamaño planteadas, la N100-M10, para la cual se desarrolla el diseño de experimentos como se puede observar en la Tabla 18:

Tabla 18.

Estructura para el diseño de experimentos Instancia N100-M10

Tratamientos	***Replicas***		
	1	***2***	***3***
-1	1536,95	1406,13	1248,47
a	1424,96	1300,26	1272
b	1137,59	1444,46	1480,85
ab	1151,26	1168,2	1329,06
c	1053,39	1660,42	1650,39
ac	990,41	1069,49	1258,77
bc	1202,44	1673,48	1698,55
abc	1305,14	1012,15	1426,63

Contrario a la instancias anteriormente analizadas, en la *Figura 32* no se aprecia de manera clara la distribución de los datos, por tanto se muestra de forma complementaria la *Figura 33*, la cual es el resultado de la prueba de normalidad Kolmogórov-Smirnov donde se concluye que el valor-p es mayor al nivel de significancia y por tanto, no hay evidencia suficiente para rechazar la hipótesis nula correspondiente a la normalidad de los datos.

Por el contrario, de acuerdo con el histograma, se puede apreciar que los datos poseen un sesgamiento a la derecha; adicional a esto, se comprueba el supuesto de que los residuos tienen una varianza constante y que no están del todo correlacionados entre sí, salvo aquellos en lo que se presenta un patrón de ascenso a partir de la observación 16.

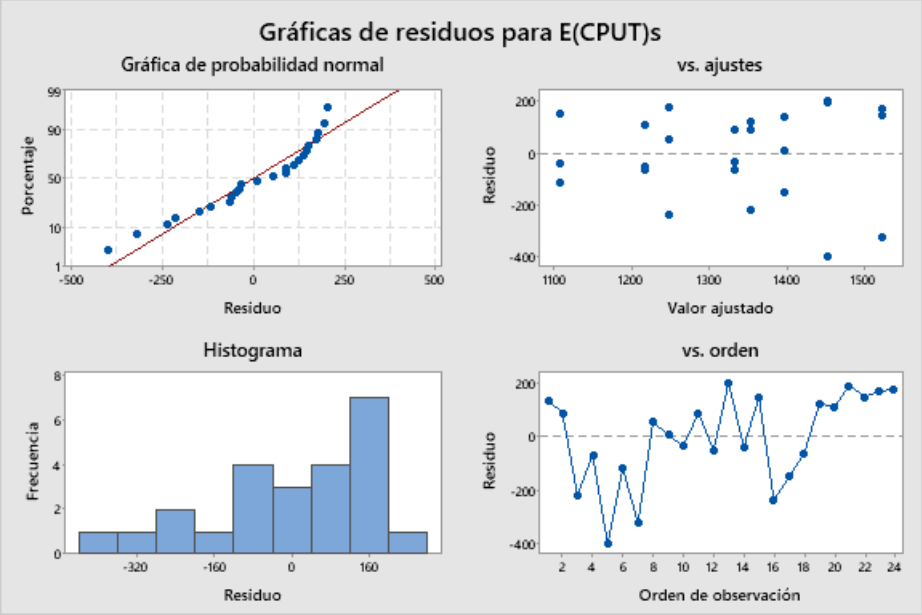


Figura 32. Gráfica de residuos para $E(CPUT)_s$. Instancia N100-M10

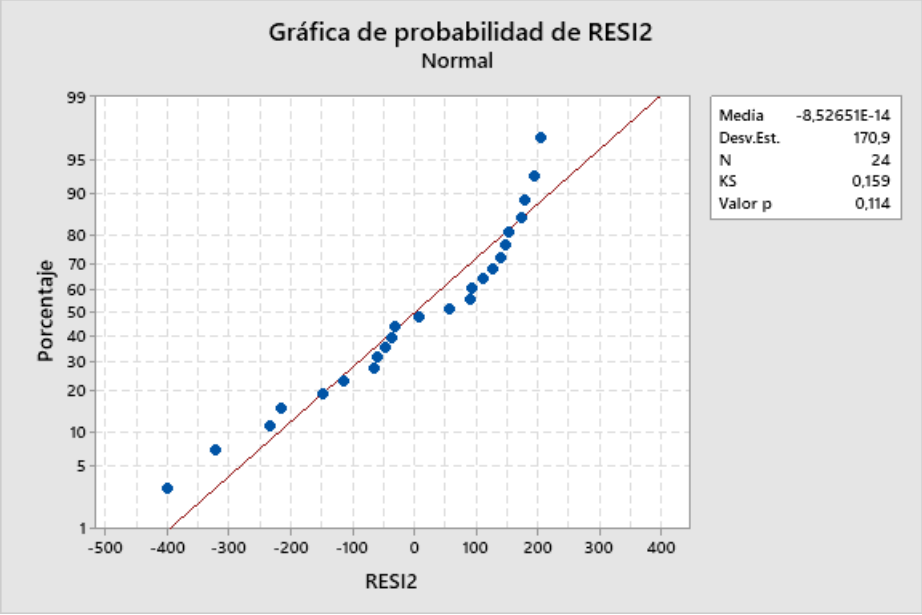


Figura 33. Grafica de probabilidad normal para los residuos

Ahora bien, en la

Tabla **19** se muestra el análisis de varianza ANOVA, el cual complementado la *Figura 34* permite la identificación del principal factor cuyos efectos son altamente significativos en la variable respuesta $E(CPUT)_s$, para instancias grandes el factor principal el tamaño de la población inicial con un valor-p del 0,025.

Tabla 19.
Análisis de Varianza. Instancia N100-M10

Fuente	***GL***	***SC Ajust.***	***MC Ajust.***	***Valor F***	***Valor p***
Modelo	7	385142	55020	1,31	0,308
Lineal	3	258726	86242	2,05	0,147
Tamaño de la población	1	257258	257258	6,12	0,025
Probabilidad de cruce	1	1042	1042	0,02	0,877
Numero de generaciones	1	426	426	0,01	0,921
Interacciones de 2 términos	3	118531	39510	0,94	0,444
Tamaño de la población*Probabilidad de cruce	1	1	1	0,00	0,996
Tamaño de la población*N° de generaciones	1	66926	66926	1,59	0,225
Probabilidad de cruce*N° de generaciones	1	51603	51603	1,23	0,284
Interacciones de 3 términos	1	7885	7885	0,19	0,671
Tamaño de la población*Probabilidad de cruce*N° de generaciones	1	7885	7885	0,19	0,671
Error	16	672052	42003		
Total	23	1057193			

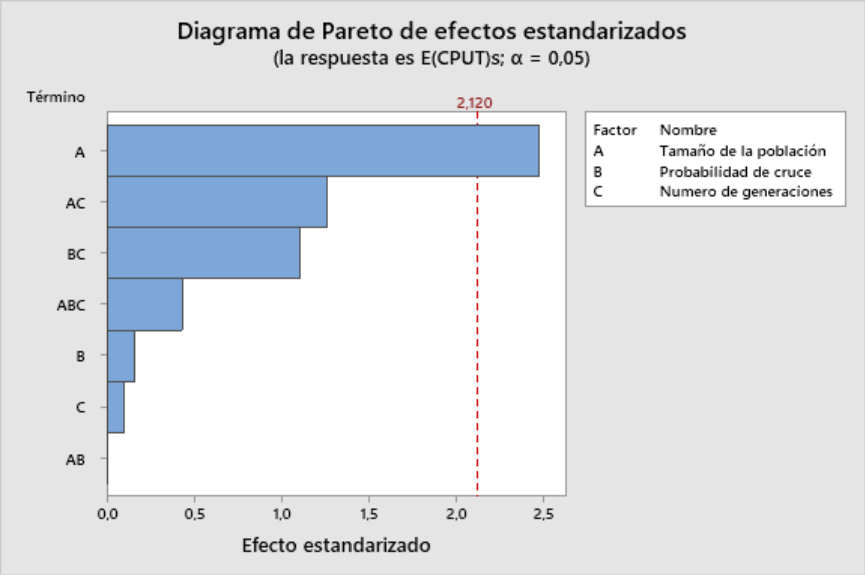


Figura 34. Diagrama de Pareto de efectos estandarizados. Instancia N100-M10.

Adicionalmente, en la *Figura 35* se reafirma que tamaño del a población es el principal factor cuyos efectos son significantes en la variable respuesta y para el cual se obtienen mejores resultados en su nivel más alto, contrarios a la probabilidad de cruce y el número de generaciones, los cuales ofrecen soluciones promedio sin destacar algún nivel del diseño experimental.

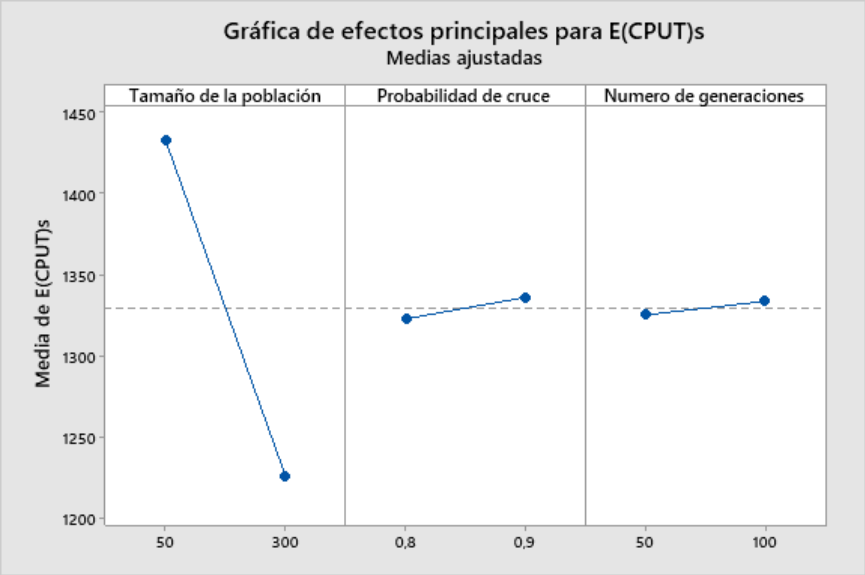


Figura 35. Gráfica de efectos principales para $E(CPUT)_s$. Instancia N100-M10

10. Resultados Computacionales

En esta sección, se evalúan los resultados de $E(CPUT)_s$ obtenidos mediante los dos métodos propuestos, el método exacto denominado Fuerza Bruta y el método aproximado correspondiente a la metaheurística basada en el Algoritmo Genético. En primera instancia se comparan los resultados obtenidos por cada uno de los métodos para las instancias propuestas en la sección 8.1.

Posteriormente, para instancias del problema de tamaño mediano y grande se evalúan los resultados de $E(CPUT)_s$ obtenidos por medio del algoritmo GA; el método fuerza bruta, por el contrario, no se considera pertinente para este tipo de instancias debido a que no provee resultados en un tiempo computacional razonable.

Los resultados de la ejecución fueron obtenidos usando un computador con procesador Intel® Core™ i7-8700 CPU @ de 3,20 GHz, memoria RAM instalada de 8 GB y un sistema operativo Microsoft Windows 10 Pro-Versión 10.0 (Build 17763) de 64 bits. El software empleado fue Matlab en la versión: 9.6.0.1099231 (R2019a) Update 1, con el solver Global Optimization Toolbox y el solver Optimization Toolbox.

10.1. Resolución instancias pequeñas

10.1.1. Fuerza Bruta. Como se puede apreciar en la Tabla 20, se evaluaron los resultados de $E(CPUT)_s$ y se determinaron las soluciones óptimas para cada una de las instancias planteadas; así mismo se puede observar conforme al patrón de los resultados que el incremento en el número de

máquinas reduce el costo total por unidad de tiempo de programación $E(CPUT)_s$, esto debido a que, según lo establece la función objetivo, el "makespan" es inversamente proporcional al $E(CPUT)_s$.

Tabla 20.

Resultados computacionales para problemas de tamaño pequeño mediante fuerza bruta

n x m	***Solución óptima***										***E(CPUT)s***	***CPU time***
8 x 5	6	3	8	5	1	7	2	4			660,3	2,698 s
8 x 10	6	8	3	7	2	5	4	1			399,5	3,923 s
8 x 15	6	3	8	5	4	7	1	2			369	4,708 s
8 x 20	6	8	3	4	5	7	1	2			357,8	5,598 s
9 x 5	7	3	4	5	6	9	8	1	2		777,6	20,962 s
9 x 10	7	8	5	6	4	2	9	1	3		419,3	32,057 s
9 x 15	6	4	7	2	9	5	8	1	3		407,4	41,084 s
9 x 20	7	8	2	4	6	5	9	1	3		384,8	51,753 s
10 x 5	6	7	8	1	3	4	5	2	10	9	854,4	234,213 s
10 x 10	6	8	1	7	9	10	2	4	3	5	561,6	327,369 s
10 x 15	6	7	2	10	8	5	9	1	3	4	379,5	435,662 s
10 x 20	6	3	1	7	2	10	9	4	5	8	330,2	549,254 s

Adicionalmente, por medio de la tabla presentada anteriormente es posible observar como el tiempo computacional aumenta considerablemente conforme aumenta el tamaño de problema, esto corrobora las afirmaciones realizadas en relación con el incremento exponencial del tiempo computacional como consecuencia del aumento en alguno de los parámetros *n x m* que definen el tamaño del problema.

10.1.2. Algoritmo genético. Con base en el análisis estadístico desarrollado en la sección anterior, se calibró el algoritmo de tal manera que obtuviera resultados certeros y lo suficientemente cercanos a la solución óptima. La Tabla 21 muestra los resultados obtenidos para instancias pequeñas por medio del algoritmo GA:

Tabla 21.

Resultados computacionales para problemas de tamaño pequeño mediante GA

n x m	***Solución óptima***										***E(CPUT)$_s$***
8 x 5	3	8	6	5	1	7	2	4			684,65
8 x 10	6	3	8	5	4	7	2	1			428,58
8 x 15	6	3	8	5	4	7	1	2			369,01
8 x 20	6	8	3	4	5	7	1	2			357,81
9 x 5	7	8	5	3	6	9	4	1	2		809,08
9 x 10	7	5	8	6	4	2	9	1	3		456,78
9 x 15	7	6	9	5	1	2	4	8	3		440,45
9 x 20	7	8	4	2	6	5	1	9	3		392,79
10 x 5	6	7	3	8	1	2	9	10	5	4	945,34
10 x 10	6	7	10	8	1	4	2	9	5	3	610,38
10 x 15	6	7	10	8	1	2	3	9	5	4	413,50
10 x 20	6	7	2	10	9	1	3	5	8	4	380,53

Ahora bien, una vez evaluados por medio de los dos métodos los valores de $E(CPUT)_s$, es posible determinar en qué medida los resultados arrojados por el algoritmo GA se aproximan a la solución exacta hallada mediante el método de fuerza bruta. Los dos métodos se comparan a través del porcentaje de desviación relativa RPD, para el cual se considera como la mejor solución encontrada "*Best"* el valor de $E(CPUT)_s$ determinado mediante el método de fuerza bruta.

$$RPD = \frac{AG_{SOL} - Best}{Best}$$

En la Tabla 22 se presenta el porcentaje de desviación relativa del algoritmo GA en relación con el método de solución exacto, esto para cada una de las instancias planteadas para problemas de tamaño pequeño. Si bien se ha afirmado en repetidas oportunidades que el algoritmo solo provee soluciones aproximadas, para este tipo de instancias e incluso instancias de menor tamaño es posible determinar soluciones optimas haciendo uso del algoritmo GA.

Tabla 22.

RPD del algoritmo GA sobre la solución exacta

n x m	*Fuerza Bruta*	*Algoritmo GA*	*RPD*
8 x 5	660,3	684,65	3,7%
8 x 10	399,5	428,58	7,3%
8 x 15	369,0	369,01	0,0%
8 x 20	357,8	357,81	0,0%
9 x 5	777,6	809,08	4,0%
9 x 10	419,3	456,78	8,9%
9 x 15	407,4	440,45	8,1%
9 x 20	384,8	392,79	2,1%
10 x 5	854,4	945,34	10,6%
10 x 10	561,6	610,38	8,7%
10 x 15	379,5	413,5	9,0%
10 x 20	330,2	380,53	15,2%

Adicionalmente, en la *Figura 36* se observa una tendencia al alza del porcentaje de desviación relativo que conforme el tamaño del problema se hace mayor, lo cual es razonable dado que a medida que el tamaño del problema aumenta resulta más complejo determinar la solución y a su vez la posibilidad de caer en un óptimo local es considerable.

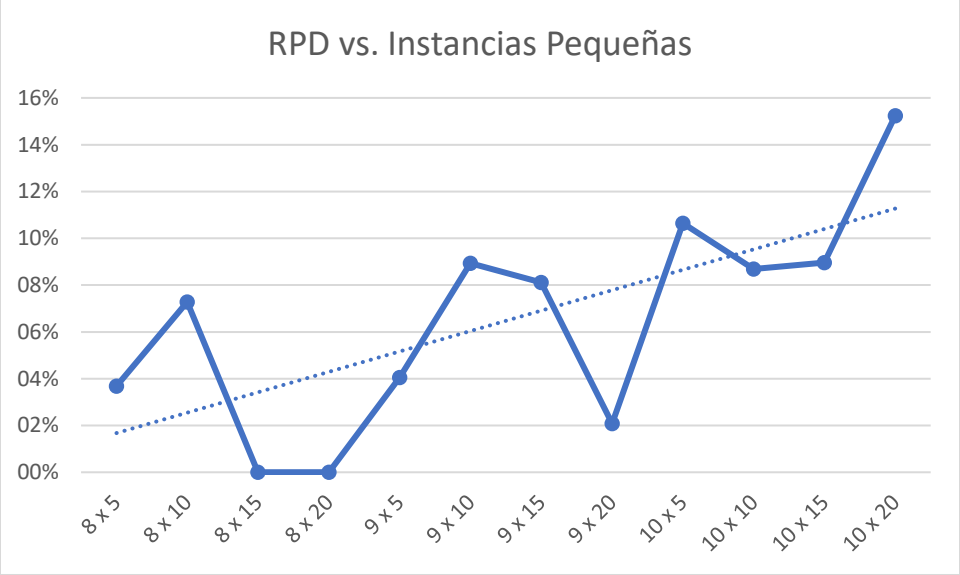


Figura 36. Tendencia RPD vs Instancias Pequeñas

10.2. Resolución instancias grandes

Como se mencionó anteriormente, el modelo fuerza bruta tiene como limitación el número de trabajos, once (11) como máximo, por lo tanto no fue considerado para dar solución a las instancias asociadas a problemas de tamaño grande, no obstante, los resultados obtenidos por medio del algoritmo GA son evaluados tomando como referencia los obtenidos mediante las metaheurísticas desarrolladas por Mishra & Shrivastava.

Mishra & Shrivastava (2018) compararon los resultados de cuatro algoritmos: Optimización basada en el proceso de enseñanza (TLBO), Algoritmo de Jaya (una palabra sánscrita que significa victoria), Optimización por Enjambre de Partículas (PSO) y Recocido Simulado (SA) determinando que el rendimiento general de todos los algoritmos revela que TLBO y Jaya tienen un potencial considerable para resolver problemas combinatorios discretos como PFSP. Ahora bien, antes de realizar la respectiva comparación es necesario establecer los siguientes precedentes:

Los resultados asociados a la variable respuesta $E(CPUT)_s$ y más específicamente a los costos asociados al mantenimiento de inventario, IC_{wp} y IC_{bw}, presentaron diferencias con respecto al investigación desarrollada por (Mishra & Shrivastava, 2018), incluso desde el desarrollo del modelo exacto. La variabilidad en la solución puede deberse a que no es posible conocer todas las consideraciones que en su momento Mishra & Shrivastava (2018) tuvieron; así mismo la variabilidad puede asociarse a la interpretación y las consideraciones establecidas por los autores en el desarrollo del presente trabajo investigativo.

Dado lo anterior fue necesario establecer vínculos con los autores vía e-mail, esto con el fin de esclarecer dudas respecto al cálculo del costo de mantenimiento de inventario permitiendo establecer que Mishra & Shrivastava (2018) asumen que se incurre en un costo de mantenimiento de inventario solo para aquellos trabajos que se retrasan más allá de su fecha de vencimiento, es decir, solo para trabajos tardíos, en dado caso la variable binaria (theta) es 1 si el tiempo de finalización en cualquier máquina es mayor que la fecha de vencimiento o de lo contrario es '0'. Por ejemplo, un trabajo que se procesa en tres máquinas con un tiempo de finalización de 100, 200 y 300 respectivamente. La fecha de vencimiento dada del trabajo es 220. Ahora, dado que el tiempo de finalización del trabajo en la máquina 1 y 2 es menor que la fecha de vencimiento, por lo tanto, no se incurre en costos, es decir, theta es 0; sin embargo, el costo de inventario se incurre solo para la tercera máquina.

Una vez dado a conocer lo anterior, se presenta la comparación y análisis respectivo de los resultados de $E(CPUT)_s$ en instancias de problemas de tamaño mediano-grande. En la Tabla 23 se presentan los resultados computacionales de referencia para la comparación de $E(CPUT)_s$, dichos valores fueron adaptados de (Mishra & Shrivastava, 2018), de donde se extrajeron los valores mínimo y máximo de $E(CPUT)_s$ entre las cuatro meta heurísticas para cada una de las instancias de problemas de tamaño mediano-grande planteadas.

Tabla 23.

Resultados computacionales de referencia para comparación de E(CPUT)s

n x m	***Min***	***Max***	***Media***
20 x 5	931,63	1000,50	947,19
20 x 10	302,33	367,61	314,01
20 x 20	66,64	96,64	79,04
30 x 10	166,23	298,20	192,99

Continuación Tabla 23.
Resultados computacionales de referencia para comparación de E(CPUT)s

n x m	***Min***	***Max***	***Media***
30 x 15	60,26	116,54	73,53
50 x 5	1418,80	2227,60	1636,22
50 x 10	69,13	143,83	87,32
50 x 20	31,21	75,22	41,81
100 x 5	2448,30	3656,00	2857,62
100 x 10	44,83	109,23	60,59
200 x 10	305,97	764,90	428,86
500 x 20	741,40	2205,60	1151,56

Adicionalmente se realizaron 5 réplicas de ejecución del algoritmo GA para cada una de las instancias establecidas; cabe aclarar que dado que se desconocían los parámetros iniciales mediante los cuales se determinaron los resultados computacionales de referencia anteriormente mostrados se optó por permitir que los parámetros iniciales se distribuyeran uniformemente conforme a lo establecido anteriormente en la Tabla 5. Los resultados para $E(CPUT)_s$ obtenidos en cada una de las réplicas se presentan a continuación en la Tabla 24.

Tabla 24.
Resultados computacionales obtenidos mediante el algoritmo GA

n x m	***1***	***2***	***3***	***4***	***5***	***Min***	***Max***	***Media***
20 x 5	1656,74	1550,57	1655,58	1613,83	1621,6	1550,57	1656,74	1619,66
20 x 10	499,7	442,48	541,33	499,31	512,32	442,48	541,33	499,03
20 x 20	60,03	113,49	86,61	75,65	69,55	60,03	113,49	81,07
30 x 10	497,26	451,6	433,59	408,95	499,85	408,95	499,85	458,25
30 x 15	58,4	67,99	82,71	59,07	79,95	58,4	82,71	69,62
50 x 5	4046,27	4107,32	4092,45	4013,24	3930,43	3930,43	4107,32	4037,94
50 x 10	587,19	590,96	512,47	501	512,93	501	590,96	540,91
50 x 15	1,22	5,55	4,56	2,6	2,23	1,22	5,55	3,23
100 x 5	8620,14	8720,52	8658,7	8515,02	8669,05	8515,02	8720,52	8636,69
100 x 10	749,5	748,96	707,59	599,65	784,35	599,65	784,35	718,01
200 x 10	1232,98	1419,71	1391,03	1104,46	1084,52	1084,52	1419,71	1246,54
500 x 10	3623,43	3421,31	3385,55	3356,04	3081,95	3081,95	3623,43	3373,66

A partir de los datos obtenidos de $E(CPUT)_s$ en cada una de las réplicas realizadas y como se puede evidenciar en la *Figura 37*, la variabilidad en la resolución del PSFP mediante el algoritmo genético no es significa dado que la mayoría de las veces converge a un resultado en particular. Del mismo modo se puede observar como el costo por unidad de tiempo se programación se eleva considerablemente al programar gran cantidad de trabajos en un número reducido de máquinas, como es el caso de la instancia *100x5*; el caso contrario ocurre cuando la cantidad de máquinas es tal que el makespan se hace considerable y por consiguiente el valor de $E(CPUT)_s$ insignificante.

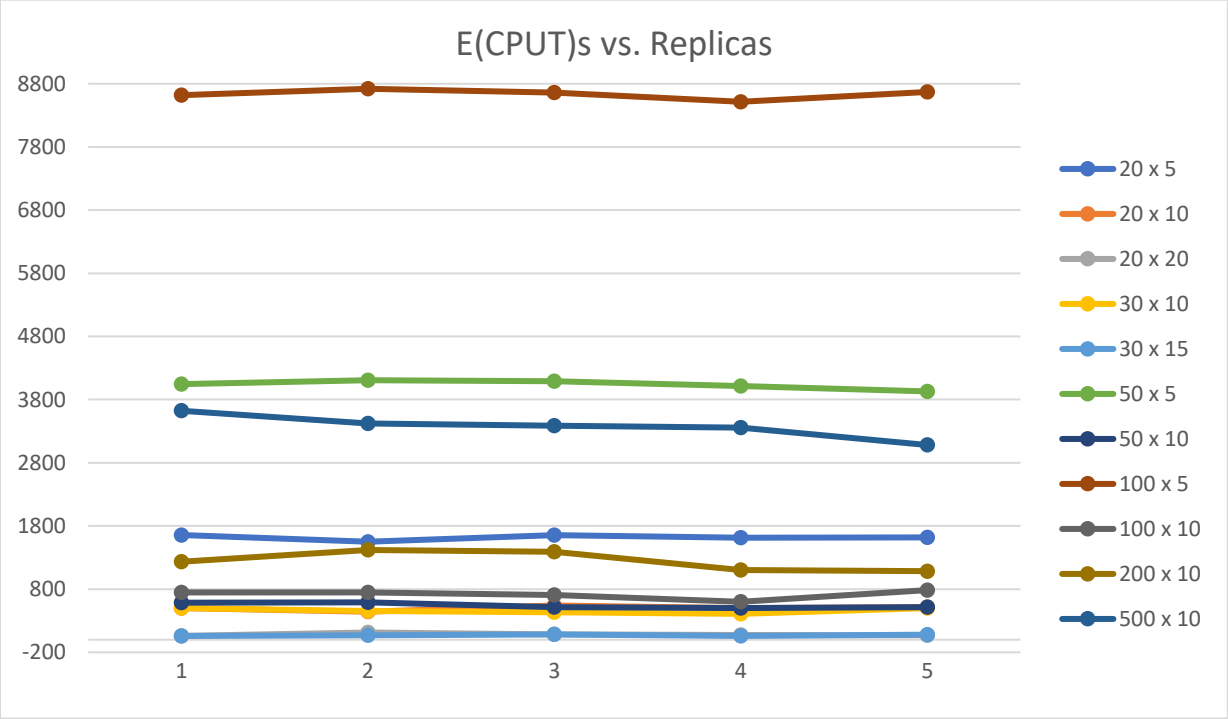


Figura 37. E(CPUT)s por replica

Ahora bien, a partir de las réplicas realizadas se procedió como se observa en la Tabla 25 con el cálculo de intervalos de confianza, los cuales corresponden a el rango de valores, cuya distribución es normal y en el cual se encuentra, con alta probabilidad, el valor real de $E(CPUT)_s$.

Tabla 25.

RPD del algoritmo GA sobre los resultados computacionales de referencia

	(Mishra & Shrivastava, 2018)						
n x m	***Min***	***Max***	***Media***	***LI***	***LS***	***Media***	***RPD***
20 x 5	931,63	1000,50	947,19	1581,77	1657,56	1619,66	58%
20 x 10	302,33	367,61	314,01	467,53	530,52	499,03	27%
20 x 20	66,64	96,64	79,04	63,07	99,06	81,07	0%
30 x 10	166,23	298,20	192,99	423,37	493,13	458,25	42%
30 x 15	60,26	116,54	73,53	59,65	79,60	69,62	0%
50 x 5	1418,80	2227,60	1636,22	3975,93	4099,96	4037,94	78%
50 x 10	69,13	143,83	87,32	502,13	579,69	540,91	249%
50 x 15	31,21	75,22	41,81	1,68	4,79	3,23	N/A
100 x 5	2448,30	3656,00	2857,62	8569,31	8704,07	8636,69	134%
100 x 10	44,83	109,23	60,59	655,31	780,71	718,01	500%
200 x 10	305,97	764,90	428,86	1109,70	1383,38	1246,54	N/A
500 x 10	741,40	2205,60	1151,56	3203,80	3543,51	3373,66	N/A

Si bien los resultados computacionales obtenidos poseen diferencias significativas en 7 de las 9 instancias comparables, como era de esperarse debido a diferencias en cálculo del costo de mantenimiento de inventarios, se observa un patrón de comportamiento conforme a las características del sistema productivo, es decir, el valor de E(CPUT)s disminuye conforme aumenta el número de máquinas, debido a que una mayor cantidad de máquinas implica un aumento en el valor para el "makespan", el cual es inversamente proporcional a la variable respuesta.

Aquellas instancias para las cuales no aplica comparación alguna se debe a que fueron alteradas en el número de máquinas, lo anterior debido a que el gran número de lotes de trabajos representa un valor considerable de "makespan", lo que conlleva a que los resultados para *E(CPUT)s* sean insignificantes. Cable resaltar los intervalos determinados para las instancias 20x15 y 30x15, los

cuales corresponden a las instancias para la cuales la brecha comparativa es considerablemente menor.

Por otro lado, de igual manera a como se evidenciaba para las instancias de tamaño pequeño, en la *Figura 38* se observa una tendencia al alza del porcentaje de desviación relativa conforme aumenta el tamaño del problema y por consiguiente la complejidad para encontrar una solución óptima del mismo.

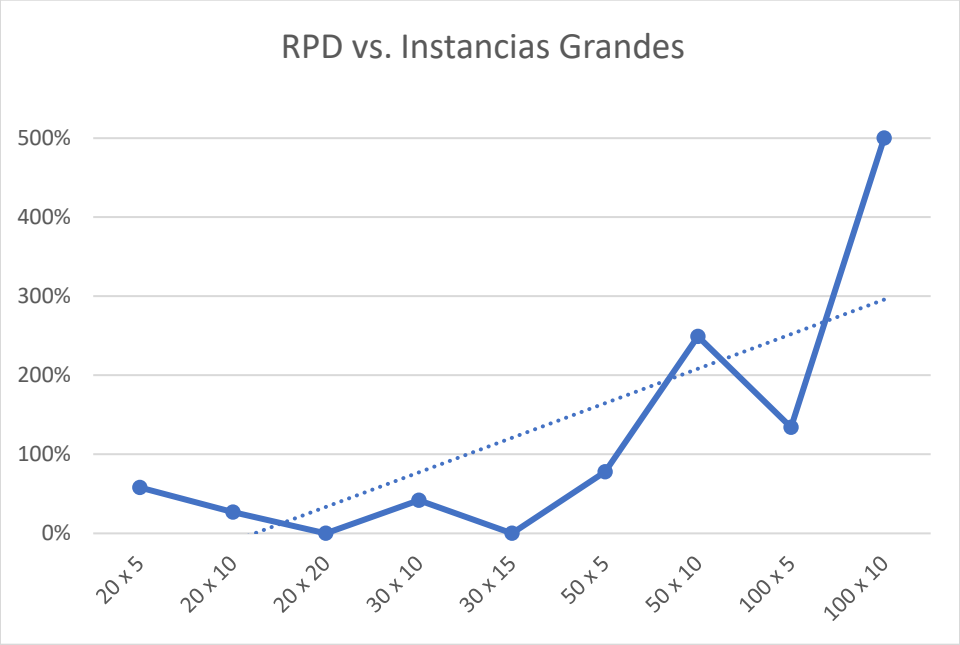


Figura 38. Tendencia RPD vs Instancias Grandes

11. Conclusiones

- De acuerdo con el análisis bibliométrico el tema solución al problema de “flow-shop” permutado empleando técnicas metaheurísticas se considera un tema pertinente para el desarrollo de la presente investigación. Desde los años cincuenta, con la divulgación de la regla de Jhonson, numerosos autores han desarrollado investigaciones relacionadas los entornos de producción tipo “flow-shop”; hoy por hoy la programación de la producción sigue desempeñando un papel de vital importancia en la planificación y operación del sistema productivo, esto en términos de costos, reducción del inventario y el adecuado flujo de producción.

- El modelo de programación matemática contempla un objetivo basado en costos, el cual busca minimizar el costo total por unidad de tiempo de programación *E(CPUT)*, para ello se tienen en cuenta los costos de mantenimiento de inventario y los costos de penalización debido a la demora del lote y se evalúa en dos conjuntos de instancias; para las instancias de tamaño pequeño se determinan soluciones exactas en tiempos computacionales relativamente bajos; no obstante para instancias del problema de gran tamaño grande emplea un tiempo considerablemente significativo por tratarse de un modelo que emplea búsqueda exhaustiva o fuerza bruta en la determinación de la solución.

- Adicionalmente para optimizar la función objetivo, se planteó el desarrollo de un algoritmo GA, para el cual mediante el proceso de calibración se concluyó el tamaño de la población inicial corresponde al principal factor que influyen de manera significativa en la

variable respuesta $E(CPUT)_s$ en instancias de tamaño grande, por lo cual se opta por emplear el nivel alto de dicho factor con el propósito de determinar mejores soluciones. Otros factores analizados son la probabilidad de cruce y el número de generaciones para los cuales, pese a no representar efectos significativos, se tienen mejores resultados de $E(CPUT)_s$ en su niveles bajo y alto respectivamente.

- La evaluación de desempeño de la metaheurística calibrada arrojó que el algoritmo GA ejecutado para instancias de tamaño pequeño provee soluciones competentes frente al modelo fuerza bruta, con un porcentaje de desviación relativa inferior al 10%, lo cual permite evidenciar el gran potencial que tiene la empleabilidad de metaheurísticas como el Algoritmo Genético en la generación de respuestas oportunas para el apoyo de la toma de decisiones relacionadas los recursos de producción disponibles en un sistema productivo tipo "flow-shop".

- Para instancias de tamaño mediano-grande, pese a que no fue posible realizar la comparación entre los valores de $E(CPUT)_s$ determinados mediante los dos métodos propuestos, Fuerza Bruta y Algoritmo GA, puede afirmarse basados en comparaciones con otras metaheurísticas presentadas en la literatura[6] que el enfoque genético desarrollado en el presente trabajo investigativo provee soluciones acertadas para cada uno de los escenarios propuestos. Encontrándose que para dos de las nueve instancias comparables, es decir, en el 22% de las instancias el algoritmo propuesto superó las demás metaheurísticas encontrando

[6] Mishra, A., & Shrivastava, D. (2018). A TLBO and a Jaya heuristics for permutation flow shop scheduling to minimize the sum of inventory holding and batch delay costs. Computers and Industrial Engineering, 124, 509–522. https://doi.org/10.1016/j.cie.2018.07.049

mejores resultados; adicionalmente cabe resaltar que para instancias menores a 100 trabajos el desempeño del algoritmo evaluado en términos del porcentaje promedio de desviación relativa fue del 34%.

12. Recomendaciones

Las investigaciones adicionales en el estudio propuesto pueden extenderse en las siguientes direcciones:

- El desarrollo de nuevos modelos exactos como la programación lineal entera y la implementación de metaheurísticas alternas a partir de las cuales se pueda comparar y validar los resultados obtenidos para PFSP.

- Contemplar nuevas restricciones, consideraciones, parámetros o atributos para el problema, que le permitan modelar de manera más cercana a la realidad de la industria actual, tales como no esperas, buffers de tamaño limitados o no, transportes, entre otros.

- Evaluar nuevas instancias para el modelo actual o para un modelo modificado con el fin de comparar el desempeño del algoritmo genético para diferentes tamaños del PFSP.

- Abordar el PFSP bajo un enfoque multiobjetivo, en el cual además de minimizar el costo total por unidad de tiempo de programación, se logre minimizar otros factores como el como el “makespan” o el tiempo medio de flujo.

Referencias Bibliográficas

Abouei Ardakan, M., Hakimian, A., & Rezvan, M. T. (2014). A branch-and-bound algorithm for minimising the number of tardy jobs in a two-machine flow-shop problem with release dates. *International Journal of Computer Integrated Manufacturing*, *27*(6), 519–528. https://doi.org/10.1080/0951192X.2013.820349

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

Arora, D., & Agarwal, G. (2016). Meta-heuristic approaches for flowshop scheduling problems: a review. *International Journal of Advanced Operations Management*, *8*(1), 1. https://doi.org/10.1504/IJAOM.2016.076203

Bauman, J., & Józefowska, J. (2006). Minimizing the earliness-tardiness costs on a single machine. *Computers and Operations Research*, *33*(11), 3219–3230. https://doi.org/10.1016/j.cor.2005.02.037

Benkalai, I., Rebaine, D., Gagné, C., & Baptiste, P. (2017). Improving the migrating birds optimization metaheuristic for the permutation flow shop with sequence-dependent set-up times. *International Journal of Production Research*, *55*(20), 6145–6157. https://doi.org/10.1080/00207543.2017.1327732

Bülbül, K., Kaminsky, P., & Yano, C. (2004). Flow shop scheduling with earliness, tardiness, and intermediate inventory holding costs. *Naval Research Logistics*, *51*(3), 407–445. https://doi.org/10.1002/nav.20000

Cao, D., & Chen, M. (2003). Parallel flowshop scheduling using Tabu search. *International*

Journal of Production Research, *41*(13), 3059–3073. https://doi.org/10.1080/0020754031000106443

Castillo, E., Conejo, A. J., & Pedregal, P. (2002). *Formulacion y Resolución de Modelos de Programacion Matemática en Ingeniería y Ciencia.*

Chen, J. S., Pan, J. C. H., & Lin, C. M. (2008). A hybrid genetic algorithm for the re-entrant flow-shop scheduling problem. *Expert Systems with Applications*, *34*(1), 570–577. https://doi.org/10.1016/j.eswa.2006.09.021

Cheng, B. Y., Leung, J. Y. T., & Li, K. (2017). Integrated scheduling on a batch machine to minimize production, inventory and distribution costs. *European Journal of Operational Research*, *258*(1), 104–112. https://doi.org/10.1016/j.ejor.2016.09.009

Ciavotta, M., Meloni, C., & Pranzo, M. (2013). Minimising general setup costs in a two-stage production system. *International Journal of Production Research*, *51*(8), 2268–2280. https://doi.org/10.1080/00207543.2012.716172

Cogollo Flórez, J. M. (2017). Métodos exactos y heurísticos en la solución de problemas de redes de transporte en las cadenas de suministros. *Lupa Empresarial*, *N° 18*, 44–63.

Colina, Y. B. (2011). Aplicaciones de programación lineal, entera y mixta. *Ingeniería Industrial. Actualidad y Nuevas Tendencias*, *II*(7), 85–104. Retrieved from http://redalyc.org/articulo.oa?id=215024822007

Cruz Chavez, M. A., Moreno Bernal, P., & Peralta Abarca, J. del C. (2014). Aplicación de la teoría de la complejidad en optimización combinatoria. *Narraciones de La Ciencia y La Tecnologia INVENTIO*, (2007–1760).

Díaz, A., Glover, F., Ghaziri, H. M., González, J., Laguna, M., Moscato, P., & Tseng, F. T. (1996). Optimización heurística y redes neuronales. *Paraninfo*.

Dorado, J., Gestal, M., Rivero, D., Rabuñal, J. R., & Pazos, A. (2010). Introduccion a Los Algortimos Geneticos. In *Digitalia*.

Fogel, D. (2005). *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence* (Third edit; W.-I. Press, Ed.). Retrieved from https://www.wiley.com

Gutiérrez Pulido, H., & De la Vara Salazar, R. (2008). *Analisis y diseño de experimentos* (McGRAW-HIL).

H. Papadimitriou, C., & Steiglitz, K. (1982). Combinatorial Optimization: Algorithms and Complexity. In *IEEE Transactions on Acoustics, Speech, and Signal Processing* (Vol. 32). https://doi.org/10.1109/TASSP.1984.1164450

Hansen, P., Mladenovic, N., & Moreno Pérez, J. A. (2003). Búsqueda de Entorno Variable. *Inteligencia Artificial Revista Iberoamericana de Inteligencia Artificial*, *07*, 77–92.

Hassanzadeh, A., Rasti-Barzoki, M., & Khosroshahi, H. (2016). Two new meta-heuristics for a bi-objective supply chain scheduling problem in flow-shop environment. *Applied Soft Computing Journal*, *49*, 335–351. https://doi.org/10.1016/j.asoc.2016.08.019

Hillier, F. S., & Lieberman, G. J. (2001). *Introducción a la Investigación de Operaciones.* (7ma. Edici). Ciudad de Mexico: Mc. Graw.Hill.

Krajewski, L., Ritzman, L., & Malhotra, M. (2009). Administración de Operaciones. In *Administración de operaciones* (Octava edi, Vol. 20). https://doi.org/10.4067/S0718-07642009000500001

Li, G., Sambandam, N., Sethi, S. P., & Zhang, F. (2018). *Flow shop scheduling with jobs arriving at di ff erent times*. *206*(September), 250–260. https://doi.org/10.1016/j.ijpe.2018.10.010

Logendran, R., & Mehravaran, Y. (2013). Non-permutation flowshop scheduling with dual resources. *Expert Systems with Applications*, *40*(13), 5061–5076.

https://doi.org/10.1016/j.eswa.2013.03.007

Maldonado, C. E. (2013). Un Problema Fundamental De La Investigación: Los Problemas P vs NP (A Fundamental Problem in Research: P vs NP Problems). *Revista Logos, Ciencia & Tecnología*, 10–20. https://doi.org/10.2139/ssrn.2738938

Martí, R. (2003). *Procedimientos Metaheurísticos en Optimización Combinatoria*. 1–60.

MathWorks. (n.d.). MathWorks, Inc.

Mehravaran, Y., & Logendran, R. (2012). Non-permutation flowshop scheduling in a supply chain with sequence-dependent setup times. *International Journal of Production Economics*, *135*(2), 953–963. https://doi.org/10.1016/j.ijpe.2011.11.011

Minitab. (n.d.). Minitab © 2019 All rights Reserved.

Mishra, A., & Shrivastava, D. (2018). A TLBO and a Jaya heuristics for permutation flow shop scheduling to minimize the sum of inventory holding and batch delay costs. *Computers and Industrial Engineering*, *124*, 509–522. https://doi.org/10.1016/j.cie.2018.07.049

Molina-Sánchez, L. P., & González-Neira, E. M. (2016). GRASP to minimize total weighted tardiness in a permutation flow shop environment. *International Journal of Industrial Engineering Computations*, *7*(1), 161–176. https://doi.org/10.5267/j.ijiec.2015.6.004

Paul, S. K., & Azeem, A. (2010). Minimization of work-in-process inventory in hybrid flow shop scheduling using fuzzy logic. *International Journal of Industrial Engineering : Theory Applications and Practice*, *17*(2), 115–127.

Pinedo, M. L. (2012). *Scheduling: Theory, Algorithms, and Systems* (4ta. Edici). https://doi.org/10.1007/978-1-4614-2361-4

Rasti-Barzoki, M., Hejazi, S. R., & Mazdeh, M. M. (2013). A Branch and Bound Algorithm to Minimize the Total Weighed Number of Tardy Jobs and Delivery Costs. *Applied*

Mathematical Modelling, *37*(7), 4924–4937. https://doi.org/10.1016/j.apm.2012.10.001

Rathinam, B. (2015). Rule based heuristic approach for minimizing total flow time in permutation flow shop scheduling. *Tehnicki Vjesnik - Technical Gazette*, *22*(1), 25–32. https://doi.org/10.17559/TV-20130704132725

Render, B., & Heizer, J. H. (2009). Principios De Administración. In *Administración y Economía*. https://doi.org/10.1017/CBO9781107415324.004

Rodes, F. (2011). *Algoritmos Proyectivos de separacion para problemas de Programacion Lineal Entera*.

Ruiz, A. (2009). Herramientas Estadísticas - Comparación De Más De Dos Muestras : Anova (Parte I). *Universidad Pointificia de Comillas*, (Parte I), 1–22. Retrieved from http://web.cortland.edu/matresearch/ANOVA-I.pdf

Schaller, J., & Valente, J. M. S. (2013). A comparison of metaheuristic procedures to schedule jobs in a permutation flow shop to minimise total earliness and tardiness. *International Journal of Production Research*. https://doi.org/10.1080/00207543.2012.663945

Scudder, G. D., & Hoffmann, T. R. (1987). The use of cost-based priorities in random and flow shops. *Journal of Operations Management*, *7*(1–2), 217–232. https://doi.org/10.1016/0272-6963(87)90018-0

Varmazyar, M., & Salmasi, N. (2012). Sequence-dependent flow shop scheduling problem minimising the number of tardy jobs. *International Journal of Production Research*, *50*(20), 5843–5858. https://doi.org/10.1080/00207543.2011.632385

Vidal Esmorís, A. (2013). *Trabajo fin de Máster Algoritmos Heurísticos en Optimización*. 85.

Xiao, Y., Yuan, Y., Zhang, R. Q., & Konak, A. (2015). Non-permutation flow shop scheduling with order acceptance and weighted tardiness. *Applied Mathematics and Computation*,

270(15), 312–333. https://doi.org/10.1016/j.amc.2015.08.011

Yang, J. (2009). Two machine flow shop scheduling problem with weighted WIP costs. *Computers and Operations Research*. https://doi.org/10.1016/j.cor.2007.09.014

Yang, J., & Posner, M. E. (2010). Flow shops with WIP and value added costs. *Journal of Scheduling*, *13*(1), 3–16. https://doi.org/10.1007/s10951-009-0130-z

Yang, J., Society, M., & Engineering, F. (2018). Special Cases on Two Machine Flow Shop Scheduling with Weighted WIP Costs Special Cases on Two Machine Flow Shop Scheduling with Weighted WIP Costs ∗. *International Journal of Management Science*, *15*(2).

Yu, C., Semeraro, Q., & Matta, A. (2018). A genetic algorithm for the hybrid flow shop scheduling with unrelated machines and machine eligibility. *Computers and Operations Research*, Vol. 100, pp. 211–229. https://doi.org/10.1016/j.cor.2018.07.025