EDUCACIÓN
SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLÓGICO NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE CIUDAD MADERO

DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

MAESTRÍA EN CIENCIAS DE LA COMPUTACIÓN

"POR MI PATRIA Y POR MI BIEN"

TESIS

ALGORITMO HÍBRIDO PARA EL PROBLEMA DINÁMICO DE CALENDARIZACIÓN DE TAREAS (DYNAMIC JOB SHOP SCHEDULING PROBLEM)

Que para obtener el Grado de

Maestro en Ciencias de la Computación

Presenta

Ing. Arsenio de Jesús Lizardi Durán

G13071650

N. CVU de CONACyT: 1007962

Director de Tesis

Dra. Guadalupe Castilla Valdez

No. CVU de CONACyT: 281385

Co-director de Tesis

Dr. Juan Frausto Solís

Cd. Madero, Tamaulipas Diciembre 2021

Cd. Madero, Tam. 10 de diciembre de 2021

OFICIO No. : U.181/21
ASUNTO: AUTORIZACIÓN DE IMPRESIÓN DE TESIS

C. ARSENIO DE JESÚS LIZARDÍ DURÁN
No. DE CONTROL G13071650
P R E S E N T E

Me es grato comunicarle que después de la revisión realizada por el Jurado designado para su Examen de Grado de Maestría en Ciencias de la Computación, se acordó autorizar la impresión de su tesis titulada:

"ALGORITMO HÍBRIDO PARA EL PROBLEMA DINÁMICO DE CALENDARIZACIÓN DE TAREAS (DYNAMIC JOB SHOP SCHEDULING PROBLEM)"

El Jurado está integrado por los siguientes catedráticos:

PRESIDENTE:	DR.	JUAN JAVIER GONZÁLEZ BARBOSA
SECRETARIO:	DR.	JUAN FRAUSTO SOLÍS
VOCAL:	DRA.	GUADALUPE CASTILLA VALDEZ
SUPLENTE:	DR.	RODOLFO ABRAHAM PAZOS RANGEL
DIRECTOR DE TESIS:	DRA.	GUADALUPE CASTILLA VALDEZ
CO-DIRECTOR:	DR.	JUAN FRAUSTO SOLÍS

Es muy satisfactorio para la División de Estudios de Posgrado e Investigación compartir con usted el logro de esta meta. Espero que continúe con éxito su desarrollo profesional y dedique su experiencia e inteligencia en beneficio de México.

A T E N T A M E N T E
Excelencia en Educación Tecnológica®
"Por mi patria y por mi bien"®

MARCO ANTONIO CORONEL GARCÍA
JEFE DE LA DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

EDUCACIÓN | TECNOLÓGICO NACIONAL DE MÉXICO
INSTITUTO TECNOLÓGICO DE CIUDAD MADERO
DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

c.c.p.- Archivo
MACG 'jar'

DECLARACION DE ORIGINALIDAD, PROPIEDAD INTELECTUAL, CESION DE DERECHOS Y/O CONFIDENCIALIDAD.

Declaro y prometo que este documento de tesis es producto de mi trabajo original y que no infringe los derechos de terceros, tales como derechos de publicación, derechos de autor, patentes y similares.

Además, declaro que en las citas textuales que he incluido y en los resúmenes que he realizado de publicaciones ajenas, indico explícitamente los datos de los autores y publicaciones. En caso de infracción de los derechos de terceros derivados de este documento de tesis, acepto la responsabilidad de la infracción y relevo de esta a mi director y codirector de tesis, así como al Tecnológico Nacional de México/Instituto Tecnológico de Ciudad Madero y sus autoridades.

Atentamente

Ing. Arsenio de Jesús Lizardi Durán

Contenido

Índice de Figuras

Índice de Tablas

Abstract

The Job Shop Scheduling Problem (JSSP) is an NP-hard problem where the application of exact methods has limitations. Numerous methods based on metaheuristics have been developed to solve the JSSP problem, but there are few works dedicated to solving the Dynamic Job Shop Scheduling Problem (DJSSP), which models the unexpected events that occur in real-world manufacturing systems. In this thesis, we address DJSSP with three types of dynamic events: machine breakdowns, new job arrivals, and changes in the processing time; with a hybrid algorithm based on simulated annealing and a local search strategy. This algorithm is evaluated using a set of benchmark instances and experimentation is performed to compare the performance of the developed algorithm with those reported in the literature. The numerical results obtained indicate that the proposed method produces better solutions for the benchmark instances, compared with the methods reported on the literature.

Capítulo 1 Introducción

1.1 Introducción

La calendarización de tareas es un problema muy común en el mundo real, este problema se encuentra presente en una gran variedad de entornos de trabajo, como líneas de producción en fábricas, planeación de vuelos y servicios en hospitales entre otros.

El problema de calendarización de tareas (JSSP, por sus siglas en inglés) es uno de los problemas NP-duros (Brucker et al., 2007) más difíciles, por lo cual la aplicación de métodos exactos está limitada a escalas pequeñas. Existen diversos métodos basados en metaheurísticas para resolver el problema JSSP en su modelo estático, no obstante, en el mundo real la calendarización de tareas rara vez existe de manera estática. Esto puede volver ineficiente la calendarización inicial, y se requiere aplicar estrategias de calendarización dinámica (también conocida como recalendarización) para actualizar la calendarización original basada en cambios.

Con este trabajo se pretende el uso de un algoritmo metaheurístico híbrido para la resolución del Problema Dinámico de Calendarización de Tareas (DJSSP, por la denominación en inglés del *Dynamic Job Shop Scheduling Problem*).

1.2 Planteamiento del problema

***Dynamic Job Shop Scheduling* (DJSSP)**

JSSP consiste en un conjunto de metas concurrentes y en conflicto para ser satisfechas usando un número finito de máquinas. Cada trabajo tiene un orden de procesamiento en las máquinas que especifica las restricciones de precedencia. La importancia del trabajo *j* relativa a los otros trabajos en el sistema es denotada por el peso w_j. La principal restricción de trabajos y máquinas es que una máquina solo puede trabajar con una operación a la vez y adelantar cualquier operación en cualquier máquina está prohibido. Además, se asume que los tiempos de procesamiento son conocidos cuando los trabajos llegan al taller y las máquinas siempre están disponibles cuando no están siendo usadas por otro trabajo.

En el JSSP dinámico, los trabajos llegan en distintos tiempos. JSSP dinámico puede ser subclasificado como determinístico o estocástico basado en la forma de la especificación de los tiempos de liberación de trabajo. DJSSP determinístico supone que los tiempos de liberación son conocidos con anticipación. En DJSSP estocástico los tiempos de liberación son variables aleatorias descritas por una distribución de probabilidad conocida. (Lin et al., 1997)

1.3 Objetivos

Objetivo General

Desarrollar un algoritmo híbrido para el Problema Dinámico de Calendarización de Tareas que tenga un desempeño similar a los algoritmos del estado del arte.

Objetivos específicos

- Implementar el algoritmo de recocido simulado básico para el problema JSSP.
- Analizar y seleccionar eventos dinámicos, indicadores de evaluación y estrategias de calendarización dinámica para el problema JSSP.

- Implementar y evaluar el algoritmo de recocido simulado para el problema JSSP Dinámico (DJSSP).
- Analizar y seleccionar estrategias de mejora para el algoritmo de recocido simulado para el problema DJSSP.
- Desarrollar la experimentación para evaluar el desempeño de los algoritmos implementados.
- Desarrollar la evaluación estadística de los algoritmos implementados.

1.4 Justificación y beneficios

Justificación

Existe un escaso número de trabajos que aborden el problema e incluyan comparación con resultados del estado del arte.

Beneficios

El problema de asignación de tareas dinámico está presente en diversas áreas, y en muchos casos se requieren soluciones de manera rápida.

1.5 Alcances y limitaciones

Alcances

El trabajo aborda el problema JSSP dinámico, utilizando un algoritmo metaheurístico híbrido.

Limitaciones

La versión del problema que se aborda es la versión mono-objetivo. Además, solo se considerarán tres tipos de eventos dinámicos.

Capítulo 2 Fundamentación teórica

2.1 Optimización

La forma estándar de un problema de optimización continua es (Boyd & Vandenberghe, 2004)

$$Minimizar\ f(x)$$

$$Sujeto\ a$$

$$g_i(x) \leq 0, \qquad i = 1, \dots, m$$

$$h_j(x) = 0, \qquad j = 1, \dots, p$$

donde

f: es la función objetivo a minimizar sobre el vector x de n variables

$g_i(x) \leq 0$ se denominan restricciones de desigualdad

$h_j(x) = 0$ se denominan restricciones de igualdad y

$m \geq 0$ y $p \geq 0$

Si $m = p = 0$ el problema es sin restricciones. Por convención, la forma estándar define un problema de minimización. Un problema de maximización puede tratarse negando la función objetivo.

2.2 Problemas de optimización

Un problema de optimización involucra encontrar la mejor solución de un conjunto exponencialmente grande de soluciones. Revisar cada solución linealmente tomaría una gran cantidad de tiempo debido a la cantidad de soluciones. Existen métodos para solucionar estos problemas en tiempo polinomial, tales como programación linear, algoritmos *greedy* o programación dinámica. En otros casos, es necesario implementar soluciones aproximadas con el fin de obtener soluciones en los tiempos requeridos por el entorno de aplicación real.

Un problema de optimización se especifica definiendo las instancias, las soluciones, y los costos que conlleva dicha solución. Las instancias son las posibles entradas del problema y cada instancia tiene un conjunto de soluciones potenciales cuyo tamaño aumenta en forma exponencial con el tamaño de la instancia. Una solución es válida si cumple ciertos criterios determinados por la instancia. Cada solución tiene un costo o valor asociado, el cual se busca maximizar o minimizar dependiendo del problema.

Algunos ejemplos de problemas de optimización son el problema de *bin packing*, el problema de la mochila, y el problema del agente viajero.

El problema de *bin packing* se define como: Dado un conjunto finito I de objetos, un tamaño $s(i)$ en $Z+$ para cada i en I, una capacidad de caja entero positivo B, y un entero positivo K. ¿Existe alguna partición de I en conjuntos disjuntos I_1, … I_k tales que la suma de los tamaños de los objetos en cada I_j sea B o menos? (Garey & Johnson, 1990)

El problema de la mochila puede definirse como: Dado un conjunto N de objetos que está formado por n objetos, cada uno con valor p_j y peso w_j, y una capacidad c de la mochila. (Usualmente, todos los valores son enteros positivos). El objetivo es seleccionar un subconjunto de N tal que el valor total de los objetos seleccionados sea maximizado y el peso total no exceda la capacidad c de la mochila. (Kellerer et al., 2004)

El problema del agente viajero puede definirse como: Se tiene un vendedor que debe visitar un conjunto de N ciudades, iniciando y terminando su recorrido en su ciudad de origen, con la condición de visitar una sola vez cada ciudad con excepción del origen. El problema consiste en encontrar la ruta más corta a seguir.

2.3 Clasificación de problemas

Un problema fácil o tratable es un problema cuya solución puede ser obtenida por un algoritmo con un tiempo de solución como una función polinomial de tamaño del problema n. Los algoritmos con tiempo polinomial son considerados fáciles. Un problema es llamado problema P o problema en tiempo polinomial si el número de pasos que se necesitan para encontrar la solución está dado por un polinomio en n y tiene al menos un algoritmo para resolverlo.

Por otra parte, un problema duro o intratable requiere tiempo de solución que es una función exponencial de n, y por lo tanto se les considera ineficientes. Un problema es llamado polinomial no determinístico (NP) si su solución puede ser adivinada y evaluada en tiempo polinomial, y no hay una regla conocida para adivinarla (por ende, es no determinístico). Consecuentemente, las soluciones adivinadas no garantizan ser óptimas o cercanas al óptimo. De hecho, no existe un algoritmo conocido hasta ahora para resolver problemas NP-duros en tiempos de orden polinomial, y solo soluciones aproximadas o heurísticas son posibles. Por tanto, los métodos heurísticos y metaheurísticos son muy prometedores para obtener soluciones aproximadas; por lo general subóptimas, para esta clase de problemas.

Un problema es llamado NP-Completo si es un problema NP y todos los problemas en NP son reducibles a él por medio de ciertos algoritmos de reducción. El algoritmo de reducción tiene un tiempo polinomial. (Yang, 2008)

2.4 *Job Shop Scheduling Problem* (JSSP)

JSSP consiste en un conjunto de metas concurrentes y en conflicto para ser satisfechas usando un número finito de máquinas. Cada trabajo tiene un orden de procesamiento en las máquinas que especifica las restricciones de precedencia. La importancia del trabajo j relativa a los otros trabajos en el sistema es denotada por el peso w_j. La principal restricción de trabajos y máquinas es que una máquina solo puede trabajar con una operación a la vez y adelantar cualquier operación en cualquier máquina está prohibido. Además, se asume que los tiempos de procesamiento son conocidos cuando los trabajos llegan al taller y las máquinas siempre están disponibles cuando no están siendo usadas por otro trabajo. (Lin et al., 1997) El JSSP es clasificado como un problema NP-duro, ya que el problema del agente viajero también lo es, y este puede representarse como un caso especial del JSSP con un solo trabajo, donde las ciudades son las máquinas y el agente es el trabajo.

Algunas clases de JSSP incluyen, JSSP determinístico, JSSP flexible, JSSP estático, JSSP dinámico, entre otros. (Abdolrazzagh-Nezhad & Abdullah, 2017)

Existen diferentes métodos para encontrar soluciones a instancias del problema. Los principales avances han sido por medio de algoritmos metaheurísticos.

2.5 Algoritmos metaheurísticos

La mayoría de los problemas de optimización son clasificados como problemas NP-duros. Por lo tanto, solo instancias de tamaño pequeño pueden ser resueltas usando métodos exactos. En su lugar, se consideró la posibilidad de usar soluciones alternativas (métodos de aproximación) que pueden encontrar una solución suficientemente buena en un tiempo razonable, estos métodos se pueden clasificar como heurísticos y metaheurísticos. La diferencia más significativa entre ellos es que los heurísticos son más dependientes del problema que los metaheurísticos; los heurísticos pueden ser aplicados eficientemente a un problema en específico, pero se vuelven insuficientes para otros problemas. Por otro lado, los metaheurísticos son sistemas de algoritmos genéricos que pueden ser aplicados a casi todos los problemas de optimización. (Abdel-Basset et al., 2018)

2.5.1 Búsqueda Local

La búsqueda local es un método ampliamente usado para resolver problemas de optimización.

Un enfoque de búsqueda local empieza generalmente en una solución elegida aleatoriamente, y va buscando soluciones potencialmente mejores en su vecindario, hasta que termina en una solución óptima local, posiblemente cercana al óptimo. (Michiels et al., 2018)

2.5.2 Búsqueda Tabú (TS)

La búsqueda Tabú consiste en marcar como “Tabú” los espacios de búsqueda ya visitados para promover la diversificación; con el fin de explorar el espacio de búsqueda y evitar quedar atrapado en un óptimo local, la búsqueda Tabú aplica un método de búsqueda local intensiva. Hay dos características principales en la búsqueda Tabú: memoria adaptiva y exploración responsiva. La primera (llamada lista Tabú), guarda la historia de las acciones realizadas en el tiempo de búsqueda para esquivar el cerco entre ciclos. La segunda utiliza a la primera para que el proceso de búsqueda se enfoque en las buenas regiones y soluciones para más intensificación, y además para explorar las nuevas regiones con el fin de ampliar la exploración. (Abdel-Basset et al., 2018)

2.5.3 Recocido Simulado (SA)

El Recocido simulado es una metaheurística para aproximar el óptimo global en un espacio de búsqueda grande para problemas de optimización. (Van Laarhoven & Aarts, 1987)

En este método se genera aleatoriamente una solución, si la solución es mejor a la actualmente encontrada, se acepta, en caso contrario, se utiliza una probabilidad de aceptación de Boltzmann *p,* la cual va disminuyendo de acuerdo con el número de iteraciones.

El término recocido simulado proviene del proceso con el mismo nombre que existe en la metalurgia. Consiste en el enfriamiento progresivo de un metal de tal manera que sus moléculas van adoptando poco a poco una configuración de mínima energía. A medida que la temperatura disminuye, se va ralentizando el movimiento de las moléculas y éstas tienden a adoptar paulatinamente las configuraciones de menor energía. (Kirkpatrick et al., 1983)

El algoritmo empieza con una solución inicial aleatoria x^C y una temperatura alta T. Entonces, otra solución x^n se genera aleatoriamente como vecino de la solución inicial y la diferencia en el valor de la función Δ se calcula como:

$$\Delta = f(x^n) - f(x^C) \quad (1)$$

Si delta es menor, automáticamente x^n se vuelve la solución actual a partir de la cual la búsqueda continúa. En caso contrario, se acepta con una probabilidad:

$$P(\Delta, T) = e^{(-\Delta/T)} \quad (2)$$

Finalmente, se disminuye T. En la figura 2.1 se muestra el algoritmo descrito.

```
Generar solucion x^C;
//Rmax es el número de iteraciones
para r = 1 hasta Rmax hacer
     mientras (criterio de terminación insatisfecho) hacer
          calcular x^n ∈ N(x^C);
          //crea una nueva solución x^n perturbando x^C
          calcular Δ;
          si (Δ < 0) entonces
               x^C = x^n
          sino
               x^C = x^n con probabilidad P
          Fin mientras
          reducir T;
Fin para
```

Figura 2.1 Algoritmo de la metaheurística de Recocido Simulado

2.5.4 Algoritmo genético (GA)

El algoritmo genético emula el proceso de evolución biológico por cromosomas definiendo los siguientes operadores: selección, cruza, y muta. Los cromosomas se manejan como soluciones candidatas para un problema dado y son evaluadas de acuerdo con su fitness. La selección de padres para la cruza es un proceso para generar nuevas soluciones.

Durante la cruza, algunas partes de dos cromosomas seleccionados son intercambiadas. Las partes de los cromosomas pueden ser cambiadas de diferentes formas. En la muta, algunas partes de los cromosomas son cambiadas aleatoriamente para escapar de óptimos locales. Pero los mejores cromosomas pueden ser perdidos al crear nuevos cromosomas, para evitar esto, se utiliza elitismo para copiar el mejor o los mejores cromosomas a la nueva población. (Abdel-Basset et al., 2018)

Capítulo 3 Estado del arte

Se realizó un análisis de diferentes investigaciones que han propuesto métodos diversos para resolver este problema, y se compararon las características relevantes y las diferencias encontradas entre estos.

Kundakcı & Kulak (2016) introdujeron metodologías genéticas híbridas para minimizar el *makespan* en este tipo de problemas. Generaron varios problemas *benchmark* con número de trabajos, de máquinas, y diferentes eventos dinámicos, y realizaron experimentación numérica para evaluar el desempeño de las metodologías propuestas. Los resultados indican que los métodos propuestos por ellos produjeron soluciones superiores a problemas *benchmark* que las reportadas en la literatura.

Baykasoğlu & Karaslan (2017) propusieron un método utilizando *Greedy Randomized Adaptive Search Procedure* (GRASP) que toma en cuenta el tiempo de entrega de las órdenes y tiempos de preparación dependientes de la secuencia. Además, se integraron la decisión de orden de aceptación/rechazo y el mecanismo de revisión de salida de orden con la decisión de calendarización para satisfacer los requerimientos de tiempo límite mientras se ejecutan ajustes de capacidad. Se empleó una lógica basada en programación de metas que fue usada para evaluar cuatro objetivos: media de *tardiness*, inestabilidad de calendarización, *makespan*, y tiempo de flujo medio. Adicionalmente a la estrategia de recalendarización dirigida por eventos, se desarrolló una estrategia de recalendarización periódica y ambas estrategias se compararon para cuatro problemas diferentes. Los resultados obtenidos demostraron que el método propuesto es un enfoque factible para problemas de recalendarización bajo ambientes dinámicos.

Por otro lado, Wang et al. (2019) propusieron un algoritmo de optimización de enjambre para DJSSP con llegadas de trabajo aleatorias. Se consideró el asunto de como recalendarizar los trabajos recién llegados aleatoriamente para buscar tanto desempeño como estabilidad. Se usó un modelo de programación lineal entero mixto para minimizar tres objetivos. Luego, cuatro estrategias de emparejamiento fueron modificadas para determinar el horizonte de recalendarización. Después, un algoritmo de optimización de enjambre (PSO) con mejoras se propuso para resolver el DJSSP. Las estrategias de mejora consistieron en un esquema de

decodificación modificado, un enfoque de inicialización de la población diseñando un nuevo mecanismo de transformación, y un nuevo modelo de movimiento de partículas que introduce cambios de posición y peso de inercia aleatorio. Los experimentos mostraron que las estrategias modificadas son estadísticamente mejores que las estrategias con las que se compararon.

Por último, Mohan et al. (2019) presentan una revisión de técnicas de calendarización para DJSSP. Se abordaron los conceptos de DJSSP, eventos dinámicos, indicador de evaluación, estrategia de calendarización dinámica, métodos de calendarización dinámicos, y sistema de calendarización. Los métodos de calendarización se dividieron en dos clases: métodos precisos y métodos aproximados.

En la tabla 3.1 se muestra una comparación entre el contenido, resultados, y aporte de cada trabajo revisado anteriormente en el estado del arte, y en la tabla 3.2 se muestra la comparación entre la experimentación realizada en cada uno.

Tabla 3.1 Cuadro comparativo del estado del arte

Trabajo y autor	**Contenido**	**Resultados**	**Aporte**
Kundakcı & Kulak, (2016). "Hybrid genetic algorithms for minimizing makespan in dynamic job shop scheduling problema".	Un algoritmo híbrido genético para minimizar el *makespan* en DJSSP.	El artículo presenta una plataforma de referencia comprehensiva para la comparación de modelos y algoritmos con aplicaciones para DJSSP.	Las instancias generadas por los autores sirven como punto de referencia.
Baykasoğlu, A., & Karaslan, F. S. (2017). "Solving comprehensive dynamic job shop scheduling problem by using a GRASP-based approach".	Un método utilizando GRASP que toma en cuenta el tiempo de entrega de las órdenes y tiempos de preparación dependientes de la secuencia.	La experimentación demostró que el algoritmo propuesto es un enfoque competitivo para resolver el DJSSP.	Se puede comparar el desempeño con el algoritmo propuesto.
Wang, Z., Zhang, J., & Yang, S. (2019). "An improved particle swarm optimization algorithm for dynamic job shop scheduling problems with random job arrivals".	Un algoritmo de optimización de enjambre para DJSSP con llegadas de trabajo aleatorias.	Los resultados de la experimentación demostraron que el algoritmo es efectivo para resolver DJSSP con llegadas de nuevos trabajos aleatorias.	Se puede comparar el desempeño con el algoritmo propuesto.
Mohan, J., Lanka, K., & Rao, A. N. (2019). "A Review of Dynamic Job Shop Scheduling Techniques".	Una revisión de técnicas de calendarización para DJSSP.	Se presentaron varios métodos de calendarización dinámica, y se definieron algunos aspectos que podrían expandir la investigación.	Se puede implementar uno de los aspectos que podrían expandir la investigación.

Tabla 3.2 Comparación de la experimentación de los trabajos del estado del arte

Trabajo y autor	**Algoritmo/ método**	**Eventos dinámicos**	**Objetivo(s)**	**Instancias *benchmark***
Kundakcı & Kulak, (2016).	Algoritmo genético híbrido	Descomposición de máquinas, llegada de nuevos trabajos, cambio de tiempo de procesamiento	*Makespan*	Disponibles
Baykasoğlu, A., & Karaslan, F. S. (2017).	GRASP	Llegadas de nuevos trabajos, descomposición de máquinas, llegada de ordenes urgentes, cambios en las fechas de entrega y cancelación de ordenes	*Mean tardiness, Schedule unstability, makespan, mean Flow time.*	No disponibles
Wang, Z., Zhang, J., & Yang, S. (2019).	*Particle swarm*	Llegadas de trabajo aleatorias	*Makespan*	No disponibles
Mohan, J., Lanka, K., & Rao, A. N. (2019).	La investigación no aborda algoritmo, es estrictamente teórica.	Eventos relacionados con las partes, con las máquinas, con los procesos, y otros eventos	Indicadores temporales, indicadores económicos, e indicadores de sistema.	La investigación no aborda algoritmo, es estrictamente teórica.

Capítulo 4 Descripción del problema

4.1 Definición formal del problema

En el JSSP dinámico, los trabajos llegan en distintos tiempos. JSSP dinámico puede ser subclasificado como determinístico o estocástico basado en la forma de la especificación de los tiempos de liberación de trabajo. DJSSP determinístico supone que los tiempos de liberación son conocidos con anticipación. En DJSSP estocástico los tiempos de liberación son variables aleatorias descritas por una distribución de probabilidad conocida. (Lin et al., 1997)

El DJSSP es un problema de optimización combinatoria y considera que n trabajos son calendarizados al principio de la calendarización, y un conjunto n' de nuevos trabajos llegan después del inicio para ser procesados en m máquinas. El problema está sujeto a lo siguiente:

- Cada máquina solo puede realizar una operación de cualquier trabajo a la vez.
- Una operación de un trabajo solo puede ser realizada en una sola máquina a la vez
- Todas las máquinas están disponibles en el tiempo 0.
- Una vez que una operación ha sido procesada en una máquina, no debe ser interrumpida excepto por la descomposición de una máquina. Si una operación es interrumpida por la descomposición de una máquina, el tiempo de procesamiento restante es igual al tiempo total de procesamiento menos el tiempo de procesamiento completado.
- Una operación de un trabajo no puede ser realizada hasta que sus operaciones precedentes hayan sido completadas.
- No hay ruteo flexible para cada trabajo.
- El tiempo de procesamiento de operación y el número de máquinas operables se saben por adelantado. Pero habrá cambios en el tiempo de procesamiento de operación y las máquinas pueden descomponerse.

Las siguientes notaciones se usan en la formulación del problema:

$\boldsymbol{j}$:	Trabajos viejos ($\boldsymbol{j} = \boldsymbol{1}, \ldots, \boldsymbol{n}$)
$\boldsymbol{j'}$ **:**	Nuevos trabajos ($j = 1, 2, \ldots, n'$)
$\boldsymbol{i}$ **:**	Máquinas ($i = 1, \ldots, m$)
$\boldsymbol{A}$ **:**	Conjunto de todas las restricciones de ruteo $(i, j) \rightarrow (h, j)$
$\boldsymbol{A'}$ **:**	Conjunto de todas las constantes de ruteo incluyendo trabajos nuevos $(i, j') -> (h, j')$
$\boldsymbol{P_{ij}}$ **:**	Tiempo de procesamiento de la operación (i, j)
$\boldsymbol{P'_{ij'}}$ **:**	Tiempo de procesamiento de la nueva operación (i, j')
$\boldsymbol{rp}$ **:**	Tiempo de inicio del periodo de recalendarización
$\boldsymbol{tm_i}$ **:**	Tiempo que la máquina i va a estar inactiva al inicio del periodo de recalendarización
$\boldsymbol{Z_{ijk}}$:	1, si J_j precede a J_k en la máquina M_i ($= 0$ en caso contrario)
$\boldsymbol{Z_{ij'k}}$:	1, si $J'_{j'}$ precede a J_k en la máquina M_i ($= 0$ en caso contrario)
$\boldsymbol{t_{ij}}$:	1 si J_j será procesada en la máquina M_i después de recalendarizar ($= 0$ en caso contrario)
$\boldsymbol{t_{ij'}}$:	1 si $J'_{j'}$ será procesada en la máquina M_i después de recalendarizar ($= 0$ en caso contrario)
$\boldsymbol{C_{ij}}$ **:**	Tiempo de terminación del trabajo j en la máquina i
$\boldsymbol{C'_{ij'}}$ **:**	tiempo de terminación del trabajo j' en la máquina i
$\boldsymbol{Y_{ij}}$ **:**	tiempo de inicio de la operación (i, j)
$\boldsymbol{Y'_{ij'}}$ **:**	tiempo de inicio de la nueva operación (i, j')

Formulación del modelo:

$$Min\ C_{max} \tag{3}$$

Sujeto a:

$$C_{max} \geq C_{ij}\ (i = 1, \ldots, m\ j = 1, \ldots, n) \tag{4}$$

$$C_{max} \geq C'_{ij'}\ (i = 1, \ldots, m\ j' = 1, \ldots, n') \tag{5}$$

$$C_{ij} = Y_{ij} + P_{ij}\ (i = 1, \ldots, m\ j = 1, \ldots, n) \tag{6}$$

$$C'_{ij'} = Y'_{ij'} + P'_{ij'}\ (i = 1, \ldots, m\ j' = 1, \ldots, n') \tag{7}$$

$$Y_{hj} - Y_{ij} \geq P_{ij}\ para\ cada\ (i,j) -> (h,j) \in A \quad (8)$$

$$Y'_{hj'} - Y'_{ij'} \geq P'_{ij'} para\ cada\ (i,j') -> (h,j') \in A' \quad (9)$$

$$M_{Z_{ijk}} + (Y_{ij} - Y_{ik}) \geq P_{ik}\ (i = 1,\dots,m, 1 \leq j \leq k \leq n) \quad (10)$$

$$M_{Z_{ij'k}} + (Y'_{ij'} - Y'_{ik}) \geq P'_{ik}\ (i = 1,\dots,m, 1 \leq j' \leq k \leq n) \quad (11)$$

$$M(1 - Z_{ijk}) + (Yik - Y_{ij}) \geq P_{ij}\ (i = 1,\dots,m, 1 \leq j \leq k \leq n) \quad (12)$$

$$M(1 - Z_{ij'k}) + (Y'_{ik} - Y'_{ij'}) \geq P'_{ij'}\ (i = 1,\dots,m, 1 \leq j' \leq k \leq n) \quad (13)$$

$$Y_{ij} \geq (tm_i + rp)t_{ij}\ (i = 1,\dots,m\ j = 1,\dots,n) \quad (14)$$

$$Y'_{ij'} \geq (tm_i + rp)tij'\ (i = 1,\dots,m\ j' = 1,\dots,n) \quad (15)$$

$$Y_{ij} \geq 0\ para\ todo\ (i = 1,\dots,m\ j = 1,\dots,n) \quad (16)$$

$$Y'_{ij'} \geq 0\ para\ todo\ (i = 1,\dots,m\ j' = 1,\dots,n) \quad (17)$$

$$Z_{ijk} = 0\ o\ 1\ (i = 1,\dots,m\ j = 1,\dots,n) \quad (18)$$

$$Z_{ij'k} = 0\ o\ 1\ (i = 1,\dots,m\ j' = 1,\dots,n') \quad (19)$$

$$t_{ij} = 0\ o\ 1\ (i = 1,\dots,m\ j = 1,\dots,n) \quad (20)$$

$$t_{ij'} = 0\ o\ 1\ (i = 1,\dots,m\ j' = 1,\dots,n') \quad (21)$$

En esta formulación, la función objetivo consiste en minimizar el *makespan* (C_{max}) como indica la ecuación 3. Las restricciones 4 y 5 se aseguran de que C_{max} sea mayor o igual al tiempo de terminación del trabajo j y el trabajo j' en la máquina i. El tiempo de terminación de cada operación se calcula en las restricciones 6 y 7. Las restricciones 8 y 9 son restricciones de precedencia, donde se establece que cada operación puede ser ejecutada cuando su operación precedente fue ejecutada. Las ecuaciones 10 a 13 satisfacen el requerimiento de que solo un trabajo puede ser procesado en una máquina en cualquier tiempo. Debido a que Z_{ijk} es un entero entre 0 y 1, al agregar M, un entero positivo lo suficientemente grande, tiene el efecto de eliminar la restricción asociada. Las ecuaciones 14 y 15 hacen cumplir las operaciones, en el periodo de recalendarización, para empezar después

del tiempo de recalendarización. Además, cada operación puede ser iniciada con su respectiva máquina inactiva. Las restricciones 16 y 17 especifican la no negatividad. Y el conjunto de restricciones 18 a 21 especifican que deben ser enteros entre 0 y 1. (Kundakcı & Kulak, 2016)

4.2 Representación del problema

El problema puede ser representado por un grafo disjunto, con un nodo por cada operación, 0 y f como dos nodos que representan el inicio y final de todas las tareas. Para cada dos operaciones consecutivas en el mismo trabajo existe un arco dirigido y. Para cada par de operaciones que emplean la misma máquina, existen dos arcos en sentidos opuestos que indican la precedencia. Cada nodo tiene asociado un peso, que indica el tiempo de ejecución en la operación. (Haro et al., 2004) La figura 4.1 representa el grafo disjunto asociado a un problema de 3x3.

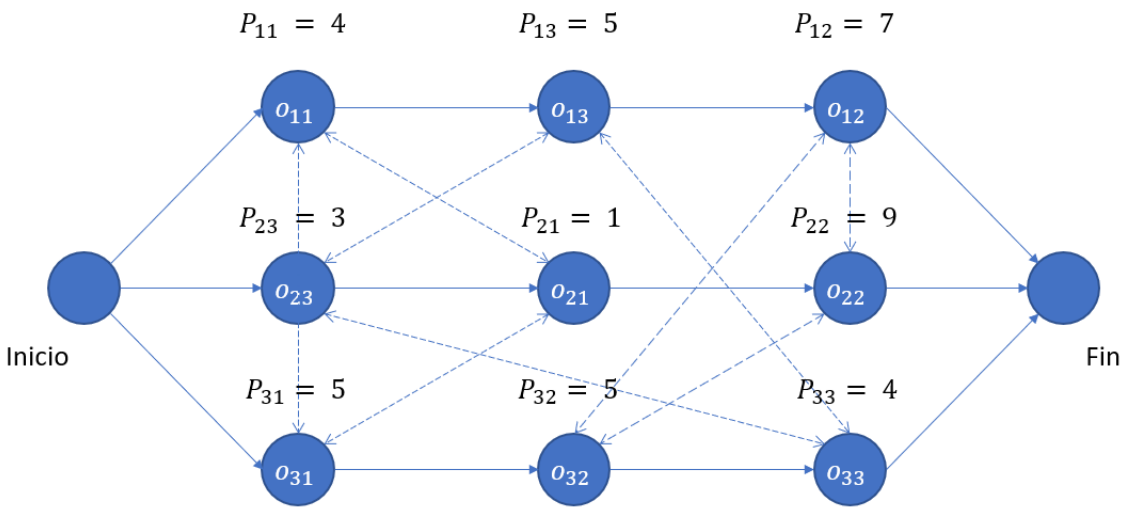


Figura 4.1 Representación del JSSP en forma de grafo disjunto.

4.3 Representación de la solución

Las soluciones candidatas del problema pueden ser representadas con un diagrama de Gantt, que se utiliza para mostrar el tiempo de dedicación para las diferentes operaciones que requieren realizarse en los diferentes trabajos a lo largo de un tiempo total requerido hasta

completar todos los trabajos. En la figura 4.2 se muestra una solución candidata para una instancia pequeña del problema.

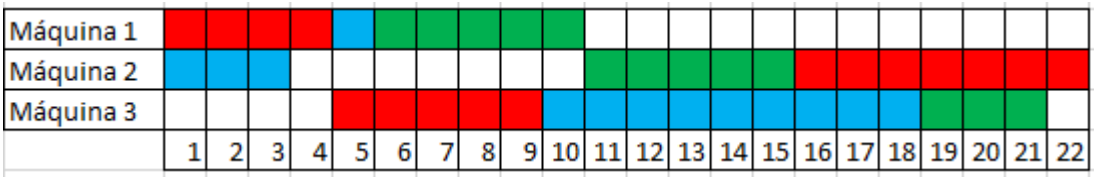


Figura 4.2 Diagrama de Gantt de una solución candidata al problema de la figura 2

4.4 Complejidad del problema

El JSSP, al ser el problema del cual se derivan los problemas de *Flow shop*, que son casos especiales considerados NP-completos, también se considera NP-completo. El problema *job shop* de 2 máquinas se considera NP-completo, por lo tanto, un problema *job shop* de más de 2 máquinas también es NP-completo. (Garey et al., 1976)

Capítulo 5 Metodología de solución

5.1 Metodología

- Desarrollar un estudio sobre el problema JSSP estático.
- Hacer una revisión del marco teórico de JSSP dinámico.
- Implementar el algoritmo de recocido simulado para JSSP.
- Aplicar un método de sintonización analítica de recocido simulado para JSSP.
- Analizar y seleccionar los eventos dinámicos, los indicadores de evaluación y las estrategias de calendarización dinámica para el problema JSSP.
- Analizar y seleccionar las instancias para el problema JSSP Dinámico (DJSSP) a utilizar para evaluar el desempeño de los algoritmos desarrollados.
- Implementar y evaluar el algoritmo de recocido simulado para DJSSP.
- Analizar y seleccionar las estrategias de mejora para el algoritmo de recocido simulado para el problema DJSSP.
- Realizar la evaluación experimental del desempeño de los algoritmos.
- Realizar la evaluación estadística de los algoritmos.

5.2 Sintonización de parámetros:

El método ANDYMARK (Frausto-Solís et al., 2006) define una manera de encontrar la longitud del ciclo interno de SA

L_k	Longitud de la cadena de Markov
C_k	Temperatura
$\propto$	Factor de enfriamiento
β	Factor de incremento de la cadena de Markov
N	Número de ejecuciones de la cadena de Márkov
S	Solución
$Z(S)$	Costo de la solución

$\Delta \boldsymbol{Zmax}$	Deterioro máximo de una solución
$\Delta \boldsymbol{Zmin}$	Deterioro mínimo de una solución

En el principio del proceso, la longitud de la cadena de Markov es muy pequeña ($L_k = L_1 \approx 1$). En el tiempo en que k tiende a infinito, el valor de C_k se decrementa de acuerdo con la función de enfriamiento hasta que se alcanza la temperatura final, con una función:

$$C_{k+1} = \propto C_k \tag{22}$$

Donde alfa esta normalmente en el rango de $0.7 \leq \propto \leq 0.99$.

En esto sentido, la longitud de cada cadena de Markov debe ser incrementada de manera inversa al incremento de C_k. Esto significa que L_k debe incrementarse hasta alcanzar L_{max} en C_f aplicando un incremento en el factor de la cadena de Markov. La función de enfriamiento se aplica varias veces hasta que la temperatura final se alcanza. La cadena de Markov se puede modelar como:

$$L_{k+1} = \beta L_k \tag{23}$$

L_k representa la longitud de la cadena de Markov actual en una temperatura dada, esto significa el número de iteraciones del ciclo de Metrópolis para una temperatura k. Entonces L_{k+1} representa la longitud de la siguiente cadena de Markov. En este modelo de Markov, Beta representa un incremento del número de iteraciones en el siguiente ciclo de Metrópolis.

Si la función de enfriamiento se aplica una y otra vez hasta que $C_k = C_f$, la siguiente función geométrica se obtiene fácilmente:

$$C_f = \propto^n C_1 \tag{24}$$

Conociendo la temperatura inicial C_1 y la final C_f y el coeficiente de enfriamiento $\propto$, el número de veces que el ciclo de metrópolis se ejecuta se puede calcular como:

$$N = \frac{ln\, C_f - ln\, C_1}{ln \propto} \tag{25}$$

Aplicando sistemáticamente la ecuación 23 otra función geométrica se obtiene:

$$L_{max} \;=\; \beta^n \, L_1 \tag{26}$$

Cuando se sabe n, el valor del coeficiente de incremento β se calcula como:

$$\beta \;=\; exp(\frac{ln\, L_{max} - ln\, L_1}{n}) \tag{27}$$

Cuando se conocen L_{max}, L_1 y β, la longitud de cada cadena de Markov para cada ciclo de temperatura se puede calcular utilizando la ecuación 23. De esta manera, L_k se calcula dinámicamente desde $L_k \;=\; 1$ para $c1$ hasta L_{max} en C_f.

La probabilidad de aceptar una solución nueva es cercana a 1 a altas temperaturas, así que el deterioro de la función de costo es máximo. La temperatura inicial C_1 se asocia con el deterioro máximo permitido y la probabilidad de aceptación definida. Definiendo S_i como la solución actual y S_j una nueva solución propuesta, y $Z(S_i)$ y $Z(S_j)$ como los costos asociados a S_iy S_j; los deterioros máximos y mínimos se expresan como $\Delta Zmax$ y $\Delta Zmin$. Entonces, la probabilidad de $P(\Delta Zmax)$ de aceptar una solución con la máxima deterioración es (26) y entonces C_1 se puede calcular como (29). De manera similar, la temperatura final se establece de acuerdo con la probabilidad $P(\Delta Zmin)$ de aceptar una nueva solución con el deterioro mínimo (véase la ecuación 30).

$$\exp\left(\frac{-\Delta Zmax}{C_1}\right) = P(\Delta Zmax) \tag{28}$$

$$C_1 = \frac{-\Delta Zmax}{\ln\,(P(\Delta Zmax))} \tag{29}$$

$$C_f = \frac{-\Delta Zmin}{\ln\,(P(\Delta Zmin))} \tag{30}$$

Con estos parámetros, se pueden encontrar soluciones cerca del óptimo o en algunos casos, el óptimo. La temperatura inicial puede ser extremadamente alta porque, de acuerdo con la ecuación 29, C_1 es extremadamente afectada por ($\Delta Zmax$) (Frausto-Solis et al., 2007).

5.3 Sintonización de SA con sintonización de longitud del ciclo de Markov

Se implementó el método *Andymark* de sintonización de parámetros para el algoritmo SA desarrollado para JSSP estático. En la figura 5.4 se muestra el pseudocódigo del algoritmo implementado.

```
1   SA(T0, Tf, alfa, L0, Lf, beta):
2    Tk = T0
3     SolInicial = LS()
4    Makespan = calcularMakespan(SolInicial)
5    Sol = BestSol = SolInicial
6    Mientras (Tk >= Tf):
7     Lk = L0
8     Mientras (Lk < Lf):
9      SolNew = perturbación(Sol)
10     MakespanNew = calcularMakespan(SolNew)
11     Incremento = MakespanNew – Makespan
12     Si (incremento < 0):
13      Sol = SolNew
14      Makespan = MakespanNew
15      Sino:
16      U = rand()
17      Boltz = exp(-(incremento / Tk))
18       Si (U < Boltz):
19       Sol = SolNew
20       Makespan = MakespanNew
21     Lk += Lk * beta
22    Tk = Tk * alfa
23    Si (Sol < BestSol):
24     BestSol = Sol
25  Fin_SA
```

Figura 5.1 Pseudocódigo del algoritmo implementado.

5.4 Metodología de solución de eventos dinámicos

5.4.1 Descompostura de máquina *(Machine breakdown)*:

Una máquina M_n deja de estar disponible por un tiempo t, y el trabajo que se está realizando continúa al terminar el tiempo t. El tiempo entre dos fallas de máquina y su tiempo de reparación se asume que sigue una distribución exponencial. El tiempo promedio entre fallas y el tiempo promedio de reparación son dos parámetros relacionados con la descomposición de las máquinas. (L. Zhang et al., 2013)

Trabajos	Máquinas y tiempos de procesamiento					
J1	4	M1	5	M3	7	M2
J2	3	M2	1	M1	9	M3
J3	5	M1	5	M2	4	M3
Descomposición	15	M2	3			

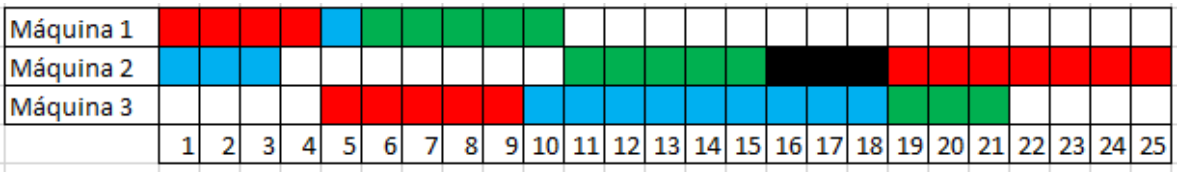


Figura 5.2 Diagrama de Gantt demostrando la descomposición de una máquina.

5.4.2 Llegada de nuevo trabajo (*New job arrival)*:

Un nuevo trabajo J_n se agrega a los trabajos existentes, con su propio orden de secuencia y tiempos de ejecución. En DJSSP, la distribución de llegada de trabajos sigue una distribución de Poisson. Por tanto, el tiempo entre llegadas de trabajo se acerca una distribución exponencial. (L. Zhang et al., 2013)

Trabajos	Máquinas y tiempos de procesamiento					
J1	4	M1	5	M3	7	M2
J2	3	M2	1	M1	9	M3
J3	5	M1	5	M2	4	M3
J4: 6	3	M3	5	M2	6	M1

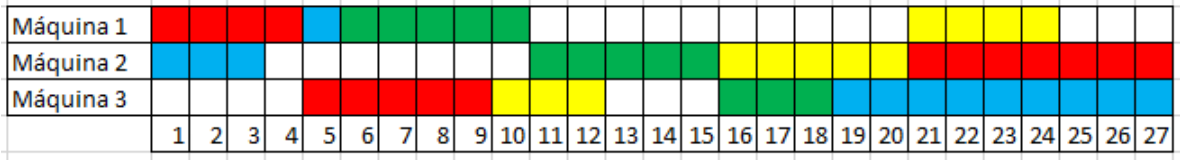


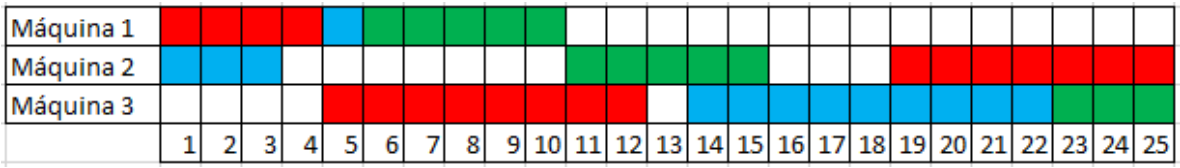
Figura 5.3 Diagrama de Gantt con un trabajo J4 añadido.

5.4.3 Cambio en el tiempo de procesamiento (*Change in the processing time)*:

El tiempo de procesamiento t para un trabajo $J_n$en una maquina M_n cambia al nuevo tiempo t_n.

Ejemplo:

Trabajos	Máquinas y tiempos de procesamiento					
J1	4	M1	5	M3	7	M2
J2	3	M2	1	M1	9	M3
J3	5	M1	5	M2	4	M3
J2		M3	8			



Trabajos	Máquinas y tiempos de procesamiento					
J1	4	M1	5	M3	7	M2
J2	3	M2	1	M1	9	M3
J3	5	M1	5	M2	4	M3

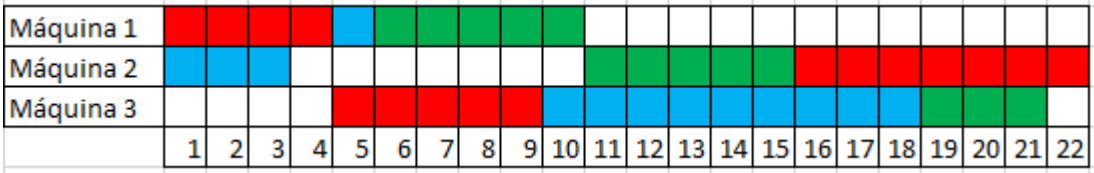


Figura 5.4 Diagrama de Gantt demostrando el nuevo tiempo de procesamiento.

5.5 Búsqueda local implementada.

Se implementó una búsqueda local como estrategia de mejora de la solución, ésta búsqueda va realizando una perturbación en el vecindario para encontrar alguna mejora en la solución. En caso de que no haya una mejora de la solución en n iteraciones, se conserva la mejor encontrada. En la figura 5.5 se muestra el seudocódigo del método implementado.

```
1     LS(solucion, iteraciones):
2      solInicial = solucion
3      makespaninicial = calcularMakespan(solInicial)
4      Mientras (cambios <= iteraciones):
5       Bestsol = solInicial
6       Bestmakespan = calcularMakespan(bestsol)
7       soltemp = perturbacion(sol)
8       makespantemp = calcularMakespan(soltemp)
9       Si (bestmakespan > makespantemp):
10       Bestsol = soltemp
11       Bestmakespan = makespantemp
12      Else:
13       Cambios++
14    Fin LS
```

Figura 5.5 Pseudocódigo de la búsqueda local implementada.

Capítulo 6 Resultados y discusión

6.1 Instancias de prueba

Los resultados se obtuvieron a partir de las instancias provistas por Kundakcı & Kulak (2016), que constan de 15 instancias diferentes, de tamaños 5x5, 6x5, 8x5, 10x5, 10x6, 15x5, 10x8, 15x5, 10x8, 10x9, 20x5, 10x10, 22x5, 12x10, 13x10, 20x7 y 15x10.

6.2 Detalles experimentales

Para el algoritmo implementado se realizaron 20 ejecuciones de cada instancia midiendo tiempo de ejecución y calidad de solución (*makespan*).

De las 20 ejecuciones se obtuvo el promedio y se seleccionó el mejor valor obtenido (mejor *makespan*) en cada una, para realizar la comparación a detalle.

La ejecución de los algoritmos se realizó en una computadora personal con las siguientes características:

- Procesador: Intel(R) Core(TM) i5-9600K CPU @ 3.70GHz
- Memoria: 16gb DDR4 a 2666 MHz
- Sistema operativo: Windows 10 Pro

6.3 Análisis de resultados

En la tabla 6.1 se muestra el mejor obtenido en 20 ejecuciones por los métodos de recocido simulado implementados. La columna SA corresponde a la primera versión del algoritmo. Las columnas 3, 5, y 7 corresponden al mejor valor de *makespan* encontrado por cada una de las tres variantes que incorporan búsqueda local en el método de solución SA; SAV2_AC, en el que se ejecutó la búsqueda local cuando una solución fue aceptada por tener menor *makespan*. SAV2_PB, en el que la búsqueda local se aplica cuando una solución fue aceptada por medio de la probabilidad de Boltzmann, y SAV2_ALL, que ejecuta la búsqueda en cuando se acepta la solución en cualquiera de los dos casos mencionados. Las columnas 4, 6, y 8 muestran la diferencia de cada uno de los algoritmos con respecto al algoritmo SA sin

búsqueda local. En la tabla 6.2 se muestra el promedio obtenido por cada método en cada instancia.

Tabla 6.1 Comparación entre los diferentes métodos SA realizados.

Instancia	**SA**	**SAV2_AC**	**Diferencia (%)**	**SAV2_CB**	**Diferencia (%)**	**SAV2_ALL**	**Diferencia (%)**
5X5	40	40	0.00	41	-2.50	40	0.00
6X5	532	532	0.00	532	0.00	532	0.00
8X5	662	662	0.00	662	0.00	662	0.00
10X5	636	616	3.14	616	3.14	616	3.14
10X6	772	677	12.31	677	12.31	677	12.31
15X5	993	993	0.00	993	0.00	993	0.00
10X8	703	688	2.13	705	-0.28	705	-0.28
10X9	871	831	4.59	834	4.25	855	1.84
20X5	1163	1147	1.38	1147	1.38	1147	1.38
10X10	1055	1045	0.95	1029	2.46	985	6.64
22X5	1521	1458	4.14	1484	2.43	1458	4.14
12X10	860	854	0.70	851	1.05	802	6.74
13X10	969	959	1.03	901	7.02	908	6.30
20X7	1506	1426	5.31	1435	4.71	1436	4.65
15X10	1164	1111	4.55	1110	4.64	1098	5.67

Tabla 6.2 Promedio de los resultados obtenidos de los métodos SA realizados

Instancia	SA	SAV2_AC	SAV2_CB	SAV2_ALL
5X5	52.3	54.9	52.15	58.55
6X5	594	604.25	615.15	616.95
8X5	788	727.55	763.1	757.7
10X5	780	741.75	802.05	783.65
10X6	895	829.2	842.5	880.3
15X5	1171	1196.75	1178.55	1190.8
10X8	914	921.3	916	905.1
10X9	1113	1128.55	1077.55	1057.55
20X5	1448	1467.15	1452.25	1418.3
10X10	1257	1289.7	1243.15	1294.4
22X5	1719	1649.8	1670.2	1715.25
12X10	1016	1030.75	996.75	1027.05
13X10	1145	1152	1108.4	1132.85
20X7	1774	1697.15	1690.75	1664.4
15X10	1353	1319.2	1352.25	1365.5

Se puede observar que, en su mayoría, en las segundas versiones se encontró una mejora con respecto a la primera versión, exceptuando las instancias donde se encontró la misma, o una peor solución.

En la tabla 6.3, se muestra la comparación de los métodos de recocido simulado con los métodos genéticos reportados por Kundakcı & Kulak (2016). Las columnas SAV2_AC, SAV2_CB, y SAV2_ALL bajo recocido simulado corresponden a los métodos reportados en este artículo. Las columnas GA, GAKK, GASPT, GALPT, GASRPT, GALRPT, GAM, y TS bajo algoritmo genético corresponden a los diferentes métodos reportados por Kundakcı & Kulak.

En la tabla 6.4 se muestra una comparación de la mejor solución encontrada por cada grupo de algoritmos. Las columnas SA y GA/TS corresponden al mejor resultado encontrado en las categorías de recocido simulado y algoritmo genético, respectivamente. La última columna indica la diferencia entre los mejores resultados encontrados en cada categoría.

Tabla 6.3 Comparación de los resultados obtenidos con los resultados reportados por Kundakcı & Kulak.

INSTANCIA	RECOCIDO SIMULADO			ALGORITMO GENÉTICO							
	SAV2_AC	SAV2_CB	SAV2_ALL	GA	GAKK	GASPT	GALPT	GASRPT	GALRPT	GAM	TS
5X5	40	41	**40**	57	53	53	51	52	53	51	75
6X5	532	532	**532**	581	557	572	573	579	557	557	778
8X5	662	662	**662**	757	757	699	779	706	699	699	850
10X5	616	616	**616**	624	624	624	624	624	624	624	911
10X6	677	677	**677**	710	682	738	707	733	682	682	1342
15X5	993	993	**993**	1018	1001	1032	1115	1001	1001	1001	1688
10X8	**688**	705	705	1317	1112	1191	1269	1111	1034	1027	1508
10X9	**831**	834	855	1757	1049	1635	1755	1078	1050	1049	2537
20X5	1147	1147	**1147**	1900	1564	1657	1898	1361	1468	1361	2097
10X10	1045	1029	**985**	3339	1410	2310	2120	1455	1453	1389	3655
22X5	1458	1484	**1458**	2807	1494	2079	2046	1458	1458	1458	2950
12X10	854	851	**802**	2995	1046	1640	1883	1069	1054	1002	3136
13X10	959	**901**	908	3227	1082	2354	2186	1175	1063	1016	3445
20X7	1344	1340	1436	3439	1340	2927	2388	1409	1333	**1326**	3541
15X10	1111	1110	**1098**	4541	1324	3242	3176	1428	1335	1280	4531

Los mejores resultados están resaltados con negritas. Se puede observar que, en 13 de las 15 instancias, se encontró una solución mejor a la reportada; en una se encontró la misma solución, y solo en una no hubo mejora, a pesar de tener un valor con poca diferencia respecto a la solución mejor encontrada.

En la tabla 6.4, se muestra la comparación de los tiempos de ejecución de los métodos SA en milisegundos y segundos, como indican los encabezados.

Tabla 6.4 Tiempos de ejecución de los métodos.

Instancia	*Recocido Simulado (ms)*			*Algoritmo Genético (s)*							
	SAV2_AC	SAV2_CB	SAV2_ALL	GA	GAKK	GASPT	GALPT	GASRPT	GALRPT	GAM	TS
5X5	0.000045	0.000043	**0.00005**	2	1	2	2	1	1	0.89	0.22
6X5	**0.000041**	0.000043	0.000054	12	5	9	11	6	6	6	0.53
8X5	0.000054	**0.000042**	0.000046	10	4	9	12	6	6	6	0.36
10X5	0.000063	0.000066	**0.000053**	46	15	38	39	16	13	18	2
10X6	0.000075	**0.000073**	0.0001	173	42	104	95	42	49	48	3
15X5	**0.000096**	0.000098	0.000098	290	74	226	228	74	83	94	5
10X8	0.000141	0.000144	0.000141	384	295	375	324	339	298	313	5
10X9	**0.000211**	0.000211	0.00023	627	365	435	587	406	375	431	15
20X5	**0.000203**	0.000212	0.000214	348	339	345	341	309	342	268	17
10X10	**0.000224**	0.000233	0.000229	368	367	364	346	313	360	278	16
22X5	0.000224	0.000255	**0.000203**	559	354	529	435	373	377	295	51
12X10	0.000288	0.000299	**0.000271**	387	318	352	301	376	322	346	30
13X10	0.000312	0.000292	**0.000284**	337	329	314	328	320	280	327	32
20X7	**0.000049**	0.000053	0.000052	510	349	459	504	247	345	294	92
15X10	0.000177	0.000474	**0.000363**	490	448	415	472	458	467	450	114

Los menores tiempos de ejecución están resaltados con negritas. Como se puede observar, las tres versiones del algoritmo presentan un tiempo de ejecución mucho menor a los tiempos reportados.

En la tabla 6.5 se muestra la comparación de los métodos de recocido simulado con los métodos de algoritmo heurístico Kalman (HKA por sus siglas en inglés) reportados por (H. Zhang et al., 2021). Las columnas SAV2_AC, SAV2_CB y SAV2_ALL bajo recocido simulado corresponden a los métodos reportados en este artículo. Las columnas HKA, HKA VON NEUMANN, e IHKA MOORE bajo HKA corresponden a los diferentes métodos reportados.

En la tabla 6.6 se muestra una comparación de la mejor solución encontrada por cada grupo de algoritmos. Las columnas SA, GA/TS, y HKA corresponden al mejor resultado encontrado en las categorías de recocido simulado, recocido genético, y HKA respectivamente. La última columna indica la diferencia entre los mejores resultados encontrados en cada categoría.

Tabla 6.5 Comparación de los resultados obtenidos con los resultados reportados por H. Zhang et al.

INSTANCIA	**RECOCIDO SIMULADO**			**HKA**		
	SAV2_AC	SAV2_CB	SAV2_ALL	HKA	IHKA VON NEUMANN	IHKA MOORE
5X5	40	41	40	51	51	51
6X5	532	532	532	557	552	557
8X5	662	662	662	699	699	699
10X5	616	616	616	624	624	624
10X6	677	677	677	682	682	682
15X5	993	993	993	1001	1001	1001
10X8	688	705	705	947	953	944
10X9	831	834	855	1005	1005	1005
20X5	1147	1147	1147	1202	1217	1218
10X10	1045	1029	985	1292	1287	1289
22X5	1458	1484	1458	1458	1458	1458
12X10	854	851	802	912	914	918
13X10	959	901	908	949	947	952
20X7	1344	1340	1436	1246	1260	1246
15X10	1111	1110	1098	1112	1130	1122

Tabla 6.6 Comparación de los mejores resultados obtenidos por cada grupo de algoritmos

INSTANCIA	SA	GA/TS	DIF(%)	HKA	DIF(%)
5X5	40	51	-27.50	51	-27.50
6X5	532	557	-4.70	552	-3.76
8X5	662	699	-5.59	699	-5.59
10X5	616	624	-1.30	624	-1.30
10X6	677	682	-0.74	682	-0.74
15X5	993	1001	-0.81	1001	-0.81
10X8	688	1027	-49.27	944	-37.21
10X9	831	1049	-26.23	1005	-20.94
20X5	1147	1361	-18.66	1202	-4.80
10X10	1029	1389	-34.99	1287	-25.07
22X5	1458	1458	0.00	1458	0.00
12X10	851	1002	-17.74	912	-7.17
13X10	901	1016	-12.76	947	-5.11
20X7	1340	1326	1.04	1246	7.01
15X10	1110	1280	-15.32	1112	-0.18

Se puede observar que en 13 de las 16 instancias hubo una mejora en la solución encontrada con respecto a ambos algoritmos reportados, en una se encontró el mismo resultado, y en una no se encontró mejora, aunque solo hay diferencia significativa con el resultado obtenido por el algoritmo HKA.

6.4 Evaluación estadística de los algoritmos

Se hizo un análisis estadístico de los resultados mediante una prueba de Friedman comparando los 3 diferentes métodos implementados entre sí; y una prueba de rangos con signo Wilcoxon comparando los resultados obtenidos por los 3 diferentes métodos implementados con los métodos genéticos y Kalman reportados en la literatura.

En la prueba de Friedman, se obtuvo un valor de chi cuadrada de 1.15 con 2 grados de libertad, y un *p-value* de 0.5623. Comparándolo con la tabla de chi cuadrada, el valor obtenido en la prueba es menor al valor de la tabla, demostrando en este caso que no hay diferencia significativa entre los valores obtenidos por los 3 diferentes métodos propuestos.

En la tabla 6.7, se muestran los resultados de la prueba de Wilcoxon.

Tabla 6.7 Resultados de la prueba de rangos con signo Wilcoxon.

		SAV2_AC	SAV2_CB	SAV2_ALL
GA	W	0	0	0
	p-value	0.00006104	0.00006104	0.00006104
GAKK	W	1	0	8
	p-value	0.0008898	0.001094	0.003438
GASPT	W	0	0	0
	p-value	0.00006104	0.00006104	0.00006104
GALPT	W	0	0	0
	p-value	0.00006104	0.00006104	0.00006104
GASRPT	W	0	4	4
	p-value	0.001094	0.001617	0.002572
GALRPT	W	4	9	7
	p-value	0.002578	0.004121	0.004719
GAM	W	5	12	8
	p-value	0.003165	0.006968	0.00573
TS	W	0	0	0
	p-value	0.00006104	0.00006104	0.00006104
HKA	W	16	19	12
	p-value	0.02379	0.02141	0.01202
IHKA_VN	W	16	19	12
	p-value	0.02379	0.02138	0.01202
IHKA_M	W	13	19	12
	p-value	0.01431	0.02141	0.01202

En todos los casos, se demostró la hipótesis alternativa, donde la media de la diferencia entre los valores es 0, demostrando que la diferencia es significativa entre los valores obtenidos por los métodos implementados y los valores reportados en el estado del arte.

Capítulo 7 Conclusiones y trabajos futuros

7.1 Conclusiones

Con este proyecto, se presenta un método para encontrar soluciones a instancias dinámicas del problema de calendarización de tareas (JSSP). Se utilizó un nuevo algoritmo híbrido compuesto por una combinación de recocido simulado, y búsqueda local.

Los resultados experimentales permiten observar que, en la mayoría de las instancias, existe una mejora entre el método propuesto y los métodos reportados por el estado del arte. Además, se muestra que hay una gran diferencia entre los tiempos de ejecución. Los métodos genéticos presentan tiempos en segundos, y los métodos de recocido simulado presentan tiempos en milisegundos.

7.2 Principal contribución

La contribución principal de este proyecto es un algoritmo híbrido para una versión del DJSSP que obtiene resultados comparables con el estado del arte. Este algoritmo genera resultados de buena calidad en tiempos cortos.

7.3 Trabajos futuros

Algunas de las posibles líneas de investigación a futuro para mejorar el desempeño de los algoritmos enfocados al problema podrían ser:

- Elaborar metaheurísticas para eventos dinámicos no abarcados por este trabajo.
- Implementar estrategias de paralelización para el algoritmo
- Modificar el diseño del algoritmo para un enfoque multiobjetivo
- Obtener las mejores soluciones para las instancias utilizadas.

Referencias

Abdel-Basset, M., Abdel-Fatah, L., & Sangaiah, A. K. (2018). Chapter 10 - Metaheuristic Algorithms: A Comprehensive Review. En A. K. Sangaiah, M. Sheng, & Z. Zhang (Eds.), *Computational Intelligence for Multimedia Big Data on the Cloud with Engineering Applications* (pp. 185–231). Academic Press. https://doi.org/10.1016/B978-0-12-813314-9.00010-4

Abdolrazzagh-Nezhad, M., & Abdullah, S. (2017). *Job shop scheduling: Classification, constraints and objective functions*.

Baykasoğlu, A., & Karaslan, F. S. (2017). Solving comprehensive dynamic job shop scheduling problem by using a GRASP-based approach. *International Journal of Production Research*, *55*(11), 3308–3325. https://doi.org/10.1080/00207543.2017.1306134

Boyd, S. P., & Vandenberghe, L. (2004). *Convex optimization*. Cambridge University Press.

Brucker, P., Sotskov, Y. N., & Werner, F. (2007). Complexity of shop-scheduling problems with fixed number of jobs: A survey. *Mathematical Methods of Operations Research*, *65*(3), 461–481. https://doi.org/10.1007/s00186-006-0127-8

Frausto-Solis, J., Román, E. F., Romero, D., Soberon, X., & Liñán-García, E. (2007). Analytically tuned simulated annealing applied to the protein folding problem. *International Conference on Computational Science*, 370–377.

Frausto-Solís, J., Sanvicente-Sánchez, H., & Imperial-Valenzuela, F. (2006). ANDYMARK: an analytical method to establish dynamically the length of the

markov chain in simulated annealing for the satisfiability problem. *Asia-Pacific Conference on Simulated Evolution and Learning*, 269–276.

Garey, M. R., & Johnson, D. S. (1990). *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co.

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

Haro, S., Sánchez-Martín, P., & Collado, J. (2004). *Secuenciación de tareas mediante metaheurísticos*.

Kellerer, H., Pferschy, U., & Pisinger, D. (2004). Introduction. En H. Kellerer, U. Pferschy, & D. Pisinger (Eds.), *Knapsack Problems* (pp. 1–14). Springer. https://doi.org/10.1007/978-3-540-24777-7_1

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

Kundakcı, N., & Kulak, O. (2016). Hybrid genetic algorithms for minimizing makespan in dynamic job shop scheduling problem. *Computers & Industrial Engineering*, *96*, 31–51. https://doi.org/10.1016/j.cie.2016.03.011

Lin, S.-C., Goodman, E. D., & Punch III, W. F. (1997). A Genetic Algorithm Approach to Dynamic Job Shop Scheduling Problem. *ICGA*, 481–488.

Michiels, W., Aarts, E. H. L., & Korst, J. (2018). Theory of Local Search. En R. Martí, P. M. Pardalos, & M. G. C. Resende (Eds.), *Handbook of Heuristics* (pp. 299–339). Springer International Publishing. https://doi.org/10.1007/978-3-319-07124-4_6

Mohan, J., Lanka, K., & Rao, A. N. (2019). A Review of Dynamic Job Shop Scheduling Techniques. *Procedia Manufacturing*, *30*, 34–39. https://doi.org/10.1016/j.promfg.2019.02.006

van Laarhoven, P. J. M., & Aarts, E. H. L. (1987). Simulated annealing. En P. J. M. van Laarhoven & E. H. L. Aarts (Eds.), *Simulated Annealing: Theory and Applications* (pp. 7–15). Springer Netherlands. https://doi.org/10.1007/978-94-015-7744-1_2

Wang, Z., Zhang, J., & Yang, S. (2019). An improved particle swarm optimization algorithm for dynamic job shop scheduling problems with random job arrivals. *Swarm and Evolutionary Computation*, *51*, 100594. https://doi.org/10.1016/j.swevo.2019.100594

Yang, X. (2008). Introduction to mathematical optimization. *From linear programming to metaheuristics*.

Zhang, H., Buchmeister, B., Li, X., & Ojstersek, R. (2021). Advanced Metaheuristic Method for Decision-Making in a Dynamic Job Shop Scheduling Environment. *Mathematics*, *9*(8), 909. https://doi.org/10.3390/math9080909

Zhang, L., Gao, L., & Li, X. (2013). A hybrid genetic algorithm and tabu search for a multi-objective dynamic job shop scheduling problem. *International Journal of Production Research*, *51*(12), 3516–3531. https://doi.org/10.1080/00207543.2012.751509