UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MORELOS

FACULTAD DE CONTADURÍA, ADMINISTRACIÓN E INFORMÁTICA

Maestría en optimización y cómputo aplicado

TESIS

Recocido Simulado y Hill Climbing con doble vecindad para el Problema de Calendarización en Talleres de Manufactura

Que para obtener el Grado de Maestro

Presenta

Ángel Ricardo Diez González

Director de Tesis

Dr. Federico Alonso Pecina

Co-director de tesis

Dr. Marco Antonio Cruz Chávez

CUERNAVACA, MORELOS …,2022

RESUMEN

Uno de los problemas clásicos del área de optimización combinatoria es el Problema de Calendarización para Talleres de Manufactura (del *inglés Job shop Scheduling Problem*), el cual se puede entender como un proceso de toma de decisiones y asignación de recursos para completar un número de tareas con períodos de tiempo determinados. Actualmente el estudio de este problema sigue vigente, ya que está clasificado como de complejidad NP-Completo y por lo tanto no existen métodos que puedan encontrar la mejor solución con algoritmos determinísticos que presentan prueba de optimalidad en un tiempo polinomial.

En esta tesis se propone un algoritmo aproximado de cuatro fases para resolver el problema del Job Shop. Este algoritmo presenta una implementación de Recocido Simulado junto a una Mejora Iterativa de Búsqueda Local ampliada a una Doble Vecindad. Las instancias usadas en la experimentación pertenecen a *benchmarks* clásicos de la literatura y los resultados fueron comparados con los óptimos conocidos de dichas instancias. Se encontró el óptimo en más de la mitad de las instancias de prueba.

ABSTRACT

The Job Shop Scheduling Problem is one of the most well-known problems of Combinatorial Optimization, it can be seen as a process of decision-making and resource allocation for the completion of a number of tasks within a certain time. The study of this problem is still relevant today as it has been classified as NP- Complete in complexity and thus there isn't a method that can find the best solution by means of a deterministic algorithm with optimality proof in a polynomial time.

In this thesis we propose a four-step approximate algorithm to solve the Job Shop problem. This algorithm features an implementation of Simulated Annealing along with an Iterative Improvement Local Search expanded to a Double Neighborhood. The instances used in the experimentation belong to classical benchmarks from the literature and the results were compared with the known optima of these instances. The algorithm found the optimum for more than half the instances used as a benchmark.

GLOSARIO

Benchmark	Una instancia o grupo de instancias utilizadas para medir y comparar resultados para un tipo de problema
Calendarización	Orden de procesamiento de tareas con relación a los recursos disponibles
Hill Climbing	(HC) Escalado de Colina
Instancia	Datos de entrada que representan caso específico de un tipo de problema
Job Shop Scheduling Problem	(JSSP) Problema de Calendarización en Talleres de Manufactura
Makespan	Duración total de una calendarización que corresponde al tiempo final de la última tarea asignada
Metaheurística	Entorno algorítmico de alto nivel e independiente del problema, que proporciona un conjunto de pautas o estrategias para desarrollar algoritmos de optimización heurística.
Optimización Combinatoria	Rama de las ciencias de la computación enfocada en resolver problemas de decisión con relación a maximizar o minimizar una función objetivo.

ÍNDICE

LISTA DE FIGURAS

LISTA DE TABLAS

1 INTRODUCCIÓN

1.1 Antecedentes

El problema de calendarización en talleres de manufactura (Job Shop Scheduling Problem), es un problema que existe desde la década de 1950 a partir del crecimiento de la industria.

El Job Shop es un proceso de toma de decisiones que se ocupa de la asignación de recursos a un conjunto de tareas en períodos de tiempo determinados. El resultado de esta asignación es una calendarización.

Este problema por su naturaleza compleja, resulta poco práctico tratar de resolverlo a mano, salvo por instancias muy pequeñas o simplificadas. Fue a partir de la llegada de nuevas tecnologías y el acceso extendido a las computadoras que fue posible retomar el problema del job shop desde un punto de vista computacional.

Hasta el día de hoy, el problema sigue siendo tratado, aplicando una cantidad de técnicas y enfoques que faciliten obtener soluciones buenas y prácticas.

1.2 Planteamiento del problema

El problema de calendarización en talleres de manufactura (Job Shop) pertenece al área de optimización combinatoria y es catalogado dentro de las ciencias computacionales como 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.

En el problema de Job Shop, se tiene una serie de trabajos (jobs en inglés), los cuales están divididos en operaciones, cada operación es atendida por una máquina (machine en inglés) (Syarif et al., 2021).

A estos conceptos se agregan una serie de consideraciones tales como:

- Una máquina no puede procesar más de una operación a la vez.
- Las operaciones iniciadas no pueden ser detenidas.
- No pueden cancelarse trabajos.
- No se pueden procesar al mismo tiempo ningún par de operaciones pertenecientes a un mismo trabajo
- Cada operación debe finalizar antes de empezar otra de un mismo trabajo.
- Las operaciones de un trabajo llevan un orden predeterminado y no es posible cambiarlo

La asignación de máquinas a las operaciones de cada trabajo genera una calendarización. Su objetivo es optimizar algún aspecto del proceso, tal como la minimización del tiempo de finalización de la última tarea, esto es representado por una función objetivo. Esta función objetivo, o función de costo, permite comparar soluciones y medir la calidad de estas.

1.3 Objetivos

1.3.1 Objetivo General

Implementar un algoritmo clásico de Recocido Simulado y de Escalado de Colina para el Job Shop Scheduling Problem. El algoritmo de Escalado de Colina aplicará una búsqueda local de doble vecindad.

1.3.2 Objetivos específicos

- Definir una estructura de vecindad doble para el problema de Job Shop Scheduling Problem.
- Evaluar la eficacia de un algoritmo de recocido simulado con una búsqueda local con doble vecindad para el problema de asignación de recursos para el problema en talleres de manufactura.
- Realizar un análisis comparativo de los resultados obtenidos con los obtenidos en la literatura.
- Evaluación de la eficacia cuando se aplica la doble vecindad en la búsqueda local.

1.4 Hipótesis

Al implementar el algoritmo de recocido simulado y una búsqueda local con doble vecindad, se obtendrán soluciones óptimas del problema de Job Shop Scheduling en un subconjunto de las instancias proporcionadas por la literatura. Al utilizar la doble vecindad también se obtendrán soluciones de un error relativo de máximo 5% con respecto al óptimo en la mayoría de las instancias (al menos un 80%).

1.5 Justificación

La optimización combinatoria consiste en investigar y diseñar enfoques de soluciones a problemas cuyo espacio de soluciones crece de forma exponencial conforme crece el tamaño del problema. Este grupo de problemas se llevan estudiando desde hace más de sesenta años, debido principalmente a que muchos de estos son catalogados como de complejidad NP-duros (NP-hard).

Los problemas NP-duros están caracterizados por no tener un método de resolución que arroje resultados en un tiempo polinomial, es decir, los problemas NP-duros son catalogados como intratables. Mientras no se encuentre una solución óptima a este tipo de problemas, es de interés investigar y desarrollar técnicas para tratarlos de la mejor manera.

Dentro del área de la optimización combinatoria se puede encontrar los problemas de calendarización. Los problemas de calendarización siguen vigentes en la actualidad, pues sus principios son requeridos por la capacidad de implementarse ampliamente en distintas áreas. El Job Shop pertenece a los problemas de calendarización y por ende al área de la optimización combinatoria. La intratabilidad del Job Shop radica en la enorme cantidad de posibles soluciones que puede tener cada instancia.

El problema de calendarización en talleres de manufactura (Job Shop), es un modelo específico de calendarización enfocado a la producción en la industria. En él, se busca asignar los recursos disponibles para realizar una serie de tareas o trabajos, minimizando el tiempo total de producción.

El Job Shop ha sido el origen de otros modelos de calendarización con el mismo objetivo de producción, pero con enfoques más específicos y aunque otros modelos de calendarización puedan responder mejor a instancias específicas, el Job Shop sigue siendo el modelo más general del problema.

Muchas técnicas se han propuesto para resolver este problema, entre ellas los llamados métodos aproximados, de los cuales se pueden encontrar los de búsqueda local y las metaheurísticas.

1.6 Organización de la tesis

En el capítulo 2 se da una introducción al problema de calendarización de trabajos en talleres de manufactura, su historia, clasificación y la descripción conceptual del problema con su modelación matemática. También se hace una revisión de otros trabajos que se han hecho sobre el tema y las técnicas usadas para resolverlo.

En el capítulo 3 se presenta la metodología usada en este proyecto, se muestra el algoritmo propuesto para resolver este problema y como está dividido. En este capítulo se explica en detalle la estructura de vecindad, el comportamiento de los algoritmos y el mecanismo de búsqueda y mejora de soluciones.

En el capítulo 4 se presentan los resultados obtenidos al evaluar instancias clásicas de la literatura mediante el algoritmo propuesto en esta tesis. Los resultados son comparados con los óptimos registrados de dichas instancias.

En el capítulo 5 se abordan las conclusiones a las que se llegó después de observar los datos expuestos en el capítulo anterior. Además, se hablará de los trabajos a futuro que pudieran desprenderse de esta tesis con base a los resultados.

2 ESTADO DEL ARTE

2.1 Introducción

El problema de calendarización en talleres de manufactura (Job Shop) puede rastrearse hasta (Johnson, 1954), donde se presenta un escenario de calendarización con casos en los que dos o hasta tres máquinas debían recibir distintas etapas de producción de artículos diferentes, sosteniendo que existe un método de resolución óptimo para dos máquinas.

El problema del Job Shop ha sido catalogado como un problema NP completo en (Garey et al. 1976), donde se señala que, para los casos con más de dos máquinas, no existe un método que resuelva el problema dentro de un tiempo polinomial. Esto significa que el tiempo requerido para resolver una instancia cualquiera incrementa de manera exponencial con respecto a sus dimensiones.

El Job Shop Scheduling Problem (JSSP) pertenece al área de optimización combinatoria, y es parte de un grupo de problemas conocidos como problemas de calendarización de producción (Production Scheduling Problems o PSP). Los PSP son divididos en (Graves, 1981) de acuerdo a su complejidad de procesamiento:

- *Una máquina, una tarea*: Todos los trabajos consisten de una sola operación que debe ser realizada por una sola máquina.
- *Múltiples máquinas, una tarea:* Todos los trabajos consisten de una sola operación y esta puede ser realizada por una máquina cualquiera disponible.
- *Flow Shop, Multitarea:* Los trabajos están formados por más de una operación y poseen una restricción de precedencia en su realización.
- *Job Shop, Multitarea:* Los trabajos están formados por más de una operación y poseen una restricción de precedencia en su realización. Cada máquina también está limitada a una serie de operaciones definidas.

El trabajo de calendarización es definido de forma general en (Verderame & Floudas, 2008), como la asignación de recursos dentro de una planta de producción, de esta forma las operaciones son destinadas a unidades de procesamiento (máquinas) para así determinar el tiempo transcurrido y el material procesado. Sin embargo, el proceso de producción es complejo, con eventos inesperados y requerimientos específicos para cada caso. Esto hace que para realizar el trabajo de calendarización sea necesario hacer suposiciones para simplificar el problema, formando así un modelo abstracto.

Existen varias consideraciones usadas para representar el proceso de producción. Al modificar las características o restricciones del problema se obtienen modelos de calendarización distintos. En (Lin et al. 2012) se agrupan estos modelos en cinco categorías que se diferencian entre sí de acuerdo a sus objetivos, restricciones y los datos de salida que se obtienen. Los modelos de calendarización que se describen son los siguientes:

- Problema de Talleres de Manufactura (Job Shop Problem o JSP): Cada tarea es realizada por una máquina específica que no puede ser sustituida, el objetivo es minimizar el tiempo de producción (Makespan en inglés). Es el modelo básico del cual se desprenden los demás.
- Problema de Talleres de Manufactura Flexible (Flexible Job Shop Problem o FJSP): Comparte las características del JSP, con la diferencia de que se puede elegir asignar

cada tarea entre una o más máquinas. Su objetivo es el de minimizar el Makespan así como balancear las cargas de trabajo de las máquinas.

- Secuencia de Operaciones Integradas y Selección de Recursos (Integrated Operation Sequence and Resource Selection o iOS/RS): Para este modelo las secuencias de las operaciones no son fijas y el tamaño de lote y tamaño de carga unitaria deben considerarse en la calendarización. El tiempo de inicio de cada operación depende del tiempo de finalización de las operaciones anteriores, así como del tiempo de finalización para el tamaño de la carga unitaria.
- Modelo de Calendarización Integrado con Multi-Planta (Integrated Scheduling Model with Multi-Plant): Toma en cuenta los tiempos de manufactura, así como de transporte, logística y control de inventario.
- Modo de Logística y Manufactura con Recolección y Entrega (Manufacturing and Logistic Mode with Pickup and Delivery): Extiende el modelo anterior al considerar cambios en la calendarización para alcanzar tiempos de entrega. Las restricciones pueden alterarse a cambio de incrementar un valor de penalización.

Los modelos de calendarización nos proporcionan un marco de trabajo con el cual abordar el problema y poder encontrar mejores soluciones. Dependiendo de la consideración de restricciones adicionales, el JSSP se ha diversificado en nuevas variantes, como se puede observar en (Masmoudi, 2019) donde se considera el uso de energía en las soluciones o en (Ying & Lin, 2020) donde la espera entre operaciones de un mismo trabajo no está permitida.

Esta tesis se centrará específicamente en el modelo de JSP y en la propuesta de un algoritmo para resolverlo.

2.2 Algoritmos de optimización para el JSP

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. En (Zhang et al. 2019) se hace una recopilación de los algoritmos que se han propuesto para el problema del Job Shop y son divididos 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. 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), donde se propuso una serie de reglas de decisión para encontrar la calendarización óptima a un problema de producción de dos y tres máquinas. Para los casos con dos máquinas, Johnson comienza identificando los conjuntos *A* y *B*, donde A_i es el tiempo de operación de un trabajo en la primera máquina y B_i es el tiempo de operación en la segunda máquina. También considera un conjunto *X*, que representa el tiempo que la segunda máquina permanece inactiva. Al calcular los valores de cada elemento de *X* asociados a los elementos de *A* y *B*, Johnson obtiene una forma general que traduce en siete pasos. Estos pasos son una serie de consideraciones para elegir pares de operaciones de los conjuntos *A* y *B*, encontrando así la secuencia óptima para minimizar el makespan de la calendarización. Este método también es expuesto para resolver problemas con tres máquinas y aunque funciona para casos específicos no lo resuelve de manera general.

Más tarde en (Wagner, 1959) se llegaría a la conclusión de que es posible encontrar la solución óptima para el JSP a partir de técnicas de programación lineal, sin embargo, también se menciona la impracticidad del método debido al alto tiempo computacional que se requiere. Así mismo, otro acercamiento con programación lineal fue producido en (Manne, 1960), que, aunque con resultados más eficientes, se concluye mencionando que un método aproximado aplicado a problemas grandes, resultaría en un menor costo computacional.

En (Lomnicki, 1965) se subraya que a pesar del trabajo hecho en (Johnson, 1954) para encontrar soluciones óptimas a problemas con dos máquinas, otros 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 combinatoria. Este método consiguió resultados óptimos para instancias específicas.

Otra implementación del método Branch and Bound fue presentada por (McMahon & Florian, 1975) quienes con su versión mejoraron los resultados de la literatura en su momento. Por su parte, en (Hefetz & Adiri, 1982) se trabajó en eficientizar el problema de calendarización para dos máquinas, mientras que en (Potts & Van Wassenhove, 1985) se hizo para una sola máquina con hasta cuarenta trabajos.

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 resolverlos de manera exacta. El JSP no es una excepción y es por eso 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.

En (Mirshekarian, & Šormaz, 2016) se menciona 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. Actualmente estas reglas heurísticas usadas para generar soluciones han evolucionado en lo quese conoce como metaheurísticas, término identificado en (Sörensen et al., 2018) como “una implementación específica de un algoritmo basado en un marco de alto nivel (o en una combinación de conceptos de diferentes marcos) diseñado para encontrar una solución a un problema de optimización específico”. A continuación, se mencionan algunos de estos métodos.

2.2.2.1 Algoritmos Genéticos

Este método es descrito en (Festa, 2014) 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 acervo genético y la generación de individuos que sobreviven mejor en un entorno competitivo.

A partir de la analogía descrita, en (Della Croce et al. 1995) se detallan los pasos generales a seguir para aplicar Algoritmos Genéticos a un problema de optimización, los cuales son:

1. El espacio de búsqueda de todas las posibles soluciones se asigna a un conjunto de cadenas finitas y se elige un umbral de iteración de varias generaciones.
2. Se selecciona un conjunto de soluciones al azar, construyendo la población inicial.
3. Se calcula un valor para la calidad de la solución para cada individuo a partir de su función objetivo.
4. Se selecciona de forma aleatoria un conjunto de individuos para que se reproduzca.
5. A partir del conjunto seleccionado, se genera una nueva generación de individuos aplicando diferentes operadores genéticos.
6. Se eliminan de la población individuos anteriores para ingresar nuevos individuos.
7. Se calculan nuevos valores para los nuevos individuos y se incluyen en la población como una nueva generación
8. Si se alcanza el umbral de iteración, el proceso finaliza y se devuelve la mejor solución, si no el algoritmo regresa al paso 4.

El método de Algoritmos Genéticos (GA por sus iniciales en inglés) fue propuesto originalmente en (Holland, 1975), y fue en (Davis, 1985) donde se adaptó y aplicó este método por primera vez al JSP, argumentando que al ser un método no determinista por sus mecanismos aleatorios de cruzamiento y mutaciones, se evita caer en óptimos locales, mejorando así las soluciones. A su vez, en (Falkenauer & Bouffouix, 1991) se implementó GA para resolver el problema del Job Shop con restricciones de fechas de entrega y salida. Ese mismo año en (Nakano & Yamada, 1991) se presentó otra implementación de Algoritmos Genéticos para JSP, añadiendo un procedimiento (armonización global) para modificar soluciones infactibles y convertirlas en soluciones factibles, con el propósito de obtener más individuos con los cuales trabajar. En (Liu et al. (2021) se propone un algoritmo basado en GA para resolver JSSP pero que tiene en cuenta objetivos adicionales a la minimización del makespan.

2.2.2.2 Búsqueda Tabú

Este método fue propuesto y descrito a detalle en (Glover, 1986), donde sostiene que uno de los obstáculos más grandes que tienen que afrontar los métodos heurísticos para resolver problemas de optimización, es el de quedar atrapados en óptimos locales. Glover propuso la Búsqueda Tabú como otro método para resolver el problema de los óptimos locales, junto con otros a los que agrupó en categorías como aleatoriedad controlada, estrategias de aprendizaje y descomposición inducida.

En (Festa, 2014) se describe a la Búsqueda Tabú como una estrategia que hace uso de estructuras de memoria, que permiten movimientos que incrementan los valores a optimizar, esto con el fin de poder escapar de los mínimos locales. Durante la búsqueda, este algoritmo utiliza una estructura de datos especial llamada lista tabú, del cual recibe su nombre, para almacenar información sobre las soluciones generadas en la última iteración. El proceso

comienza con una solución dada y se mueve de la solución actual a una solución nueva dentro del vecindario. Para evitar volver a un mínimo local recién visitado, los movimientos inversos que conducen a ese mínimo local están prohibidos (se vuelven tabú), para una serie de iteraciones que pueden ser fijas o variables.

En (Hurink et al.1994) se presentan los pasos a seguir para el algoritmo de Búsqueda Tabú de la siguiente forma:

1. Calcular una solución inicial dentro del espacio de soluciones.
2. Realizar un cambio permitido que no genere una solución tabú, en caso contrario terminar el algoritmo.
3. Calcular la nueva solución y reemplazar la solución actual.
4. Actualizar la lista tabú.
5. Repetir hasta completar el criterio de parada (número de iteraciones, tiempo desde la última mejora, etc.).

El algoritmo de Búsqueda Tabú fue implementado por primera vez al JSP por (Taillard, 1994), quien adaptó el método en su forma convencional secuencial, así como una versión paralelizada en ambiente distribuido, aunque concluye que ésta última no se adapta bien al problema. (Dell'Amico & Trubian, 1993) aplicaron también Búsqueda Tabú al JSP contribuyendo con un método para encontrar soluciones factibles que les permitió obtener mejores soluciones. Otra aplicación de Búsqueda Tabú para el problema de Job Shop fue hecha en (Hurnk et al, 1994) para una variante denominada de máquinas multipropósito (del inglés Multi-Purpose Machine), donde las operaciones pueden ser trabajadas por más de una máquina según criterios establecidos por la instancia, de forma similar a la categoría de problemas de flow shop. Regresando al problema clásico de JSP, en (Barnes & Chambers, 1995) se aplicó una versión de Búsqueda Tabú donde guardaban soluciones prometedoras durante la búsqueda, a las cuales se podía volver si se consideraba oportuno, además, en caso de no tener movimientos posibles fuera de la lista tabú, se vaciaba la lista y se reiniciaba la búsqueda desde la última solución para evitar parar el algoritmo de forma anticipada.

2.2.2.3 Algoritmo de Colonia de Hormigas

El algoritmo de Optimización de Colonia de Hormigas (ACO, por sus siglas en inglés Ant Colony Optimization), fue concebido en (Colorni et al. 1991) donde fue desarrollado para resolver problemas de optimización y fue aplicado inicialmente al problema del agente viajero (TSP). Este algoritmo se modeló a partir del comportamiento de colonias de hormigas al buscar comida. Cada hormiga, o individuo, solo puede realizar acciones simples y no tiene conocimiento explícito de las acciones del resto, sin embargo, como grupo son capaces de encontrar rutas eficientes hacia sus fuentes de alimento. Esta capacidad se debe a su forma de comunicación, en la que cada hormiga deja un rastro de feromonas tras de sí, este rastro es percibido por otras que podrán seguirlo y reforzarlo con más feromonas o evaporándose si nadie lo sigue. Las hormigas tienen mayor probabilidad de seguir un camino con alta concentración de feromonas, sin embargo, existe la posibilidad de que una hormiga escoja un camino diferente y descubra una mejor ruta, evitando así quedar atrapado en óptimos locales.

Visualizando el JSP en forma de grafo, los autores de (Colorni et al, 1994) adaptaron su metodología para aplicarla con resultados aceptables, esto como parte de una serie de trabajos para demostrar la robustez de su algoritmo aplicado a diversos problemas de optimización. En (Blum & Sampels, 2004) se implementaron ACO para resolver Group

Shop Scheduling Problem, una generalización de los problemas de Open Shop en la que mejoraron los resultados de diversas instancias, y Job Shop, en donde fueron los primeros en resolver la instancia ft10 de Fisher and Thompson (una instancia de prueba clásica de 10x10) por medio de Colonia de Hormigas. En (Udomsakdigool & Kachitvichyanukul, 2008) se propuso una variante del algoritmo en donde se define más de una estructura de colonia, en la que cada una registra la mejor ruta obtenida para luego ser contrastadas en un registro global que puede o no ser actualizado durante las iteraciones. Por su parte, en (Chaouch et al, 2017) se aplicó también ACO para resolver el problema de calendarización de talleres de manufactura distribuida (DJSP por sus siglas en inglés Distributed Job Shop Scheduling), otra versión del JSP en la que se cuenta con más de un conjunto idénticos de máquinas llamadas fábricas (factories), de las cuales se puede elegir donde procesar las operaciones de los trabajos a realizar.

2.2.2.4 Recocido Simulado

El algoritmo de Recocido Simulado, o SA (del inglés Simulated Annealing), fue presentado originalmente en (Kirkpatrick et al.1983), donde se desarrolló este método para resolver problemas de optimización combinatoria con un enfoque de mejora iterativa. Los autores reconocen la tendencia de este enfoque de quedar atrapado en óptimos locales y mencionan que debe haber un factor de aleatoriedad en las soluciones iniciales para evitarlos. Para esto, el algoritmo se apoya en la física estadística para hacer una analogía con el proceso físico de recocido, del cual toma su nombre, así como otros principios.

El recocido es un tratamiento que se le da a algunos metales y otros materiales, esto con el fin de mejorar sus características físicas. En (Avner, 1988) se describe el recocido como el proceso en que una estructura distorsionada regresa a un estado libre de tensiones por medio de la aplicación de calor, al que le sigue un enfriamiento lento desde una temperatura determinada. En (Van Laarhoven & Aarts, 1987) se explica que, empezando con una temperatura suficientemente alta, las partículas del material se ordenarán de manera aleatoria y mediante un enfriamiento controlado es posible para dichas partículas alcanzar una configuración estable. En esta analogía, las distintas configuraciones de las posiciones que pueden tomar los átomos en un metal, representan el espacio de soluciones de un problema de optimización y una configuración estable de estos átomos corresponde a la solución óptima de una instancia.

El Recocido Simulado está basado en un algoritmo del tipo de mejora iterativa, por lo que comparte algunas características tales como una estructura de vecindario para el espacio de soluciones y el uso de una solución inicial. En (Bertsimas & Tsitsiklis, 1993) se enlistan los siguientes puntos como los elementos básicos del SA:

- Un conjunto finito de soluciones, *S*.
- Una función de costo calculada a partir de una solución del conjunto de soluciones, $f(s) \mid s \in S$.
- Un vecindario derivado de una solución dada $s_i \in N(s)$.
- La probabilidad de poder escoger entre por lo menos una nueva solución del vecindario actual.
- Una función no incremental de temperatura sobre tiempo T(t).
- Un estado inicial s_0 del conjunto *S*.

El algoritmo parte de una solución inicial aleatoria que será sustituida por nuevas soluciones mediante una búsqueda local. Estas soluciones son comparadas con referencia a su función

de costo y pueden o no seleccionarse dependiendo de su calidad. Los pasos para aplicar este método son los siguientes:

1. Calcular una solución inicial dentro del espacio de soluciones.
2. Definir la temperatura inicial y un factor de enfriamiento.
3. Definir un número de iteraciones.
4. Calcular una nueva solución derivada del vecindario de la solución actual y compararlas mediante su función de costo.
5. Si la nueva solución es de mejor calidad que la solución actual se sustituye, si no, se evalúa la función de aceptación.
6. La función de aceptación compara un número aleatorio entre 0 y 1 con el factor de probabilidad de Boltzmann. Esto determina si la nueva solución es aceptada a pesar de su menor calidad.
7. Repetir hasta alcanzar el número de iteraciones definido.
8. Disminuir la temperatura actual por el factor de enfriamiento y repetir desde el paso 4.
9. El algoritmo termina una vez la temperatura disminuya más allá de un umbral determinado.

La característica principal de este método es su mecanismo de aceptación de nuevas soluciones, el cual le permite seleccionar de forma probabilística peores valores en vez de solo descartarlos. Este comportamiento incrementa el espacio de búsqueda del que se cuenta y pretende solucionar el problema de los algoritmos de mejora iterativa al darle la posibilidad escapar de óptimos locales.

El Mecanismo de aceptación del Recocido Simulado está formado por dos conceptos físicos principales, el factor de distribución de Boltzmann y el algoritmo de Metropolis.

La distribución de Boltzmann es una herramienta de la física estadística que representa el comportamiento de las partículas de un material en estados diferentes con relación a su energía y temperatura. De esta distribución se tiene el factor de Boltzmann, el cual está dado por $e^{(-E/k_B T)}$, donde E es la energía del sistema y es representada en el algoritmo por la diferencia entre las funciones de costo entre soluciones, k_B es la constante de Boltzmann y T es la temperatura para ese momento. Por su parte, el algoritmo de Metropolis, presentado por primera vez en (Metropolis et al, 1953), permite la simulación de los cambios dados en un sistema físico hacia su equilibrio térmico, en el cual, a partir de un estado inicial, se realizan pequeñas modificaciones, y si la nueva configuración reduce la energía del sistema entonces es aceptada. Si por el contrario la energía del sistema incrementa, la nueva configuración puede ser aceptada probabilísticamente usando el factor de Boltzmann descrito anteriormente.

A continuación, en la figura 2.1, se presenta el pseudocódigo general para el algoritmo de Recocido Simulado.

```
inicia
s₀ = solución inicial
T = temperatura
ΔT = factor de enfriamiento
Tf = temperatura final
Num = número de iteraciones del ciclo
MaxIteraciones = iteraciones totales del ciclo
mientras T > Tf
      mientras Num < MAXIteraciones
            sᵢ = nueva solución, sᵢ ∈ N(s₀)
            si f(sᵢ) < f(s₀) entonces s₀=sᵢ
            si no
                  ΔE = f(s') - f(sᵢ)
                  r=random()
                  si r < exp(-ΔE/k*T) entonces sᵢ=s'
            Num++
      T=T*ΔT
fin
```

Fig. 2.1 Seudocódigo para Recocido Simulado

Una de los estudios tempranos del Recocido Simulado al problema del Job shop se puede observar en (Van Laarhoven et al,1992), en donde se explican detalles tales como la implementación del algoritmo al JSP y la definición del vecindario, así como el uso de un factor de enfriamiento para disminuir la temperatura en cada iteración hasta llegar a una temperatura final. A partir de sus resultados, los autores concluyen que un factor de enfriamiento de menor valor produce soluciones de mayor la calidad a expensas de un tiempo de ejecución mayor, también mencionan que para problemas grandes el SA es superior a un algoritmo de mejora iterativa simple y aunque se muestra más lento que otros métodos, es relevante por ser un método sencillo, fácil de aplicar, y que puede alcanzar soluciones de mejor calidad.

Con respecto al relativamente alto tiempo de ejecución del Recocido Simulado con respecto a otras heurísticas, en (Yamada & Nakano, 1996) se propone una forma de eficientizar la búsqueda local de nuevas soluciones para el algoritmo de SA. Para esto se definen los llamados bloques críticos, grupos de operaciones adyacentes dentro de una misma máquina que pertenecen a la ruta crítica de la calendarización a la que pertenecen, el vecindario es entonces formado por nuevas soluciones donde las permutaciones de pares de operaciones se dan únicamente sobre aquellas que pertenecen a un bloque crítico.

En (Kolonko, 1999) se menciona la posibilidad de que el Recocido Simulado regrese a óptimos locales previos, dadas suficientes corridas con respecto al tamaño del espacio de búsqueda, dando así soluciones subóptimas. Para solucionar este problema, el autor propone una hibridación de SA con Algoritmos Genéticos (GA). En esta implementación, se toma la estructura de GA como base, y se aplican corridas independientes de SA para la creación de cada individuo de la población y sus descendientes. Esta metodología pretende evitar el regreso a soluciones que deriven en óptimos locales.

En (Satake et al, 1999) se propone una estructura de vecindario alternativa a las 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. Esta estructura de vecindad genera nuevas soluciones recorriendo

operaciones sobre los espacios vacíos visibles de la gráfica y realiza cambios según ciertas consideraciones previas. La estructura de vecindad se presenta aplicada a Recocido Simulado pues los autores reconocen la tendencia del vecindario a proponer soluciones dentro de un óptimo local, el Recocido Simulado permite sortear este inconveniente por su mecanismo de aceptación de soluciones.

En (Cruz-Chavez, 2014) se presenta otro mecanismo de generación de vecindario 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 coste 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 rápidamente y de buena calidad. Este mecanismo de generación de vecindarios es aplicado a un algoritmo de Recocido Simulado con reinicio, reportando buenos resultados. En (Cruz-Chávez et al., 2019) una implementación de Recocido Simulado es ejecutada en un ambiente paralelo por medio de hilos independientes.

2.2.2.5 Otros Algoritmos

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 pues existen una gran cantidad de nuevas estrategias y algoritmos para resolver este tipo de problemas de optimización. A continuación, se hace mención de algunos otros métodos que se han propuesto aplicados al JSSP con buenos resultados.

Uno de los algoritmos más usados para resolver problemas de optimización es el de Búsqueda Local (del inglés Local Search). Este algoritmo sirve como base para metaheurísticas como Búsqueda Tabú y Recocido Simulado. De acuerdo con (Rabadi, 2016) el algoritmo consiste en realizar un seguimiento de la mejor solución registrada hasta el momento, mientras se aplica iterativamente una operación de mutación que transforma aleatoriamente una solución en otra, actualizando la solución registrada cada vez que se encuentra una mejor opción. Una implementación de este algoritmo es el llamado Escalado de Colina (Hill Climbing). En (Kato et al., 2018) se utiliza el algoritmo de Hill Climbing como parte de su propuesta para resolver el problema de Flexible Job Shop. También se menciona que el Hill Climbing es un método fácil de implementar y que entra en la categoría de algoritmos avaros, pues en cada iteración solo se elige una nueva solución, si esta representa una mejora con respecto a la anterior, de tal manera que tiene una gran probabilidad de quedar atrapado en un óptimo local.

El algoritmo de Optimización por Enjambre de Partículas (PSO, del inglés Particle Swarm Optimization) fue presentado por primera vez en (Kennedy & Eberhart, 1995). Este algoritmo tiene su origen en modelos para la simulación del comportamiento social que presentan ciertos animales que se mueven en grupo de forma coordinada, como aves y peces, tomando como metáfora la capacidad que tienen estos animales de encontrar puntos clave para su subsistencia, tales como comida o refugio, de forma eficiente. Este método está conformado entonces por una población de elementos (el Enjambre), los cuales poseen una posición y una velocidad, que recorren aleatoriamente el espacio de soluciones del problema. Cuando alguno de los elementos del enjambre (partículas) encuentra un punto prometedor en el espacio (Solución) comparte esta información uno a uno con el resto de la población, quienes irán modificando su comportamiento para moverse rumbo a dicho punto, el

comportamiento del enjambre se irá modificando cada vez que alguna de sus partículas encuentre una mejor solución durante su recorrido. En (Lin et al. 2010) se puede encontrar un acercamiento al JSP mediante PSO.

Otra metaheurística relativamente nueva es la de Evolución Diferencial o DE (del inglés Differential Evolution), publicada originalmente en (Storn & Price, 1997) y pertenece al grupo de algoritmos evolutivos. Este algoritmo trata de imitar la evolución de organismos a través de una cantidad de iteraciones y conservando aquellos mejor adaptados a su entorno. Este método inicia con una población de individuos (soluciones) la cual mediante mutaciones y cruzamientos cambian a través de varias iteraciones, aceptando aquellos cambios en la población que arrojen mejores soluciones. Este algoritmo comparte varios aspectos con el método de Algoritmos Genéticos, sin embargo, sus autores sostienen que resulta más fácil de implementar y suele converger más rápido en comparación.

Aún más reciente es el Algoritmo de Luciérnagas o FA (del inglés Firefly Algorithm) propuesto por (Yang, 2008). Este algoritmo, como sugiere su nombre, está inspirado en el comportamiento que presentan los grupos de luciérnagas durante el proceso de apareamiento, donde aquellos individuos capaces de generar luces más brillantes se vuelven más atractivos para el resto y tienen más probabilidades de atraer pareja. Para traducir esta metáfora, se tiene entonces una población de luciérnagas que representan diferentes soluciones y se considera que cualquier luciérnaga puede atraer a cualquier otro miembro del grupo siempre y cuando la luminosidad de la primera sea mayor que la de la segunda, a su vez que la luminosidad disminuye con la distancia. Durante cada iteración del algoritmo, la luciérnaga más brillante "vuela" aleatoriamente mientras que el resto se mueve con dirección a aquella más brillante desde su posición, generando entonces nuevas soluciones y registrando la mejor solución encontrada. En (Udaiyakumar & Chandrasekaran, 2014) se encuentra una adaptación del Algoritmo de Luciérnaga para resolver el problema de Job Shop. Cabe destacar que los autores mencionan que, dependiendo de la configuración u omisión de algunos parámetros, el FA puede verse como una variación del algoritmo de Optimización por Enjambre de Partículas.

Algoritmos como PSO y FA son catalogados como de Inteligencia de Enjambre o SI (Swarm Intelligence). En esta categoría se encuentran algoritmos como el de Colonia de Abejas o ABC (Artificial Bee Colony) que pretende imitar el comportamiento de las abejas al buscar néctar y que es usado en (Sundar, et al.,2015) para resolver una variante del JSP. Otro algoritmo perteneciente a SI es el denominado Optimización de Lobo Gris GWO (Grey-Wolf Optimization) que se inspira en el comportamiento jerárquico y de cacería de una jauría de lobos, en (Jiang & Zhang, 2018) se emplea este algoritmo para resolver instancias de job shop y flow shop. El Algoritmo de Murciégalo (Bat Algorithm) es un algoritmo iterativo basado en enjambre que emula el comportamiento de murciélagos para encontrar comida por medio de ecolocalización, este algoritmo es usado para resolver el problema de job shop en (Chen et al., 2019). Otros ejemplos aplicaciones de algoritmos basados en Enjambre para resolver el JSSP se pueden encontrar en (Yu, 2019), (Semlali, 2019) y (Anuar, 2019).

Entre otro tipo de métodos usados para resolver el JSSP se encuentran las llamadas hiper heurísticas, las cuales son una colección de heurísticas menores de las cuales el algoritmo puede escoger para resolver instancias específicas de un problema, dependiendo de consideraciones definidas *a priori*. En (Hart & Sim, 2016) se muestra la implementación de una hiper heurística para resolver el JSSP en la que se tiene un conjunto de reglas de despacho para priorizar operaciones en una calendarización. Estas reglas son aplicadas

mediante una programación genética que selecciona aquella que resulte de mayor provecho para el tipo de instancia específica. Se puede también mencionar el uso de Machine Learning para resolver el JSSP, como se muestra en (Wang, 2020) y su aplicación de un algoritmo de Deep Learning.

2.3 Representaciones del JSP

Anteriormente se dio de forma general una definición del Problema de Talleres de Manufactura, así como una lista de sus elementos principales y las consideraciones a las que está sujeto. A continuación, se presentan de forma detallada las partes de este problema.

2.3.1 Notación del JSSP

El problema del Job Shop está formado por un conjunto finito *J* de trabajos con cardinalidad n: $J = \{J_1, J_2, \ldots, J_n\}$, y $n = |J|$; un conjunto finito M de máquinas con cardinalidad m: $M = \{M_1, M_2, \ldots, M_m\}$ y $m = |M|$; y un conjunto de O que consiste en $n * m$ operaciones $O = \{i_1, i_2, \ldots, i_{m*n}\}$, 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 duración de proceso p, de tal forma que para una operación cualquiera $i \in O$, su tiempo final está dado por $(s_i + p_i)$, que es la suma de su tiempo de inicio más el tiempo de proceso.

El tiempo total de una calendarización o makespan es igual al tiempo en el que finaliza la última operación asignada, es decir por el tiempo final de la última operación calendarizada, que se representa como $MAX(s_i + p_i)$. Este valor se obtiene, mediante la función de costo o función objetivo.

A continuación, se muestra en la tabla 2.1 los elementos descritos de forma resumida:

Tabla 2.1 Notación para JSSP

SIMBOLO	DESCRIPCION
$\boldsymbol{J}$	Conjunto de trabajos
$\boldsymbol{n}$	Número total de trabajos
$\boldsymbol{M}$	Conjunto de máquinas
$\boldsymbol{m}$	Número total de máquina
$\boldsymbol{O}$	Conjunto de operaciones
$\boldsymbol{n * m}$	Número total de operaciones
$\boldsymbol{s_i}$	Tiempo de inicio de la operación i
$\boldsymbol{p_i}$	Tiempo de proceso de la operación i
$(\boldsymbol{s_i + p_i})$	Tiempo de terminación de la operación i
$\boldsymbol{MAX(s_i + p_i)}$	Máximo tiempo de terminación del conjunto de operaciones (Makespan)

2.3.2 Modelo Matemático para el Problema de Optimización

El modelo matemático del JSSP representa la minimización de la función objetivo, así como las restricciones que se deben tener en cuenta. A continuación, en la figura 2.3 se muestra el planteamiento formal del problema

1)	$min\ f = [\frac{max}{j\ \in\ O}(s_j\ +\ p_j)]$
2)	$\forall\ j\ \in\ O \qquad s_j\ \geq\ 0$
3)	$\forall\ i,j\ \in\ O\ \mid (i\ \preccurlyeq\ j)\ \in\ J_k \qquad s_i\ + p_i\ \leq\ s_j$
4)	$\forall\ i,j\ \in\ O\ \mid (i,j)\ \in\ M_k \qquad s_i\ + p_i\ \leq\ s_j\ \vee s_j\ + p_j\ \leq\ s_i$

Ecuación 2.1 Modelo matemático para la optimización de JSSP

En la ecuación 2.1, (1) describe la función de costo como la minimización del makespan, representada por el tiempo inicial más el tiempo de proceso de la última operación en la calendarización. En (2) define la restricción en la que el tiempo de inicio de cualquier operación debe ser de un valor no negativo. (3) se refiere a la restricción de precedencia entre cualquier par de operaciones dentro de un mismo trabajo, donde una operación posterior no puede iniciar antes que el tiempo final de la anterior. Por último, (4) define la restricción de capacidad de las máquinas, en la que cada máquina solo es capaz de procesar una operación a la vez y esta debe llegar a su tiempo final antes que otra pueda comenzar.

2.3.3 Modelo de Grafo Dirigido

EL JSSP también puede ser representado en la forma de un grafo dirigido $G(O, A, E)$, donde O es el conjunto de nodos del grafo, los cuales están conformados por las operaciones del problema y que tienen un valor igual a su tiempo de procesamiento, adicionalmente se tienen dos nodos ficticios "I" y "$*$" con valor de tiempo de procesamiento 0 y que representan el inicio y final del grafo. El conjunto A está formado por una serie de arcos dirigidos que denotan el orden de precedencia que existe entre operaciones pertenecientes a un mismo trabajo. Por último, el conjunto E define la relación entre operaciones que deben ser procesadas por una misma máquina mediante arcos no dirigidos. A continuación, en la figura 2.2 se muestra un ejemplo de grafo para una instancia de 3x3 (tres trabajos y tres máquinas).

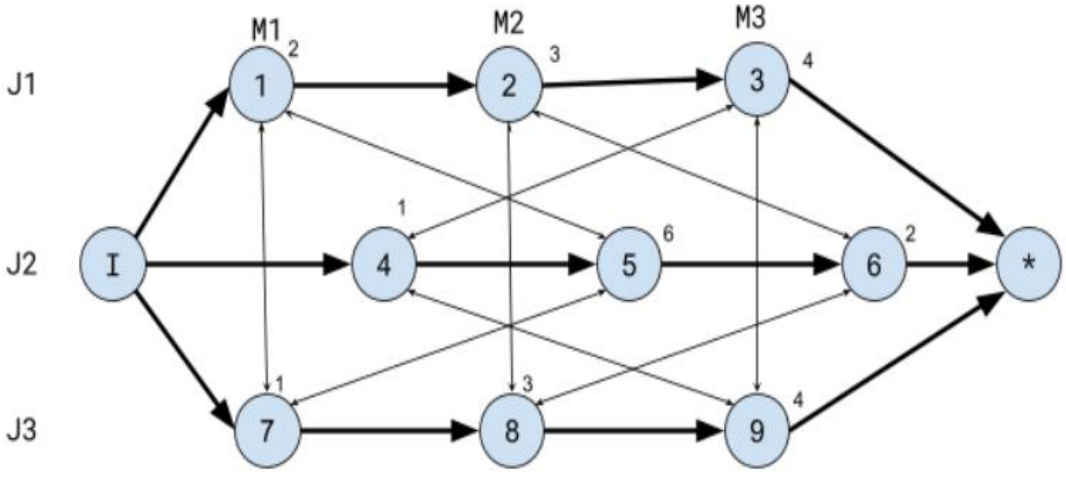


Fig. 2.2 Modelo de grafo para instancia de 3x3

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. Por su parte, los arcos

no dirigidos que forman los subconjuntos de operaciones $\{1, 5, 7\}$, $\{2, 6, 8\}$ y $\{3, 4, 9\}$, representan las máquinas 1, 2 y 3 respectivamente.

El modelo de grafo dirigido también es usado para representar las diferentes calendarizaciones que se pueden obtener para una instancia cualquiera. La siguiente figura 2.3 muestra una posible calendarización para el ejemplo anterior de 3x3.

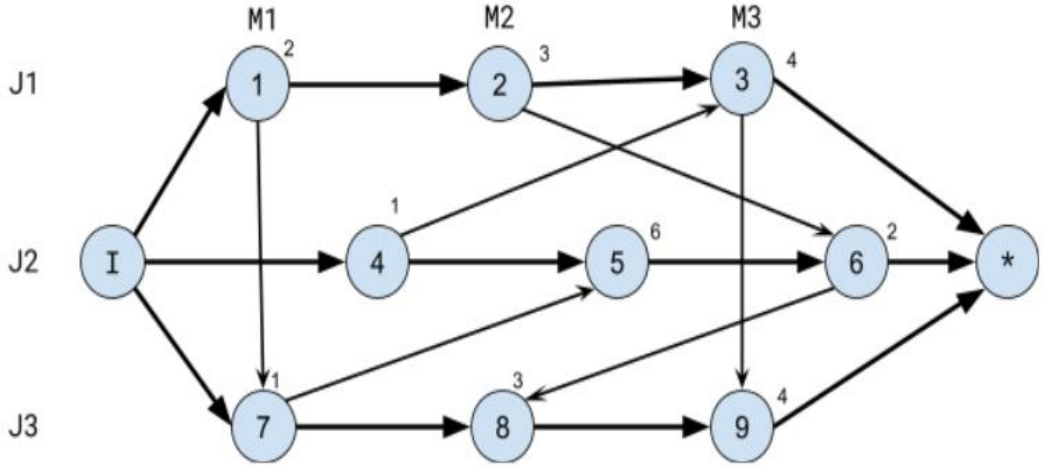


Fig. 2.3 Modelo de grafo para posible solución

En este grafo se pueden observar tres subconjuntos de operaciones equivalentes a cada una de las tres máquinas disponibles. Estos subconjuntos muestran el orden específico en que las operaciones deben ser procesadas por cada máquina en esta calendarización. Se tiene entonces que la máquina 1 debe procesar en orden las operaciones $\{1, 7, 5\}$, la máquina 2 debe procesar $\{2, 6, 8\}$ y la máquina 3 las operaciones $\{4, 3, 9\}$.

Para obtener el makespan a partir de esta representación, se debe recorrer el grafo teniendo en cuenta los valores de los nodos para encontrar la ruta más larga desde el nodo "*I*", que tiene un valor inicial de 0, hasta el nodo final "$*$".

2.3.4 Diagrama de Gantt

Otra forma de representar una calendarización para el problema del Job Shop, 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. Esta 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.

A continuación, la tabla 2.2 muestra en forma de diagrama de Gantt el ejemplo de calendarización para el problema de 3x3 mencionado en la sección anterior.

Tabla 2.2 Grafica de Gantt para posible solución

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
M1	1		7	5														
M2			2							6		8						
M3	4					3									9			

En esta tabla se observan los subconjuntos de operaciones asignadas a cada máquina, las cuales respetan la restricción de capacidad. Además, debido a la restricción de precedencia entre elementos de un mismo trabajo, se puede apreciar la existencia de tiempos de holgura (tiempos de espera entre pares de operaciones) durante la calendarización.

El diagrama de Gantt es una herramienta útil que permite evaluar de forma intuitiva la calidad de una solución, pues muestra de forma gráfica los tiempos de espera entre operaciones dentro de cada máquina. Sin embargo, este método se vuelve poco práctico conforme las dimensiones del problema crecen.

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 híbrido de 4 fases, basado en Recocido Simulado combinado con una búsqueda local sencilla (Hill Climbing) e implementando una estructura de doble vecindad, tal como se observa esquematizado en la figura 3.1.

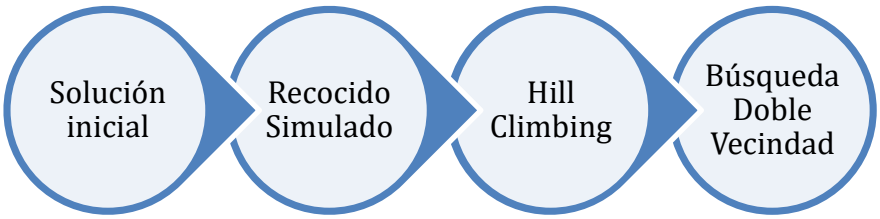


Fig. 3.1 Esquema de solución del algoritmo

Durante este capítulo se describirá cada uno de los pasos del esquema anterior.

3.2 Datos iniciales para el problema de Job Shop

Las instancias que se usarán para evaluar este algoritmo son tomadas de la literatura y se encuentran en el formato presentado en la OR-Library. Cada Instancia consiste de un archivo de texto que contiene la información del número de trabajos y el número de máquinas, seguido de una matriz para las operaciones. La matriz de operaciones está dispuesta en filas, cada una representando un trabajo diferente; y columnas, donde se encuentras las operaciones de izquierda a derecha y cuya posición indica el orden de precedencia para el problema. Cada operación está formada por pares de números que corresponden a la máquina en que debe ser procesada dicha operación, seguido del valor de su tiempo de operación. La matriz de operaciones puede visualizarse tal como se ejemplifica en la tabla 3.1.

Tabla 3.1 Matriz de datos de entrada para instancia de 6x6 donde M es el número de máquina y t es el tiempo

	M	t	M	t	M	T	M	t	M	t	M	t
J_0	2	1	0	3	1	6	3	7	5	3	4	6
J_1	1	8	2	5	4	10	5	10	0	10	3	4
J_2	2	5	3	4	5	8	0	9	1	1	4	7
J_3	1	5	0	5	2	5	3	3	4	8	5	9
J_4	2	9	1	3	4	5	5	4	0	3	3	1
J_5	1	3	3	3	5	9	0	10	4	4	2	1

3.3 Generación de Soluciones Iniciales

Para representar el problema dentro del algoritmo, lo primero es definir una solución inicial a partir de los datos de entrada y con ellos ordenarlos en una nueva estructura para generar una nueva matriz de operaciones. A cada combinación de máquina-trabajo se le asigna un identificador de operación, el cual es número que se obtiene de multiplicar el número de

trabajos más el número de operación. Este identificador almacena datos relevantes y sirve para representar cada operación, tal como se observa en la figura 3.2. Con este identificador es posible obtener la máquina a la que debe programarse y el tiempo que durará la operación.

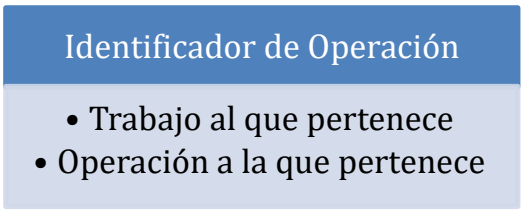


Fig. 3.2 Estructura de datos de las operaciones

A partir de los datos de entrada se obtiene la información para el tamaño de la instancia a resolver, los cuales son asignados a las variables *J* y *M*, que representan el número de trabajos y de máquinas respectivamente. Con el tamaño definido por estas variables y el resto de información provista por los datos de entrada, se procede a crear la nueva matriz de operaciones (tabla 3.2) con base en la estructura antes mencionada.

Tabla 3.2 Matriz de operaciones para una instancia de 6x6

JOBS	OPERACIONES					
J_1	0	1	2	3	4	5
J_2	6	7	8	9	10	11
J_3	12	13	14	15	16	17
J_4	18	19	20	21	22	23
J_5	24	25	26	27	28	29
J_6	30	31	32	33	34	35

3.3.1 Representación de la solución

Las soluciones para este algoritmo están representadas por una cadena, la cual está formada por todas las operaciones que conformen el problema, en la tabla 3.3 se puede observar un ejemplo.

Tabla 3.3 Inicio de una cadena de solución para instancia de 6x6

0	24	30	31	12	13	14	18	6	...

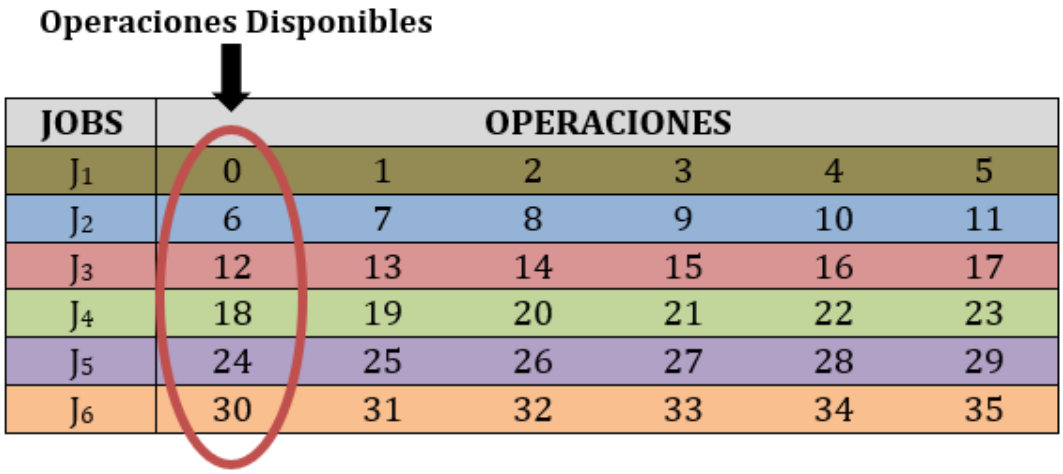
Esta cadena resulta de utilidad pues mediante intercambios en las posiciones de las operaciones es posible generar nuevas soluciones, creando así una estructura de vecindad que se explicará más adelante.

Al inicio del algoritmo, es necesario generar una primera cadena que represente una solución inicial aleatoria. Esta cadena se construye con base en la matriz de operaciones obtenida anteriormente y debe tener en cuenta la restricción de precedencia de cada operación a lo largo del proceso de selección. A continuación, se muestran los pasos a seguir de este proceso:

1. Identificar las operaciones disponibles de la matriz de soluciones.
2. Seleccionar una de las operaciones disponibles de forma aleatoria.

3. Colocar la operación seleccionada en el extremo de la cadena.
4. Actualizar el conjunto de operaciones disponibles.
5. Repetir hasta acabar de ordenar todas las operaciones del problema

El primer paso para generar la cadena de la solución inicial es identificar las operaciones disponibles. Para esto se toman en cuenta aquellas operaciones que no tienen problemas de precedencia, es decir, que en caso de tener una o más operaciones previas, estas ya hayan sido agregadas a la cadena de solución. En la figura 3.3 se muestra un ejemplo donde se señala el grupo de operaciones disponible para un momento en particular.

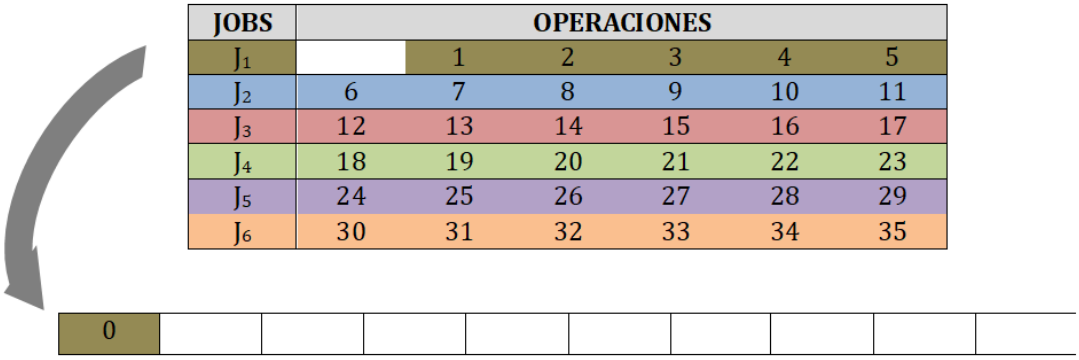


JOBS	OPERACIONES					
J_1	0	1	2	3	4	5
J_2	6	7	8	9	10	11
J_3	12	13	14	15	16	17
J_4	18	19	20	21	22	23
J_5	24	25	26	27	28	29
J_6	30	31	32	33	34	35

Fig. 3.3 Ejemplo de operaciones disponibles al inicio de la construcción de la cadena de solución

Cabe mencionar que, en el primer paso para la construcción de la solución inicial, las operaciones disponibles para agregar a la cadena son igual a las primeras operaciones de cada trabajo, ya que son las únicas que no dependen de una operación previa para poder procesarse.

El segundo paso para la generar la solución inicial es la de seleccionar aleatoriamente una operación dentro de las que se identificaron previamente. Esta operación es agregada a la cadena de solución como parte del paso tres, tal como se muestra en la figura 3.4 y deja de considerarse dentro de la matriz de operaciones, salvo por dar paso libre a la siguiente operación de su línea de trabajo que responda a su orden de precedencia.

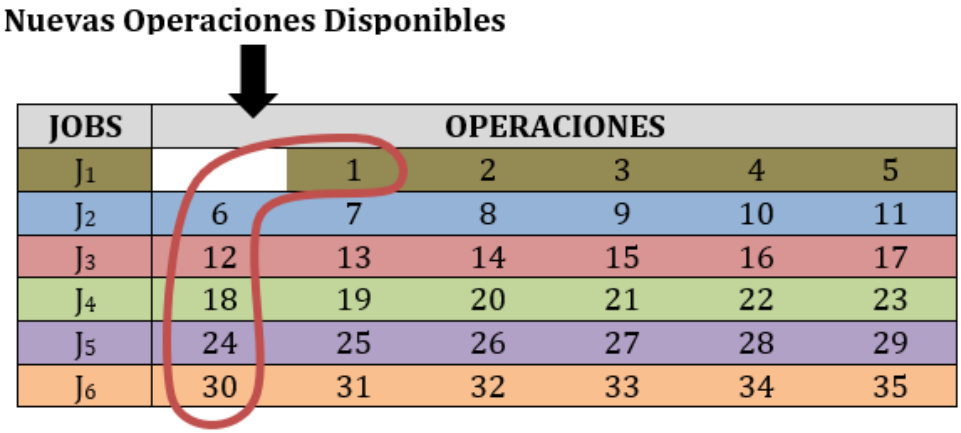


JOBS	OPERACIONES					
J_1		1	2	3	4	5
J_2	6	7	8	9	10	11
J_3	12	13	14	15	16	17
J_4	18	19	20	21	22	23
J_5	24	25	26	27	28	29
J_6	30	31	32	33	34	35

0									

Fig. 3.4 Operación seleccionada de forma aleatoria y añadida a la cadena

Para el paso cuatro se debe reconsiderar el conjunto de operaciones disponibles en la matriz de operaciones como muestra el ejemplo de la figura 3.5. Todas aquellas operaciones que ya han sido asignadas a la cadena de solución dejan de ser tomadas en cuenta. Este proceso de selección y anexión de operaciones de la matriz a la cadena de solución sigue hasta acabar con todas las operaciones disponibles para el problema.



JOBS	OPERACIONES					
J_1		1	2	3	4	5
J_2	6	7	8	9	10	11
J_3	12	13	14	15	16	17
J_4	18	19	20	21	22	23
J_5	24	25	26	27	28	29
J_6	30	31	32	33	34	35

Fig. 3.5 Nuevo subconjunto de operaciones disponibles en la matriz de operaciones

El siguiente movimiento para llegar a la cadena de solución cuyo inicio es mostrado en la tabla 3.3, se muestra en la figura 3.6. Nuevamente se selecciona de forma aleatoria una operación que no esté restringida por el orden de precedencia del trabajo al que pertenece, ya sea porque es la primera operación de dicho trabajo o porque la operación previa ya ha sido asignada a la cadena de solución. En este caso se opta por la operación 24 y se agrega en la siguiente posición dentro de la cadena de solución.

JOBS	OPERACIONES					
J_1		1	2	3	4	5
J_2	6	7	8	9	10	11
J_3	12	13	14	15	16	17
J_4	18	19	20	21	22	23
J_5		25	26	27	28	29
J_6	30	31	32	33	34	35

0	24								...

Fig. 3.6 Asignación de una segunda operación a la cadena de soluciones

Siguiendo los pasos que se han mencionado para formar la cadena de solución, eventualmente se puede llegar a una cadena tal como la del ejemplo de la tabla 3.3. Para este caso en particular, la tabla de operaciones y la cadena de solución se puede observar en la figura 3.7, donde la siguiente operación que puede ser agregada a la cadena debe ser elegida del conjunto de operaciones disponibles para ese momento.

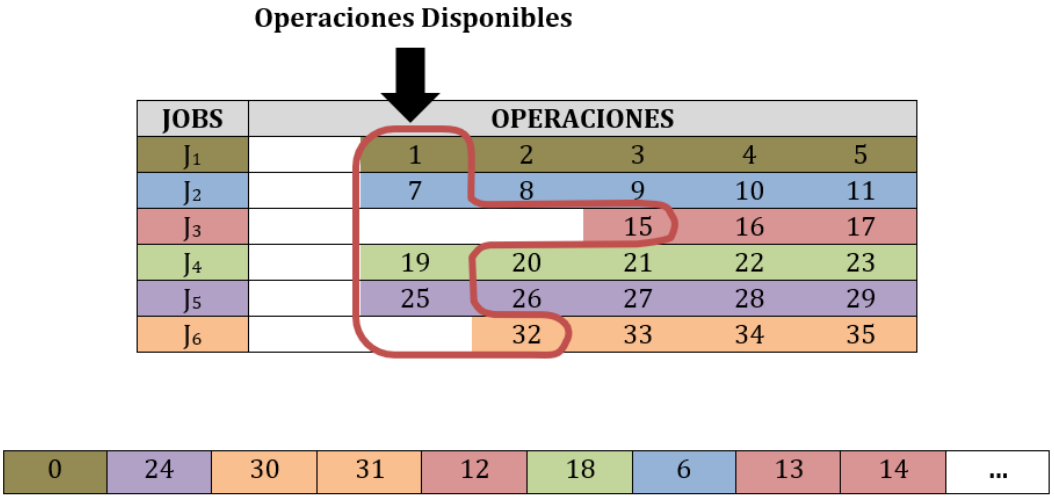


Fig. 3.7 Asignación de una nueva operación a la cadena de solución

Una vez completa la cadena de la solución inicial, es posible manipularla mediante ciertos movimientos para crear nuevas soluciones y formar un vecindario de búsqueda local.

3.3.2 Construcción de una Calendarización a partir la Cadena de Solución.

A lo largo del algoritmo es necesario comparar diferentes soluciones entre sí, evaluando su calidad para poder tomar decisiones en el proceso de optimización. Para conocer la calidad de una solución, es necesario conocer su valor de costo asociado a su función objetivo, el cual está dado por el makespan, es decir, el tiempo en el que termina de procesarse de la última operación en una calendarización. A partir de la cadena de solución descrita anteriormente, se puede construir una calendarización representada por una matriz, y así conocer el makespan de dicha solución.

La matriz de calendarización de una solución está organizada por filas, las cuales representan las máquinas disponibles para el problema, y columnas, para las operaciones. Las operaciones asignadas a cada fila son aquellas que por definición de la instancia deben ser procesadas por la máquina que representa, y la posición en la que se encuentran determina el orden en el que deben ser procesadas por dicha máquina. A continuación, la tabla 3.4 representa una matriz de calendarización para el mismo ejemplo de la instancia de 6x6 que se utilizó en la sección anterior. Las operaciones del mismo color pertenecen al mismo trabajo.

Tabla 3.4 Ejemplo de matriz de calendarización para posible solución de instancia de 6x6

Machines	Operaciones					
M_1	1	19	33	15	28	10
M_2	30	18	6	25	2	16
M_3	0	24	12	20	7	35
M_4	31	13	21	3	29	11
M_5	26	8	22	17	34	5
M_6	32	14	27	4	9	23

Para construir la matriz de calendarización a partir de una cadena de solución cualquiera, se debe consultar la maquina asociada a cada operación de acuerdo al orden en el que están organizadas en la cadena. Las operaciones son entonces asignadas en orden a la posición que

les corresponde dentro de la matriz. En la figura 3.8 se puede observar el proceso de construcción de la matriz de calendarización para la parte inicial de una cadena de solución.

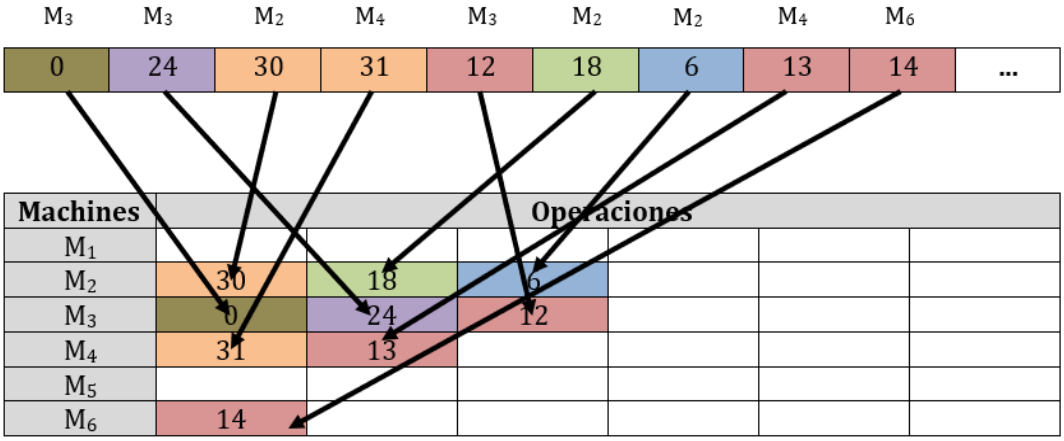


Fig. 3.8 Construcción de matriz de calendarización a partir de la cadena de solución

Una vez formada la nueva matriz, es posible obtener el makespan de la solución que estamos evaluando, es decir, realizar la calendarización. Para esto se deben calcular los tiempos de inicio y fin de cada operación, tomando en consideración la restricción de precedencia dentro de cada trabajo, así como el orden en que el que está organizada la matriz.

Para calcular los tiempos de inicio y fin de cada operación, se evalúan por columnas aquellas cuyas operaciones previas, ya sea por su posición dentro del trabajo o máquina al que están asignados, ya hayan sido previamente calculadas. En la tabla 3.5 se observa la calendarización obtenida del ejemplo mostrado anteriormente por la tabla 3.4.

Tabla 3.5 Calendarización basada en la tabla 3.4

TRABAJO	OPERACION	TIEMPO
1	0	0
1	1	1
1	2	19
1	3	25
1	4	32
1	5	54
2	6	8
2	7	20
2	8	25
2	9	35
2	10	45
2	11	55
3	12	10
3	13	15
3	14	19
3	15	27
3	16	36
3	17	43
4	18	3
4	19	8

4	20	15
4	21	20
4	22	35
4	23	45
5	24	1
5	25	16
5	26	19
5	27	27
5	28	36
5	29	39
6	30	0
6	31	4
6	32	7
6	33	16
6	34	51
6	35	55

Esta forma de evaluar nuevas soluciones resulta ventajosa, pues al construir individualmente cada cadena nos aseguramos que las nuevas soluciones sean siempre factibles. Esto siempre y cuando se respete en todo momento las restricciones de precedencia para cada trabajo y sus operaciones.

3.4 Recocido Simulado

Una vez generada la solución inicial, el siguiente paso es el de implementar un algoritmo clásico de Recocido Simulado. En esta sección se explicará a fondo el proceso que se sigue para la implementación del Recocido Simulado, así como la estructura de vecindad y el proceso de la búsqueda local requerida para dicho algoritmo.

3.4.1 Algoritmo General de Recocido Simulado

El algoritmo de Recocido Simulado empleado para este proyecto, sigue la premisa clásica de una búsqueda local de mejora iterativa, donde cada nueva solución derivada del vecindario, se evalúa con respecto a la mejor solución previa y se acepta o no bajo el criterio del algoritmo de metrópolis y el factor de probabilidad de Boltzmann

En la figura 3.9 se muestra el diagrama de flujo del algoritmo de Recocido Simulado. El diagrama puede separarse en tres partes importantes: Las condiciones iniciales, el ciclo interno y el ciclo externo.

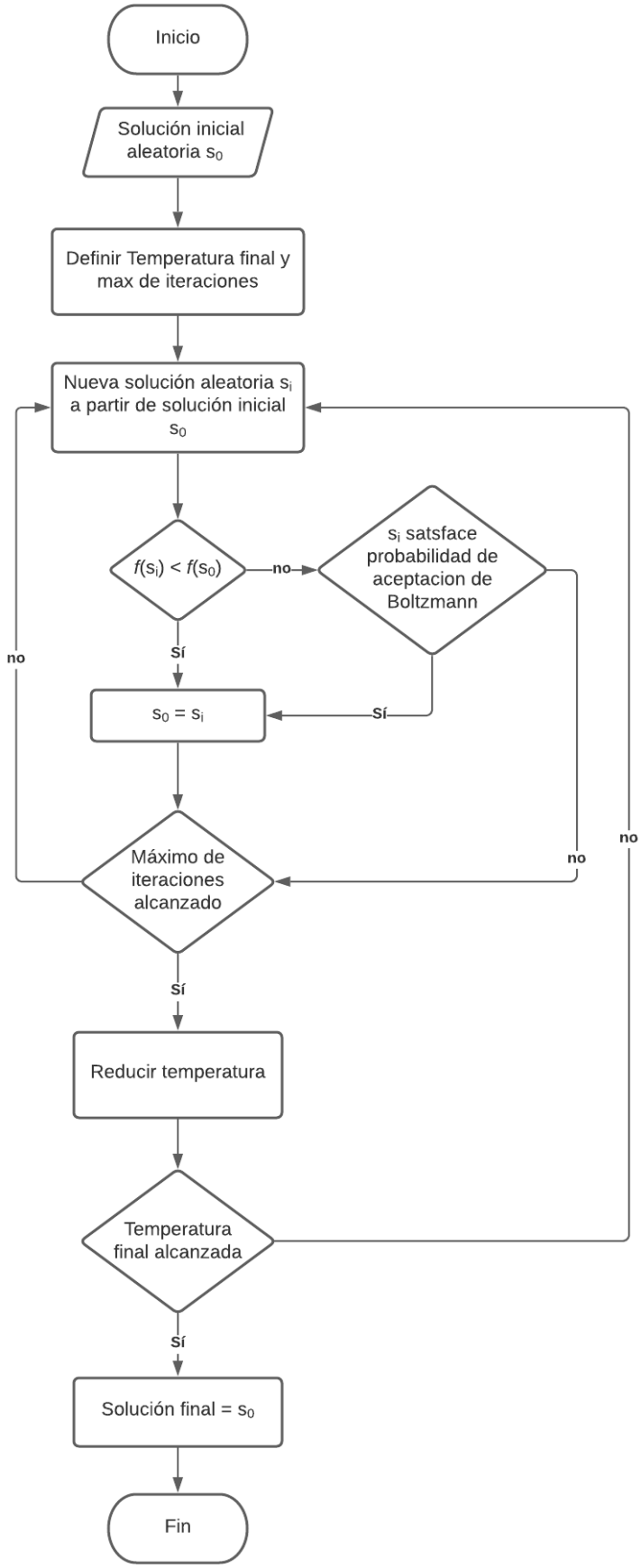


Fig. 3.9 Diagrama de flujo de algoritmo de Recocido Simulado

Dentro de las condiciones iniciales se encuentra la solución inicial y las condiciones de parada para los ciclos. La solución inicial es la que empieza el algoritmo y se trata de la primera solución construida de forma aleatoria a partir de los datos de entrada para el problema. Para las condiciones de parada se define el número máximo de iteraciones que durará el ciclo interno, así como los parámetros de temperatura que servirán de criterio para el ciclo externo, dígase la temperatura inicial, final y el factor de enfriamiento.

En el ciclo interno se realiza una búsqueda local, la cual comienza con una nueva solución aleatoria tomada del vecindario y que será evaluada con la solución previa para decidir si se acepta o rechaza. En este ciclo se encuentra también el algoritmo de metrópolis, con el factor de probabilidad de Boltzmann como criterio de aceptación para soluciones de peor calidad. Este ciclo se repite hasta alcanzar un número definido de iteraciones para después dar paso al ciclo externo.

El ciclo externo se encarga de repetir la búsqueda local del ciclo interno y reducir la temperatura del Recocido. A lo largo de cada iteración la temperatura disminuye por el factor de enfriamiento definido en las condiciones iniciales, hasta detenerse al llegar a la temperatura final.

Como resultado del Recocido Simulado se obtiene una solución final que servirá para la siguiente etapa del algoritmo general.

3.4.2 Estructura de Vecindad y Búsqueda Local Aleatoria

Una de las principales características de los problemas de optimización combinatoria que contribuye a su intratabilidad, es la enorme cantidad de posibles soluciones que existen para cada instancia específica, ya que evaluar cada una de ellas resulta impráctico. Por lo tanto, para abordar de forma más manejable este tipo de problemas es necesario diseñar métodos aproximados que sean rápidos, aunque no puedan garantizar encontrar el óptimo global. Una manera de explorar el espacio de soluciones es a partir de una estructura de vecindad. La estructura de vecindad es una forma de organizar la información proporcionada por una solución inicial, de tal manera que sea posible obtener nuevas soluciones a partir de ella. Al conjunto de nuevas soluciones obtenidas de esta manera se le conoce como vecindario y al proceso de evaluarlas en busca de mejores soluciones, como búsqueda local.

La estructura de vecindad de este algoritmo tiene como base la cadena de solución descrita en la sección anterior. La solución inicial usada para comenzar el algoritmo de Recocido Simulado está dada por la primera cadena de solución, construida aleatoriamente a partir de los datos de entrada del problema. Esta solución inicial servirá para definir el vecindario del cual se desprenderán las nuevas soluciones que se evaluarán durante el proceso de búsqueda local que se realizará a lo largo del algoritmo.

El proceso de búsqueda local se puede resumir en los siguientes pasos:

1. Definir solución inicial.
2. Seleccionar nueva solución de entre el vecindario.
3. Comparar la solución actual con la nueva solución, aceptarla si mejora la anterior.
4. Repetir desde el paso 1 hasta alcanzar un criterio establecido.

La solución inicial de acuerdo al paso 1 de la lista anterior, está dada en un principio por la cadena de solución formada por los datos de entrada del problema. Subsecuentemente, esta

solución será reemplazada por nuevas cadenas de solución que cumplan con los criterios requeridos. Para el caso del algoritmo de Recocido Simulado estos criterios están dados por el mecanismo de aceptación del algoritmo de metrópolis y la función de Boltzmann.

Para generar nuevas soluciones a partir de una cadena de solución inicial, se debe reorganizar el orden de pares de operaciones dentro de ella. Los movimientos de estos pares de operaciones definen nuevas cadenas de solución para que sean evaluadas de acuerdo a su calidad.

En la figura 3.10 se puede observar el vecindario para construir nuevas soluciones, en la que, a partir de una solución inicial, se selecciona un par de operaciones de entre la cadena. A este par de operaciones se les intercambian sus posiciones dentro de la cadena, siempre y cuando no se viole ninguna restricción de precedencia, formando así una nueva cadena de la que se puede obtener una calendarización diferente.

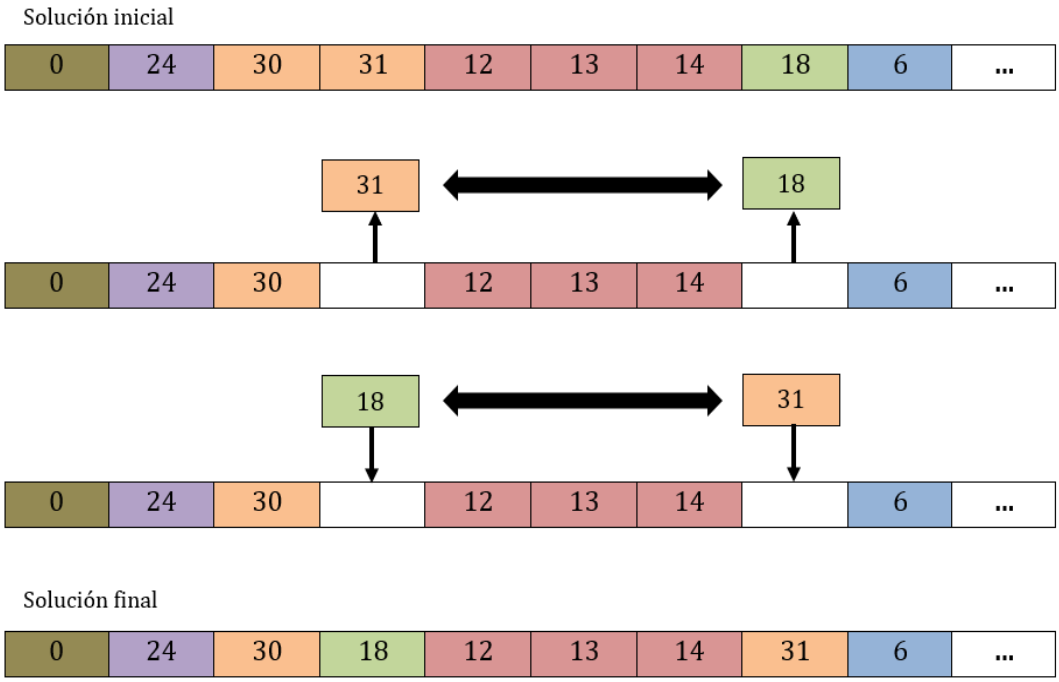


Fig. 3.10 Creación de nueva solución a partir del intercambio de un par operaciones

Para esta implementación de Recocido Simulado, los pares de operaciones son seleccionados de manera aleatoria. Sin embargo, los movimientos entre pares de operaciones están limitados por la restricción de precedencia de las mismas, evitándose dos tipos de movimientos durante la exploración del vecindario. Es decir, los movimientos válidos son aquellos en los que, al intercambiar un par de operaciones, no se viole ninguna restricción de precedencia en las operaciones de cualquier trabajo.

El primer movimiento inválido resulta de seleccionar un par de operaciones pertenecientes al mismo trabajo, sin importar si las operaciones son adyacentes o no. En la figura 3.11 se puede observar un esquema para ambos casos de este movimiento restringido.

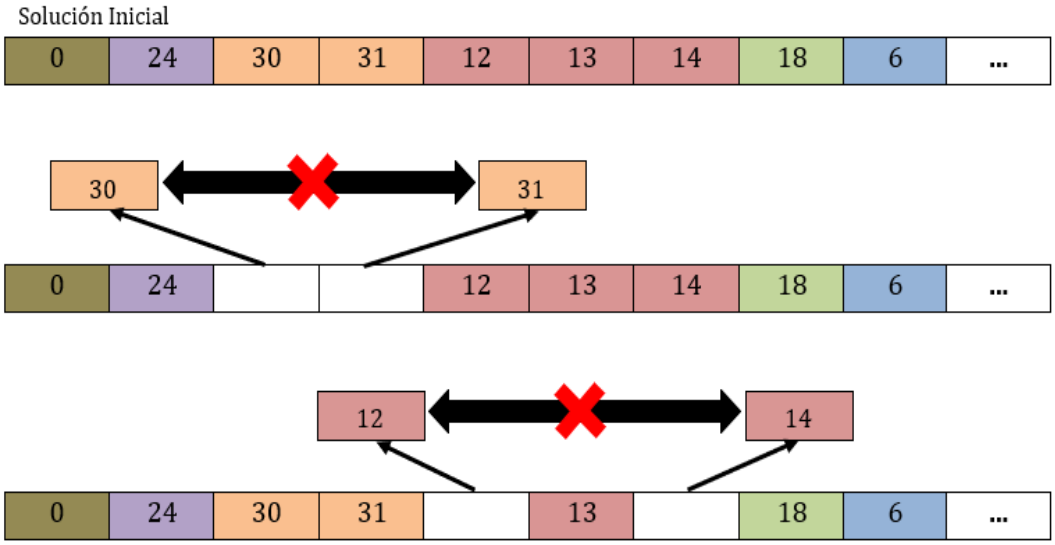


Fig. 3.11 Movimiento inválido debido a selección de operaciones de un mismo trabajo

El segundo tipo de movimiento restringido ocurre cuando se intenta intercambiar un par de operaciones, pero existen una o más operaciones intermedias que pertenecen al mismo trabajo de cualquiera de las operaciones seleccionadas. Estas operaciones intermedias evitan el movimiento de una o ambas operaciones a la derecha o a l izquierda, tal como se muestra en a figura 3.12.

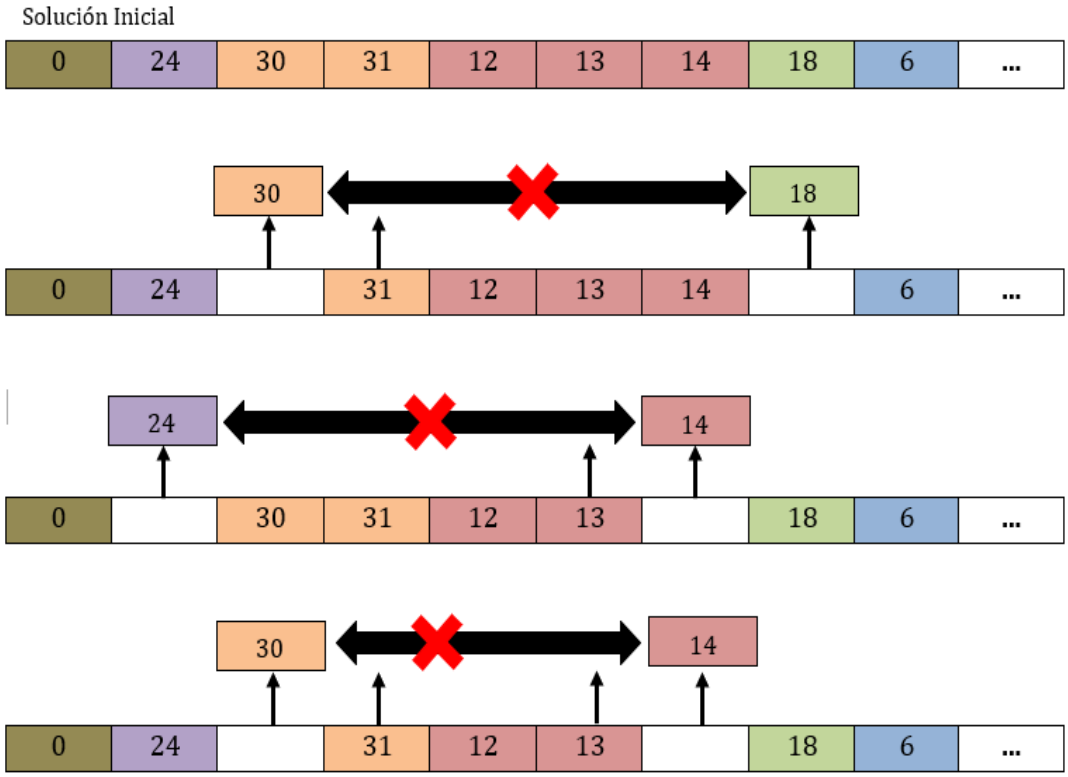


Fig. 3.12 Movimiento inválido debido a cruce de operaciones durante el cambio de pares seleccionados

3.4.3 Sintonización de Variables

Parte indispensable para el correcto desempeño de una implementación de Recocido Simulado, son los valores asignados a las variables encargadas de controlar el

comportamiento de los ciclos interno y externo del algoritmo. Estas variables corresponden a cuatro valores: Temperatura inicial (T_i), temperatura final (T_f), constante de enfriamiento (α) y la condición de parada del ciclo interno (I).

En (Siarry, 2016) se advierte no existe una regla general para definir estos valores y por lo tanto es necesario recurrir a una sintonización, es decir, a un ajuste empírico de los mismos. Para esta implementación se seleccionaron los siguientes valores arbitrarios, que servirán de punto de referencia a lo largo de la sintonización.

El valor propuesto para la temperatura inicial se fijó a que sea igual al valor de la solución inicial. Para empezar a sintonizar el algoritmo, se tomó como punto de partida los siguientes valores: La temperatura final se propuso un valor de 0.01 mientras que para la constante de enfriamiento y el número de iteraciones del ciclo interno se fijaron a 0.99 y 1000, respectivamente. Las últimas dos variables tienen una relación directa con el número de iteraciones que durará el ciclo interno y externo del algoritmo, lo que supone una mejora en la calidad de las soluciones a expensas de mayor tiempo de ejecución.

A continuación, se muestra la exploración de diferentes valores a los parámetros ya mencionados. Esta exploración se realizó con las instancias la15 (20x5), abz5 (10x10), la30(20x10) y la40(15x15). Cada instancia fue evaluada con veinte corridas independientes del algoritmo de Recocido Simulado para encontrar los valores de calidad de solución y tiempo promedio

Tabla 3.6 Sintonización de constante de enfriamiento (Alpha)

INSTANCIAS		0.8	0.85	0.9	0.99	0.999
la15	Z (best)	1250	1248	1248	1226	1218
	Z (avrg)	1274.1	1269.05	1268	1245.65	1231.85
	t	0.5434	0.74715	1.14525	11.9577	120.4175
abz5	Z (best)	1301	1296	1299	1275	1256
	Z (avrg)	1327	1323.05	1317.45	1290.8	1275
	t	0.3304	0.452	0.695	7.2275	73.35725
la30	Z (best)	1479	1483	1465	1454	1451
	Z (avrg)	1519.9	1513.55	1502.65	1482.3	1467
	t	1.0537	1.44055	2.226	23.2068	235.1403
la40	Z (best)	1393	1401	1380	1358	1340
	Z (avrg)	1423.4	1427.5	1403.7	1383.8	1363.35
	t	0.81955	1.1238	1.7233	18.0379	181.6836

En la tabla 3.7 se muestran los resultados de las pruebas para diferentes valores de Alpha. Se observa que entre menor es el valor asignado a Alpha, la calidad de la solución es mejor, lo cual es un comportamiento esperado. Sin embargo, aunque la calidad de las soluciones cuando Alpha es igual a 0.999 es la mejor de entre los parámetros evaluados, el tiempo de ejecución promedio se incrementa de forma significativa. Este incremento representa diez veces más que el registrado cuando Alpha es igual a 0.99 mientras que la mejora en la calidad no se incrementa de forma significativa, por lo tanto, se mantendrá Alpha en 0.99.

Tabla 3.7 Sintonización del número de iteraciones del ciclo interno

INSTANCIAS		100	200	300	500	1000	2000	3000	5000
la15	Z (best)	1235	1243	1228	1231	1235	1221	1226	1225
	Z (avrg)	1264.6	1261.25	1254.9	1251.6	1246.1	1242	1241.85	1235.55
	t	1.19	2.37	3.57	6.01	11.9	23.81	35.73	59.55
abz5	Z (best)	1293	1281	1269	1279	1274	1266	1272	1265
	Z (avrg)	1317	1305.3	1297.15	1302.45	1291.6	1288.3	1286.4	1277.7
	t	0.72	1.44	2.18	3.61	7.23	14.51	21.98	36.17
la30	Z (best)	1483	1479	1473	1450	1461	1440	1456	1446
	Z (avrg)	1507.55	1503.9	1494.7	1485.7	1486.5	1475.4	1472.8	1467.55
	t	2.32	4.63	6.96	11.63	23.13	46.48	69.57	116.26
la40	Z (best)	1378	1365	1376	1379	1378	1363	1347	1353
	Z (avrg)	1414.7	1401.15	1399.55	1394.9	1388.2	1379.9	1376.75	1372.8
	t	1.79	3.59	5.40	9.02	18.08	36.1	54.32	90.43

Durante las evaluaciones para definir el número de iteraciones del ciclo interno (tabla 3.8) se observó un comportamiento esperado en la relación entre el número de iteraciones y la calidad de las soluciones. El valor final se fijó en 2000 iteraciones dado que la mejora que representa 3000 iteraciones a las soluciones es de pocas unidades, mientras que el tiempo de ejecución sigue incrementando.

Tabla 3.8 Sintonización de temperatura inicial

INSTANCIAS		Z	1000	800	500
la15	Z (best)	1221	1217	1207	1218
	Z (avrg)	1242.16	1242.62	1242.18	1245.14
	t	19.24	18.75	17.99	17.04
abz5	Z (best)	1261	1259	1269	1265
	Z (avrg)	1287.34	1286.3	1288.2	1288.02
	t	11.708	11.09	10.84	10.28
la30	Z (best)	1465	1461	1457	1472
	Z (avrg)	1480.4	1478.45	1480.1	1483.15
	t	37.94	35.27	34.43	32.59
la40	Z (best)	1358	1365	1373	1355
	Z (avrg)	1381.4	1384.15	1386	1375.6
	t	29.18	27.52	26.85	25.48

Durante la sintonización de la temperatura inicial, se tomó como punto de referencia el valor de la solución inicial (Z) de cada instancia, lo que hace que el valor de esta variable dependa de la instancia que se esté resolviendo. Sin embargo, las instancias escogidas para evaluar el algoritmo inician con valores arriba de las mil unidades, generando una temperatura inicial elevada. En la tabla 3.9 se observa que las soluciones no varían significativamente, por lo que se eligió empezar con una temperatura inicial de 500 unidades para eliminar iteraciones innecesarias. Además de eso, en nuestras experimentaciones pudimos comprobar que el porcentaje de aceptación con dicho valor es superior al 95%.

Tabla 3.9 Sintonización de constante de enfriamiento

INSTANCIAS		1	0.5	0.1	0.01
la15	Z (best)	1214	1222	1218	1208
	Z (avrg)	1246.94	1245.6	1243	1241.04
	T	12.47	18.41	17.01	21.56
abz5	Z (best)	1264	1265	1265	1265
	Z (avrg)	1290.08	1291.34	1289.12	1285.54
	T	7.49	8.34	10.24	12.97
la30	Z (best)	1459	1461	1446	1459
	Z (avrg)	1481.24	1481.72	1481.24	1477
	T	23.87	26.48	32.59	41.3
la40	Z (best)	1358	1344	1364	1355
	Z (avrg)	1382.84	1385.1	1383.14	1377.9
	T	18.67	20.72	25.43	32.22

En la tabla 3.10 se muestran las evaluaciones para definir la temperatura final T_f. Durante la sintonización de este parámetro, se observó que el incremento de tiempo con relación al tamaño de T_f, es menos pronunciado que en las otras variables. Debido a esto se decidió mantener el valor de T_f en 0.01, pues es el que arroja mejores soluciones. También se comprobó que, en promedio, el porcentaje de aceptación con esa temperatura es inferior al 5%.

3.5 Hill Climbing

El algoritmo de escalado de montaña, del inglés Hill Climbing, es otro nombre con el que se conoce al proceso de Búsqueda Local, esto en alusión al comportamiento que tiene este algoritmo de moverse siempre hacia arriba, es decir, de solo seleccionar mejores soluciones.

Para el algoritmo general de este proyecto se utilizaron dos enfoques en la exploración del vecindario: Simple y con doble vecindad.

3.5.1 Algoritmo General Aleatorio

El Hill Climbing es un algoritmo de mejora iterativa, lo que significa que a lo largo de su ejecución se seleccionan únicamente soluciones con mejor calidad en comparación con la solución anterior. Al igual que el algoritmo de Recocido Simulado, el Hill Climbing parte de una solución inicial aleatoria que define el vecindario a explorar.

En la figura 3.13 se puede observar el diagrama de flujo para el algoritmo de Hill Climbing. Este diagrama puede dividirse en dos partes: Condiciones iniciales y el ciclo de búsqueda local.

En la etapa de condiciones iniciales se define la solución inicial, la cual en el algoritmo general está dada por la solución final producida al final del Recocido Simulado. Esta solución sirve para definir el vecindario a explorar y se considera como la solución actual con la que compararán las nuevas soluciones. Además, se define el número máximo de iteraciones que durará el ciclo de búsqueda, siendo esta la condición de parada del algoritmo.

Durante el ciclo de búsqueda local se selecciona una nueva solución del vecindario, la cual se evalúa para después compararse con la solución actual, siendo aceptada únicamente si es

de mejor calidad y rechazándose de lo contrario. La nueva solución se selecciona de forma aleatoria mediante el mismo proceso que el descrito en la sección 3.4.2.

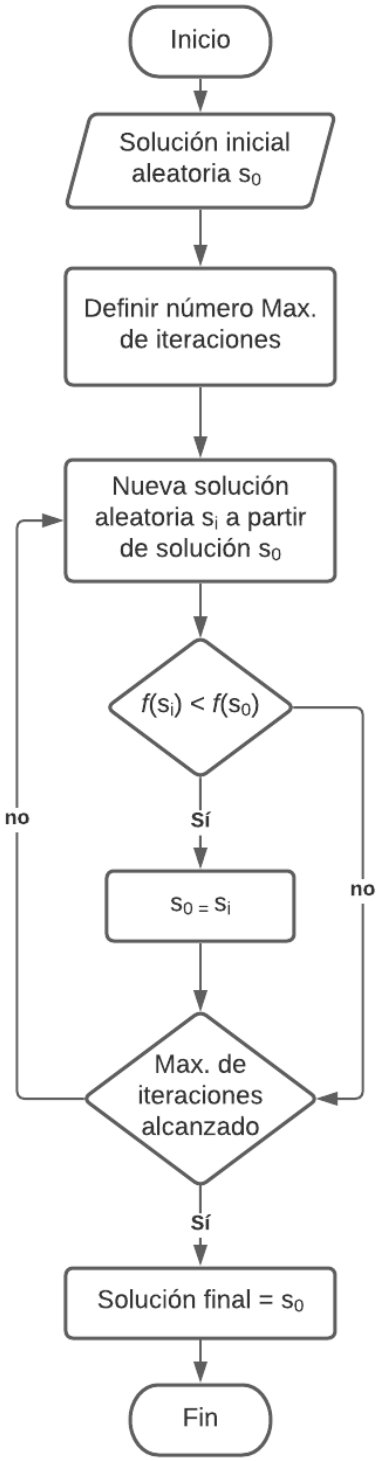


Fig. 3.13 Diagrama de Búsqueda Local (Hill Climbing)

Aunque comparten similitudes en su estructura los algoritmos de Recocido Simulado y Hill Climbing, este último no tiene ningún mecanismo de aceptación para reevaluar las soluciones rechazadas, a diferencia del primero. Esta característica convierte al Hill Climbing en un método sencillo de implementar y rápido de ejecutar, sin embargo, depende enteramente del tamaño de su vecindario para poder encontrar soluciones de calidad.

A continuación, se describe la implementación de otros enfoques para el algoritmo de Hill Climbing para este proyecto, con respecto a la obtención de nuevas soluciones a partir de la exploración del vecindario.

3.5.2 Algoritmo de Hill Climbing Determinista

Esta versión del algoritmo de Hill Climbing comparte las mismas características generales que el descrito en la sección anterior. La diferencia radica entonces en la forma en la que se explora el vecindario para obtener nuevas soluciones.

El término determinista usado para describir el algoritmo, deriva del patrón de exploración, en el que se realiza una búsqueda exhaustiva y ordenada del vecindario. Esta búsqueda inicia con el intercambio ordenado de pares de operaciones sobre la solución inicial, seguido de un segundo tipo de cambios, igualmente ordenados, denominados corrimientos. Este procedimiento itera hasta encontrar un mejor candidato o hasta recorrer el vecindario en su totalidad.

En la figura 3.14 se muestra un diagrama de flujo del algoritmo, en el que se aprecia el mecanismo de exploración del vecindario. Al igual que el diagrama anterior, este se puede dividir en dos etapas: Condiciones iniciales y ciclo de búsqueda.

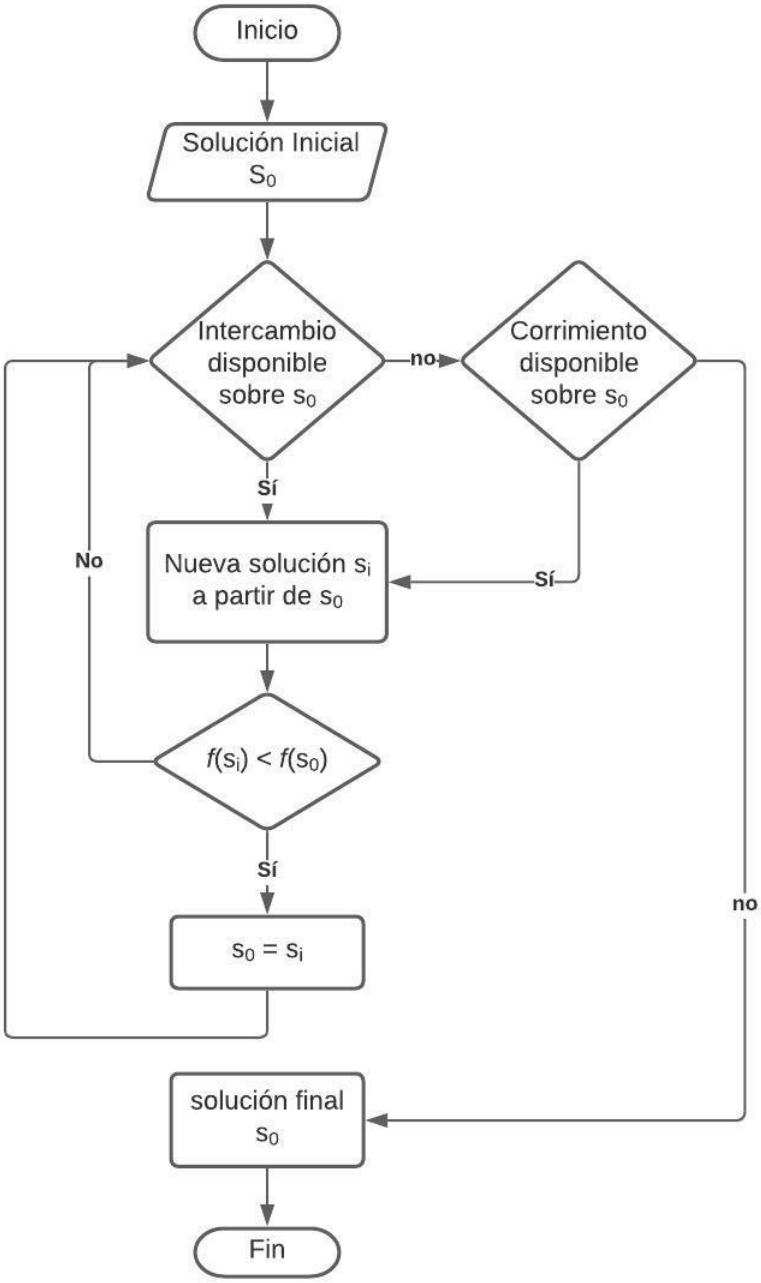


Fig. 3.14 Diagrama de flujo de Hill Climbing Determinista

A diferencia del método de intercambio aleatorio usado por el algoritmo de Recocido Simulado y la versión anterior de Hill Climbing, este algoritmo no requiere definir una condición de parada externa para detener el ciclo de búsqueda, ya que esta continúa hasta no poder encontrar una mejor solución en el vecindario.

3.5.2.1 Intercambios y corrimientos en la estructura de vecindad

Este procedimiento de búsqueda local parte de una solución inicial, la cual está dada como una cadena de solución y que define el vecindario sobre el que se realiza la búsqueda local. Este procedimiento consta de movimientos ordenados entre pares de operaciones, los cuales se efectúan hasta encontrar una mejor solución o acabar con las opciones brindadas por el vecindario.

Para agotar todas las posibles soluciones proporcionadas por el vecindario, este método de búsqueda consiste en dos etapas: Intercambios y corrimientos.

La etapa de intercambios es similar en procedimiento a lo mostrado en la sección 3.4.2, siendo la única diferencia el criterio de selección de los pares de operaciones. Para este método de búsqueda, la selección de pares se realiza siguiendo un orden predefinido, realizando todas las combinaciones posibles dentro de la cadena de solución. En caso de no encontrar una solución de mejor calidad durante la etapa de intercambios, se procede con la etapa de corrimientos

Durante la etapa de corrimientos se selecciona una a una las operaciones de la cadena de solución inicial, adelantando su posición por cada iteración, formando así nuevas soluciones. Durante la etapa de corrimientos, se selecciona una operación para realizar la modificación y la nueva posición a la que se trasladará. Todas las operaciones que se encuentran entre la operación y la posición seleccionadas, se correrán de forma uniforme para permitir el movimiento, manteniendo al final el orden en el que se encuentran. En la figura 3.15 se muestra un ejemplo del proceso de corrimiento producto del movimiento de la primera operación hacia la quinta posición. En esta figura y en las subsecuentes, se representa las operaciones de la cadena de solución con letras en lugar del número de operación, para identificar las posiciones originales de la cadena inicial, así como el resultado de los intercambios realizados

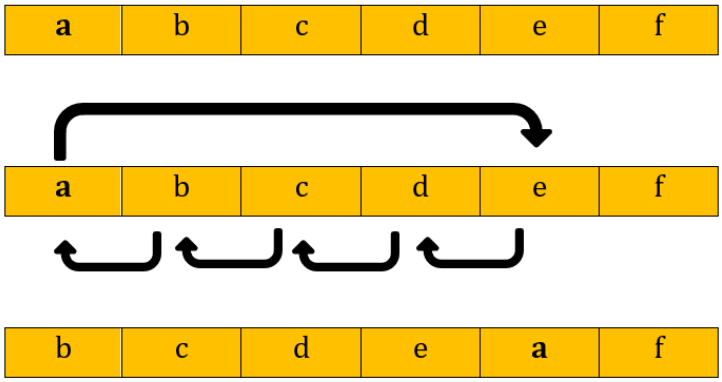


Fig. 3.15 Ejemplo de corrimiento sobre una cadena

Una vez que se agotan las posibles combinaciones ofrecidas por la solución inicial durante la etapa de corrimiento, el proceso de búsqueda termina, arrojando la última solución obtenida.

Tanto para la etapa de intercambio como de corrimiento, las restricciones de precedencia deben respetarse durante la selección de operaciones, teniendo en cuenta las limitaciones expuestas en la sección 3.4.2.

3.5.3 Búsqueda local de Doble Vecindad

Esta versión comparte la misma estructura que el algoritmo de Hill Climbing de la sección anterior, en cuanto que está conformado por un ciclo de búsqueda exhaustivo del vecindario para encontrar nuevas soluciones a partir de una solución inicial. La principal característica de este algoritmo es su método de búsqueda, que amplía enormemente el tamaño del vecindario a explorar.

En la figura 3.16 se muestra el diagrama de flujo del algoritmo. En este diagrama se puede observar cómo se añade un ciclo interno de búsqueda idéntico al ciclo externo. El ciclo de búsqueda externo realiza cada una de las posibles combinaciones a partir de una solución inicial, generando para cada una de ellas una solución temporal, la cual se convierte a su vez en una solución inicial que alimenta el ciclo de búsqueda interno.

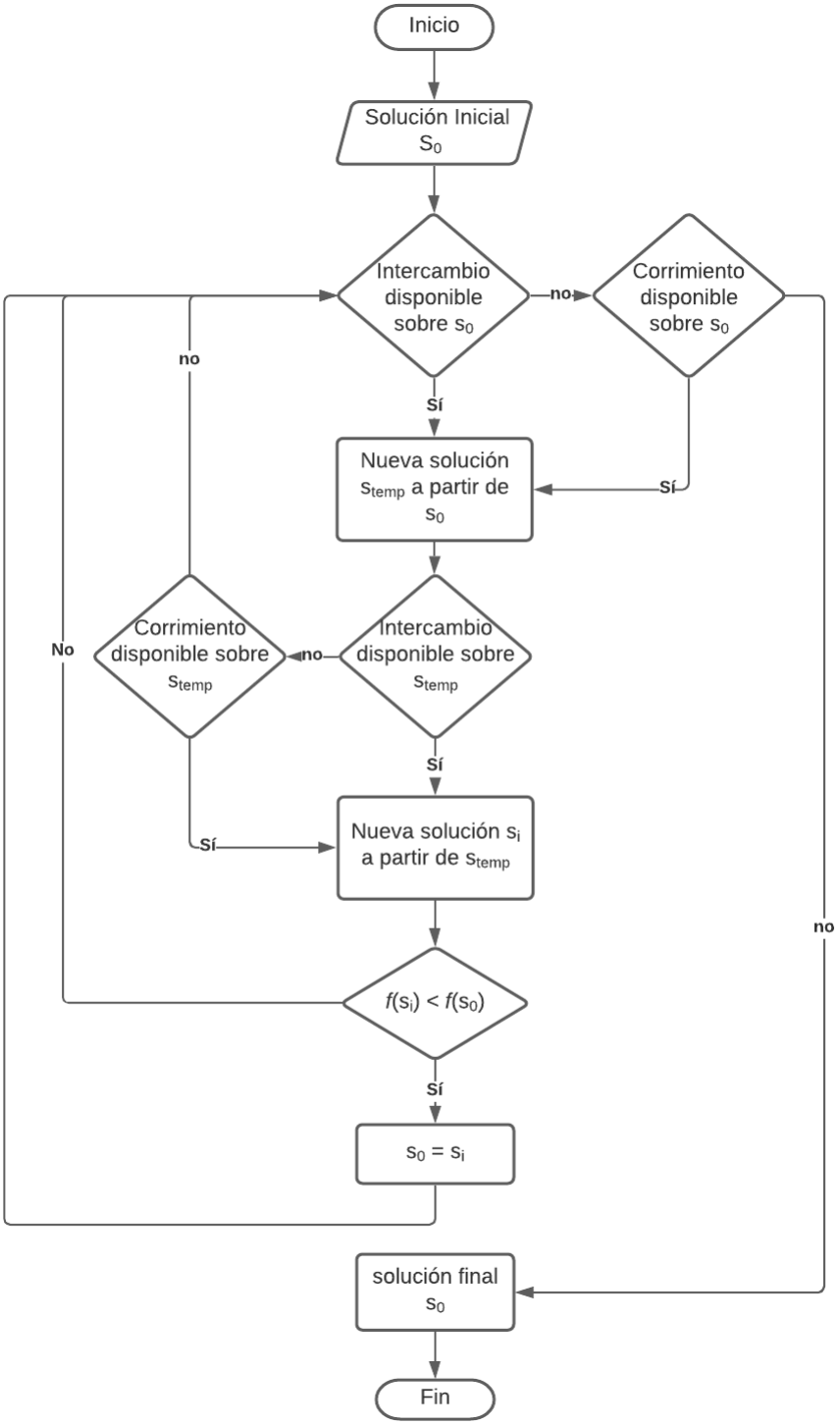


Fig. 3.16 Diagrama de flujo de Hill Climbing con Doble Vecindad

Las soluciones obtenidas por el ciclo de búsqueda interno se comparan con la solución inicial, reemplazándola en caso de ser de mejor calidad o avanzando el ciclo externo en caso de agotar las combinaciones posibles. El ciclo de búsqueda externo continúa iterando, generando nuevos ciclos de búsqueda internos hasta encontrar una mejor solución o hasta acabar con todas las posibilidades ofrecidas por la doble vecindad.

El método de búsqueda de ambos ciclos está organizado de la misma manera que la descrita en la sección anterior, realizando cambios ordenados de operaciones mediante una serie de intercambios y corrimientos. La tabla 3.10 ejemplifica parte del comportamiento que siguen los cambios de operaciones de la Búsqueda Local simple, y de la cual tienen como base los ciclos de búsqueda externo e interno de la Búsqueda con Doble Vecindad. El ciclo de búsqueda genera cada combinación de la tabla durante cada iteración, siempre que dicha modificación no contradiga las restricciones de precedencia del problema, y finaliza hasta encontrar una mejor solución o hasta acabar con la última combinación posible.

Tabla 3.10 Búsqueda Local exhaustiva mediante combinaciones posibles

s inicial	a	b	c	d	e	f
intercambio 1	**b**	**a**	c	d	e	f
intercambio 2	**c**	b	**a**	d	e	f
intercambio 3	**d**	b	c	**a**	e	f
intercambio 4	**e**	b	c	d	**a**	f
intercambio 5	**f**	b	c	d	e	**a**
intercambio 6	a	**b**	**c**	d	e	f
intercambio n	...					
corrimiento 1	b	**a**	c	d	e	f
corrimiento 2	b	c	**a**	d	e	f
corrimiento 3	b	c	d	**a**	e	f
corrimiento 4	b	c	d	e	**a**	f
corrimiento 5	b	c	d	e	f	**a**
corrimiento 6	a	c	**b**	d	e	f
corrimiento n	...					

De esta forma, la Búsqueda Local de Doble Vecindad toma cada una de las combinaciones producidas por la búsqueda Local Simple y las convierte en soluciones iniciales sobre las cuales se realiza una nueva Búsqueda Local.

En la tabla 3.11 se puede observar de forma gráfica el comportamiento que sigue la Búsqueda Local de Doble Vecindad, así como la ejecución de los ciclos de búsqueda externo e interno. En esta representación, los bloques de color azul y verde corresponden a las iteraciones del ciclo externo, mientras que los bloques rojo y amarillo representan el ciclo interno.

Tabla 3.11 Representación gráfica de la revisión sistemática de la Búsqueda Local con Doble Vecindad

S inicial		a	b	c	d	e	f
Intercambios	Intercambio 1	**b**	**a**	c	d	e	f
	Intercambio 1	**a**	**b**	c	d	e	f
	Intercambio 2	**c**	a	**b**	d	e	f
	Intercambio 3	**d**	a	c	**b**	e	f
	Intercambio n	...					

	Corrimiento 1	a	**b**	c	d	e	f
	Corrimiento 2	a	c	**b**	d	e	f
	Corrimiento 3	a	c	d	**b**	e	f
	Corrimiento n	…					
	Intercambio 2	**c**	b	**a**	d	e	f
	Intercambio 1	**b**	**c**	a	d	e	f
	Intercambio 2	**a**	b	**c**	d	e	f
	Intercambio 3	**d**	a	b	**c**	e	f
	Intercambio n	…					
	Corrimiento 1	b	**c**	a	d	e	f
	Corrimiento 2	b	a	**c**	d	e	f
	Corrimiento 3	b	a	d	**c**	e	f
	Corrimiento n	…					
	Intercambio n	…					
	Intercambio n	…					
	Corrimiento n	…					
Corrimientos	Corrimiento 1	b	**a**	c	d	e	f
	Intercambio 1	**a**	**b**	c	d	e	f
	Intercambio 2	**c**	a	**b**	d	e	f
	Intercambio 3	**d**	a	c	**b**	e	f
	Intercambio n	…					
	Corrimiento 1	a	**b**	c	d	e	f
	Corrimiento 2	a	c	**b**	d	e	f
	Corrimiento 3	a	c	d	**b**	e	f
	Corrimiento n	…					
	Corrimiento 2	b	c	**a**	d	e	f
	Intercambio 1	**c**	**b**	a	d	e	f
	Intercambio 2	**a**	c	**b**	d	e	f
	Intercambio 3	**d**	c	a	**b**	e	f
	Corrimiento n	…					
	Corrimiento 1	c	**b**	a	d	e	f
	Corrimiento 2	c	a	**b**	d	e	f
	Corrimiento 3	c	a	d	**b**	e	f
	Corrimiento n	…					
	Corrimiento n	…					
	Intercambio n	…					
	Corrimiento n	…					

Como se puede observar en la tabla, la implementación de este mecanismo de Búsqueda Local incrementa enormemente el vecindario, brindando más opciones para encontrar mejores soluciones durante la exploración, al mismo tiempo que mantiene un marco de referencia para obtenerlas.

4 EXPERIMENTACIÓN Y RESULTADOS

4.1 Introducción

A lo largo de este capítulo se encuentran los resultados obtenidos a partir del algoritmo propuesto. Este algoritmo fue ejecutado utilizando una computadora tipo laptop con las características que se muestran en la tabla 4.1.

Tabla 4.1 Descripción del equipo utilizado

PROCESADOR	MEMORIA	SIST. OPERATIVO
Intel i5-8300H - 2.30GHz	12 GB	Windows 10

Los resultados que se mostrarán a continuación se obtuvieron al evaluar las instancias listadas en la tabla 4.2. Las instancias se encuentran dispuestas por nombre, tamaño de la instancia según el número de trabajos y máquinas, y el valor numérico del optimo conocido de la instancia. Las instancias del bloque *abz* se tomaron de (Adams, Balas & Zawack, 1988), las instancias *ft* de (Fisher & Thompson, 1963), las instancias *la* de (Lawrence, 1984) y la instancia *yn* de (Yamada & Nakano, 1992).

Tabla 4.2 Lista de instancias a evaluar

INSTANCIA	TAMAÑO	OPT	INSTANCIA	TAMAÑO	OPT
abz5	10x10	**1234**	la11	20x5	**1222**
abz6	10x10	**943**	la12	20x5	**1039**
abz7	20x15	**656**	la13	20x5	**1150**
abz8	20x15	**645**	la14	20x5	**1292**
ft06	6x6	**55**	la15	20x5	**1207**
ft10	10x10	**930**	la16	10x10	**945**
ft20	20x5	**1165**	la17	10x10	**784**
la01	10x5	**666**	la18	10x10	**848**
la02	10x5	**655**	la19	10x10	**842**
la03	10x5	**597**	la20	10x10	**902**
la04	10x5	**590**	la21	15x10	**1046**
la05	10x5	**593**	la25	15x10	**977**
la06	15x5	**926**	la28	20x10	**1216**
la07	15x5	**890**	la30	20x10	**1355**
la08	15x5	**863**	la36	15x15	**1268**
la09	15x5	**951**	la40	15x15	**1222**
la10	15x5	**958**	yn1	20x20	**886**

4.2 Comportamiento del algoritmo

El algoritmo propuesto, tal como se explica a detalle en el capítulo 3, está conformado por cuatro fases, a través de las cuales se busca mejorar progresivamente hasta llegar a la solución final.

Durante el proceso de sintonización de parámetros para la fase de Recocido Simulado, se realizaron una serie de pruebas con el algoritmo completo. Para estas pruebas se utilizaron

los parámetros propuestos inicialmente, así como los obtenidos después de la sintonización. En la tabla 4.3 se muestran los resultados alcanzados por el algoritmo completo, con los parámetros de la fase de Recocido Simulado sin sintonizar y ya sintonizados. Se realizaron cincuenta corridas para obtener el resultado promedio de cada instancia. Las soluciones resaltadas en negro muestran un óptimo alcanzado, mientras que el tiempo esta medido en segundos.

Tabla 4.3 Comparativa de resultados obtenidos antes y después de sintonizar parámetros del Recocido Simulado, integrado al algoritmo final.

INSTANCIA	SIN SINTONIZAR			SINTONIZADO		
	MENOR	PROMEDIO	TIEMPO	MENOR	PROMEDIO	TIEMPO
abz5	1242	1273.84	30.45	1239	1269.48	36.55
abz6	948	975.78	27.76	**943**	970.72	36.6
abz7	709	725.4	24447.9	702	724	11117.38
abz8	714	746.82	23704.87	717	746.22	9360.23
ft06	**55**	**55**	1.78	**55**	55	3.65
ft10	955	991.74	31.55	955	993.18	33.51
ft20	1184	1211.08	291.01	1183	1205.46	280.33
la01	**666**	666.02	5.23	**666**	666	8.5
la02	**655**	670.82	6.56	**655**	668.86	9.53
la03	**597**	618.4	5.73	**597**	615.8	9.34
la04	**590**	603.56	5.75	**590**	602.16	8.89
la05	**593**	593	5.12	**593**	593	8.49
la06	**926**	**926**	19.86	**926**	926	24.9
la08	**863**	**863**	22.29	**863**	863.06	27.01
la10	**958**	**958**	20.04	**958**	958	25.05
la13	**1150**	**1150**	112.29	**1150**	1150	120.82
la15	**1207**	**1207**	127.78	**1207**	1207	141.23
la21	1088	1125.76	342.48	1078	1118	376.54
la25	995	1044.2	371.25	994	1046.04	336.39
la28	1243	1288.7	2726.76	1242	1290.8	2997.51
la30	**1355**	1394.3	2322.33	**1355**	1389.7	2546.6
la36	1297	1365.62	918.32	1305	1367.84	936.79
la40	1299	1326.08	2887.4	1259	1327.72	1169.6
yn1	964	991.82	63196.69	956	988.14	25255.86

En la tabla anterior se puede observar que los resultados arrojados por el algoritmo en el que los parámetros de la fase de Recocido Simulado fueron sintonizados, son mejores en la menor solución que aquellos producidos por el algoritmo con parámetros sin sintonizar, con la excepción de las instancias abz8 y la36. Esto se puede explicar por las características aleatorias que son parte, tanto de la fase de construcción de solución inicial, como del mismo Recocido Simulado. No obstante, la mejora en los resultados se observa en el resto de la tabla.

A continuación, en la figura 4.1 se muestra una gráfica de la diferencia de los resultados obtenidos por el promedio de las soluciones arrojadas por el algoritmo sin sintonizar y el sintonizado. En esta gráfica, un valor positivo refleja una mejor calidad en las soluciones encontradas por el algoritmo sintonizado, mientras que un valor negativo implica mejores soluciones encontradas por el algoritmo sin sintonizar.

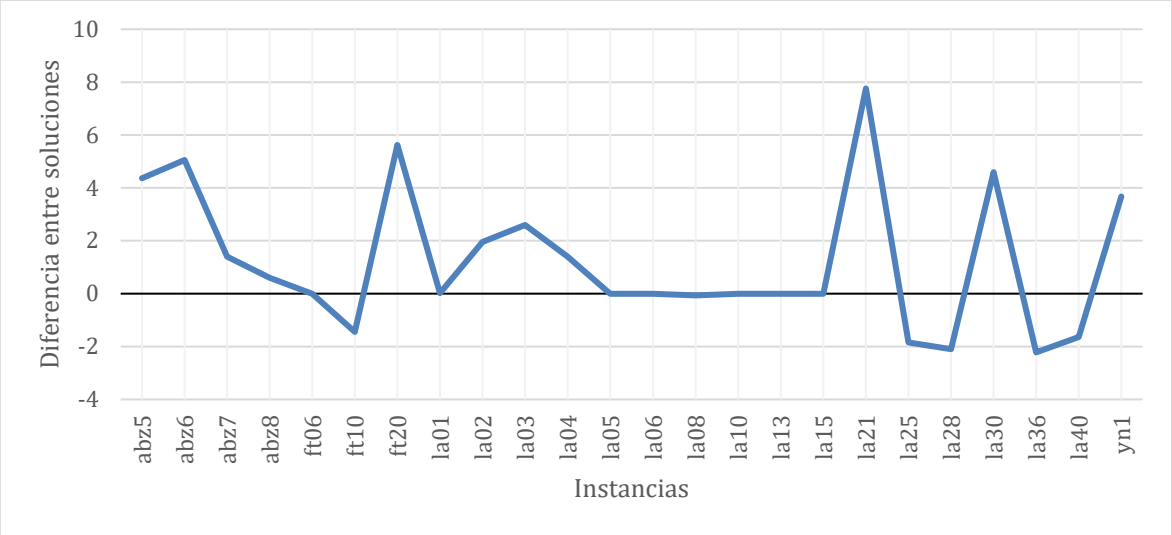


Fig. 4.1 Diferencia de las soluciones promedio de comportamiento entre algoritmo sin sintonizar y sintonizado

Tal como se puede observar, de las 24 instancias usadas para evaluar el comportamiento de la sintonización, 11 instancias tuvieron mejores resultados después de haber sido sintonizado, 5 instancias reportaron mejores soluciones sin sintonizar, mientras que 8 instancias se mantuvieron iguales o con una diferencia de menos de 0.01 unidades.

Otro detalle que se puede notar es que, aunque los resultados de las pruebas con parámetros sintonizados son mejores que los no sintonizados, la diferencia es de pocas unidades (8 en el mayor). Esto se puede deber al comportamiento de algoritmo, donde gran parte de la mejora de la solución final ocurre durante la fase de Hill Climbing con doble vecindad. Este proceso de mejora se puede apreciar en la tabla 4.4, donde se muestra el desarrollo de cuatro instancias de prueba de diferentes tamaños. Para cada una de las instancias se muestra la solución promedio de treinta corridas del algoritmo.

Tabla 4.4 Mejora de solución a través del algoritmo. Donde Z es el valor promedio de solución y T el tiempo promedio en segundos.

INSTANCIA		INICIAL	SA	HC	DV
la15	Z	1484.83	1238.20	1221.1	1207
	t	-	21.62	0.07	114.89
abz5	Z	1644.23	1291.43	1286.53	1272.6
	T	-	13.05	0.01	21.5
la30	Z	1830.97	1478.87	1460.03	1388.27
	T	-	41.42	0.27	2600.58
la40	Z	1780.77	1378.03	1366.47	1323.8
	t	-	32.19	0.15	1090.64

En la tabla se puede apreciar que en todas las fases de la propuesta algorítmica se obtiene una mejora de la solución. En la primera fase se obtiene una solución inicial factible con un algoritmo aleatorio. En la siguiente fase se mejora la solución con un Algoritmo de Recocido Simulado (SA). La mejor solución que obtiene Recocido Simulado, se manda a un algoritmo de Hill Climbing (HC) para llegar a un óptimo local. Finalmente, la salida de Hill Climbing se envía a un algoritmo de Hill Climbing con Doble Vecindad (DV) para escapar del óptimo

local en el que pudiera estar atrapado y realizar una búsqueda en más vecindarios. La solución que obtenga DV es la salida final del algoritmo.

4.3 Resultados

A continuación, se muestran los resultados obtenidos tras evaluar instancias de la literatura con la propuesta algorítmica de Recocido Simulado con una Mejora Iterativa de Doble Vecindad.

Los valores óptimos usados en la comparación de resultados para las instancias *ft* y *la* se obtuvieron de (Pongchairerks, 2019); el óptimo de las instancias *abz* fue tomado de (Satake et al, 1999), mientras que para la instancia *yn* se consultó (Cruz-Chávez et al., 2019). Los datos presentados se obtuvieron tras la evaluación de 50 corridas del algoritmo para cada instancia evaluada, registrando el valor de la menor solución obtenida, así como el promedio de todos resultados. Además, se presenta la desviación estándar relacionada al promedio, así como el error relativo en porcentaje (ecuación 4.1), dado por la diferencia de la mejor solución encontrada y la solución óptima.

$$\varepsilon_r = \frac{\varepsilon_a}{V_{opt}}$$

Ecuación 4.1 Formula del error relativo

En la tabla 4.5 se muestran los datos obtenidos de la experimentación, donde se puede apreciar que, de las 34 instancias evaluadas, se obtuvieron 18 óptimos (52.9%). En la mayoría de las instancias, se obtuvo un error relativo al óptimo menor al 5%. Solamente en las instancias abz7, abz8 y yn1 el error relativo al óptimo fue superior al 5%. Se puede observar que en general, la propuesta algorítmica se desempeña de forma eficaz para resolver varias instancias del Job Shop Scheduling Problem. El tiempo que se tarda en finalizar todo el algoritmo es admisible en la mayoría de los casos. El error relativo con respecto al óptimo fue de menos de 5% en 31 de las 34 instancias utilizadas (el 91.17% de las instancias), con lo que se comprueba la hipótesis planteada.

Tabla 4.5 Resultados experimentales comparados con el óptimo conocido de las instancias usadas

INSTANCIA	OPT.	MENOR	PROMEDIO	DESV. EST.	Err. (%)	TIEMPO
abz5	**1234**	1239	1269.48	13.61	0.41%	36.55
abz6	**943**	**943**	970.72	10.43	0.00%	36.6
abz7	**656**	702	724	9.55	7.01%	11117.38
abz8	**645**	717	746.22	10.16	11.16%	9360.23
ft06	**55**	**55**	**55**	0	0.00%	3.65
ft10	**930**	955	993.18	17.67	2.69%	33.51
ft20	**1165**	1183	1205.46	14.71	1.55%	280.33
la01	**666**	**666**	**666**	0	0.00%	8.5
la02	**655**	**655**	668.86	8.28	0.00%	9.53
la03	**597**	**597**	615.8	7.49	0.00%	9.34
la04	**590**	**590**	602.16	5.5	0.00%	8.89
la05	**593**	**593**	**593**	0	0.00%	8.49
la06	**926**	**926**	**926**	0	0.00%	24.9
la07	**890**	**890**	890.02	0.14	0.00%	26.53
la08	**863**	**863**	863.06	0.31	0.00%	27.01
la09	**951**	**951**	**951**	0	0.00%	25.97
la10	**958**	**958**	**958**	0	0.00%	25.05
la11	**1222**	**1222**	**1222**	0	0.00%	117.74
la12	**1039**	**1039**	**1039**	0	0.00%	118.83
la13	**1150**	**1150**	**1150**	0	0.00%	120.82
la14	**1292**	**1292**	**1292**	0	0.00%	113.21
la15	**1207**	**1207**	**1207**	0	0.00%	141.23
la16	**945**	946	975.82	9.74	0.11%	28.07
la17	**784**	785	799.18	8.05	0.13%	879.12
la18	**848**	859	879.12	11.02	1.30%	30.82
la19	**842**	855	885.4	10.43	1.54%	33.16
la20	**902**	907	932.68	11.78	0.55%	31.99
la21	**1046**	1078	1118	21.49	3.06%	376.54
la25	**977**	994	1046.04	20.4	1.74%	336.39
la28	**1216**	1244	1293.5	25.03	2.30%	2509.37
la30	**1355**	**1355**	1389.7	18.25	0.00%	2546.6
la36	**1268**	1305	1367.84	25.66	2.92%	936.79
la40	**1222**	1259	1329.84	24.99	3.03%	1074.56
yn1	**884**	956	988.14	14.14	8.14%	25255.86

La relación entre los resultados obtenidos y los valores óptimos de cada instancia se pueden apreciar en la siguiente gráfica (figura 4.2), donde la línea azul corresponde al optimo conocido, la línea verde a la mejor solución alcanzada por el algoritmo, la línea amarilla al promedio y la línea gris a la peor solución alcanzada para cada instancia.

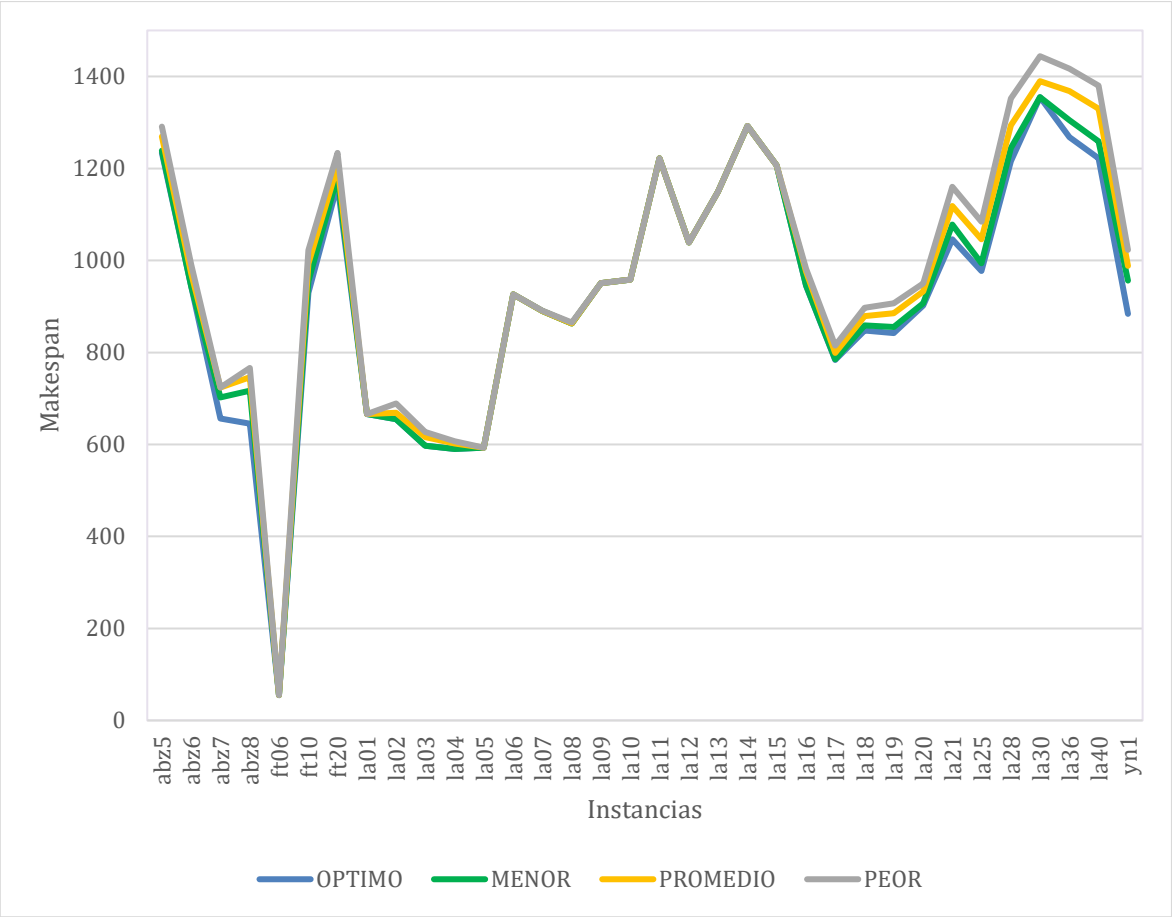


Fig. 4.2 Gráfica de resultados de la propuesta algoritmica en relación con los optimos reportados en la literatura

Con esta gráfica se puede comparar de manera visual la calidad de las soluciones encontradas, esto en relación con su proximidad con los resultados óptimos para cada instancia (línea azul). De la gráfica se observa que los resultados se encuentran muy próximos unos de otros en la mayoría de las instancias. Las líneas verde, amarilla y gris, correspondientes a los resultados del algoritmo, muestran una consistencia entre la mejor y peor solución para cada caso. Por su parte la línea de la mejor solución encontrada (verde) se encuentra especialmente cerca de la línea azul, lo que significa una aproximación al optimo. En varios segmentos de la gráfica se observa un solapamiento de ambas líneas, verde y azul, lo que corresponde a una coincidencia con los óptimos comparados de la literatura. Para varias instancias este solapamiento se extiende para todas las medidas, lo que significa que todas las corridas arrojaron el óptimo.

5 CONCLUSIONES Y TRABAJO FUTURO

5.1 Conclusiones

Se presentó un algoritmo heurístico de cuatro fases para resolver el problema de Calendarización en Talleres de Manufactura (JSSP), el cual consiste de una solución inicial aleatoria factible, la cual es mejorada a continuación por una implementación de Recocido Simulado y dos instancias de Mejora Iterativa mediante Búsqueda Local (Hill Climbing), una de las cuales implementa una estructura de doble vecindad para incrementar el tamaño del vecindario durante la exploración.

Para evaluar el algoritmo se resolvieron 34 instancias de diferentes tamaños para el problema de JSSP y los resultados fueron comparados con los óptimos conocidos de dichas instancias. Estas instancias fueron tomadas de los benchmarks *abz*, *ft*, *la* y *yn*.

Este algoritmo fue capaz resolver las instancias con las que fue evaluado, alcanzando el óptimo conocido en la mayoría de los casos. La calidad de las soluciones que no alcanzaron el óptimo tienen una diferencia menor al 5% con la excepción de las instancias abz7 con 7.01%, yn1 con 7.9%, y la abz8, siendo la más alejada con 11.16%. Por último, se puede observar que para las instancias más grandes el tiempo de ejecución se incrementa de forma cuantiosa, lo cual se puede atribuir al uso de la estructura de Doble Vecindad que amplía el vecindario a explorar durante el proceso de búsqueda. Esto puede representar una desventaja para el algoritmo dependiendo del grado en que se considere el tiempo como un factor limitante.

5.2 Trabajo futuro

A continuación, se enlistan algunas propuestas para trabajos futuros que no pudieron realizarse en esta tesis por cuestiones de tiempo o por la complejidad del problema:

- Evaluar el algoritmo con más instancias de la literatura.
- Evaluar el grado en el que intercambiar el orden de las fases del algoritmo pueda mejorar o empeorar el resultado de las soluciones.
- Integrar la estructura de doble vecindad a la búsqueda efectuada por el Recocido Simulado.
- Implementar una paralelización del algoritmo para disminuir los tiempos de ejecución de las instancias más grandes.
- Reemplazar la fase de Recocido Simulado con otra metaheurística como Búsqueda Tabú para evaluar su desempeño.

REFERENCIAS

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

Anuar, N. I., Fauadi, M. H., & Saptari, A. (2019). Performance Evaluation of Continuous and Discrete Particle Swarm Optimization in Job-Shop Scheduling Problems. *IOP Conference Series: Materials Science and Engineering*, *530.* doi:10.1088/1757-899x/530/1/012044

Avner, S. H. (1988). *Introducción a la metalurgia física* (2da ed.). MacGraw-Hill.

Barnes, J. W., & & Chambers, J. B. (1995). Solving the job shop scheduling problem with tabu search. *IIE Transactions, 27*(2), 257–263. doi:10.1080/07408179508936739

Bertsimas, D., & Tsitsiklis, J. (1993). Simulated Annealing. *Statistical Science, 8*(1), 10–15. doi:10.1214/ss/1177011077

Blum, C., & Sampels, M. (2004). An Ant Colony Optimization Algorithm for Shop Scheduling Problems. *Journal of Mathematical Modelling and Algorithms, 3*(3), 285–308. doi:10.1023/b:jmma.0000038614.39977.6f

Chaouch, I., Driss, O. B., & Ghedira, K. (2017). A Modified Ant Colony Optimization algorithm for the Distributed Job shop Scheduling Problem. *Procedia Computer Science, 112*, 296–305. doi:10.1016/j.procs.2017.08.267

Chen, X., Zhang, B., & Gao, D. (2019). An Improved Bat Algorithm for Job Shop Scheduling Problem. *2019 IEEE International Conference on Mechatronics and Automation (ICMA)* (pp. 439-443). Tianjin, China: IEEE. doi:10.1109/icma.2019.8816578

Colorni, A., Dorigo, M., & Maniezzo, V. (1991). Distributed Optimization by Ant Colonies. In F. Varela, & P. Bourgine (Ed.), *Proceedings of the European Conference on Artificial Life, ECAL'91* (pp. 134-142.). Paris: Elsevier Publishing.

Colorni, A., Dorigo, M., M. V., & Trubian, M. (1995). Ant system for Job-shop scheduling. *JORBEL - Belgian Journal of Operations Research, Statistics, and Computer Science, 34*(1), 39-53.

Cruz-Chávez, M. A. (2014). Neighbourhood generation mechanism applied in simulated annealing to job shop scheduling problems. *International Journal of Systems Science, 46*(15), 2673–2685. doi:10.1080/00207721.2013.876679

Cruz-Chavez, M. A., Cruz, M. H., Zavala-Díaz, J. C., Hernandez, J. A., Rodriguez-León, A., Prince, J. C., . . . Salinas, O. H. (2019). Hybrid Micro Genetic Multi-Population Algorithm with Collective Communication for the Job Shop Scheduling Problem. *IEEE Access, 7*, 82358–82376. doi:10.1109/access.2019.2924218

Cruz-Chávez, M. A., Peralta-Abarca, J. d., & Cruz-Rosales, M. H. (2019). Cooperative Threads with Effective-Address in Simulated Annealing Algorithm to Job Shop Scheduling Problems. *Applied Sciences, 9*(16), 3360. doi:doi:10.3390/app9163360

Davis, L. (1985). Job Shop Scheduling with Genetic Algorithms. In J. J. Grefensttete (Ed.), *Proceedings of the first international conference on genetic algorithms and their applications* (pp. 136–140). Psychology Press.

Dell'Amico, M., & Trubian, M. (1993). Applying tabu search to the job-shop scheduling problem. *Annals of Operations Research, 41*(3), 231–252. doi:10.1007/bf02023076

Della Croce, F., Tadei, R., & Volta, G. (1995). A genetic algorithm for the job shop problem. *Computers & Operations Research, 22*(1), 15–24. doi:10.1016/0305-0548(93)e0015-l

Falkenauer, E., & Bouffouix, S. (1991). A genetic algorithm for job shop. Proceedings. *Proceedings. 1991 IEEE International Conference on Robotics and Automation. 1*, pp. 824-829. Sacramento, CA: IEEE. doi:doi:10.1109/robot.1991.131689

Festa, P. (2014). A brief introduction to exact, approximation, and heuristic algorithms for solving hard combinatorial optimization problems. *2014 16th International*

Conference on Transparent Optical Networks (ICTON). Graz, Austria: IEEE. doi:doi:10.1109/icton.2014.6876285

Fisher, H., & Thompson, G. L. (1963). Probabilistic learning combinations of local job-shop scheduling rules. In J. F. Muth, & G. L. Thompson (Ed.), *Industrial Scheduling* (pp. 225-251). Englewood Cliffs, New Jersey: Prentice Hall.

Garey, M. R., Johnson, D. S., & Sethi, R. (1976). The Complexity of Flowshop and Jobshop Scheduling. *Mathematics of Operations Research, 1*(2), 117-129. doi:10.1287/moor.1.2.117

Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research, 13*(5), 533–549. doi:10.1016/0305-0548(86)90048-1

Graves, S. C. (1981). A Review of Production Scheduling. *Operations Research, 29*(4), 646–675. doi:10.1287/opre.29.4.646

Hart, E., & Sim, K. (2016). A hyper-heuristic ensemble method for static job-shop scheduling. *Evolutionary computation, 24*(4), 609-635. doi:doi.org/10.1162/EVCO_a_00183

Hefetz, N., & Adiri, I. (1982). An Efficient Optimal Algorithm for the Two-Machines Unit-Time Jobshop Schedule-Length Problem. *Mathematics of Operations Research, 7*(3), 354–360. doi:10.1287/moor.7.3.354

Holland, J. H. (1992). *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence.* MIT press.

Jiang, T., & Zhang, C. (2018). Application of Grey Wolf Optimization for Solving Combinatorial Problems: Job Shop and Flexible Job Shop Scheduling Cases. *IEEE Access, 6*, 26231–26240. doi:10.1109/access.2018.2833552

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

Kato, E. R., de Aguiar, G. D., & Tsunaki, R. H. (2018). A New Approach to Solve the Flexible Job Shop Problem Based on an Hybrid Particle Swarm Optimization and Random-Restart Hill Climbing. *Computers & Industrial Engineering, 125*, 178-189. doi:10.1016/j.cie.2018.08.022

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

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

Kolonko, M. (1999). Some new results on simulated annealing applied to the job shop scheduling problem. *European Journal of Operational Research, 113*(1), 123–136. doi:10.1016/s0377-2217(97)00420-7

Lawrence, S. (1984). Resource constrained project scheduling: An experimental investigation of heuristic scheduling techniques (Supplement). *Graduate School of Industrial Administration.* Pittsburgh, Pennsylvania.: Carnegie-Mellon University.

Lin, L., Hao, X. C., Gen, M., & Jo, J. B. (2011). Network modeling and evolutionary optimization for scheduling in manufacturing. *Journal of Intelligent Manufacturing, 23*(6), 2237–2253. doi:10.1007/s10845-011-0569-6

Lin, T. L., Horng, S. J., Kao, T. W., Chen, Y. H., Run, R. S., Chen, R. J., & Kuo, I. H. (2010). An efficient job-shop scheduling algorithm based on particle swarm optimization. *Expert Systems with Applications, 37*(3), 2629-2636. doi:doi:10.1016/j.eswa.200

Liu, S. C., Chen, Z. G., Zhan, Z. H., Jeon, S. W., Kwong, S., & Zhang, J. (2021). Many-Objective Job-Shop Scheduling: A Multiple Populations for Multiple Objectives-Based Genetic Algorithm Approach. *IEEE Transactions on Cybernetics*, 1-15. doi:10.1109/TCYB.2021.3102642

Lomnicki, Z. A. (1965). A "Branch-and-Bound" Algorithm for the Exact Solution of the Three-Machine Scheduling Problem. *Journal of the Operational Research Society, 16*(1), 89–100. doi:10.1057/jors.1965.7

Manne, A. S. (1960). On the Job-Shop Scheduling Problem. *Operations Research, 8*(2), 219–223. doi:10.1287/opre.8.2.219

Masmoudi, O., Delorme, X., & Gianessi, P. (2019). Job-shop scheduling problem with energy consideration. *International Journal of Production Economics, 216*, 12-22. doi:10.1016/j.ijpe.2019.03.021

McMahon, G., & Florian, M. (1975). On Scheduling with Ready Times and Due Dates to Minimize Maximum Lateness. *Operations Research, 23*(3), 475–482. doi:10.1287/opre.23.3.475

Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., & Teller, E. (1953). Equation of State Calculations by Fast Computing Machines. *The Journal of Chemical Physics, 21*(6), 1087–1092. doi:10.1063/1.1699114

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

Nakano, R., & Yamada, T. (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.

Pongchairerks, P. (2019). A Two-Level Metaheuristic Algorithm for the Job-Shop Scheduling Problem. *Complexity, 2019*, 1–11. doi:10.1155/2019/8683472

Potts, C. N., & Van Wassenhove, L. N. (1985). A Branch and Bound Algorithm for the Total Weighted Tardiness Problem. *Operations Research, 33*(2), 363–377. doi:10.1287/opre.33.2.363

Rabadi, G. (Ed.). (2016). *Heuristics, Metaheuristics and Approximate Methods in Planning and Scheduling* (Vol. 236). Springer, Cham. doi:doi.org/10.1007/978-3-319-26024-2

Satake, T., Morikawa, K., Takahashi, K., & Nakamura, N. (1999). Simulated annealing approach for minimizing the makespan of the general job-shop. *International Journal of Production Economics, 60-61*, 515–522. doi:10.1016/s0925-5273(98)00171-6

Semlali, S. C., Riffi, M. E., & Chebihi, F. (2019). Memetic chicken swarm algorithm for job shop scheduling problem. *International Journal of Electrical and Computational Engineering (IJECE), 9*(3), 2075-2082. doi:10.11591/ijece.v9i3.pp2075-2082

Siarry, P. (2016). Simulated Annealing. In P. Siarry (Ed.), *Metaheuristics.* Springer. doi:doi.org/10.1007/978-3-319-45403-0_2

Sörensen, K., Sevaux, M., & Glover, F. (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

Storn, R., & Price, K. (1997). Differential Evolution – A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces. *Journal of Global Optimization, 11*, 341–359. doi:doi.org/10.1023/A:1008202821328

Sundar, S., Suganthan, P. N., Jin, C. T., Xiang, C. T., & Soon, C. C. (2015). A hybrid artificial bee colony algorithm for the job-shop scheduling problem with no-wait constraint. *Soft Computing, 21*(5), 1193-1202. doi:10.1007/s00500-015-1852-9

Syarif, A., Pamungkas, A., Kumar, R., & Gen, M. (2021). Performance Evaluation of Various Heuristic Algorithms to Solve Job Shop Scheduling Problem (JSSP). *International Journal of Intelligent Engineering & System, 14*(2), 334-343. doi:doi.org/10.22266/ijies2021.0430.30

Taillard, É. D. (1994). Parallel Taboo Search Techniques for the Job Shop Scheduling Problem. *ORSA Journal on Computing, 6*(2), 108–117. doi:10.1287/ijoc.6.2.108

Udaiyakumar, K. C., & Chandrasekaran, M. (2014). Application of Firefly Algorithm in Job Shop Scheduling Problem for Minimization of Makespan. *Procedia Engineering, 97*, 1798-1807. doi:10.1016/j.proeng.2014.12.333

Udomsakdigool, A., & Kachitvichyanukul, V. (2008). Multiple colony ant algorithm for job-shop scheduling problem. *International Journal of Production Research, 46*(15), 4155–4175. doi:10.1080/00207540600990432

Van Laarhoven, P. J., & Aarts, E. H. (1987). Simulated Annealing. In *Simulated Annealing: Theory and Applications* (Vol. 37, pp. 7–15). Springer, Dordrecht. doi:doi:10.1007/978-94-015-7744-1_2

Van Laarhoven, P. J., Aarts, E. H., & Lenstra, J. K. (1992). Job Shop Scheduling by Simulated Annealing. *Operations Research, 40*(1), 113–125. doi:10.1287/opre.40.1.113

Verderame, P. M., & Floudas, C. A. (2008). Integrated Operational Planning and Medium-Term Scheduling for Large-Scale Industrial Batch Plants. *Industrial & Engineering Chemistry Research, 47*(14), 4845–4860. doi:10.1021/ie8001414

Wagner, H. M. (1959). An integer linear-programming model for machine scheduling. *Naval Research Logistics Quarterly, 6*(2), 131–140. doi:10.1002/nav.3800060205

Wang, T., Payberah, A. H., & Vlassov, V. (2020). CONVJSSP: Convolutional Learning for Job-Shop Scheduling Problems. *2020 19th IEEE International Conference on Machine Learning and Applications (ICMLA).* doi:doi:10.1109/icmla51294.2020.00229

Yamada, T., & Nakano, R. (1992). A genetic algorithm applicable to large-scale job-shop instances. In B. M. R. Manner (Ed.), *Parallel instance solving from nature 2* (pp. 281-290). Amsterdam, North-Holland.

Yamada, T., & Nakano, R. (1996). Job-Shop Scheduling by Simulated Annealing Combined with Deterministic Local Search. In I. H. Osman, & J. P. Kelly (Eds.), *Meta-Heuristics* (pp. 237–248). Boston, MA: Springer. doi:10.1007/978-1-4613-1361-8_15

Yang, X. S. (2008). *Nature-inspired metaheuristic algorithms* (2da ed.). Luniver Press.

Ying, K. C., & Lin, S. W. (2020). Solving no-wait job-shop scheduling problems using a multi-start simulated annealing with bi-directional shift timetabling algorithm. *Computers & Industrial Engineering, 146*, 106615. doi:10.1016/j.cie.2020.106615

Yu, S. (2019). Differential Evolution Quantum Particle Swarm Optimization for Solving Job-shop Scheduling Problem. *2019 Chinese Control And Decision Conference (CCDC)*, (pp. 559-564). doi:10.1109/ccdc.2019.8833361

Zhang, J., Ding, G., Zou, Y., Qin, S., & Fu, J. (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