UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MORELOS

UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MORELOS

FACULTAD DE CONTADURÍA, ADMINISTRACIÓN E INFORMÁTICA

MAESTRÍA EN OPTIMIZACIÓN Y CÓMPUTO APLICADO

Diseño de una Estructura de Vecindad Variable en Función de su Línea de Tiempo Para el Problema de Talleres de Manufactura

T E S I S

QUE PARA OBTENER EL GRADO DE

MAESTRÍA EN OPTIMIZACIÓN Y CÓMPUTO APLICADO

PRESENTA

PEDRO BELLO CAMPUSANO

DIRECTOR DE TESIS

DR. MARCO ANTONIO CRUZ CHÁVEZ

REVISORES:

DR. FEDERICO ALONSO PECINA

DR. JOSE ALBERTO HERNANDEZ AGUILAR

DR. MARCO ANTONIO CRUZ CHAVEZ

DRA. JESUS DEL CARMEN PERALTA ABARCA

DR. RAFAEL RIVERA LÓPEZ

Facultad de Contaduría, Administración e Informática

CUERNAVACA, MORELOS. ABRIL,2026

RESUMEN

El problema de Calendarización para Talleres de Manufactura (Job Shop Scheduling Problem) es un problema de optimización combinatoria que busca programar un conjunto de tareas en distintas máquinas de manera óptima, respetando restricciones como secuencialidad, capacidad de recursos y no interrupciones. Este problema, es de gran relevancia en entornos industriales, está orientado a minimizar criterios como el tiempo de producción. El JSSP pertenece a los problemas NP, ya que no existe un método exacto que lo pueda resolver en tiempo polinomial. Además, su complejidad lo ubica dentro de los problemas NP más difíciles conocidos como NP-duros.

En esta tesis se aborda la implementación de una nueva estructura de vecindad variable para el problema del JSSP. La estructura propuesta se centra en permutar únicamente pares adyacentes que forman parte de una o varias rutas críticas y que pasan por una línea de tiempo, identificadas mediante el método del CPM (Critical Path Method). La Estructura de Vecindad Variable en Función de su Línea de Tiempo (EVVFLT) se caracteriza por su rapidez, ya que al enfocarse en operaciones adyacentes dentro de una o varias rutas críticas, reduce significativamente el número de evaluaciones necesarias en cada iteración de esta forma se realizan generalmente permutaciones en la ruta crítica. Aunque logra impactar directamente en varias operaciones críticas, esta estrategia también revela ciertas limitaciones.

El método empleado demanda tiempo considerable en la corrección de ciclos dada la representación simbólica utilizada, lo cual, en ocasiones, desvía la búsqueda hacia soluciones demasiado alejadas de la vecina esperada, que debería guardar gran similitud con la anterior al permutar uno o varios pares adyacentes. Además, deja fuera operaciones que, aun perteneciendo a la ruta crítica, carecen de un par adyacente con el cual intercambiarse. Para superar estas barreras y no quedar atrapados en óptimos locales, se recurre a una meta heurística del algoritmo de recocido simulado con recalentamiento (Simulated Annealing). Esta técnica, que combina con elegancia la exploración y la explotación, permite abrir caminos hacia soluciones de mayor calidad.

ABSTRACT

The Job Shop Scheduling Problem (JSSP) is a combinatorial optimization problem that aims to schedule a set of tasks across different machines in an optimal way, while satisfying constraints such as precedence, resource capacity, and non-preemption. This problem is highly relevant in industrial environments, as it is often oriented toward minimizing production time. The JSSP is classified as NP-hard, since the search space grows exponentially with problem size. Its complexity is further increased by the interdependence between machines and jobs, where operations depend on others performed on different machines.

This thesis addresses the implementation of a novel Variable Neighborhood Structure in Function of its Timeline (EVVFLT) for the JSSP. The proposed structure focuses on permuting only adjacent pairs of operations that belong to one or more critical paths along a timeline, identified through the Critical Path Method (CPM). The EVVFLT is characterized by its efficiency, as it significantly reduces the number of evaluations per iteration by concentrating on adjacent operations within the longest path.

While this strategy directly impacts several critical operations, it also presents limitations. The method requires considerable time for cycle correction, which may at times drive the search toward solutions too distant from the expected neighbor. Moreover, it overlooks critical operations that lack an adjacent pair for swapping. To address these limitations and avoid entrapment in local optima, a metaheuristic based on the Simulated Annealing with reheating algorithm is employed. This approach effectively balances exploration and exploitation, thereby improving solution quality and expanding the search toward more promising regions of the solution space.

GLOSARIO

CPM	Método de gestión de proyectos utilizada para planificar y coordinar las actividades de un proyecto.
Metaheurísticas	Procedimientos de métodos aproximados que están diseñados para resolver problemas difíciles de optimización combinatoria.
Heurísticas	Métodos sencillos empíricos probados y aportados por investigadores.
Benchmarks	Colección de datos (instancias) de referencias utilizados para medir y comparar resultados de experimentación para un determinado tipo de problema.
Calendarización	Programación de tareas con relación a los recursos disponibles secuenciados de la mejor manera.
Instancia	Datos de entrada de diferentes tamaños que representan casos específicos de un tipo de problema de calendarización con valores para evaluar resultados en este caso de calendarización.
Makespan	Duración total de una calendarización o programación de recursos que indica el tiempo final de la última tarea asignada.
Tiempo Polinomial	Tiempo razonable que tarda un algoritmo en resolver un problema, el número de pasos que necesita el algoritmo está limitado por una potencia del tamaño de entrada, como n, n^2, n^3, etc.

DEDICATORIA

A mi madre, **Hortensia Campusano Ortuño**, por su gran amor incondicional y su entrega hasta el último momento. Siempre se preocupó profundamente por sus hijos, y especialmente por mí, su bebé Pedro Bello. Su legado de resiliencia y reflexión me acompaña día a día; con sus palabras, “el corazón sabe cuándo actúa mal”, me enseñó a vivir con honestidad y conciencia.

A mi padre, **Alberto Bello Sánchez**, hombre de familia humilde y campesino, que siempre soñó con darnos una vida mejor. Desde pequeño nos inculcó metas altas: al nacer mi hermana, dijo que sería secretaria; al nacer yo, que sería licenciado y maestro, un sueño que él nunca pudo alcanzar. Aunque nos dejó a los siete años, su presencia y guía siempre han estado con nosotros, y hoy sus deseos se reflejan en nuestros logros y mientras haya vida vamos por más. A morirnos sabios, que solo así cada día entiendo el propósito de mi vida. Saludos donde quiera que se encuentren, Doña Horte y Don Beto Bello.

A mis hermanos **María de Jesús, María Manuela, Miguel Ángel y Nicolás**, por ser una fuente constante de inspiración, apoyo y cariño. Cada uno de ustedes ha contribuido de manera especial en mi vida, brindándome fortaleza en los momentos difíciles y alegría en cada logro alcanzado. Esta meta también les pertenece, porque sin su compañía y aliento, este camino habría sido mucho más difícil.

Con todo mi cariño y gratitud,

AGRADECIMIENTOS

En primer lugar, deseo expresar mi profundo agradecimiento al **Consejo Nacional de Humanidades, Ciencias y Tecnología (CONAHCYT)** por el invaluable apoyo económico brindado, sin el cual la realización de este trabajo no hubiera sido posible.

A mi familia **Bello Campusano**, por su respaldo incondicional, por su comprensión en los momentos difíciles y por ser siempre un motor de inspiración y fortaleza en cada etapa de mi vida académica y personal.

De manera muy especial, extiendo mi más sincero agradecimiento a mi director de tesis, Dr. **Marco Antonio Cruz Chávez** quien ha sido un gran ejemplo a seguir. Desde que yo cursaba el bachillerato, hace más de treinta años, lo tuve como un referente y estándar en mi mente. A lo largo del tiempo, la vida me llevó por otros caminos que me alejaron de los estudios; sin embargo, cuando se presentó la oportunidad de retomar mi formación académica y alcanzar las metas que alguna vez me propuse, tuve la dicha de reencontrarlo. Ha sido para mí no solo un excelente asesor y tutor, sino también un muy buen amigo, cuyo apoyo y guía han dejado una huella imborrable en mi trayectoria.

Asimismo, expreso mi sincero agradecimiento a mi **comité tutoral**, quienes con sus valiosas observaciones, sugerencias y orientaciones me ayudaron a darle una mejor estructura y claridad a esta tesis. Gracias a sus comentarios y aportaciones, este trabajo alcanzó una mayor solidez académica y metodológica.

ÍNDICE

LISTA DE FIGURAS

LISTA DE TABLAS

LISTA DE SEUDOCÓDIGO

1 INTRODUCCIÓN

1.1 Antecedentes

El problema de calendarización en talleres de manufactura, conocido como Job Shop Problem (JSP), surgió en la década de 1950 como una respuesta a las necesidades derivadas del crecimiento acelerado de la industria manufacturera. Uno de los primeros avances significativos en este ámbito fue el trabajo de **Johnson (1954),** quien desarrolló un algoritmo eficiente para problemas de secuenciación en talleres de manufactura con dos máquinas, marcando el inicio en el estudio de la programación de operaciones.

El JSP es un problema complejo, el cual consiste en procesar varias operaciones individuales que son requeridas para terminar varios trabajos en varias máquinas. Este es un problema de optimización combinatoria donde el espacio de soluciones es $(N!)^m$ que crece de manera exponencial a medida que se aumentan las máquinas y los trabajos. Un problema pequeño de 6X6, tiene un total de 1.393140695040E17 combinaciones. Existen tres variaciones del Job Shop Problem (JSP), siendo este el problema principal, en el que un conjunto de trabajos debe procesarse en máquinas específicas siguiendo un orden estricto y respetando restricciones de precedencia. Una de sus variantes, el Job Shop Scheduling Problem (JSSP), amplía este enfoque al incorporar la secuenciación de tiempos de procesamiento, centrándose en programar con precisión el inicio y la finalización de cada operación para optimizar criterios como el makespan o el retraso en las actividades.

Otra variante, el Flexible Job Shop Problem (FJSP) introduce flexibilidad, permitiendo que una operación pueda realizarse en distintas máquinas elegibles, lo que implica tomar decisiones tanto de asignación de máquina como de secuenciación. Finalmente, otra variante de gran relevancia es el Open Shop Problem (OSP) el cual relaja aún más las restricciones, eliminando la necesidad de seguir un orden fijo entre las operaciones de un mismo trabajo; aunque cada operación sigue asignada a una máquina específica, su ejecución puede realizarse en cualquier orden, ofreciendo mayor libertad en comparación con el JSP. La complejidad aumenta al avanzar de JSP a FJSP, mientras que el OSP, aunque más manejable, puede ser menos restrictivo que los anteriores.

En resumen, debido a la complejidad del JSSP y la ausencia de soluciones exactas eficientes, las heurísticas y meta heurísticas continúan siendo herramientas clave para su abordaje. Sin embargo, es necesario seguir desarrollando métodos que superen las

limitaciones actuales. En este contexto, las consideraciones de múltiples rutas críticas mediante estructuras de vecindad representan una estrategia prometedora para mejorar la calidad de las soluciones y ampliar las posibilidades de optimización en estos problemas de calendarización complejos.

1.2 Planteamiento del problema

El problema de calendarización de talleres de manufactura, conocido como Job Shop, es una cuestión fundamental en el área de la optimización combinatoria. Fue identificado como un problema de complejidad NP-Completo por **Garey y Johnson (1979) y Papadimitriou y Steiglitz (1998) lo** que significa que, hasta ahora, no se ha encontrado un método eficiente, capaz de resolverlo de manera óptima dentro de un tiempo razonable. Esta clasificación subraya la dificultad inherente del problema y resalta la necesidad de enfoques aproximados o heurísticos para abordar su resolución en escenarios prácticos.

Estas condiciones definen un entorno altamente estructurado donde se busca optimizar criterios como minimizar el tiempo total de finalización (makespan) o mejorar la utilización de los recursos disponibles. El JSSP se caracteriza por manejar una cantidad n de trabajos j, cada uno compuesto por una secuencia de operaciones O que deben completarse en un orden específico para finalizar el proceso. Estas operaciones se asignan a m máquinas, que son recursos compartidos con las siguientes restricciones:

- Capacidad exclusiva de las máquinas: Una máquina puede procesar únicamente una operación a la vez.
- No interrupciones: Una vez iniciada, una operación debe completarse sin pausas.
- Irreversibilidad de los trabajos: Los trabajos no pueden ser cancelados ni interrumpidos durante su procesamiento.
- Exclusividad de los trabajos: Un trabajo no puede estar siendo procesado en más de una máquina al mismo tiempo.
- Secuencia obligatoria: Cada operación de un trabajo debe comenzar únicamente cuando la operación previa haya finalizado.
- Restricciones de precedencia: El orden de las máquinas asignadas a las operaciones es fijo y no puede ser alterado.

1.3 Complejidad del problema

La teoría de la complejidad computacional es una rama de la informática que analiza los recursos necesarios, como tiempo y espacio, para resolver problemas mediante computadoras. Los recursos incluyen la memoria utilizada por el sistema y la duración que toma ejecutar un proceso. Los problemas se pueden categorizar según su nivel de dificultad, y la teoría de la complejidad los clasifica en las siguientes categorías:

1) Problemas tipo *P* (Tiempo Polinomial): Son aquellos que pueden resolverse computacionalmente de forma eficiente, dentro de un tiempo polinomial razonable utilizando métodos exactos. Para estos problemas existen algoritmos de tipo polinomial acotados con funciones asintóticas tales como $O(\log n)$ para los logarítmicos, $O(n)$ para los lineales, $O(n^2)$ para los cuadráticos, y $O(n^3)$ para los cúbicos. El diseño de algoritmos busca optimizar la complejidad, prefiriendo aquellos con comportamiento logarítmico, ya que garantizan tiempos de ejecución razonables independientemente de la capacidad del equipo.

2) Problemas *NP* (No Polinomial): Los problemas *NP* comprenden aquellos cuya solución no puede verificarse en tiempo polinómico. Algunos problemas *NP* pueden reformularse de manera que pasen a resolverse como problemas tipo *P* al relajar ciertas restricciones del mismo.

3) Problemas NP-Completos: Son considerados los más difíciles dentro del conjunto de problemas NP debido a su gran complejidad. Estos problemas al igual que la familia NP requiere una cantidad excesiva de recursos computacionales, y hasta ahora no se conocen algoritmos exactos capaces de encontrar soluciones óptimas. Las aproximaciones son la única alternativa práctica para abordar este tipo de problemas.

La teoría de la complejidad computacional es fundamental para entender y clasificar problemas, diseñando algoritmos que optimicen el uso de recursos y permitan abordar desafíos en computación.

1.4 Objetivos

1.4.1 Objetivo General

Diseñar e implementar una estructura de vecindad de tamaño variable que integre un mecanismo para identificar múltiples rutas críticas, con el propósito de realizar permutaciones en las operaciones pertenecientes a dichas rutas críticas y evaluar su impacto en la optimización.

1.4.2 Objetivos específicos

1. Diseñar una nueva estructura de vecindad que permita identificar múltiples rutas críticas dentro del JSSP, integrando un enfoque flexible de tamaño variable para explorar las permutaciones de operaciones.

2. Desarrollar un mecanismo para generar soluciones iniciales factibles de manera aleatoria, garantizando el cumplimiento de las restricciones y proporcionando una base para la optimización.

3. Implementar la nueva estructura de vecindad dentro de un algoritmo de búsqueda local, evaluando su eficacia para explorar soluciones vecinas y mejorar los tiempos de programación.

4. Realizar un análisis de los resultados obtenidos, comparando la eficacia de la estructura propuesta frente a métodos tradicionales y evaluar su impacto en la optimización del makespan y otras métricas clave del JSSP.

1.5 Alcances

El estudio se limita a analizar cómo la perturbación de las operaciones de las rutas críticas en una línea de tiempo puede impactar la mejora de las soluciones. En este enfoque, se permutan operaciones que pertenecen a las rutas críticas, lo cual busca enfocar la exploración del espacio de soluciones en regiones relevantes del problema. Se espera con ello alcanzar soluciones óptimas o muy cercanas a las mejores cotas conocidas, incrementando así la eficiencia del proceso de optimización.

1.6 Hipótesis

Si en una solución se identifican múltiples rutas críticas y estas se descomponen estratégicamente en una línea de tiempo, será posible realizar permutaciones efectivas en las operaciones pertenecientes a dichas rutas, lo que podría acelerar significativamente la obtención de una nueva solución optimizada, mejorando el makespan y reduciendo la complejidad del proceso de búsqueda.

1.7 Justificación

La identificación y comprensión de múltiples rutas críticas en el JSSP refleja la complejidad y diversidad de escenarios operativos que se presentan en entornos industriales reales. Cada ruta crítica puede representar una combinación única de tareas, recursos y restricciones que afectan el tiempo de finalización de un proyecto o un producto. Al abordar estas rutas críticas de manera implícita, se abre la posibilidad de mejorar la planificación y la eficiencia en los sectores productivos, optimizando el uso de los recursos y reduciendo los tiempos de espera. Además, en la literatura existente sobre la identificación y gestión de múltiples rutas críticas en el JSSP es aún muy limitada, lo que ofrece una oportunidad significativa para contribuir con nuevas soluciones y métodos que puedan ser aplicados en la práctica industrial, mejorando la competitividad y productividad. Además, los problemas grandes aun no tienen solución, lo que representa un área de oportunidad para aportar nuevas técnicas.

1.8 Organización de la Tesis

El capítulo 1 presenta el contexto teórico del Job Shop Problem (JSP), explicando su complejidad y variantes como el Job Shop Scheduling Problem (JSSP), el Flexible Job Shop Problem (FJSP) y el Open Shop Problem (OSP). Se destaca que el JSSP es un problema NP-Completo y es muy difícil de resolver, lo que justifica el uso de heurísticas y meta heurísticas para encontrar soluciones aproximadas. Además, se plantean los objetivos del estudio, que se enfocan en el diseño e implementación de una estructura de vecindad para identificar y manipular múltiples rutas críticas dentro del JSSP, permitiendo permutaciones estratégicas que mejoren el tiempo de producción (*makespan)*.

El capítulo 2 en el estado del arte aborda el problema de calendarización de trabajos en talleres de manufactura, exponiendo su origen, clasificación y modelado matemático. Además, se presenta un análisis detallado de investigaciones previas, destacando las

técnicas y enfoques más relevantes utilizados para resolver este problema, incluyendo métodos exactos, heurísticos y meta heurísticos, con el fin de contextualizar los avances y desafíos actuales en esta área.

En el capítulo 3 se detalla la metodología utilizada en esta investigación, presentando el algoritmo diseñado para resolver el problema y su estructura. Se explican a fondo la organización del vecindario, el comportamiento de los algoritmos y el procedimiento para explorar y mejorar soluciones.

El capítulo 4 expone los resultados obtenidos tras evaluar instancias clásicas de la literatura mediante el algoritmo Estructura de Vecindad Variable en Función de su Línea de Tiempo (EVVFLT) propuesto. Estos resultados se comparan con los valores óptimos previamente registrados para algunas instancias. También se lleva a cabo la estadística inferencial.

Finalmente, en el capítulo 5 se presentan posibles líneas de investigación futuras que pueden desarrollarse a partir de esta tesis. Una de las principales conclusiones obtenidas es que cada tipo de implementación en la calendarización produce una representación simbólica distinta de sus rutas críticas.

2 ESTADO DEL ARTE

2.1 Introducción

La calendarización y secuenciación de tareas en entornos de producción ha sido un desafío fundamental desde mediados del siglo XX, impulsado por el crecimiento acelerado de las industrias y la necesidad de optimizar recursos en sistemas cada vez más complejos. En este contexto, surgen problemas clásicos como el Job Shop Problem (JSP), que establece las bases para modelar la asignación de trabajos a máquinas específicas, respetando restricciones de precedencia y secuenciación. Este problema marcó el inicio de una línea de investigación que se expandió rápidamente debido a su aplicabilidad en la manufactura, logística y gestión de recursos.

2.2. Antecedentes

Los antecedentes muestran que, en la calendarización del JSSP, generalmente se encuentran múltiples rutas críticas. La mayoría de los investigadores emplean heurísticas y meta heurísticas para realizar movimientos perturbadores, pero suelen aplicar su metodología únicamente a una sola ruta crítica. Como consecuencia, existen muy pocas investigaciones que aborden el caso de múltiples rutas críticas.

Este contexto constituye la base de nuestra investigación, la cual busca realizar permutaciones de pares adyacentes en un intervalo de tiempo identificado mediante el método CPM (Critical Path Method), y que pertenezcan a varias rutas críticas o, al menos, a una de ellas.

El Job Shop Problem (JSP) en la calendarización de talleres de manufactura tiene sus raíces en el trabajo de **Johnson (1954),** quien presentó un escenario en el que dos máquinas procesaban diversas etapas de producción para varios productos. En su investigación, se concluyó que es posible encontrar un método óptimo de solución al trabajar con solo dos máquinas. Sin embargo, con tres o más máquinas este método ya no funciona y pierde su practicidad.

En **Manne** (**1960),** propuso el uso de técnicas de programación lineal para encontrar soluciones óptimas al JSP, destacando su capacidad para abordar las restricciones de secuenciación y la no interferencia entre las máquinas. Sin embargo, aunque esta formulación resulta costosa para problemas de gran tamaño, se sugiere que podría ser útil aplicarla en situaciones específicas, considerando los costos computacionales involucrados.

En **Lomnicki (1965)** destaca que, aunque **Johnson (1954)** desarrolló un método eficaz para encontrar soluciones óptimas en problemas con dos máquinas, los casos con tres o más máquinas *(m ≥ 3)* aún carecen de un enfoque similar. Para abordar esta limitación, Lomnicki adaptó al JSP el método de Ramificación y Poda (Branch and Bound), introducido por **Heuser et al. (1963)** en el contexto del Problema del Agente Viajero (Traveling Salesman Problem), otro desafío clásico en la optimización combinatoria. Esta adaptación permitió obtener soluciones óptimas en instancias específicas del JSP.

Garey y Johnson (1979) en su libro Computers and Intractability: A Guide to the Theory of NP-Completeness abordan el concepto de NP-completos, una categoría de problemas computacionales difíciles de resolver. Proporcionan una introducción exhaustiva a los problemas NP-completos y NP-duros, explorando su teoría, presentando ejemplos y técnicas para mostrar la complejidad de estas tareas. El libro discute el impacto de la NP-completitud en la calendarización y el diseño de algoritmos, explicando por qué algunos problemas no tienen soluciones eficientes y cómo enfrentarlos.

En **Festa (2014)** se ofrece una visión general de los algoritmos exactos en la optimización combinatoria compleja, destacando el uso de programación lineal aproximada y meta heurísticas. Comienza con métodos exactos como ramificación y poda, y programación dinámica, antes de abordar estrategias de aproximación más comunes, como la búsqueda local, búsqueda con vecindad variable, métodos aleatorios adaptativos y algoritmos voraces combinados con búsqueda dispersa. El estudio concluye que, aunque algunos problemas de optimización combinatoria pueden resolverse de manera exacta, muchos son intratables desde el punto de vista computacional, lo que lleva al uso de técnicas de aproximación y heurísticas cuando los métodos exactos no son viables.

Kirkpatrick (1983) presenta el recocido simulado como una técnica para optimización, inspirada en el enfriamiento de metales. El algoritmo explora el espacio de soluciones evitando óptimos locales al aceptar soluciones peores con una probabilidad decreciente. Detalla parámetros como la temperatura y la tasa de enfriamiento, mostrando cómo puede encontrar soluciones óptimas globales. La técnica se aplica eficazmente en diversos problemas de optimización, destacándose como una herramienta útil en investigación de operaciones y diseño de algoritmos.

Jorapur et al. (2016) emplean un algoritmo genético para abordar de forma eficiente los problemas de calendarización de talleres. Su estudio pone especial énfasis en la generación de la población inicial y en los operadores de recombinación. A partir de una población inicial bien definida, es posible obtener soluciones más prometedoras que abarquen distintos niveles de complejidad. La inicialización se realiza utilizando una representación basada en trabajos, combinada con una asignación aleatoria de estos y

operadores de reproducción como cruce de un punto, cruce de tres puntos y mutaciones.

Kelley y Walker (1959) introdujeron el Método de la Ruta Crítica (CPM), El CPM permite construir un diagrama de red, calcular los tiempos de inicio y fin de las actividades, e identificar la secuencia crítica que define la duración mínima del proyecto. Inicialmente aplicado en la construcción, el CPM se expandió a sectores como la manufactura, destacándose por mejorar la eficiencia y reducir costos. Su aporte es considerado un pilar en la gestión de proyectos.

En **Levy et al. (1963),** la ruta crítica de un proyecto se define como la secuencia de actividades más larga en duración desde el inicio hasta el final del proyecto. Esta ruta determina el tiempo mínimo necesario para completar el proyecto en su totalidad. El tiempo total de cada camino se obtiene sumando los tiempos de todas las actividades que lo conforman. El método de la ruta crítica es una técnica poderosa, pero conceptualmente sencilla, utilizada para analizar, planificar y programar proyectos grandes y complejos. Esta herramienta permite identificar qué actividades tienen un impacto directo en la duración total del proyecto y cuál es la mejor forma de programar todas las tareas involucradas, con el fin de cumplir una fecha objetivo al menor costo y tiempo posible.

La estimación precisa de los tiempos en proyectos con alta incertidumbre es un desafío clave en la gestión de proyectos. En este contexto, **Slyke (1963)** explora en su artículo la aplicación de métodos Monte Carlo para mejorar la estimación de tiempos en proyectos gestionados con la técnica PERT que también identifica rutas críticas. Al incorporar simulaciones Monte Carlo, se logra modelar la incertidumbre de manera más precisa, generando distribuciones probabilísticas que permiten prever con mayor exactitud los tiempos de finalización del proyecto. Este enfoque proporciona a los gestores herramientas más eficaces para la toma de decisiones bajo incertidumbre, ampliando así las capacidades del método PERT en la planificación y programación de proyectos.

La gestión eficiente del tiempo y los recursos es clave en la programación de proyectos, especialmente en aquellos con múltiples actividades interrelacionadas. En este contexto, **Alexander et al. (1995)** presenta en su artículo algoritmos que identifican y optimizan rutas casi críticas en gráficos de actividades de proyectos. Estos métodos mejoran la planificación al considerar rutas que, aunque no sean las más críticas, influyen significativamente en los tiempos de finalización y el uso de recursos. El estudio evalúa la eficacia de estos algoritmos frente a técnicas tradicionales, destacando su utilidad en la programación de proyectos complejos.

Por otro lado, la calendarización de talleres de manufactura ha evolucionado considerablemente a lo largo del tiempo, impulsada por la necesidad de optimizar la asignación de recursos y mejorar la eficiencia en la producción. En este contexto, **Jain y Meeran (1999)**, en su artículo Deterministic Job-Shop Scheduling: Past, Present and Future, ofrecen una revisión integral sobre los avances en este campo. El estudio abarca desde los primeros enfoques tradicionales hasta técnicas más avanzadas, incluyendo algoritmos exactos y heurísticos. Además, los autores destacan tendencias emergentes y áreas que requieren mayor investigación, proponiendo nuevas direcciones para optimizar la programación en entornos industriales. Este trabajo se ha consolidado como una referencia fundamental en el estudio del Job-Shop Scheduling.

En el trabajo de **Satake et al. (1999)** se propone una estructura de vecindad alternativa a las basadas en la ruta crítica para el problema de programación Job-Shop (JSSP). El enfoque se basa en la representación de una solución mediante una gráfica de Gantt. A partir de esta representación se identifican los espacios de ocio o tiempos de holgura entre operaciones. Sobre estos espacios se realizan permutaciones de operaciones que permiten generar nuevas soluciones factibles. La idea está inspirada en el proceso intuitivo de calendarización manual realizado por una persona al observar un diagrama de Gantt. Las permutaciones se realizan considerando restricciones de precedencia y de recursos.

Heuser y Wynne (1963) presentan un estudio de caso que aplica el Método de la Ruta Crítica (CPM) en la calendarización de trabajos en talleres industriales. En este trabajo, se demuestra cómo el CPM puede adaptarse para coordinar múltiples tareas en entornos productivos complejos. A través de la identificación de actividades críticas y la optimización del cronograma de actividades, lograron reducir el tiempo total de procesamiento y mejorar el uso de los recursos. Este enfoque destaca la versatilidad del CPM, extendiendo su aplicación más allá de la gestión de proyectos tradicionales hacia la calendarización eficiente en entornos industriales.

La gestión de proyectos en entornos con alta incertidumbre requiere herramientas que permitan modelar de forma precisa la variabilidad en los tiempos de ejecución. En este contexto, **Chanas y Zieliński (2001)** exploran el análisis de rutas críticas en redes de proyectos con tiempos de actividad inciertos, representados mediante conjuntos difusos. Los autores proponen un modelo basado en la teoría de conjuntos difusos para calcular la ruta crítica considerando esta incertidumbre. Esta propuesta amplía las técnicas tradicionales de análisis de rutas críticas, ofreciendo una herramienta más flexible para abordar escenarios donde los tiempos no pueden definirse con exactitud.

Steinhöfel et al. (2003) analizan la eficacia de las técnicas de búsqueda en problemas de optimización combinatoria y cómo estas pueden quedar atrapadas en soluciones locales de baja calidad. En su investigación contabilizan las rutas críticas y observan que, conforme las soluciones mejoran, el número de rutas críticas disminuye progresivamente, hasta plantear que la solución óptima presenta únicamente una ruta crítica. Este resultado ofrece una visión estructural valiosa, ya que vincula la calidad de la solución con la simplificación de las rutas críticas, aportando un criterio adicional para evaluar el progreso en la optimización. Aunque su investigación deja muchas dudas, ya que no explican cómo obtienen los resultados.

Cruz-Chávez y Frausto-Solís (2006) introducen el algoritmo CPA (Critical Path Approximation), diseñado para identificar de manera eficiente el camino crítico (CPA). Este algoritmo selecciona un conjunto de operaciones en el JSSP, garantizando que el 99% de ellas pertenezcan al CPA. Al compararlo con el método de la ruta crítica (CPM), se demuestra que CPA es más eficiente en términos de costo y rendimiento, especialmente cuando se utiliza en conjunto con el algoritmo de recocido simulado. Además, se destaca la influencia significativa de la estructura de vecindad en el desempeño de las meta heurísticas, enfatizando la necesidad de evaluarla constantemente.

Ravi et al. (2010) propone una mejora al algoritmo de optimización por colonias de hormigas (ACO) para resolver problemas de gestión de proyectos mediante el Método de Camino Crítico (CPM). El enfoque incorpora una estimación empírica de duraciones de actividades, lo que permite identificar de manera más efectiva caminos críticos. Los resultados muestran que el ACO supera al CPM tradicional en redes de proyectos con tiempos determinísticos, ofreciendo soluciones de mayor calidad para la planificación y control de proyectos complejos.

Banu et al. (2014) realiza una revisión exhaustiva de las técnicas de inteligencia artificial aplicadas. Se analizan diversas metodologías, incluyendo redes neuronales, algoritmos genéticos, sistemas multi-agentes, recocido simulado, optimización por colonias de abejas y hormigas, y algoritmos de enjambre de partículas. Estas técnicas, aunque no garantizan soluciones óptimas, son valoradas por su capacidad para proporcionar soluciones de alta calidad en tiempos reducidos, destacándose como herramientas eficaces en la resolución de problemas de optimización combinatoria complejos. El estudio también resalta que, a pesar de que la mayoría de las investigaciones se centran en la aplicación de algoritmos a problemas de referencia, las técnicas de IA ofrecen resultados prácticos y efectivos en la calendarización de talleres de manufactura, superando a métodos heurísticos tradicionales.

Cruz-Chávez (2014) propone un mecanismo para la generación de vecindad. Este enfoque consiste en generar un vecino factible mediante la permutación de un par de operaciones adyacentes. A diferencia de otros métodos, este mecanismo se basa únicamente en la identificación de una ruta crítica. Se enfoca en permutar operaciones que no presenten tiempo de holgura. Mediante fundamentos teóricos y experimentales, se demuestra que esta estrategia garantiza la factibilidad del vecino generado.

Majid et al. (2017) En las últimas décadas, ha habido un creciente interés en enfoques meta heurísticos para resolver el JSSP, ya que estos pueden generar soluciones superiores a las obtenidas con heurísticas tradicionales. Este artículo clasifica los JSSP en 14 categorías, incluyendo variantes deterministas, flexibles, estáticas, dinámicas, cíclicas, y estocásticas, entre otras. El JSSP determinista, uno de los más comunes, implica que los trabajos y tiempos de procesamiento son conocidos y fijos. Las clasificaciones de JSSP también pueden basarse en el proceso de llegada de trabajos, distinguiendo entre JSSP estáticos, donde todos los trabajos y máquinas están disponibles desde el inicio, y dinámicos, que manejan la llegada y disponibilidad de trabajos con variación en tiempo.

Sun et al. (2018) introduce el algoritmo de enjambre de salpas (colonias gelatinosas marinas) basado en bloques en el camino crítico (SSA_BCP) para minimizar el makespan en el problema de calendarización de trabajos reentrantes (RJSSP) y se diseña una regla de codificación extendida llamada reentrant-smallest-order-value (RSOV) para transformar los individuos del SSA en permutaciones de trabajos. Además, se definen cuatro estructuras de vecindario y una operación de inserción basada en bloques para evitar búsquedas en regiones no válidas, implementando una búsqueda local eficiente que integra múltiples vecindarios. Los resultados de simulación y comparaciones demuestran la efectividad del SSA_BCP en comparación con otros métodos.

En **Kurdi (2019)** las meta heurísticas como los algoritmos genéticos (GA) se han utilizado ampliamente para resolver el JSSP. Sin embargo, una limitación frecuente es que los operadores de cruce (crossover) intercambian componentes de los padres de forma más o menos aleatoria, lo que puede llevar a perder los rasgos importantes de los padres de calidad.

Para superar esta debilidad, Kurdi propone un GA denominado GA-CPG-GT que integra dos innovaciones: primero, utiliza la información del camino crítico de la solución para identificar los genes (material genético) que contienen rasgos relevantes; segundo, aplica un operador de cruce basado en el algoritmo de Bernard Giffler y George Thompson modificado para aprovechar ese conocimiento y guiar el intercambio genético. El autor

pone a prueba su propuesta sobre instancias de referencia, compara versiones con y sin la guía de la ruta crítica, y también la contrasta con cinco trabajos previos similares. Los resultados muestran mejoras en calidad de solución gracias al intercambio selectivo propuesto.

En **Yasser (2020**) se analiza las limitaciones del método tradicional CPM en la programación de proyectos y propone una versión mejorada llamada Critical Path Segments (CPS). Este nuevo enfoque divide las actividades en segmentos de tiempo, usa solo relaciones final-a-inicio y realiza el análisis con un solo pase hacia adelante. Los casos de estudio demuestran que CPS mejora la precisión, el control y la identificación de actividades críticas, ofreciendo una herramienta más detallada y efectiva que el CPM, aunque el autor reconoce que aún existen aspectos por perfeccionar, como las restricciones temporales y el seguimiento de avances fuera de secuencia.

En **Ukamaka (2020)** se aplica tanto el método Critical Path Method (CPM) como el Program Evaluation and Review Technique (PERT) a un proyecto de construcción de vivienda en Nigeria, determinando mediante cálculos de pase hacia adelante y atrás las fechas más tempranas y últimas de ejecución de las actividades, así como la trayectoria crítica de cada modelo. Los resultados muestran que la duración estimada por CPM fue de 151 días, mientras que PERT estimó 150 días con una probabilidad de cumplimiento del 99.87 %, lo que indica que ambos métodos son eficaces, pero PERT ofrece una pequeña ventaja en precisión. Ukamaka concluye que, aunque la diferencia en tiempo es mínima, PERT demuestra ser un poco más eficiente que CPM para evaluar y programar proyectos de construcción, y que la implementación de cualquiera de los dos métodos aporta un control significativo del tiempo en la gestión de proyectos.

Goto y Murray (2020) proponen un marco para identificar todos los caminos más largos en un grafo dirigido ponderado, bajo la condición de que no existan ciclos de peso positivo, con el objetivo de determinar las rutas de mayor costo acumulado entre nodos en redes. Esta metodología es especialmente útil en la gestión de proyectos de ruta crítica, donde es esencial identificar y comparar estas rutas para establecer la cadena crítica. El enfoque integra la identificación de rutas y la minimización de la duración total del proyecto, abordando múltiples objetivos de optimización. Al reformular el problema como una satisfacción de restricciones en programación lineal entera mixta, se pueden encontrar soluciones óptimas que reducen la duración del proyecto.

En su trabajo **Alper et al. (2020)** adapta el algoritmo Single Seekers Society (SSS), originalmente diseñado para problemas de optimización continua, para abordar

problemas de optimización combinatoria. Esta adaptación modifica componentes y estructuras del SSS para ajustarse a la naturaleza discreta de los problemas combinatorios. El algoritmo modificado se aplica a diversos problemas, incluyendo el enrutamiento de vehículos con recogida y entrega simultánea, el enrutamiento con recogida y entrega mixtas, y los problemas de calendarización en flujo y en talleres. Los resultados experimentales muestran que el SSS adaptado ofrece un rendimiento satisfactorio en comparación con otros enfoques existentes

Colonia de hormigas en recocido simulado **Ying y Lin (2020)**. Este artículo presenta un recocido simulado de inicio múltiple con un algoritmo de calendarización de cambio bidireccional. Para mejorar la calidad de la solución, utiliza un procedimiento de horario de turno bidireccional que permite retrasar el horario y, en consecuencia, amplía el espacio de búsqueda. Simulando la búsqueda que hacen las hormigas. La principal contribución de este estudio es proporcionar un método alternativo que intenta modelar el comportamiento de como las hormigas inician su ruta y generan varias en busca de alimento.

Kyunghwan (2020) propone el método Generalized Resource Constrained Critical Path Method (eRCPM), una versión mejorada del camino crítico que incorpora las restricciones reales de recursos en la programación de proyectos de construcción. A diferencia del CPM tradicional, este método considera cómo la disponibilidad limitada de recursos puede alterar la secuencia y duración de las actividades, permitiendo identificar relaciones adicionales llamadas resource links. Con este enfoque, la calendarización refleja de manera más precisa las condiciones reales del proyecto y ofrece un análisis más confiable del tiempo disponible (float) y de las actividades realmente críticas. Kim concluye que el eRCPM mejora la precisión, continuidad y sostenibilidad en la planificación, al reducir retrasos y hacer más realista la gestión del tiempo y los recursos en proyectos de construcción.

El artículo de **Fan (2021)** propone un algoritmo híbrido basado en el meta-heurístico Jaya algorithm (que promueve soluciones hacia lo mejor y lejos de lo peor) para resolver el problema de programación de talleres flexibles (FJSP), poniendo un énfasis especial en la gestión de múltiples caminos críticos una complicación que muchas investigaciones previas han pasado por alto. El método integra operadores Jaya junto con estrategia de búsqueda tabú para refinar soluciones, empleando una codificación de dos vectores, y dirige el proceso de optimización hacia la reducción del makespan considerando la existencia simultánea de múltiples trayectorias críticas. En conclusión, los experimentos en instancias de referencia muestran que esta combinación alcanza mejoras significativas frente a algoritmos convencionales, demostrando que abordar

explícitamente múltiples caminos críticos en FJSP puede fortalecer la calidad de la programación y la utilización de recursos.

El Flexible Job Shop Scheduling Problem (FJSP) representa una extensión del problema clásico de programación de talleres, en la que cada operación puede asignarse a diferentes máquinas, aumentando la flexibilidad, pero también la complejidad de la planificación. **Yin et al. (2023)** abordan este problema mediante algoritmos evolutivos multiobjetivo, diseñados para explorar de manera eficiente el espacio de soluciones y equilibrar objetivos conflictivos, como minimizar el tiempo total de finalización y mejorar la utilización de recursos. Su estudio propone un enfoque adaptado al FJSP que combina representaciones específicas, operadores genéticos y mecanismos de búsqueda orientados a garantizar diversidad y calidad en las soluciones, incluyendo la identificación de múltiples rutas críticas para optimizar la eficiencia en entornos productivos dinámicos.

Bourreau et al. (2024) explora la integración del vector de Bierwith en el Algoritmo de Optimización Aproximada Cuántica (QAOA) para abordar JSSP utilizando enfoques cuánticos. El JSSP es representado por un gráfico disyuntivo orientado, donde las soluciones deben ser acíclicas para ser factibles. El vector de Bierwith (vector por repetición), introducido en 1995, permite mapear vectores a grafos disyuntivos acíclicos, facilitando la enumeración eficiente de soluciones. Este estudio demuestra cómo el vector de Bierwith puede integrarse en el QAOA para abordar el JSSP complejo de manera eficiente, destacando el potencial de los enfoques cuánticos en problemas de optimización.

En **Wang (2024)** aborda el problema de calendarización distribuida de talleres de manufactura (DJSP), que ha ganado relevancia en la manufactura moderna debido a la necesidad de asignar trabajos a fábricas distribuidas geográficamente y determinar la secuencia de operaciones en cada máquina. Para optimizar el makespan en el DJSP, se propone un modelo de programación lineal entera mixta (MILP) y un algoritmo memético auto-adaptativo (SMA). El SMA incorpora codificación independiente para la asignación de trabajos y la secuencia de operaciones, junto con estrategias de búsqueda local para mejorar la calidad de las soluciones. Los experimentos numéricos realizados en 120 instancias del DJSP, basadas en benchmarks de calendarización de talleres de trabajo, demuestran la eficacia del SMA, logrando la actualización de 30 récords y obteniendo los 91 mejores resultados en comparación con otros algoritmos existentes.

Hao et al. (2024) proponen un algoritmo híbrido basado en Path Relinking y Tabu Search (PRTS) para resolver el problema de calendarización de talleres de flujo híbrido (HFSP).

El enfoque busca minimizar el makespan superando las limitaciones de codificaciones tradicionales, especialmente en la gestión de múltiples rutas críticas. El PRTS utiliza codificación completa para representar todos los horarios semi-activos factibles y mejorar la exploración del espacio de soluciones. También introduce estructuras de vecindario adaptadas y una evaluación acelerada. Los resultados obtenidos en benchmarks muestran una mejora significativa frente a métodos anteriores, estableciendo nuevos mejores resultados para instancias complejas.

Sayoti et al. (2022) propusieron una adaptación del algoritmo Golden Ball (GB) para resolver el Job Shop Scheduling Problem (JSSP). El GB, inspirado en la dinámica del fútbol, busca optimizar el makespan mediante la interacción entre soluciones. Probado en 36 instancias de la OR-Library, el método mostró resultados competitivos y tiempos de cómputo reducidos frente a otros algoritmos, aunque los autores sugieren su hibridación futura para mejorar el rendimiento en algunas instancias.

Viana et al. (2020) desarrollaron la metaheurística mXLSGA (Multi-Crossover Local Search Genetic Algorithm), un algoritmo genético con búsqueda local y cruces múltiples que combina operadores genéticos con técnicas mejoradas de exploración. Este enfoque busca equilibrar la exploración y explotación del espacio de soluciones mediante estrategias de cruce variadas y refinamiento local. Aunque el método mostró un desempeño razonable y resultados competitivos en algunos casos, no superó significativamente a otros algoritmos existentes, evidenciando la necesidad de ajustes adicionales para mejorar su eficiencia y calidad de solución.

En **Mahmud et al. (2022)** trabajaron con algoritmos evolutivos híbridos con mecanismo de cambio basado en el desempeño de cada uno, combinando MODE, PSO y búsqueda local inspirada en Tabu Search, junto con estrategias de selección y verificación de diversidad. La variante sHEA mostró el mejor desempeño frente a los algoritmos constituyentes y a 26 métodos existentes. Sin embargo, la selección de operadores de MODE y la decisión de cambio podrían optimizarse considerando simultáneamente la calidad de la solución y la diversidad de la población. Además, la lista Tabú no registra todos los movimientos, y la estrategia de switching podría compararse con aprendizaje por refuerzo. Como extensión, los autores sugieren aplicar estos algoritmos a problemas más amplios de cadena de suministro para evaluar su efectividad en contextos complejos.

2.3 Representaciones del JSSP

2.3.1 Notación para JSSP

El problema del Job Shop está formado por un conjunto finito de trabajos con un número

de elementos n: $J = \{J1, J2, \ldots, Jn\}$, y $n = |J|$; un conjunto finito M de máquinas con cardinalidad m: $M = \{M1, M2, \ldots, Mm\}$ y $m = |M|$; y un conjunto de O que consiste en $n * m$ operaciones $O = \{i1, i2, \ldots, im*n\}$, donde $n, m \geq 1$. De estos conjuntos se tiene que para cada operación $i \in O$ existe un trabajo $J_k \in J \mid i \in J_k$ al que pertenece la operación, así como una máquina $Mk \in M$ en la que dicha operación debe ser procesada. Cada operación i tiene además un tiempo de inicio s y un tiempo de duración del proceso p, de tal forma que para una operación cualquiera $i \in O$, su tiempo final está dado por ($si + pi$), que es la suma de su tiempo de inicio más el tiempo de procesamiento.

El tiempo total de una calendarización o makespan es igual al tiempo en el que finaliza la última operación asignada, es decir por el tiempo final de la última operación calendarizada, que se representa como $MAX(si + pi)$. Este valor se obtiene mediante la función objetivo. A continuación, se muestra en la tabla 1 los elementos descritos de forma resumida:

	SIMBOLO	DESCRIPCIÓN
(1)	J	Conjunto de trabajos
(2)	n	Número total de trabajos
(3)	M	Conjunto de máquinas
(4)	m	Número total de máquinas
(5)	O	Conjunto de operaciones
(6)	$n * m$	Número total de operaciones
(7)	si	Tiempo de inicio de operación i
(8)	pi	Tiempo de proceso de la operación i
(9)	$(s_i + p_i)$	Tiempo de terminación de la operación i
(10)	$MAX (s_i + p_i)$	Máximo tiempo de terminación del conjunto de operaciones (makespan)

Tabla 1 Nomenclatura del modelo matemático

2.3.2 Modelo matemático

En el modelo matemático del JSSP, la ecuación (1) de la siguiente Fig. 1 describe la función objetivo como minimizar el makespan, representada por el tiempo de inicio más el tiempo de procesamiento de la última operación en la calendarización. En la ecuación (2) el conjunto de restricciones en la que el tiempo de inicio de cualquier operación debe

ser de mayor o igual que cero. La ecuación (3) se refiere al conjunto de restricciones de precedencia entre cualquier par de operaciones dentro de un mismo trabajo, donde la siguiente operación no puede iniciar antes que el tiempo de término de la anterior. Por último, la ecuación (4) se define como el conjunto de restricciones de capacidad de las máquinas, en la que cada máquina solo puede procesar una operación a la vez y esta debe terminarse antes que empiece la otra.

(1) $$Min\ f = [\frac{max\ (si + pj}{J\ \in\ O}]$$

(2) $$\forall j \in O \quad s_j \geq 0$$

(3) $$\forall i, j \in O \mid (i \prec j) \in J_k \quad s_i + p_i \leq s_j$$

(4) $$\forall\ i,j\ \in\ O \mid (i,j)\ \in\ Mk\ si\ +\ pi\ \leq\ sj\ \vee\ si\ +\ pi\ \leq\ si$$

Fig. 1 Modelo matemático

2.3.3 Modelo de grafo dirigido

EL JSSP también puede ser representado en la forma de un grafo dirigido $G(O, A, E)$, donde O es el conjunto de nodos del grafo, los cuales están conformados por las operaciones del problema y que tienen un valor igual a su tiempo de procesamiento, adicionalmente se tienen dos nodos ficticios "I" y "$*$" con valor de tiempo de procesamiento 0 y que representan el inicio y final del grafo. El conjunto A está formado por una serie de arcos dirigidos que denotan el orden de precedencia que existe entre operaciones pertenecientes a un mismo trabajo. Por último, el conjunto E define la relación entre operaciones que deben ser procesadas por una misma máquina mediante arcos no dirigidos. A continuación, en la Fig. 2 se muestra un ejemplo de grafo para una instancia de 3x3 (tres trabajos y máquinas por simplicidad ya que un grafo de 6X6 es muy difícil de entender y representar).

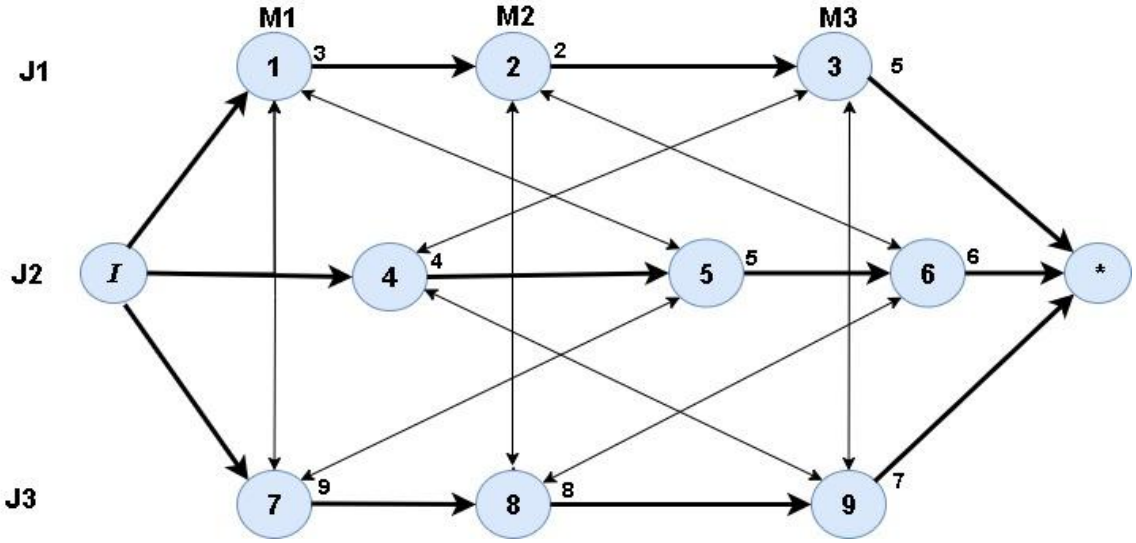


Fig. 2 Grafo dirigido

En este ejemplo se puede observar cómo los arcos dirigidos forman subconjuntos de operaciones que representan cada uno de los diferentes trabajos definidos en la instancia, de manera que el subconjunto de operaciones {1, 2, 3} corresponde al trabajo 1, el subconjunto {4, 5, 6} corresponde al trabajo 2 y el {7,8,9} al trabajo 3 El modelo de grafo dirigido también es usado para representar las diferentes calendarizaciones que se pueden obtener para una instancia cualquiera. La siguiente Fig. 3 muestra una posible calendarización para el ejemplo anterior de 3X3.

En este grafo se pueden observar tres subconjuntos de operaciones equivalentes a cada una de las tres máquinas disponibles. Estos subconjuntos muestran el orden específico en que las operaciones deben ser procesadas por cada máquina en esta calendarización. Se tiene entonces que la máquina 1 debe procesar en orden las operaciones {1,7,5,}, la máquina 2 debe procesar {2, 6, 8} y la máquina 3 las operaciones {4, 3, 9}. Para obtener el makespan a partir de esta representación, se debe recorrer el grafo teniendo en cuenta los valores arriba de los nodos que representan los tiempos de procesamiento.

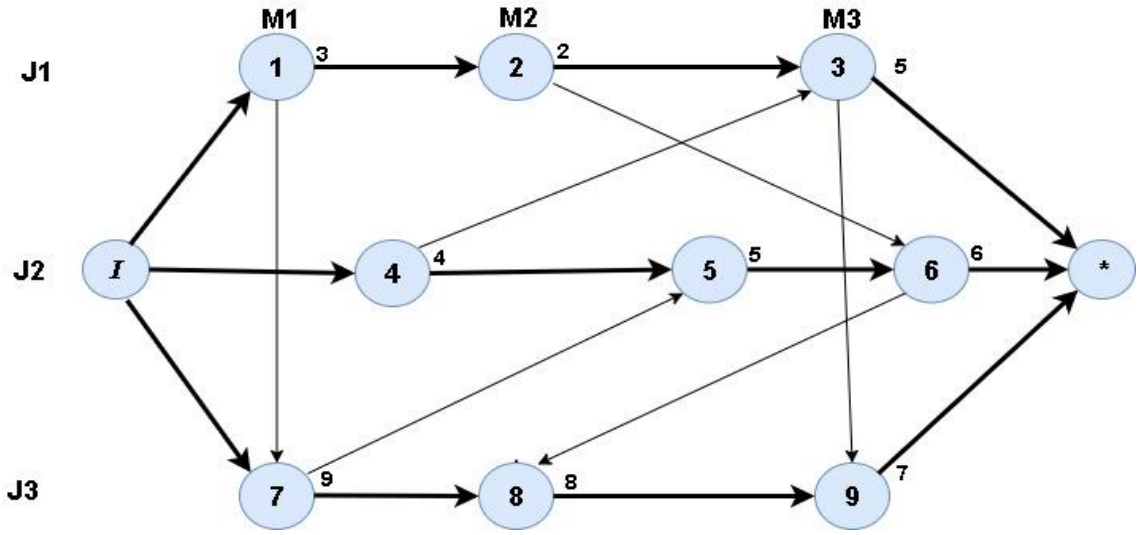


Fig. 3 Posible calendarización del JSSP

2.3.4 Diagrama de Gantt

Al analizar el diagrama de Gantt presentado en la Fig. 4 se pueden identificar aspectos relevantes sobre el comportamiento de la calendarización propuesta. En primer lugar, es posible observar cómo las operaciones de un mismo trabajo (representadas con el mismo color) se distribuyen a lo largo del eje temporal en distintas máquinas, respetando el orden de precedencia entre ellas.

Asimismo, se detectan espacios de inactividad (huecos) entre algunas operaciones dentro de ciertas máquinas, lo cual evidencia tiempos de ocio que podrían representar oportunidades de mejora en la utilización de los recursos. Por otro lado, la correcta asignación de operaciones sin traslapes en una misma máquina confirma que se han respetado las restricciones.

Esta visualización facilita la evaluación del makespan (tiempo total de procesamiento), ya que permite identificar visualmente la operación final de la calendarización. En conjunto, el diagrama de Gantt constituye una herramienta esencial para validar soluciones, asegurando que no se presenten traslapes entre operaciones.

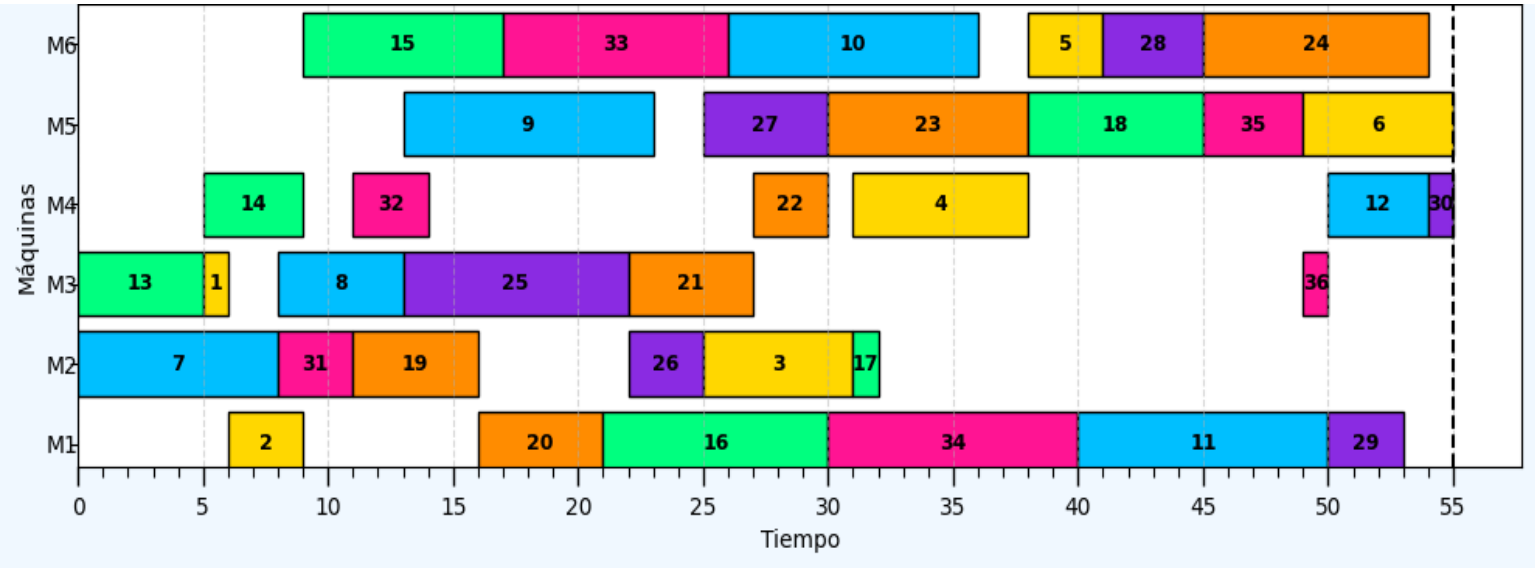


Fig. 4 Diagrama de Gantt

2.4 Estructuras de vecindad

Las estructuras de vecindad de acuerdo a **Mladenovic et al. (1997),** son reglas que determinan cómo moverse dentro del espacio de soluciones, definiendo qué soluciones se consideran vecinas de una solución actual. Estas reglas especifican los cambios pequeños o controlados que pueden aplicarse para generar nuevas alternativas, como intercambiar elementos, desplazar posiciones o modificar valores específicos. La estructura de Vecindad N1 es usada en esta tesis.

Principales estructuras de vecindad y sus usos:

1. N1 (Intercambio adyacente): permutar dos operaciones consecutivas en la misma máquina.

2. N2 (Inserción): mover una operación de una posición a otra dentro de la misma máquina.

3. N3 (Desplazamiento de bloque): mover un bloque continuo de operaciones a otra posición.

4. N4 (Inversión): invertir el orden de un bloque de operaciones consecutivas.

5. N5 (Intercambio de bloques): intercambiar dos bloques de operaciones entre posiciones distintas.

6. N6 (Ruido aleatorio): alterar aleatoriamente un bloque o parte de la secuencia.

7. N7 (Combinada o jerárquica): cambiar de tipo de vecindad progresivamente durante la búsqueda.

La Fig.5, titulada Bloque de operaciones, muestra un bloque conformado por tres operaciones que pertenecen a la ruta crítica y forman parte de la solución inicial. Para ilustrar la posibilidad de permutación de pares adyacentes, se utilizan los círculos que representan las operaciones 14 (azul), 4 (verde) y 12 (rosa). En este bloque, la operación 14 tiene como adyacente a la 4; la operación 4 tiene como adyacentes a la 14 y a la 12; y la operación 12 tiene como adyacente a la 4. Esta estructura indica que cada operación puede permutarse con su respectivo vecino, mostrando de manera práctica cómo se aplica la vecindad N1, que consiste en intercambiar pares consecutivos de operaciones dentro del bloque.

El resultado de esta permutación produce una nueva solución muy cercana a la solución inicial, la cual se muestra en la Fig. 6. Este intercambio es el mas sencillo de todas las estructuras de vecindad, aun asi, este problema tiene un total de $(N!)^m$ en este caso $(6!)^6$ de soluciones distintas.

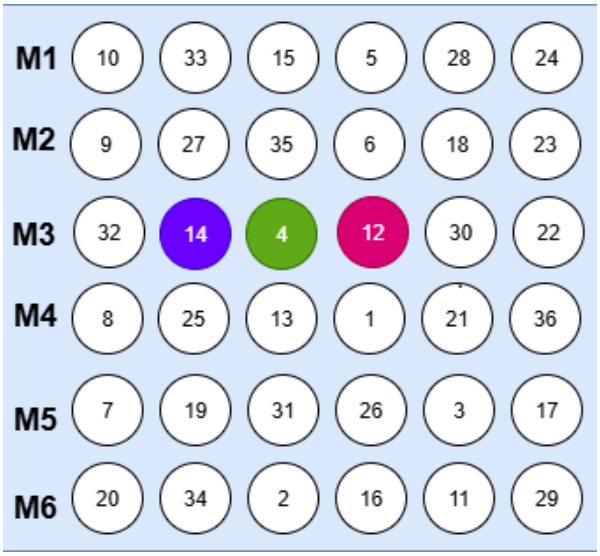


Fig. 5 Bloque de operaciones

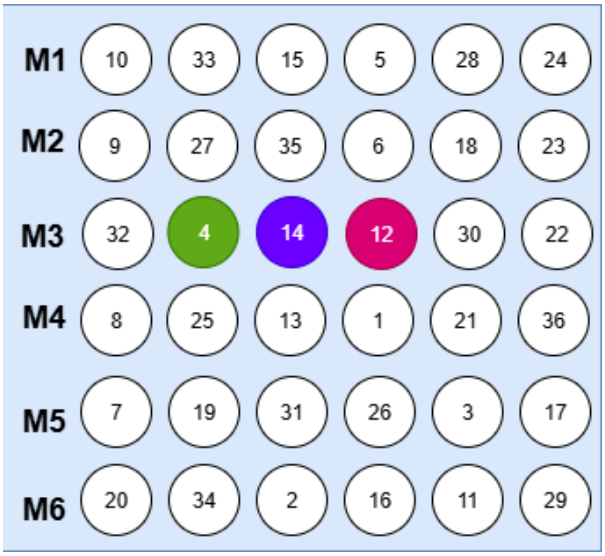


Fig. 6 Estructura de vecindad N1

3 METODOLOGÍA

3.1 Introducción

La metodología propuesta para abordar el problema de calendarización de talleres de manufactura se fundamenta en la aplicación de la meta heurística de recocido simulado **Kirkpatrick (1983)**. Esta técnica permite la exploración controlada del espacio de soluciones mediante un mecanismo que acepta tanto mejoras como ciertos empeoramientos temporales, lo cual resulta útil para evitar la convergencia prematura hacia óptimos locales. El enfoque incorpora el uso del método de la ruta crítica (Critical Path Method) para identificar las operaciones que contienen la duración total de la calendarización. A partir de ello, se limita la vecindad a permutaciones entre pares adyacentes de operaciones (N1) exclusivamente dentro de las rutas críticas de cada máquina. Esta estrategia dirigida reduce el espacio de búsqueda, concentrando los esfuerzos computacionales en configuraciones con mayor potencial de mejora. Las instancias empleadas para la experimentación corresponden a la biblioteca especializada **Or-Library (Beasley, 1990),** ampliamente utilizada en la literatura del área.

El procedimiento inicia con la carga de los datos, los cuales describen los trabajos con operaciones y máquinas involucradas. A continuación, se genera una lista inicial aleatoria de operaciones. Esta lista se ajusta conforme a las restricciones de precedencia, asegurando que las operaciones de cada trabajo conserven su orden secuencial. A partir de la lista corregida, se construye una solución inicial factible, la cual se guarda como mejor solución. Posteriormente, se procede a inicializar los parámetros del algoritmo, incluyendo la temperatura inicial, En esta implementación, el criterio de paro no se establece cuando la temperatura es menor que 0.00001, ya que el proceso de recalentamiento provoca que la temperatura no llegue a alcanzar dicho valor. Por esta razón, la ejecución del algoritmo finaliza cuando se cumple alguna de las siguientes condiciones: se alcanza el tiempo máximo de 3600 segundos, se encuentra la solución óptima. La longitud de la cadena de Markov (que determina la cantidad de soluciones generadas por nivel de temperatura), y la tasa de enfriamiento, que regula la disminución gradual de la temperatura.

Una vez completada la solución inicial, se ejecuta el proceso iterativo de recocido simulado, donde se construye una lista de máquinas previas para identificar predecesores inmediatos y aplicar el método del CPM. Se seleccionan y permutan

operaciones pertenecientes a las rutas críticas, dentro de cada máquina, a fin de generar nuevas soluciones con potencial de mejora. Luego, se verifica nuevamente la precedencia de las operaciones y por último se recalcula el makespan. La aceptación de la nueva solución depende de la variación del makespan (ΔE) y de una función probabilística basada en la temperatura actual. Si la solución es aceptada, se actualiza el estado actual y, en caso de mejora, se registra como la mejor solución hasta el momento. En caso contrario, se restaura el estado anterior. Si el algoritmo acumula 1000 soluciones rechazadas consecutivas, se incrementa temporalmente la temperatura con el fin de escapar de posibles óptimos locales. El proceso concluye cuando se alcanza el criterio de paro predefinido 3600 segundos o se alcanzó el óptimo, momento en el cual se registra y guarda la mejor solución obtenida y el programa termina.

La Figura 7 muestra el diagrama de flujo del recocido simulado, que representa de forma general el funcionamiento de los algoritmos. En él se esquematizan las etapas clave, desde la carga de la instancia, la generación y evaluación de soluciones vecinas, hasta el control de temperatura y las condiciones de parada. Este flujo permite visualizar cómo se guía la búsqueda hacia una solución optimizada dentro del tiempo límite.

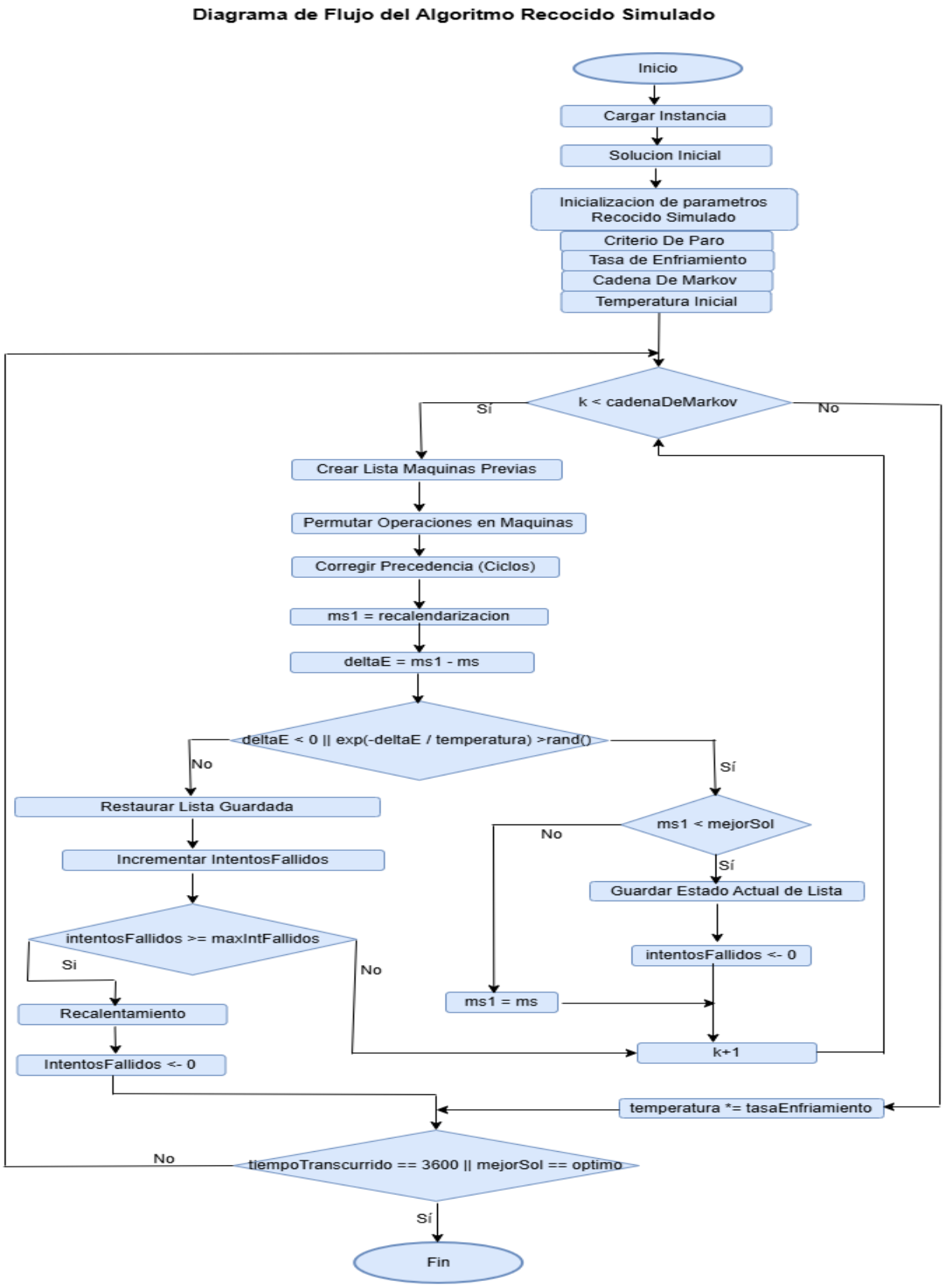


Fig. 7 Diagrama de flujo Recocido Simulado

3.2 Procedimiento para solución inicial

3.2.1 Instancia Ft06 con datos originales

En la Fig. 8 se muestra la instancia original Ft06 de Fisher y Thompson, esta es un estándar ampliamente utilizado en la literatura para evaluar algoritmos de programación de tareas. Esta instancia proporciona un conjunto concreto de trabajos y operaciones, permitiendo evaluar la eficacia de los algoritmos de calendarización bajo condiciones controladas y compararlas con resultados previos obtenidos en estudios de referencia. Al emplear esta instancia, podemos asegurarnos de que las pruebas sean consistentes y que los resultados obtenidos sean comparables con los de otros enfoques en el ámbito del JSSP.

Jn	*m, p*	*m, p*	*m, p*	*m, p*	*m, p*	*m, p*
J1	2, 1	0, 3	1, 6	3, 7	5, 3	4, 6
J2	1, 8	2, 5	4, 10	5, 10	0, 10	3, 4
J3	2, 5	3, 4	5, 8	0, 9	1, 1	4, 7
J4	1, 5	0, 5	2, 5	3, 3	4, 8	5, 9
J5	2, 9	1, 3	4, 5	5, 4	0, 3	3, 1
J6	1, 3	3, 3	5, 9	0, 10	4, 4	2, 1

Fig. 8 Instancia original Ft06

En el contexto de la calendarización de la instancia previamente mencionada, la letra *m* representa el identificador de la máquina en la cual se ejecuta una operación, mientras que *p* corresponde al tiempo de procesamiento requerido para completar dicha operación en esa máquina específica. En la implementación desarrollada, se realizó un ajuste en la numeración de las máquinas, incrementando en una unidad el valor original asignado a cada máquina (véase Fig. 9). Esta modificación se llevó a cabo con el propósito de evitar complicaciones asociadas al uso de una máquina con índice cero (máquina 0), la cual podría generar ambigüedades en el manejo de índices dentro del programa.

De este modo, la numeración de las máquinas inicia desde 1, lo cual resulta más intuitivo tanto para la asignación de operaciones como para el análisis de los resultados generados. A manera de ejemplo, la operación 1 del trabajo 1 (*J1*) presenta los valores $m = 3$ y $p = 1$, lo que indica que dicha operación debe ser ejecutada en la máquina 3 y requiere 1 unidad de tiempo de procesamiento.

Jn	*m, p*	*m, p*	*m, p*	*m, p*	*m, p*	*m, p*
J1	3, 1	1, 3	2, 6	4, 7	6, 3	5, 6
J2	2, 8	3, 5	5, 10	6, 10	1, 10	4, 4
J3	3, 5	4, 4	6, 8	1, 9	2, 1	5, 7
J4	2, 5	1, 5	3, 5	4, 3	5, 8	6, 9
J5	3, 9	2, 3	5, 5	6, 4	1, 3	4, 1
J6	2, 3	4, 3	6, 9	1, 10	5, 4	3, 1

Fig. 9 Trabajos con sus operaciones en colores y aumentada la maquina en 1

La matriz de trabajos 6 × 6, representada en la Fig. 10 organiza las operaciones de forma clara y estructurada. Cada fila corresponde a un trabajo específico (*J1, J2, ..., J6*), mientras que las seis columnas representan las operaciones dentro de ese trabajo, sumando un total de 36 operaciones distribuidas equitativamente. Esta disposición facilita el ordenamiento y la programación de las tareas según las restricciones de precedencia del JSSP.

Cada operación cuenta con un identificador único que permite rastrearla y gestionarla individualmente. Para saber a qué trabajo pertenece una operación y en qué lugar de su secuencia se encuentra, se puede usar una fórmula simple. En este caso, se toma el número de la operación de la Fig. 10, que es 27(*op*), y con base en el número de máquinas *m = 6*, se calcula la fila con la formula *(op - 1) / m + 1* implementada es (27−1) / 6 + 1 = 5, y la columna como *(op - 1) % m + 1* es (27−1) % 6 + 1 = 3. Esto significa que la operación 27 está en la fila 5 y columna 3 de la matriz de trabajos.

Dichos identificadores aseguran que se respete el orden de cada trabajo y simplifican el procesamiento en el algoritmo de programación.

La instancia se presenta como una tabla de *n* filas por *m* columnas, donde cada fila está diferenciada por un color distinto. Las filas están etiquetadas como *J1* a *J6* en el eje vertical, y las columnas como *Op1* a *Op6* en el eje horizontal. Cada celda contiene un número del 1 al 36, colocados de izquierda a derecha y de arriba hacia abajo, facilitando la identificación visual de las operaciones dentro de cada trabajo. Cada fila o trabajo tiene un color diferente y sus *n* operaciones son del mismo color:

J1: Amarillo

J2: Azul

J3: Verde

J4: Naranja

J5: Morado

J6: Rojo

Fig. 10 Matriz de trabajos de 6X6 con identificador único (index) de cada operación

Después de la explicación visual presentada en la figura anterior, se da inicio al desarrollo detallado. El seudocódigo 1 comienza con la etapa fundamental de carga de datos desde un archivo de entrada hacia las estructuras de datos definidas en lenguaje C.

```
typedef struct {    // Declaración de la estructura para representar los trabajos
    int m;          // Máquina
    int p;          // Tiempo de procesamiento
    int s;          // Tiempo de inicio
    int c;          // Tiempo de finalización
    int op;         // Número de operación
    int ls;         // Tiempo de inicio más tardío
    int lf;         // Tiempo de finalización más tardío
    int cpm;        // Pertenece a ruta crítica
    int h;          // Holgura
} Trabajo;          // Nombre de la estructura

typedef struct{     // Definición de la estructura para representar las operaciones
        int j;      // Operación
        int p;      // Tiempo de procesamiento
        int s;      // Tiempo de inicio
        int c;      // Tiempo de finalización
        int op;     // Operación a la que pertenece la máquina
        int cpm;    // Pertenece a ruta critica
}Maquina;           // Nombre de la estructura
```

Esta inicialización es esencial para preparar toda la información que será utilizada en las siguientes fases del procedimiento de optimización.

3.2.2 Carga de datos de instancia a estructuras

El seudocódigo 1 lectura e inicialización de datos, en la línea (1), se abre el archivo *file.txt* en modo lectura. En la línea (2), si el archivo no se encuentra, entonces en la línea (3) se imprime el mensaje “No se encontró el archivo”, y en la línea (4) se termina la función, si no se encuentra el archivo. En la línea (5), se leen los valores *n* y *m* desde el archivo. Luego, en la línea (6), se inicializan las estructuras *maq* y *trab* con ceros. A partir de la línea (7), se recorre cada fila *i* desde 1 hasta *n*, y en la línea (8), para cada columna *j* desde 1 hasta *m*, se ejecuta la línea (9) donde se leen los valores de máquina *trab[i][j].m* y el tiempo de procesamiento *trab[i][j].p.* En la línea (10), se inicializa la estructura *maq* con base en los datos de *trab*, y finalmente, en la línea (11), se cierra el archivo.

Algorithm 1: `cargarInstancia()` - Cargar datos desde archivo

Input: Ninguno (lee de archivo)
Output: Matrices `trab[n][m]` y `maq[n][m]` inicializadas

```
1  Abrir archivo file.txt en modo lectura;
2  if archivo no encontrado then
3      Imprimir "No se encontró el archivo";
4      return;
5  Leer n y m desde el archivo;
6  initStructCeros(n, m, maq, trab);
7  for i ← 1 to n do
8      for j ← 1 to m do
9          Leer trab[i][j].m y trab[i][j].p;
10 initMaqTrab(n, m, maq, trab);
11 Cerrar el archivo;
```

Seudocódigo 1 Carga de datos de archivo original

3.2.3 Lista aleatoria no factible

El seudocódigo 2, denominado Generación de Lista Aleatoria No Factible, es uno de los más sencillos. Consiste en llenar de forma ordenada un arreglo unidimensional con los enteros del 1 al $n \cdot m$. Posteriormente, se realizan aproximadamente $n \cdot m$ permutaciones entre estos elementos para alterar su orden de manera aleatoria. El resultado es una lista desordenada que, aunque inicialmente no cumple con las restricciones de precedencia, servirá como base para el proceso de calendarización en etapas posteriores.

Algorithm 2: `generarListaCalAle(n, m, listaCal)` - Lista aleatoria

Input: Enteros n, m; arreglo `listaCal[1..n*m]`
Output: `listaCal` permutada aleatoriamente

```
1 for i ← 1 to n · m do
2 |   listaCal[i] ← i;
3 Inicializar semilla del generador aleatorio;
4 for i ← n · m 2 do
5 |   j ← número aleatorio entre 1 y i;
6 |   Intercambiar listaCal[i] ↔ listaCal[j];
```

Seudocódigo 2 Generación de lista aleatoria no factible

Las operaciones de la matriz 6X6 de la Fig. 11 forman una lista de 36 elementos para el desarrollo de la metodología. Esta lista es generada aleatoriamente y es el resultado del algoritmo 2, aun no cumple con las restricciones de precedencia del modelo matemático. Se puede observar que las operaciones en azul 8,9,7,10,12,11 no cumplen con las restricciones de precedencia aún, después del ordenamiento y desplazamiento deberán de quedar 7,8,9,10,11,12 distribuidas por toda la lista, cumpliendo así con las restricciones de precedencia.

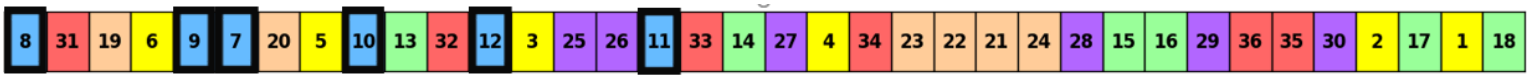


Fig. 11 Lista aleatoria, no factible

3.2.4 Ordenamiento y corrección de precedencia en una lista desordenada

El objetivo del seudocódigo 3 es reestructurar la lista de operaciones almacenadas en un arreglo unidimensional, garantizando que los elementos correspondientes a cada fila de una matriz *n* × *m* queden organizados de manera correcta. Esto se logra sin modificar el orden relativo entre filas y celdas, permitiendo que la secuencia respete las restricciones de precedencia establecidas en la programación inicial. Para ello, se emplea un método sencillo de ordenamiento basado en intercambios sucesivos (burbuja), que permite reubicar los elementos dentro de cada fila sin afectar la posición, la formula $fila = \frac{lista[i]-1}{m} + 1$ nos indica en qué fila de la matriz se encuentra cada operación.

La Tabla 2 presenta los cálculos de ciertas operaciones de la lista anterior mostrada en la Fig. 11, utilizando la fórmula previamente mencionada. Por ejemplo, la operación en el índice 1 de la Fig. 11 corresponde a la operación 8. Al aplicar la fórmula, se determina que la operación 8 se encuentra en la segunda fila, lo que indica que la operación pertenece al *J2*. En la Tabla 2, la primera columna enumera los índices de cada operación; la columna Operación lista diez operaciones que serán ordenadas; la columna Cálculo de fila muestra el resultado de aplicar la fórmula matemática; y la columna fila resultante indica la fila correspondiente a cada operación en la matriz de trabajos.

Índice	Operación	Cálculo de fila	Fila resultante
1	8	(8 -1) / 6 + 1	2
2	31	(31 -1) / 6 + 1	6
3	19	(19 -1) / 6 + 1	4
4	6	(6 -1) / 6 + 1	1
5	9	(9 -1) / 6 + 1	2
6	7	(7 -1) / 6 + 1	2
7	20	(20 -1) / 6 + 1	4
8	5	(5 -1) / 6 + 1	1
9	10	(10-1) / 6 + 1	2

Tabla 2 Cálculo de fila de cada operación

Paso 1: Encontrar los valores(operaciones) agrupados por filas (*Jn*) de la Tabla 2. Esto representa a una matriz desordenada con únicamente los 9 valores analizados, si se realizaran los 36 cálculos de listaCal, tendríamos una matriz completa no ordenada.

Fila 1: [6, 5]
Fila 2: [8, 9, 7, 10]
Fila 4: [19, 20]
Fila 6: [31]

Paso 2: Ordenar dentro de cada fila

Fila 1: [6, 5] se intercambia [5, 6]

Fila 2: [8, 9, 7,10]

8 < 9, no cambia.
8 > 7, intercambiamos → [7, 9, 8,10]
9 > 8, intercambiamos → [7, 8, 9,10]

Fila 4: [19, 20] No necesita cambios.

Fila 6: [31] No necesita cambios.

Los pasos anteriores pueden ser representados por el siguiente algoritmo o seudocódigo 3:

Algorithm 3: `corregirPrecedencia(n, m, listaCal)` - Corregir orden de operaciones

Input: n, m, arreglo `listaCal[1..n*m]`
Output: `listaCal` reordenada por precedencia en filas

```
1 for i ← 1 to n · m do
2     filaActual ← ⌊(listaCal[i] − 1)/m⌋ + 1;
3     for j ← i to n · m do
4         filaSiguiente ← ⌊(listaCal[j] − 1)/m⌋ + 1;
5         if filaActual = filaSiguiente y listaCal[i]
           >listaCal[j] then
6             Intercambiar listaCal[i] ↔ listaCal[j];
```

Seudocódigo 3 Corrección de precedencia.

Resultado del algoritmo 3, lista de operaciones del trabajo 2 en azul con marco negro con restricciones de precedencia correcta.

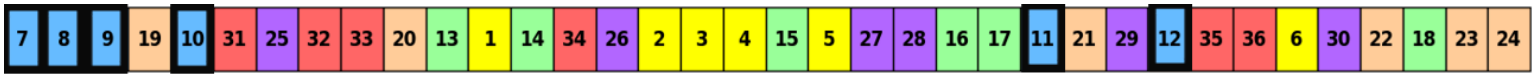


Fig. 12 Lista de operaciones con precedencia correcta

3.3 Proceso de calendarización inicial representado en una gráfica de Gantt

El método de la lista utilizado para la calendarización y el ordenamiento de las operaciones se encuentra detallado en la referencia **Díez (2022)** en su investigación de Recocido Simulado y Hill Climbing con Doble Vecindad Para el Problema de Calendarización en Talleres de Manufactura. Este enfoque es muy sencillo de implementar y resulta altamente práctico, ya que permite optimizar el proceso de búsqueda de soluciones mediante una lista iterativa eficiente. Su simplicidad lo hace adecuado para aplicaciones como la calendarización de trabajos, donde las operaciones deben ser reordenadas o programadas de acuerdo con restricciones específicas, sin requerir una compleja estructura algorítmica.

Este proceso implica asignar cada operación a una máquina disponible en el momento adecuado, respetando las restricciones de precedencia entre las operaciones dentro de cada trabajo. Se buscará una máquina libre en la fila correspondiente, y se actualizarán los tiempos de inicio y finalización de la operación en la matriz de máquinas (Gráfica de Gantt). Este procedimiento continuará hasta que todas las operaciones estén calendarizadas, lo que nos permitirá calcular el makespan final, es decir, el tiempo de finalización de la última operación.

Una vez ordenada la lista para calendarizar la cual se nombra listaCal se puede seguir con la calendarización.

Para llevar a cabo la calendarización, necesitamos dos instancias de las estructuras Trabajo y Maquina en lenguaje c. Las llamaremos trab y maq con dimensiones de *n x m* por ejemplo: Ft06(6x6) corresponde a la estructura *trab[f][c].op* , donde *f* representa la fila , *c* la columna y *op* que contiene la operación en el trabajo al que pertenece que no es otra cosa más que el identificador único al que se refiere la Fig. 10

Instancia original Ft06 con datos de trabajos en que máquina se calendarizan y su tiempo de procesamiento Fig. 13

	m, p	m, p	m, p	m, p	m, p	m, p
J1	3, 1	1, 3	2, 6	4, 7	6, 3	5, 6
J2	2, 8	3, 5	5, 10	6, 10	1, 10	4, 4
J3	3, 5	4, 4	6, 8	1, 9	2, 1	5, 7
J4	2, 5	1, 5	3, 5	4, 3	5, 8	6, 9
J5	3, 9	2, 3	5, 5	6, 4	1, 3	4, 1
J6	2, 3	4, 3	6, 9	1, 10	5, 4	3, 1

Fig. 13 Instancia original Ft06 representada como estructura trab[f][c].op

El intercambio de información entre ambos elementos trab y listaCal permite organizar las operaciones respetando las restricciones de precedencia para realizar la calendarización y calcular el tiempo total de finalización (makespan).

Ya con la lista ordenada para que no se traslapen los trabajos y generen ciclos se inicia la calendarización tomando un elemento a la vez de listaCal.

7	8	9	19	10	31	25	32	33	20	13	1	14	34	26	2	3	4	15	5	27	28	16	17	11	21	29	12	35	36	6	30	22	18	23	24

Para calendarizar la primer operación  de la lista anterior, encontramos su fila y columna en la instancia original Ft06 de trabajos. Una vez que se haya calculado la coordenada de cada operación en la lista, es decir, la fila *f* como *f = (op - 1) / m + 1* y la columna *c* como *c = (op - 1) % m* + 1, se procederá a calendarizar las operaciones

correspondientes en la matriz de máquinas, la cual será representada como una gráfica de Gantt. Una vez determinadas las coordenadas *f = 2 y c = 1*, se convierten en las coordenadas de la matriz *trab[f][c].op.*

Cabe mencionar, que el identificador de estas coordenadas corresponde a la operación 7, la cual es la primera operación del trabajo J2. Los valores 2,8 son la máquina donde se va a llevar a cabo la operación y su tiempo se procesamiento. Consulte la Fig.14, donde la operación 7 está agendada en la máquina 2 como la primera operación dentro del rango 7...12.

El tiempo de inicio de la operación 7 es 0 y su tiempo de finalización es 8. Se debe de tomar en cuenta otro factor que es el tiempo inicio más tardío de las dos operaciones anteriores y consiste en tomar el mayor valor de sus dos predecesores. El primer predecesor, es el predecesor de la lista en este caso la operación 7 su predecesor es 0 ya que es el primero en la lista. El segundo predecesor se encuentra en la estructura de máquinas. Se deben de comparar el tiempo de finalización de ambas operaciones y tomar el tiempo de finalización mayor. En este caso la operación 7 inicia en 0 y termina en 8, siendo 0 su mayor tiempo de inicio, tal y como lo ilustra la Fig. 1

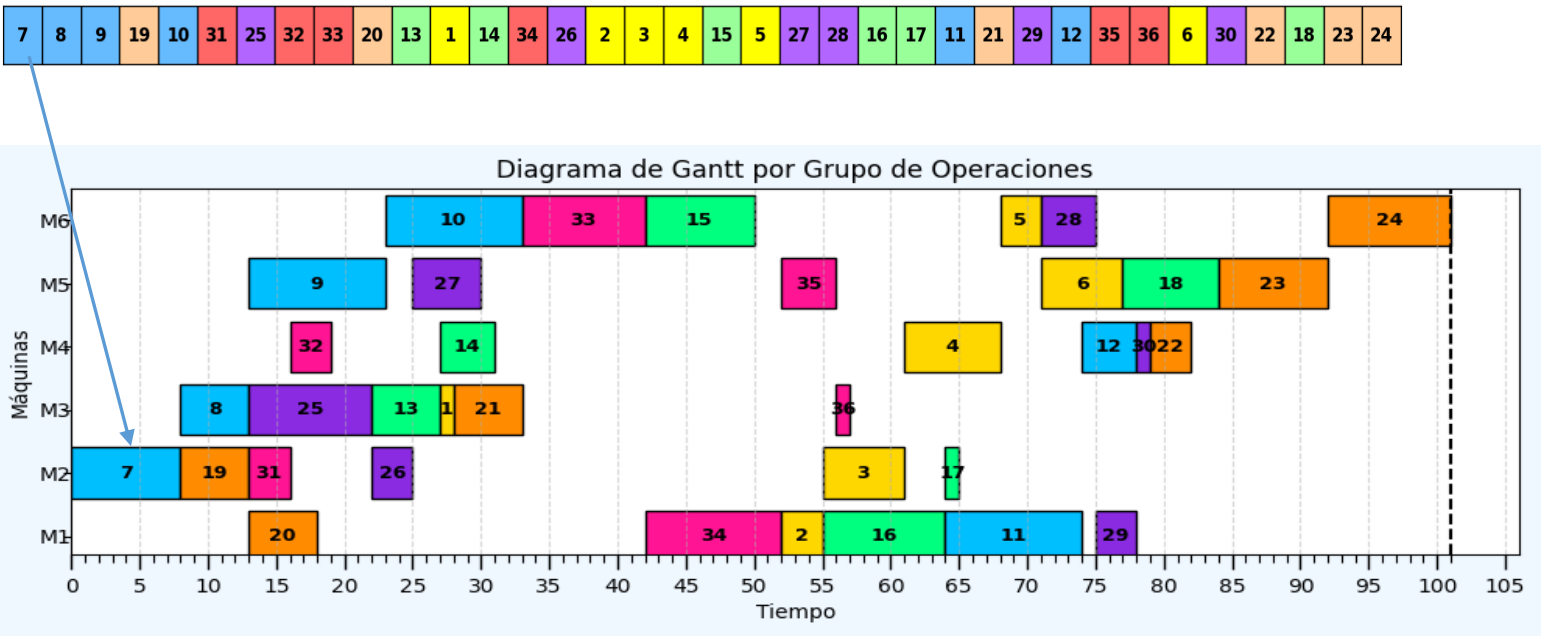


Fig. 14 Calendarización de la operación 7

El proceso se repite con cada elemento de listaCal, la fig. 15 muestra una operación más, la 8 calendarizada en máquina 3 y tiempo de procesamiento 5, datos tomados de la instancia original Ft06 Fig.13

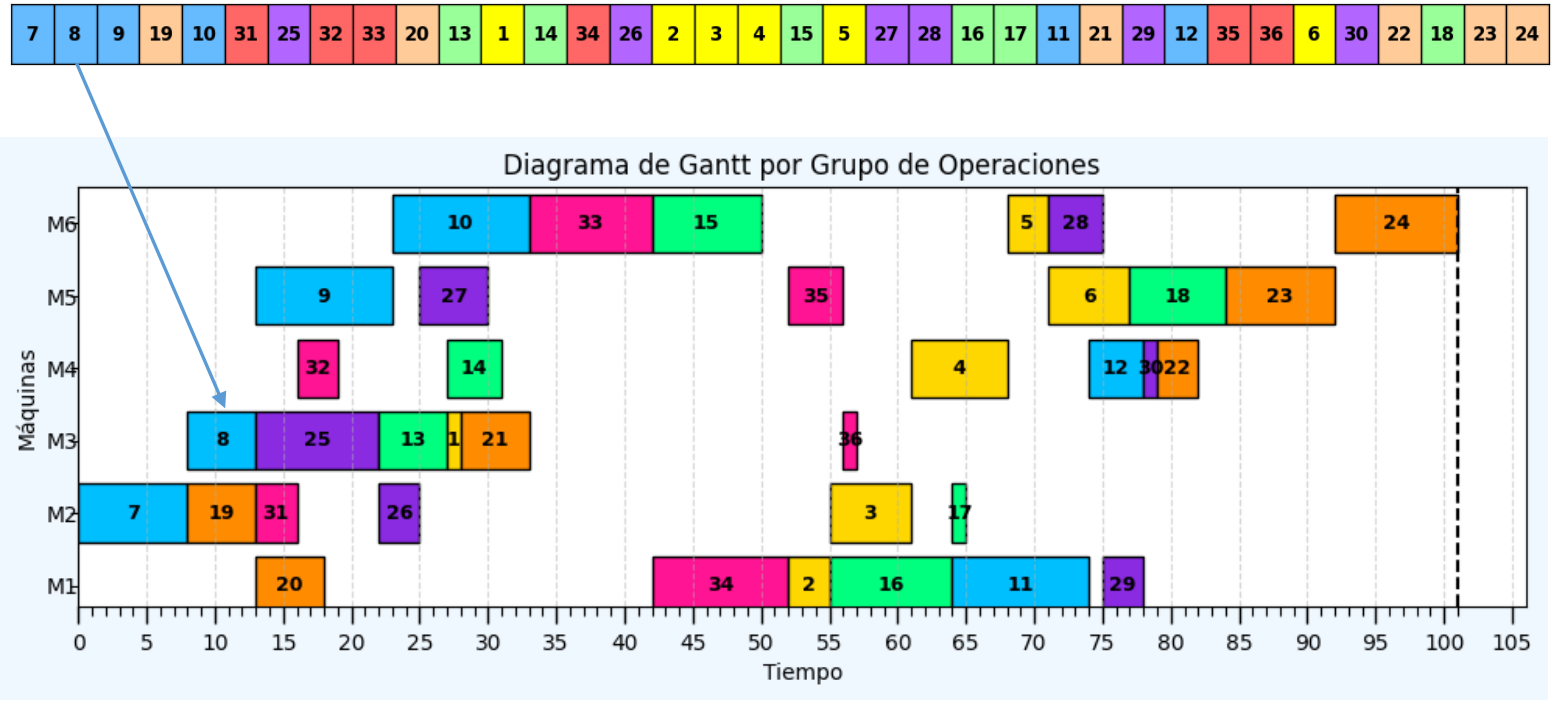


Fig. 15 Calendarización de la operación 8

3.4 Recalendarización

El algoritmo 4: recalTotal tiene como propósito agendar o re-agendar todas las operaciones en listaCal, la cual contiene los índices para obtener una calendarización. Exactamente el mismo proceso que la Fig. 14 y 15 anteriores y se lleva a cabo cuántas veces sea necesario y mientras dure el proceso de calendarización.

Explicación línea por línea del algoritmo 4: línea (1) se comienza reiniciando las estructuras maq y trab a ceros mediante la función reset() para evitar números rezagados de ejecuciones anteriores. Línea (2) para cada *i* desde 1 hasta *n·m*, se realiza el siguiente procedimiento: línea (3) se obtiene la operación *op* de la posición *i* en *listaCal* y línea (4) se calculan la fila *f*, la columna c, el número de máquina (*nm)* y el tiempo de finalización anterior en la fila *c1*. Línea (5) para cada *k* desde 1 hasta *n*, se verifica si línea (6) la posición maq[*nm][k*] se encuentra libre (*f = 0*), y en tal caso línea (7) calcula el tiempo de finalización anterior en la máquina *c2* y se determina el máximo *my* entre *c1* y *c2*. Línea (8) se asignan los tiempos de inicio y finalización en *trab[f][c]* y *maq[nm][k].* Línea (9) en caso de que el tiempo de finalización *maq[nm][k].c* sea mayor que *ms*. Línea (10) se actualiza el valor de *ms* con el nuevo máximo encontrado. Línea (11) se sale del ciclo, ya que la operación ha sido correctamente agendada, y finalmente línea (12) se devuelve el valor final de *ms* como el makespan total del conjunto de operaciones.

Algorithm 4: `recalTotal(n, m, maq, trab)` - Recalendarizar

Input: n, m, estructuras `maq[n][m]`, `trab[n][m]`
Output: Entero `ms` (makespan total)

```
reset(n, m, maq, trab);
for i ← 1 to n · m do
    op ← listaCal[i];
    Calcular f, c, nm, c1;
    for k ← 1 to n do
        if maq[nm][k].f = 0 then
            Calcular c2, my ← máx(c1, c2);
            Asignar trab[f][c].s, c, maq[nm][k] con tiempos;
            if maq[nm][k].c >ms then
                ms ← maq[nm][k].c;
            break;
return ms;
```

Seudocódigo 4 Recalendarización total

La Fig. 16 representa una calendarización inicial factible, donde las operaciones pertenecientes al mismo trabajo están diferenciadas por color. Esta visualización permite identificar claramente la secuencia de cada trabajo y verificar el cumplimiento de las restricciones de precedencia. Además, la estructura está lista para ser optimizada, facilitando la mejora del makespan mediante técnicas de optimización como la Estructura de Vecindad de Tiempo Variable en Función de su Línea de Tiempo (EVVFLT), la cual se explica a detalle en los siguientes segmentos.

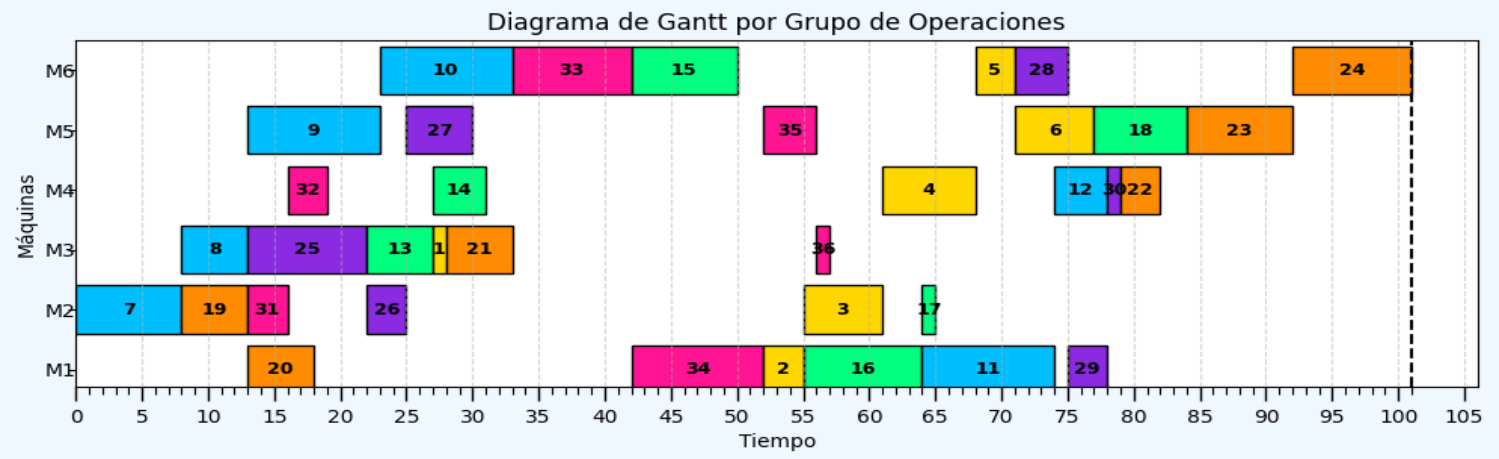


Fig. 16 Solución inicial de una matriz de máquinas 6x6; makespan 101

3.5 CPM (Critical Path Method)

El Método de la Ruta Crítica (CPM, por sus siglas en inglés: Critical Path Method) es una técnica ampliamente utilizada en la programación de proyectos y en la planificación de operaciones industriales mostrado en **Project Management Institute (2017)**. Su propósito es identificar las actividades críticas, es decir, aquellas cuya duración impacta directamente en el tiempo total de ejecución del proyecto o de la calendarización.

En el contexto de una matriz de calendarización, ya sea organizada por trabajos o por máquinas, el cálculo del CPM se realiza mediante dos recorridos secuenciales. El primero, en sentido directo (de izquierda a derecha), acumula los tiempos más tempranos posibles de inicio y finalización de cada operación. El segundo recorrido, en sentido inverso (de derecha a izquierda), determina los tiempos más tardíos permisibles sin afectar el makespan. Aquellas operaciones cuya holgura es igual a cero conforman la ruta crítica.

Este procedimiento permite detectar una o varias rutas críticas simultáneas, las cuales representan la secuencia de operaciones con mayor impacto sobre el tiempo total del sistema.

Barridos del CPM:

Este procedimiento se descompone en dos fases principales:

a) Primer barrido del CPM (de izquierda a derecha)

Este paso coincide con el procedimiento de calendarización previamente ejecutado. En él se determina el tiempo más temprano de inicio (ES) y finalización (EF) de cada operación. Específicamente, el valor de inicio (s) registrado durante la calendarización es equivalente al tiempo más temprano de inicio menos uno únicamente valido en implementación de esta tesis (ES = s – 1). Por ello, este barrido no se lleva a cabo, únicamente se le resta 1 al tiempo de inicio de cada operación.

b) Segundo barrido del CPM (de derecha a izquierda)

Esta fase se basa en un recorrido inverso de la lista de operaciones, considerando sus predecesores y sucesores. El proceso comienza generando una lista de predecesores que contiene pares de operaciones donde cada par está compuesto por una operación y su sucesora directa.

A continuación, se identifican las coordenadas de cada operación y su sucesor dentro de la matriz de trabajos. Si una operación no tiene sucesor, se asigna su tiempo de finalización más tardío (LF) con base en el makespan total de la calendarización, o bien,

tomando como referencia el inicio más temprano de la siguiente operación en la misma fila, si esta existe. Posteriormente, se calcula su tiempo de inicio más tardío (LS) como la diferencia entre LF y el tiempo de procesamiento de la operación.

Cuando una operación sí cuenta con un sucesor, se determinan sus coordenadas, y se comparan los tiempos de inicio más tardío (LS) tanto de dicha operación sucesora como de la siguiente en la misma fila. El menor valor se asigna como el LF de la operación actual, y su LS se obtiene restando la duración de la operación.

Con estos valores, se calcula la holgura de cada operación como la diferencia entre el tiempo más tardío y el más temprano, ya sea en inicio o finalización:

- Holgura = LF − EF
- Holgura = LS − ES

Una holgura de cero indica que la operación es crítica, es decir, que cualquier retraso en su ejecución afectaría el makespan del lote de producción. Por tanto, dichas operaciones son marcadas como parte de la ruta crítica tanto en la matriz de trabajos como en la de máquinas, lo que garantiza una representación precisa en herramientas visuales como el diagrama de Gantt.

3.6 Crear Lista de Máquinas Previas

Tiene como objetivo construir la lista de predecesores y sucesores de las operaciones en función de las máquinas que las procesan, de manera que pueda ser utilizada posteriormente en el cálculo del método de la ruta crítica (CPM). Generando así una lista de predecesores con el doble de tamaño que la lista inicial.

En primer lugar, para cada operación definida por el par de índices *(i,j),* donde *i* representa el trabajo y *j* la operación dentro de dicho trabajo, se registra la correspondencia con la operación respectiva en la estructura *maq*. Esta relación se almacena en la variable *posLista*, la cual permite recuperar los índices de la operación cuando sea necesario.

Posteriormente, para cada operación listada en *listaCal*, se construye la lista *listaMaqPrev* de la siguiente forma: en la posición impar (2k−1) se asigna el identificador de la operación actual, mientras que en la posición par (2k) se registra el sucesor de dicha operación en la máquina correspondiente, o en su defecto, el valor 0 si no existe sucesor.

Una vez recorridas todas las operaciones, se determina el tamaño total de la lista como $nLis = 2 \cdot n \cdot m + 1$, donde n es el número de trabajos y m el número de operaciones por trabajo. Finalmente, se procede a llamar a la función calcularCPM, la cual se encarga de

obtener los tiempos más tempranos y más tardíos de ejecución considerando la información previamente organizada. El seudocódigo 5 muestra el procedimiento para generar la lista de máquinas previas, necesarias para el cálculo de la ruta crítica.

```
Algorithm 5: crearListaMaqPrev(n, m, maq, trab, ms)
  Input: n, m, estructuras maq[n][m], trab[n][m], makespan ms
  Output: Lista listaMaqPrev
1 for i ← 1 to n do
2 |   for j ← 1 to m do
3 |   |   posLista[maq[i][j].j] ← (i, j);
4 for k ← 1 to n · m do
5 |   listaMaqPrev[2k - 1] ← listaCal[k];
6 |   Obtener (i, j) de posLista[listaCal[k]];
7 |   listaMaqPrev[2k] ← sucesor o 0;
8 nLis ← 2 · n · m + 1;
9 Llamar calcularCPM(n, m, trab, listaMaqPrev, nLis, ms);
```

Seudocódigo 5 Generación de lista con operaciones predecesoras

3.7 Calcular CPM

Tiene como objetivo identificar las operaciones que pertenecen a la ruta crítica dentro de la calendarización, el procedimiento es presentado en el seudocódigo 6. Para ello, el procedimiento inicia recorriendo la lista de operaciones en orden inverso, es decir, desde el penúltimo elemento hasta el primero. En cada iteración, se obtiene la operación actual a partir de la lista de predecesores y se identifica su respectivo sucesor. Posteriormente, se calculan la fila y la columna que corresponden a la operación analizada, verificando si esta posee un sucesor definido.

En caso de que el sucesor sea igual a cero, se asigna el valor de *lf* (tiempo de finalización más tardío) como el fin de la última operación o bien como el inicio de la operación siguiente. En caso contrario, se calculan la fila y la columna correspondientes al sucesor, y el valor de *lf* se establece como el mínimo entre los tiempos de inicio de las operaciones posteriores.

Con el valor de *lf* determinado, se procede a calcular el tiempo de inicio más tardío (*ls*) mediante la relación *ls = lf – p*, donde *p* corresponde a la duración de la operación. A continuación, se obtiene la holgura como *h = lf – (s – 1 + p).* Con base en este cálculo, se clasifica la operación como crítica si la holgura resulta igual a cero, asignando el indicador *cpm = 1*Finalmente, en caso de que la operación cumpla con esta condición,

se marca explícitamente como operación crítica dentro de la estructura de datos maq[nm][k].cpm. Véase la Fig. 17, donde los dos primeros valores representan el recorrido hacia la derecha, mientras que los dos números inferiores representan el recorrido de derecha a izquierda del CPM. La complejidad resultó O(n) ya que se usaron los procedimientos de calendarización como el primer recorrido hacia la derecha, el apéndice A muestra la complejidad temporal del algoritmo.

Algorithm 6: `calcularCPM(n, m, trab, predecesores, nLis, ms)`

Input: n, m, `trab[n][m]`, `predecesores[1..nLis]`, makespan `ms`
Output: Campos `ls`, `lf`, `h`, `cpm` actualizados

```
1  for i ← nLis − 2 0 paso −2 do
2      op ← predecesores[i] ;
3      suc ← predecesores[i + 1] ;
4      Calcular f, c y si tiene sucesor;
5      if suc = 0 then
6          lf ← (última o siguiente op);
7      else
8          Calcular f2, c2 del sucesor;
9          lf ← mín(ls_sig1, ls_sig2);
10     ls ← lf − p;
11     h ← lf − (s − 1 + p);
12     cpm ← 1 si h = 0;
13     Si cpm = 1, marcar en maq[nm][k];
```

Seudocódigo 6 Cálculo de la ruta crítica y tiempos de holgura

3.8 Representación Simbólica del CPM

La Fig. 17 ilustra gráficamente el proceso del CPM, en ella se observa la asignación de operaciones por máquina, donde cada operación se identifica con un nodo que contiene cuatro valores enteros. Los dos valores superiores corresponden al barrido hacia la derecha (ES y EF), y los dos inferiores al barrido hacia la izquierda (LS y LF). Las operaciones marcadas con flechas en color negro más gruesas pertenecen a la ruta crítica. Por ejemplo, la máquina 1 (M1) realiza las operaciones 20, 34, 2, 16, 11 y 29, de las cuales solo las operaciones 34 y 2 forman parte de la ruta crítica y, por tanto, son candidatas a ser permutadas en el proceso de optimización con la estructura de vecindad variable en función de su tiempo.

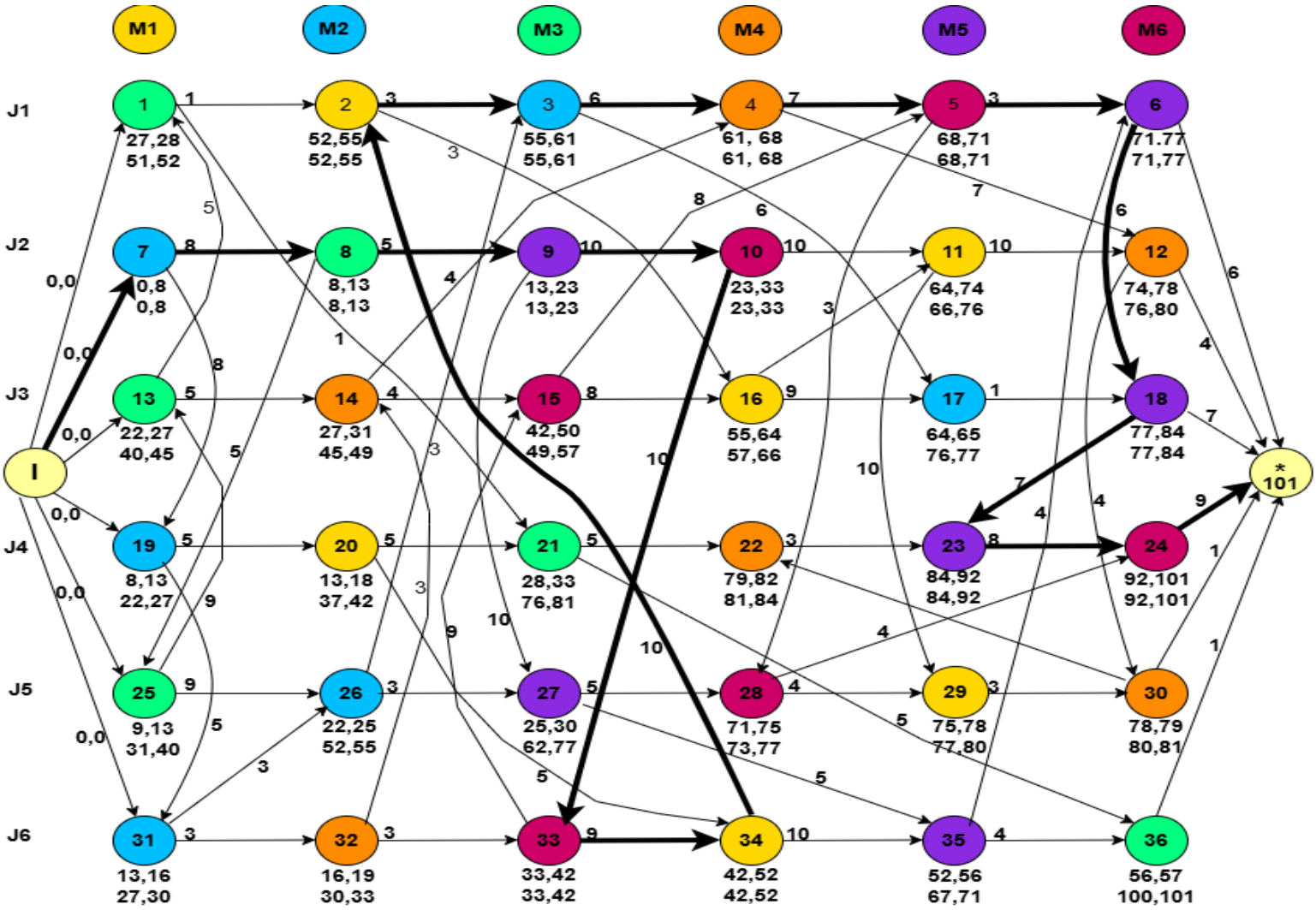


Fig. 17 Cálculo del método de la ruta crítica en matriz de máquinas

La Figura 18 es una representación gráfica, mediante un grafo de la matriz de trabajos, en la cual se destacan las actividades que forman parte de las rutas críticas. Esta visualización corresponde a la misma calendarización de operaciones mostradas previamente en la Fig. 17, permitiendo así una interpretación complementaria del análisis del método de la ruta crítica.

Los encabezados de M1…M6 que representan las máquinas únicamente están para fines ilustrativos, si se desea ver que operaciones lleva a cabo cada máquina, ver la Fig.17 donde la M1(color naranja) realiza las operaciones 20,34, 2,16,11y 29.

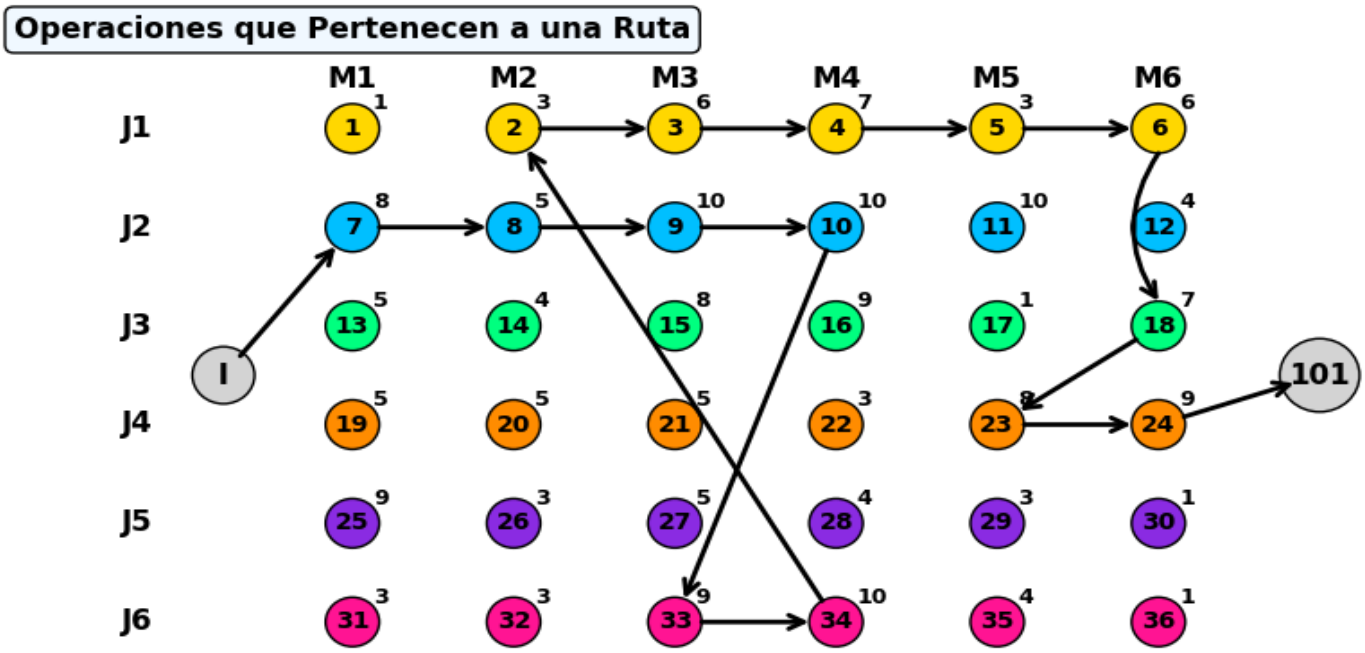


Fig. 18 Grafo dirigido de matriz de trabajos con sus operaciones pertenecientes a una ruta crítica.

Una vez determinada la ruta crítica en la Fig.17, otro paso consiste en representar la ruta critica mediante una gráfica de Gantt Fig. 19, en la cual se resaltan en color negro las operaciones que pertenecen a dicha ruta. Esta representación permite identificar de manera visual los pares de operaciones candidatos a permutar.

En primer lugar, se analizan las operaciones 10 y 33, que son adyacentes, se ejecutan en la máquina 6 y forman parte de la ruta crítica. Bajo estas condiciones, se consideran permutables.

La estructura de vecindad se define en función de la línea de tiempo, la cual se determina mediante un número aleatorio generado entre 1 y el makespan. Este valor establece un intervalo de referencia que afecta únicamente a aquellas operaciones cuyo inicio y fin en la secuencia temporal lo contienen, delimitando así el subconjunto de operaciones elegibles para permutación.

Para efectos ilustrativos, en la gráfica se marca la línea de tiempo en rojo con el valor 30. A partir de este criterio, se seleccionan todas las operaciones que pertenecen a la ruta crítica y cuyo intervalo de ejecución incluye dicho valor.

El procedimiento se aplica de la siguiente manera:

1. Se identifica el inicio de la operación 10 en la unidad de tiempo 23, lo cual marca el inicio del intervalo de tiempo. Este límite se representa mediante una línea punteada ubicada a la izquierda de la línea de tiempo (30).
2. Posteriormente, se determina la finalización de la operación 33, la cual define el fin del intervalo de tiempo. Este límite se representa mediante una segunda línea punteada roja, a la derecha de la línea de tiempo.

De esta forma, se establece un rango comprendido entre las unidades de tiempo 23 y 42, que abarca un total de 19 unidades de tiempo. En consecuencia, cualquier número aleatorio generado dentro de este rango obliga a la permutación del par de operaciones 10 y 33, puesto que ambas cumplen con la condición de pertenecer a la ruta crítica.

A continuación, se verifica el resto de operaciones adyacentes en la gráfica de Gantt:

- La operación 15, aunque adyacente a la operación 33, no pertenece a la ruta crítica y, por tanto, no es permutable.
- La operación 14, ubicada en la máquina M4, es atravesada por la línea de tiempo (30). Sin embargo, debido a que no tiene un par adyacente y no forma parte de la ruta crítica, también es descartada.
- En el caso de que las operaciones 21 y 1 si pertenecieran a otra ruta crítica, dichas operaciones serían igualmente consideradas para permutación. Aunque son adyacentes no son permutadas ya que no pertenecen a ninguna ruta crítica.
- Finalmente, la operación 9 sí pertenece a la ruta crítica y presenta como par adyacente a la operación 27. No obstante, dado que la operación 27 no forma parte de la ruta crítica y, además, existe un intervalo de inactividad entre ambas, este par no se considera en el proceso, por lo tanto tampoco es permutable.
- El tamaño de la vecindad es considerablemente reducido y resulta equivalente al número de máquinas, dado que, en el mejor de los casos, existirían $m = 6$ pares de operaciones permutables que se alinean con la línea de tiempo. En la Fig. 19 se identifican cuatro pares adyacentes permutables: (10, 33), (34, 2), (6, 18) y (18, 23); todos ellos forman parte de una ruta crítica. Cabe señalar que dichos pares permutables se encuentran en diferentes líneas de tiempo, lo que evidencia la complejidad de su impacto en la secuencia de operaciones.

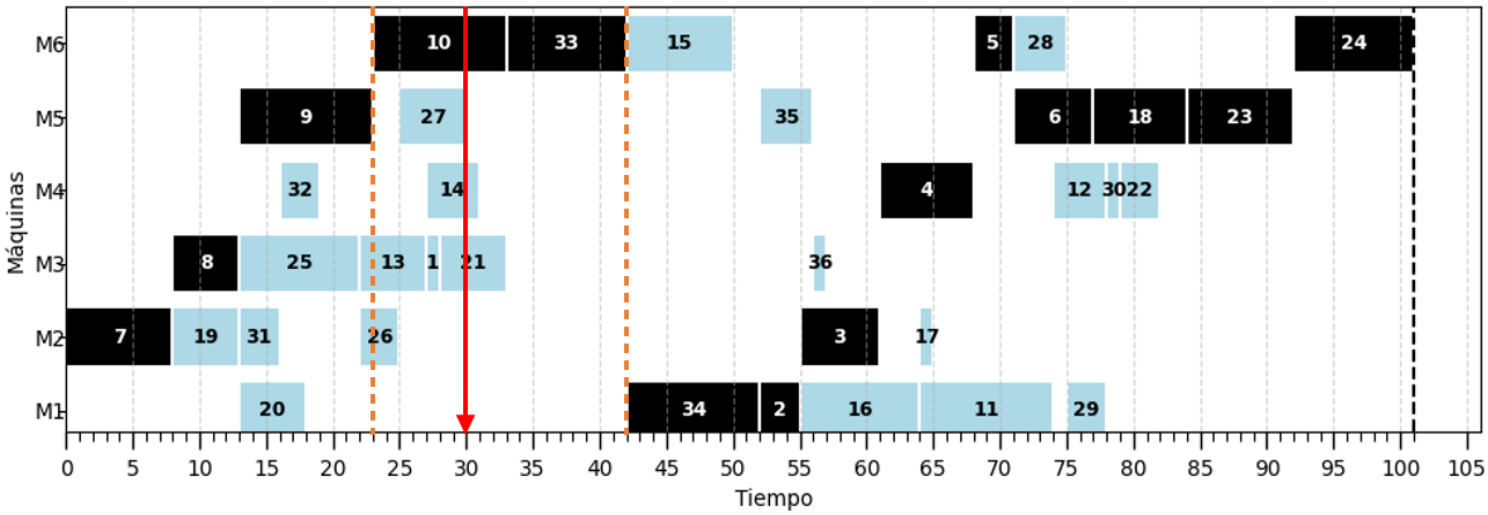


Fig. 19 Gráfica de Gantt con operaciones en negro que pertenecen a una ruta crítica equivalente a la misma ruta de las Fig. 17 y 18

3.9 Permutación de pares de operaciones en listaCal

Una vez determinada la ruta crítica (CPM) mediante el procedimiento previamente descrito, se procede a la aplicación de técnicas de búsqueda local con el fin de explorar soluciones alternativas en el tamaño de la vecindad. Una de estas técnicas consiste en la permutación de pares de operaciones dentro de una máquina y en la lista de operaciones calculada, lo cual permite evaluar cambios en el ordenamiento y su impacto en los tiempos de finalización.

El Algoritmo o seudocódigo 7 describe el procedimiento para realizar la permutación de un par de operaciones. Inicialmente, se identifican las posiciones correspondientes a las operaciones seleccionadas dentro de la lista. En caso de encontrarse ambas operaciones, se ejecuta el intercambio en la secuencia, modificando así el orden en el que serán procesadas. Si alguna de las operaciones no se localiza, el procedimiento reporta un mensaje de error, asegurando la trazabilidad del proceso.

Este mecanismo constituye una base fundamental para el desarrollo de la estrategia de optimización, ya que la variación en el orden de las operaciones puede generar diferentes configuraciones de calendarización, impactando de manera significativa el makespan y la ruta crítica.

Algorithm 7: `permutarPares(n, m, op1, op2, listaCal)`

Input: Operaciones `op1`, `op2`, `listaCal[1..n*m]`
Output: Lista `listaCal` con los valores permutados

```
1 Buscar posiciones pos1 y pos2 de op1 y op2;
2 if ambas posiciones encontradas then
3 |  Intercambiar listaCal[pos1] y listaCal[pos2];
4 else
5 |  Imprimir error: no se encontraron ambas operaciones;
```

Seudocódigo 7 Permutación de pares en listaCal

3.10 Permutación de pares adyacentes en estructura de máquinas

También es necesario realizar permutaciones en la estructura de máquinas para completar el procedimiento. El Algoritmo o seudocódigo 8: permutarOpMaq(n, m, ms, maq, trab, listaCal, mejorSol) inicia generando un valor aleatorio para la variable *lineaDeTiempo*, comprendido entre 1 y *ms*, tal como se ilustra el algoritmo 8. A continuación, se recorre cada fila *i* desde 1 hasta *n*, y dentro de este ciclo se itera sobre cada columna *j* desde 2 hasta *m − 1*. Durante este proceso, se evalúa si la operación *maq[i][j].op* está activa en el instante correspondiente a *lineaDeTiempo* y, adicionalmente, si cumple con la condición de tener el campo *cpm = 1*. En caso afirmativo, se verifica si los vecinos inmediatos también cumplen con la misma condición.

Con base en esta verificación, se aplican las permutaciones necesarias siguiendo la lógica previamente definida, la cual contempla tres posibles casos que aseguran la correcta reestructuración de las operaciones en la secuencia de máquinas. Asimismo, si existe otro par adyacente que cumpla con la condición en el mismo instante de tiempo, también se lleva a cabo la permutación. Esta capacidad de permutar todas las operaciones marcadas con *maq[i][j].cpm = 1* en cualquier máquina constituye una de las principales fortalezas del algoritmo, ya que amplía las posibilidades de exploración de soluciones dentro del espacio de búsqueda y funciona con varias rutas críticas.

Algorithm 8: `permutarOpMaq(n, m, ms, maq, trab, listaCal, mejorSol)`

Input: n, m, ms, estructuras `maq`, `trab`, arreglo `listaCal`, valor `mejorSol`
Output: Lista `listaCal` permutada si aplica

```
1 lineaDeTiempo ← valor aleatorio entre 1 y ms;
2 for i ← 1 to n do
3     for j ← 2 to m − 1 do
4         if operación maq[i][j] activa en línea de tiempo y con cpm = 1
          then
5             Verificar vecinos con cpm y holguras;
6             Realizar permutaciones según lógica definida (3 casos);
```

Seudocódigo 8 Permutación de operaciones en máquinas

3.11 Sintonización del Recocido Simulado

El recocido simulado, como se ha señalado previamente, es un método inspirado en el proceso físico de enfriamiento de metales, aplicado en la resolución de problemas complejos de optimización combinatoria, entre ellos el JSSP. Este enfoque permite recorrer el espacio de soluciones de manera probabilística, admitiendo en ciertas ocasiones alternativas de menor calidad con el fin de evitar quedar atrapado en óptimos locales. La efectividad de este procedimiento está estrechamente vinculada a la adecuada configuración de sus parámetros, la cual se expone a continuación:

1. Criterio de paro

Se evaluaron dos valores para la temperatura mínima: 0.001 y 0.0000001. Aunque ambos valores ofrecen resultados similares en calidad de soluciones, se adoptó el valor más pequeño (0.0000001) para permitir una búsqueda más prolongada en fases avanzadas, útil para instancias difíciles, sin aumentar significativamente el tiempo de ejecución. Cabe mencionar que el criterio de paro nunca alcanza el valor mencionado, ya que, al hacerse uso del recalentamiento, la temperatura sube cuando el recocido simulado después de 1000 iteraciones no mejora. Dando lugar a que el criterio de paro en realidad sean 3600 segundos o cuando el óptimo se haya alcanzado.

2. Tamaño de la cadena de Markov

El número de iteraciones del ciclo interno, conocido como cadena de Markov, se ajustó inicialmente en un rango de entre 3 y 15, lo que permitió alcanzar un equilibrio adecuado entre el tiempo de cómputo y la calidad de la solución obtenida. En este contexto, la cadena de Markov se define como la cantidad de soluciones generadas y evaluadas en

una misma temperatura antes de proceder a la etapa de enfriamiento; es decir, representa la longitud del ciclo de exploración asociado a un valor fijo de temperatura.

Valores más altos, como 30, tienden a incrementar el tiempo de ejecución, sin aportar mejoras significativas en los resultados, además, en instancias grandes, este valor no resulta adecuado, ya que provoca un consumo elevado de tiempo sin generar beneficios proporcionales en la calidad de la solución. Por este motivo, se estableció un valor de 10, dado que la lista de operaciones mantiene siempre la mejor solución encontrada y se actualiza de inmediato cada vez que se identifica una mejor. En caso de que, durante 10 intentos consecutivos, no se observe ninguna mejora, se recupera la mejor solución registrada en la lista.

3. Temperatura inicial

La temperatura inicial se calculó de manera dinámica como el producto de la solución inicial por 100,000, con el objetivo de garantizar una exploración amplia desde el inicio y prevenir estancamientos prematuros en óptimos locales. El valor de 100,000 se eligió debido a que, al considerar directamente la solución inicial como temperatura de partida, el programa concluye su ejecución en pocos segundos. De esta forma, se estableció una temperatura elevada, lo que permite realizar un mayor número de iteraciones y, en consecuencia, favorecer una exploración más exhaustiva del espacio de soluciones.

4. Tasa de enfriamiento

Tras un proceso de experimentación exhaustiva, en el que se evaluaron valores de la tasa de enfriamiento comprendidos entre 0.5 y 0.9999999, se determinó que el mejor valor era 0.99996. Esta constante garantiza un enfriamiento lo suficientemente lento como para mantener la diversificación de soluciones en instancias pequeñas, al tiempo que proporciona un equilibrio adecuado entre exploración y explotación en instancias de mayores. Es importante señalar que, cuando la tasa de enfriamiento es demasiado rápida (por ejemplo, 0.5), puede ser necesario incrementar significativamente la longitud de la cadena de Markov hasta valores de 30,000 para compensar dicha rapidez. No obstante, en este estudio se optó por utilizar una tasa de 0.99996 junto con una cadena de Markov de 10, ya que esta combinación mostró el mejor desempeño. Aunque una tasa de enfriamiento acelerada acompañada de una cadena de Markov extensa puede ofrecer resultados similares a los obtenidos, la configuración .99996 y 10 seleccionada representa un balance entre rendimiento computacional y calidad de las soluciones obtenidas, este comportamiento se observa en la Fig. 20 Cadenas de Markov.

Los siguientes puntos 5,6,7 no son parte de recocido simulado, pero se optaron con el fin de encontrar mejores soluciones:

5. Estrategia de recalentamiento

Se incorporó la estrategia de recalentamiento debido a que, en instancias de gran tamaño, después de 25 minutos de ejecución la solución deja de mejorar de manera significativa. Por ello, cuando no se obtiene una mejora tras 1,000 intentos consecutivos, la temperatura se incrementa de forma aleatoria. Este procedimiento permite elevar la temperatura hasta el doble de su valor actual, lo que ayuda al algoritmo a escapar de óptimos locales profundos y a reactivar la exploración en distintas regiones del espacio de soluciones. Cabe señalar que la efectividad de esta técnica es variable, pues en algunos casos resulta beneficiosa y en otros no produce mejoras notables.

6. Tiempo máximo de ejecución

Se estableció un tiempo límite de 3,600.01 segundos (una hora) para todas las instancias. El algoritmo puede detenerse antes si alcanza el óptimo conocido o si la temperatura desciende por debajo del umbral mínimo, asegurando condiciones controladas y comparables en las pruebas.

7. Recuperación de la mejor solución

El algoritmo conserva la mejor solución encontrada y la actualiza únicamente cuando se detecta una mejora. En caso de que, dentro de la cadena de Markov, se registren 10 ejecuciones consecutivas sin mejora, la solución previamente almacenada es recuperada. Este mecanismo resulta particularmente útil, dado que en el espacio de soluciones suele existir únicamente un par permutable por máquina. En instancias pequeñas, el recalentamiento tiende a activarse después de aproximadamente 15 minutos de ejecución; en cambio, en instancias de mayor tamaño, la búsqueda puede prolongarse, siempre que no se superen las mil iteraciones consecutivas sin mejora.

8. Resultados experimentales de la instancia YN1 con variaciones en la cadena de Markov y el enfriamiento

En la Fig. 20 se presentan los resultados obtenidos al probar la instancia YN1, considerando cadenas de Markov con valores de 300, 250, 200, 150, 100, 50 y 0 en el eje X. Para cada una de estas configuraciones, se analizaron diferentes tasas de enfriamiento (0.996, 0.9996 y 0.99996). Los resultados muestran que, en la mayoría de los casos, el makespan más bajo se alcanza cuando la cadena de Markov se encuentra en un valor de 10, lo que evidencia un comportamiento consistente del algoritmo frente a las variaciones de la longitud de la cadena y las tasas de enfriamiento.

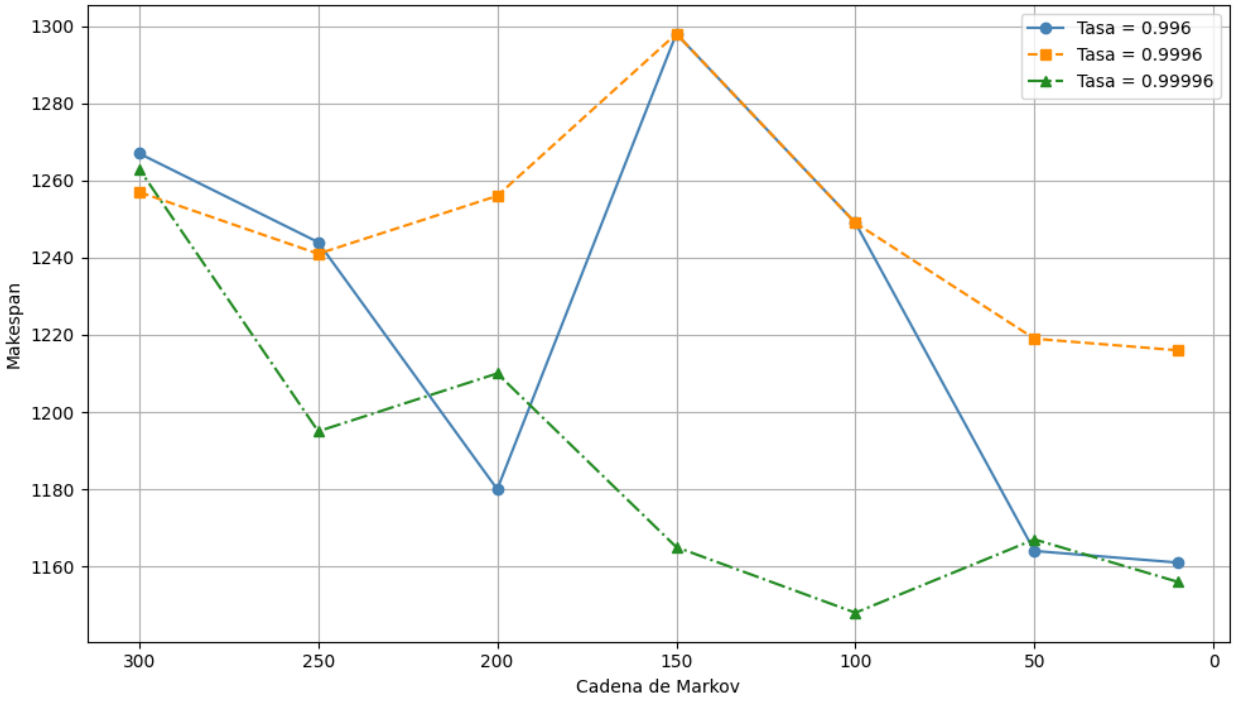


Fig. 20 Cadenas de Markov

3.12 Recocido simulado para la optimización

En el último paso antes de reiniciar el ciclo de recalendarización, la preparación se llevó a cabo en los 8 algoritmos anteriores, el seudocódigo 9 de muestra el procedimiento del recocido simulado que consiste en buscar la mejor solución al problema de calendarización. Inicialmente, se cargan los datos de la instancia desde el archivo (línea 1) y se genera una lista de operaciones completamente aleatoria (línea 2), que se corrige para mantener las restricciones de precedencia (línea 3). Se calcula el makespan inicial (línea 4), y se guarda el estado actual de la lista como respaldo (línea 5), inicializando la mejor solución conocida con este valor (línea 6) y registrando el valor de la solución inicial (línea 7). La temperatura inicial se establece como el producto de la temperatura base por el makespan inicial (línea 8), y se registra el tiempo de inicio del algoritmo (línea 9).

Mientras la temperatura sea mayor que el criterio de paro (línea 10), se ejecuta la cadena de Markov: se genera la lista de precedencias por máquina (línea 12), se realizan permutaciones sobre las operaciones de la ruta crítica (línea 13), y se corrige nuevamente la lista de operaciones (línea 14). Se calcula el nuevo makespan ms1 (línea 15) y se evalúa el cambio de energía deltaE entre la solución actual y la anterior (línea

16). Si el cambio es favorable o se acepta probabilísticamente según la temperatura (línea 17), se actualiza el makespan *ms* (línea 18).

Si la nueva solución mejora la mejor solución conocida (línea 19), se actualiza el registro de la mejor solución y se guarda el estado actual (líneas 20-22), reiniciando el contador de intentos fallidos (línea 23). En caso de que la solución no se acepte, se restaura la mejor lista de operaciones conocida (línea 25) y se incrementa el contador de intentos fallidos (línea 26). Cuando los intentos fallidos superan el límite permitido (línea 27), se recalcula la temperatura en función de la mejor solución, y un valor aleatorio (línea 28), reinicia el contador de intentos fallidos. Durante todo el proceso, se mantiene actualizado el tiempo transcurrido (línea 29), y si se excede el límite o se alcanza el óptimo deseado, se finaliza el ciclo principal (líneas 30, 31). Finalmente, la temperatura se reduce según la tasa de enfriamiento (línea 30), permitiendo que el algoritmo explore nuevas soluciones de manera progresivamente más selectiva.

Algorithm 9: sa() - Optimización con recocido simulado

Input: Parámetros: n, m, tempInicial, criterioDeParo, tasaEnfriamiento, tiempoMaximo, optimo, cadenaDeMarkov, maxIntFallidos

Output: Mejor solución encontrada mejorSol

```
1  cargarInstancia();
2  generarListaCalAle(n, m, listaCal);
3  corregirPrecedencia(n, m, listaCal);
4  ms ← recalTotal(n, m, maq, trab);
5  guardarEstadoActual(n, m, listaCal, listaCalGuardada);
6  mejorSol ← ms;
7  solInicial ← ms;
8  temperatura ← tempInicial · solInicial;
9  tiempoDeInicio ← clock();
10 while temperatura > criterioDeParo do
11     for k ← 0 to cadenaDeMarkov do
12         crearListaMaqPrev(n, m, maq, trab, ms);
13         permutarOpMaq(n, m, ms, maq, trab, listaCal,
            mejorSol);
14         corregirPrecedencia(n, m, listaCal);
15         ms1 ← recalTotal(n, m, maq, trab);
16         deltaE ← ms1 − ms;
17         if deltaE < 0 o e^(−ΔE/T) > aleatorio then
18             ms ← ms1;
19             if ms1 < mejorSol then
20                 mejorSol ← ms1;
21                 guardarEstadoActual(n, m, listaCal,
                    listaCalGuardada);
22                 intentosFallidos ← 0;
23         else
24             restaurarListaCal(n, m, listaCal,
                listaCalGuardada);
25             intentosFallidos++;
26         if intentosFallidos ≥ maxIntFallidos then
27             temperatura ← mejorSol · (rand() % 100 + 2);
28             intentosFallidos ← 0;
29         tiempoTranscurrido ← tiempo desde inicio en segundos;
30     if tiempoTranscurrido > tiempoMaximo o mejorSol ≤ optimo
        then
31         break;
32     temperatura ← temperatura · tasaEnfriamiento;
```

Seudocódigo 9 Recocido simulado

4.EXPERIMENTACIÓN Y RESULTADOS

Descripción del equipo utilizado, tabla 3, las pruebas se llevaron a cabo en el cluster de la minigrid de la Universidad Autónoma del Estado de Morelos (UAEM), el cual tiene las siguientes características:

PROCESADOR	MEMORIA / ALMACENAMIENTO	SISTEMA. OPERATIVO
CPU: 58 nodos con un total de 186 cores. **GPU: 2 nodos con un total de 1792 cores.**	**CPU: 171 GB de RAM, 34 Terabytes de HD.** **GPU: 72 GB de RAM, 2 Terabytes de HD.**	**Linux CentOS**

Tabla 3 Descripción del equipo utilizado

4.1 Parámetros del algoritmo utilizado

Para el proceso de optimización mediante recocido simulado se establecieron los siguientes parámetros: la temperatura inicial se calculó a partir de la solución inicial obtenida ejemplo: (1487) multiplicada por 100,000, dando TempInicial = 148,700,000; el criterio de paro se definió como un valor mínimo de temperatura de 0.0000001; el esquema de enfriamiento consideró un decremento progresivo con una tasa de reducción de 0.99996 por iteración; y se estableció una longitud de cadena de Markov igual a 10, lo que implica que en cada nivel de temperatura se generan 10 soluciones candidatas. Con estos valores, el número aproximado de niveles de temperatura necesarios para alcanzar el criterio de paro se estimó mediante k = (Tf / Ti) / (alpha), obteniéndose aproximadamente 863,250 niveles de temperatura, que al multiplicarse por la longitud de la cadena de Markov resultan en un total de cerca de 8,632,500 evaluaciones de soluciones a lo largo del proceso de búsqueda. El algoritmo realiza la cantidad de operaciones en aproximadamente de 41 a 43 minutos en instancias grandes. Al agregarse la técnica de recalentamiento se incrementa el tiempo de ejecución, máximo una hora.

Para la experimentación se utilizaron benchmarks reconocidos de la **Or-Library Beasley, J. E. (1990)**. Con el fin de mantener uniformidad y practicidad, se seleccionaron un total de 40 instancias, todas con número de filas y columnas de misma longitud. Esto permitió

evaluar los algoritmos manteniendo consistencia en la estructura de las matrices de operaciones y máquinas.

Para el análisis de estadística inferencial desarrollado más adelante, se utilizaron únicamente 24 instancias de tamaño pequeño o mediano (hasta la instancia la40), debido a que algunos de los autores de las heurísticas SysLS, GBA, TB y mXLSGA no realizaron pruebas con instancias de mayor tamaño, como yn1, yn2, yn3, yn4, ta21, ta22, ta23 y ta24. A pesar de esta limitación, el objetivo principal de este estudio es comprobar la existencia de diferencias significativas entre los métodos evaluados.

Tamaño de instancia	Nombre de instancias
10 × 10	abz5, abz6, ft06, ft10, la16, la17, la18, la19, la20, orb01, orb02, orb03, orb04, orb05, orb06, orb07, orb08, orb09, orb10
15 × 15	la36, la37, la38, la39, la40, ta01, ta02, ta03, ta04, ta05, ta06, ta07, ta08
20 × 20	yn1, yn2, yn3, yn4, ta21, ta22, ta23, ta24

Tabla 4 Instancias de prueba

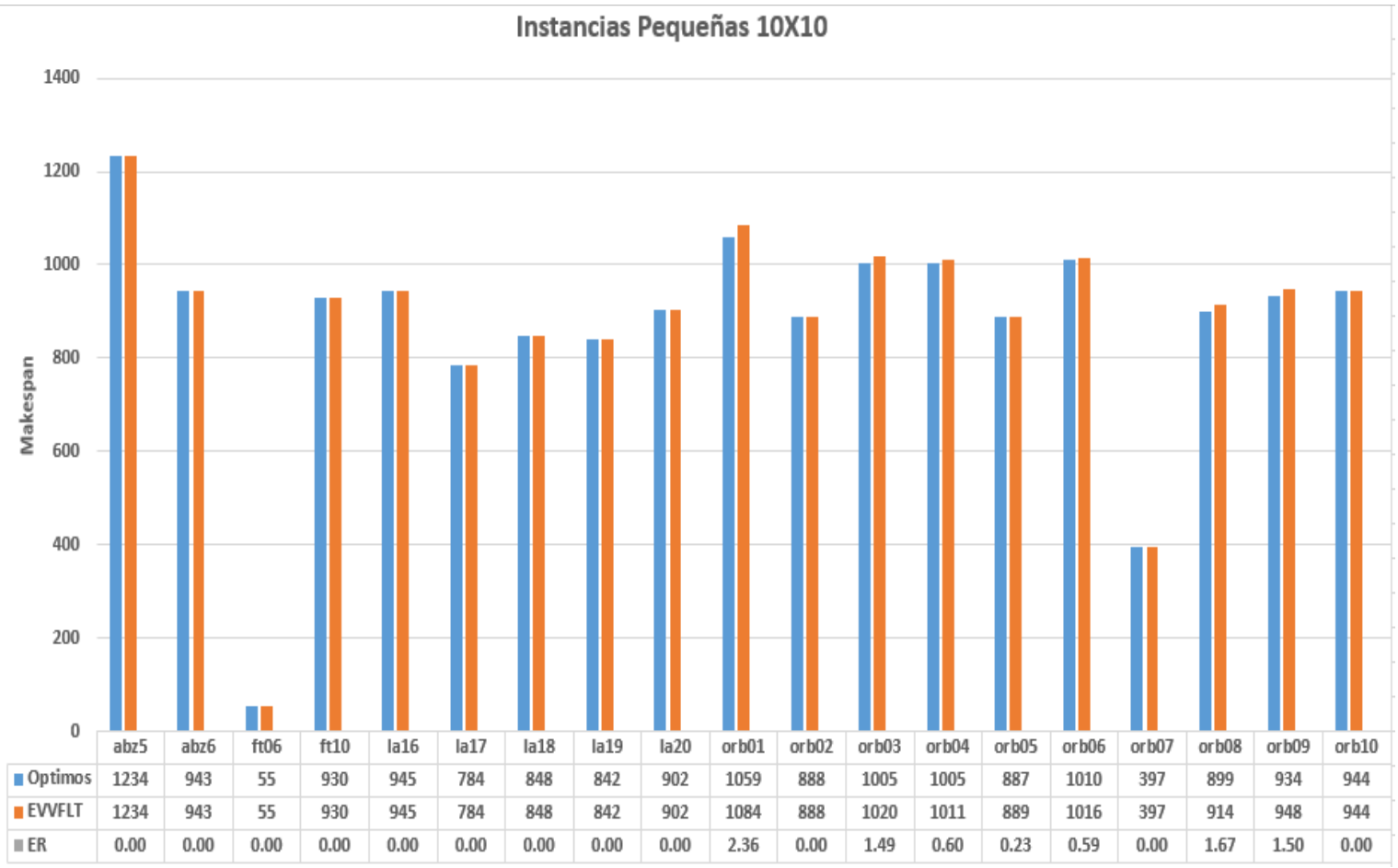
4.2 Diseño de los experimentos

Se realizaron 30 ejecuciones por cada instancia, utilizando en cada una temperatura inicial diferente. Esta variabilidad se debe a que la solución inicial se genera de manera aleatoria y, por lo tanto, tiene una probabilidad muy baja de repetirse entre ejecuciones. Por ejemplo, para la instancia YN1, las soluciones iniciales se encontraron en un rango de valores de 1610 a 2223, lo que genera diversidad en los puntos de partida del algoritmo. Este comportamiento ejemplifica la complejidad del problema, ya que incluso para una sola instancia se podrían tener un gran número de posibles puntos de inicio, reflejando la dificultad de explorar eficientemente todo el espacio de soluciones. El llevar a cabo múltiples ejecuciones permite no solo capturar esta diversidad, sino también demostrar la efectividad del algoritmo propuesto EVVFLT, evaluando su desempeño de manera robusta frente a distintos puntos de partida y asegurando la consistencia de los resultados obtenidos.

4.3 Presentación de resultados

Se corrieron 30 pruebas de cada instancia y se seleccionó la mejor solución para su análisis. Se presentan gráficas con los resultados obtenidos, considerando un tiempo de ejecución promedio de una hora por instancia. En el caso de instancias pequeñas, la calidad de las soluciones encontradas es alta, mostrando un buen ajuste entre tiempo de cómputo y desempeño. Para instancias de tamaño mediano y grande, los resultados pueden considerarse aceptables, ya que la complejidad del problema incrementa de forma significativa y limita la posibilidad de alcanzar soluciones óptimas en tiempos razonables. No obstante, los resultados obtenidos permiten observar el comportamiento del algoritmo propuesto EVVFLT y demostrar su capacidad para ofrecer soluciones factibles y de buena calidad dentro de un marco de tiempo restringido, lo que refuerza su efectividad frente a la dificultad inherente del problema.

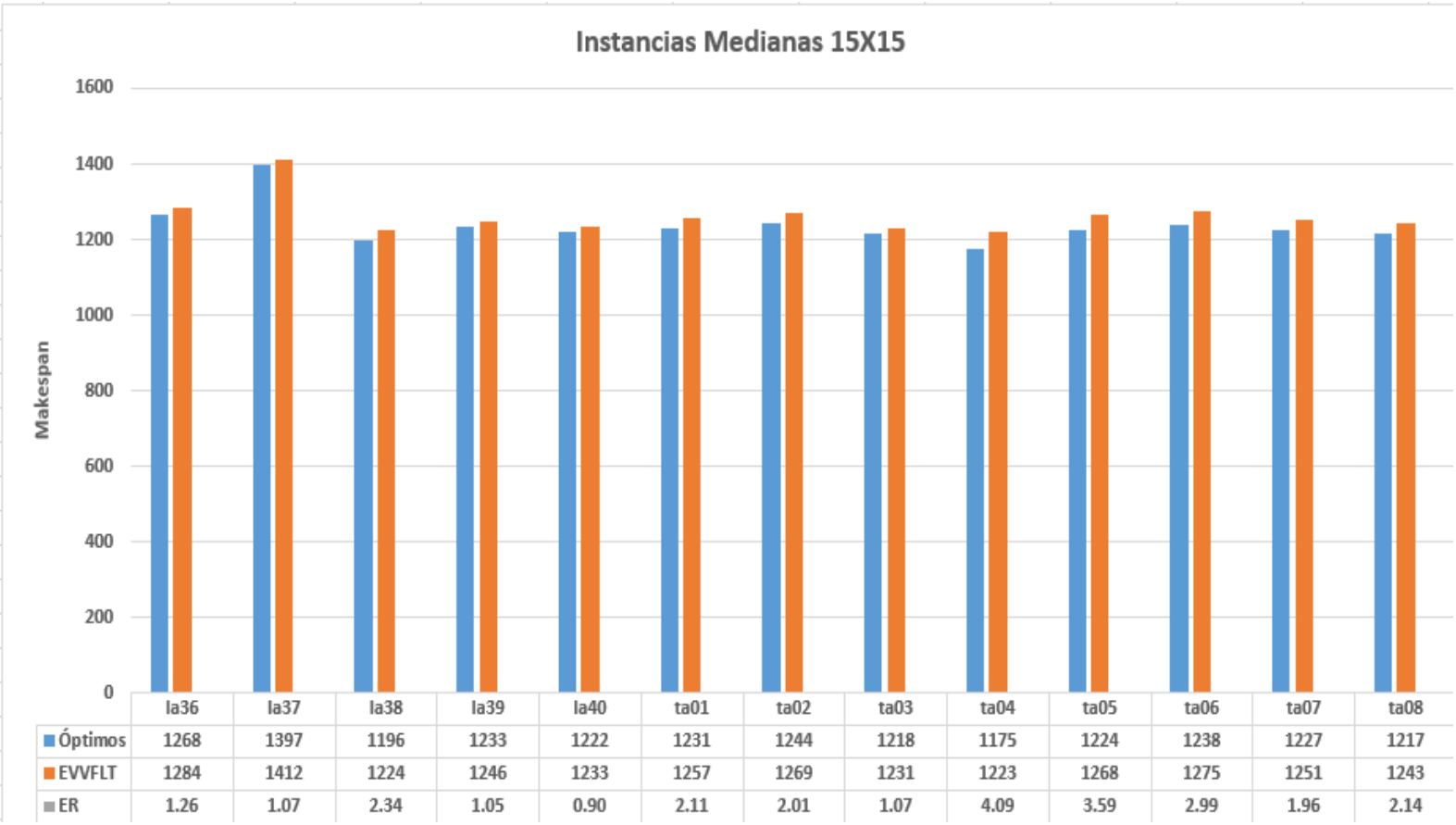
La Fig. 21, titulada Óptimos vs Soluciones encontradas por EVVFLT, presenta de manera gráfica los resultados obtenidos por el algoritmo en comparación con los valores óptimos reportados en la literatura. La figura incluye, además, el porcentaje de error relativo (ER) correspondiente a cada instancia, lo que permite evaluar de forma clara la precisión y efectividad del método propuesto.



	abz5	abz6	ft06	ft10	la16	la17	la18	la19	la20	orb01	orb02	orb03	orb04	orb05	orb06	orb07	orb08	orb09	orb10
Optimos	1234	943	55	930	945	784	848	842	902	1059	888	1005	1005	887	1010	397	899	934	944
EVVFLT	1234	943	55	930	945	784	848	842	902	1084	888	1020	1011	889	1016	397	914	948	944
ER	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	2.36	0.00	1.49	0.60	0.23	0.59	0.00	1.67	1.50	0.00

Fig. 21 Óptimos vs Soluciones encontradas por EVVFLT

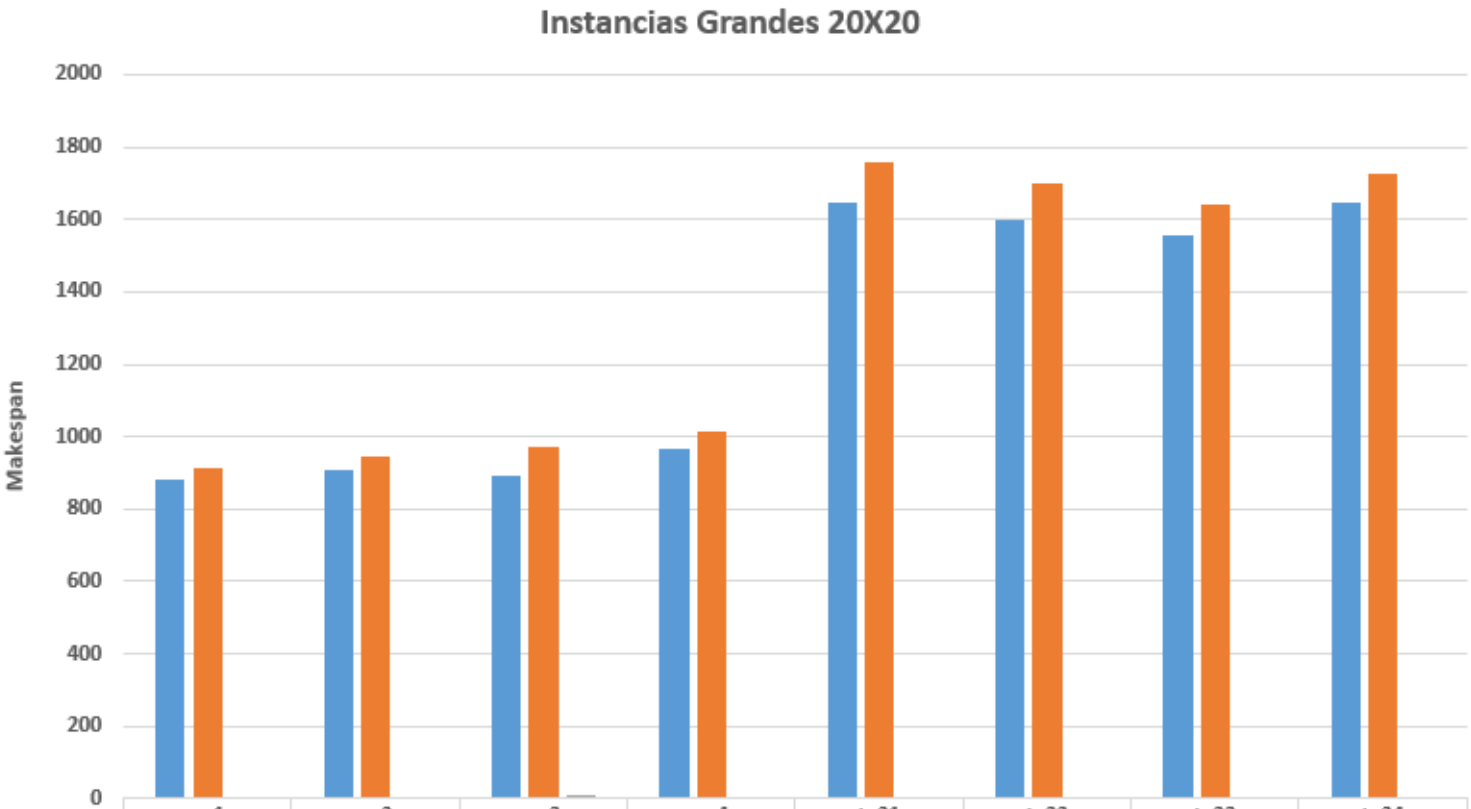
La Fig. 22, titulada resultados para instancias medianas, ilustra de manera gráfica las soluciones obtenidas por el algoritmo EVVFLT en comparación con los valores óptimos conocidos. La figura permite observar la tendencia del desempeño del método frente a problemas de tamaño intermedio, destacando tanto la proximidad de las soluciones a los óptimos como el porcentaje de error relativo (ER) en cada instancia. En general, se aprecia que el error relativo se incrementa ligeramente en comparación con las instancias pequeñas, lo que refleja el aumento de la complejidad del problema y la dificultad de mantener la misma precisión a medida que crece el tamaño de las instancias. Este análisis facilita evaluar la consistencia y robustez del algoritmo frente a problemas de complejidad moderada.



	la36	la37	la38	la39	la40	ta01	ta02	ta03	ta04	ta05	ta06	ta07	ta08
Óptimos	1268	1397	1196	1233	1222	1231	1244	1218	1175	1224	1238	1227	1217
EVVFLT	1284	1412	1224	1246	1233	1257	1269	1231	1223	1268	1275	1251	1243
ER	1.26	1.07	2.34	1.05	0.90	2.11	2.01	1.07	4.09	3.59	2.99	1.96	2.14

Fig. 22 Resultados para instancias medianas

La Fig. 23, titulada resultados para instancias grandes, se aprecia que el error relativo tiende a aumentar con respecto a instancias pequeñas y medianas, aunque el incremento no es significativo. Esto evidencia que el algoritmo mantiene un desempeño consistente y robusto incluso al enfrentar problemas de mayor tamaño y complejidad.



	yn1	yn2	yn3	yn4	ta21	ta22	ta23	ta24
Óptimos	884	907	892	968	1644	1600	1557	1646
EVVFLT	914	944	970	1012	1757	1702	1639	1727
ER	3.39	4.08	8.74	4.55	6.87	6.38	5.27	4.92

Fig. 23 Resultados para instancias grandes

4.4 Análisis de resultados

El método propuesto presenta diversas fortalezas, entre las que destaca su simplicidad de implementación y ejecución. En particular, se basa en el uso de una estructura de datos lineal, lo que facilita la realización de operaciones durante el recorrido de las matrices y contribuye a mejorar el tiempo de cómputo. No obstante, el análisis de resultados también permite identificar ciertas limitaciones. En algunos casos, cuando no existe un par de operaciones adyacentes en la línea de tiempo, el método no realiza permutaciones y transfiere el control directamente al algoritmo de recocido simulado. Asimismo, no se cuenta con un criterio preciso para determinar el momento en el que el algoritmo alcanza un estado en el que existen pocos pares adyacentes pertenecientes a las rutas críticas, lo que reduce la efectividad de la búsqueda y limita el impacto de las permutaciones en la mejora de la solución.

Otro aspecto observado es el recalentamiento, en algunas ocasiones se activa y logra mejorar la solución, pero en otras simplemente se alcanza el tiempo máximo de ejecución sin obtener mejoras adicionales. Esto parece deberse a la presencia de óptimos locales muy profundos. Asimismo, cuando la temperatura se incrementa demasiado, el algoritmo requiere un tiempo considerable para regresar al punto desde el cual se produjo dicho incremento, lo que puede afectar la eficiencia global del proceso.

Adicionalmente, se analizó la cantidad de ciclos generados en la lista que contiene las operaciones a permutar. Dicha lista presenta una restricción adicional, ya que funciona como una cola, el primer elemento debe atenderse antes de pasar al siguiente, y así sucesivamente. Esto provoca que el orden de atención de las operaciones se mantenga rígido, lo cual añade una dificultad extra al proceso de búsqueda de soluciones, pues las permutaciones en las máquinas se reflejan de manera distinta a las de la lista.

Como consecuencia, se generan numerosos ciclos que conducen a cada solución por un rumbo distinto, lo que puede alejarla considerablemente de la solución vecina que se esperaba obtener. Sin embargo, esto también podría tener un aspecto positivo, ya que permitiría explorar un espacio de búsqueda más amplio. No obstante, no se puede afirmar con certeza si este efecto resulta realmente beneficioso.

4.5 Estadística inferencial

La estadística inferencial permite extraer conclusiones sobre una población a partir de muestras, estimando parámetros desconocidos, evaluando hipótesis y prediciendo comportamientos futuros, siempre considerando un margen de error y un nivel de confianza. Entre sus herramientas principales se encuentran los intervalos de confianza, pruebas de hipótesis, valor p, nivel de significancia (α) y medidas de variabilidad como el error estándar. Además, se utiliza para comparar resultados de distintos métodos aplicados a un mismo problema, identificando cuál presenta mejor desempeño o diferencias significativas.

La Tabla 5 resultados experimentales del método EVVFLT presenta los resultados experimentales obtenidos al aplicar el método heurístico, sobre el conjunto de instancias de prueba. Cada instancia fue ejecutada 30 veces de manera independiente, con el propósito de evaluar la calidad, estabilidad y eficiencia del algoritmo. En la primera columna se muestra el nombre de la instancia, seguido del óptimo conocido, que representa el mejor valor reportado en la literatura o la mejor conocida y sirve como referencia de comparación. La tercera columna indica la mejor solución obtenida por el

método EVVFLT, mientras que la cuarta presenta el promedio de las soluciones alcanzadas en las 30 ejecuciones. Posteriormente, la desviación estándar mide la variabilidad de los resultados con respecto al promedio, indicando el grado de consistencia del método.

La columna correspondiente al error relativo (RE) expresa el porcentaje de desviación de la mejor solución respecto al óptimo conocido, lo que permite valorar la precisión del algoritmo. A continuación, la peor solución refleja el rendimiento menos favorable observado entre las repeticiones, y finalmente, la columna de tiempo reporta el tiempo de cómputo (en segundos) requerido por el programa para obtener las soluciones. En conjunto, estos indicadores permiten analizar el comportamiento general del método EVVFLT en términos de eficiencia, estabilidad y cercanía al óptimo en las distintas instancias evaluadas.

Instancia	Óptimo	Mejor	Promedio	Desviación Estandar	RE Δ %	Peor	Tiempo Seg
abz5	1234	**1234**	1261.00	28.99	0.00	1274	1045.42
abz6	943	**943**	946.00	6.71	0.00	948	1877.11
ft06	55	**55**	55.00	0.00	0.00	55	0.02
ft10	930	**930**	960.00	14.44	0.00	1000	1680.01
la16	945	**945**	980.00	20.70	0.00	1016	97.09
la17	784	**784**	791.00	5.17	0.00	797	248.23
la18	848	**848**	860.00	7.01	0.00	880	115.69
la19	842	**842**	857.00	8.86	0.00	875	652.37
la20	902	**902**	909.00	3.11	0.00	915	1273.46
orb01	1059	1084	1129.00	48.93	2.36	1158	3600.01
orb02	888	**888**	900.00	10.77	0.00	926	759.01
orb03	1005	1020	1082.00	23.97	1.49	1132	3600.01
orb04	1005	1011	1045.00	12.07	0.60	1063	3600.01
orb05	887	889	933.00	16.92	0.23	962	3600.01
orb06	1010	1016	1059.00	21.70	0.59	1112	3600.01
orb07	397	**397**	406.00	5.99	0.00	421	51.05
orb08	899	914	972.00	26.98	1.67	1014	3600.01
orb09	934	948	965.00	11.32	1.50	998	3600.01
orb10	944	**944**	984.00	25.39	0.00	1040	112.22
la36	1268	1284	1344.00	37.23	1.26	1436	3600.01
la37	1397	1412	1471.00	18.30	1.07	1508	3600.01
la38	1196	1224	1290.00	20.74	2.34	1344	3600.01
la39	1233	1246	1306.00	22.86	1.05	1358	3600.01
la40	1222	1233	1279.00	16.45	0.90	1335	3600.01
ta01	1231	1257	1290.00	17.66	2.11	1321	3600.01

ta02	1244	1269	1312.00	22.12	2.01	1332	3600.01
ta03	1218	1231	1299.00	26.24	1.07	1338	3600.01
ta04	1175	1223	1273.00	28.89	4.09	1361	3600.01
ta05	1224	1268	1321.00	22.86	3.59	1386	3600.01
ta06	1238	1275	1309.00	18.10	2.99	1346	3600.01
ta07	1227	1251	1301.00	19.19	1.96	1342	3600.01
ta08	1217	1243	1303.00	21.89	2.14	1356	3600.01
yn1	884	914	948.00	19.26	3.39	979	3600.01
yn2	907	944	988.00	26.67	4.08	1107	3600.01
yn3	892	970	1043.00	30.58	8.74	1101	3600.01
yn4	968	1012	1057.00	14.65	4.55	1083	3600.01
ta21	1644	1757	1852.00	31.39	6.87	1927	3600.01
ta22	1600	1702	1784.00	45.41	6.38	1925	3600.01
ta23	1557	1639	1755.00	40.51	5.27	1830	3600.01
ta24	1646	1727	1827.00	34.33	4.92	1870	3600.01

Tabla 5 Estadística de la heurística EVVFLT

Para el análisis de estadística inferencial se utilizaron únicamente 24 instancias de tamaño pequeño y mediano (hasta la instancia la40), debido a que algunos de los autores de las heurísticas SysLS, GBA, TB y mXLSGA no realizaron pruebas con instancias de mayor tamaño, como yn1, yn2, yn3, yn4, ta21, ta22, ta23 y ta24. A pesar de esta limitación, el objetivo principal de este estudio es comprobar la existencia de diferencias significativas entre los métodos evaluados. El objetivo de la estadística inferencial es demostrar que existen diferencias en los métodos usados.

Para aplicar correctamente estos métodos es necesario verificar supuestos estadísticos, principalmente la normalidad de los datos y la homocedasticidad de las varianzas entre grupos. La normalidad permite el uso de pruebas paramétricas, como t de Student o ANOVA, mientras que la homocedasticidad asegura que las diferencias observadas se deban a los factores estudiados y no a variaciones desiguales en los datos. **Kruskal y Wallis (1952).**

- Hipótesis nula (H_0): las varianzas de las poblaciones son iguales.
- Hipótesis alternativa (H_1): al menos una de las varianzas difiere.

4.5.1 Normalidad

La Fig. 24 muestra las gráficas de normalidad (Q-Q plots) correspondientes a las seis heurísticas analizadas, donde los datos faltantes se mantuvieron como NaN y fueron

omitidos del análisis para evitar la introducción de sesgos artificiales. En estas gráficas, los valores del eje X representan los valores teóricos, es decir, los cuantiles esperados de una distribución normal ideal con media y desviación estándar equivalentes a las de la muestra. Los valores del eje Y corresponden a los valores observados, que son los datos reales ordenados de menor a mayor para cada heurística.

Si los datos siguieran una distribución normal perfecta, los puntos se alinearían a lo largo de una línea recta ascendente, como lo muestra la primer grafica de EVVFLT. Sin embargo, en los resultados obtenidos se observa una desviación significativa de los puntos con respecto a dicha tendencia, especialmente en las heurísticas GBA y mXLSGA, donde algunos valores altos (superiores a 4 y 6, respectivamente) generan colas derechas pronunciadas. Esta dispersión en los valores extremos, junto con la acumulación de observaciones en valores bajos o cercanos a cero, indica una clara asimetría y sesgo positivo en los datos. Por lo tanto, se concluye que las distribuciones de las heurísticas no cumplen con el supuesto de normalidad, siendo recomendable aplicar pruebas no paramétricas o transformaciones de datos antes de realizar análisis estadísticos que requieran dicha condición.

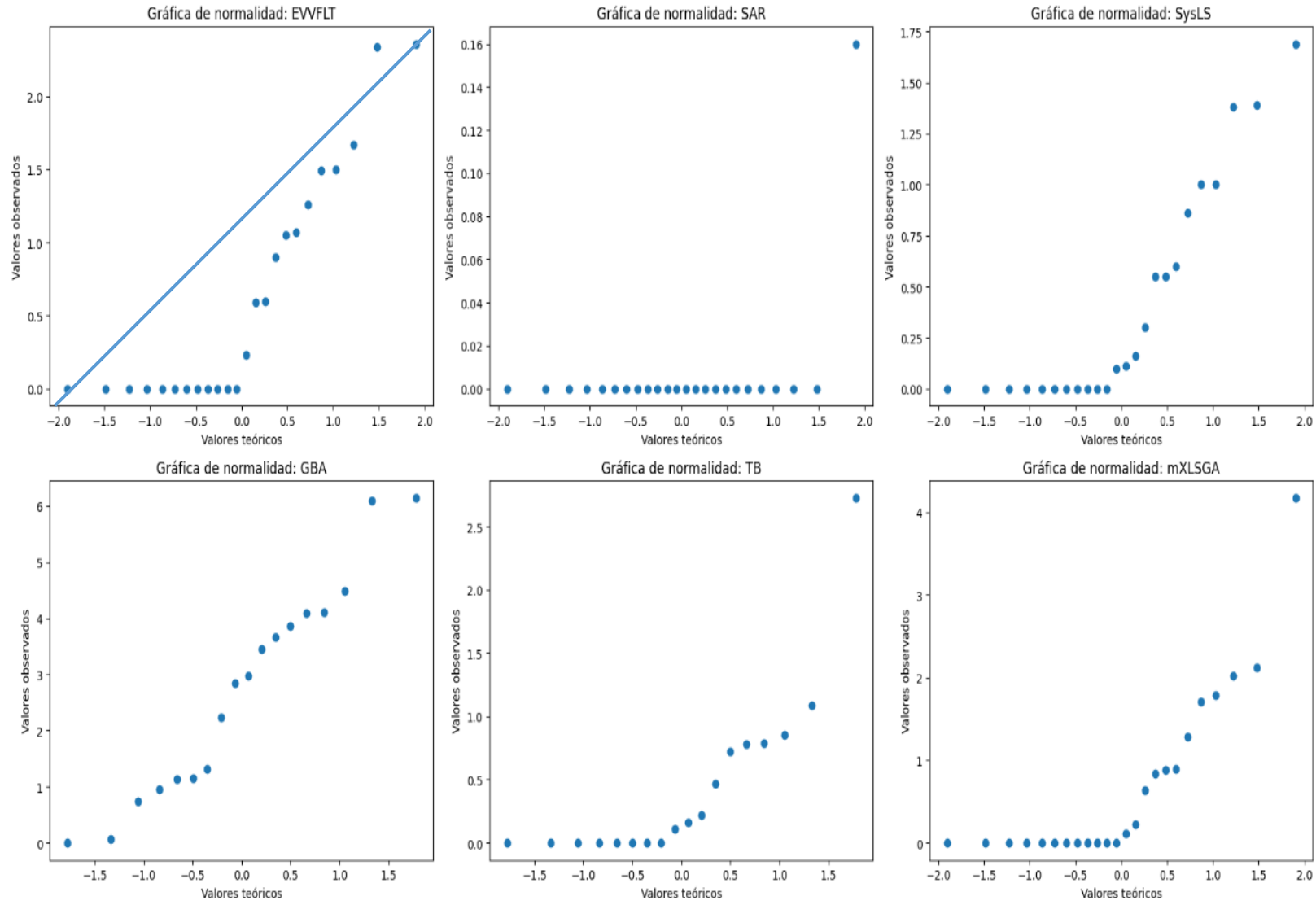


Fig. 24 Normalidad de los datos descartada

La prueba de Bartlett Fig. 25 Gráfica de bigotes, es una herramienta estadística utilizada para verificar el supuesto de homogeneidad de varianzas o homocedasticidad entre varios grupos. La prueba de Bartlett evalúa si las diferencias observadas entre las varianzas de los grupos son atribuibles al azar o si existen evidencias suficientes para concluir que las varianzas difieren significativamente. Su hipótesis nula establece que todas las varianzas poblacionales son iguales, mientras que la hipótesis alternativa plantea que al menos una de ellas es diferente. Si el valor *p* obtenido es menor que el nivel de significancia (por ejemplo, α = 0.05), en nuestro caso se rechaza la hipótesis nula, indicando que las varianzas no son homogéneas. Ya que los datos se distribuyen

de forma diferente en cada heurística. La heurística SAR parece que todos los valores son 0 pero en realidad están todos por debajo de 1% de error relativo.

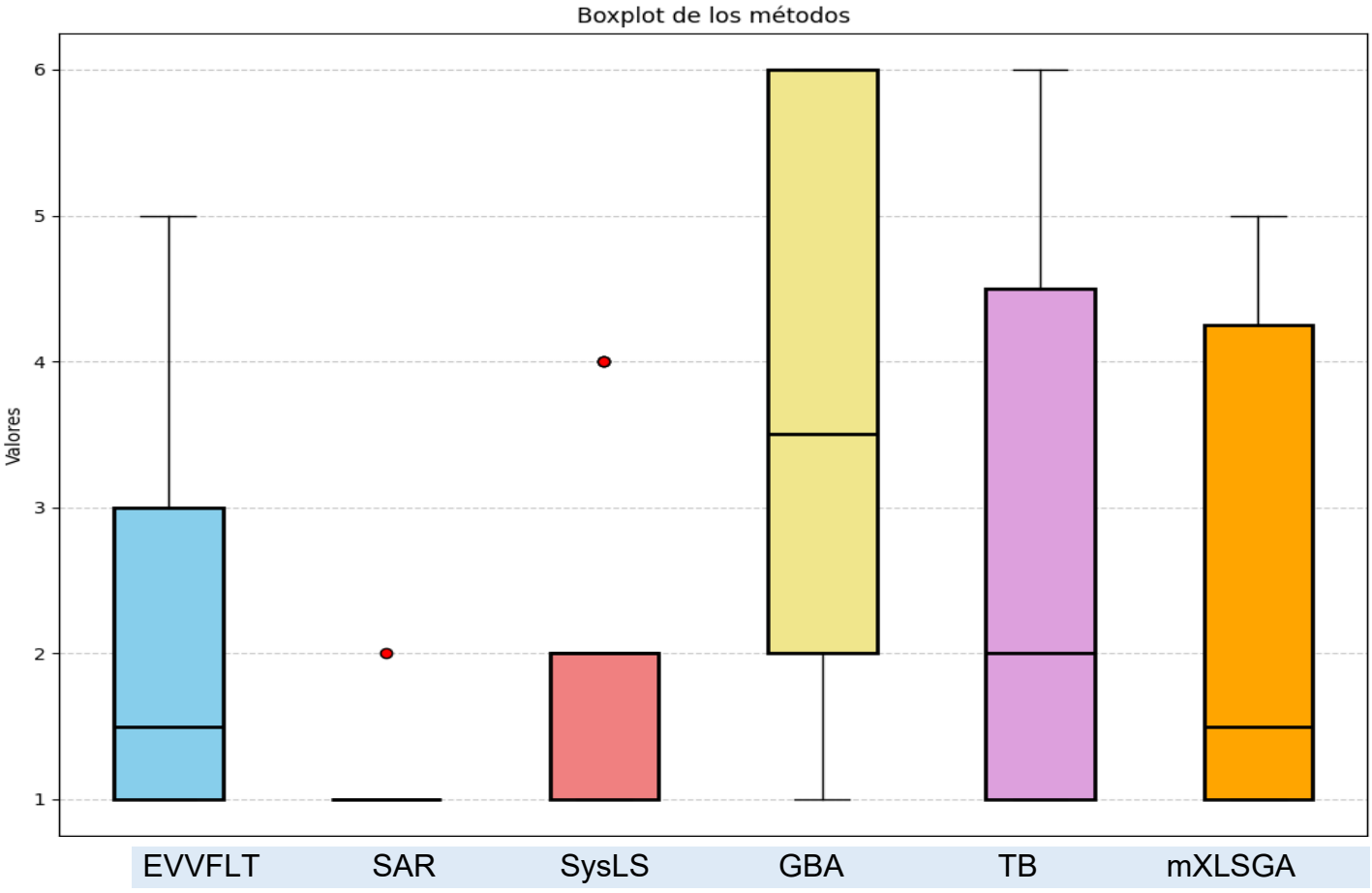


Fig. 25 Gráfica de bigotes con distribución de datos

La Tabla 6 presenta la comparación de los resultados obtenidos por las seis heurísticas analizadas (EVVFLT, SAR, SysLS, GBA, TB y mXLSGA) en las instancias de tipo chico y mediano, junto con sus respectivos valores promedio de ranking. Los resultados muestran que la heurística SAR alcanzó el mejor desempeño general, al registrar el valor promedio de ranking más bajo (1.04), seguida por SysLS (2.20) y EVVFLT (2.33), las cuales demostraron un comportamiento eficiente y consistente en la mayoría de las instancias. En contraste, las heurísticas mXLSGA (2.58), TB (3.20) y GBA (4.58) obtuvieron valores de ranking más elevados, lo que refleja un rendimiento comparativamente menor. Con base en estos resultados, es posible clasificar el desempeño de las heurísticas en tres niveles: alto (SAR, SysLS y EVVFLT), medio

(mXLSGA y TB) y bajo (GBA), evidenciando diferencias significativas en su efectividad para resolver instancias de menor tamaño. Cabe destacar que dichas diferencias podrían acentuarse más al considerar instancias de mayor complejidad o tamaño, donde las variaciones en la calidad de las soluciones y la eficiencia de cada método tienden a ser más notorias.

Instancia	Tamaño	EVVFLT	SAR	SysLS	GBA	TB	mXLSGA
abz5	10x10	0.00 (1)	0.00 (1)	0.16 (5)	1.15 (4)	0.16 (6)	0.00 (1)
abz6	10x10	0.00 (1)	0.00 (1)	0.00 (1)	1.32 (6)	0.00 (1)	0.00 (1)
ft06	6X6	0.00 (1)	0.00 (1)	0.00 (1)	0.00 (1)	0.00 (1)	0.00 (1)
ft10	10x10	0.00 (1)	0.00 (1)	0.86 (4)	3.45 (5)	* (6)	0.00 (1)
la16	10x10	0.00 (1)	0.00 (1)	0.00 (1)	0.74 (2)	0.00 (1)	0.00 (1)
la17	10x10	0.00 (1)	0.00 (1)	0.00 (1)	0.06 (2)	0.00 (1)	0.00 (1)
la18	10x10	0.00 (1)	0.00 (1)	0.00 (1)	0.95 (2)	0.00 (1)	0.00 (1)
la19	10x10	0.00 (1)	0.00 (1)	0.00 (1)	3.66 (2)	0.00 (1)	0.00 (1)
la20	10x10	0.00 (1)	0.00 (1)	0.55 (5)	1.13 (6)	0.00 (1)	0.00 (1)
orb01	10x10	2.36 (5)	0.00 (1)	0.10 (2)	6.15 (6)	0.47 (3)	0.84 (4)
orb02	10x10	0.00 (1)	0.00 (1)	0.11 (3)	2.84 (4)	0.11 (3)	0.11 (3)
orb03	10x10	1.49 (3)	0.00 (1)	1.69 (4)	6.09 (6)	1.09 (2)	1.79 (5)
orb04	10x10	0.60 (3)	0.00 (1)	0.60 (3)	2.97 (6)	0.79 (5)	0.00 (1)
orb05	10x10	0.23 (5)	0.00 (1)	0.00 (1)	4.10 (6)	0.22 (3)	0.22 (3)
orb06	10x10	0.59 (3)	0.00 (1)	0.30 (2)	4.11 (5)	* (6)	0.89 (4)
orb07	10x10	0.00 (1)	0.00 (1)	0.00 (1)	3.87 (5)	* (6)	0.00 (1)
orb08	10x10	1.67 (4)	0.00 (1)	0.00 (1)	4.49 (5)	* (6)	0.88 (3)
orb09	10x10	1.50 (4)	0.00 (1)	0.00 (1)	2.24 (5)	* (6)	0.64 (3)
orb10	10x10	0.00 (1)	0.00 (1)	0.00 (1)	* (5)	* (5)	0.00 (1)
la36	15X15	1.26 (4)	0.00 (1)	0.55 (2)	* (6)	0.78 (3)	2.12 (5)
la37	15X15	1.07 (4)	0.00 (1)	1.00 (2)	* (6)	0.85 (3)	1.28 (5)
la38	15X15	2.34 (3)	0.00 (1)	1.00 (2)	* (6)	2.73 (4)	4.18 (5)
la39	15X15	1.05 (3)	0.00 (1)	1.38 (4)	* (6)	0.72 (2)	2.02 (5)
la40	15X15	0.90 (3)	0.16 (2)	1.39 (4)	* (6)	0.00 (1)	1.71 (5)
Ranking		2.33	1.04	2.20	4.58	3.20	2.58

Tabla 6 Tabla comparativa con ranking de las 6 heurísticas

Cuando estos supuestos no se cumplen, se recurre a métodos no paramétricos, que no requieren que los datos se ajusten a una distribución específica y son más robustos frente a valores atípicos o distribuciones asimétricas. Un ejemplo de ello es la prueba de **Kruskal–Wallis (1952)** utilizada para comparar dos o más grupos independientes

cuando los datos no son normales o las varianzas son heterogéneas. Este método se basa en los rangos de los datos en lugar de sus valores absolutos. Para calcularlos, se ordenan todos los valores observados de menor a mayor y se les asigna un número consecutivo (rango), de modo que el valor más pequeño recibe el rango 1 y el más grande el rango *N*, siendo *N* el total de observaciones. Si existen empates, se asigna a cada valor empatado el promedio de los rangos correspondientes. Posteriormente, se calcula la suma de rangos de cada grupo y se analiza si las diferencias entre estas sumas son lo suficientemente grandes como para considerar que las medianas de los grupos difieren de forma significativa.

La finalidad del método es obtener el valor *Z* de contraste, el cual permite determinar si existen diferencias estadísticamente significativas entre los grupos analizados. Este valor se compara con una región crítica o región de rechazo definida por el nivel de significancia seleccionado. Si el valor *Z* calculado se encuentra fuera de la región de rechazo, se concluye que existen diferencias significativas entre los grupos; en caso contrario, no se rechaza la hipótesis nula de igualdad de medianas.

El uso de rangos permite comparar grupos con diferentes distribuciones y escalas, proporcionando una medida robusta del orden relativo de los datos en lugar de su magnitud numérica. La siguiente Tabla 7 Rangos muestra los resultados necesarios para llevar a cabo el método de Kruskal–Wallis.

Instancia	EVVFLT	SAR	SysLS	GBA	TB	mXLSGA
abz5	34	34	74	104	74	34.00
abz6	34	34	34	107	34	34.00
ft06	34	34	34	34	34	34.00
ft10	34	34	93	124	*	34.00
la16	34	34	34	88	34	34.00
la17	34	34	34	68	34	34.00
la18	34	34	34	97	34	34.00
la19	34	34	34	125	34	34.00
la20	34	34	81.5	103	34	34.00
orb01	120	34	69	132	80	91.00
orb02	34	34	71	122	71	71.00
orb03	110	34	113	131	102	115.00
orb04	84.5	34	84.5	123	90	34.00
orb05	78	34	34	127	76.5	76.50
orb06	83	34	79	128	*	95.00
orb07	34	34	34	126	*	34.00
orb08	112	34	34	130	*	94.00
orb09	111	34	34	118	*	86.00
orb10	34	34	34	*	*	34.00
la36	105	34	81.5	*	89	117.00
la37	101	34	98.5	*	92	106.00

la38	119	34	98.5	*	121	129.00
la39	100	34	108	*	87	116.00
la40	96	74	109	*	34	114.00
	1628	856	1535	1987	1155	1618.50

Tabla 7 Suma de rangos de las 6 heurísticas

La nomenclatura empleada en la función de **Kruskal–Wallis (1953)** se detalla a continuación para una mejor comprensión del procedimiento estadístico. El símbolo *Z* representa el estadístico de contraste, utilizado para determinar si existen diferencias significativas entre los grupos analizados. La letra *C* corresponde al número total de heurísticas consideradas en el estudio, mientras que n_i denota el número de observaciones asociadas al *i*-ésimo algoritmo. Por su parte, *N* representa el total de observaciones obtenidas al sumar todas las muestras de los algoritmos incluidos en el análisis.

El parámetro *Ri* indica la suma de los rangos correspondientes al *i*-ésimo algoritmo. Para su cálculo, los valores se ordenan de menor a mayor y se asignan rangos consecutivos del 1 al *m*, sumándose luego los rangos pertenecientes a cada grupo o heurística. La variable *t* se refiere al número de resultados repetidos o empates, los cuales se consideran para ajustar correctamente el valor del estadístico y mantener la validez de la prueba.

El nivel de significancia utilizado en este análisis es 0.05, lo que implica un 5 % de probabilidad de que las diferencias observadas se deban al azar. Finalmente, el valor *p* expresa la probabilidad de aceptar la hipótesis nula bajo los datos observados; cuando este valor es mayor que 0.05, no se rechaza la hipótesis nula, mientras que un valor inferior a dicho umbral indica la existencia de diferencias estadísticamente significativas entre las medianas de los grupos evaluados.

$$Z = \frac{\frac{12}{N(N+1)} \sum_{i=1}^{C} \frac{R_i^2}{n_i} - 3(N+1)}{1 - \frac{\sum(t^3 - t)}{N^3 - N}}$$

$$Z = \frac{\frac{12}{132(132+1)} \frac{1628^2}{24} + \frac{856^2}{24} + \frac{1535^2}{24} + \frac{1987^2}{18} + \frac{1155^2}{18} + \frac{1618.5^2}{24} - 3(132+1)}{1 - \frac{(67^3 - 67) + (3^3 - 3) + (3^3 - 3) + (3^3 - 3) + (2^3 - 2) + (2^3 - 2) + (2^3 - 2) + (2^3 - 2)}{132^3 - 132}}$$

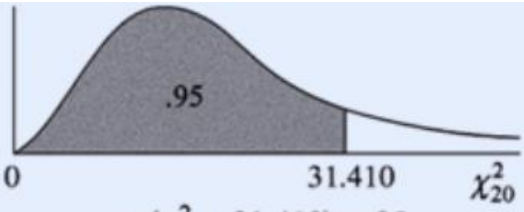
$$Z = \frac{\frac{12}{17556}\left(\frac{2{,}651{,}184}{24} + \frac{732{,}736}{24} + \frac{2{,}356{,}225}{24} + \frac{3{,}948{,}169}{18} + \frac{1{,}333{,}025}{18} + \frac{2{,}620{,}892.25}{24}\right) - 3(132+1)}{1 - \frac{(67^3 - 67) + (3^3 - 3) + (3^3 - 3) + (3^3 - 3) + (2^3 - 2) + (2^3 - 2) + (2^3 - 2) + (2^3 - 2)}{132^3 - 132}}$$

$$Z = \frac{(0.000684)\,(110466 + 30{,}530.67 + 98{,}176.04 + 219{,}342.72 + 74{,}612.5 + 109{,}203.84) - 399}{1 - \frac{(300{,}696) + (24) + 24) + (24) + (6) + (6) + (6) + (6)}{2{,}299{,}836}}$$

$$Z = \frac{40.055}{0.8692}$$

$$Z = 46.08$$

La Tabla 8 presenta los valores críticos de la distribución Ji-cuadrada (χ^2), utilizada para determinar la región de rechazo en la prueba de Kruskal–Wallis. El resultado obtenido para el estadístico de contraste fue Z = 46.08. Para determinar su significancia, se compara con el valor crítico de la distribución Ji-cuadrada (χ^2), correspondiente a 5 grados de libertad (gl = k−1 = 6−1) y un nivel de significancia de α=0.05, cuyo valor es 11.07.



$p(\chi^2 < 31.410) = .95$

g.l.	$\chi^2_{.005}$	$\chi^2_{.025}$	$\chi^2_{.05}$	$\chi^2_{.90}$	$\chi^2_{.95}$	$\chi^2_{.975}$	$\chi^2_{.99}$	$\chi^2_{.995}$
1	.0000393	.00982	.00393	2.706	3.891	5.024	6.635	7.879
2	.0100	.0506	.103	4.605	5.991	7.378	9.210	10.597
3	.0717	.216	.352	6.251	7.815	9.348	11.345	12.838
4	.207	.484	.711	7.779	9.488	11.143	13.277	14.860
5	.412	.831	1.145	9.236	11.070	12.832	15.086	16.750
6	.676	1.237	1.635	10.645	12.592	14.449	16.812	18.548
7	.989	1.690	2.167	12.017	14.067	16.013	18.475	20.278
8	1.344	2.180	2.733	13.362	15.507	17.535	20.090	21.955
9	1.735	2.700	3.325	14.684	16.919	19.023	21.666	23.589
10	2.156	3.247	3.940	15.987	18.307	20.433	23.209	25.188

Tabla 8 Distribución Ji-cuadrada

La Fig. 26 muestra la distribución Ji-cuadrada (χ^2) con un nivel de significancia de $\alpha = 0.05$ y 5 grados de libertad, donde se delimita claramente la región de rechazo. Esta región representa los valores del estadístico que exceden el valor crítico de *11.07*, correspondiente al límite superior del área bajo la curva con una probabilidad del 5 %. En el gráfico se observa cómo el valor obtenido *Z=46.08* se encuentra muy por encima del valor crítico, ubicándose dentro de la región de rechazo. Esto confirma visualmente que existen diferencias significativas entre las heurísticas analizadas, apoyando la conclusión estadística de que la hipótesis nula debe ser rechazada.

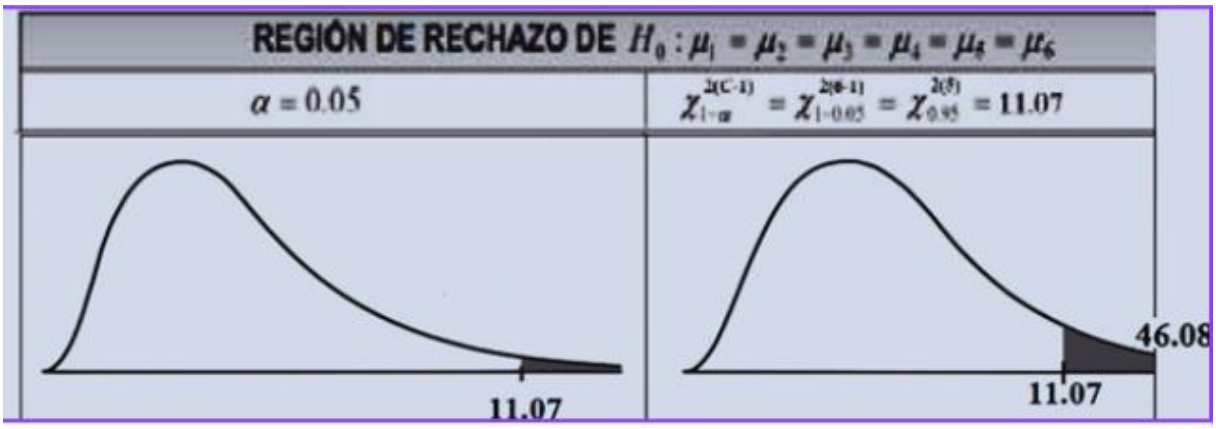


Fig. 26 Región de rechazo para la prueba de hipótesis Ji-Cuadrada (α = 0.05)

4.6 Análisis estadístico Post-Hoc Wilcoxon

En la prueba de rangos con signo de Wilcoxon **Fligner (1985)**, el estadístico principal se representa como W y se define como el menor valor entre las sumas de rangos positivos y negativos, es decir: *W=min(W+,W−)* donde *W+* corresponde a la suma de los rangos con signo positivo y *W−* a la suma de los rangos con signo negativo. Este estadístico *W* refleja la magnitud y dirección de las diferencias entre los pares de observaciones analizadas.

En este estudio se cuenta con 24 observaciones, por lo que se utiliza directamente el estadístico W para contrastar la hipótesis nula con los valores críticos de las tablas de Wilcoxon, sin recurrir a la aproximación mediante el valor *Z*.

Esta conversión solo se aplica cuando el número de observaciones es mayor a 25, ya que en esos casos la distribución de W puede aproximarse a una distribución normal estándar.

La interpretación del estadístico W se realiza comparando el valor calculado con los valores críticos tabulados para el nivel de significancia seleccionado ($\alpha = 0.05$) y el número de observaciones consideradas. Si el valor obtenido de W es menor o igual al

valor crítico, se rechaza la hipótesis nula (H_0), lo que indica que existen diferencias estadísticamente significativas entre las dos heurísticas comparadas. En cambio, si *W* es mayor que el valor crítico, no se rechaza H_0, concluyéndose que no hay evidencia suficiente para afirmar que las medianas de las heurísticas difieren de manera significativa. Este procedimiento permite identificar, de forma precisa y par a par, qué heurísticas presentan comportamientos distintos en su desempeño, complementando el análisis global obtenido mediante la prueba de Kruskal–Wallis.

Los valores críticos del estadístico *W* se determinan a partir de las tablas de referencia de la prueba de Wilcoxon, que establecen los límites de aceptación y rechazo en función del número de observaciones *(n)* y del nivel de significancia (α) seleccionado. En este caso, dichos valores se ilustran en la Tabla 9 *valores críticos para la prueba de rango con signos*, donde se muestra la correspondencia entre el tamaño de la muestra y el valor crítico de *W*. A partir de esta tabla se selecciona el valor crítico adecuado para 24 muestras y un nivel de significancia del 5%, el cual sirve como punto de referencia para decidir si el resultado obtenido conduce o no al rechazo de la hipótesis nula.

n	Unilateral α=0.01 Bilateral α=0.02	Unilateral α=0.025 Bilateral α=0.05	Unilateral α=0.05 Bilateral α=0.10
5	0	1	2
6	0	2	3
7	1	3	4
8	2	5	6
9	3	6	8
10	5	8	10
11	7	10	13
12	10	13	17
13	13	17	21
14	16	21	26
15	20	25	30
16	25	29	36
17	29	34	41
18	34	40	47
19	40	46	53
20	46	52	60
21	52	58	67
22	58	65	75
23	65	73	83
24	73	81	91
25	81	89	100

Tabla 9 Valores críticos para la prueba de rango con signos

En la comparación entre las heurísticas mediante la prueba de rangos con signo de Wilcoxon, se consideró un total de 24 resultados $n = 24$, obteniéndose los valores de $W+= 296$, $W- = 0$. De acuerdo con la definición del estadístico, $W = min(W+, W-)$, por lo que en este caso $W = 0$. Al contrastar este resultado con el valor crítico tabulado $Wc = 81$ obtenido de la Tabla 8 de valores críticos para la prueba de rango con signos para $n = 24$ y un nivel de significancia de α=0.05, se observa que el valor calculado es menor que el valor crítico. Por consiguiente, Tabla 10 comparación EVVFLT vs SAR se rechaza la hipótesis nula H_0, concluyendo que existen diferencias estadísticamente significativas

entre las heurísticas analizadas, lo que indica que una de ellas presenta un mejor desempeño respecto a la otra en la comparación realizada.

Instancia	Tamaño	EVVFLT	SAR	Di	|Di|	Ri
abz5	10x10	34	34	0	0	6.5
abz6	10x10	34	34	0	0	6.5
ft06	6X6	34	34	0	0	6.5
ft10	10x10	34	34	0	0	6.5
la16	10x10	34	34	0	0	6.5
la17	10x10	34	34	0	0	6.5
la18	10x10	34	34	0	0	6.5
la19	10x10	34	34	0	0	6.5
la20	10x10	34	34	0	0	6.5
orb01	10x10	120	34	86	86	24
orb02	10x10	34	34	0	0	6.5
orb03	10x10	110	34	76	76	20
orb04	10x10	84.5	34	50.5	50.5	16
orb05	10x10	78	34	44	44	14
orb06	10x10	83	34	49	49	15
orb07	10x10	34	34	0	0	6.5
orb08	10x10	112	34	78	78	22
orb09	10x10	111	34	77	77	21
orb10	10x10	34	34	0	0	6.5
la36	15X15	105	34	71	71	17
la37	15X15	101	34	67	67	16
la38	15X15	119	34	85	85	23
la39	15X15	100	34	66	66	17
la40	15X15	96	74	22	22	13

Tabla 10 Comparación EVVFLT vs SAR

Prueba de rangos con signo de Wilcoxon Tabla 11 comparación EVVFLT vs SysLS

- Número de observaciones: $n = 24$
- Suma de rangos positivos: $W+ = 195$
- Suma de rangos negativos: $W- = 115$
- Estadístico calculado: $W = min(W+,W-)$

- Valor crítico *(α = 0.05): Wc = 81*
- Comparación: *W=115 > Wc = 81*
- Decisión: No se rechaza la hipótesis nula (H_0)
- Conclusión: No existen diferencias estadísticamente significativas entre las heurísticas EVVFLT y SysLS; ambas muestran un comportamiento de desempeño similar en las instancias analizadas. Esto puede deberse al hecho de que son muy pocas instancias y de tamaño chico y mediano, en los cuales la probabilidad de obtener un buen resultado es muy alta.

Instancia	Tamaño	EVVFLT	SysLS	D_i	$\|D_i\|$	R_i
abz5	10x10	34	74	-40	40	18.00
abz6	10x10	34	34	0	0	5.00
ft06	6X6	34	34	0	0	5.00
ft10	10x10	34	93	-59	59	22.00
la16	10x10	34	34	0	0	5.00
la17	10x10	34	34	0	0	5.00
la18	10x10	34	34	0	0	5.00
la19	10x10	34	34	0	0	5.00
la20	10x10	34	81.5	-47.5	47.5	20.00
orb01	10x10	120	69	51	51	21.00
orb02	10x10	34	71	-37	37	17.00
orb03	10x10	110	113	-3	3	11.00
orb04	10x10	84.5	84.5	0	0	5.00
orb05	10x10	78	34	44	44	19.00
orb06	10x10	83	79	4	4	12.00
orb07	10x10	34	34	0	0	5.00
orb08	10x10	112	34	78	78	24.00
orb09	10x10	111	34	77	77	23.00
orb10	10x10	34	34	0	0	5.00
la36	15X15	105	81.5	23.5	23.5	16.00
la37	15X15	101	98.5	2.5	2.5	10.00
la38	15X15	119	98.5	20.5	20.5	15.00
la39	15X15	100	108	-8	8	13.00
la40	15X15	96	109	-13	13	14.00

Tabla 11 Comparación EVVFLT vs SysLS

Tabla 12. Comparación: EVVFLT vs GBA

- Número de observaciones: $n = 18$
- Suma de rangos positivos: $W+ = 171$
- Suma de rangos negativos: $W - = 1$
- Estadístico calculado: $W = min(W+, W-)$
- Valor crítico $(\alpha = 0.05)$: $Wc = 40$
- Comparación: $W = 1 < Wc = 40$
- Decisión: Se rechaza la hipótesis nula (H_0)
- Conclusión: Existen diferencias estadísticamente significativas entre las heurísticas EVVFLT y GBA, lo que indica que presentan comportamientos distintos en su desempeño sobre las instancias evaluadas.

Instancia	Tamaño	EVVFLT	GBA	D_i	$\|D_i\|$	R_i
abz5	10x10	34	104	-70	70	13
abz6	10x10	34	107	-73	73	14
ft06	6X6	34	34	0	0	1
ft10	10x10	34	124	-90	90	16
la16	10x10	34	88	-54	54	10
la17	10x10	34	68	-34	34	6
la18	10x10	34	97	-63	63	11
la19	10x10	34	125	-91	91	17
la20	10x10	34	103	-69	69	12
orb01	10x10	120	132	-12	12	3
orb02	10x10	34	122	-88	88	15
orb03	10x10	110	131	-21	21	5
orb04	10x10	84.5	123	-38.5	38.5	7
orb05	10x10	78	127	-49	49	9
orb06	10x10	83	128	-45	45	8
orb07	10x10	34	126	-92	92	18
orb08	10x10	112	130	-18	18	4
orb09	10x10	111	118	-7	7	2

Tabla 12 Comparación EVVFLT vs GBA

La siguiente Tabla 13 Comparación EVVFLT vs TB presenta los resultados obtenidos en la comparación entre dos heurísticas.

Datos utilizados:

- $n = 18$, número de observaciones analizadas.
- $W^- = 50.5$ suma de los rangos con signo negativo.
- $W^+ = 120.5$ suma de los rangos con signo positivos.
- $W = 50.5$ estadístico de prueba
- $Wc = 40$ valor crítico para $n = 18$.

Interpretación:

- Dado que $W > Wc$, no se rechaza la hipótesis nula.
- Esto implica que no existen diferencias estadísticamente significativas entre las heurísticas comparadas.
- Por tanto, el desempeño de ambas heurísticas puede considerarse equivalente en las condiciones del experimento.

Instancia		EVVFLT	TB	D_i	$\|D_i\|$	R_i
abz5	10x10	34	74	-40	40	16.5
abz6	10x10	34	34	0	0	4
ft06	6X6	34	34	0	0	4
la16	10x10	34	34	0	0	4
la17	10x10	34	34	0	0	4
la18	10x10	34	34	0	0	4
la19	10x10	34	34	0	0	4
la20	10x10	34	34	0	0	4
orb01	10x10	120	80	40	40	16.5
orb02	10x10	34	71	-37	37	15
orb03	10x10	110	102	8	8	11
orb04	10x10	84.5	90	-5.5	5.5	10
orb05	10x10	78	76.5	1.5	1.5	8
la36	15X15	105	89	16	16	14
la37	15X15	101	92	9	9	12
la38	15X15	119	121	-2	2	9
la39	15X15	100	87	13	13	13
la40	15X15	96	34	62	62	18

Tabla 13 Comparación EVVFLT vs TB

Tabla 14 Comparación EVVFLT vs mXLSGA

Datos utilizados:

- $n = 24$ número de observaciones analizadas.
- $W^- = 135.5$ suma de los rangos con signo negativo.
- $W^+ = 164.5$ suma de los rangos con signo positivo.
- $W = 135.5$ estadístico de prueba, determinado como $W=min(W+,W-)$
- $Wc = 81$ valor crítico correspondiente a $n = 24$.

Interpretación:

- Dado que $W > Wc$, no se rechaza la hipótesis nula.

Instancia	Tamaño	EVVFLT	mXLSGA	D_i	$\|D_i\|$	R_i
abz5	10x10	34	34	0	0	6
abz6	10x10	34	34	0	0	6
ft06	6X6	34	34	0	0	6
ft10	10x10	34	34	0	0	6
la16	10x10	34	34	0	0	6
la17	10x10	34	34	0	0	6
la18	10x10	34	34	0	0	6
la19	10x10	34	34	0	0	6
la20	10x10	34	34	0	0	6
orb01	10x10	120	91	29	29	22
orb02	10x10	34	71	-37	37	23
orb03	10x10	110	115	-5	5	13.5
orb04	10x10	84.5	34	51	50.5	24
orb05	10x10	78	77	2	1.5	12
orb06	10x10	83	95	-12	12	16.5
orb07	10x10	34	34	0	0	6
orb08	10x10	112	94	18	18	19.5
orb09	10x10	111	86	25	25	21
orb10	10x10	34	34	0	0	6
la36	15X15	105	117	-12	12	16.5
la37	15X15	101	106	-5	5	13.5
la38	15X15	119	129	-10	10	15
la39	15X15	100	116	-16	16	18
la40	15X15	96	114	-18	18	19.5

Tabla 14 Comparación EVVFLT vs mXLSGA

Los resultados del análisis estadístico generaron diferencias en los resultados obtenidos por cada una de las seis heurísticas analizadas; sin embargo, desde un punto de vista estadístico, únicamente dos de ellas muestran diferencias significativas en su desempeño. Esto sugiere que, aunque los valores observados varían entre heurísticas, la mayoría presentan un comportamiento similar bajo las mismas condiciones de prueba. Además, la distribución de los datos, representada en la gráfica de caja y bigotes, permite visualizar la dispersión y tendencia central de los resultados, evidenciando las variaciones entre heurísticas y confirmando la presencia de algunas diferencias puntuales en los algoritmos con mejor desempeño. Además, debido al tamaño de las instancias todas medianas y chicas no permite ver una diferencia notable en las comparaciones.

5 CONCLUSIONES Y TRABAJOS FUTUROS

5.1 Conclusión

El algoritmo EVVFLT demostró ser eficiente, obteniendo soluciones de buena calidad en instancias de distintos tamaños. La selección de la mejor solución tras 30 ejecuciones por instancia permitió evaluar su efectividad frente a la aleatoriedad de la solución inicial y la complejidad del problema. Aunque el error relativo tiende a aumentar ligeramente en instancias medianas y grandes, el incremento no es significativo, lo que evidencia la robustez del método.

Durante el proceso de optimización, EVVFLT puede romper explícitamente varias rutas críticas cuando existen en la calendarización, lo que contribuye a explorar diferentes configuraciones y a diversificar las soluciones, también por medio del método se limita a un tamaño de vecindad reducido. No obstante, presenta la limitación de que, si una operación pertenece a la ruta crítica y otra no, aunque sean adyacentes, el algoritmo las ignora, incrementando el número de intentos fallidos y reduciendo la posibilidad de mejora en esos casos y consumiendo tiempo y forzar una detención sin antes haber explorado lo suficiente.

Además, la generación de ciclos en la lista de operaciones puede alejar algunas soluciones de sus vecinos más cercanos, dificultando la convergencia hacia óptimos locales o globales. En conjunto, EVVFLT se presenta como una alternativa confiable para generar soluciones factibles y de calidad en problemas de calendarización complejos, aunque con limitaciones asociadas a la gestión de rutas críticas y corrección de ciclos.

5.2 Trabajos Futuros

Como continuación de esta investigación, se proponen varias líneas de trabajo futuro que podrían mejorar la eficiencia y robustez del algoritmo EVVFLT en problemas de calendarización:

1. Identificación y contabilización de múltiples rutas críticas: Se sugiere desarrollar métodos que permitan detectar y registrar todas las rutas críticas presentes en la calendarización, no solo la principal. Esto permitiría una gestión más precisa de las operaciones que afectan directamente el tiempo total del proyecto.
2. Optimización de bifurcaciones en las rutas críticas: Implementar estrategias para mover operaciones que inicien una bifurcación en las rutas críticas, con el objetivo

de explorar configuraciones alternativas que podrían reducir el tiempo de finalización global.

3. Ampliación de las estructuras de vecindad: Incorporar nuevas estructuras de vecindad para generar permutaciones más diversas y efectivas en bloques, mejorando la capacidad del algoritmo para escapar de óptimos locales y explorar de manera más profunda el espacio de soluciones.
4. Uso de calendarización sin ciclos: Desarrollar mecanismos que aseguren la generación de calendarizaciones libres de ciclos, garantizando soluciones factibles y consistentes en cada iteración del algoritmo.

APENDICE A

1.Introducción a la complejidad temporal

La complejidad temporal mide cuánto tiempo tarda un algoritmo en ejecutarse en función del tamaño de los datos de entrada. No indica el tiempo exacto en segundos, sino cómo crece el tiempo de ejecución cuando aumenta la cantidad de datos, lo que permite evaluar la eficiencia y escalabilidad de un algoritmo.

Se analiza generalmente en tres casos: mejor caso, caso promedio y peor caso, y se representa mediante notaciones como *O, Ω y Θ*, que describen el crecimiento relativo del tiempo en función del tamaño de la entrada. Esta información es clave para comparar algoritmos y seleccionar el más eficiente para resolver un problema específico.

En resumen, la complejidad temporal es una herramienta fundamental en el diseño y optimización de algoritmos, ya que permite anticipar el comportamiento de un procedimiento y tomar decisiones informadas para mejorar el rendimiento de sistemas computacionales.

2.Complejidad temporal de los procedimientos utilizados

El procedimiento corregirPrecedencia recorre todas las posiciones de la lista de precedencias mediante un ciclo externo y, para cada posición, recorre todas las posiciones siguientes mediante un ciclo interno, comparando y corrigiendo elementos que pertenecen a la misma fila. Cada operación dentro del ciclo interno, como el cálculo de la fila o la posible permutación de elementos, requiere un tiempo constante. El número total de iteraciones del ciclo interno para todas las posiciones se puede expresar como:

Tiempo_ciclo_interno = suma desde $i = 1$ hasta N de $(N - i)$

donde ***N*** es el número total de operaciones, es decir, $N = n * m$, siendo n el número de trabajos y m el número de máquinas. Por lo tanto, la función temporal de este procedimiento se puede escribir como:

Tiempo_corregirPrecedencia = $c1 * N + c2 * (N * (N - 1)) / 2$

y considerando solo el término dominante:

Tiempo_corregirPrecedencia ≈ $(n * m)^2$

El procedimiento calcularCPM recorre las precedencias en orden inverso mediante un ciclo externo y, cuando se encuentra una operación crítica, ejecuta un ciclo interno para actualizar los tiempos de las máquinas correspondientes. El coste de este ciclo interno

depende del número de operaciones a revisar por máquina o por trabajo; en general puede expresarse como K. De este modo, su función temporal se puede expresar como:

Tiempo_calcularCPM = $\Theta(N * K)$

donde $N = n * m$. En particular, si el coste del ciclo interno es proporcional al número de operaciones de cada máquina ($K = n$), se obtiene:

Tiempo_calcularCPM = $\Theta(n2 * m)$

Al ejecutar ambos procedimientos de manera consecutiva, el tiempo total requerido se puede expresar como:

Tiempo_total = Tiempo_corregirPrecedencia + Tiempo_calcularCPM

Sustituyendo los términos dominantes, se obtiene:

Tiempo_total = $(n * m)^2 + n^2 * m$

Complejidad temporal del procedimiento calcularCPM según el coste del ciclo interno

- Caso 1: Ciclo interno constante
  El ciclo interno revisa un número fijo de operaciones por cada operación.
  Complejidad: $O(N)$
- Caso 2: Ciclo interno proporcional al número de operaciones de la misma máquina
  Cada máquina tiene aproximadamente n operaciones (uno por trabajo).
  Complejidad: $O(N * n) = O(n^2 * m)$
- Caso 3: Ciclo interno proporcional al total de operaciones
  En el escenario extremo en el que cada ciclo interno recorre todas las operaciones.
  Complejidad: $O(N^2) = O((n * m)^2)$

Por lo tanto, siempre que $K = O(N)$, el término dominante es $(n * m)^2$, lo que indica que el tiempo total de ejecución de ambos procedimientos crece aproximadamente como el cuadrado del producto del número de trabajos por el número de máquinas. Este análisis se mantiene para el mejor, promedio y peor caso.

REFERENCIAS

1. **Johnson, S. M (1954).** Optimal two- and three-stage production schedules with setup Times Included. Naval Research Logistics Quarterly, 1(1), 61–68. doi:10.12/nav.3800010110
2. **Kirkpatrick,et al. (1983).** Optimization by Simulated Annealing. *Science, 220*(4598), 671–680.https://doi.org/10.1126/science.220.4598.671
3. **Or-Library Beasley, J. E. (1990).** Or-Library is a collection of test data sets for a variety of Operations Research (OR) problems. Recuperado el 22 de abril de 2023, de Brunel.ac.uk
4. **Festa, P. (2014)**. A brief introduction to exact, approximation, and heuristic algorithms for solving hard combinatorial optimization problems. 2014 16th International 48
5. **Manne, A.S. (1960)** On the Job-Shop Scheduling Problem. Operations Research, 8, 219-223.http://dx.doi.org/10.1287/opre.8.2.219
6. **Jorapur, V. , Puranik, V. , Deshpande, A. and Sharma, M. (2016)** A Promising Initial Population Based Genetic Algorithm for Job Shop Scheduling Problem. Journal of Software Engineering and Applications, 208214. doi: 10.4236/jsea.2016.95017.
7. **Papadimitriou C. H., Steiglitz.(1998).”** Combinatorial Optmization Algorithms and Complexity”. ISBN. 0-486-40258-4. USA. 1998
8. **Cruz-Chávez, M. A. (2014).** Neighborhood generation mechanism applied in simulated annealing to job shop scheduling problems. International Journal of Systems Science, 46(15), 2673–2685. doi:10.1080/00207721.2013.876679
9. **Levy, F. K., Thompson, G. L., & Wiest, J. D. (1963, September 1).** The ABCs of the Critical Path Method. Harvard Business Review. Retrieved from https://hbr.org/1963/09/the-abcs-of-the-critical-path-method
10. **Satake, T., Morikawa, K., Takahashi, K., & Nakamura, N. (1999)**. Simulated annealing approach for minimizing the makespan of the general job-shop. International Journal of Production Economics, 60-61, 515–522. doi:10.1016/s0925-5273(98)00171-6
11. **Ying, K. C., & Lin, S. W. (2020)**. Solving no-wait job-shop scheduling problems using a multistart simulated annealing with bi-directional shift timetabling algorithm. Computers & Industrial Engineering, 146, 106615. doi:10.1016/j.cie.2020.106615
12. **Giffler, B., & Thompson, G. L. (1960).** *Algorithms for Solving Production-Scheduling Problems. Operations Research, 8(4), 487–503.* doi:10.1287/opre.8.4.487
13. **Kelley, J. E. (1961).** *Critical-Path Planning and Scheduling: Mathematical Basis. Operations Research, 9(3), 296–320.* doi:10.1287/opre.9.3.296
14. **RM Van Slyke (1963)** Monte Carlo methods and the PERT problem Operations Research.
15. **Heuser, W. A., & Wynne, B. E. (1963)** An Application of the Critical Path Method to Job Shop Scheduling—A Case Study. Management Science, MT-3(2), 128–144. doi:10.1287/mantech.3.2.128
16. **Davis, E. W., & Patterson, J. H. (1975).** A Comparison of Heuristic and Optimum Solutions in Resource-Constrained Project Scheduling. Management Science, 21(8), 944–955. doi:10.1287/mnsc.21.8.944
17. **Garey, M. R. DS Johnson (1979)** Computers and Intractability: A Guide to the Theory of NP-Completeness-Freedman, 1979. San Francisco, ***CA*.**

18. **E.H.L Laarhoven. Aarts et al (1989)** Simulated annealing: An introduction. Statistica Neerlandica 43, nr. 1.
19. **Applegate, D., & Cook, W. (1991).** A computational study of the job-shop scheduling problem. ORSA Journal on Computing, 3(2), 149–156. https://doi.org/10.1287/ijoc.3.2.149
20. **Alexander, C. A. Jr. (1995).** Near-critical path algorithms for program activity graphs. *ACM Transactions on Mathematical Software, 21*(4), 486–494. https://doi.org/10.1145/216900.216901
21. **Jain, A. S., & Meeran, S. (1999).** Deterministic job-shop scheduling: Past, present and future. European Journal of Operational Research, 113(2), 390–434. doi:10.1016/s0377-2217(98)00113-1
22. **Singh, G., & Zinder, Y. (2000).** Worst-case performance of critical path type algorithms. International Transactions in Operational Research, 7(4-5), 383–399. doi:10.1111/j.1475-3995.2000.tb00206.x
23. **Chanas, S., & Zieliński, P. (2001**) *Critical path analysis in the network with fuzzy activity times. Fuzzy Sets and Systems, 122(2), 195–204.* doi:10.1016/s0165-0114(00)00076-2
24. **Shyi-Ming Chen, & Tao-Hsing Chang. (2001)** Finding multiple possible critical paths using fuzzy PERT. IEEE Transactions on Systems, Man and Cybernetics, Part B (Cybernetics), 31(6), 930–937. doi:10.1109/3477.969496
25. **Steinhöfel, K., Albrecht, A., & Wong, C. K. (2003).** *An experimental analysis of local minima to improve neighbourhood search. Computers & Operations Research, 30(14), 2157– 2173. doi:10.1016/s0305-0548(02)00128-4*
26. **Cruz-Chávez, M. A., & Frausto-Solís, J. (2006**). A New Algorithm That Obtains an Approximation of the Critical Path in the Job Shop Scheduling Problem. MICAI 2006: Advances in Artificial Intelligence, 450–460. doi:10.1007/11925231_42
27. **Gaurav Singh CSIRO (2008**) Performance of Critical Path Type Algorithms with Communication Delay Mathematical and Information sciences, Private Bag 10, South Clayton, VIC 3169, Australia ing. Author's email: Gaurav.Singh@csiro.au
28. **N. Ravi Shankar et al (2010)** Critical Path Method in a Project Network using Ant Colony Optimization. International Journal of Computational Intelligence Research ISSN 0973-1873 Volume 7, Number 1 (2011), pp. 7–16 © Research India Publications http://www.ripublication.com/ijcir. Department of Applied Mathematics, GIS, GITAM University, Visakhapatnam, INDIA.
29. **Cruz-Chávez, M. A. (2014**) Neighbourhood generation mechanism applied in simulated annealing to job shop scheduling problems. International Journal of Systems Science, 46(15), 2673–2685. doi:10.1080/00207721.2013.876679
30. **Kruskal, W. H., & Wallis, W. A. (1952)**. Use of ranks in one-criterion variance analysis. *Journal of the American Statistical Association*, 47(260), 583-621. http://doi.org/10.1080/01621459.1952.10483441
31. **Majid Abdolrazzagh et al (2017)** Job Shop Scheduling: Classification, Constraints and Objective Functions. International Science Index, Computer and Information Engineering Vol:11, No:4, 2017 waset.org/Publication/10006691
32. **Sun, Z.-X., et al (2018)** Salp Swarm Algorithm Based on Blocks on Critical Path for Reentrant Job Shop Scheduling Problems. Lecture Notes in Computer Science, 638–648. doi:10.1007/978-3-319-95930-6_64

33. **Hu, J., Zhou et al (2018)**. Interconnect Optimization Considering Multiple Critical Paths. Proceedings of the 2018 International Symposium on Physical Design - ISPD '18. doi:10.1145/3177540.3178237

34. **Goto, H., Murray, A.T. (2019)** Small-m method for detecting all longest paths. *OPSEARCH* 56, 824–839 https://doi.org/10.1007/s12597-019-00385-0
35. **Mohamed Kurdi (2019)** An effective genetic algorithm with a critical-path-guided Giffler and Thompson crossover operator for job shop scheduling problem. International Journal of Intelligent Systems and Applications in Engineering. Corresponding AuthorEmail: mohamed.kurdi@adu.edu.tr
36. **Yasser M. R. Aboelmagd (2020)** Smart Critical Path Method As A Modified Detailed Scheduling Technique Civil Engineering Department, College of Engineering, University of Business and Technology, UBT, Jeddah, Saudi Arabia DOI: 10.21608/jesaun.2020.107365
37. **Alper Hamzadayı et al (2020)** Solving combinatorial optimization problems with single seekers society algorithm. Knowledge-Based Systems 201–202 (2020)Doi.org/10.1016/j.knosys.2020.106036.
38. **Kyunghwan Kim (2020)** Generalized Resource-Constrained Critical Path Method to Improve Sustainability in Construction Project Scheduling Kyunghwan Kim Department of Architecture, Konkuk University, 120, Neungdong-ro, Gwangjin-gu, Seoul 05029, Korea; kykim@konkuk.ac.kr

39. **Fligner, M. A. (1985).** *Pairwise versus joint ranking: Another look at the Kruskal–Wallis statistic*. Biometrika, 72(3), 705-709. https://doi.org/10.1093/biomet/72.3.705

40. **Orumie Ukamaka Cynthia (2020)** Implementation of Project Evaluation and Review Technique (PERT) and Critical Path Method (CPM): A Comparative Study. Department of Mathematics/Statistics, University of Port Harcourt, Nigeria. doi: 10.35840/2633-8947/6504
41. **Jiaxin Fan et al (2021)** A hybrid Jaya algorithm for solving flexible job shop scheduling problem considering multiple critical paths. Journal of Manufacturing Systems journal homepage: www.elsevier.com/locate/jmansys https://doi.org/10.1016/j.jmsy.2021.05.018
42. **Prasanna Venkatesan Ramani, et al (2022).** Application of linear scheduling in water canal construction with a comparison of critical path method. Journal of Construction in Developing Countries, 27(1): 189–212. https://doi.org/10.21315/jcdc2022.27.1.11
43. **Deng, J.; Jian, W. (2022)** Estimating Construction Project Duration and Costs upon Completion Using Monte Carlo Simulations and Improved Earned Value Management. Buildings 2022, 12, 2173. https://doi.org/ 10.3390/buildings12122173
44. **Oladimeji O.A et al (2023),** Network Analysis on Phase 1B Building of the University Health Centre, FUTA, using Critical Path Method (CPM) and Program Evaluation Review Technique (PERT). International Journal of Mechanical and Civil Engineering 6(1), 40-52. DOI: 10.52589/IJMCEGCEQS8DC
45. **Yin Fei Tan, et at (2023**) "The critical path adjustment mechanism for dynamic job shop scheduling problem", Proc. SPIE 12709, Fourth International Conference on Artificial Intelligence and Electromechanical Automation (AIEA 2023), 127096D (19 October 2023); https://doi.org/10.1117/12.2684702

46. **Q. Meng, Et al, (2023) "**Two-Stage Assembly Flow Shop Scheduling Problem with Sequence-Dependent Setup Times," *2023 IEEE 19th International Conference on Automation Science and Engineering (CASE)*, Auckland, New Zealand, 2023, pp. 1-6, doi: 10.1109/CASE56687.2023.10260629. keywords: Sequential analysis;Job shop scheduling; Computer aided software engineering; Automation; Perturbation methods; Heuristic algorithms; Social sciences.
47. **Hu K, Wang L Et al (2023)** An improved genetic algorithm with dynamic neighborhood search for job shop scheduling problem. Math Biosci Eng. 2023 Sep 11;20(9):17407-17427. doi: 10.3934/mbe.2023774. PMID: 37920060.
48. **Yin Fei Tan Et at (2023)** "The critical path adjustment mechanism for dynamic job shop scheduling problem", Proc. SPIE 12709, Fourth International Conference on Artificial Intelligence and Electromechanical Automation (AIEA 2023), 127096D (19 October 2023); https://doi.org/10.1117/12.2684702
49. **Eric Bourreau (2024)** Indirect Job-Shop coding using rank: application to QAOA (IQAOA) https://doi.org/10.48550/arXiv.2402.18280
50. **Zhou, H., Liu, H., Lv, C., Zhang, C., & Shen, W. (2024**). A path relinking with tabu search algorithm for solving hybrid flow shop scheduling problem considering multiple critical paths. Computers & Operations Research, 170, 106783. https://doi.org/10.1016/j.cor.2024.106783
51. **Wang, G (2024)** A Self-Adaptive Memetic Algorithm for Distributed Job Shop Scheduling Problem. Mathematics 2024, 12, 683. https://doi.org/10.3390/math12050683
52. ***Duan L.(2006)*** *Applying systematic local search to job shop scheduling problems* (Tesis de maestría). Simon Fraser University. Recuperado de https://summit.sfu.ca/_flysystem/fedora/sfu_migrate/3952/etd2481.pdf
53. **Shichen Tian, Et al(2024)** A genetic algorithm with critical path-based variable neighborhood search for distributed assembly job shop scheduling problem, Swarm and Evolutionary Computation, Volume 85, 2024, 101485, ISSN 2210-6502,https://doi.org/10.1016/j.swevo.2024.101485. (https://www.sciencedirect.com/science/article/pii/S221065022400018X)
54. **Project Management Institute (2017)**. *A Guide to the Project Management Body of Knowledge (PMBOK® Guide) – Sixth Edition.* https://www.pmi.org/standards/pmbok
55. **Sayoti, F Et al (2016)** Optimization of makespan in job shop scheduling problem by Golden Ball Algorithm. *Indonesian Journal of Electrical Engineering and Computer Science, 4*(3), 542–547. https://doi.org/10.11591/ijeecs.v4.i3.pp542-547
56. **Viana, M. S. Et al (2020)** *An improved local search genetic algorithm with multi-crossover for job shop scheduling problem.* Federal University of São Carlos & University of São Paulo, Brazil.

57. **Mahmud, S., Et al (2022).** Switching strategy-based hybrid evolutionary algorithms for job shop scheduling problems. Journal of Intelligent Manufacturing, 33, 1939–1966. https://doi.org/10.1007/s10845-022-01940-1

58. **Kruskal & Wallis (1952).** Use of ranks in one-criterion variance analysis. *Journal of the American Statistical Association*, 47(260), 583–621. https://doi.org/10.1080/01621459.1952.10483441

Cuernavaca, Morelos a 16 de abril del 2026.

MTRO. ELMER IVAN SANCHEZ RABADAN
ENCARGADO DE DESPACHO DE LA DIRECCION DE LA FACULTAD
DE CONTADURIA, ADMINISTRACION E INFORMATICA DE LA UAEM.
PRESENTE.

En mi carácter de revisor de tesis, hago de su conocimiento que he leído con interés la tesis para obtener el grado de la Maestría en Optimización y Cómputo Aplicado, del estudiante Pedro Bello Campusano, con matrícula 10003388, con el título **Diseño de una Estructura de Vecindad Variable en Función de su Línea de Tiempo Para el Problema de Talleres de Manufactura,** por lo cual, me permito informarle que después de una revisión cuidadosa de dicha tesis, me permito expresar mi **VOTO APROBATORIO** por lo que de mi parte no existe inconveniente para que el estudiante continúe con los trámites que la Universidad Autónoma del Estado de Morelos tenga establecidos para obtener el grado mencionado.

Atentamente
Por una humanidad culta

Dr. Federico Alonso Pecina
Profesor-Investigador de Tiempo Completo
Facultad de Contaduría, Administración e Informática

UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MORELOS

Se expide el presente documento con firma electrónica UAEM, soportada por el certificado vigente a la fecha de su elaboración y con efectos plenos de conformidad con los LINEAMIENTOS EN MATERIA DE FIRMA ELECTRÓNICA PARA LA UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MORELOS PUBLICADOS en el ÓRGANÓ INFORMATIVO UNIVERSITARIO "ADOLFO MENÉNDEZ SAMARÁ" número 117 de fecha 20 de abril de 2021.

Sello electrónico

FEDERICO ALONSO PECINA | Fecha:2026-04-16 14:31:38 | FIRMANTE

7X8KuTf4CUAjR8moBfUkgh1AAYbvZEE1A1+4sSB8VDBeuoFjikHhvqmu3fpZgPX/A6CsPddC7NJJzgDfdMtBSNz9ajQ5i6sDLdNQQgJyLT2toYtEErVsoIV+WCmn/bk3D4XEqWGidIXoPsHJajdBQFF8KwaJPD2o/X9t1/o6F6wrcHDPyjSCnUqQAPgP4kWAyT2sehjaHsL1kEs8tXqvzxOinqpB8taW7dH7VtQ378ICMY7Hp2itWuWKNjY+8wz2Ot4/0M2zbxcB6llRq2aBDdZs1LB9aHHTQoWkMRU2IJbKXwbUcM8TO4ohaTBlH1Hxcr5lzm7qRTgVTIYZ/60JUw==

Puede verificar la autenticidad del documento en la siguiente dirección electrónica o escaneando el código QR ingresando la siguiente clave:

QYZnREqtk

https://efirma.uaem.mx/noRepudio/oxmU3p7dTvq9TLJZAB4gyKDsergDBofu

Cuernavaca, Morelos a 13 de abril del 2026.

MTRO. ELMER IVAN SANCHEZ RABADAN
ENCARGADO DE DESPACHO DE LA DIRECCION DE LA FACULTAD
DE CONTADURIA, ADMINISTRACION E INFORMATICA DE LA UAEM.
PRESENTE.

En mi carácter de revisor de tesis, hago de su conocimiento que he leído con interés la tesis para obtener el grado de la Maestría en Optimización y Cómputo Aplicado, del estudiante Pedro Bello Campusano, con matrícula 10003388, con el título **Diseño de una Estructura de Vecindad Variable en Función de su Línea de Tiempo Para el Problema de Talleres de Manufactura,** por lo cual, me permito informarle que después de una revisión cuidadosa de dicha tesis, me permito expresar mi **VOTO APROBATORIO** por lo que de mi parte no existe inconveniente para que el estudiante continúe con los trámites que la Universidad Autónoma del Estado de Morelos tenga establecidos para obtener el grado mencionado.

Atentamente
Por una humanidad culta

Dra. Jesús del Carmen Peralta Abarca
Profesora- investigadora
Facultad de Contaduría, Administración e Informática

UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MORELOS

Se expide el presente documento con firma electrónica UAEM, soportada por el certificado vigente a la fecha de su elaboración y con efectos plenos de conformidad con los LINEAMIENTOS EN MATERIA DE FIRMA ELECTRÓNICA PARA LA UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MORELOS PUBLICADOS en el ÓRGANÓ INFORMATIVO UNIVERSITARIO "ADOLFO MENÉNDEZ SAMARÁ" número 117 de fecha 20 de abril de 2021.

Sello electrónico

JESUS DEL CARMEN PERALTA ABARCA | Fecha:2026-04-16 16:10:27 | FIRMANTE

di8r2PxdxYREagYUGl2hA2Oe+d7CK4QnqiR6G3YR53KHKECwPMB82LoJ2/mVifFgmljW0Zzo935BL6EjWrG5kSlfPRg3PVydrDYZT3MHnNPHBk9zTQSIxBL1Iy7qElkWUl0sYIm9V+mKv9SUGAMtUTLLh+AzmrzIvdkh4gouwiIfjeiWH/HpbRTs6UrtL4Y0tDGK46aT1zA98dDy5rtuaT59UKRtuKtxguUmqodw9L3XELWDRVUR6BqnE1lEyu5xrRJxc/JDq05EIa3JtwGUTw1DxoDsyTMN5dHuEcmJEcnawwhM3OcEztEzje/onu6sJw8yxfztzCuA72Q5RY92hw==

Puede verificar la autenticidad del documento en la siguiente dirección electrónica o escaneando el código QR ingresando la siguiente clave:

dzF7rZ906

https://efirma.uaem.mx/noRepudio/I2kKbeS6j4TlxSo1bmdR2uInaJoZED2c

Cuernavaca, Morelos a 13 de abril del 2026.

MTRO. ELMER IVAN SANCHEZ RABADAN
ENCARGADO DE DESPACHO DE LA DIRECCION DE LA FACULTAD
DE CONTADURIA, ADMINISTRACION E INFORMATICA DE LA UAEM.
PRESENTE.

En mi carácter de revisor de tesis, hago de su conocimiento que he leído con interés la tesis para obtener el grado de la Maestría en Optimización y Cómputo Aplicado, del estudiante Pedro Bello Campusano, con matrícula 10003388, con el título **Diseño de una Estructura de Vecindad Variable en Función de su Línea de Tiempo Para el Problema de Talleres de Manufactura,** por lo cual, me permito informarle que después de una revisión cuidadosa de dicha tesis, me permito expresar mi **VOTO APROBATORIO** por lo que de mi parte no existe inconveniente para que el estudiante continúe con los trámites que la Universidad Autónoma del Estado de Morelos tenga establecidos para obtener el grado mencionado.

Atentamente
Por una humanidad culta

Dr. Rafael Rivera López
Profesor- investigador
Departamento de Sistemas y Computación
Instituto Tecnológico de Veracruz

UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MORELOS

Se expide el presente documento con firma electrónica UAEM, soportada por el certificado vigente a la fecha de su elaboración y con efectos plenos de conformidad con los LINEAMIENTOS EN MATERIA DE FIRMA ELECTRÓNICA PARA LA UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MORELOS PUBLICADOS en el ÓRGANÓ INFORMATIVO UNIVERSITARIO "ADOLFO MENÉNDEZ SAMARÁ" número 117 de fecha 20 de abril de 2021.

Sello electrónico

RAFAEL RIVERA LÓPEZ | Fecha:2026-04-17 05:47:59 | FIRMANTE

xUsNuznTm20iTtkr6r68bzr99bFZasFuYXTjHVkXEzoIN53qQlUHN/M5gdLBHkSMn78glmm1GZ0Yap3noHtB8PELu3dXm3n0E2rLm+bqc4an+CDAyWmV1a50B/QaGTksO9nFV2ZSWcM9SL+eP6ccpeonRb4h1PLVU7gJrCmh7wB1x4vATSQqJOhyV6lFNuBttkKH9gjSFmzHBXMJexcwdeVrdokDozykbFC+Bj1ZFRTmzsBCZ4svj+ztekfVdAPKCMLPjPBBUUXHy1lSh+Y8JWbb5O79Odabl3WNwhthjnWr1K1iEMxWVUPBIzLflHSSYSalcgMKIQjffZnZmBjqNQ==

Puede verificar la autenticidad del documento en la siguiente dirección electrónica o escaneando el código QR ingresando la siguiente clave:

euYwv5mlB

https://efirma.uaem.mx/noRepudio/qCgPa215K1YM5O67hu08CfDUGWlLXGcJ

Cuernavaca, Morelos a 13 de abril del 2026.

MTRO. ELMER IVAN SANCHEZ RABADAN
ENCARGADO DE DESPACHO DE LA DIRECCION DE LA FACULTAD
DE CONTADURIA, ADMINISTRACION E INFORMATICA DE LA UAEM.
PRESENTE.

En mi carácter de revisor de tesis, hago de su conocimiento que he leído con interés la tesis para obtener el grado de la Maestría en Optimización y Cómputo Aplicado, del estudiante Pedro Bello Campusano, con matrícula 10003388, con el título **Diseño de una Estructura de Vecindad Variable en Función de su Línea de Tiempo Para el Problema de Talleres de Manufactura,** por lo cual, me permito informarle que después de una revisión cuidadosa de dicha tesis, me permito expresar mi **VOTO APROBATORIO** por lo que de mi parte no existe inconveniente para que el estudiante continúe con los trámites que la Universidad Autónoma del Estado de Morelos tenga establecidos para obtener el grado mencionado.

Atentamente
Por una humanidad culta

Dr. Marco Antonio Cruz Chávez
Profesor- investigador
Centro de Investigación en Ingeniería y Ciencias Aplicadas

UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MORELOS

Se expide el presente documento con firma electrónica UAEM, soportada por el certificado vigente a la fecha de su elaboración y con efectos plenos de conformidad con los LINEAMIENTOS EN MATERIA DE FIRMA ELECTRÓNICA PARA LA UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MORELOS PUBLICADOS en el ÓRGANÓ INFORMATIVO UNIVERSITARIO "ADOLFO MENÉNDEZ SAMARÁ" número 117 de fecha 20 de abril de 2021.

Sello electrónico

MARCO ANTONIO CRUZ CHAVEZ | Fecha:2026-04-20 19:10:49 | FIRMANTE

yG0RFszzhrFtCmVu/LWr+HOTRJ6dfJ1AJlqLdxUv+5DxProO+jmbDWWV8eah9PziPJw6giJznTJ2awaYDOsHIWFYN+mEA/+NW9U49SEoWLYoSLI8eVHQhLhOSWIBuPb8yqRF5Q+NqAbXGeuz+n9ZSugVnGGVBU7CDhzHIhsKMWTBNdXLmblqFCU8413q4rrpO/0f8wZ+BeqArLnrq7d1JkFsQWwyR/9OrZVN9BsqncdRjAlSpNYs4FGpC6eQArFzKtSMbwJPpTlwk6uKw+6FSsTr1UCWsC65xffuIidlGoZOCK5gWy/UD3Z+xsdz0sD/MiAAIXCt9+505+LB7NKTiA==

Puede verificar la autenticidad del documento en la siguiente dirección electrónica o escaneando el código QR ingresando la siguiente clave:

ziBSdHRDY

https://efirma.uaem.mx/noRepudio/XZy3ZVABGaeveePwkFP5hQQC15H9H2H5

Cuernavaca, Morelos a 13 de abril del 2026.

MTRO. ELMER IVAN SANCHEZ RABADAN
ENCARGADO DE DESPACHO DE LA DIRECCION DE LA FACULTAD
DE CONTADURIA, ADMINISTRACION E INFORMATICA DE LA UAEM.
PRESENTE.

En mi carácter de revisor de tesis, hago de su conocimiento que he leído con interés la tesis para obtener el grado de la Maestría en Optimización y Cómputo Aplicado, del estudiante Pedro Bello Campusano, con matrícula 10003388, con el título **Diseño de una Estructura de Vecindad Variable en Función de su Línea de Tiempo Para el Problema de Talleres de Manufactura,** por lo cual, me permito informarle que después de una revisión cuidadosa de dicha tesis, me permito expresar mi **VOTO APROBATORIO** por lo que de mi parte no existe inconveniente para que el estudiante continúe con los trámites que la Universidad Autónoma del Estado de Morelos tenga establecidos para obtener el grado mencionado.

Atentamente
Por una humanidad culta

Dr. José Alberto Hernández Aguilar
Profesor- investigador
Facultad de Contaduría, Administración e Informática

UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MORELOS

Se expide el presente documento con firma electrónica UAEM, soportada por el certificado vigente a la fecha de su elaboración y con efectos plenos de conformidad con los LINEAMIENTOS EN MATERIA DE FIRMA ELECTRÓNICA PARA LA UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MORELOS PUBLICADOS en el ÓRGANÓ INFORMATIVO UNIVERSITARIO "ADOLFO MENÉNDEZ SAMARÁ" número 117 de fecha 20 de abril de 2021.

Sello electrónico

JOSE ALBERTO HERNANDEZ AGUILAR | Fecha:2026-04-20 21:40:40 | FIRMANTE

T9AMumMpz5uBqe4ahYGpBS7DkUaOeLpzFw6mjFQI4TYzkTGbbxc5kjvTGdJUfep9P8sXttlX092ZDsVVJxnWcOEiTzru9PeKQ4gEK1cm5h+5HMAQwSsM+ZuLrqaxfB7jStCqfnh
M0mtPIGmc3mQ1r6I6ty3Kda2AAYpqwKjwTacW+33UXD5vBgP7GJq1ZxmKWEBQWKXQgIurWLK9513OjTM7G1ia0G3va66GXHTQOqpTr8JjD2VXLIuZZYm3RLUr52Attu9jXkO
Stz7/LC0Enh/OYEcA5c9ZbESyWkx+bgubZzGEMkQgyG2NrnfSiJBQ0tkVh2XlPlU9pOZlddGUUQ==

Puede verificar la autenticidad del documento en la siguiente dirección electrónica o escaneando el código QR ingresando la siguiente clave:

73RMyboUV

https://efirma.uaem.mx/noRepudio/fu9ZfDhVo86MQjcLRcmnrs9oFiRuIWeO