UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MORELOS

FACULTAD DE CONTADURÍA, ADMINISTRACIÓN E INFORMÁTICA
LICENCIATURA EN INFORMÁTICA

Evaluación de la Factibilidad de Soluciones Generadas con Estructuras de Vecindad para el Problema de Talleres de Manufactura

T E S I S

Que para obtener el Grado de LICENCADO EN INFORMÁTICA

Presenta

PEDRO BELLO CAMPUSANO

Director de Tesis
DR. MARCO ANTONIO CRUZ CHAVEZ
Co-Director
DRA. JUANA ENRÍQUEZ URBANO
Revisores:
DR. MARTÍN HERIBERTO CRUZ ROSALES
DR. FEDERICO ALONSO PECINA
DR. JOSÉ ALBERTO HERNÁNDEZ AGUILAR
DRA. MIREYA FLORES PICHARDO
DR. ULISES CRUZ JACOBO

CUERNAVACA, MORELOS JUNIO,2023

AGRADECIMIENTOS

Dr. Marco Antonio Cruz Chávez, por las enseñanzas, la disposición de trabajar, el empuje y sobre todo, por el tiempo invertido en un proyecto que, más que, una tesis para titulación es un proyecto de vida personal. Gracias por todas las horas de trabajo y por el legado, es bueno saber que existen personas que hacen su parte por mejorar un poco el mundo en que vivimos.

Dra. Juana Enríquez Urbano por la disponibilidad de hacer un esfuerzo a cambio de un solo reconocimiento, por ser un ejemplo a seguir, sus trabajos previos de investigación fueron la base y puntos clave para poder llevar a cabo este trabajo de investigación, sobre todo en la búsqueda local iterada.

Ángel Estrada y Revisores de Tesis: Dr. Martín Heriberto Cruz Rosales, Dr. Federico Alonso Pecina, Dr. José Alberto Hernández Aguilar, Dra. Mireya Flores Pichardo y Dr. Ulises Cruz Jacobo, por la coordinación que se emprendió para llevar a cabo esta tesis y la revisión lo más pronto posible. Por la actitud positiva y determinación por ayudar y apoyar a terminar este proyecto y por hacer todo lo que está a su alcance.

DEDICATORIA

A mis padres, **Hortensia Campusano**, por ensañarme lo que es el espíritu de lucha, tus consejos, tu perseverancia y hacerme entender que la vida no tiene sentido si no se lucha por lo que quieres, y que la felicidad consiste en lograr tus sueños, tus metas y que la verdadera identidad es luchar por lo que quieres ser no quien te tocó ser. Por enseñarme a vivir cada día al máximo y por hacerme entender de que la vida se vive bien si se está, bien consigo mismo.

Alberto Bello Sánchez por la enseñanza indirecta, tu partida a mis 7 años le dio un giro inesperado a mi vida, madurez temprana y aprendí a valorarte a pesar de la ausencia, aunque nunca he podido darte un abrazo físicamente. Siempre has estado en mi corazón y en mi mente. Saludos donde quiera que estes papá.

A mis hermanos **María de Jesús, Manuela, Nicolás y Miguel ángel** somos una buena familia, siempre nos hemos cuidado y apoyado los unos a los otros. Es bueno saber que en los momentos difíciles tienes alguien con quien contar, eso nos debe de motivar a seguir siendo mejores personas. Gracias por todo el apoyo hermanos.

Perla Aurora Martínez las experiencias son únicas para cada persona, todo es aprendizaje, todo tiene un por qué y estamos aquí por algo, por algo nos cruzamos en nuestros caminos, los tiempos son perfectos, me siento muy feliz con tu compañía, ahora vamos a luchar porque esto sea perpetuo.

A toda mi **familia Bello Sánchez y Campusano Ortuño,** las raíces están bien fundadas y estoy orgulloso de ser quien soy y sobre todo llevar los apellidos Bello Campusano que el prestigio nos lo hemos ganado.

RESUMEN

El problema de Calendarización para Talleres de Manufactura (Job Shop Scheduling Problem) es un problema complejo del área de optimización combinatoria, el cual se puede definir cómo un proceso de toma de decisiones y asignación de recursos para realizar operaciones en diferentes centros de trabajo en tiempo determinado y óptimo. En la teoría de la complejidad computacional, este problema está clasificado como NP-Completo, de tal forma que aún no existen métodos exactos que puedan encontrar la solución óptima en un tiempo polinomial, solo existen heurísticas y metaheurísticas acotadas en tiempo polinomial que pueden dar soluciones muy cercanas al óptimo.

En este trabajo de tesis se evalúa la factibilidad y la eficacia de una nueva estructura de vecindad definida en una línea de tiempo propuesta, partiendo de una solución aleatoria, la cual desde el inicio se hace factible a través de métodos heurísticos. El primero, es un método convencional de Nakano y Yamada (1991), el cual tiene el nombre de algoritmo de calendarización, el segundo, es un método de armonización global que permite seguir calendarizando y corrigiendo soluciones no factibles. Posteriormente, se implementa el algoritmo de búsqueda local iterada, en el cual se aplica la estructura de vecindad con línea de tiempo (EVLT), que es la estructura de vecindad propuesta y se realiza una comparación de eficacia con respecto a la estructura de vecindad clásica de permutación de pares adyacentes sin tiempos de ocio (EVPA) Cruz-Chávez (2014). Para las pruebas, se utilizan los benchmarks que se encuentran en la página de Or-Library (1990) con instancias simétricas, para realizar la comparación de los resultados obtenidos. En base a los resultados obtenidos se observa que al aplicar la EVLT existe una mejora con respecto a la EVPA; la estructura de vecindad propuesta genera resultados más rápidamente para instancias pequeñas además de que el número de soluciones no factibles obtenidas en la búsqueda local iterada es inferior al 13%. En la mayor parte de las pruebas realizadas con los benchmarks utilizados y con EVLT se alcanza mejor makespan que con la EVPA.

GLOSARIO

Metaheurísticas. Procedimientos de métodos aproximados que están diseñados para resolver problemas difíciles de optimización combinatoria, basándose en una metodología de fácil comprensión.

Heurísticas. Métodos empíricos aportados por la experiencia del investigador los cuales no presentan una prueba forma de optimalidad y factibilidad.

Tiempo Polinomial. No existe un algoritmo determinístico que lo pueda resolver, por lo tanto, el tiempo es medible y puede ser representado por un polinomio.

Benchmarks. Puntos de referencias utilizados para medir y comparar resultados para un tipo de problema.

Calendarización. Programación de tareas con relación a los recursos disponibles secuenciados de la mejor manera.

Instancia. Datos de entrada que representan caso específico de un tipo de problema de calendarización o especificación de los valores de los parámetros.

Makespan. Duración total de una calendarización o programación de recursos que indica el tiempo final de la última tarea asignada.

Optimización Combinatoria. Rama de las ciencias de la computación enfocada en resolver problemas de decisión con relación a maximizar o minimizar una función objetivo.

Algoritmos Determinísticos. Siempre que se ejecutan producen un resultado idéntico o similar.

Estructura de vecindad. Función que genera una nueva solución muy similar de la solución de la que se partió, aplicando un pequeño cambio perturbatorio en la solución de inicio.

ÍNDICE

LISTA DE FIGURAS

LISTA DE TABLAS

LISTA DE ALGORITMOS

CAPITULO I: INTRODUCCIÓN

1.1 Introducción al JSP

El problema de talleres de manufactura (Job Shop Problem) es un problema complejo, el cual consiste en procesar distintas operaciones individuales que son requeridas para terminar diversos trabajos en varias máquinas. Este es un problema de optimización combinatoria que crece de manera exponencial a medida que se aumentan las máquinas y los trabajos. Existen diferentes tipos de problemas de programación de talleres de manufactura, de acuerdo con Arish y Young (2001), pero todos comparten el mismo objetivo básico que es, encontrar un horario óptimo de fabricación y por ende aumentar sus ganancias al disminuir el tiempo de producción.

Un problema de 6x6, tiene un total de $(N!)^m$ combinaciones, el cual se puede considerar un problema pequeño y fácil de solucionar computacionalmente, pero a medida que se aumenta el número de máquinas y el número de trabajos, el crecimiento en combinaciones es exponencial. Existen pocos métodos exactos que puedan determinar una solución a este problema en tiempo polinomial, y de forma determinística, por lo que, todos los avances que existen acerca del tema son basados en heurísticas y metaheurísticas, aun cuando son métodos de aproximación se pueden obtener soluciones factibles. Considerando que, por cada heurística aplicada se obtienen diferentes resultados, se puede observar un área de oportunidad para la investigación de estos problemas de calendarización, con el objetivo de mejorar la calidad en cada uno de estos resultados.

Existen diferentes metaheurísticas diferentes que pueden ser utilizadas para resolver problemas de calendarización. Algunos métodos populares incluyen programación entera, programación de restricciones y algoritmos de búsqueda heurística (métodos aproximados). Cada método tiene sus propias debilidades y fortalezas, por lo que es importante elegir el algoritmo adecuado para cada instancia de algún problema en particular. Como ya se mencionó, estos métodos de aproximación se diseñan mediante un conocimiento empírico de los

investigadores y representan un enfoque distinto al de los métodos exactos para encontrar soluciones cercanas a la solución óptima del problema.

1. 2 Planteamiento del problema

Como se mencionó anteriormente, el problema de calendarización en talleres de manufactura (Job Shop) pertenece al área de optimización combinatoria y es catalogado dentro de las ciencias computacionales con complejidad NP-Completo por Garey (1976). En tanto que no existe aún un algoritmo de solución eficiente para resolver óptimamente este tipo de problemas en tiempo polinomial.

Dentro de un JSSP se tiene una cantidad *n* de trabajos (*j*), que deberán ejecutarse y terminarse utilizando una cantidad *m* de recursos compartidos máquinas (*m*). Cada trabajo tendrá operaciones (*O*) que deberán completarse en un orden específico. Las operaciones se deben llevar a cabo en máquinas específicas y requieren un tiempo de procesamiento (*p*) en esa máquina. Las restricciones son:

- Una máquina no puede procesar más de una operación a la vez.
- Las operaciones no se pueden detener una vez que ya se haya iniciado el proceso.
- Los trabajos no pueden ser cancelados.
- Un trabajo no puede estar en dos máquinas a la vez.
- Cada operación debe de iniciar cuando termina la otra
- Las operaciones tienen una orden de máquina de precedencia y no se pueden cambiar.

La asignación de máquinas a las operaciones de cada trabajo genera una calendarización. El objetivo es optimizar algún aspecto del proceso, como finalización de la última tarea, dicha tarea es representada mediante un modelo matemático que se le conoce como función objetivo. Esta función objetivo, o función

de costo se conoce como makespan o tiempo que dura la elaboración del último trabajo u operación.

1.3 Complejidad del problema

La teoría de la complejidad computacional es la parte de la computación, que estudia los recursos y el tiempo requeridos para resolver un problema. Los recursos son el espacio o memoria de la computadora y el tiempo, es la duración de un proceso. Todos los problemas pueden ser clasificados de acuerdo con el grado de dificultad. La teoría de la complejidad computacional los clasifica en:

Los problemas tipo P o de Tiempo Polinomial:

Se puede representar con alguna de estas categorías de ecuaciones logarítmicas O (log *n*), las lineales O(*n*), las cuadráticas O(n^2), las cúbicas O(n^3). Pueden ser representados por un polinomio. La importancia de los algoritmos en cuanto a la complejidad se basa en diseñar algoritmos que cuenten con una complejidad logarítmica, para que independientemente del equipo en el que sea ejecutado, el tiempo de cómputo sea polinomial.

NP Este tipo de problemas está clasificado entre la categoría de los problemas intratables o difíciles de resolver, ya que no cuentan con algoritmos exactos que den una solución al problema en tiempo polinomial, solo hay métodos aproximados (heurísticas) que los acotan en tiempo polinomial obteniendo soluciones muy cercanas a la mejor solución.

NP Completos es un subconjunto de los problemas NP que son los más difíciles de resolver, o también conocidos como los problemas intratables de acuerdo con Papadimitriou (1998). La solución es tan extensa que no puede ser representada en un polinomio o en una función polinómica.

1.4 Objetivos

1.4.1 Objetivo General

Evaluar la factibilidad de las soluciones generadas con una nueva estructura de vecindad.

1.4.2 Objetivos específicos

- Diseñar una nueva estructura de vecindad
- Generar soluciones factibles aleatorias
- Aplicar la nueva estructura de vecindad con una búsqueda local iterada
- Analizar resultados

1.4.3 Alcance

El presente trabajo muestra únicamente el funcionamiento de la nueva estructura de vecindad con respecto al método clásico de permutación de pares adyacentes (EVPA) y da una medida de eficacia y de factibilidad en la ejecución de la estructura de vecindad, se aplica una búsqueda local iterada para comparar resultados.

1.5 Hipótesis

Al permutar pares adyacentes en una línea de tiempo que posiblemente pertenezcan a una o más rutas críticas, se mejorara el tiempo requerido para terminar los trabajos o makespan.

1.6 Justificación

El problema de talleres de manufactura es un área de gran oportunidad por ser un problema difícil de resolver, ya que no existen métodos que den solución a un problema y se aplique a todos los demás. Cada problema se clasifica en categorías, por ende, la solución de un problema pequeño no es la misma para uno grande con mayor complejidad. Las previas investigaciones utilizan heurísticas y metaheurísticas, pero todas se basan en una sola ruta crítica. En este proyecto se intenta romper múltiples rutas críticas en una línea de tiempo, un tema muy poco explorado y sin documentación que la respalde.

1.7 Organización de la tesis

En el capítulo 1. Se habla de manera general del problema a tratar, definición de algunos conceptos como complejidad computacional, se realiza el planteamiento del problema, los objetivos, alcances y la justificación.

En el capítulo 2. Trata de una introducción a los problemas de calendarización de talleres de manufactura, de su historia, del origen del problema y como se ha desarrollado en la actualidad y hacia donde va dirigido; También se muestra una investigación de las aportaciones que han hecho previos investigadores y la clasificación que le han dado a dicho problema.

En el capítulo 3. Se presenta la metodología a desarrollar y se detallan cada uno de sus pasos, así como una representación gráfica de los mismos. Se implementan los algoritmos de forma detallada. Se aplica el método de permutación de pares adyacentes a ambas estructuras EVPA vs EVLT, de igual manera se aplica un método de búsqueda local para optimizar las soluciones encontradas.

En el capítulo 4. Se muestran los resultados computacionales obtenidos, se realiza la comparación y el análisis con sus respectivos, además, se presentan gráficas con las diferencias o distancias entre cantidades de los datos obtenidos de ambos

métodos. Así como, la evaluación de la factibilidad de la nueva solución EVLT con las cantidades y porcentajes de ciclos que genera.

2 ESTADO DEL ARTE

2.1 Introducción

El Job Shop Scheduling Problem (JSSP) pertenece al área de optimización combinatoria, y a la categoría de calendarización de producción (Production Scheduling Problems o PSP). Existen un gran número de problemas de calendarización y optimización combinatoria en la industria, relacionados a la materia de investigación de operaciones, donde cada problema requiere un conocimiento específico de acuerdo con el tipo de programación que se va a realizar. Algunos ejemplos serian como, la logística y manufactura con recolección y entrega que manejan las grandes compañías como Amazon o como la calendarización de los vuelos en un aeropuerto que son muy complejos y difíciles de comprender, así como, los problemas de calendarización de horarios de las escuelas. Estos están fuera del alcance de esta tesis. Sin embargo, todo sistema de calendarización tiene sus fundamentos y principios ya que todos los problemas de calendarización comparten muchas características.

Los problemas de calendarización y producción están divididos en tres principales grupos. Cada uno de ellos, con características diferentes y por lo tanto se han implementado varios métodos para tratar de resolverlos más eficientemente Syarif et al. (2021), tienen diferentes aplicaciones y cada uno es una variante de los PSP.

- Calendarización de Talleres de Manufactura Abierto (OSSP). Su principal característica es que aquí se pueden programar los trabajos de forma arbitraria y no se cuentan con restricciones de precedencia ni de máquinas. De tal forma que es muy práctico llevarlo a cabo.

- Problema de Talleres de Manufactura Flexible (FJSP) es una variante de la programación abierta, donde la principal característica del problema es que todas las operaciones deben de realizarse en máquinas específicas. Se debe procesar un conjunto de trabajos en todas las máquinas del taller siguiendo un determinado flujo.

- Problema de Calendarización de Talleres de Manufactura (JSSP) Esta otra variante de los problemas de talleres de manufactura, tienen como principal característica, que cada trabajo consta de un conjunto de operaciones que deben de procesarse en un orden específico, este orden es conocido como restricción de precedencia y por esto es más difícil de trabajar.

2.2 Algoritmos de Optimización para el JSP

El JSP es el modelo clásico para resolver el problema de calendarización en talleres de manufactura y con el paso del tiempo se han propuesto diferentes algoritmos para resolverlo. Zhang et al. (2019), hacen una recopilación de varios de los algoritmos que se han propuesto para tratar de solucionar el problema. Estos algoritmos se pueden clasificar en dos categorías: Métodos de optimización exacta y métodos aproximados.

2.2.1 Métodos Exactos

Los métodos exactos se caracterizan por buscar la solución óptima del conjunto de soluciones para la función objetivo de un problema de optimización de manera matemática. Este tipo de métodos fueron los primeros en aplicarse al JSP. Entre los primeros esfuerzos para encontrar soluciones exactas se encuentra el de Johnson (1954), el cual propuso una serie de reglas de decisión para encontrar la calendarización óptima a un problema de producción de dos y tres máquinas. La opinión del origen del problema de JSP es mixta sobre quién propuso por primera vez el problema del taller en su forma actual sin embargo, hubo trabajos anteriores en la programación de talleres y todos ellos parecen haberse remontado a principios de la década de 1950, dentro de la investigación sobre optimización industrial.

En investigación de operaciones, la regla de Johnson es el método más común para programar trabajos en dos centros de trabajo. Su objetivo principal es encontrar una secuencia óptima de trabajos para reducir el makespan, además de enfocarse en reducir la cantidad de tiempo de inactividad entre los dos centros de trabajo.

Más tarde Manne (1960), llegó a la conclusión de que es posible encontrar la solución óptima para el JSP a partir de técnicas de programación lineal. Esta propuesta es para la aplicación de un programa lineal discreto; se muestra la evolución del problema tanto en las restricciones de secuenciación como en las restricciones de no interferencia para equipos individuales, sin embargo, de los problemas realistas a gran escala, esta formulación parece involucrar un alto costo, en recursos y tiempo, pero, aun así, puede valer la pena experimentar con algunos gastos informáticos.

Lomnicki (1965), subrayó que a pesar del trabajo hecho en Johnson (1954) para encontrar soluciones óptimas a problemas con más de dos máquinas, problemas de tamaño $m \geq 3$ siguen sin tener métodos de resolución similares. La propuesta de Lomnicki fue una adaptación al JSP del método de Ramificación y Poda (del inglés Branch and Bound) nombrado e introducido en Little et al. (1963) donde fue aplicado originalmente al Problema del Agente Viajero (del inglés Traveling Salesman Problem), otro problema clásico de optimización.

Se ha visto que los métodos exactos pueden obtener soluciones óptimas para algunas instancias del JSP, especialmente el método de Branch and Bound. Sin embargo, todos comparten el mismo problema del tiempo requerido para su solución, por esta razón, recientemente se ha llevado la atención a la aplicación de otro tipo de métodos: Los métodos aproximados.

2.2.2 Métodos Aproximados

Conforme las herramientas para resolver problemas han mejorado, así también los problemas de optimización combinatoria han ido creciendo y debido a su naturaleza de complejidad exponencial, se ha hecho cada vez más difícil de resolverlos de manera exacta. El JSP no es una excepción y es por eso por lo que

gradualmente se han adoptado métodos de aproximación para resolverlo. Como su nombre lo indica, estos métodos buscan encontrar soluciones aproximadas aceptables ya sea en su calidad o en la rapidez para obtenerlas. Se observa que los investigadores en su estudio teórico han utilizado heurísticas, algoritmos genéticos, algoritmos de recocido simulado, algoritmos de optimización de colonias de abejas y algoritmos híbridos.

Dentro de los métodos aproximados, todas las pruebas de han realizado tomando en cuenta solo una ruta crítica debido al alto costo de su cálculo. Sin embargo, existe múltiples rutas críticas en Levy et al. (1963) la ruta crítica es la ruta más larga (en el tiempo) de principio a fin, indica el tiempo mínimo necesario para completar todo el proyecto, el tiempo necesario para recorrer cada camino es la suma de los tiempos asociados a todos los trabajos en el camino, el método de la ruta crítica, es una técnica potente y sencilla para analizar, planificar y programar proyectos grandes y complejos, dicha metodología proporciona un medio para determinar qué trabajos o actividades, de los muchos que componen un proyecto, son críticos en cuanto a su efecto en el tiempo total del proyecto y la mejor manera de programar todos los trabajos y para cumplir una fecha objetivo con un costo mínimo.

El problema de programación de taller abierto es un tipo especial de problema de programación, que tiene *n* trabajos, cada uno con *m* máquinas y un conjunto de operaciones que se asignan sin restricciones. En la revisión de la literatura sobre problemas de programación de talleres abiertos Anand y Panneerselvam (2015), los clasificaron según sus diferentes medidas de rendimiento, para conocer el mínimo de la suma de los tiempos de finalización de los trabajos o makespan.

Satake et al. (1999), proponen una estructura de vecindad alternativa a las ya conocidas basadas en la ruta crítica para el JSSP. Esta estructura está inspirada en el comportamiento humano de una calendarización manual a partir de la representación de una solución por medio de una gráfica de Gantt, dicha estructura de vecindad genera nuevas soluciones recorriendo operaciones sobre los espacios vacíos visibles de la gráfica de Gantt y realiza cambios según ciertas consideraciones previas como reacomodo de otras operaciones.

Uno de los primeros métodos aproximados es mencionado en Festa (2014), describen que un algoritmo genético es un método descrito como una metaheurística análoga a los principios de la selección natural y la teoría evolutiva, donde la competencia entre individuos da como resultado que los más aptos sobrevivan y se reproduzcan, pasando sus genes a nuevas generaciones, durante la reproducción o cruzamiento, el material genético de las nuevas generaciones proviene de la recombinación del de los padres, junto con la reinserción de material genético perdido por causa de mutaciones, estos mecanismos de selección, mutación y cruzamiento repetidos provocan la evolución continua del cambio genético y la generación de individuos que sobreviven mejor en un entomo competitivo.

Jorapur et al. (2016)**,** usan otro algoritmo genético para resolver de manera eficiente los problemas de programación de talleres, se enfocan en la atención a los aspectos de la población inicial en los algoritmos genéticos y mucha atención a los operadores de recombinación, a partir de la población inicial, se pueden proporcionar mejores y más prometedoras soluciones, que comprenden diversas poblaciones con mayor complejidad.

En Cruz-Chávez (2014)**,** se presentó otro mecanismo de generación de vecindad para obtener nuevas soluciones para el JSP. Este mecanismo toma como base el concepto de la ruta crítica presente en las calendarizaciones del Job Shop y propone que las permutaciones de operaciones para generar nuevas soluciones sean únicamente entre aquellas pertenecientes a dicho grupo. Sin embargo, debido al costo computacional para calcular la ruta crítica en cada nueva solución, la evaluación se limita a buscar pares de operaciones sin tiempo de holgura, con la finalidad de generar soluciones factibles de una forma eficaz.

Mirshekarian y Sormaz (2016), mencionan que, debido a que los métodos exactos llegan a perder su aplicabilidad al incrementar el tamaño del problema, es necesario recurrir a heurísticas y estrategias para solucionar las instancias en tiempos razonables, ya que los métodos exactos, cuando el problema crece exponencialmente el tiempo de ejecución también crece y en la mayoría de los casos de forma exponencial.

Maryam y Nguyen (2017), presentaron un método directo y relativamente eficiente para resolver problemas de calendarización, dicho método determina las órdenes de trabajo para cada máquina, la evaluación la basan en la combinación de reglas de envío, el tiempo de procesamiento más corto de cada operación, la fecha de vencimiento más temprana de cada trabajo, la menor demora de las operaciones en cada secuencia y la idea de primero en llegar, primero en servir. Los modelos también se resuelven con el algoritmo cuello de botella, conocido como uno de los métodos heurísticos más comunes y confiables.

Recuento de metodologías usadas para resolver el JSP en la industria 4.0; El JSP es el modelo clásico para resolver el problema de calendarización en talleres de producción y con el paso del tiempo se han propuesto diferentes algoritmos para resolverlo, Zhang et al (2019), mencionan que la programación se debe tratar como un sistema de fabricación inteligente y distribuido respaldado por tecnologías de fabricación novedosas y emergentes, como colecta masiva con los sistemas de información, con el fin de transferir la calendarización tradicional a la programación distribuida e inteligente; la carga de trabajo computacional puede ser reducida considerablemente, si el sistema es más flexible y ágil. El actual sistema de programación de talleres se está volviendo cada vez más complejo, dinámico y flexible pues no solo se necesita considerar JSP en múltiples máquinas, múltiples recursos, incluso varias fábricas y logística y métodos híbridos.

Colonia de hormigas en recocido simulado Ying y Lin (2020). En este artículo presentan un recocido simulado de inicio múltiple con un algoritmo de cronograma de cambio bidireccional para mejorar la calidad de la solución, 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 fue proporcionar un método alternativo que intenta modelar el comportamiento de como las hormigas inician su ruta y generan varias en busca de alimento.

En Liu et al (2023), toman en cuenta los procedimientos, las máquinas, el tiempo requerido, el material usado, todo eso para poder llevar a cabo la minimización del makespan ya que todo proceso de producción tiene como objetivo generar recursos o ganancias, pero antes se deben de tomar en cuenta varios factores que deben

de enfrentar los problemas de calendarización como son máquinas no operables, retrasos en tiempo de entrega de proveedores, trabajadores no capacitados.

Zhi y Yingjian (2022)**,** consideran el medio ambiente como uno de los desafíos más difíciles para la fabricación moderna, el problema de programación del taller de trabajo flexible verde (GFJSP), toman en cuenta el consumo de energía y efectos de aprendizaje de los trabajadores. Se desarrolla el algoritmo de búsqueda de gorrión multiobjetivo (IMOSSA) con programación lineal entera mixta (MILP) con solucionador de CPLEX, este algoritmo simula a un grupo de aves en busca de comida y como se alertan al momento de que se aproxima un depredador enviando alertas a todo el grupo de aves.

La optimización de enjambre de partículas Anuar et al (2019) es un método de optimización basado en el comportamiento social de los organismos, como manadas de pájaros o enjambres de abejas, también se puede modificar para resolver problemas en espacios discretos. Este método has sido capaz de optimizar el JSP de forma notoria.

En Tian et al (2020), se propone un algoritmo de ADN (ácido desoxirribonucleico) para resolver la programación de talleres de manufactura, el algoritmo de ADN tiene complejidad $O(n^2)$. Con base en las operaciones de ADN del modelo Adleman-Lipton, una estrategia de codificación apropiada para generar todas las soluciones posibles en paralelo utilizando la computación de ADN, los resultados muestran que la propuesta funciona mejor que la heurística comparativa.

Uno de los algoritmos más utilizados es la búsqueda local y el cual es la base de muchos de los métodos usados en problemas de optimización de acuerdo con Rabadi (2016), es el algoritmo que consiste en realizar un seguimiento de la mejor solución registrada hasta el momento, el cual consiste en buscar en su vecindad otra mejor solución, si la encuentra, la reemplaza como nueva mejor solución actual y continúa con el proceso de mejora, hasta que no se puede mejorar la mejor solución actual.

Existen 2 subgrupos en los que podemos dividir las búsquedas:

- De búsqueda local consiste en modificar iterativamente y de manera inteligente una solución inicial hasta que ninguna solución cercana mejore a

la actual. El hecho de ir descartando soluciones que no superan a la mejor obtenida hasta el momento hace que estas metaheurísticas de búsqueda local se conviertan en algoritmos voraces. El principal inconveniente de las búsquedas locales es que se pueden quedar ancladas en óptimos locales con facilidad**.**

- De búsqueda global extienden su búsqueda al resto del espacio de soluciones, evitando de esta manera la localidad de las metaheurísticas de búsqueda local.

Los antecedentes muestran que cuando se trata de la calendarización sobre el JSSP generalmente se encuentran múltiples rutas críticas, la mayoría de los investigadores utilizan heurísticas y metaheurísticas para hacer movimientos con perturbaciones y aplican su metodología a una sola ruta crítica, no toman en cuenta esas múltiples rutas críticas. Este antecedente es la base de esta investigación al tratar de hacer permutaciones de pares adyacentes en una línea de tiempo que posiblemente pertenezcan a varias rutas críticas y, por consiguiente, esperar mejores resultados que los de permutación en pares adyacentes.

Los métodos descritos anteriormente son algunos de los más destacados dado el tiempo que han estado presentes y su confiabilidad en resultados, eso no quiere decir que sean los únicos, ya que existe una gran cantidad de nuevas estrategias y algoritmos para resolver este tipo de problemas de optimización. Dentro de los métodos aproximados se puede ver que los investigadores tratan de imitar el comportamiento de animales, así como los fenómenos naturales para tratar de solucionar los problemas de calendarización a pesar de los esfuerzos aún no se tiene una solución concreta que ponga fin a la investigación de los problemas.

2.3 Representaciones del JSSP

2.3.1 Notación para JSSP

El problema de calendarización de talleres de manufactura está formado por un conjunto finito de trabajos con cardinalidad *n: J* = {*J*1,*J*2,…*Jn*}, y *n* |*J*| un conjunto finito *M* de máquinas con cardinalidad *m*: *M* = {*M*1,*M2*….*Mm*} y *m* = |*M*| y un conjunto

de O que consiste en $n * m$ operaciones $\mathcal{O} = \{ i1, i2, \ldots im*n\}$ 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 $M_k \in M$ en la que dicha operación debe ser procesada. Cada operación i tiene además un tiempo de inicio s y un tiempo de procesamiento 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 objetivo considerado en este trabajo es minimizar el makespan (Cmax), que consiste en minimizar el intervalo de tiempo entre el inicio del procesamiento del primer trabajo (tiempo de referencia 0) y el tiempo de terminación del procesamiento del último trabajo, es decir, el intervalo de tiempo en el que se procesa completamente la totalidad de los trabajos (órdenes de producción). Se consideran los siguientes supuestos:

El tiempo total de una calendarización o makespan es igual al intervalo de tiempo entre el inicio del procesamiento del primer trabajo y el tiempo de terminación del procesamiento del último trabajo, el decir, el intervalo de tiempo en el que se procesa completamente la totalidad de los trabajos el cual se representa como *MAX (si+pi)*, donde *MAX* se obtiene mediante la función objetivo, en la Tabla 2.1 se muestran los elementos descritos anteriormente de forma resumida.

SÍMBOLO	DESCRIPCIÓN
J	Conjunto de trabajos
n	Número total de trabajos
M	Conjunto de máquinas
m	Número total de máquinas
O	Conjunto de operaciones
*n * m*	Número total de operaciones
si	Tiempo de inicio de operación *i*
p_i	Tiempo de proceso de la operación *i*
$(s_i + p_i)$	Tiempo de terminación de la operación *i*
$MAX\ (s_i + p_i)$	Máximo tiempo de terminación del conjunto de operaciones (makespan)

Tabla 2.1. Notación para JSSP

2.3.2 Modelo Matemático Para el Problema de Optimización

En el modelo matemático del JSSP, la ecuación (1) de la siguiente Tabla 2.2 se escribe la función objetivo a minimizar el makespan, representada por el tiempo de inicio más el tiempo de procesamiento de la última operación que pertenece a un trabajo en la calendarización. En la ecuación (2) describe el conjunto de restricciones en la que el tiempo de inicio de cualquier operación debe ser de un 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	$Min\ f = [\ \underset{j \in O}{MAX}\ (s_j + p_j)]$
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	$\forall i, j \in O \mid (i, j) \in M_k\ s_i + p_i \leq s_j\ V\ s_i + p_i \leq s_i$

Tabla. 2.2 Modelo matemático para la ecuación de optimización

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. En la figura 2.1 se muestra un ejemplo de grafo para una instancia de 3x3 (tres trabajos y máquinas por su simplicidad ya que un grafo de 6X6 es más difícil de entender y representar).

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.

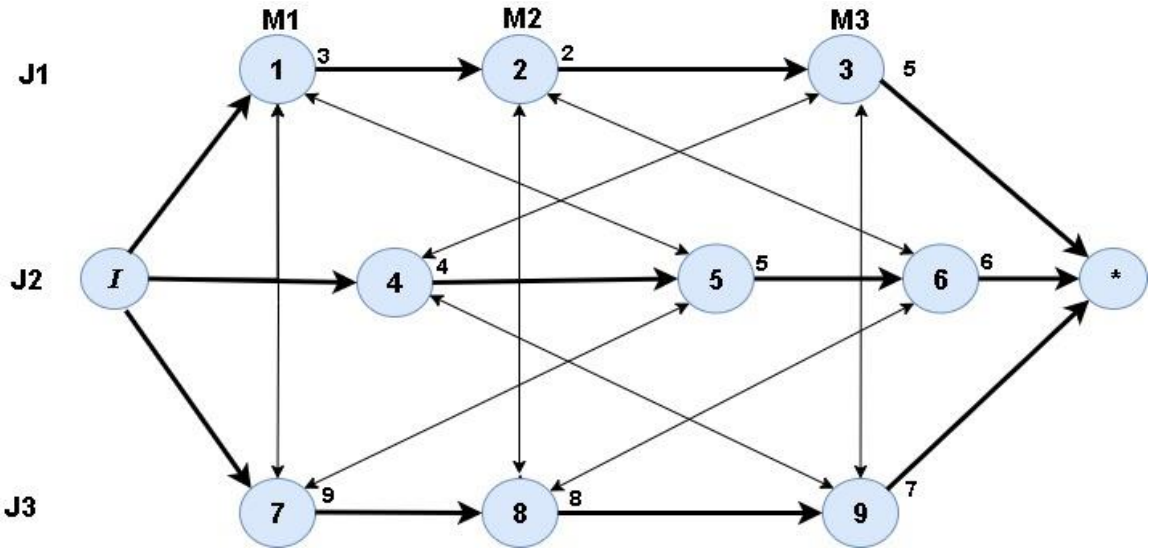


Figura 2.1 Grafo dirigido

La figura 2.2 muestra una posible calendarización para el ejemplo de 3X3, donde 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.

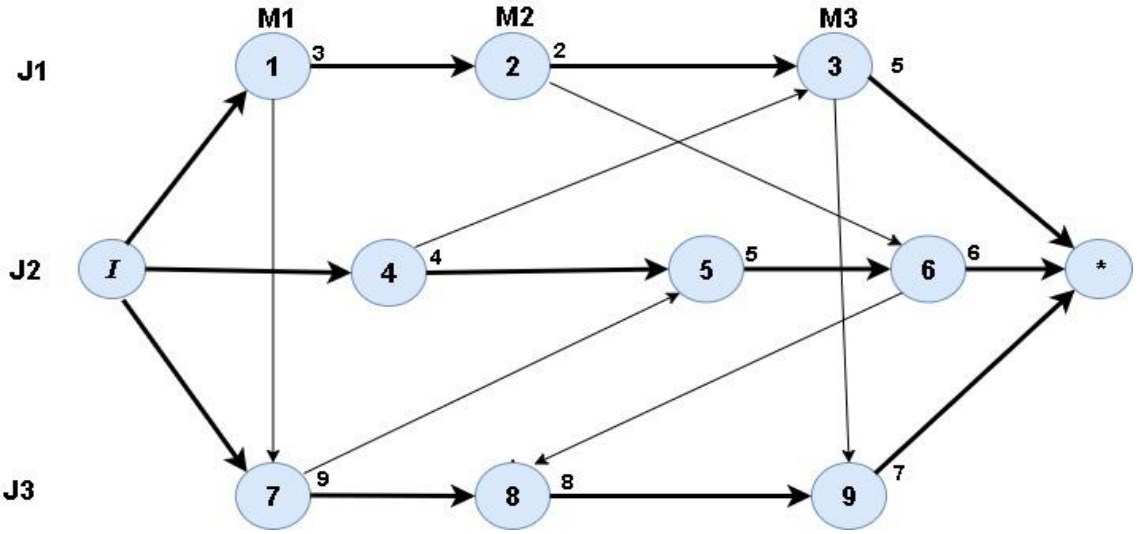


Figura 2.2 Posible calendarización

2.3.4 Diagrama de Gantt

Otra forma de representar una calendarización para el problema del JSP, es mediante un diagrama de Gantt, este tipo de esquema permite visualizar la asignación de cada operación, su tiempo de inicio y tiempo de proceso a lo largo de una tabla, dicha tabla está dividida en un número de filas igual a la cantidad de máquinas definidas por el problema y se extiende en columnas de unidades que llegan hasta el valor total del makespan. En la figura 2.3 se puede apreciar la calendarización del grafo anterior con el orden en que se ejecutan los trabajos por ejemplo el trabajo verde representa el trabajo 1 con operaciones en sus respectivas máquinas.

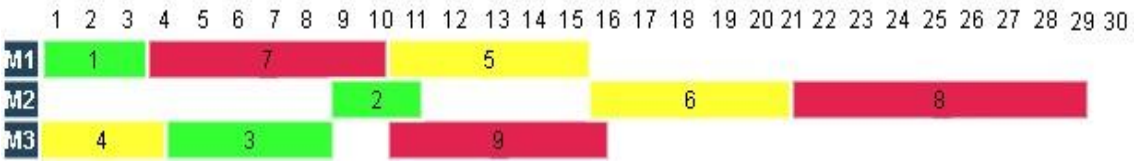


Figura 2.3 Diagrama de Gantt

3 METODOLOGÍA

3.1 Introducción

En los capítulos anteriores se habló sobre las características del JSSP, así como de los métodos más comunes para tratar de solucionarlo. En esta tesis se propone un algoritmo con una nueva estructura de vecindad, que consiste en permutar pares adyacentes en todas las máquinas que se encuentren en la misma línea de tiempo (Valores entre 1 y el makespan), en la figura 3.1 se muestra el esquema a seguir.

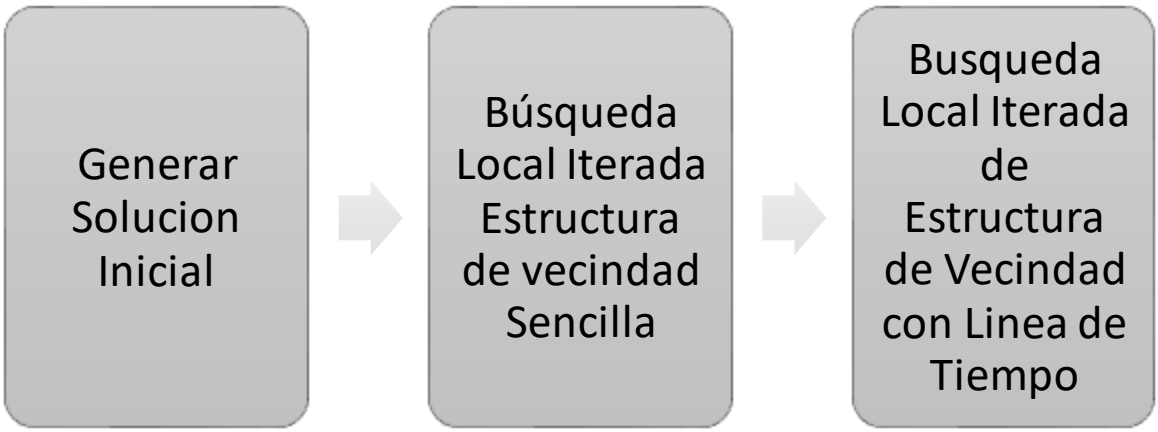


Figura 3.1 Esquemas de procedimientos para evaluación para estructuras de vecindad.

3.2 Instancias de Prueba para el JSSP

Las instancias de datos de prueba que se usan para evaluar este algoritmo son tomadas de la literatura y se encuentran en el formato presentado en la página de Or-Library (1990) en la Tabla 3.1 se muestra una instancia que consiste en un archivo de texto el cual contiene la información del número de trabajos y el número de máquinas y su tiempo de procesamiento. Las filas *Jn*, representan los trabajos (*j*) que a su vez están divididos en 6 columnas de operaciones (*O*) y dentro de cada columna las máquinas (*m*) en las cuales se tiene que llevar a cabo cada operación, el segundo valor de cada columna es su tiempo de procesamiento (*p*) que indica cuanto tiempo dura esa operación. La Tabla 3.1 representa las máquinas y su tiempo de procesamiento como (*m,p*).

Estos valores no cambian a lo largo de la tesis, ya que representan las restricciones de precedencia y no deben ser modificados.

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

Tabla 3.1 Datos de entrada de instancia 6X6 llamada (mt06)

3.3 Generación de la Solución Inicial.

Para poder generar una solución inicial aleatoria, se tienen que seguir varios procedimientos, los cuales consisten en cargar los datos de entrada, inicializar las estructuras de datos, procesar la información y producir los resultados deseados, los pasos se describen a continuación:

3.3.1 Generación de la Estructura de Trabajos

Para poder trabajar con el algoritmo convencional de Nakano y Yamada (1991) y poder llevar a cabo la calendarización, es necesario establecer el marco de trabajo, el primer elemento esencial, es una estructura de trabajos en el lenguaje de programación de programación “c” con sus respectivos campos.

3.3.2 Representación Gráfica De la Estructura de Trabajos

La figura 3.2 se muestra una representación de una estructura 6X6. La cual consiste en 7 elementos, de 0 a 6 {0,1,2,3,4,5,6}. El elemento *0*, se utiliza como máquina ficticia donde se inicializa la calendarización, la fila (*Jn)* representa un trabajo completo, es decir, cada columna es equivalente a una operación con sus respectivos campos:

- f: Es un campo usado como bandera, para indicar si el trabajo ha sido agendado o aún no. Este campo o variable contiene los valores de *1* y *0* únicamente donde *0* es no agendado, y *1* es equivalente agendado.
- m: Representa la máquina y es el principal elemento para la calendarización.
- p: Es el tiempo de procesamiento y una variable constante que indica cuánto tiempo dura la operación.
- s: Tiempo de inicio, este campo indica cuándo se inicia la operación.
- c: Tiempo de término, indica cuándo termina la operación de ese trabajo.

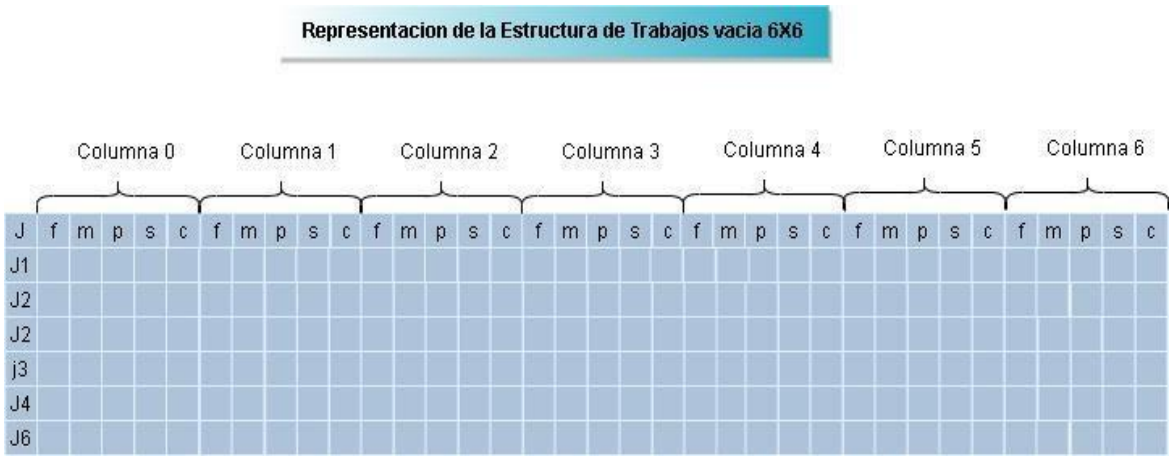


Figura 3.2 Estructura de trabajos vacía

A continuación, se muestra el fragmento del código en lenguaje de programación en “c” en el cual se observa la declaración de las estructuras de máquinas y trabajos para poder trabajar la calendarización; más adelante en el apéndice se presenta un fragmento de código y sus usos.

```
typedef struct {
    int  m;
    int p;
    int s;
    int c;
    int f;
}Trabajo;
```

```
typedef struct{
    int j;
    int p;
    int s;
    int c;
    int  f;
} Maquina;
```

3.3.3 Representación Gráfica de la Estructura De Máquinas

En la figura 3.3 se muestra que la estructura de máquinas es idéntica a la estructura de trabajos, con la excepción de que está estructura tiene un campo *j* que representa los trabajos.

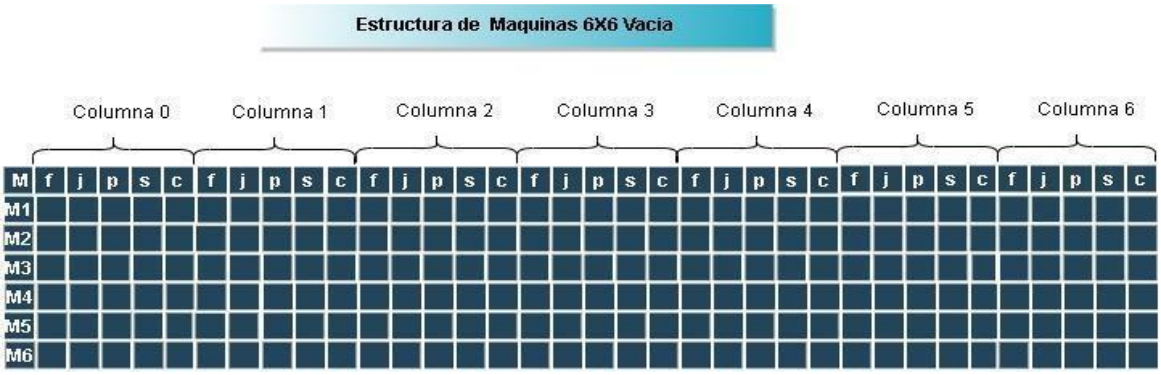


Figura 3.3 Estructura de máquinas vacía.

3.3.4 Preparación de Matriz de Estructura de Trabajos

- Todos los campos del primer elemento (columna 0) se inicializan a cero, es de suma importancia identificar el campo, de tiempo de término (*c*) de todos los elementos de la estructura, ya que, al momento de calendarizar, es el

único al que se hace referencia en ambas estructuras solo cuando se inicia la calendarización.

- El campo *f* (bandera) con valor inicial en cero, indica que el trabajo no está agendado.
- Las máquinas(*m*) fueron incrementadas en 1, ya que el archivo original de instancias inicia con máquina 0, la cual es una máquina ficticia.
- Los tiempos de procesamiento (*p*) se inicializan con los datos de la instancia de prueba o Benchmarks.
- Los campos de tiempo de inicio (*s*) y tiempo de término (*c*) tomarán valores una vez calendarizado el trabajo.

La figura 3.4 muestra una representación de la estructura de trabajos que está inicializada y lista para calendarizar:

Datos de Entrada de Intancia 6X6 (mt06)

J	f	m	p	s	c																								
J1	0	0	0	0	0	0	3	1			0	1	3			0	2	6			0	4	7			0	6	3			0	5	6		
J2	0	0	0	0	0	0	2	8			0	3	5			0	5	10			0	6	10			0	1	10			0	4	4		
J3	0	0	0	0	0	0	3	5			0	4	4			0	6	8			0	1	9			0	2	1			0	5	7		
J4	0	0	0	0	0	0	2	5			0	1	5			0	3	5			0	4	3			0	5	8			0	6	9		
J5	0	0	0	0	0	0	3	9			0	2	3			0	5	5			0	6	4			0	1	3			0	4	1		
J6	0	0	0	0	0	0	2	3			0	4	3			0	6	9			0	1	10			0	5	4			0	3	1		

Fig. 3.4 Datos de entrada

3.3.5 Preparación de Matriz de Estructuras de Máquinas

La segunda estructura para poder establecer el marco de trabajo es la estructura de máquinas figura 3.5 que cuenta con las mismas características que la matriz de trabajos.

- Se inicializan todos los campos de la columna 0, todos a ceros, esto es porque al iniciar la calendarización en la columna 1 o trabajo 1 es donde se inicia la calendarización.

- A las máquinas se les asigna de forma aleatoria los trabajos que tienen que procesar, los valores son los mismos que los del Benchmarks de prueba cargados en la matriz de trabajos.

- Los tiempos de procesamiento (*p*) son copiados de la matriz de trabajos. Valores que no cambian a lo largo del proceso de calendarización.

Fig.3.5 Máquinas inicializadas

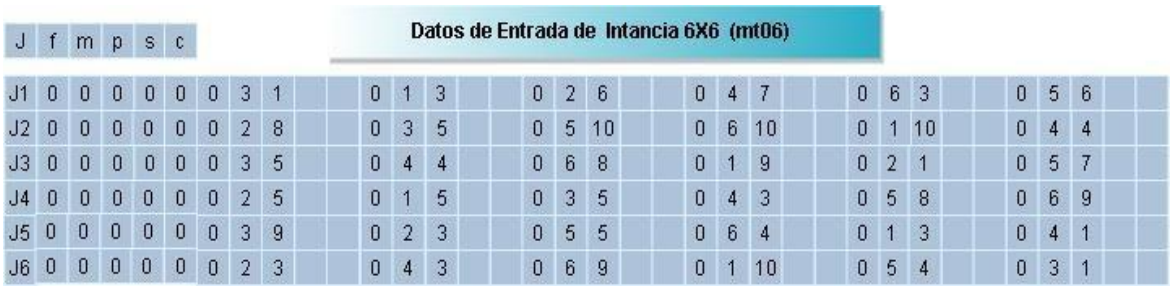
Estos son todos los datos iniciales necesarios para empezar con el proceso de calendarización. Cabe mencionar, que es de suma importancia entender que los valores de la máquina 1 (M1), por ejemplo, corresponden a trabajos que van a ser procesados en esa máquina y el tiempo que dura dicha operación (*p*). Lo mismo sucede con la estructura de trabajos, donde la fila contiene el listado de las máquinas donde se van a procesar las operaciones y su tiempo de procesamiento(*p*), esto es esencial y punto clave para entender la calendarización.

Otra importante observación es la columna cero en ambas estructuras, en el caso de la estructura de trabajos (ver figura 3.4) representa la máquina 0, dicha máquina es ficticia, ya que la máquina cero no ejecuta ningún trabajo, de la misma forma, con la estructura de máquinas, el trabajo cero tampoco existe, es un trabajo ficticio, pero estos campos son necesarios, al momento de calendarizar porque se hace referencia a la primera columna (0), ya que, las operaciones en la segunda columna al menos un trabajo, forzosamente deberá de iniciar en 1.

3.3.6 Algoritmo de Calendarización

Una vez preparado el escenario con ambas matrices, actualizadas con los datos necesarios, se implementa el algoritmo de calendarización y el algoritmo de

armonización global como se muestra en la figura 3.6; ambos son necesarios para poder calendarizar y obtener una solución inicial factible, como se puede apreciar en la figura, los campos de inicio y de término de la primera columna en adelante están todos vacíos, para que cuando se cumplan las condiciones y criterios de calendarización los valores sean asignados de acuerdo a sus tiempos de calendarización.



J	f	m	p	s	c																												
J1	0	0	0	0	0	0	3	1			0	1	3			0	2	6			0	4	7			0	6	3			0	5	6		
J2	0	0	0	0	0	0	2	8			0	3	5			0	5	10			0	6	10			0	1	10			0	4	4		
J3	0	0	0	0	0	0	3	5			0	4	4			0	6	8			0	1	9			0	2	1			0	5	7		
J4	0	0	0	0	0	0	2	5			0	1	5			0	3	5			0	4	3			0	5	8			0	6	9		
J5	0	0	0	0	0	0	3	9			0	2	3			0	5	5			0	6	4			0	1	3			0	4	1		
J6	0	0	0	0	0	0	2	3			0	4	3			0	6	9			0	1	10			0	5	4			0	3	1		

Figura 3.6 Estructura de trabajos inicializada

Ya con ambas matrices listas para calendarizar y llevar a cabo el primer algoritmo que es el algoritmo de calendarización de Nakano y Yamada (1991), se observa que ambas estructuras de datos tienen los campos (*f*) inicializados en ceros, que significa que no se han agendado, el campo (*j*), que es el número de trabajo en la matriz de máquinas de color azul fuerte (figura 3.7) indica el trabajo a ser procesado y su respectivo tiempo de procesamiento (*p*) de cada operación, y así consecutivamente de cada columna de la uno hasta la columna 6. La segunda estructura de la figura 3.7 contiene los datos iniciales de prueba de la página de Or-Library (1990).

Calendarizacion

M	f	j	p	s	c	f	j	p	s	c																									
M1					0	0	4	7			0	5	6			0	3	1			0	6	3			0	2	6			0	1	3		
M2					0	0	2	8			0	5	10			0	3	5			0	6	10			0	1	10			0	4	4		
M3					0	0	4	4			0	3	5			0	2	1			0	1	9			0	6	8			0	5	7		
M4					0	0	6	9			0	5	8			0	1	5			0	3	5			0	4	3			0	2	5		
M5					0	0	2	3			0	3	9			0	5	5			0	4	1			0	1	3			0	6	4		
M6					0	0	6	9			0	5	4			0	1	10			0	4	3			0	3	1			0	2	3		

Datos de Entrada de Intancia 6X6 (mt06)

J	f	m	p	s	c	f	m	p	s	c																									
J1					0	0	3	1			0	1	3			0	2	6			0	4	7			0	6	3			0	5	6		
J2					0	0	2	8			0	3	5			0	5	10			0	6	10			0	1	10			0	4	4		
J3					0	0	3	5			0	4	4			0	6	8			0	1	9			0	2	1			0	5	7		
J4					0	0	2	5			0	1	5			0	3	5			0	4	3			0	5	8			0	6	9		
J5					0	0	3	9			0	2	3			0	5	5			0	6	4			0	1	3			0	4	1		
J6					0	0	2	3			0	4	3			0	6	9			0	1	10			0	5	4			0	3	1		

Fig.3.7 Escenario para Calendarización

3.3.7 Pseudocódigo para Algoritmo de Calendarización

El procedimiento se realiza en la matriz de máquinas, los valores de las variables *n* y *m*, se obtienen del valor de máquinas y trabajos del archivo de instancia (mt06): $n*m$ = 6*6, que representa las dimensiones de las matrices; que es el proceso de calendarización qué se debe de llevar a cabo en cada fila interactuando entre ambas matrices y sus campos. En el Algoritmo 1 Calendarización, se muestra un procedimiento finito y preciso para poder encontrar la solución, detallando cada uno de sus pasos a seguir.

Algoritmo 1 Calendarización

Inicio algoritmo Calendarización
1 **Mientras** no estén agendadas todas las operaciones
2 Inicializar variable para detectar ciclo a 1
3 **Mientras** no se recorran todas las máquinas
4 **Mientras** no se recorran todos los trabajos
5 **Si** el campo ***f*** del trabajo en turno es 0 **entonces**
6 Seleccionar Trabajo
7 Ir a estructura de trabajos
8 Buscar máquina
9 **Si** la bandera de la máquina seleccionada es 0 ***entonces***
10 **Si** trabajo es igual a máquina
11 Buscar tiempo de término mayor del anterior
12 Agendar
13 Asignar cero a variable ciclo
14 **Fin si**
15 **Fin si**
16 **Sino**
17 Continuar con siguiente máquina
18 **Fin sino**
19 **Hasta** que se recorran todos los trabajos
20 **Hasta** que se recorran todas las máquinas
21 **Si** ciclo es igual a 1
22 Llamar algoritmo de armonización global
23 **Fin si**
24 **Fin Mientras** condición de paro (n*m)
Fin Algoritmo calendarización

Algoritmo 1 Calendarización

Paso 1: Mientras no estén agendadas todas las operaciones

El primer paso del proceso de calendarización es iniciar con un ciclo infinito, utilizando el lenguaje de programación ya mencionado, este bucle se repite hasta que se hayan agendado todas operaciones de los trabajos, su condición de paro es el paso 24 cuando la variable agendada sea igual $n * m$. Para la instancia de 6X6 será igual a 36 operaciones agendadas para cumplir la condición de paro del ciclo infinito.

Paso 2: Inicializar variable ciclo a 1

El ciclo es un impedimento para agendar un trabajo y poder generar una solución inicial factible, cuando se agenda una operación se asigna 0 a la variable ciclo para evitar llamar al algoritmo de armonización global, al asignarle 0 indica que el proceso de calendarización va por buen camino.

Paso 3: Mientras no se recorran todas las máquinas

Este es un ciclo anidado dentro del ciclo externo y es lo mismo que un ciclo para el JSP, este bucle es para recorrer máquina por máquina buscando candidatos para calendarizar, el recorrido se inicia en la máquina 1 (M1) se compara la primera operación en turno o próxima a ser agendada, que su campo f sea igual a 0, si se cumple la condición para calendarizar, se agenda, de lo contrario se termina este ciclo y se continúa con las siguientes máquinas consecutivamente M2, M3, M4, M5 y M6.

Paso 4: Mientras no se recorran todos los trabajos

Este bucle está dentro del ciclo de máquinas y recorre todas las columnas de cada fila o máquina, tiene como condición que el campo *f* del trabajo en turno sea *0*, Si se cumple, entonces, es considerado candidato para calendarizar, de ser 1 su valor, es ignorado y el ciclo continúa con el siguiente trabajo que su campo *f* sea *0*.

Paso 5: Si la bandera del trabajo es 0 entonces

Cuando los ciclos para recorrer máquinas y trabajos encuentran un candidato, se podrán proseguir con los **Pasos 6-13**, de lo contrario se salta hasta el **Paso 21** y continúa su ejecución, hasta llegar al final de las iteraciones de los bucles y si la variable ciclo es igual 1 se llama al algoritmo de armonización global.

Paso 6: Seleccionar Trabajo

Se lleva a cabo solo si se cumple el paso 5. Selecciona el trabajo o campo *j* en la matriz de máquinas que indica que ese trabajo se va a considerar como candidato a ser calendarizado y realizar comparaciones, si se cumplen se agenda.

Paso 7: Ir a estructura de trabajos

Solo se lleva a cabo si se cumplen los pasos anteriores 5 y 6. Con el valor almacenado en el campo *j* de la matriz de máquinas se hace referencia a esa fila en la matriz de trabajos, es decir, si, el valor almacenado en la variable *j* es 1, se va al trabajo 1 (*J1*) para seguir con el procedimiento de selección de candidatos.

Paso 8: Buscar máquina

Solo se lleva a cabo si se cumplen los pasos 5, 6 y 7 y su tarea es la de seleccionar el número de máquina que se encuentra en la matriz de trabajos, si, el campo *f* es *0* entonces, selecciona la máquina, que corresponderá al valor almacenado en el campo *m* de la matriz de trabajos. El valor que se encuentra en *m* representa la máquina donde el trabajo es procesado.

Paso 9: Si la bandera de la máquina seleccionada es 0 entonces

Si el campo *f*, de la estructura de trabajos es *0,* entonces es candidata para ser agendada.

Paso 10: Si trabajo es igual a máquina

Solo se lleva a cabo si se cumplen los pasos 5, 6, 7 ,8 y 9. Si, la maquina y el trabajo coinciden, entonces, se procede a agendar para que, se cumpla el valor que se encuentra almacenado en el campo *m* de dicho trabajo, el cual tiene que ser igual a la máquina de donde proviene ese trabajo, puesto que se tienen que cumplir las restricciones de precedencia que indican que el primer trabajo de *j* se debe de realizar en la máquina que tiene el campo *m*.

Paso 11: Buscar tiempo de término mayor

Solo se lleva a cabo si se cumplen los pasos 5,6,7,8,9 y 10. Una vez que se den los criterios de calendarización, la ecuación 3 del modelo matemático indica que un trabajo va a iniciar cuando se termine el otro, es por eso que se necesita encontrar el tiempo de término mayor de las operaciones anteriores entre ambas matrices.

Paso 12: Agendar

Solo se lleva a cabo si se cumplen los pasos 5,6,7,8,9,10 y 11. Para realizar este paso se debe de tomar en cuenta el paso anterior el cual fue, seleccionar el tiempo de término mayor (campo *c*) de una de las dos estructuras, se suma 1 al tiempo de término mayor del trabajo o de la maquina anterior, lo cual c + 1 = *s* como tiempo de inicio (campo *s*), el tiempo de término (campo *c*) es igual a (*s* + *p*) – 1 = *c*.

Paso 13: Asignar cero a variable ciclo

Si se llegó hasta este punto entonces ya no existe el ciclo. El ciclo para el JSP es cuando se realizan todas las iteraciones y no se pudo agendar ningún trabajo, se debe reiniciar la variable ciclo a cero para evitar llamar el algoritmo de armonización global.

Paso 14: Fin Si

Si no se cumplieron los pasos del 5 al 13, entonces, se continúa con la siguiente máquina y se termina el ciclo que recorren los trabajos ya que la calendarización tiene que ser en orden, la cual es otra restricción del modelo matemático.

Paso 15: Fin Si

Se tiene que se cumplió la condición del paso.

Paso 16: Fin Sino

Fin de condición.

Paso 17: Continuar con siguiente máquina

Se agendó trabajo o no se continúa con la siguiente fila de máquinas.

Paso 18: Fin sino

Fin de condición.

Paso 19: Hasta que se recorran todos los trabajos

Condición de paro del ciclo interno correspondiente a los trabajos.

Paso 20: Hasta que se recorran todas las máquinas

Condición de paro del ciclo externo correspondiente a las máquinas.

Paso 21*:* Si, ciclo es igual a 1

Se recorrieron todas las máquinas y trabajos y no se pudo agendar ninguna operación, el proceso de calendarización se encuentra en un ciclo infinito que no permite agendar ninguna operación y generar una solución inicial factible, entonces es necesario llamar al algoritmo de armonización global para poder agendar por lo menos un trabajo.

Paso 22: Llamar algoritmo de armonización global

Para poder generar una solución factible, siempre y cuando no se puedan agendar trabajos por la presencia de un ciclo, se debe de llamar al algoritmo de armonización global, para poder agendar una operación y romper el ciclo se necesita el algoritmo de armonización global de Nakano y Yamada (1991), el cual se explica a detalle para poder generar una solución inicial factible.

Pseudocódigo Para Algoritmo Armonización Global

	Inicio **Algoritmo 2** Armonización Global
1.	Seleccionar trabajo aleatoriamente
2.	Seleccionar máquina que no esté agendada en matriz de trabajos.
3.	Ir a máquina selecionada:
4.	Buscar trabajo que corresponda al que fue seleccionado aleatoriamente
5.	Mover trabajo al inicio para que pueda ser agendado
	Fin Algoritmo Armonización Global

Paso I Algoritmo armonización global:

De forma aleatoria se selecciona un trabajo, en la matriz de trabajos.

Paso II Algoritmo armonización global:

Seleccionar máquina que no esté agendada en matriz de trabajos. Ya que se llevó a cabo el paso **I**, se recorre la estructura de trabajos buscando la máquina cuyo campo *f* sea igual a *0* que indica que es la próxima maquina a ser agendada.

Paso III Algoritmo armonización global:

Ir a máquina seleccionada, una vez que se seleccionó la máquina disponible o próxima candidata a ser agendada en las estructuras de trabajos, se tiene que ir a la máquina que contiene el campo *j*. Será la máquina elegida para calendarizar a la próxima operación al romper el ciclo.

Paso IV Algoritmo armonización global:

Buscar trabajo que corresponda al que fue seleccionado aleatoriamente, dentro de la matriz de máquinas ahora se recorren los trabajos buscando el que fue seleccionado de forma aleatoria para tomarlo con su tiempo de procesamiento en donde quiera que se encuentre y pasarlo al inicio de no agendados de la lista de modo de que, sea el próximo a ser calendarizado y así es como se rompe el ciclo.

Paso V Algoritmo armonización global:

Mover el trabajo al inicio para que pueda ser agendado, las calendarizaciones deben de seguir un orden, al igual que, los trabajos se deben de agendar de izquierda a derecha hasta el final de la matriz.

Paso 23: Fin si

Se cumplió la condición.

Paso 24: Fin Mientras condición de paro (n*m)

Una vez agendados todos los trabajos en todas las máquinas termina la ejecución.

3.3.8 Representación del Algoritmo 1 Calendarización de Forma Grafica

Para entender mejor el proceso de calendarización y rompimiento de un ciclo se ejecutan los pasos del algoritmo 1 calendarización y algoritmo 2 armonización global, con las imágenes de las matrices amplificadas para una mejor lectura. El procedimiento se puede observar en figura 3.8 (primera comparación).

Los pasos del 1 al 3 se ejecutan cada vez que se lleva a cabo una nueva iteración del ciclo que recorren las estructuras, es por eso que a continuación se detallan los pasos del 3 al 9 para comprender el algoritmo de adentro hacia afuera.

En la figura 3.8 se observan los pasos del 3 al 9 de forma gráfica: con campos y las flechas que indican el flujo.

Paso 3. Mientras no se recorran todas las máquinas:
Se inicia el recorrido en **M1**.

Paso 4. Mientras no se recorran todos los trabajos:
Este ciclo nos permitirá recorrer de la columna 1 a la columna 6 en busca de candidatos.

Paso 5. Si el campo *f* es *0* entonces:
Es candidata para agendar.

Paso 6. Seleccionar Trabajo.
Se toma el valor que se encuentra en el campo *j* que es 4.

Paso 7. Ir a estructura de trabajos:
El valor almacenado en el campo *j* del paso anterior es 4, entonces se va a J4.

Paso 8. Buscar máquina:
Máquina cuyo campo *f* sea *0* y se toma el valor de su campo *m* que es 2.

Paso 9. Si trabajo es igual a máquina:
m2 es igual a M1?, "**no**", por lo tanto, no se puede agendar, se rompe el ciclo y se continúa con siguiente iteración.

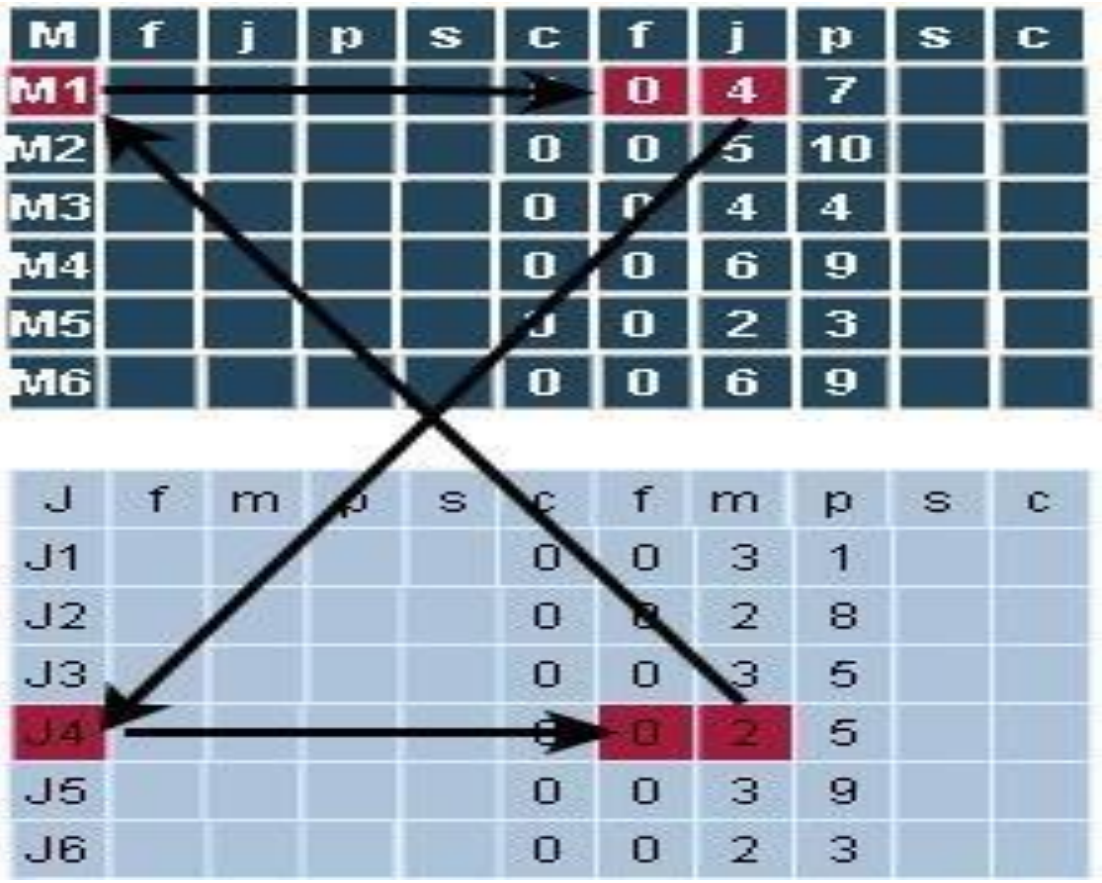


Figura 3.8 Primer comparación.

Segunda Comparación:

El mismo procedimiento de los pasos del 3 al 9 se repiten con cada una de las máquinas, hasta que, se cumplan los criterios de calendarización y se pueda calendarizar una operación, y así, poder continuar con la segunda máquina, figura 3.9 (segunda comparación) con el mismo procedimiento, pero diferentes valores.

Paso 3. Mientras no se recorran todas máquinas:
Se inicia el recorrido en **M2**.

Paso 4. Mientras no se recorran todos los trabajos:
Este ciclo nos permitirá recorrer de la columna 1 a la columna 6.

Paso 5. Si el campo *f* es *0* entonces:
Es candidata para agendar.

Paso 6. Seleccionar Trabajo:
Se toma el valor que se encuentra en el campo *j* que es 5.

Paso 7. Ir a estructura de trabajos:

El valor almacenado en el campo *j* del paso anterior es 5, entonces se va a J5.

Paso 8. Buscar máquina:
Máquina cuyo campo *f* sea *0* y se toma el valor de su campo *m* que es 3.

Paso 9. Si trabajo es igual a máquina:
m3 es igual a M2? "**no**", por lo tanto, no se puede agendar. Se rompe el ciclo y se continúa con siguiente iteración.

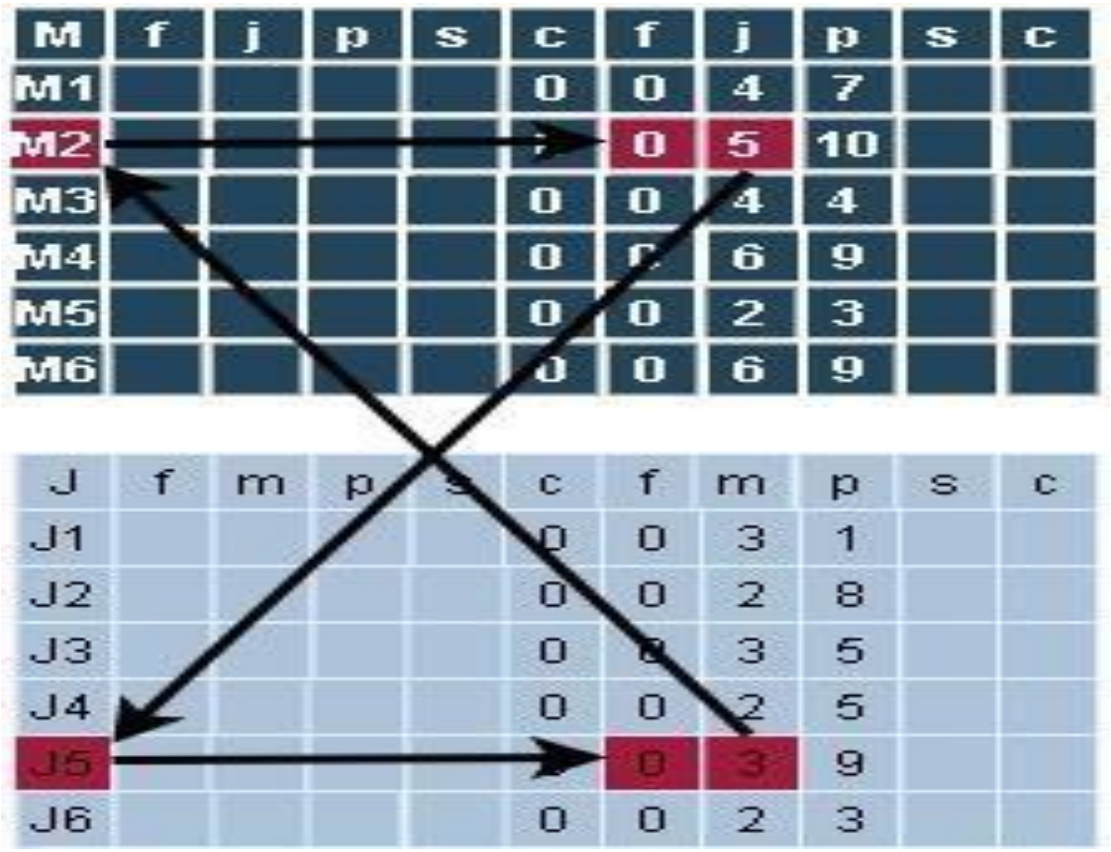


Figura 3.9. Segunda comparación

Tercera Comparación

El mismo procedimiento de los pasos anteriores se tiene que repetir con cada una de las máquinas, hasta que, se cumplan los criterios de calendarización y se pueda calendarizar una operación, continuar con la tercera máquina, figura 3.10 (tercera comparación) con el mismo procedimiento, pero diferentes valores.

Paso 3. Mientras no se recorran todas las máquinas:
Se inicia el recorrido en **M3**.

Paso 4. Mientras no se recorran todos los trabajos:
Este ciclo nos permite recorrer de la columna 1 a la columna 6.

Paso 5. Si el campo *f* es *0* entonces:

Es candidata para agendar.

Paso 6. Seleccionar Trabajo:

Se toma el valor que se encuentra en el campo *j* que es 4.

Paso 7. Ir a estructura de trabajos:

El valor almacenado en el campo ***j*** del paso anterior es 4, entonces se va a J4.

Paso 8. Buscar máquina:

Máquina cuyo campo *f* sea *0* y se toma el valor de su campo *m* que es 2.

Paso 9. Si trabajo es igual a máquina:

m2 es igual a M3? "**no**", por lo tanto, no se puede agendar. Se rompe el ciclo y se continúa con siguiente iteración.

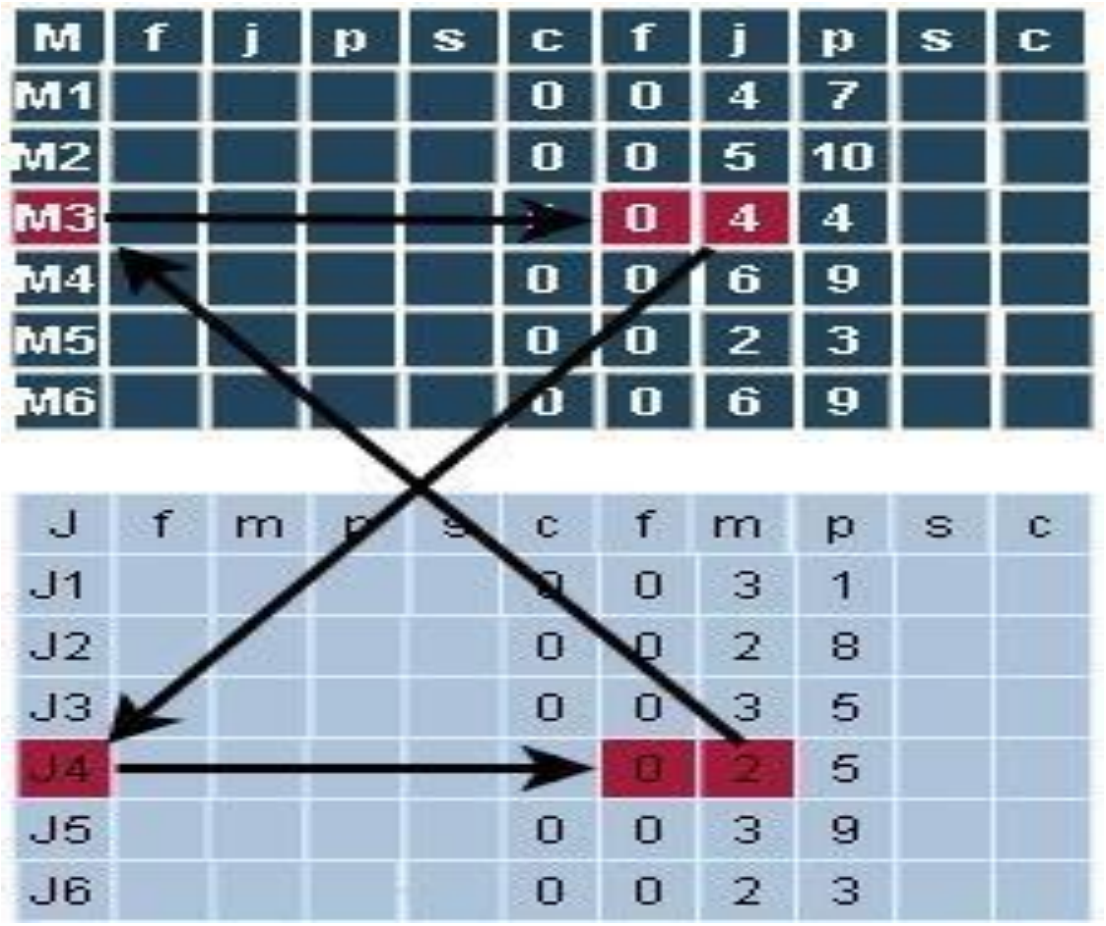


Figura 3.10 Tercera comparación

Cuarta Comparación

El mismo procedimiento de las comparaciones anteriores se tiene que repetir con cada una de las máquinas, hasta que se cumplan los criterios de calendarización, y continuar con la cuarta máquina, figura 3.11(cuarta comparación).

Paso 3. Mientras no se recorran todas las máquinas:
Se inicia el recorrido en **M4**

Paso 4. Mientras no se recorran todos los trabajos:
Este ciclo nos permitirá recorrer de la columna 1 a la columna 6.

Paso 5. Si el campo *f* es *0* entonces:
Es candidata para agendar.

Paso 6. Seleccionar Trabajo:
Se toma el valor que se encuentra en el campo *j* que es 6.

Paso 7. Ir a estructura de trabajos:
El valor almacenado en el campo *j* del paso anterior es 6, entonces se va a J6.

Paso 8. Buscar máquina:
Máquina cuyo campo *f* sea *0* y se toma el valor de su campo *m* que es 2

Paso 9: Si trabajo es igual a máquina:
m2 es igual a M6? "**no**", por lo tanto, no se puede agendar, se rompe el ciclo y se continúa con siguiente iteración.

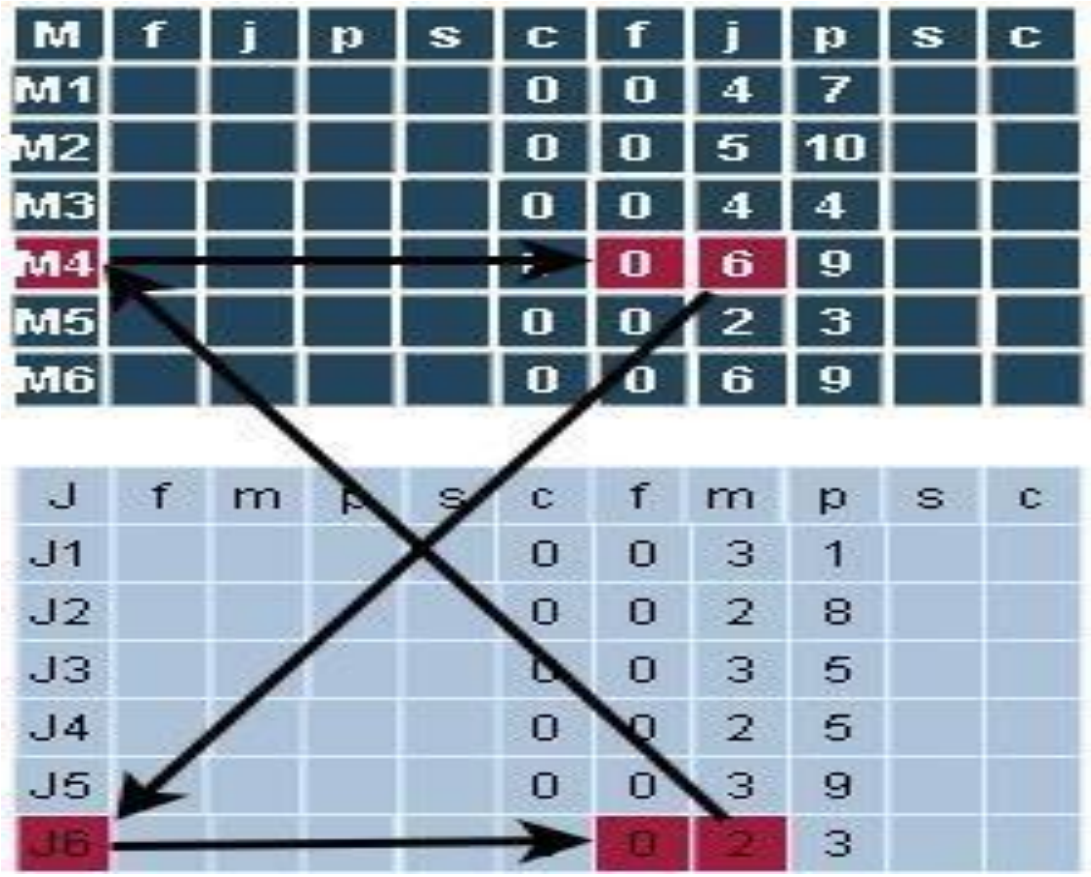


Fig. 3.11 Cuarta comparación.

Quinta Comparación

El mismo procedimiento de los pasos anteriores se tiene que repetir con cada una de las máquinas, hasta que, se cumplan los criterios de calendarización para continuar con la quinta máquina, figura 3.11 (quinta comparación)

Paso 3. Mientras no se recorran todas las máquinas:
Iniciar el recorrido en **M5**.

Paso 4. Mientras no se recorran todos los trabajos:
Este ciclo nos permite recorrer de la columna 1 a la columna 6.

Paso 5. Si el campo *f* es *0* entonces:
Es candidata para agendar.

Paso 6. Seleccionar Trabajo:
Se toma el valor que se encuentra en el campo *j* que es 2.

Paso 7. Ir a estructura de trabajos:
El valor almacenado en el campo *j* del paso anterior es 2, entonces se va a J2.

Paso 8. Buscar máquina:
Máquina cuyo campo *f* sea *0* y se toma el valor de su campo *m* que es 2.

Paso 9. Si trabajo es igual a máquina:
m2 es igual a M5? No, por lo tanto, no se puede agendar. Se rompe el ciclo y se continúa con siguiente iteración.

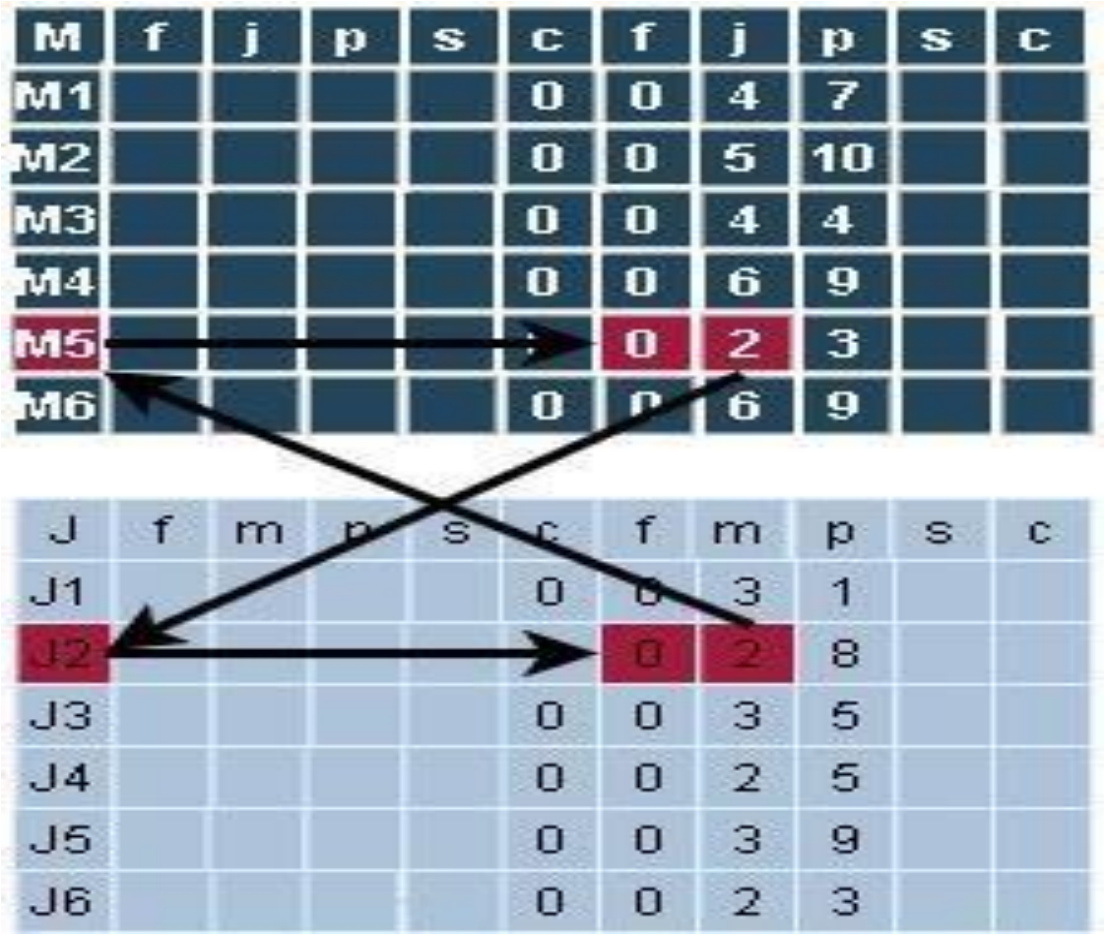


Figura 3.12 Quinta comparación.

Sexta Comparación

Se aplica el mismo procedimiento que los casos anteriores con cada una de las máquinas, hasta que, se cumplan los criterios de calendarización y continuar con la sexta máquina, figura 3.13 (sexta comparación)

Paso 3. Mientras no se recorran todas las máquinas:
Iniciar el recorrido en **M6**.

Paso 4. Mientras no se recorran todos los trabajos:
Este ciclo nos permitirá recorrer de la columna 1 a la columna 6

Paso 5. Si el campo *f* es *0* entonces:

Es candidata para agendar.

Paso 6. Seleccionar Trabajo:
Se toma el valor que se encuentra en el campo *j* que es 6.

Paso 7. Ir a estructura de trabajos:
El valor almacenado en el campo *j* del paso anterior es 6, entonces se va a J6.

Paso 8. Buscar maquina:
Máquina cuyo campo *f* sea *0* y se toma el valor de su campo *m* que es 6.

Paso 9. Si trabajo es igual a máquina:
m2 es igual a M6? "**no**", por lo tanto, no se puede agendar.

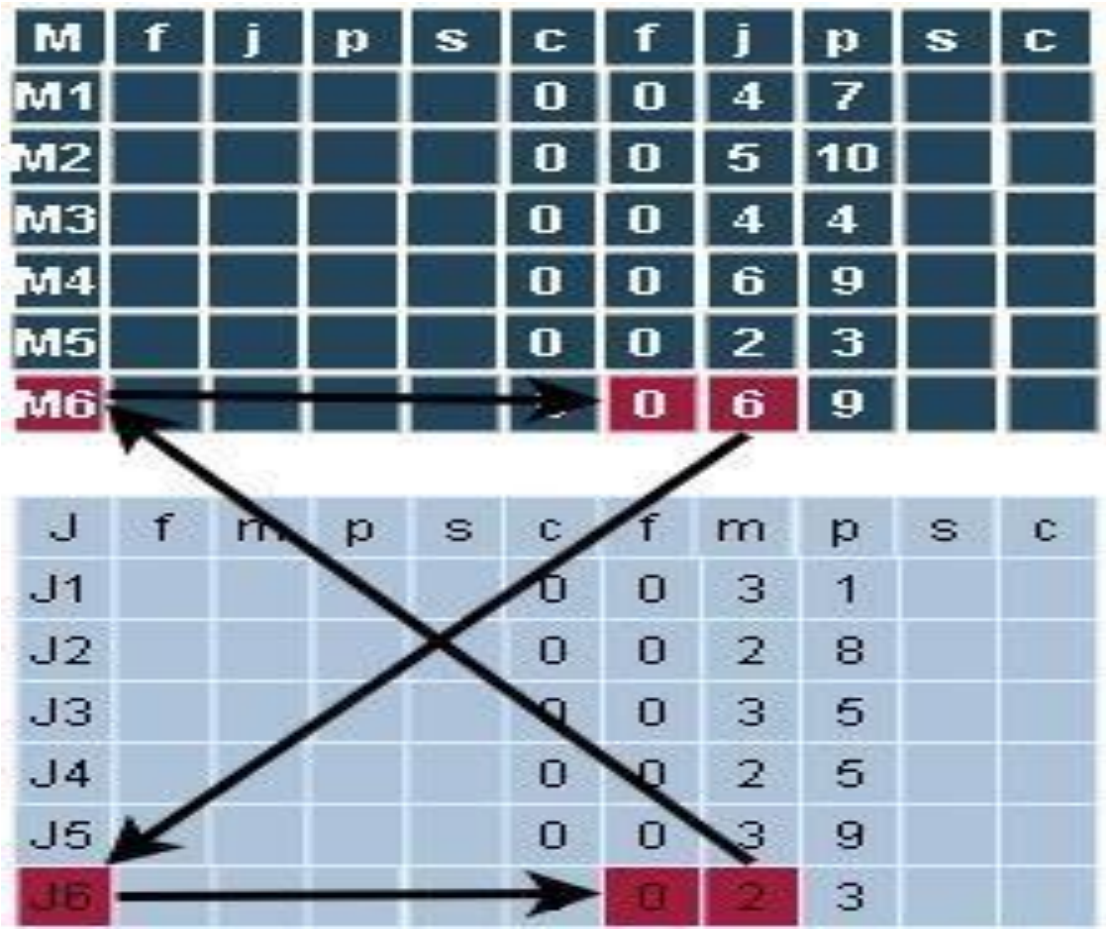


Fig. 3.13 Sexta comparación.

3.3.8 Presencia de un Ciclo

En el procedimiento de las imágenes anteriores, al intentar calendarizar los trabajos, se pude apreciar que ninguno cumplió con los criterios de calendarización, por lo tanto, estamos ante un ciclo, para poder agendar una operación y romper el ciclo se necesita el algoritmo de armonización global de Nakano y Yamada (1991), a continuación, se explica a detalle dicho algoritmo para poder generar una calendarización factible.

3.3.9.1 Rompimiento de Ciclo con Algoritmo de Armonización Global

1.- Seleccionar trabajo aleatorio:

El primer paso consiste en ir a la matriz de trabajos y de forma aleatoria seleccionar un trabajo, en la figura 3.14 se observa que fue seleccionada de forma aleatoria el trabajo número 2 o J2 representado en rojo.

2.- Seleccionar máquina que no esté agendada:

Dentro de la estructura de trabajos buscar la máquina que esté disponible, esto es que en su campo *f* su valor sea *0* y poder seleccionar la máquina correspondiente que en este caso es *2* en fila J2 y después ir a M2 donde se va a romper el ciclo.

Figura 3.14 Selección de trabajo aleatorio

3.- Ir a máquina seleccionada:

El paso 3 se ilustra en la figura 3.15 (Máquina para romper ciclo), en la matriz de máquinas se busca la máquina *2* en este caso (*M2*) es donde se rompe el ciclo. La columna 1 tiene los campos *j, p* con los valores *5,10* que corresponden al próximo trabajo a ser agendado, como no se cumplen los criterios de calendarización, se tienen que reemplazar los valores 5, 10 por unos que sí puedan ser calendarizados.

4) Buscar máquina que fue seleccionada aleatoriamente

En la figura 3.15 ya se tienen identificados los dos valores a ser permutados para romper el ciclo el 5,10 que es el próximo trabajo a ser agendado y los campos *j* y *p* de la columna 2 con los valores de trabajo y tiempo de procesamiento 2,8 que fue la máquina elegida para romper el ciclo, es decir, romper el ciclo consiste en intercambiar los valores 2, 8 por 5,10 para poder continuar con la calendarización.

Figura 3.15 Maquina para romper ciclo

La figura 3.16 muestra el intercambio de valores entre columnas de la misma máquina, dicho de otra manera, se intercambian máquinas con sus respectivos tiempos de procesamiento.

Figura 3.16 Ciclo roto

3.3.9.2 Primer Trabajo Agendado

Una vez roto el ciclo volvemos a reiniciar con el procedimiento de calendarización descrito anteriormente (ver figura 3.17):

Paso 3. Mientras no se recorran todas las máquinas:
Se inicia el recorrido en **M1**.

Paso 4. Mientras no se recorran todos los trabajos:
Este ciclo nos permitirá recorrer de la columna 1 a la columna 6 en busca de candidatos.

Paso 5. Si el campo *f* es *0* entonces:
Es candidata para agendar.

Paso 6. Seleccionar Trabajo.
Se toma el valor que se encuentra en el campo *j* que es 4.

Paso 7. Ir a estructura de trabajos:
El valor almacenado en el campo *j* del paso anterior es 4, entonces se va a J4.

Paso 8. Buscar máquina:
Máquina cuyo campo *f* sea *0* y se toma el valor de su campo *m* que es 2.

Paso 9. Si trabajo es igual a máquina:
m2 es igual a M1?, "**no**", por lo tanto, no se puede agendar, se rompe el ciclo y se continúa con siguiente iteración.

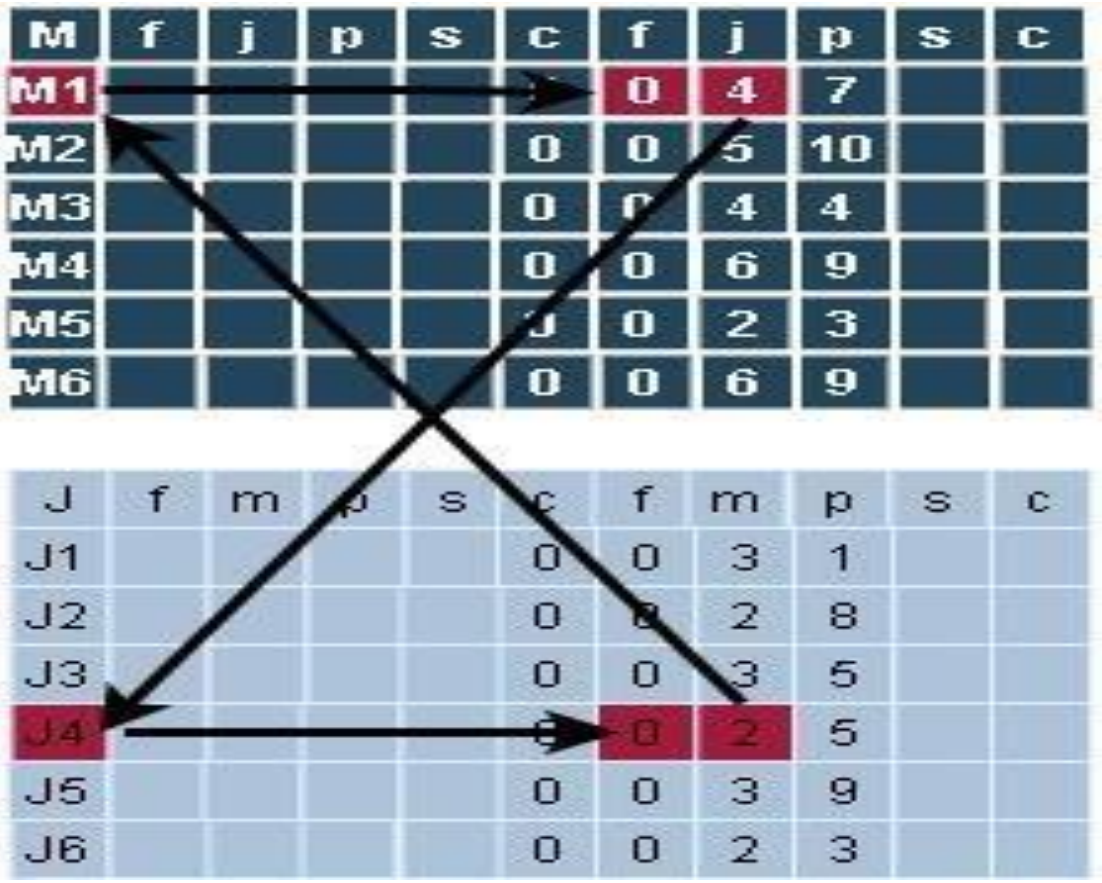


Figura 3.17 primer intento segunda pasada

Se sigue buscando un candidato por lo tanto el turno a la máquina 2.

Paso 3. Mientras no se recorran todas las máquinas:

Se inicia el recorrido en **M2**.

Paso 4. Mientras no se recorran todos los trabajos:
Este ciclo nos permitirá recorrer de la columna 1 a la columna 6 en busca de candidatos.

Paso 5. Si el campo *f* es *0* entonces:
Es candidata para agendar.

Paso 6. Seleccionar Trabajo.
Se toma el valor que se encuentra en el campo *j* que es 2.

Paso 7. Ir a estructura de trabajos:
El valor almacenado en el campo *j* del paso anterior es 2, entonces se va a J2.

Paso 8. Buscar máquina:
Máquina cuyo campo *f* sea *0* y se toma el valor de su campo *m* que es 2.

Paso 9. Si trabajo es igual a máquina:
m2 es igual a M2?, "**no**", por lo tanto, no se puede agendar, se rompe el ciclo y se continúa con siguiente iteración.

Para llevar a cabo este paso se tiene que hacer referencia al campo *c* del trabajo anterior, se puede notar que los 2 campos *c* de ambas matrices son ceros aún, en la figura 3.18 del primer trabajo agendado se pueden apreciar sus valores, por lo tanto, se busca el mayor de los 2 y como son iguales ambos, se toma 0.

Paso 12: Agendar

Para realizar este paso se debe de tomar en cuenta el paso anterior que fue, seleccionar el tiempo de término mayor (campo *c*) de entre ambas estructuras, en este caso ambos tiempos son cero, para poder asignar el tiempo de inicio al primer trabajo se suma 1 al tiempo de término mayor, lo cual 0 + 1 = 1 como tiempo de inicio (campo s), el tiempo de término (campo c) es igual a (s + p) – 1 = c. En la figura 3.18 del primer trabajo agendado se pueden ver los valores asignados en verde a los campos s y c indicando el inicio de la calendarización.

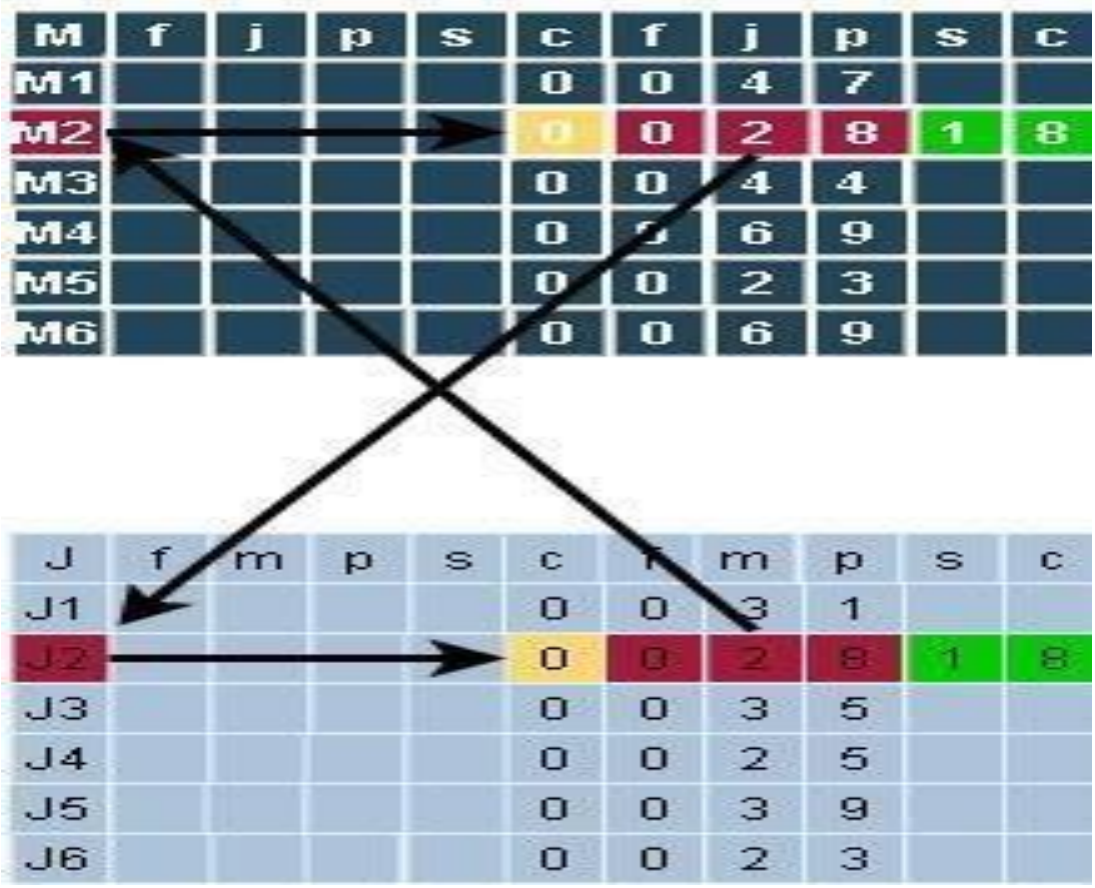


Figura 3.18 Primer trabajo agendado

Paso 13: Asignar cero a variable ciclo

Como ya se agendó un trabajo en esta iteración, ya no existe el ciclo en la calendarización, se tiene que continuar con las iteraciones hasta terminar de forma secuencial, es necesario asignarle cero a la variable ciclo para no llamar el algoritmo de armonización global.

3.3.9.3 Representación Gráfica del Primer Trabajo Agendado

En la figura 3.19, se observa que en cada trabajo asignado el campo *f* tiene el valor 1 el cual representa un trabajo agendado. Al campo *s* se le asigna 1 porque el tiempo de término de uno de los 2 valores de la máquina o el trabajo anterior es 0. El campo *c* toma su valor con la suma c = (s + p)-1, una vez agendado el trabajo en la matriz de máquinas, los mismos valores se ponen en la matriz de trabajos.

Fig. 3.19 Trabajo asignado

Para poder calendarizar todas las operaciones en la matriz de máquinas, se tiene que hacer uso del método de calendarización, así como del algoritmo de armonización global y cuando ya no se encuentra ningún candidato a ser agendado, se recurre al rompimiento del ciclo para poder generar una solución factible.

3.4 Búsqueda Local Iterada Par Adyacente

Al realizar el procedimiento de calendarización detalladamente en los pasos anteriores se llega a la solución inicial, como se muestra la figura 3.20 con todas las operaciones de cada trabajo calendarizado, se puede notar el número 79, que es el tiempo máximo que se toma en terminar todas las operaciones de todos los trabajos conocido como makespan.

Solucion Inicial

M	f	j	p	s	c	f	j	p	s	c																									
M1					0		1	3	2	4		4	5	20	24		6	10	35	44		2	10	48	57		5	3	60	62		3	9	63	71
M2					0		2	8	1	8		1	6	9	14		4	5	15	19		6	3	20	22		5	3	39	41		3	1	72	72
M3					0		1	1	1	1		2	5	9	13		3	5	14	18		4	5	25	29		5	9	30	38		6	1	53	53
M4					0		3	4	19	22		6	3	23	25		1	7	26	32		4	3	33	35		2	4	58	61		5	1	63	63
M5					0		2	10	14	23		1	6	38	43		5	5	44	48		6	4	49	52		4	8	53	60		3	7	73	79
M6					0		6	9	26	34		1	3	35	37		2	10	38	47		3	8	48	55		5	4	56	59		4	9	61	69

Solucion Inicial Intancia 6X6 (mt06)

J	f	m	p	s	c	f	m	p	s	c																									
J1					0		3	1	1	1		1	3	2	4		2	6	9	14		4	7	26	32		6	3	35	37		5	6	38	43
J2					0		2	8	1	8		3	5	9	13		5	10	14	23		6	10	38	47		1	10	48	57		4	4	58	61
J3					0		3	5	14	18		4	4	19	22		6	8	48	55		1	9	63	71		2	1	72	72		5	7	73	79
J4					0		2	5	15	19		1	5	20	24		3	5	25	29		4	3	33	35		5	8	53	60		6	9	61	69
J5					0		3	9	30	38		2	3	39	41		5	5	44	48		6	4	56	59		1	3	60	62		4	1	63	63
J6					0		2	3	20	22		4	3	23	25		6	9	26	34		1	10	35	44		5	4	49	52		3	1	53	53

Fig. 3.20 Solución inicial

En la figura 3.21 se muestra una representación de la gráfica de Gantt con la solución inicial del problema de calendarización, cada renglón representa una máquina y en cada máquina está el conjunto de trabajos calendarizados. El trabajo 1 marcado en color verde indica que en ese trabajo 1 se lleva a cabo una operación en máquina 1, otra operación en la máquina 2, máquina 3, hasta la máquina 6 y se observa que no hay un traslape en el línea de tiempo y esto indica que es una solución factible, la cual se puede aplicar en la vida real en un taller de manufactura.

Véase la secuencia de operaciones del trabajo 1, la cual indica las restricciones de precedencia y debe de respetarse esa secuencia de operaciones en cada una de las máquinas, por ejemplo, el trabajo uno siempre debe de empezar en la máquina 3 con duración de una unidad de tiempo y seguir en la máquina 1 con 3 unidades de tiempo y continuar su calendarización pasando por el resto de las máquinas. Los tiempos de ocio que indican que la maquina esta sin hacer nada, este tiempo de ocio se muestran en los espacios en blanco, por ejemplo, existe un tiempo de ocio en la máquina 1 entre el trabajo 1 y 4, el 1 termina de realizarse en el minuto cuatro y el próximo trabajo que va a llevar a cabo la máquina llega hasta el minuto 20. Entonces, la máquina tiene 16 minutos de tiempo de ocio.

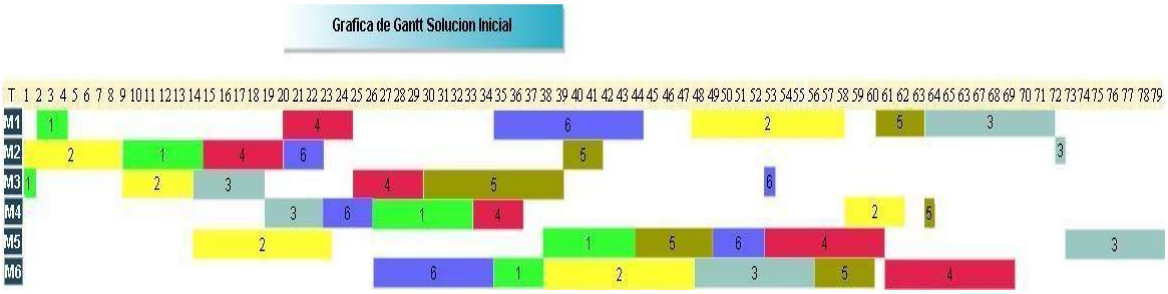


Figura 3.21 Grafica de Gantt solución inicial

3.4.1 Búsqueda Local Iterada

La búsqueda local iterada es uno de los algoritmos más sencillos para optimización en este tipo de problemas de calendarización. Es un procedimiento en el cual se somete la solución inicial de tipo aleatorio, normalmente, a una cierta cantidad de repeticiones, en dónde se lleva a cabo le ejecución de una estructura de vecindad con la finalidad de encontrar una mejor solución y reemplazarla por la solución actual, hasta que, se cumpla la condición de paro o se haya encontrado una solución buscada, en la figura 3.22, *S* es el espacio del conjunto de soluciones de un problema, N son los vecinos de la solución inicial y *s* representa con un punto a la solución inicial donde comienza la búsqueda de local iterada. A través de una estructura de vecindad definida como *N*, se puede encontrar la vecindad de soluciones *S'* de una solución *s*. Si *s'* es una solución vecina de *s* entonces la función o estructura de vecindad que permuta pares adyacentes de operaciones se define como $N(s) = \left\{ s' \in S' \vee s \overset{\beta}{\rightarrow} s' \right\}$, donde una solución vecina *s'* es alcanzada desde la solución *s* aplicando una pequeña perturbación (en este caso permutación) de un par o varios pares de operaciones, aquí *β* define el número de perturbaciones realizadas para encontrar un nuevo vecino *s'* de la vecindad *S'*.

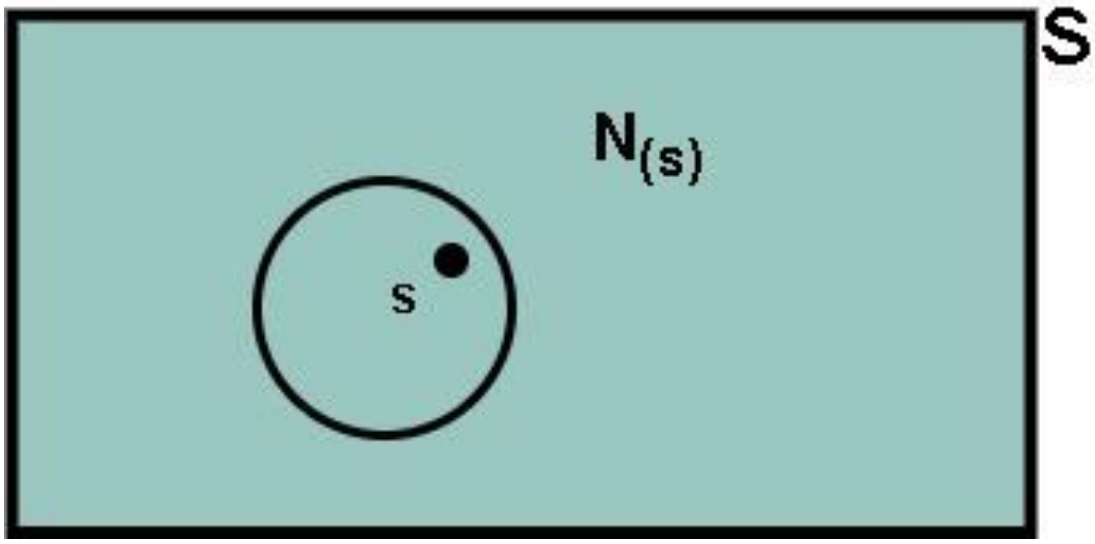


Fig. 3.22 Búsqueda local iterada

En esta sección, se hace la búsqueda local iterada con el algoritmo, propuesto por Cruz-Chávez (2014)**,** dónde se realizan permutaciones en pares adyacentes sin tiempo de ocio, al realizar una permutación se genera un nuevo vecino es por eso que las permutaciones de pares adyacentes son una estructura de vecindad sencilla, puesto que, al hacer la permutación el nuevo vecino puede ser igual mejor o peor, pero está muy cercano a la permutación previa o vecino. Se realiza la búsqueda local iterada en la solución inicial para poder optimizarla ya que ambos métodos son muy eficientes, a continuación, se realiza la descripción del algoritmo de búsqueda local:

Paso 1. Iniciar con solución inicial.

Paso 2. Mientras no se cumpla la condición de paro, aquí se puede controlar cuántas soluciones se quieren evaluar.

Paso 3. Solución inicial es el punto de partida.

Paso 4. Repetir n veces.

Paso 5. Solución actual es igual a la solución inicial.

Paso 6. Repetir.

Paso 7. Comparar el resultado que retorna la solución con la mejor solución.

Paso 8. Si la solución retornada es mejor que la solución actual entonces.

Paso 9. Reemplazar solo si solución es mejor.

Paso 11. Mientras no se cumpla el criterio de paro.

Paso 12. Dentro de la búsqueda local iterada si la solución es mejor que la solución actual entonces.

Paso 13. Se reemplaza la solución por la mejor solución dentro de la búsqueda local.

Paso 15. Termina la búsqueda local iterada.

Paso 16. Se cumple criterio de paro de soluciones iniciales.

Algoritmo 3 Búsqueda Local

```
    Inicio Búsqueda local iterada
1   S_ILS = Solución aleatoria
2   Mientras no se cumpla condición de paro
3       S_ILS = Solución aleatoria
4       Hacer
5           S_Actual = solución inicial S
6           Hacer
7               S´_LS =  función factibilidad(Actual)
8               Si(f(S´_LS) < f(S_Actual)) entonces
9                   S_actual = S´_LS
10              Fin si
11              Mientras no se cumpla criterio de paro de búsqueda local
12                  Si(f(s_actual) <= f(S_ILS)) entonces
13                      S_ILS = S_actual
14                  Fin Si
15              Fin Mientras
16      Mientras no se cumpla criterio de paro de búsqueda local iterada
    Fin Búsqueda local iterada
```

Algoritmo 3 Búsqueda Local

Selección aleatoria de par adyacente para permutación

Para poder llevar a cabo la permutación de operaciones entre pares adyacentes en la matriz de máquinas, cuyos campos y tiempos de término (*c*) de la operación anterior sean igual al tiempo de inicio (*s*) de la siguiente operación + 1 y que tengan la propiedad de que no existe tiempo de ocio entre ellos. Se requiere llevar a cabo la permutación para volver a recalendarizar, en la figura 3.23 permutación par adyacente, se observa que el trabajo 5, tiene 62 unidades de tiempo de termino, en la siguiente columna el trabajo 3 su tiempo de inicio es igual al tiempo de término del trabajo 5 , 62+1 = 63, entonces, no existe tiempo de ocio entre ambas, lo que la hace candidata para la permutación conocida como permutación de par adyacente.

Fig. 3.23 Permutación par adyacente

Representación de permutación de par adyacente y matrices listas para recalendarizar.

Una vez realizada la permutación de pares adyacentes se inician a ceros los campos *s* y *c* de ambas matrices ya que se vuelve a generar una nueva solución. La figura 3.24 par adyacente permutado está lista para ser recalendarizada y generar una nueva solución.

Fig. 3.24 Par adyacente permutado

3.4.2 Solución Mejorada con Búsqueda Local en Par Adyacente

La figura 3.25 Solución optimizada, se le aplicó el algoritmo de búsqueda local iterada, la cual consiste en buscar la mejor solución del espacio de soluciones a solución actual, la búsqueda local iterada sigue buscando soluciones dentro del espacio de soluciones, hasta encontrar la mejor. La figura muestra múltiples rutas críticas o makespans, ya que terminan varios trabajos en 55 que es el tiempo óptimo para una instancia de 6X6.

Figura 3.25 Solución optimizada

3.4.3 Grafica de Gantt de Solución Optimizada

La figura 3.26 Gráfica de Gantt optimizada es la representación gráfica de la solución a la instancia de 6X6 optimizada con la búsqueda local iterada. No hay traslapes en los trabajos y los tiempos de ocio están reducidos en la mayoría de las máquinas.

Fig. 3.26 Gráfica de Gantt optimizada.

3.5 Estructura de Vecindad Línea de Tiempo

Se toma como base la misma solución inicial, solo que a continuación se realiza la búsqueda local iterada con permutaciones en la estructura de vecindad línea de tiempo. La figura 3.27 Solución inicial, línea de tiempo sin optimizar muestra la solución inicial.

Solucion Inicial

M	f	j	p	s	c	f	j	p	s	c																									
M1					0		1	3	2	4		4	5	20	24		6	10	35	44		2	10	48	57		5	3	60	62		3	9	63	71
M2					0		2	8	1	8		1	6	9	14		4	5	15	19		6	3	20	22		5	3	39	41		3	1	72	72
M3					0		1	1	1	1		2	5	9	13		3	5	14	18		4	5	25	29		5	9	30	38		6	1	53	53
M4					0		3	4	19	22		6	3	23	25		1	7	26	32		4	3	33	35		2	4	58	61		5	1	63	63
M5					0		2	10	14	23		1	6	38	43		5	5	44	48		6	4	49	52		4	8	53	60		3	7	73	79
M6					0		6	9	26	34		1	3	35	37		2	10	38	47		3	8	48	55		5	4	56	59		4	9	61	69

Solucion Inicial Intancia 6X6 (mt06)

J	f	m	p	s	c	f	m	p	s	c																									
J1					0		3	1	1	1		1	3	2	4		2	6	9	14		4	7	26	32		6	3	35	37		5	6	38	43
J2					0		2	8	1	8		3	5	9	13		5	10	14	23		6	10	38	47		1	10	48	57		4	4	58	61
J3					0		3	5	14	18		4	4	19	22		6	8	48	55		1	9	63	71		2	1	72	72		5	7	73	79
J4					0		2	5	15	19		1	5	20	24		3	5	25	29		4	3	33	35		5	8	53	60		6	9	61	69
J5					0		3	9	30	38		2	3	39	41		5	5	44	48		6	4	56	59		1	3	60	62		4	1	63	63
J6					0		2	3	20	22		4	3	23	25		6	9	26	34		1	10	35	44		5	4	49	52		3	1	53	53

Figura 3.27 Solución inicial línea de tiempo sin optimizar.

3.5.2. Permutaciones en Línea de Tiempo

Explicación del algoritmo línea de tiempo:

Pseudocódigo Algoritmo Línea de Tiempo

```
Inicio Algoritmo 4 Línea de Tiempo

1.    Mientras no se haya permutado un par
2.        Generar intervalo aleatoriamente
3.        Mientras no se recorran todas las máquinas
4.            Si existe par adyacente
5                 Permutar
6         Fin Mientras
7     Fin Mientras 1 par permutado
Fin Algoritmo Línea de Tiempo
```

Paso 1. Mientras no se haya permutado un par.

El algoritmo de línea de tiempo tiene una limitación, es decir, si un trabajo se realiza como última operación en más de 2 máquinas el algoritmo línea de

tiempo no lo toma en cuenta, para que se realice la permutación el intervalo de tiempo tiene que atravesar el par adyacente.

Paso 2. Generar intervalo aleatoriamente.
De forma aleatoria se genera un número entre 1 y el makespan

Paso 3. Mientras no se recorran todas las máquinas.
La vecindad del algoritmo son las *n* máquinas de cada instancia, sólo se puede permutar un par adyacente por máquina.

Paso 4. Si existe par adyacente.
Al recorrer las máquinas y encontrarse un par adyacente que cumple con el criterio, es candidato a ser permutado.

Paso 5. Hacer la permutación intercambiar máquina con su tiempo de procesamiento.

Paso 6. Fin mientras se recorrieron todas las máquinas.

Paso 7. Fin mientras un par permutado.
Para evitar que no se haga ninguna permutación y se generen soluciones erróneas este ciclo se repite hasta que se permute un par adyacente como mínimo. En la gráfica de Gantt figura. 3.28 se puede ver representada la flecha con que la línea de tiempo atraviesa todas las máquinas y sus operaciones.

3.5.1 Representación Gráfica de la Estructura Línea de tiempo

La nueva estructura de vecindad, lo que pretende es intercambiar todas las operaciones que pasen en una línea de tiempo. La figura 3.28 tiene una línea de tiempo aleatorio entre 1 y 79 que representa el makespan, la flecha representa esa línea de tiempo que es el número 26 generado de forma aleatoria con el algoritmo 4 Línea de tiempo.

En la imagen el número 26 que es la línea de tiempo o intervalo, pasa por tres operaciones diferentes máquinas 3, 4 y 6, mientras que, en la máquina 3, la línea

de tiempo pasa por el trabajo 4, el cual es candidato a ser permutado por ser par adyacente sin tiempo de ocio y corresponde al trabajo 5. En la máquina 4, la línea de tiempo pasa entre los trabajos 6 y 1, por lo tanto, ese par de operaciones se permuta. En la maquina 6, la línea de tiempo pasa cuando inicia el trabajo 6, lo cual lo hace candidato a ser permutado con su par adyacente sin tiempo de ocio que es el trabajo 1.

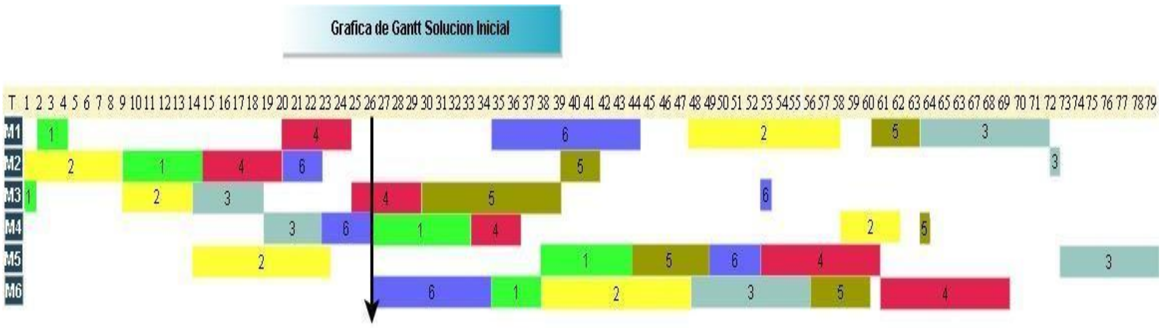


Figura 3.28. Gráfica de Gantt solución inicial con línea de tiempo

En la Fig. 3.29 Candidatos a permutar en línea de tiempo, se representan en rojo los candidatos a ser permutados EVLT.

Cadidatos Intercambio Intervalo de Tiempo

M	f	j	p	s	c	f	j	p	s	c																									
M1					0		1	3	2	4		4	5	20	24		6	10	35	44		2	10	48	57		5	3	60	62		3	9	63	71
M2					0		2	8	1	8		1	6	9	14		4	5	15	19		6	3	20	22		5	3	39	41		3	1	72	72
M3					0		1	1	1	1		2	5	9	13		3	5	14	18		4	5	25	29		5	9	30	38		6	1	53	53
M4					0		3	4	19	22		6	3	23	25		1	7	26	32		4	3	33	35		2	4	58	61		5	1	63	63
M5					0		2	10	14	23		1	6	38	43		5	5	44	48		6	4	49	52		4	8	53	60		3	7	73	79
M6					0		6	9	26	34		1	3	35	37		2	10	38	47		3	8	48	55		5	4	56	59		4	9	61	69

Fig. 3.29 Candidatos a permutar en línea de tiempo

En la siguiente figura 3.30, se muestran los trabajos ya permutados y la matriz lista para ser reagendada y para poder verificar su factibilidad. La factibilidad consiste en verificar que al momento de reagendar la nueva solución no se generen ciclos, en este caso, sí genera ciclos, ya que al momento de realizar más de una permutación adyacente cercana una de la otra genera el efecto inverso de cómo se rompen los ciclos.

Fig. 3.30 Candidatos a ser permutados

3.5.3 Solución Optimizada con Búsqueda Local Para Línea de Tiempo

Una vez aplicada la búsqueda local iterada a la nueva estructura de vecindad, se obtiene una solución óptima, ya que es una instancia muy pequeña y se puede encontrar fácilmente el resultado. La figura 3.31, muestra la misma solución Inicial que par adyacente ya optimizada con la estructura de línea de tiempo, ambas obtienen el resultado óptimo para esta instancia que es de 55 unidades de tiempo.

Fig. 3.31 Solución optimizada con línea de tiempo

3.5.4 Gráfica de Gantt Solución Optimizada Línea de Tiempo

La Figura 3.32 Grafica de Gantt, con Línea de tiempo presenta la solución optimizada de igual forma que se hizo con la de par adyacente los resultados son los mismos, ya que es el óptimo sólo se puede apreciar que las calendarizaciones o los tiempos de finalización son diferentes.

Figura 3.32 Grafica de Gantt con Línea de Tiempo.

4 RESULTADOS COMPUTACIONALES

4.1 Equipo de Cómputo Utilizado

A lo largo de este capítulo se encuentran los resultados obtenidos a partir del algoritmo propuesto. El algoritmo búsqueda local iterada fue ejecutado utilizando una computadora tipo laptop con las características que se muestran en la Tabla 4.1. Descripción del equipo utilizado.

PROCESADOR	MEMORIA	SISTEMA. OPERATIVO
Intel(R) Celeron(R) CPU N3350 @ 1.10GHz 1.10 GHz	4.00 GB	Windows 10

Tabla 4.1 Descripción del equipo utilizado

4.2 Resultados de Búsqueda Local Iterada

Los resultados que se muestran en esta sección se obtienen con las instancias simétricas de la Or-Lib. La Tabla 5 presenta resultados obtenidos con un algoritmo de búsqueda local iterada, aplicando dos tipos de estructura de vecindad para comparar la eficiencia y eficacia de la estructura de vecindad propuesta. Las columnas se encuentran listadas por tamaño de problema, instancia, cantidad de soluciones (S), tiempo (t), makespan (Ms) , iteraciones y en el caso de EVLT la cantidad de ciclos y sus porcentajes. La primera columna indica el tamaño de cada una de las instancias, inicia con la más pequeña que es 6 X 6 y termina con la más grande que es 20X20 todas simétricas. La segunda columna es el nombre de la instancia obtenida de la Or-library (1990), donde instancia es el nombre abreviado y utilizado por todos los investigadores. *S* indica la cantidad de soluciones aleatorias generadas, *t* es el tiempo en segundo que se toma en calendarizar el total de soluciones de *S*. La búsqueda local es la encargada de encontrar el mejor makespan. En cada permutación se puede generar otra solución que puede ser mejor, igual o peor, pero ese es el procedimiento de la búsqueda local iterada para

encontrar la mejor solución. Se ejecutó la búsqueda local iterada 2 horas a cada instancia es por eso que cada instancia aparece en 2 filas con diferentes resultados (1 hora de ejecución por fila). Se realizaron 2 horas de prueba por instancia para comparar los resultados y ver la diferencia, a excepción de mt06 que solo se realizó una vez debido a que el óptimo global se obtiene muy rápido, pero a pesar de la rapidez existe diferencia en tiempo de ejecución entre EVPA y EVLT. *Ms* es el makespan de la mejor solución obtenida por la búsqueda local. Para definir el número de iteraciones en el algoritmo de búsqueda local iterada, es importante tomar en cuenta el tamaño de la instancia del problema, ya que la fórmula *N = n*(m-1)* representa el tamaño de la vecindad o el número máximo de posibles permutaciones que pueden ser diferentes, donde cada permutación genera un nuevo vecino (nueva solución). Para la instancia de 6X6 su vecindad es 6 * 5 = 30 posibles soluciones distintas debido a las permutaciones que pueden ser diferentes. De igual manera, la fórmula aplica para las instancias de toda la tabla. Una de 10X10 el tamaño de la vecindad es 10*9 = 90 y para una de 20X20, el tamaño de la vecindad es 20*19 = 380. Para la EVLT el vecindario es diferente porque el tamaño de la vecindad es el número de máquinas, ya que máximo se realiza una permutación por máquina en el caso de una de 10X10 el vecindario tiene máximo 10 posibles permutaciones. Con este razonamiento, para instancias de 10X10 la búsqueda local genera *S* = 4500 soluciones iniciales las cuales se sometieron a la búsqueda local iterada con la siguiente fórmula *iteraciones = S*N*300, donde N = n*(m-1)*, e iteraciones indica el número total de permutaciones realizadas en la búsqueda local iterada, la constante de *300* es para ejecutar la búsqueda local iterada un tiempo aproximado de 1 hora. En una instancia de 10X10 se realizaron 121 500 000 permutaciones (iteraciones) en aproximadamente 1 hora. El mismo procedimiento se realiza para las instancias de 20X20, se aplica la misma fórmula de *iteraciones*, la única diferencia es que debido a su tamaño se ajustaron las soluciones generadas a *S* = 162 que se llevó aproximadamente 1 hora de ejecución, son pocas soluciones pero el objetivo es ver el comportamiento y la cantidad de ciclos que genera. La segunda mitad de la tabla corresponde a la estructura de vecindad propuesta llamada línea de tiempo (EVLT). la primer columna S es la cantidad de soluciones generadas. La columna t representa el tiempo en segundos que se tomó en realizar las soluciones en la columna S. La columna Ms es son las mejores soluciones encontradas incluyendo ya la búsqueda local. las iteraciones

tienen la misma fórmula. línea de tiempo tiene 2 columnas adicionales que son ciclos qué es la cantidad de ciclos que se generaron en relación con el número de iteraciones. La última columna porcentaje de ciclos representa el porcentaje de las soluciones en relación con las iteraciones. La Tabla 4.2, muestra con respecto a la eficacia que la mejor optimización con búsqueda local iterada fue realizada aplicando EVLT en casi todos los casos a excepción de uno solo en una ejecución (instancia seta3). Se muestra que la estructura de par adyacente no genera ciclos. Cruz-Chávez (2014) en su artículo menciona que no existen ciclos al hacer este tipo de permutaciones en pares adyacentes sin tiempo de holgura y esto queda demostrado con estos resultados. En el caso de EVLT, se observa que el porcentaje de ciclos encontrados con respecto al número de iteraciones realizadas por la búsqueda local, siempre es menor al 0.13% esto indica que la eficacia para generar soluciones factibles partiendo de la EVLT es muy buena y no requiere de utilizar de forma constante un procedimiento de corrección para obtener una solución factible como es el algoritmo de armonización global de Nakano y Yamada, esto permite que la eficiencia en la ejecución de búsqueda local sea muy buena como se observa en la columna de tiempo de ejecución la cual es muy parecida en tiempo de ejecución a la búsqueda local ejecutada con EVPA (columna 8 vs columna 4).

.

		Par Adyacente (EVPA)				Línea de Tiempo (EVLT)					
Tamaño	Instancia	S	t	Ms	Iteraciones	S	t	Ms	Iteraciones	Ciclos	%Ciclos
6X6	(mt06)	1500	146	55	13500000	100	13	55	900000	1	0.01
10X10	abz5	4500	3535	1422	121500000	4500	3519	1378	121500000	26466	0.022
10X10	abz5	4500	3499	1468	121500000	4500	3522	1390	121500000	26213	0.021
10X10	(mt10)	4500	3661	1275	121500000	4500	3798	1253	121500000	25891	0.020
10X10	(mt10)	4500	3798	1216	121500000	4500	3816	1200	121500000	26180	0.021
10X10	seta1	4500	3768	1224	121500000	4500	3871	1216	121500000	26930	0.022
10X10	seta1	4500	3681	1112	121500000	4500	3690	1089	121500000	19695	0.016
10X10	seta2	4500	3691	1094	121500000	4500	3696	1082	121500000	19923	0.016
10X10	seta2	4500	3545	938	121500000	4500	3583	869	121500000	12595	0.010
10X10	seta3	4500	3548	922	121500000	4500	3579	874	121500000	12282	0.010
10X10	seta3	4500	3696	1016	121500000	4500	3727	1018	121500000	16035	0.013
10X10	seta4	4500	3727	1021	121500000	4500	3733	1006	121500000	15850	0.013
10X10	seta4	4500	3646	1016	121500000	4500	3522	988	121500000	26085	0.021

10X10	seta5	4500	3544	1026	121500000	4500	3507	972	121500000	25885	0.021
10X10	seta5	4500	3648	1084	121500000	4500	3673	1044	121500000	24316	0.020
10X10	(BCI2)	4500	3719	1076	121500000	4500	3676	1051	121500000	24727	0.020
10X10	(BCI2)	4500	3758	1405	121500000	4500	3649	1360	121500000	20268	0.016
10X10	(MON2)	4500	3740	1378	121500000	4500	3641	1342	121500000	20441	0.016
10X10	(MON2)	4500	3664	1080	121500000	4500	3719	1068	121500000	14620	0.012
20X20	NY1	162	3480	1534	18468000	162	3296	1416	18468000	14669	0.073
20X20	NY1	162	3469	1515	18468000	162	3297	1399	18468000	14441	0.078
20X20	NY2	162	3481	1615	18468000	162	3265	1478	18468000	23068	0.124
20X20	NY2	162	3477	1562	18468000	162	3258	1454	18468000	21934	0.118
20X20	NY3	162	3679	1627	18468000	162	3354	1437	18468000	15849	0.085
20X20	NY3	162	3648	1690	18468000	162	3365	1427	18468000	14422	0.078
20X20	NY4	162	3471	1580	18468000	162	3523	1582	18468000	15360	0.083
20X20	NY4	162	3643	1708	18468000	162	3520	1550	18468000	15786	0.085

Tabla 4.2 Resultados obtenidos con algoritmo de búsqueda local

La figura 4.1 presenta la gráfica de resultados de la Instancia (Mt06) 6X6, la cual muestra la comparación de soluciones generadas con EVPA y EVLT. Con la estructura de vecindad par adyacente se tomó 1500 soluciones iniciales sometidas a la búsqueda local iterada con la misma fórmula $N = n^* (m\text{-}1) * 300$ o (6*5*300) iteraciones, para encontrar el óptimo que son 55 unidades de tiempo. A la estructura de vecindad línea de tiempo le tomó 100 soluciones iniciales sometidas a la búsqueda local iterada con la misma fórmula de $N = n * (m\text{-}1) * 300$ para encontrar el tiempo óptimo de 55. Esto demuestra que realizar más de una permutación por iteración, para esta instancia línea de tiempo es más eficaz.

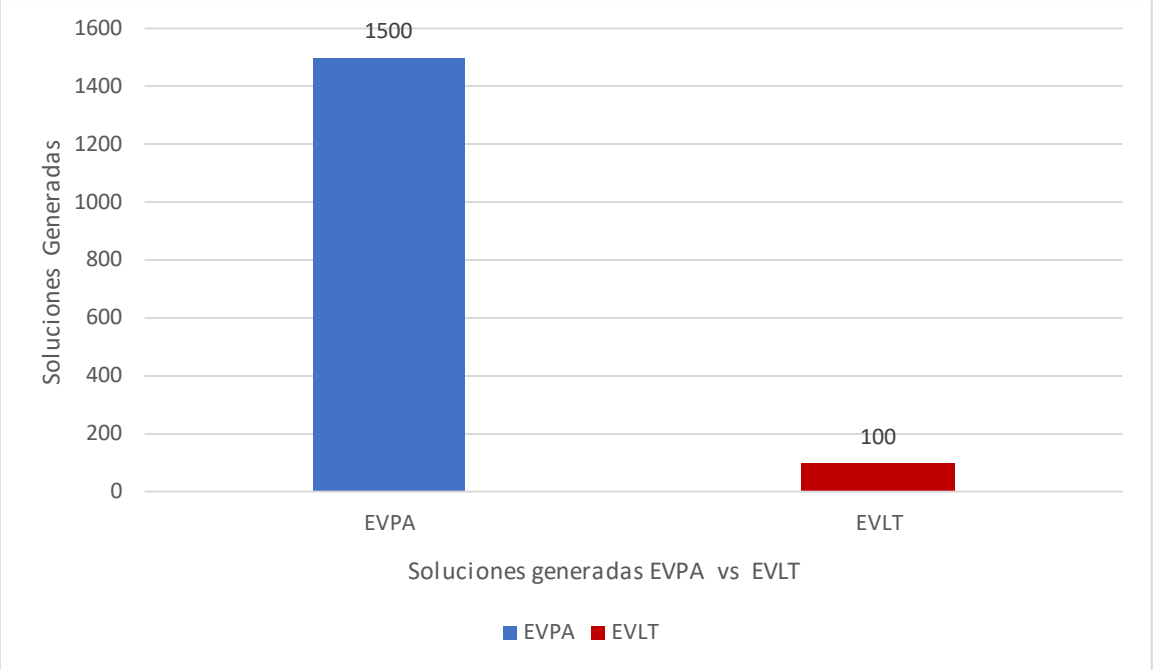


Figura 4.1 Comparación de soluciones en la búsqueda local iterada. Instancia 6x6

La figura 4.2 presenta la gráfica de resultados de la Instancia (Mt06) 6X6, se presenta la comparación del tiempo que le toma a cada una de las estructuras de vecindad para encontrar la solución óptima. La EVPA se muestra en azul y la EVLT en rojo. A la EVPA le tomo 146 segundos en encontrar la solución óptima. A la EVLT le tomo 13 segundos encontrarla. Esto demuestra que realizar más de una permutación por iteración, en la mayoría de los casos el algoritmo de línea de tiempo en más eficiente para esta instancia.

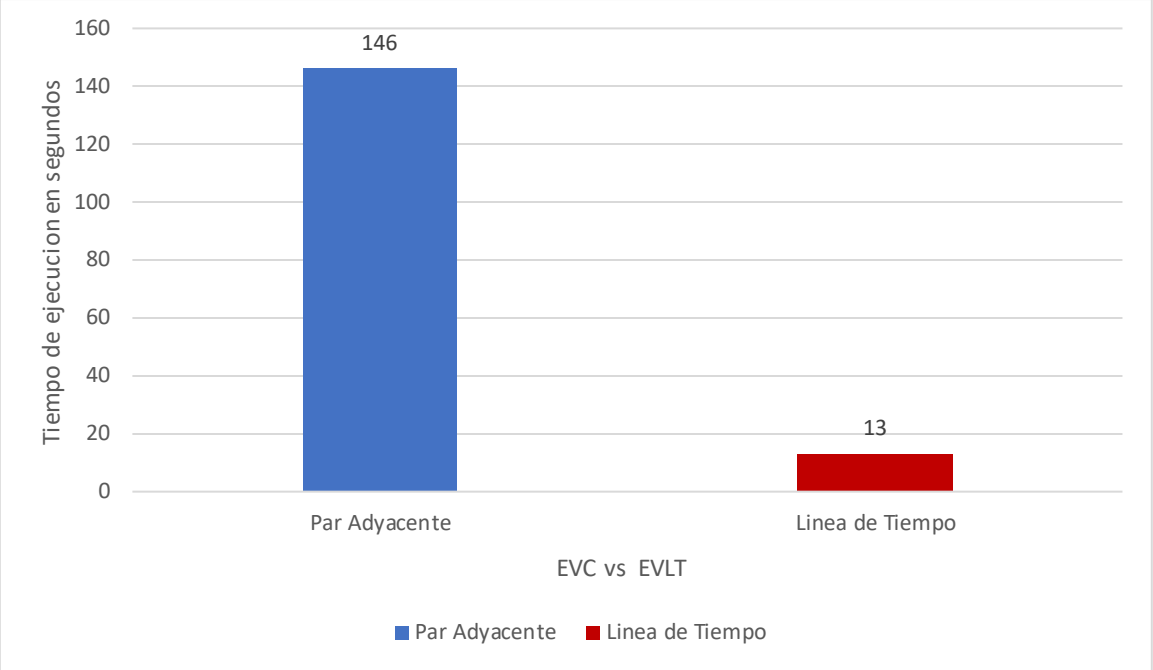


Figura 4.2. Comparación del tiempo requerido en la búsqueda local iterada. Instancia 6x6.

La figura 4.3 muestra la gráfica con 8 instancias del mismo tamaño de 10X10 que compara resultados de las EVPA y EVLT utilizando el mismo número de iteraciones en la búsqueda local para cada ejecución de cada problema. EVPA se encuentra en azul y EVLT en rojo. Al comparar las 8 instancias se observa en los resultados en el makespan de ambas estructuras que los resultados obtenidos de las estructuras de vecindad por cada problema son muy cercanos, no es destacada la diferencia que hay entre ambas soluciones, pero se puede observar que en la mayoría de los casos gana EVLT a excepción de la instancia SETA3, donde EVPA le gana por 2 unidades de tiempo a EVLT. Esto puede ser porque EVPA tuvo una buena solución de inicio en la búsqueda local iterada y EVLT tuvo una mala y no la pudo superar en el lapso de tiempo en que se ejecutó. A pesar de que cada par adyacente no genera ciclos en EVPA y EVLT si los genera, los resultados son mejores en EVLT, aun cuando en EVLT se tiene que llamar el algoritmo de armonización global de Nakano y Yamada (1991) para poder seguir calendarizando.

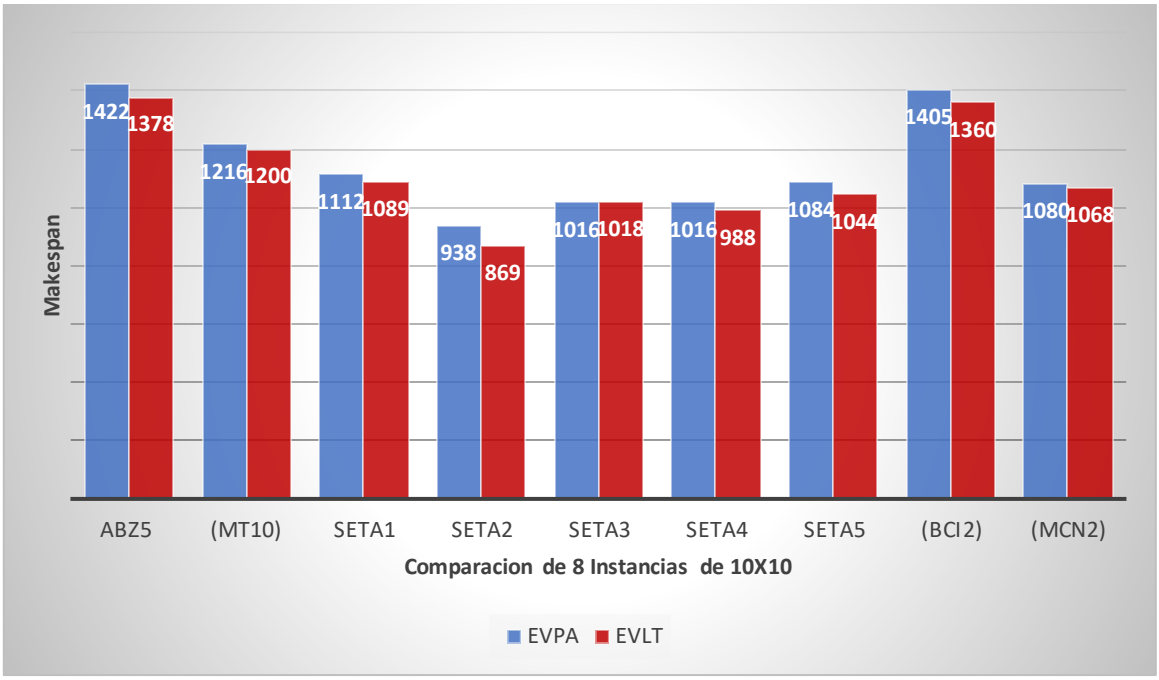


Figura 4.3 Comparación del makespan Instancias 10X10

Con la finalidad de comparar los resultados en instancias de tamaño grande, la figura 4.4 presenta la gráfica del makespan obtenido para diferentes instancias de tamaño grande. Se presentan los resultados de las instancias de 20X20, ambas instancias tienen el mismo tiempo de ejecución cercano a una hora y el mismo número de soluciones generadas en la búsqueda local iterada con la misma cantidad de iteraciones. Se presentan cuatro instancias de 20X20 para evaluar la eficacia. EVPA es la barra en azul y EVLT la barra en rojo. Se observa que la estructura de vecindad par adyacente se ve superada por la estructura de línea de tiempo en la mayoría de los casos a excepción de la instancia NY4, donde EVPA le supera a EVLT con 2 unidades menos de tiempo. (El mismo comportamiento que tuvo la instancia de SETA3 en las estructuras de 10X10 (ver figura 4.3), lo que hace pensar que hay un comportamiento similar en algún momento o bien esto puede ser por que la estructura de vecindad de par adyacente haya tenido una buena solución inicial a diferencia de la EVLT, que pudo haber tenido una mala solución inicial y no la pudo superar en el lapso de 1 hora qué es el tiempo aproximado que duró cada una de las pruebas.

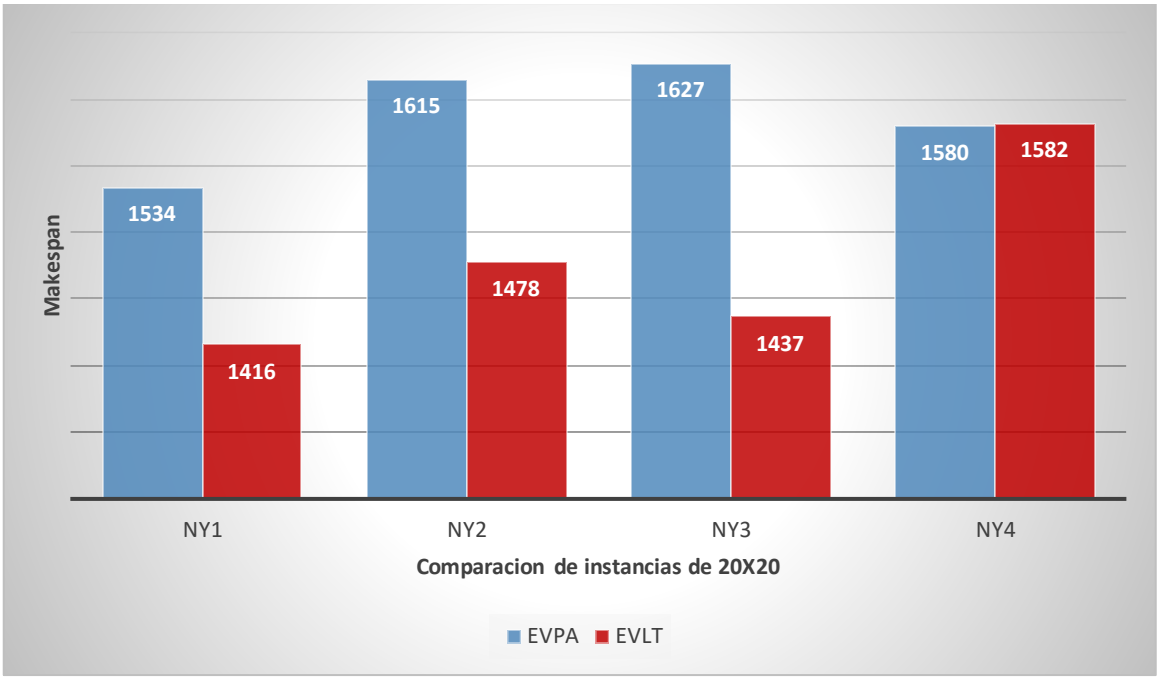


Figura 4.4 Diferencias en makespan EVPA vs EVLT

En la figura 4.5, se comparan dos instancias de problemas con respecto a los ciclos generados cuando se aplica la búsqueda local iterada, las instancias utilizados son una de 10X10 contra una instancia de 20X20, ambas soluciones se muestran en rojo y ambas aplican EVLT. La lógica indica que una instancia que es la mitad del tamaño de la otra, se obtendrían la mitad de los ciclos, pero los resultados no son los que se esperaban. La figura 4.5 muestra la cantidad de ciclos generados en la instancia de 10x10 y la otra de 20X20, en ambas se generó 162 soluciones iniciales con EVLT y se ejecutó búsqueda local iterada. En la instancia de 10X10 se generaron 671 ciclos contra 14,621 ciclos generados en la instancia de 20x20, una cantidad mínima de ciclos en la instancia pequeña con respecto a los ciclos generados para la instancia grande, esto puede ser debido a que las permutaciones que se realizan en la línea de tiempo para la instancia grande son de un número mayor en la mayoría de los casos lo cual da la posibilidad de la formación de un mayor número de ciclos.

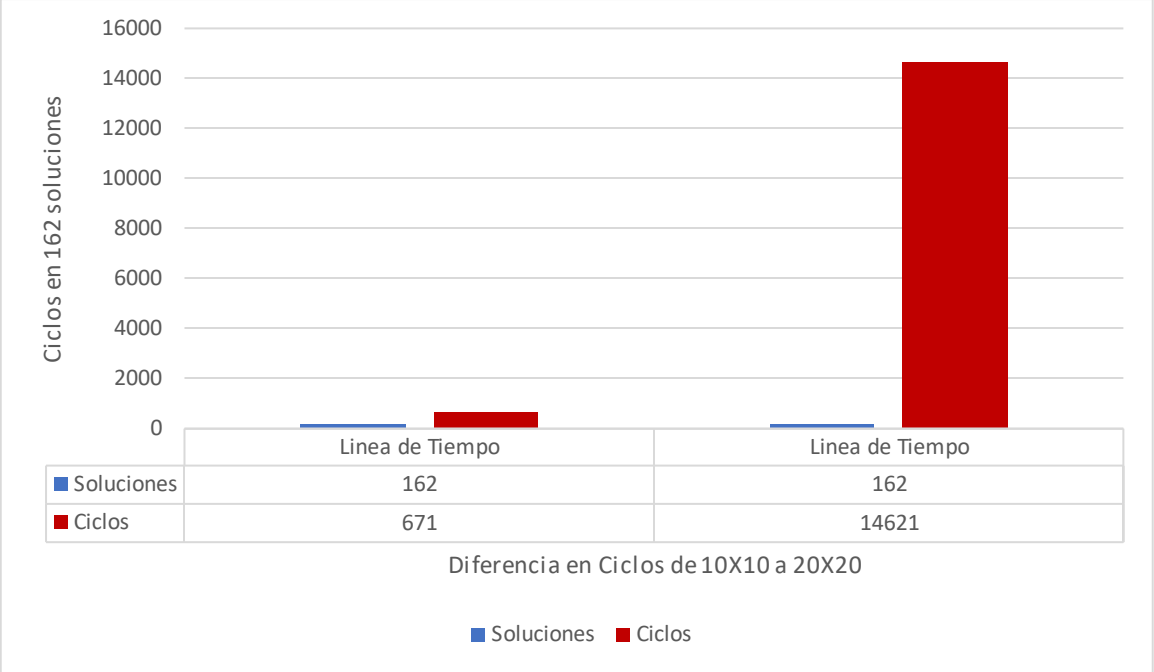


Fig. 4.5 Crecimiento de Ciclos en EVLT

4.3 Comportamiento del algoritmo de búsqueda local Iterada

El algoritmo de búsqueda local iterada, tal como se explicó a detalle en el capítulo 3, está conformado por 3 fases, a través de las cuales se busca mejorar progresivamente la solución inicial, hasta llegar a la mejor solución con la búsqueda local. Al momento de la experimentación se encuentran ciclos al aplicar EVLT, esto se puede deber a que se hace más de una permutación por iteración, una permutación por máquina. En una matriz de 6 máquinas existen hasta 6 posibilidades para permutar en una sola iteración. En el caso cuando se aplica EVPA en el algoritmo de búsqueda local iterada, no se encuentra ningún ciclo, lo cual demuestra bajo esta experimentación lo encontrado en Cruz-Chávez, (2014), pero aplicando en este trabajo búsqueda local iterada en lugar de la heurística de recocido simulado.

5 CONCLUSIONES Y TRABAJOS FUTUROS

5.1 Conclusión

De acuerdo con el objetivo planteado, se presentó una nueva estructura de vecindad aplicada a una búsqueda local iterada para resolver el problema de calendarización. El procedimiento en su primera fase consistió en generar una solución inicial aleatoria factible. En la segunda fase se implementó el algoritmo de calendarización y, por último, la búsqueda local iterada para optimizar la solución y después se procede a comparar los resultados. Se comprobó que al hacer más de una permutación en una solución aplicando EVLT se encuentra un porcentaje mínimo de ciclos en las soluciones generadas, inferior al 0.13% en instancias grandes y aplicando la VLPA no se obtienen ciclos.

A pesar de que la nueva estructura de vecindad propuesta genera ciclos, el tiempo que toma en computarizar los resultados son menores que al aplicar EVPA y también la optimización de resultados es mejor. En base a los resultados obtenidos y mostrados, se llega a la conclusión de que EVLT es más eficiente y eficaz que EVPA cuando se aplica en el algoritmo de búsqueda local iterada.

5.2 TRABAJOS FUTUROS

1) Desarrollar un algoritmo que obtenga de forma explícita los caminos de las múltiples rutas críticas, para utilizar línea de tiempo y solo permutar los pares que pertenecen a las rutas críticas. De tal manera que sea más eficiente esta estructura de vecindad.

2) Realizar 30 pruebas por cada instancia para hacer una comparación estadística.

3) Utilizar una mejor metaheurística como recocido simulado

REFERENCIAS

Johnson (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

Or-Library (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 website: https://www.brunel.ac.uk/~mastjjb/jeb/info.html

Festa (2014). A brief introduction to exact, approximation, and heuristic algorithms for solving hard combinatorial optimization problems. 2014

Nakano y Yamada (1991). Conventional genetic algorithm for job-shop problems. In M.K. Kenneth, & L. B. Booker (Ed.), Proceedings of the 4th International Conference on Genetic Algorithms (pp. 474–479). Morgan Kaufmann.

Manne. (1960) On the Job-Shop Scheduling Problem. Operations Research, 8, 219-223. http://dx.doi.org/10.1287/opre.8.2.219

Zhi y Yingjian (2022) Minimizing the makespan and carbon emissions in the green flexible job shop scheduling problem with learning effects https://doi.org/10.1038/s41598-023-33615-z

Maryam y Nguyen (2017) Journal of Applied Mathematics and Physics Vol.5 No.11, November 13, 2017 DOI: 10.4236/jamp.2017.511177

Jorapur et al (2016) A Promising Initial Population Based Genetic Algorithm for Job Shop Scheduling Problem. Journal of Software Engineering andApplications, 9, 208-214. doi:10.4236/jsea.2016.95017.

Papadimitriou (1998).” Combinatorial Optimization Algorithms and Complexity”. ISBN. 0-486-40258-4. USA. 1998

Anand y Panneerselvam (2015) Literature Review of Open Shop Scheduling Problems. Intelligent Information Management, 7, 33-52. doi: 10.4236/iim.2015.71004.

Cruz-Chávez (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

Levy et al (1963). The ABCs of the Critical Path Method. Harvard Business Review. Retrieved from https://hbr.org/1963/09/the-abcs-of-the-critical-path-method

Mirshekarian, S. &. (2016). Correlation of job-shop scheduling problem features with scheduling efficiency. Expert Systems with Applications, 62, 131–147. doi:10.1016/j.eswa.2016.06.014

Zhang et al (2019). Review of job shop scheduling research and its new perspectives under Industry 4.0. Journal of Intelligent Manufacturing. 30, 1809–1830. doi:10.1007/s10845-017-1350-2

Bisbal Riera Jesús (2009) "Manual de Algorítmica". Recursividad, Complejidad y Diseño de Algoritmos. Ed. UOC. Primera Edición en Castellano. ISBN: 978-84-9788-027-5. Pp.54-59. Barcelona. 2009.

Sörensen et al (2018). A History of Metaheuristics. In R. Martí, P. Panos,& M. Resende (Eds.), Handbook of Heuristics (pp. 1-18). Springer, Cham. doi:doi.org/10.1007/978-3-319-07153-4_4-1

Arisha (2001) El Baradie, M.:Job Shop Scheduling Problem: an Overview. International Conference for Flexible Automation and Intelligent Manufacturing (FAIM 01),Dublin, Ireland, July 2001, pp 682 – 693.

Liu et al (2003) "Many-Objective Job-Shop Scheduling: A Multiple Populations for Multiple Objectives- Based Genetic Algorithm Approach," in IEEE Transactions on Cybernetics, vol. 53, no. 3, pp. 1460-1474, March 2023, doi: 10.1109/TCYB.2021.3102642.

Syarif et al (2021). Performance Evaluation of Various Heuristic Algorithms to Solve Job Shop Scheduling Problem (JSSP). International Journal of Intelligent Engineering doi:doi.org/10.22266/ijies2021.0430.30

Satake et al (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

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

Rabadi (2016). Heuristics, Metaheuristics and Approximate Methods in Planning and Scheduling (Vol. 236). Springer, Cham. doi:doi.org/10.1007/978-3319-26024-2

Tian X, Liu X, Zhang H, Sun M, Zhao (2020) A DNA algorithm for the job shop scheduling problem based on the Adleman-Lipton model. PLoS ONE 15(12): e0242083. https://doi.org/10.1371/journal.pone.0242083he Adleman-Lipton

Jorapur et al (2016) A Promising Initial Population Based Genetic Algorithm for Job Shop Scheduling Problem. Journal of Software Engineering and Applications, 9, 208-214. doi:10.4236/jsea.2016.95017. 63

APENDICE

Sección I

1) Declaración de estructuras maquina y trabajos en lenguaje de programación C, utilizadas para obtener la calendarización entre ambas estructuras de vecindad.

```
typedef struct {
    int  m;
    int  p;
    int  s;
    int  c;
    int  f;
}Trabajo;


typedef struct{
    int  j;
    int  p;
    int  s;
    int  c;
    int  f;
} Maquina;
```

Sección II

2) Declaración de estructuras los valores de N y M son constantes con un valor de 51

```
Trabajo trab[N][M];  //Declaración de estructuras
Maquina maq[N][M];
```

Sección III

La función de factibilidad lleva a cabo el algoritmo 1 de calendarización y al detectar ciclos llama al algoritmo de armonización global, para generar una solución factible.

2)
```
int factibilidad(int n,int m,  Maquina (*maq)[M],Trabajo  (*trab)[M]){

    int mk, i, j, nj, k, my = 0, agendada = 0, ciclo;          //Variables
    float contadorDeCiclos = 0;
```

```
        for(;;){                        //Ciclo infinito mientras no se agenden todas
           ciclo = 1;                     //Detector de ciclos
           for(i = 1; i <= m; i++ ){      // Recorrer maquinas
                for(j = 1; j <= n; j++ ){   //Recorrer  trabajos
                  if(maq[i][j].f == 0){      //Si la maquina no está agendada entonces
                     nj = maq[i][j].j;          //Maquina a buscar
                     for(k = 1; k <= n; k++){    //Buscar maquina
                       if(trab[nj][k].f == 0){        //No esta agendado el trabajo
                          if(trab[nj][k].m == i){   //Coinciden
                             //Buscar mayor
                             my = (maq[i][j-1].c  >= trab[nj][k-1].c)?  maq[i][j-1].c:
                                    trab[nj][k-1].c;
                             //Agendar
                             maq[i][j].s = my + 1;    //Tiempo de inicio
                             maq[i][j].c = maq[i][j].s  + maq[i][j].p-1; //Tiempo determino
                             maq[i][j].f = 1;             //Agendado
                             trab[nj][k].s = maq[i][j].s;     //Copiar valores a trabajos
                             trab[nj][k].c = maq[i][j].c;
                             trab[nj][k].flag = 1;

                            if(mk < maq[i][j].c){  //Actulizar makespan
                              mk = maq[i][j].c;
                            }
                          agendada++;    //Incrementar agendadas
                          ciclo = 0;          // Ya existe ciclo en esta iteración
                          break;           //Una vez agendada salir de trabajos
                        }else{
                          break;          //Si no coinciden maq y trab ya no buscamos
                        }
                      }                    //Fin de trabajo  agendado
                    }                      //Fin de búsqueda de maquina candidata
                    break;                //Vamos a la siguiente maquina
                  }                       //Fin si de trabajo
                }                         //Cierre trabajos
             }                            //Cierre maquinas
      if(ciclo == 1){              //De existir entonces
           armonizacionGlobal(n,m,maq,trab);  //Llamar algoritmo de armonización
           contadorDeCiclos++;
       }                           //Fin si de ciclo
      if(agendada == n*m){         //Todos agendados
         break;
      }                           //Fin agendada
   }   //Fin for infinito
   return  mk;                    //Retornar valor de makespan
  }       // Fin de función factibilidad
```

UNIVERSIDAD AUTONOMA DEL ESTADO DE MORELOS
FACULTAD DE CONTADURÍA, ADMINISTRACIÓN E INFORMÁTICA

Cuernavaca, Morelos, 14 de junio del 2023

DR. FELIPE DE JESÚS BONILLA SÁNCHEZ

DIRECTOR DE LA FCAel DE LA U.A.E.M.

PRESENTE

Por este conducto, me permito informar a usted que la Tesis denominada: **"Evaluación de la Factibilidad de Soluciones Generadas con Estructuras de Vecindad para el Problema de Talleres de Manufactura**", que presenta el **C. Pedro Bello Campusano,** Pasante de la carrera **Licenciatura en Informática.**

A mi juicio, cumple con los requisitos de metodología y contenido, por lo que no tengo inconveniente en otorgarle mi voto aprobatorio para que continúe con los trámites de titulación correspondientes.

Sin otro particular, aprovecho la ocasión para enviarle un cordial saludo.

A T E N T A M E N T E

_____

Dr. Martín Heriberto Cruz Rosales

(Nombre y firma)

Se expide el presente documento firmado electrónicamente de conformidad con el ACUERDO GENERAL PARA LA CONTINUIDAD DEL FUNCIONAMIENTO DE LA UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MORELOS DURANTE LA EMERGENCIA SANITARIA PROVOCADA POR EL VIRUS SARS-COV2 (COVID-19) emitido el 27 de abril del 2020.

El presente documento cuenta con la firma electrónica UAEM del funcionario universitario competente, amparada por un certificado vigente a la fecha de su elaboración y es válido de conformidad con los LINEAMIENTOS EN MATERIA DE FIRMA ELECTRÓNICA PARA LA UNIVERSIDAD AUTÓNOMA DE ESTADO DE MORELOS emitidos el 13 de noviembre del 2019 mediante circular No. 32.

Sello electrónico

MARTIN HERIBERTO CRUZ ROSALES | Fecha:2023-06-14 12:28:37 | Firmante

R7926AVevJIwyRp3QtvOFf0OPpeHmoAWtLgWt+GjRkWWvaAuboZ73bjQl5cV/+z538y+KxCYSS7iIprWoN2iVTlg4RJygNhDxO1KlDuSaGaM6OgWvyY2YFliGMSm54fzVwzmvEI0rSZcLGsSLgpbfk17MXSRZidEWg5Rep2D0oKpqf5f0C9dt+JsrKOFtaHipBLMuVU+dGqLKqWSvxUcOI4zd6YV1EgXQn+/9fFKHBD6JHfs3xfWSlm8UBdU9AaSWJFFhnIep/g3cWBGXZ+2Oz8Z4DTNLCC/kO9E1TAsDhOeQxVQgGrObIXyo6mFZhC4r+wN12FAXhxgXSP6qBvLxQ==

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

Gl1YKgfZ3

https://efirma.uaem.mx/noRepudio/aeVOwg8S4cYbMlTokK9zLekG9RLnLakm

DR. FELIPE DE JESÚS BONILLA SÁNCHEZ

DIRECTOR DE LA FCAel DE LA U.A.E.M.

PRESENTE

Por este conducto, me permito informar a usted que la Tesis denominada: **"Evaluación de la Factibilidad de Soluciones Generadas con Estructuras de Vecindad para el Problema de Talleres de Manufactura**", que presenta el **C. Pedro Bello Campusano,** Pasante de la carrera **Licenciatura en Informática.**

A mi juicio, cumple con los requisitos de metodología y contenido, por lo que no tengo inconveniente en otorgarle mi voto aprobatorio, para que continúe con los trámites de titulación correspondientes.

Sin otro particular, aprovecho la ocasión para enviarle un cordial saludo.

A T E N T A M E N T E

_____

DR. FEDERICO ALONSO PECINA

Se expide el presente documento firmado electrónicamente de conformidad con el ACUERDO GENERAL PARA LA CONTINUIDAD DEL FUNCIONAMIENTO DE LA UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MORELOS DURANTE LA EMERGENCIA SANITARIA PROVOCADA POR EL VIRUS SARS-COV2 (COVID-19) emitido el 27 de abril del 2020.

El presente documento cuenta con la firma electrónica UAEM del funcionario universitario competente, amparada por un certificado vigente a la fecha de su elaboración y es válido de conformidad con los LINEAMIENTOS EN MATERIA DE FIRMA ELECTRÓNICA PARA LA UNIVERSIDAD AUTÓNOMA DE ESTADO DE MORELOS emitidos el 13 de noviembre del 2019 mediante circular No. 32.

Sello electrónico

FEDERICO ALONSO PECINA | Fecha:2023-06-14 13:34:28 | Firmante

84pew+XhvicaFnf9RuyFqNni9b2EUUipwwwTu+moQwBNIj7eCInUXKiw3Q2VgICW7VWs5KdgAYM4DSg8UC8h/ZPhcunKW69pi8jKA4xD++XbH0VM50FgKn1zZnWtbPPC6XIyGaLZNETR1QekVPUUizYLgVh5aNFuT7Sz1XFhyDqF7fciTo+Fmyz0t2pAmNDabO1onzfCvbzzyeH6qmaMXJ8llZYYIFsm9sO2tCWldaS56XBR/Up3y5vUy6fdQvsx/wM2fMqbW7xUu7ZvpoRhZmhYC60yLYexQ9TMCPDvHId1Qe3eRPqdbvfNJOdXFfYwf6zkKgLGTiN/1PaOZakEjw==

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

2EpxgDkWM

https://efirma.uaem.mx/noRepudio/jt7hZEDTaa5moWaGpTSYY2nPwDlKI9LA

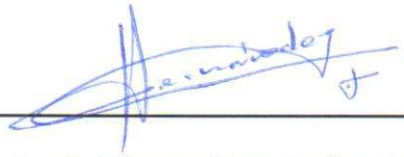
Cuernavaca, Morelos a _14__ de _Junio_ del _2023_

DR. FELIPE DE JESÚS BONILLA SÁNCHEZ
DIRECTOR DE LA FCAel DE LA U.A.E.M.
PRESENTE

Por este conducto, me permito informar a usted que la Tesis denominada: **"Evaluación de la Factibilidad de Soluciones Generadas con Estructuras de Vecindad para el Problema de Talleres de Manufactura"**, que presenta el **C. Pedro Bello Campusano,** Pasante de la carrera **Licenciatura en Informática.** A mi juicio, cumple con los requisitos de metodología y contenido, por lo que no tengo inconveniente en otorgarle mi voto aprobatorio, para que continúe con los trámites de titulación correspondientes.

Sin otro particular, aprovecho la ocasión para enviarle un cordial saludo.

A T E N T A M E N T E



Dr. José Alberto Hernández Aguilar

Se expide el presente documento firmado electrónicamente de conformidad con el ACUERDO GENERAL PARA LA CONTINUIDAD DEL FUNCIONAMIENTO DE LA UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MORELOS DURANTE LA EMERGENCIA SANITARIA PROVOCADA POR EL VIRUS SARS-COV2 (COVID-19) emitido el 27 de abril del 2020.

El presente documento cuenta con la firma electrónica UAEM del funcionario universitario competente, amparada por un certificado vigente a la fecha de su elaboración y es válido de conformidad con los LINEAMIENTOS EN MATERIA DE FIRMA ELECTRÓNICA PARA LA UNIVERSIDAD AUTÓNOMA DE ESTADO DE MORELOS emitidos el 13 de noviembre del 2019 mediante circular No. 32.

Sello electrónico

JOSE ALBERTO HERNANDEZ AGUILAR | Fecha:2023-06-14 10:17:39 | Firmante

c5ll4+f34+RrnjH7dRFeZD9D7JeJK/KLrnJLtPYgdIqNGTUc/Lp/Bbm+G6AHsvqGehg3/4t1Ldp47nr8mrj3ESiNgmWetkw6Nm3NV/NJcp4J3bboVzQD+98h5UiJbqMVExoTC9cM64OnF3jda9L+1zHwlI8jZUDbdTidGE78HUmForIeLcAxlif/fmRYkBh2HYAx230P0TrOZW/SDiahhBXYS1uTyn+rH55fA64hZjZ+Xya+azHouJKGVJioW8CduoNrCJosA5YEvzM5hiy2UJ9j3JLZSqcDwfSSPU5F4x0TyY8IjUNybBpq+utAbB9sGgsomRGBzdqu0EgCVs2OQg==

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

8AFILM6CB

https://efirma.uaem.mx/noRepudio/pui1xgHZaHWf49W2HziG1jtUo1dJqPoh

Cuernavaca, Morelos a 14 de junio del 2023

DR. FELIPE DE JESÚS BONILLA SÁNCHEZ
DIRECTOR DE LA FCAel DE LA U.A.E.M.
PRESENTE

Por este conducto, me permito informar a usted que la Tesis denominada: "**Evaluación de la Factibilidad de Soluciones Generadas con Estructuras de Vecindad para el Problema de Talleres de Manufactura**", que presenta el **C. Pedro Bello Campusano** pasante de la carrera **Licenciatura en Informática** a mi juicio, cumple con los requisitos de metodología y contenido, por lo que no tengo inconveniente en otorgarle mi voto aprobatorio, para que continúe con los trámites de titulación correspondientes.

Sin otro particular, aprovecho la ocasión para enviarle un cordial saludo.

A T E N T A M E N T E

_____

Dra. Mireya Flores Pichardo

Se expide el presente documento firmado electrónicamente de conformidad con el ACUERDO GENERAL PARA LA CONTINUIDAD DEL FUNCIONAMIENTO DE LA UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MORELOS DURANTE LA EMERGENCIA SANITARIA PROVOCADA POR EL VIRUS SARS-COV2 (COVID-19) emitido el 27 de abril del 2020.

El presente documento cuenta con la firma electrónica UAEM del funcionario universitario competente, amparada por un certificado vigente a la fecha de su elaboración y es válido de conformidad con los LINEAMIENTOS EN MATERIA DE FIRMA ELECTRÓNICA PARA LA UNIVERSIDAD AUTÓNOMA DE ESTADO DE MORELOS emitidos el 13 de noviembre del 2019 mediante circular No. 32.

Sello electrónico

MIREYA FLORES PICHARDO | Fecha:2023-06-14 09:09:59 | Firmante

BH4M6sIDPVw1YvC5xkA28d1zTkDbnm66UkrrrbZ0aI6g5OMDlBeiygHp9avEfdggmFWjv0IkPU2mCaBnwW3343/oR1sopxMV28TdnC3xVXA/7BoEzhWYmGsznV58v/toIY89vuE
DovQDk26MD73dOjgcPOYR+nOF4KVJmc32AQIw8rOVzeDpAYLvg9GwXQXnhUVoVcv3XpAK3awGVAvwKEzn/x1hboyZxazjjAyjCfMmw7EXB1EshcMFsI4J3aRp1Sok5oVJgx2
N6/hlseIygsqIJvFvHMhujXi+nsEIbzvNqfSF+o3wF7maxRX2jiv8AknQ7L0RVkI3nFPCBmH8mw==

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

YlKBm3AsN

https://efirma.uaem.mx/noRepudio/JzUcq5ugncK6PPYSqADR2K8Cm9vHGm9F

Cuernavaca, Morelos a 14 de Junio del 2023

DR. FELIPE DE JESÚS BONILLA SÁNCHEZ
DIRECTOR DE LA FCAel DE LA U.A.E.M.
PRESENTE

Por este conducto, me permito informar a usted que la Tesis denominada: **"Evaluación de la Factibilidad de Soluciones Generadas con Estructuras de Vecindad para el Problema de Talleres de Manufactura**", que presenta el **C. Pedro Bello Campusano,** Pasante de la carrera **Licenciatura en Informática.**
A mi juicio, cumple con los requisitos de metodología y contenido, por lo que no tengo inconveniente en otorgarle mi voto aprobatorio, para que continúe con los trámites de titulación correspondientes.

Sin otro particular, aprovecho la ocasión para enviarle un cordial saludo.

A T E N T A M E N T E

DR. ULISES CRUZ JACOBO

Se expide el presente documento firmado electrónicamente de conformidad con el ACUERDO GENERAL PARA LA CONTINUIDAD DEL FUNCIONAMIENTO DE LA UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MORELOS DURANTE LA EMERGENCIA SANITARIA PROVOCADA POR EL VIRUS SARS-COV2 (COVID-19) emitido el 27 de abril del 2020.

El presente documento cuenta con la firma electrónica UAEM del funcionario universitario competente, amparada por un certificado vigente a la fecha de su elaboración y es válido de conformidad con los LINEAMIENTOS EN MATERIA DE FIRMA ELECTRÓNICA PARA LA UNIVERSIDAD AUTÓNOMA DE ESTADO DE MORELOS emitidos el 13 de noviembre del 2019 mediante circular No. 32.

Sello electrónico

ULISES CRUZ JACOBO | Fecha:2023-06-14 11:01:31 | Firmante

EoaBK/fGdOBuVPaD6rZsPaGMizZTST9TN+/dvpxE1SOnht7u2vqCpUmnWsjG9lcgRzmU8cGhbH1Glmt4oCqJ/5N5ycMHA9DL4/EnuMpGSyDPjxYRQe5A3HQ0QawHIgI79BZ8+qyvFpTLtmw1kXMTMRkw/WAYFEMtGL1i/g2/w9Fc+YVWZCGebRPW7CjfxiTAED4/xktxx1+AC981zIDjOLvS4HjIoXTiQ5CzIG9INk+2385lBDuzpBjGHLYbFoEPs79MuXgkCkF6ug20MkwFR8JKQH6BIcYlF8OKZ7eao9cNWPAGpa6655GvUcRActLJf5wDlggFygWOBiPtF+J5vg==

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

DwMZEueHr

https://efirma.uaem.mx/noRepudio/l8v3HhpPkYFREtN2EH3PqKQ5s3c9XoEL