EDUCACIÓN
SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLÓGICO NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE CIUDAD MADERO

DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

DOCTORADO EN CIENCIAS DE LA INGENIERÍA

"POR MI PATRIA Y POR MI BIEN"

TESIS

MÉTODOS HEURÍSTICOS HÍBRIDOS PARALELOS PARA JOB SHOP SCHEDULING PROBLEM

Que para obtener el Título de

Doctor en Ciencias de la Ingeniería

Presenta

MC. Leonor Hernández Ramírez

D98260826

Director de Tesis

Dra. Guadalupe Castilla Valdez

Co-director de Tesis

Dr. Juan Frausto Solís

Cd. Madero, Tamaulipas Diciembre 2021

Cd. Madero, Tam. 19 de noviembre de 2021

OFICIO No. : U.158/21
ASUNTO: AUTORIZACIÓN DE IMPRESIÓN DE TESIS

C. LEONOR HERNÁNDEZ RAMÍREZ
No. DE CONTROL D98260826
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 Doctorado en Ciencias de la Ingeniería, se acordó autorizar la impresión de su tesis titulada:

"MÉTODOS HEURÍSTICOS HÍBRIDOS PARALELOS PARA JOB SHOP SCHEDULING PROBLEM"

El Jurado está integrado por los siguientes catedráticos:

PRESIDENTE:	DRA.	GUADALUPE CASTILLA VALDEZ
SECRETARIO:	DR.	JUAN FRAUSTO SOLÍS
PRIMER VOCAL:	DR.	NELSON RANGEL VALDEZ
SEGUNDO VOCAL:	DR.	PEDRO MARTÍN GARCÍA VITE
TERCER VOCAL:	DR.	LUCIANO AGUILERA VÁZQUEZ
DIRECTORA 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"®

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

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

c.c.p.- Archivo
MACG 'mdcoa'

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

_____

MCC. Leonor Hernández Ramírez

Dedicatoria

A mi esposa Jhovana y mis queridas hijas Mariana y Julieta, por ser motor de mi vida. Muero por ustedes, vivo por ustedes.

Agradecimientos

A mis directores de investigación. El Dr. Juan Frausto y la Dra. Guadalupe Castilla por su dedicación y apoyo cada día para poder realizar el trabajo de la mejor forma posible.

Al Tecnológico Nacional de México/Instituto Tecnológico de Ciudad Madero y al Laboratorio Nacional de Tecnologías de la Información (LANTI) por brindarme el acceso para utilizar el clúster Ehécatl para la ejecución de las pruebas experimentales.

Al Consejo Nacional de Ciencia y Tecnología por el apoyo económico brindado.

Índice general

Índice de Tablas

Índice de Figuras

CAPÍTULO 1

Introducción

A finales del siglo XVIII da inicio la Revolución Industrial, la cual es caracterizada por grandes transformaciones económicas, tecnológicas y sociales. Este periodo es marcado por la transición de mano de obra basada en trabajo manual y el uso de animales en trabajos de producción y transporte; al uso de máquinas para la fabricación industrial y el transporte de mercancías, lo cual ha representado una reducción significativa en el tiempo de fabricación. Sin embargo, con el paso de los años la demanda de artículos ha crecido de manera exponencial, y al mismo tiempo ha crecido la necesidad de fabricarlos utilizando el menor tiempo posible. Para tratar de reducir el tiempo de producción o fabricación se han realizado investigaciones en diferentes campos de las ciencias, tales como la investigación de operaciones [1], [2], [3] y la optimización computacional [4], [5], entre otras. Una de las soluciones encontradas en esas investigaciones es la generación de un programa, también llamado calendarización de tareas o *scheduling*, por su término en inglés.

Se puede decir que *scheduling* es la creación de un calendario de tareas para fabricar un artículo,

siguiendo una serie de pasos ordenados; por otro lado, un *schedule* es justamente el calendario de tareas correspondiente. Otra forma de definir informalmente *scheduling* es la proporcionada por [6], que señala que es un proceso de toma de decisiones, el cual es utilizado de manera regular en la industria y por empresas de servicios que buscan determinar la asignación de tareas en períodos de tiempo y cuya meta es optimizar uno o más objetivos.

Existen varios tipos de problemas de scheduling, en esta investigación se trabaja con el ya conocido Job Shop Scheduling Problem (JSSP), también llamado Calendarización de Tareas. Este es un problema abordado desde hace ya más de 50 años y es del tipo NP-duro [7], lo que indica que no existe actualmente un algoritmo que lo resuelva en tiempo polinomial. Este problema es extremadamente difícil de resolver y de una gran aplicabilidad industrial [6]. En este documento nos referiremos a ese problema indistintamente como JSSP o Job Shop Scheduling.

1.1. Antecedentes históricos

A lo largo del tiempo diversos autores han contribuido en el desarrollo de métodos, técnicas y diferentes aplicaciones para lo que conocemos hoy en día como scheduling.

Entre las referencias históricas en esta área se encuentran los trabajos de Giffler y Thompson, quienes introdujeron el término de *schedules* activos. Un *schedule* activo es un subconjunto de los programas o schedules factibles. Los mismos autores propusieron el algoritmo denominado GT (por sus apellidos, Giffler y Thompson) el cual enumera iterativamente todos los schedules activos para un problema dado [8].

En el año 1963, Fisher y Thompson propusieron tres instancias para job shop scheduling, denominadas mt06 (de 6 trabajos y 6 máquinas), mt10 (de 10 trabajos y 10 máquinas) y mt20 (de 20 trabajos y 5 máquinas) [3]. Estos tres problemas han hecho historia, ya que son usados por investigadores de todo el mundo para realizar pruebas a sus algoritmos. Cabe señalar que la instancia mt10, estuvo sin solución

óptima por más de 20 años hasta que Carlier y Pinson usando un algoritmo del tipo Branch and Bound lograron resolverla en el año 1985 [2].

Al pasar de los años, en la industria se presentan problemas cada vez más complejos que deben ser resueltos en el menor tiempo posible. La solución de estos problemas utilizando métodos clásicos por lo regular require gran poder de cómputo y el uso de un algoritmo exacto con este fin es prácticamente inviable. Por esta razón, además de su flexibilidad y versatilidad han surgido los algoritmos metaheurísticos, ya que este tipo de algoritmos pueden obtener una solución de buena calidad al problema en un tiempo razonablemente corto.

Así en el año 1975, los Algoritmos Genéticos (GA por sus siglas en inglés), fueron introducidos por Holland [9] y posteriormente fueron popularizados David Goldberg. Los GA son caracterizados por sus estrategias de búsqueda basados en poblaciones y por sus tres operadores básicos que son mutación, selección y cruza aplicados para procesos de reproducción.

Otro método aproximado utilizado exitosamente es Recocido Simulado (SA por sus siglas en inglés), el cual es uno de los más ampliamente estudiados actualmente. Este algoritmo se basa en una analogía con el proceso de recocido de metales; el cual consiste en calentar un sólido hasta derretirlo, para después enfriarlo de forma lenta hasta llegar al punto de solidificación. El algoritmo SA fue introducido por Kirkpatrick para la optimización de problemas combinatorios con mínimos locales [10]. SA utiliza técnicas de optimización no determinista, es decir no buscan la mejor solución en el entorno de la solución actual sino que generan aleatoriamente una solución cercana, la cual es aceptada como la mejor si tiene menor costo, o en caso contrario soluciones de mala calidad son aceptadas utilizando la probabilidad de Boltzman.

En el año 1991, Applegate desarrolló un algoritmo combinando Shifting Bottleneck Heuristic de Adams y un método Branch & Bound el cual demostró que la instancia mt10 de Muth y Thompson [3] ya no era un reto computacional [11]. También propusieron un conjunto de nuevas instancias conocidas como

las “Ten Tough Benchmarks”, el cual contiene 10 instancias difíciles incluyendo siete problemas que no pudieron ser resueltos incluso por su método aproximado en ese momento.

En 1993, Taillard propuso un conjunto de nuevas instancias constituidas por 80 instancias para Job Shop y 120 para Flow Shop [12], estas instancias han sido ampliamente usadas para realizar experimentaciones y posteriormente, comparación de los diferentes algoritmos creados a lo largo de los años por los investigadores actuales.

Diversos autores como Brucker [13], Martin y Shmoys [14] y Carlier [15] han trabajado y mejorado el rendimiento de los métodos exactos desde 1994 hasta la fecha, sin embargo, la eficiente aplicación de estos en los problemas NP-duros de gran tamaño es una de las barreras existentes al día de hoy debido a los altos requerimientos en tiempo de cómputo para encontrar una solución.

A falta de algoritmos exactos con aplicación viable, la investigación para desarrollar algoritmos altamente eficientes se dirige hacia la creación y utilización de algoritmos híbridos para aprovechar las ventajas de cada heurística embebida para obtener mejores soluciones a los problemas NP-duros. Combinando la hibridización junto con el uso de estrategias de cómputo paralelo para buscar reducir el tiempo de procesamiento que por lo general es muy alto.

1.2. Definición general: Job Shop Scheduling Problem

En JSSP, se tiene un conjunto de trabajos para ser procesados en una serie de máquinas. Cada trabajo está compuesto por una serie de operaciones que se deben llevar a cabo en un orden ya definido y bajo una serie de restricciones impuestas por el proceso mismo que se busca realizar.

El objetivo de JSSP es organizar el orden de procesamiento de las operaciones involucradas en el proceso, determinando para ello los tiempos de inicio de cada una, donde el cumplimiento del objetivo se

determina utilizando una o varias medidas de desempeño de acuerdo con el o los objetivos establecidos en el problema.

JSSP tiene en general dos tipos de restricciones: a) restricciones de precedencia, también conocidas como restricciones de secuencia, las cuales expresan las relaciones de precedencia (que una operación no puede iniciar si no ha terminado la anterior) entre las operaciones de cada trabajo y b) restricciones de recurso o capacidad, también llamadas restricciones conjuntivas, las cuales establecen que no más de un trabajo puede ser ejecutado en una máquina al mismo tiempo.

Un problema de scheduling se considera completamente solucionado si los tiempos de inicio de todas las operaciones son determinados y todas las restricciones de precedencia y de recursos no son violadas [16], [17], [18].

En general, el tamaño de un problema JSSP está definido por $j \times m$, donde j es el número de trabajos en $J = \{J_1, J_2, J_3, ..J_j\}$ y m es el número de máquinas del problema en $M = \{M_1, M_2, M_3, ...M_m\}$. Cada trabajo está compuesto por una serie de operaciones que se deben llevar a cabo en un orden ya definido por la instancia a resolver. Cada elemento del conjunto de trabajos J deberá estar relacionado con un elemento del conjunto de máquinas M. La relación que asocia los trabajos en J con las máquinas en M conforma una relación binaria $J \rightarrow M$, esta es llamada *operación (j,m)* [19]. El total de operaciones del problema es denotado por $O = \{O_1, O_2, O_3, \ldots O_k\}$ o siendo más específico $O = \{O_{1,1}, O_{1,2}, O_{1,3}, \ldots O_{j,m}\}$, donde $O = \{(j,m) \mid J_j \in J, M_m \in M\}$.

El conjunto de operaciones que corresponden a un mismo trabajo tiene una secuencia de procesamiento, la cual es llamada secuencia tecnológica S, tal que $S = \{S_1, S_2, S_3, \ldots S_m\}$. Por lo tanto, para cada trabajo en J se tiene la secuencia tecnológica: $O_{JS} = \{O_{j,1}, O_{j,2}, O_{j,3}, \ldots O_{j,s}\}$. Cada operación *(j, m)* tiene asociado un tiempo de procesamiento p que indica el tiempo requerido para procesar esa operación en la m-ésima máquina. Entonces, de forma general JSSP está definido mediante los siguientes conjuntos:

Conjunto de máquinas	$M = \{M_1, M_2, M_3, \ldots, M_m\}$
Conjunto de trabajos	$J = \{J_1, J_2, J_3, \ldots, J_j\}$
Conjunto de Operaciones	$O = \{O_1, O_2, O_3, \ldots, O_k\}$

Como ya se ha dicho, en JSSP es necesario encontrar el tiempo de inicio t para cada operación en O tratando de minimizar su *makespan* (C_{max}), que se define como el tiempo máximo para procesar todo el conjunto de operaciones.

1.3. Job Shop Scheduling Problem

El planteamiento matemático del Job Shop Scheduling Problem consiste en:

$$Minimizar\, C_{max} = Max(t_k + p_k) : \forall J_j \in J \tag{1.1}$$

Sujeto a:

$t_l \geq t_k + p_k$	Para todo $k, l \in O$ con k PREC l	(a)
$t_l \geq t_k + p_k$ o $t_k \geq t_l + p_l$	Para todo $k, l \in O$ con $M_k = M_l$	(b)

Donde:

t_k, t_l :	Tiempo de inicio de la operación k, l.
p_k, p_l	Tiempo de procesamiento de la operación k, l.
C_{max} :	Makespan del conjunto de operaciones.
PREC:	Indica la precedencia de las operaciones.

La restricción (a) indica la relación de precedencia de las operaciones, es decir, todas las operaciones de un trabajo deben ser realizadas en un orden preciso, tal que si *k PREC l*, entonces *l* no puede iniciar antes de que *k* termine. La restricción (b) implica que la máquina puede procesar únicamente una operación al mismo tiempo.

En general se asume que:

- Todas las máquinas están disponibles al tiempo cero.
- Cualquier fallo en las máquinas es ignorado.
- Una vez que una operación se empieza a procesar en una máquina esta no puede ser interrumpida hasta finalizar.
- Cuando una máquina termina de procesar una operación está disponible para iniciar otra.

1.4. Job Shop Scheduling Problem Multi-Objetivo

En el planteamiento anterior de JSSP el objetivo es el *makespan*; sin embargo la mayoría de los problemas de scheduling que surgen en el contexto industrial de manufactura pertenecen a la categoría de problemas de scheduling multi-objetivo, es decir varios objetivos son considerados de forma simultánea.

Este trabajo de investigación se centra en el desarrollo de un algoritmo para resolver el problema JSSP multi-objetivo. Se considera la minimización de tres objetivos de forma simultánea, *makespan, total tardiness y total flow time*. Recordar que el *makespan* es el tiempo máximo para procesar todo el conjunto de operaciones, *total tardiness* es la sumatoria de las diferencias positivas entre el makespan y la fecha de vencimiento de cada trabajo y *total flow time* es la sumatoria de los tiempos de terminación de todos los trabajos.

JSSP multi-objetivo se puede formular de la siguiente manera [20], [21]:

$$Optimizar\ F(x) = [f_1(x), f_2(x), ..., f_q(x)] \tag{1.2}$$

Sujeto a:

$$x \in S$$

Donde:

q :	Es el número de objetivos.
x :	Es el vector de variables de decisión.
S :	Representa la región factible.

Las funciones objetivo se definen mediante las ecuaciones (1.3), (1.4), y (1.5).

$$f_1 = min\,(max C_j) \forall J_j \in J \tag{1.3}$$

donde C_j es el *makespan* del trabajo j, el cual se obtiene al generar un orden de procesamiento para todas las operaciones del trabajo j.

$$f_2 = min\left(\sum_{j=1}^{n} T_j\right) = min\left(\sum_{j=1}^{n} max(0, C_j - D_j)\right) \tag{1.4}$$

donde $T_j = max(0, C_j - D_j)$ es la *tardiness* (tardanza) del trabajo j, y D_j es la fecha de vencimiento del trabajo j y es calculada con $D_j = \tau \sum_{i=1}^{m} p_{j,i}$ [22], donde $p_{j,i}$ es el tiempo requerido para procesar

el trabajo j en la máquina i, el cual está dado por la instancia a resolver. En este caso, la fecha de vencimiento del trabajo j es la suma del tiempo de procesamiento de todas sus operaciones en todas las máquinas, multiplicadas por un factor de estrechez (τ), que está en el rango $1.5 \leq \tau \leq 2.0$ [22], [23].

$$f_3 = min \sum_{j=1}^{n} C_j \tag{1.5}$$

También se asume que:

- Los tiempos de preparación de las máquinas entre operaciones no es tomado en cuenta.
- Una máquina no puede procesar más de una operación al mismo tiempo.
- No hay restricciones de precedencia entre operaciones de diferentes trabajos.
- Una vez iniciado el procesamiento de una operación se debe procesar hasta su terminación.
- Para cada operación ya está definido su tiempo de procesamiento.

1.5. Complejidad del problema

En general, algunos problemas de optimización combinatoria tienen como característica común que tienden a ser relativamente "fáciles"de plantear pero son mucho más difíciles de modelar y, en consecuencia, mucho más difíciles de resolver. La complejidad del JSSP radica en la gran cantidad de posibles soluciones que puede tener. JSSP consta de secuencias de operaciones para cada máquina. Cada secuencia se puede permutar independientemente de la secuencia de operaciones de otra máquina, esta es la razón por la cual el número total de posibles soluciones para JSSP es factorial.

Se dice entonces que Job Shop Scheduling pertenece a la clase de problemas NP-Hard para $m > 2$ [7], donde m denota el número de máquinas, lo que significa que no se podido encontrar una solución

óptima para un problema con más de dos máquinas en tiempo polinomial. También se considera uno de los problemas de optimización combinatoria que mayor desafío computacional plantea por lo que hasta la fecha actual no existe un algoritmo determinista que lo resuelva en tiempo polinomial, esta es una de las razones por las cuales hay varias instancias en las que aún no se conoce su valor óptimo [24].

1.6. Objetivos

Se detallan los objetivos a tratar en la presente investigación.

1.6.1. Objetivo general

Desarrollar métodos heurísticos híbridos de optimización para Job Shop Scheduling Problem Multi-Objetivo en cómputo paralelo utilizando OpenMP y/o MPI con desempeño mejor o equivalente que los de la literatura.

1.6.2. Objetivos específicos

1. Analizar algoritmos heurísticos para JSSP.
2. Desarrollar y evaluar métodos heurísticos híbridos para la solución secuencial de JSSP mono-objetivo y multi-objetivo.
3. Analizar las tecnologías paralelas OpenMP y/o MPI.
4. Diseñar y evaluar modelos de paralelización para los heurísticos desarrollados.
5. Realizar la evaluación experimental de los algoritmos propuestos.

1.7. Justificación del estudio

JSSP no solo es NP-Duro como ya ha sido demostrado [7], también es considerado uno de los problemas de optimización combinatoria que más desafío computacional plantea [24]. En esta investigación se planea identificar y utilizar instancias con el objetivo de solucionarlas utilizando el menor tiempo de procesamiento.

Entre las áreas de oportunidad o beneficios en el estudio de este problema se pueden mencionar:

- La aportación de un algoritmo implementado en cómputo paralelo que obtenga resultados de calidad que sean competitivos con los del estado del arte reduciendo el tiempo de procesamiento.
- Desarrollar una solución para la versión multi-objetivo del problema, ya que de acuerdo a la revisión de la literatura son muy escasos los trabajos que abordan esta versión del problema.

1.8. Estado del arte para JSSP Mono-Objetivo

En años recientes JSSP ha atraído la atención de un amplio grupo de investigadores quienes han propuesto un gran número de algoritmos heurísticos y metaheurísticos para encontrar una solución óptima o cercana al óptimo para el problema. A continuación se realiza un breve análisis sobre algunos trabajos publicados que tienen relevancia para la presente investigación.

En 2013, es implementado un algoritmo híbrido Neuro-Tabu para resolver el JSSP de forma secuencial y de forma paralela [17]. Se utiliza MPI para distribuir los cálculos entre los GPUs. En este trabajo se utilizan instancias de Taillard [12] para evaluar el desempeño. En la experimentación se utiliza un servidor de 6 núcleos Intel Core i7 (3.33GHz) con una tarjeta nVidia Tesla S2050 GPU y sistema operativo Ubuntu 10.04.4LTS de 64-bit.

En 2011 fue propuesto un algoritmo genético híbrido llamado biased random-key genetic algorithm (BRKGA) por [25] que obtuvo excelentes resultados. Utiliza una combinación de una búsqueda local con búsqueda tabú y una extensión del método gráfico para dos trabajos propuesto por [26]. El algoritmo fue evaluado utilizando un conjunto de 205 instancias, de las cuales en 57 de ellas se mejoró la mejor solución conocida. Este algoritmo fue implementado utilizando el lenguaje C++ y la experimentación se realizó en un equipo con procesador AMD Opteron 2427 (2.2 GHz) con el sistema operativo Linux Fedora release 12.

En 2015 Peng et al. [27] desarrollaron el algoritmo híbrido tabu search/path relinking (TS/PR) para el problema JSSP. Este algoritmo utiliza 30 soluciones iniciales aleatorias, las mejora utilizando TS con una cantidad pequeña de iteraciones (que fueron 500), aplica PR y crea un conjunto de soluciones de referencia para después volver a utilizar TS con un número mucho menor de iteraciones para encontrar un óptimo local de cada solución del conjunto de referencia, al final selecciona la mejor solución y aplica TS con un gran número de iteraciones (en este caso 12,500) en la cual se llega a la condición de terminación del algoritmo. Para evaluar el algoritmo se utiliza un conjunto de 205 instancias de diferentes fuentes. La experimentación se realizó utilizando una PC con procesador AMD Athlon Quad-Core (3.0 GHz) con 2 GB de RAM y sistema operativo Windows 7.

En 2016 fue propuesto un algoritmo híbrido Fast Simulated Annealing con Quenching (HFSAQ) [28]. Este algoritmo utiliza la función de densidad de probabilidad de Cauchy's con el objetivo de no quedar atrapado en óptimos locales. Si después de un número de iteraciones no se logra mejora en la calidad de la solución se introduce un ciclo de quenching (reducción rápido de temperatura y aumenta el número de iteraciones), cuando termina este ciclo se restablecen los parámetros al valor que tenían anteriormente. También incorpora una lista tabú para no realizar movimientos repetidos y de esta manera reducir el tiempo de procesamiento. Este algoritmo fue evaluado utilizando 88 instancias y obtuvo un excelente desempeño para instancias de Fisher-Thompson [3], Lawrence [29] y Applegate [11].

1.9. Estado del arte para JSSP Multi-Objetivo

JSSP mono-objetivo es un problema que ha sido investigado de forma amplia a lo largo de los años, sin embargo en el ámbito multi-objetivo existe una escasa cantidad de trabajos publicados.

En [30] se utilizan dos objetivos, *makespan* y *mean tardiness*, emplea un algoritmo Beam Search el cual es una adaptación del método Branch and Bound. Los resultados obtenidos muestran los porcentajes de error comparando cada objetivo por separado con los resultados obtenidos utilizando reglas de prioridad tanto para makespan como mean tardiness.

En [31] se utiliza un algoritmo híbrido genético con una búsqueda local. Se experimenta con 28 instancias de la literatura, en los resultados que obtiene muestra los valores para el *makespan*, *total tardiness* y *total idle time* así como su correspondiente *fitness*, el cual es obtenido asignándole un peso a cada objetivo, el cual es calculado de forma aleatoria.

En [32] se trabaja con dos objetivos, *makespan* y *mean flow time*, utiliza una metaheurística llamada Pareto Archived Simulated Annealing (PASA), el cual está basado en recocido simulado y utiliza una estrategia de recalentamiento para mejorar su búsqueda. El desempeño del algoritmo propuesto es evaluado con 82 instancias del estado del arte y se reportan los frentes de soluciones no dominadas encontrados para cada instancia utilizada.

En [33] se propone un algoritmo genético de dos etapas (2S-GA), en la primer etapa se aplica algoritmo genético para encontrar la mejor solución para cada función objetivo y en la segunda etapa las poblaciones son combinadas. Trabaja minimizando el *makespan*, *total weighted earliness* y *total weighted tardiness*. Experimenta con diversos valores del factor de estrechez para el *due date* y en los resultados muestra una comparación con dos publicaciones de la literatura.

En [23] se propone el *multi-objective evolutionary algorithm/ based on decomposition* (IMOEA/D)

para minimizar el *makespan*, *total flow time* y *total tardiness time*. Experimenta con 58 instancias *benchmark*. Utiliza dos métricas *Coverage metric* [34] y *Mean Ideal Distance* (MID) [35] para evaluar el desempeño del algoritmo propuesto.

En [36] se propone un algoritmo llamado *An artificial bee colony approach for multi-objective job shop scheduling* el cual es codificado en Matlab 2009 y utiliza las instancias de Taillard [12]. Los objetivos con los que trabaja son minimizar *makespan* y *total tardiness*.

En [37] se propone un algoritmo *Hybrid GA/local search/linear programming approach* trabajando con la instancia FT10 de [12], trata de minimizar el *weighted tardiness* y el costo de energía. Para evaluar el desempeño utiliza únicamente la métrica *hypervolume*.

En 2019 es propuesto MOMARLA, un algoritmo basado en Q-Learning [38]. Este trabajo proporciona flexibilidad para utilizar las preferencias de quienes toman las decisiones. Cada agente del algoritmo representa un objetivo específico y utiliza dos estrategias de selección de acciones para encontrar un frente de Pareto diverso y preciso.

En la Tabla 1.1, se presenta un resumen de estos trabajos relacionados para MOJSSP y los algoritmos propuestos para resolverlo.

Tabla 1.1: *Trabajos Relaccionados*

Algoritmo	Objetivos	Métricas
SA [39]	Makespan	*
SA y TA [40]	Makespan	*
Hybrid GA [31]	Makespan, total tardiness, y total idle time	*
PASA [32]	Makespan, mean flow time	*
2S-GA [33]	Makespan, total weighted earliness, y total weighted tardiness	*
IMOEA/D [23]	Makespan, total flow time, y tardiness time	C, MID
Hybrid GA/LS/LP [37]	Weighted tardiness, y energy costs	HV
MOMARLA [38]	Makespan, total tardiness	HV

* No reportado

En nuestro trabajo se presentan algoritmos que resuelven el problema con los 3 objetivos más utilizados en la literatura, obtienen mejor desempeño que los del estado del arte, reportan el tiempo de procesamiento utilizado y las soluciones no dominadas para el caso de las versiones multi-objetivo.

1.10. Organización de la tesis

Esta tesis esta organizada de la siguiente forma, en el Capítulo 1 se realiza una introducción al problema y el planteamiento formal junto con los objetivos de la investigación, en el Capítulo 2 se detalla el marco teórico del problema, los métodos propuestos para el problema JSSP mono-objetivo se presentan en el Capítulo 3, en el Capítulo 4 se detallan los métodos propuestos para el problema multi-objetivo,

los resultados de las experimentaciones se comparten en el Capítulo 5, así como el análisis de los resultados obtenidos.

CAPÍTULO 2

Marco teórico

En el presente capítulo se abordan los conceptos clave para una mejor comprensión del tema, así como los principales métodos aplicados para optimización de scheduling.

2.1. Problemas Scheduling

En scheduling, el procesamiento de un trabajo en una máquina es denotado como una operación. Existen distintas variantes del problema de scheduling y se relacionan con la secuencia de ordenación de las operaciones, esta secuencia es denominada patrón de flujo. Si todos los trabajos tienen el mismo patrón de flujo, el problema se conoce como *flow shop*, si no existe ninguna restricción de ordenamiento el problema se llama *open shop* y por último, el problema en el cual cada trabajo tiene una ordenación determinada se denomina *job shop*. A continuación se describen a mayor detalle cada uno.

2.1.1. Flow Shop Scheduling Problem (FSSP)

En los problemas de *Flow Shop Scheduling* o flujo regular, todos los trabajos tienen el mismo orden de procesamiento en todas las máquinas. Ninguna pieza visita una misma máquina más de una vez, este es un caso particular del problema Job shop [41].

En general, existe un FSSP cuando todos los puestos de trabajo comparten el mismo orden de procesamiento en todas las máquinas. En este problema, las limitaciones tecnológicas exigen que los trabajos pasen entre las máquinas en el mismo orden. Por lo tanto, hay una orden de procesamiento natural (secuencia) de las máquinas, que se caracterizan por las limitaciones tecnológicas para todos y cada uno de los trabajos en el *flow shop*.

Un problema de este tipo se puede observar en la fabricación de productos estandarizados que requieren un alto volumen de producción, un ejemplo sería un establecimiento en donde preparan hamburguesas.

2.1.2. Open Shop Scheduling Problem (OSSP)

En los problemas *Open Shop Scheduling* o de flujo aleatorio, no existe ninguna restricción en cuanto al orden de uso de las máquinas por las operaciones de cada uno de los trabajos [42].

En un problema *open shop*, un conjunto de n trabajos $J_1, J_2, \dot{J}_n$ tiene que ser procesada en un conjunto de m máquinas $M_1, M_2, \dot{M}_m$. El procesamiento de un trabajo en una máquina es denotado como una operación y la secuencia en la cual las operaciones de los trabajos son procesadas en las máquinas es irrelevante. Es asumido que los tiempos de procesamiento de todas las operaciones son conocidos. Como ya se sabe, cada máquina puede procesar a lo mucho un trabajo a la vez y cada trabajo puede ser procesado a lo mucho en una máquina a la vez.

Los problemas *open shop* se dan frecuentemente en varias situaciones dentro de la industria. Por ejemplo, considere un gran taller automotriz con estaciones de trabajo especializadas. Un coche puede requerir los siguientes trabajos: reemplazar los tubos de escape, alinear las llantas y afinación. Estas tres tareas pueden realizarse en cualquier orden. Sin embargo, dado que el sistema de escape, alineación y afinación están en diferentes estaciones de trabajo, no es posible realizar dos tareas simultáneamente. Otras aplicaciones de los problemas de open shop puede ser centros de control de calidad, fabricación de semiconductores, asignaciones de salón de clase y las comunicaciones por satélite [43] [44] [45].

2.1.3. Job Shop Scheduling Problem (JSSP)

En los problemas de scheduling conocidos como *Job Shop Scheduling Problem* (JSSP) cada trabajo es procesado en un conjunto de máquinas en un orden determinado. Un número *n* de trabajos debe ser procesado una sola vez por *m* máquinas, con un orden determinado y durante un tiempo dado.

Existen una gran variedad de técnicas para la generación de un *schedule*. Por ejemplo en [46] se presentó una heurística denominada *shifting bottleneck* para el problema de retraso (*lateness*) de Job Shop . En este método se descompone el problema de Job Shop en un número de subproblemas de una sola máquina, los cuales se resuelven uno después del otro. Cada máquina está programada de acuerdo con la solución de su subproblema correspondiente. El orden en que se resuelven los problemas de una sola máquina tiene un impacto significativo en la calidad de la solución global y en el tiempo requerido para obtener la solución. El autor afirma que su heurística produce soluciones que están cerca del óptimo.

2.2. JSSP como grafo disyuntivo

JSSP puede ser formalmente representado por medio de un grafo disyuntivo $G = (V; C \cup D)$ [47]. Donde:

- V es un conjunto de nodos que representan operaciones de los trabajos con dos nodos especiales, la fuente y el sumidero. En la Figura 2.1, estos son identificados con un "0" y un "*" respectivamente.
- C es un conjunto de arcos conjuntivos que representan la secuencia tecnológica de las máquinas para cada trabajo.
- D es un conjunto de arcos disyuntivos que representan pares de operaciones que deben ser procesadas en la misma máquina.

En el grafo disyuntivo de 3 trabajos en 3 máquinas mostrado en la Figura 2.1 tomado de [47] cada trabajo tiene varias operaciones que estan numeradas dentro de los círculos. Además, los nodos fuente y resumidero representan operaciones que deben ser agregadas al grafo para identificar el inicio y el final del JSSP. El tiempo de procesamiento para cada operación es el valor escrito encima de todos los nodos a excepción de las operaciones *dummy* ("0" y "*") porque sus tiempos de procesamiento se establecen en cero. Las flechas representan restricciones de precedencia que deben ser satisfechas. Hay dos tipos de flechas:

- Las flechas en una dirección indican que la operación *i* precede a la operación *j*.
- Las flechas en dos direcciones indican que la precedencia se puede representar como *i* precede a *j* o *j* precede a *i*.

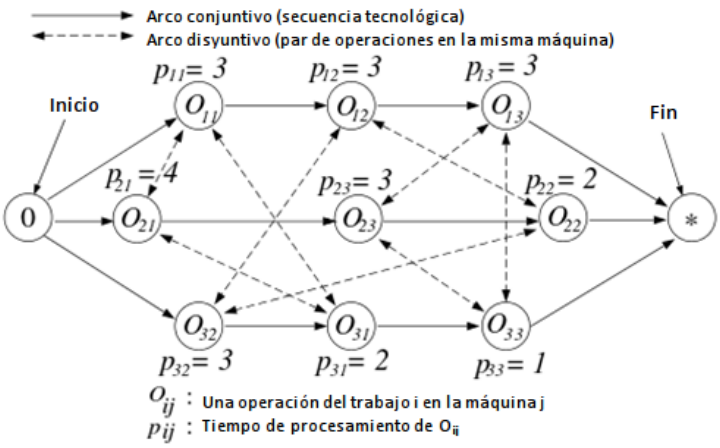

Figura 2.1: *Grafo disyuntivo de un problema de* 3×3

2.3. Algoritmos Heurísticos

El término heurístico deriva de la palabra griega *heuriskein* que significa encontrar o descubrir, y se usa en el ámbito de la optimización para describir una clase de algoritmos de resolución de problemas. En el contexto científico optimización es el procedimiento para encontrar y comparar soluciones factibles hasta que no se pueda encontrar una solución mejor [48].

Existe una infinidad de problemas teóricos y prácticos que involucran a la optimización, algunos de ellos son relativamente fáciles de resolver, sin embargo, muchos otros tipos son muy difíciles. En términos coloquiales podemos decir que un problema de optimización difícil es aquel para el que no podemos garantizar encontrar la mejor solución posible en un tiempo razonable. La existencia de una gran cantidad y variedad de problemas difíciles, que aparecen en la práctica y que necesitan ser resueltos de forma eficiente, impulsó el desarrollo de procedimientos capaces de encontrar buenas soluciones aunque no fueran óptimas.

Estos métodos, en los que la rapidez del proceso es tan importante como la calidad de la solucón obtenida, se denominan heurísticos o aproximados. Un algoritmo heurístico *es un procedimiento para resolver un problema de optimización bien definido mediante una aproximación intuitiva, en la que la*

estructura del problema se utiliza de forma inteligente para obtener una buena solución. [49]

2.4. Algoritmos Metaheurísticos

En años recientes se han producido enormes avances en la disciplina de la optimización metaheurística. Las metaheurísticas (término introducido por Fred Glover en 1986) [50], que *son una clase de métodos aproximados que están diseñados para resolver problemas difíciles de optimización combinatoria en los que los algoritmos heurísticos clásicos no son efectivos. Los metaheurísticos proporcionan un marco general para crear nuevos algoritmos híbridos combinando diferentes conceptos derivados de la inteligencia artificial, la evolución biológica y los mecanismos estadísticos* [51]. Este tipo de algoritmos tienen la capacidad de escapar de los óptimos locales y realizar una búsqueda robusta del espacio de soluciones.

A continuación se describen algunos algoritmos metaheurísticos que están actualmente entre los más empleados para la solución del Job Shop Scheduling Problem (JSSP) [42].

2.4.1. Recocido Simulado

El algoritmo recocido simulado o SA (Simulated Annealing, por sus siglas en inglés) desarrollado por Kirkpatrick [10] está inspirado en una analogía con el proceso de recocido de metales. El cual consiste en el calentamiento y posterior enfriamiento progresivo de un metal de forma 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 de forma paulatina las configuraciones de menor energía. El espacio de configuraciones (posiciones de las moléculas) que en el caso del Job Shop viene determinado por los valores de una variable de interés, normalmente la secuencia de operaciones que se desean procesar, mientras que el papel de la energía lo asume la

función que se intenta minimizar que en este caso es el *makespan*.

Este algoritmo fue propuesto para la optimización de problemas combinatorios con mínimos locales y se describe en seguida su funcionamiento. Se genera una solución inicial que puede ser aleatoria o utilizar alguna metaheurística en particular y explora las soluciones de su vecindad. Aplicando perturbaciones a la solución explora una nueva, la acepta si mejora, o en caso contrario aplica la probabilidad de Boltzmann para permitir que soluciones de mala calidad participen en la búsqueda; ésta probabilidad de aceptación irá disminuyendo conforme aumenta el número de iteraciones.

2.4.2. Scatter Search

El algoritmo Scatter Search (SS por sus siglas en inglés) es un procedimiento metaheurístico poblacional propuesto en [52]. Está basado en formulaciones y estrategias introducidas durante la década de los setenta. A principios de esta década se propusieron los conceptos y principios fundamentales del método, basados en estrategias de combinación de reglas de decisión.

Scatter Search consiste en cinco procedimientos:

- **Método de generación de soluciones diversas**. Genera una población de soluciones diversas. Cada una de estas soluciones se optimiza aplicando un método de mejora. A continuación, se extrae un subconjunto de soluciones. Este subconjunto recibe el nombre de conjunto de referencia (*refSet*).
- **Método de mejora**. Se utiliza para mejorar las soluciones del conjunto de referencia.
- **Método de actualización del conjunto de referencia**. Construye y mantiene el conjunto de referencia. A partir de este conjunto se extraen soluciones de calidad y soluciones diversas.
- **Método de generación de subconjuntos**. Se basa en la combinación de todas las soluciones del

conjunto de referencia.

- **Método de combinación de soluciones**. Combina las soluciones de los subconjuntos generados en una o más soluciones.

2.4.3. Algoritmo Genético

Este algoritmo fue introducido en 1975 por Holland [9]. Imita el procedimiento de la selección natural sobre el espacio de soluciones del problema considerado. Estos algoritmos hacen evolucionar una población de individuos sometiéndola a acciones aleatorias semejantes a las que actúan en la evolución biológica (mutaciones y recombinaciones genéticas), así como también a una selección de acuerdo con algún criterio. En función de este criterio se decide cuáles son los individuos más adaptados, que sobreviven, y cuáles los menos aptos, que son descartados. Por lo regular son efectivos para encontrar más de una solución cercana al óptimo o incluso el óptimo. Los algoritmos genéticos se caracterizan por las siguientes operaciones básicas:

- **Selección**. La función de selección de padres más utilizada es la denominada función de selección proporcional a la función objetivo, en la cual cada individuo tiene una, probabilidad de ser seleccionado como padre que es proporcional al valor de su función objetivo.
- **Cruza**. Los dos individuos seleccionados para jugar el papel de padres, son recombinados por medio de la selección de un punto de corte, para posteriormente intercambiar las secciones que se encuentran a la derecha de dicho punto.
- **Mutación**. Se considera un operador básico, que proporciona un pequeño elemento de aleatoriedad en la vecindad (entorno) de los individuos de la población. Si bien se admite que el operador de cruza es el responsable de efectuar la búsqueda a lo largo del espacio de posibles soluciones, también parece desprenderse de los experimentos efectuados por varios investigadores que el

operador de mutación va ganando en importancia a medida que la población de individuos va convergiendo.

2.5. Sintonización Analítica de Parámetros para el Recocido Simulado

El proceso de sintonización del algoritmo recocido simulado se realiza en base a un método propuesto en [53], el cual establece que la temperatura inicial y la temperatura final están en función del máximo y mínimo valor obtenido de las soluciones de la vecindad. Para este proceso se deben de realizar un conjunto de ejecuciones previas del algoritmo con el objetivo de obtener valores de parámetros que servirán para realizar la sintonización analítica del mismo.

El proceso es detallado a continuación:

Temperatura inicial ($T_{inicial}$). La probabilidad de aceptar una nueva solución a temperaturas altas es casi 1, así que su deterioro es máximo. La temperatura inicial está asociada con el máximo deterioro admitido y su probabilidad de aceptación definida. Si s_i es la solución actual y s_j es una solución nueva propuesta y $Z_{(si)}$ y $Z_{(sj)}$ son sus costos asociados respectivamente, el máximo y mínimo deterioro es expresado como ΔZ_{max} y ΔZ_{min}. Por otro lado, $P(\Delta Z_{max})$, es la probabilidad de aceptar una solución con el máximo deterioro y se calcula con la fórmula (2.5.1). Entonces, el valor de la temperatura inicial se puede calcular con la fórmula (2.5.2).

$$P(\Delta Z_{max}) = e^{(\Delta Z_{max}/T_{inicial})} \tag{2.5.1}$$

$$T_{inicial} = \frac{-\Delta Z_{max}}{\ln(P(\Delta Z_{max}))} \tag{2.5.2}$$

Temperatura Final (T_{final}). De forma similar la temperatura final (T_{final}) es determinada con (2.5.3) de acuerdo a la probabilidad $P(\Delta Z_{min})$, que es la probabilidad de aceptar una solución con el mínimo deterioro.

$$T_{final} = \frac{-\Delta Z_{min}}{\ln(P(\Delta Z_{min}))} \tag{2.5.3}$$

Alfa (α). Este parámetro determina la velocidad a la cual decrece la temperatura, para decrementos rápidos usualmente es usado 0.7 y para decrementos lentos se usa 0.99.

Esquema de enfriamiento. El valor de la temperatura actual (T_k) es decrementado en base al esquema geométrico mostrado en la ecuación 2.5.4).

$$T_{k+1} = \alpha\, T_k \tag{2.5.4}$$

Beta (β). Este parámetro determina la velocidad a la cual se incrementa L_k.

Iteraciones del ciclo de Metrópolis (L_k). Este valor puede ser constante o puede ser variable. Es el número de iteraciones en el ciclo interno o ciclo de Metrópolis a la temperatura actual (T_k). En [53] es indicado que a altas temperaturas únicamente unas pocas iteraciones se requieren para alcanzar el equilibrio estocástico, sin embargo, a bajas temperatura un nivel de exploración más exhaustivo es requerido, entonces un valor de L_k más grande debe ser usado. Si L_{min} es el valor de L_k a la temperatura inicial y L_{max} es el valor de L_k a la temperatura final, entonces la fórmula (2.5.5) is deducida.

$$L_{k+1} = \beta\, L_k \tag{2.5.5}$$

Como las funciones (2.5.4) y (2.5.5) se aplican sucesivamente en el recocido simulado desde la temperatura inicial hasta la temperatura final, entonces la temperatura final (T_{final}) y L_{max} pueden ser calculados como se muestra en (2.5.6) y (2.5.7) respectivamente.

$$T_{final} = \alpha^n \, T_{inicial} \tag{2.5.6}$$

$$L_{max} = \beta^n \, L_{min} \tag{2.5.7}$$

En las ecuaciones (2.5.6) and (2.5.7) anteriores n es el número de temperaturas que existen desde la $T_{inicial}$ hasta la T_{final}, entonces claramente se puede calcular con (2.5.8) y el valor de β con la ecuación (2.5.9).

$$n = \frac{\ln(T_{final}) - \ln(T_{initial})}{\ln(\alpha)} \tag{2.5.8}$$

$$\beta = e^{(\frac{\ln(L_{max}) - \ln(L_{min})}{n})} \tag{2.5.9}$$

La probabilidad de seleccionar la solución s_j de muestras aleatorias en la vecindad Vsi está dada por (2.5.10).

$$P(S_j) = 1 - e^{\frac{-N}{|V_{si}|}} \tag{2.5.10}$$

De la ecuación (2.5.10) el valor de N puede ser obtenida con (2.5.11)

$$N = - \mid V_{si} \mid \ln(1 - P(S_j)) = C \mid V_{si} \mid \tag{2.5.11}$$

En (2.5.11) C indica el nivel de exploración y es obtenido con (2.5.12).

$$C = \ln(P(S_j)) \tag{2.5.12}$$

Y finalmente la longitud de la cadena de Markov o iteraciones en el ciclo de Metrópolis puede ser calculado con (2.5.13).

$$L_{max} = N = C \mid V_{si} \mid \tag{2.5.13}$$

En [54] se indica que para garantizar un buen nivel de exploración en la vecindad el valor de C debe ser establecido entre $1 \leq C \leq 4.6$.

2.6. Perturbación caótica

La ecuación logística o mapa logístico es una aplicación matemática bien conocida debido a que el biólogo Robert May [55], trató de encontrar un modelo demográfico simple que explicara la dinámica de una población que se suponía que tenía crecimiento cada vez más lento a medida que se acerca a un número de personas consideradas como un límite. Esta aplicación nos dice la población en la enésima

generación basada en la población de la generación anterior. May descubrió que cambiar los valores del parámetro inical del modelo presentaba soluciones muy diferentes y, a veces, muy complejas. El mapa logístico se puede expresar matemáticamente como:

$$x_{n+1} = rx_n(1 - x_n) \tag{2.6.1}$$

Donde n es un número entre cero y uno que representa la fracción de individuos en un territorio en un instante n. El parámetro r es un número positivo que representa la relación entre la reproducción y la mortalidad de la población.

La función caótica utilizada es muy sensible a los cambios en las condiciones iniciales, y esta característica se utiliza para generar una perturbación a la solución actual para tratar de escapar del estancamiento. Entonces, esto significa que, el caos o la perturbación caótica es un proceso llevado a cabo para reiniciar la búsqueda desde otro punto en el espacio de soluciones.

2.7. Optimización Multi-Objetivo

2.7.1. Problema de Optimización Multi-Objetivo

En un problema de optimización con un solo objetivo el algoritmo termina su ejecución cuando encuentra la solución que optimiza la función objetivo, pero en un problema de optimización multi-objetivo es un poco más complejo ya que se deben optimizar varios objetivos de forma simultánea. Entonces se deben encontrar un conjunto de diferentes soluciones que optimizan cada uno de los objetivos de forma individual, este conjunto es llamado *conjunto no dominado*. Al compararse dos soluciones del conjunto no dominado una de ellas podría ser mejor en base a un objetivo y peor respecto

a otro. A continuación se describen algunos conceptos de optimización multi-objetivo.

2.7.2. Conceptos de Optimización Multi-Objetivo

Dominancia. Para un problema de minimización multi-ojetivo, se dice que la solución A domina la solución B si se cumplen las siguientes condiciones [48]:

- A no es peor que B en todos los objetivos.
- A es estrictamente mejor que B en al menos un objetivo.

En general, se dice que para que una solución domine a otra, debe ser mejor en al menos un objetivo y no peor en ninguno de ellos.

Conjunto no dominado. En un conjunto de soluciones P, el conjunto de soluciones no dominadas P_1 son aquellas que no están dominadas por ningún miembro del conjunto P. Las condiciones que debe cumplir un conjunto no dominado (P_1) son [48]

- Cualquier par de soluciones en P_1 no debe dominarse entre sí.
- Cualquier solución que no pertenezca a P_1 está dominada por al menos un miembro de P_1.

Conjunto óptimo de Pareto. Es el conjunto de soluciones no dominadas del espacio de búsqueda total [48].

Frente de Pareto. Es la representación gráfica de las soluciones no dominadas en el espacio de los objetivos del problema de optimización multi-objetivo [56].

El objetivo en un problema de optimización multi-objetivo, es obtener el conjunto de soluciones no dominadas, cuando este conjunto es obtenido, ciertos procedimientos deben aplicarse para determinar la calidad de las soluciones generadas, para eso se utilizan las métricas de desempeño que se detallan en la siguiente sección.

2.7.3. Métricas de desempeño

Cuando se realiza una comparación experimental de diferentes técnicas o algoritmos de optimización, siempre es necesario tener la noción de rendimiento. En el caso de la optimización multi-objetivo, la definición de calidad es mucho más compleja que para los problemas de optimización de un solo objetivo, porque el objetivo de optimización en sí consiste en múltiples objetivos, por ejemplo:

- La distancia del frente de Pareto calculado al frente de Pareto real debe minimizarse.
- También es deseable una buena distribución de las soluciones encontradas, en la mayoría de los casos esta debe de ser uniforme.
- La extensión del frente no dominado obtenido debe ser maximizada, es decir, para cada objetivo, un amplio rango de valores debe ser cubierto por las soluciones no dominadas.

En general, es difícil encontrar una única métrica de rendimiento que abarque todos los criterios anteriores. En la literatura, se puede encontrar una gran cantidad de métricas. A continuación se describen algunas de las más utilizadas.

Mean Ideal Distance (MID). Evalúa la cercanía de las soluciones del frente de Pareto calculado (FP_{calc}) con un punto ideal (0,0,0) [35].

$$MID = \frac{\sum_{i=1}^{Q} c_i}{Q} \tag{2.7.1}$$

Donde $c_i = \sqrt{f_{1,i}^2 + f_{2,i}^2 + f_{3,i}^2}$, y $f_{1,i}, f_{2,i}, f_{3,i}$ son los valores de la *i*ésima solución no dominada para su primer, segunda y tercer función objetivo y Q es el número de soluciones en el FP_{calc}.

Spacing (S). Evalúa la distribución de las soluciones no dominadas en el frente de Pareto calculado (FP_{calc}). Un algoritmo que encuentra un conjunto no dominado con un valor de S pequeño es mejor y cuando $S = 0$ significa que todas las soluciones están separadas de forma uniforme [57].

$$S = \sqrt{\frac{\sum_{i=1}^{Q}(d_i - \bar{d})^2}{Q}} \tag{2.7.2}$$

Donde d_i mide la distancia en el espacio de las funciones objetivo entre la *i-ésima* solución y su vecino más próximo es decir la *j-ésima* solución en el FP_{calc} por el algoritmo, $\bar{d}$ es el promedio de los d_i, es decir $\bar{d} = \sum_{i=1}^{Q} \frac{d_i}{Q}$.

$$d_i = min_j(|f_1^i(x) - f_1^j(x)| + |f_2^i(x) - f_2^j(x)| + ... + |f_M^i(x) - f_M^j(x)|), i = 1, ...Q. \tag{2.7.3}$$

Spread (Δ). La métrica spacing evalúa la distribución de puntos en el Frente de Pareto calculado (FP_{calc}), pero puede no dar una idea certera de la cobertura de las soluciones, ya que no utiliza información adicional sobre el frente de Pareto real (FP_{real}). La métrica spread, presentada en [58], propone utilizar como información adicional la distancia a los "extremos" (puntos con menores valores en cada una de las funciones objetivo) del FP_{real} para tener una medida más precisa de la cobertura del frente.

Tomando en cuenta los "extremos" del FP_{real} se evita el problema comentado de la métrica spacing, que podría dar buenos valores de distribución, aún cuando en realidad no se estuviera muestreando adecuadamente el frente real.

$$\Delta = \frac{\sum_{k=1}^{M} d_k^e + \sum_{i=1}^{Q} |d_i - \bar{d}|}{\sum_{k=1}^{M} d_k^e + Q * \bar{d}} \tag{2.7.4}$$

Donde el término d_k^e mide la distancia entre el punto "extremo" del frente de Pareto real, tomando en cuenta la *k-ésima* función objetivo, y el punto del FP_{calc} más cercano. El término d_i corresponde a una medida de la distancia, en el espacio de las funciones objetivo, entre la *i-ésima* solución del FP_{calc} y su vecino más cercano. El valor $\bar{d}$ corresponde al promedio de los d_i y M es el número de objetivos.

Hipervolume (HV). Calcula el volumen en el espacio de los objetivo que es cubierto por todas las soluciones del conjunto no dominado [59]. El hipervolume de un conjunto no dominado se mide en función de un punto de referencia, una de las formas de obtener este punto de referencia es construyendo un vector con los peores valores de la función objetivo.

$$HV = volume\left(\cup_{i=1}^{|Q|} v_i\right) \tag{2.7.5}$$

En general, un algoritmo que obtiene un valor de HV más grande es mejor.

Distancia Generacional (DG). Calcula la distancia promedio entre los puntos del FP_{calc} y el FP_{real} [60]. Si $T = \{t_1, t_2, ..., t_{|T|}\}$ es decir (FP_{real}) será un conjunto de puntos de referencia dado donde $|T|$ es la cardinalidad de T y para un conjunto de vectores objetivos no dominados $Q = \{q_1, q_2, ..., q_{|Q|}\}$ o FP_{calc} y el punto de referencia $T = \{t_1, t_2, ..., t_{|T|}\}$ la DG se puede definir formalmente de la siguiente manera:

$$DG(Q) = \frac{\left(\sum_{i=1}^{|Q|} d_i^p\right))^{1/p}}{|Q|} \tag{2.7.6}$$

Donde d_i es la distancia euclidiana desde q_i a su punto de referencia más cercano en T, en otras palabras

es la distancia euclidiana entre cada elemento i del FP_{calc} y el elemento más cercano del FP_{real} y p es un parámetro entero. En este documento, se especifica p como $p = 2$ para DG e DGI. Cuando DG = 0, significa que $FP_{calc} = PFreal$.

$$d_i = \min_{t=1}^{|T|} \sqrt{\sum_{m=1}^{M} (fm(q_i) - fm(t))^2} \tag{2.7.7}$$

Donde $fm(t)$ es el m-th valor de la función objetivo de los t-th mmiembros de T y M es el número de objetivos.

Distancia Generacional Invertida (DGI). Es una indicador invertido de la DG, propuesta por [61], en la que las distancias se miden desde el FP_{real} hasta el FP_{calc} por el algoritmo. El objetivo es reducir el principal problema de esta métrica en el caso en el que, por ejemplo, FP_{calc} tiene muy pocos puntos, y todos están agrupados.

$$DGI(Q) = \frac{\left(\sum_{j=1}^{|T|} \hat{d}_j^p\right)^{1/p}}{|T|} \tag{2.7.8}$$

Donde $\hat{d}_j$ es la distancia euclidiana desde t_j a su punto vector objetivo más cercano en Q, es decir:

$$d_j = \min_{q=1}^{|Q|} \sqrt{\sum_{m=1}^{M} (fm(t_j) - fm(q))^2} \tag{2.7.9}$$

Coverage (C). Representa el dominio que existe entre el conjunto no dominado A y el conjunto B [34]. Es la relación entre el número de soluciones en el conjunto B dominado por las soluciones en el conjunto A y el número total de soluciones en el conjunto B. Se puede definir como:

$$C(A,B) = \frac{|\{b \in B | \exists a \in A : a \preceq b\}|}{|B|} \tag{2.7.10}$$

El valor $C(A,B) = 1$ significa que todas las soluciones en B están dominadas o son iguales a las soluciones en A. De lo contrario, $C(A,B) = 0$, representa situaciones en las que ninguna de las soluciones en B está dominada por solución en A. Cuanto mayor sea el valor de $C(A,B)$, significa que más soluciones en B están dominadas por soluciones en A. Tanto $C(A,B)$ como $C(B,A)$ deben considerarse, ya que $C(B,A)$ no es necesariamente igual a $1 - C(A,B)$.

2.8. Algoritmos Multi-Objetivo

A continuación se describen algunos algoritmos multi-objetivo que se han aplicado al problema JSSP.

2.8.1. Recocido Simulado Multi-Objetivo (MOSA)

La primera adaptación de recocido simulado (SA) para resolver un problema con más de un objetivo, es decir a la optimización multi-objetivo fue propuesta en [39]. Esta primera adaptación consiste en el ajuste del criterio de aceptación para combinar todas las funciones objetivo utilizando funciones agregativas.

En general, la principal diferencia entre SA y MOSA está en el cálculo de la función de energía y el uso de un conjunto de soluciones que conforman el frente de Pareto. En los últimos años se han propuesto múltiples y diversas versiones de MOSA aplicada a diferentes problemas; uno de ellos es AMOSA propuesta en 2008 [62].

2.8.2. Threshold Accepting Multi-Objetivo (MOTA)

En [63] es propuesto un nuevo algoritmo de propósito general para la solución de problemas de optimización combinatoria. Este método llamado Threshold Accepting (TA) está basado en SA pero tiene una estructura aún más simple.

La diferencia entre SA y TA consiste en las diferentes reglas de aceptación que se utilizan; mientras que TA acepta cada nueva configuración que no es mucho peor que la anterior, SA acepta soluciones peores solo con probabilidades bastante pequeñas. Una ventaja aparente de TA es su mayor simplicidad, ya que no es necesario calcular probabilidades o tomar decisiones aleatorias.

2.9. Tecnologías para Cómputo Paralelo

Como se ha mencionado en la sección 1.5 los problemas JSSP pertenecen a la categoría de problemas de alta complejidad para los cuales resolver instancias provenientes de los entornos de manufactura reales conllevan altos costos en tiempo de procesamiento por lo que obtener soluciones en tiempos razonables exige el uso del cómputo intensivo. Esta es la razón principal de hacer uso de tecnologías de programación paralela. Para cada uno de los algoritmos implementados se analizan diferentes técnicas de programación paralela con la finalidad de seleccionar la más adecuada para resolver de forma eficiente JSSP.

Entre las tecnologías de programación para cómputo paralelo que se pueden aplicar se encuentran: OpenMP, MPI y CUDA. También se podría realizar algún método híbrido OpenMP/MPI/CUDA donde su diseño depende del tipo de algoritmos a desarrollar [64]. A continuación se hace una breve descripción de cada una de ellas.

2.9.1. OpenMP

OpenMP (Open Multi-Processing) es una API (Interfaz de Programación de Aplicaciones) que puede utilizarse para obtener de manera explícita programación paralela multi-hilo; esto es para obtener paralelismo en arquitecturas de memoria compartida [65].

Un sistema de *memoria compartida* tiene múltiples CPUs (Unidad Central de Proceso) donde todos comparten el mismo espacio de direcciones de memoria. Esto significa que el conocimiento de donde están almacenados los datos no es de interés para el usuario ya que solo se tiene acceso a una memoria por todos los CPUs en igualdad de condiciones. En la Figura 2.2 se muestra el diagrama de un sistema de memoria compartida [66].

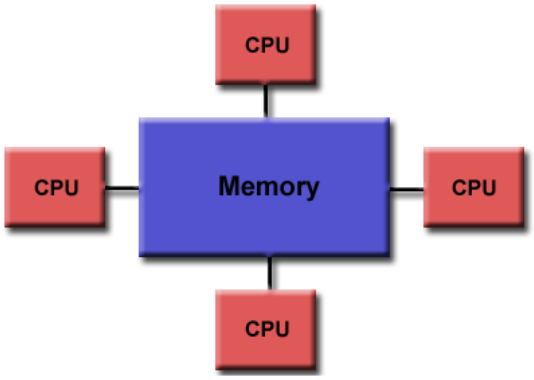

Figura 2.2: *Sistema de memoria compartida*

Los lenguajes base con los que trabaja OpenMP son Fortran, C, y C++. Actualmente es una de las herramientas más extendida en los sistemas de memoria compartida. OpenMP se basa en el modelo *fork-join* utilizado en los sistemas Unix, el cual consiste en dividir una tarea muy pesada en k hilos (*fork*) con menor peso, al final se ”recolectan” sus resultados y se unen en uno solo (*join*).

OpenMP cuenta con tres componentes principales:

1. **Directivas del compilador**. Estas directivas nos ayudan para la identificación de una región

paralela, la división de bloques de código entre los hilos, la distribución de las iteraciones de bucle entre hilos, la serialización de secciones de código y sincronización de trabajos.

2. **Rutinas de librerías**. Entre algunas funciones estas nos ayudan a establecer el número de hilos a usar, obtener el número de hilos con los que trabaja el programa, obtener la máxima cantidad posible de hilos a utilizar, devolver el número de hilo actual, devolver el máximo número de procesadores que se pueden usar en el programa.

3. **Variables de entorno**. Son un conjunto de valores dinámicos que afectan el comportamiento de los procesos en una computadora. Entre otras cosas estas variables nos ayudan a indicar el tipo de scheduling en un *for* y *pararallel for*, autorizar o desautorizar el ajuste dinámico del número de *threads* (hilos) y establecer el número de *threads* a usar (este valor también se puede cambiar con la función de librería).

2.9.2. MPI

MPI es la abreviación de *Message Passing Interface*, posiblemente es la herramienta de programación paralela más empleada en supercomputación actualmente. Se basa en el modelo fragmentado, es decir en el que varias instancias del programa se ejecutan de forma simultánea, comunicándose entre ellas mediante llamadas a la librería MPI. Estas rutinas soportan varios paradigmas de comunicación, incluyendo comunicación bidireccional.

La interfaz de MPI soporta C y Fortran, y MPI-2 permite además de esos lenguajes, Python, Java y está evolucionando para incluir más lenguajes. Una de las críticas que recibe MPI es que se requiere un tiempo mucho mayor para desarrollar aplicaciones paralelas que en el modelo OpenMP. Entre los sistemas de memoria distribuida es actualmente la herramienta más extendida. Un *sistema de memoria distribuida*, es aquel en el que cada CPU posee su propia memoria asociada. Los CPUs están conectados por una red y pueden intercambiar datos entre sus respectivas memorias si es necesario. En la Figura 2.3 podemos ver un sistema de memoria distribuida [67].

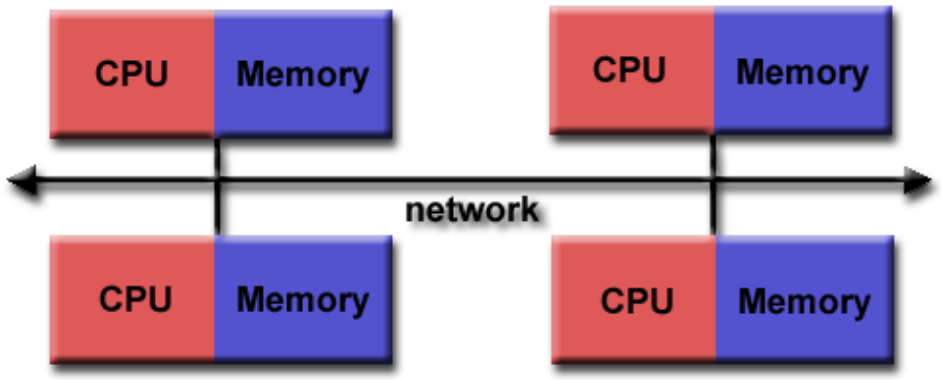

Figura 2.3: *Sistema de memoria distribuida*

2.9.3. CUDA

CUDA (Compute Unified Device Architecture) que en español significa Arquitectura Unificada de Dispositivos de Cómputo, es una plataforma de computación paralela y un modelo de programación inventado por la empresa NVIDIA. Permite aumentos impresionantes en el rendimiento de la computación al aprovechar la potencia de la unidad de procesamiento de gráficos (GPU). Recordar que el GPU, es un procesador especializado que está dirigido a las tareas de cómputo intensivo que demanda los gráficos de tiempo real en alta resolución o en 3D.

El uso de CUDA permite el cómputo GPU o GPGPU (cómputos de propósito general en unidades de procesamiento de gráficos) que es la utilización de una GPU para realizar cómputos en aplicaciones de ingeniería, medios digitales y científicos tradicionalmente manejados por una unidad de procesamiento central (CPU). La GPU actúa como un coprocesador y puede acelerar las aplicaciones gracias a su enorme potencia de procesamiento paralelo, esto en comparación con el diseño de núcleo múltiple de las CPU.

A partir del año 2012, las GPU han evolucionado hasta convertirse en sistemas multinúcleo altamente paralelos que permiten la manipulación muy eficiente de grandes bloques de datos. Este diseño es más eficaz que las CPU de propósito general para algoritmos en situaciones donde el procesamiento de

grandes bloques de datos se realiza de forma paralela.

En la imagen 2.4 se puede observar la diferencia en cuanto a cantidad de núcleos en un CPU tradicional y un GPU [68].

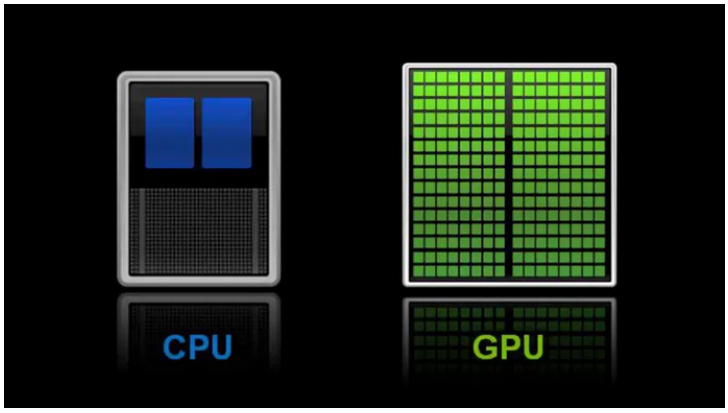

Figura 2.4: *Cantidad de núcleos de un CPU y un GPU*

Los desarrolladores de software, científicos e investigadores cada vez encuentran usos más amplios para la computación de la GPU utilizando CUDA.

Entra otras, algunas de las ventajas de CUDA se enumeran a continuación:

- Ofrece la explotación del paralelismo mediante múltiples núcleos en las GPU lo que se traduce en el lanzamiento de un gran número de hilos simultáneamente.
- Es realmente eficiente en el análisis y simulación de problemas muy difíciles como en el caso de sistemas de muchas partículas, y en general, en sistemas complejos, donde el número de agentes interactuando junto con un número inmenso de interacciones.

De acuerdo con la literatura especializada, se espera que la explotación del paralelismo que ofrecen los múltiples núcleos en las GPU y posiblemente otras nuevas arquitecturas lleven a nuevos lanzamientos y provoquen grandes aplicaciones de alto impacto no solo en videojuegos sino en todo el quehacer humano [68].

CAPÍTULO 3

Métodos propuestos para Job Shop Scheduling Problem Mono-Objetivo

En esta sección se describe la forma en que se representa una solución al problema, el algoritmo recocido simulado con la sintonización de parámetros, la versión en la que se aplica el equilibrio estocástico, el algoritmo en que se aplica el recalentamiento (*re-heat*), el algoritmo en que se aplica la perturbación caótica y las versiones ejecutadas en cómputo paralelo, todas aplicadas al problema mono-objetivo.

Para la evaluación experimental fueron seleccionadas 25 instancias de diferentes dimensiones, todas con diferente grado de complejidad. 3 (ft6, ft10, ft20) de [3], 4 (yn1, yn2, yn3, yn4) de [69], 8 (ta01-ta71) de [12], 5 (orb1-orb5) de [11] y 5 (la01, la06, la16, la21, la26) de [29].

Se emplea lenguaje C para la implementación de todos los algoritmos propuestos. La ejecución se

realiza en una de las terminales del clúster Ehécatl del Instituto Tecnológico de Ciudad Madero, la cual tiene las siguientes características: Procesador Intel® Xeon® a 2.30 GHz, Memoria: 64gb (4x16gb) ddr4-2133, sistema operativo Linux CentOS.

Cada instancia se ejecutó un número determinado de tiempo. Este tiempo depende de la cantidad de tareas u operaciones que tenga la instancia a procesar, a mayor número de tareas mayor tiempo de procesamiento se le asigna. Para evaluar el desempeño del algoritmo se midieron el valor del makespan y el tiempo de ejecución promedio para cada instancia.

3.1. Representación computacional de la solución

En todas las implementaciones realizadas la forma de representar una solución es por medio de una matriz de dos dimensiones que contiene las coordenadas de cada una de las operaciones de las que está conformada la instancia a resolver. Por ejemplo, la instancia *mt06* o *ft06* que está conformada por 6 trabajos a procesarse en 6 máquinas, tiene 36 operaciones en total (Tabla 3.1). En esta tabla se tienen dos diferentes tipos de datos, la columna M que indica la máquina en la que se deberá procesar el trabajo y de la columna TPO_1 a TPO_6 contiene el tiempo de procesamiento (número entre paréntesis) que se necesita para procesar cada operación en cada máquina. La última columna indica el número de trabajo.

Tabla 3.1: *Instancia mt06*

M	TPO_1	M	TPO_2	M	TPO_3	M	TPO_4	M	TPO_5	M	TPO_6	No. de trabajo
3	(1)	1	(3)	2	(6)	4	(7)	6	(3)	5	(6)	Trabajo 1
2	(8)	3	(5)	5	(10)	6	(10)	1	(10)	4	(4)	Trabajo 2
3	(5)	4	(4)	6	(8)	1	(9)	2	(1)	5	(7)	Trabajo 3
2	(5)	1	(5)	3	(5)	4	(3)	5	(8)	6	(9)	Trabajo 4
3	(9)	2	(3)	5	(5)	6	(4)	1	(3)	4	(1)	Trabajo 5
2	(3)	4	(3)	6	(9)	1	(10)	5	(4)	3	(1)	Tranajo 6

Las 36 operaciones de esta instancia están representadas en una matriz de tamaño 6x6 como la que se muestra en la Tabla 3.2.

Tabla 3.2: *Coordenadas para operaciones de la instancia mt06*

(0,0)	(0,1)	(0,2)	(0,3)	(0,4)	(0,5)
(1,0)	(1,1)	(1,2)	(1,3)	(1,4)	(1,5)
(2,0)	(2,1)	(2,2)	(2,3)	(2,4)	(2,5)
(3,0)	(3,1)	(3,2)	(3,3)	(3,4)	(3,5)
(4,0)	(4,1)	(4,2)	(4,3)	(4,4)	(4,5)
(5,0)	(5,1)	(5,2)	(5,3)	(5,4)	(5,5)

La solución para esta instancia se representa en una matriz de tamaño 2 x 36, en la que cada par de celdas formada por el valor de cada fila y cada columna guarda las coordenadas de cada una de las operaciones que tiene la instancia (ver Tabla 3.2).

La Tabla 3.3 representa una solución de la instancia mt06.

Tabla 3.3: *Representación de una solución a la instancia mt06*

Operación	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
Fila	1	3	0	0	5	2	1	4	0	2	2	5	4	3	3	5	1	2
Columna	0	0	0	1	0	0	1	0	2	1	2	1	1	1	2	2	2	3

19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36
4	1	3	5	2	3	0	5	1	0	5	1	2	4	3	4	0	4
2	3	3	3	4	4	3	4	4	4	5	5	5	3	5	4	5	5

A lo largo de los años, en las investigaciones realizadas con el problema JSSP mono-objetivo se han implementado varios algoritmos por diversos investigadores. A continuación se enumeran y describen los algoritmos utilizados en la presente investigación.

3.2. Recocido Simulado con sintonización manual de parámetros

Recordar que los principales parámetros del algoritmo recocido simulado son:

- ***Temperatura inicial*** $(T_{inicial})$. Es el valor de temperatura a partir de la cual se inicia la búsqueda.

- ***Temperatura final*** (T_{final}). Es el valor de la temperatura en la cual se detiene la búsqueda. Es decir es el criterio de paro del algoritmo.
- ***Alfa*** (α). Es el factor de decremento de la temperatura.
- ***Esquema de enfriamiento***. Es la función que especifica cómo se realiza el decremento de la temperatura, en este caso se usa el esquema geométrico $T_{k+1} = \alpha\, T_k$.
- ***Ciclo de Metrópolis*** (L_k). Es la cantidad de iteraciones que se realizan a cada valor de temperatura dentro del ciclo de Metrópolis.
- ***Beta*** (β). Es el factor de incremento de iteraciones en el ciclo de Metrópolis.

El Algoritmo 1 muestra el algoritmo recocido simulado. La línea 1 recibe los parámetros mencionado anteriormente temperatura inicial, temperatura final, alfa, beta y longitud del ciclo de Metrópolis. En la línea 2 la temperatura actual (T_k) es establecida. En la línea 3 es generada una solución inicial de forma aleatoria (s_i). En la línea 4 se evalúa el criterio de paro, es decir mientras la temperatura actual sea mayor o igual que la temperatura final se realiza este ciclo. En la línea 5 inicia el ciclo de Metrópolis, este ciclo se repite desde 1 hasta el valor de L_k. Dentro de este ciclo se genera una solución vecina (s_j) a la solución actual (s_i) (línea 6), esta se genera mediante una perturbación a la solución s_i. Esta perturbación consiste en intercambiar de posición dos operaciones en la solución actual hasta lograr una solución factible, posteriormente se evalúa si es mejor que la anterior (línea 7 y 8), si es mejor se utiliza para reemplazar a la solución actual y continuar la búsqueda (línea 9) en caso contrario se aplica la probabilidad de Bolztmann (línea 11 y 12) para determinar si se utiliza aunque sea una solución de peor calidad que la anterior. Para finalizar, en la línea 16 se aumenta el valor del número de iteraciones para el ciclo de metrópolis y en la 17 se aplica la función de enfriamiento para reducir la temperatura actual. La salida del algoritmo es la solución almacenada en s_i que es la que tiene un valor de makespan menor.

Con el término *sintonización manual* nos referimos al proceso de establecer el valor de los parámetros del algoritmo a prueba y error, es decir, establecer valores, realizar ejecuciones y analizar los resultados

Algorithm 1 Recocido Simulado

```
1:  procedure RECOCIDOSIMULADO(T_inicial, T_final, α, β, L_k)
2:      T_k ← T_inicial
3:      s_i ← SolucionInicialAleatoria
4:      while T_k ≥ T_final do
5:          for 1 to L_k do
6:              s_j ← perturbacion(s_i)
7:              Δ ← E(s_j) − E(s_i)
8:              if Δ < 0 then
9:                  s_i ← s_j
10:             else
11:                 if (e^{−Δ/Tk}) > aleatorio(0,1) then
12:                     s_i ← s_j
13:                 end if
14:             end if
15:         end for
16:         L_{k+1} ← β × L_k
17:         T_{k+1} ← α × T_k
18:     end while
19:     return s_i
20: end procedure
```

hasta que se consigan obtener resultados de buena calidad. Este es un proceso que requiere mucho tiempo ya que se tienen que realizar varias ejecuciones para cada una de las instancias hasta lograr los resultados que cumplan nuestras expectativas.

Los parámetros encontrados mediante el proceso antes citado que mejores resultados obtuvieron se muestran en la Tabla 3.4.

Tabla 3.4: *Parámetros utilizados en la sintonización manual*

Parámetro	Temperatura inicial	Temperatura final	α	L_k	β
Valor	10,000	0.01	0.98	No. de tareas	1

Como se puede ver, el valor de L_k depende del tamaño de la instancia a resolver, este parámetro es 36 para la instancia más pequeña y 2000 para la más grande.

Para medir el desempeño de los algoritmos mono-objetivos se calcula el Error Relativo (ER) porcentual del makespan (MKS) o makespan promedio respecto a la Mejor Solución Conocida (MSC) para cada

instancia de la siguiente forma:

$$ER = \frac{|\, MKS - MSC \,|}{MSC} \times 100 \tag{3.2.1}$$

Resultados Obtenidos con sintonización manual

Este algoritmo es ejecutado 30 veces, en la Tabla 3.5 se muestra el nombre de la instancia, su tamaño y la MSC en las columnas 1, 2 y 3 respectivamente. De la columna 4 a la 8 se muestran el mejor makespan encontrado de las 30 ejecuciones, su ER, el promedio de makespan, el error relativo promedio y el tiempo de ejecución expresado en segundos de las 30 ejecuciones.

Tabla 3.5: *Resultados de sintonización manual*

			Sintonización Manual				
Instancia	Tamaño	MSC	Mejor	ER	Promedio	ER	Tiempo de ejecución (s)
FT06-MT06	6x6	55	**55**	0.0	55.0	0.0	7.9
FT10-MT10	10x10	930	947	1.8	952.0	2.4	181.0
FT20-MT05	20x5	1165	1167	0.2	1178.9	1.2	126.5
YN1	20x20	884	901	1.9	910.3	3.0	5068.6
YN2	20x20	904	952	5.3	956.7	5.8	5173.6
YN3	20x20	892	904	1.3	908.5	1.8	5882.3
YN4	20x20	968	988	2.1	991.7	2.4	6120.7
TA01	15x15	1231	1269	3.1	1287.3	4.6	1202.9
TA11	20x15	1357	1398	3.0	1405.9	3.6	2512.4
TA21	20x20	1642	1685	2.6	1685.5	2.6	6182.8
TA31	30x15	1764	1863	5.6	1870.3	6.0	4596.3
TA41	30x20	2005	2076	3.5	2085.0	4.0	13863.2
TA51	50x15	2760	2920	5.8	2920.0	5.8	14869.8
TA61	50x20	2868	**2868**	0.0	2868.0	0.0	31482.5
TA71	100x20	5464	**5464**	0.0	5464.0	0.0	70378.6
ORB01	10x10	1059	**1059**	0.0	1079.3	1.9	232.4
ORB02	10x10	888	889	0.1	889.7	0.2	228.3
ORB03	10x10	1005	1017	1.2	1023.9	1.9	247.5
ORB04	10x10	1005	**1005**	0.0	1012.4	0.7	224.3
ORB05	10x10	887	**887**	0.0	893.0	0.7	241.1
LA01	10x5	666	**666**	0.0	666.0	0.0	22.0
LA06	15x5	926	**926**	0.0	926.0	0.0	11.1
LA16	10x10	945	**945**	0.0	947.1	0.2	172.5
LA21	15x10	1046	**1046**	0.0	1073.2	2.6	431.3
LA26	20x10	1218	**1218**	0.0	1218.0	0.0	683.2
PROMEDIO				1.5		2.1	6805.7

3.3. Recocido Simulado con sintonización analítica de parámetros

Para poder realizar el proceso de sintonización analítica, descrito en [53], se deben de realizar una serie de ejecuciones previas del algoritmo. Para estas ejecuciones previas se tienen que establecer de forma manual los parámetros ya que es necesario que el algoritmo genere algunas soluciones para poder usarlas y hacer los cálculos del proceso de sintonización analítica, para después obtener finalmente los parámetros sintonizados.

Por ejemplo, para la instancia *mt10* o *ft10* se realizan los siguientes pasos.

- 1. Establecer las probabilidades de aceptar una solución con el decremento máximo ($P(\Delta Z_{max})$) y con el decremento mínimo ($P(\Delta Z_{min})$). En nuestra investigación utilizamos los valores mostrados en la Tabla 3.6, ya que se desea que tenga un alto grado de exploración.

  Tabla 3.6: *Valores para $P(\Delta Z_{max})$ y $P(\Delta Z_{min})$*

$P(\Delta Z_{max}) =$	0.99
$P(\Delta Z_{min}) =$	0.01

  Entonces para un nivel de exploración (C) con un 99 % de probablidades de aceptar una solución se tiene que $C = ln(1 - P(\Delta Z_{max})) = ln(1 - 0.99) = 4.6$

- 2. Calcular el tamaño de la vecindad ($|V_{si}|$). La estrategia que se siguió para obtener este tamaño es utilizar la cantidad de operaciones de la instancia a resolver.

  $|V_{si}| =$ *número de trabajos* $\times$ *número de máquinas* = *Número de operaciones o tareas*

  De esta forma entonces: $|V_{si}| = 10 \times 10 = 100$

- 3. Obtener el máximo número de iteraciones (L_{max}) que se realizan en el ciclo de Metrópolis a la temperatura final

  $L_{max} = C \times |V_{si}| = 4.6 \times 100 = 460$

  El valor de L_0, es decir el número de iteraciones a la temperatura inicial se establece a 1, ya que a altas temperaturas pocas iteraciones son necesarias.

- 4. Calcular el valor para ΔZ_{max} y ΔZ_{min} con base a las ejecuciones previas del algoritmo. Entonces en este caso quedan de la siguiente forma:

  $\Delta Z_{max} =$ peor solución - mejor solución $= 2228 - 967 = 1261$

  $\Delta Z_{min} =$ segunda mejor solución - mejor solución $= 968 - 967 = 1$

- 5. Calcular la $T_{inicial}$ y la T_{final}

  $T_{inicial} = \frac{-\Delta Z_{max}}{ln(P_{aceptacion}(\Delta Z_{max}))} = \frac{-1261}{ln(0.99)} = 125,468.44$

$T_{final} = \frac{-\Delta Z_{min}}{ln(P_{aceptacion}(\Delta Z_{min}))} = \frac{-1}{ln(0.01)} = 0.2171$

- 6. Obtener el valor de n, que es el número de temperaturas que hay desde la $T_{inicial}$ hasta la T_{final}

  $n = \frac{ln(T_{final}/T_{inicial})}{ln(\alpha)} = \frac{ln(0.2171/125478.44)}{ln(0.99)} = 1320.076$

- 7. Finalmente se puede obtener el valor de β

  $\beta = e^{\frac{ln(L_{max})-ln(L_0)}{n}} = e^{\frac{ln(460)-ln(1)}{1320.076}} = 1.0046$

Este algoritmo requiere ejecuciones previas para poder tener soluciones que nos permitan utilizar las fórmulas de la sintonización analítica, proceso explicado previamente. En este caso se realizaron 15 ejecuciones previas.

Resultados obtenidos con sintonización analítica

En la Tabla 3.7 se muestra el nombre de la instancia, su tamaño y la MSC en la columna 1, 2 y 3 respectivamente. De la columna 4 a la 8 se muestran el mejor makespan encontrado, su ER, el promedio del makespan, error relativo del promedio del makespan y el tiempo de ejecución, expresado en segundos. Este algoritmo fue ejecuta 30 veces al igual que el anterior.

Los resultados de la experimentación realizada muestran que mediante la aplicación del método de sintonización analítica se logran obtener buenos resultados empleando muy poco tiempo de procesamiento en comparación con la sintonización manual. Particularmente para las instancias consideradas pequeñas, de menos de 200 tareas, como mt06, MT10, MT20, ORB02, ORBb04, LA01, LA06, LA16 y LA26, así como para la instancia de mayor tamaño TA71, la cual esta constituída por 2000 tareas, son en donde se obtiene la MSC. Por otro lado, se logran reducir considerablemente los tiempos de procesamiento, por ejemplo se puede observar que para la instancia más pequeña (6x6) el tiempo de procesamiento está por debajo del segundo lo que indica que se reduce en un 98.7 %

Tabla 3.7: *Resultados de sintonizacion analítica*

			Sintonización Analítica				
Instancia	Tamaño	MSC	Mejor	ER	Promedio	ER	Tiempo de ejecución (s)
FT06-MT06	6x6	55	**55**	0.0	55.6	1.0	0.1
FT10-MT10	10x10	930	**930**	0.0	986.0	6.0	1.2
FT20-MT05	20x5	1165	**1165**	0.0	1195.9	2.7	0.8
YN1	20x20	884	914	3.4	940.9	6.4	25.9
YN2	20x20	904	969	7.2	1000.7	10.7	22.9
YN3	20x20	892	934	4.7	960.8	7.7	22.9
YN4	20x20	968	1017	5.1	1047.1	8.2	26.6
TA01	15x15	1231	1256	2.0	1316.8	7.0	6.5
TA11	20x15	1357	1410	3.9	1468.5	8.2	12.0
TA21	20x20	1642	1744	6.2	1797.0	9.4	25.2
TA31	30x15	1764	1823	3.3	1866.9	5.8	23.0
TA41	30x20	2005	2159	7.7	2230.7	11.3	51.2
TA51	50x15	2760	2864	3.8	2901.4	5.1	59.7
TA61	50x20	2868	2952	2.9	2979.1	3.9	136.1
TA71	100x20	5464	**5464**	0.0	5464.3	0.0	513.2
ORB01	10x10	1059	1064	0.5	1126.6	6.4	1.1
ORB02	10x10	888	889	0.1	922.8	3.9	1.4
ORB03	10x10	1005	1022	1.7	1101.5	9.6	1.2
ORB04	10x10	1005	**1005**	0.0	1042.1	3.7	1.1
ORB05	10x10	887	890	0.3	945.0	6.5	1.2
LA01	10x5	666	**666**	0.0	668.2	0.3	0.2
LA06	15x5	926	**926**	0.0	926.0	0.0	0.4
LA16	10x10	945	**945**	0.0	982.4	4.0	1.3
LA21	15x10	1046	1052	0.6	1089.8	4.2	2.2
LA26	20x10	1218	**1218**	0.0	1225.3	0.6	3.6
PROMEDIO				2.1		5.3	37.6

respecto a su versión sintonizada manualmente y para la instancia TA71, que es la que más tiempo de procesamiento consume se reduce en un 99.3 % con la sintonización analítica.

3.4. Recocido Simulado con equilibrio estocástico (SASE)

Debido a que el parámetro L_k (iteraciones del ciclo de Metrópolis) puede ser muy grande para algunas instancias JSSP, es importante aplicar un proceso particular para detectar cuando el algoritmo no mejora en su búsqueda. Durante este periodo de la investigación se implementa un proceso para que el algoritmo encuentre en forma dinámica el equilibrio estocástico. Que es el proceso que determina la

iteración donde la pendiente de la función de energía permanece muy cerca de cero. Esta fase también es un procedimiento de búsqueda de recocido simulado, que puede verse como una segunda fase de Boltzmann. Equilibrio dinámico utiliza una heurística particular para detectar el equilibrio estocástico aplicando un método de mínimos cuadrados durante su ejecución. Se aplica a bajas temperaturas buscando una solución mejor que la obtenida por SA.

Sea (x, E_i) un conjunto de n puntos con $i = 1, 2, ..., n$. E_i representa el makespan de la solución en el punto x_i. El objetivo es encontrar una línea recta, que se define como $f(x_i) = mx_i + b$ que se aproxima al conjunto de puntos, donde m representa la pendiente de la línea recta, y se calcula aplicando el método de mínimos cuadrados. El parámetro b representa la intersección con el eje x. Utilizando el método estándar de mínimos cuadrados, la pendiente (m) para las n iteraciones está definida por la ecuación 3.4.1.

$$m = \frac{n\sum_{i=1}^{n} x_i E_i - (\sum_{i=1}^{n} x_i)(\sum_{i=1}^{n} E_i)}{n\sum_{i=1}^{n} x_i^2 - (\sum_{i=1}^{n} x_i)^2} \tag{3.4.1}$$

Donde: E_i es el makespan encontrado por el algoritmo en la iteración x_i.

Aplicando notación sigma podemos obtener la ecuación 3.4.2.

$$m = \frac{12\sum_{i=1}^{n} x_i E_i - 6(n-1)\sum_{i=1}^{n} E_i}{n^3 - n} \tag{3.4.2}$$

Y si hacemos que:

$$k_1 = \frac{12}{n^3 - n} \tag{3.4.3}$$

$$k_2 = \frac{6}{n^2 - n} \tag{3.4.4}$$

Entonces podemos obtener la ecuación 3.4.5, en donde k_1 y k_2 son únicamente constantes para un valor particular de n.

$$m = k_1 \sum_{i=1}^{n} iE_i - k_2 \sum_{i=1}^{n} E_i \tag{3.4.5}$$

Recordar, que la condición de equilibrio dinámico se cumple cuando la pendiente (m) de la línea recta está muy cerca de cero. Para un mayor detalle, en el Algoritmo 2 se muestra el proceso implementado.

Este es un método de regresión en el ciclo de metrópolis; en donde el número de iteraciones se considera como la variable independiente y el valor de energía de cada iteración como la variable dependiente. El criterio de detección de equilibrio es la pendiente de la función energética en el ciclo de Metrópolis.

Entonces, del Algoritmo 2 se establece que: $T_{inicial}$ como es el final del recocido simulado tiene un valor extremadamente bajo muy cercano a cero, α tiene un valor cercano a 1, en este caso 0.999 para una reducción lenta de la temperatura y n toma su valor del recocido simulado.

En resumen, este proceso se aplica para determinar la iteración en la cual la pendiente de los valores del makespan es cercana a cero, con el fin de detectar el momento preciso en el cual el ciclo de Metrópolis ha llegado al punto de equilibrio.

Este algoritmo implementa una condición de paro diferente a la del algoritmo recocido simulado tradicional en el cual se utiliza es el valor de la temperatura final. Para esto realiza el cálculo de la pendiente como ya se ha explicado.

Algorithm 2 Equilibrio Estocástico

procedure EQUILIBRIO ESTOCÁSTICO($T_{inicial}, \alpha, n$)
  $T_k \leftarrow T_{inicial}$
  $CM \leftarrow n$
  $\varepsilon \leftarrow 0.0001$
  while $m > \varepsilon$ **do**
    $k \leftarrow 0$
    $E_{sum} \leftarrow 0, sum \leftarrow 0$
    while $k \leq CM$ **do**
      $s_j \leftarrow NeighboorSolution$
      $\Delta \leftarrow E(s_i) - E(s_j)$
      if $\Delta < 0$ **then**
        $s_i \leftarrow s_j$
      else
        if $(e^{-\Delta/Tk} > random(0,1)$ **then**
          $s_i \leftarrow s_j$
        end if
      end if
      $k \leftarrow k+1$
      $sum \leftarrow sum + k * E(s_i)$
      $E_{sum} \leftarrow E_{sum} + E(s_i)$
    end while
    $T_{k+1} \leftarrow \alpha * T_k$
    $k_1 \leftarrow 12/(k*k*k-k)$
    $k_2 \leftarrow 6/(k*k-k)$
    $m \leftarrow k_1 * sum - k_2 * Esum$
  end while
  return s_i
end procedure

Resultados obtenidos SASE

Los resultados obtenidos se muestran en la Tabla 3.8 . Al utilizar la sintonización analítica y además la fase del equilibrio estocástico se obtienen resultados de buena calidad empleando poco tiempo de procesamiento.

Tabla 3.8: *Resultados Recocido Simulado con Equilibrio Estocástico*

			Sintonización Analítica + equilibrio				
Instancia	Tamaño	MSC	Mejor	ER	Promedio	ER	Tiempo de ejecución (s)
FT06-MT06	6x6	55	**55**	0.0	55.5	1.0	0.2
FT10-MT10	10x10	930	**930**	0.0	983.4	5.7	1.4
FT20-MT05	20x5	1165	**1165**	0.0	1195.7	2.6	1.0
YN1	20x20	884	910	2.9	944.0	6.8	25.1
YN2	20x20	904	946	4.6	964.8	6.7	28.6
YN3	20x20	892	925	3.7	952.9	6.8	29.0
YN4	20x20	968	1008	4.1	1015.5	4.9	25.6
TA01	15x15	1231	1257	2.1	1306.4	6.1	8.0
TA11	20x15	1357	1413	4.1	1472.5	8.5	13.3
TA21	20x20	1642	1753	6.8	1788.4	8.9	28.6
TA31	30x15	1764	1813	2.8	1845.8	4.6	27.2
TA41	30x20	2005	2154	7.4	2210.7	10.3	55.9
TA51	50x15	2760	**2760**	0.0	2782.4	0.8	60.9
TA61	50x20	2868	3067	6.9	3101.6	8.1	124.1
TA71	100x20	5464	**5464**	0.0	5468.2	0.1	543.8
ORB01	10x10	1059	1060	0.1	1125.9	6.3	1.3
ORB02	10x10	888	889	0.1	921.7	3.8	1.6
ORB03	10x10	1005	1020	1.5	1102.7	9.7	1.5
ORB04	10x10	1005	**1005**	0.0	1042.0	3.7	1.4
ORB05	10x10	887	890	0.3	949.4	7.0	1.6
LA01	10x5	666	**666**	0.0	668.3	0.3	0.3
LA06	15x5	926	**926**	0.0	926.0	0.0	0.5
LA16	10x10	945	**945**	0.0	982.6	4.0	1.3
LA21	15x10	1046	1048	0.2	1090.8	4.3	2.5
LA26	20x10	1218	**1218**	0.0	1223.2	0.4	4.8
PROMEDIO				1.9		4.9	39.6

Observaciones SASE

Los resultados muestran que el proceso de equilibrio dinámico logra reducir el Error Relativo del mejor makespan y del promedio de makespan y solo le agrega 2 segundos al tiempo promedio de ejecución del algoritmo. Además encuentra la mejor solución conocida para 10 de las instancias resueltas.

3.5. Recocido Simulado con reheat (SARH)

El proceso de reheat o recalentamiento se implementa con el objetivo de escapar de óptimos locales encontrados. Reheat se refiere al proceso mediante el cual la temperatura actual del algoritmo (T_k) se

reconfigura (se eleva) si después de cierta cantidad de iteraciones no hay mejora en la calidad de la solución. Al elevar la temperatura actual se le permite al algoritmo mayor flexibilidad para que acepte soluciones de menor calidad y de esa manera logre escapar de óptimos locales.

Para poder implementar este proceso primero se tiene que determinar cuando el algoritmo se encuentra estancado, para ello se realiza un proceso alterno que consiste en llevar el conteo de iteraciones en las cuales no hubo mejora a la solución actual, este límite de iteraciones se establece en 10. También se establece un máximo de 10 procesos de reheat a aplicar en el algoritmo, con el objetivo de que no se quede infinitamente aplicando reheats y se corra el riesgo de que nunca alcance su condición de paro.

Resultados obtenidos SARH

En la Tabla 3.9 se muestran los resultados experimentales utilizando sintonización analítica, equilibrio estocástico y aplicando recalentamiento.

Las columnas 1, 2 y 3 muestran el nombre de la instancia, su tamaño y la MSC respectivamente. De la columna 4 a la 8 se muestran el mejor makespan encontrado, su ER, el promedio del makespan, error relativo del promedio del makespan y el tiempo de ejecución, expresado en segundos.

Observaciones SARH

Al agregar el proceso de recalentamiento se logra reducir el ER del mejor makespan encontrado anteriormente de 1.9 a 1.1, el ER promedio de las 25 instancias se reduce de 4.9 a 3.8 y el tiempo promedio de ejecución se incrementa solo 19 segundos. Además en 13 de las 25 instancias se logra obtener la MSC y en 2 más se obtienen un error de 0.1.

Tabla 3.9: *Resultados de Recocido Simulado con reheat*

			SA con reheat				
Instancia	Tamaño	MSC	Mejor	ER	Promedio	ER	Tiempo de ejecución (s)
FT06-MT06	6x6	55	**55**	0.0	55.0	0.0	0.1
FT10-MT10	10x10	930	**930**	0.0	971.6	4.5	12.6
FT20-MT05	20x5	1165	**1165**	0.0	1187.4	1.9	3.3
YN1	20x20	884	909	2.8	937.6	6.1	35.5
YN2	20x20	904	940	4.0	963.8	6.6	32.3
YN3	20x20	892	918	2.9	950.5	6.6	30.4
YN4	20x20	968	983	1.5	1003.1	3.6	317.3
TA01	15x15	1231	1241	0.8	1294.4	5.1	31.0
TA11	20x15	1357	1392	2.6	1451.7	7.0	51.6
TA21	20x20	1642	1717	4.6	1787.5	8.9	38.7
TA31	30x15	1764	1786	1.2	1843.7	4.5	32.5
TA41	30x20	2005	2111	5.3	2171.6	8.3	146.0
TA51	50x15	2760	**2760**	0.0	2760.6	0.0	60.4
TA61	50x20	2868	**2868**	0.0	2888.4	0.7	263.2
TA71	100x20	5464	**5464**	0.0	5464.0	0.0	258.1
ORB01	10x10	1059	**1059**	0.0	1113.7	5.2	73.3
ORB02	10x10	888	889	0.1	920.5	3.7	4.2
ORB03	10x10	1005	1020	1.5	1079.6	7.4	21.5
ORB04	10x10	1005	**1005**	0.0	1040.2	3.5	5.6
ORB05	10x10	887	**887**	0.0	928.7	4.7	28.7
LA01	10x5	666	**666**	0.0	666.0	0.0	0.1
LA06	15x5	926	**926**	0.0	926.0	0.0	0.2
LA16	10x10	945	**945**	0.0	976.4	3.3	10.0
LA21	15x10	1046	1047	0.1	1087.5	4.0	4.0
LA26	20x10	1218	**1218**	0.0	1222.2	0.3	5.0
PROMEDIO				1.1		3.8	58.6

3.6. Recocido Simulado con perturbación caótica (SAPC)

En este proceso se utiliza una función caótica, que es muy sensible a los cambios en las condiciones iniciales, para generar una perturbación a la solución actual con el propósito de escapar de óptimos locales.

En el Algoritmo 3 se muestra el método con la perturbación implementada, este puede ser resumido en tres pasos: primero, se buscan las operaciones factibles (línea 4), luego se utiliza la función caótica para seleccionar alguna operación factible que se agregará a la nueva solución, esto se hace cuando la salida de la función caótica genera un valor mayor que 0.5 (línea 13) y en el último paso la operación se agrega a la nueva solución (línea 21). De esa forma se repite el proceso hasta terminar de agregar

todas las operaciones que conforman la instancia a la nueva solución (s_{new}). El valor del parámetro r en el algoritmo se establece en 4 porque en ese valor la función utilizada tiene el comportamiento caótico. La perturbación caótica se aplica después de 5 procesos de reheat.

Algorithm 3 Perturbación caótica

```
1:  procedure PERTURBACIONCAOTICA(s_old)
2:      r ← 4
3:      while contador < numeroTareas do
4:          buscaOperacionesFactibles()
5:          if numFactibles == 1 then
6:              indice ← 0
7:          else
8:              while repetir == TRUE do
9:                  Xn ← random(0,1)
10:                 for i ← 0 to numFactibles do
11:                     Xn1 ← (r × Xn) × (1.0 − Xn)
12:                     if Xn1 > 0.5 then
13:                         indice ← i
14:                         repetir ← FALSE
15:                         break
16:                     end if
17:                     Xn ← Xn1
18:                 end for
19:             end while
20:         end if
21:         s_new ← agregarOperacion(i)
22:         contador ← contador + 1
23:     end while
24:     return s_new
25: end procedure
```

Resultados Obtenidos SAPC

Los resultados que se detallan en la Tabla 3.10 muestran que al utilizar el proceso de recalentamiento y la perturbación caótica se logra reducir el ER promedio de las 25 instancias de 1.1 a 0.8 en la mejor solución y de 3.8 a 2.8 en su promedio. Por otro lado, es necesario señalar que el tiempo de ejecución promedio se eleva considerablemete práctimente triplicándose al pasar de 58.6 a 149.1.

Tabla 3.10: *Resultados de Recocido Simulado con caos*

			SA con Caos				
Instancia	Tamaño	MSC	Mejor	ER	Promedio	ER	Tiempo de ejecución (s)
FT06-MT06	6x6	55	**55**	0.0	55.0	0.0	0.1
FT10-MT10	10x10	930	**930**	0.0	966.9	4.0	6.8
FT20-MT05	20x5	1165	**1165**	0.0	1178.2	1.1	5.2
YN1	20x20	884	906	2.5	919.7	4.0	100.5
YN2	20x20	904	919	1.7	945.2	4.6	95.3
YN3	20x20	892	903	1.2	925.7	3.8	85.0
YN4	20x20	968	985	1.8	1015.1	4.9	111.4
TA01	15x15	1231	1248	1.4	1275.0	3.6	57.0
TA11	20x15	1357	1388	2.3	1434.9	5.7	69.5
TA21	20x20	1642	1694	3.2	1740.2	6.0	94.7
TA31	30x15	1764	1766	0.1	1789.8	1.5	161.6
TA41	30x20	2005	2103	4.9	2154.4	7.5	227.7
TA51	50x15	2760	**2760**	0.0	2760.0	0.0	148.5
TA61	50x20	2868	**2868**	0.0	2868.0	0.0	738.2
TA71	100x20	5464	**5464**	0.0	5464.0	0.0	1771.5
ORB01	10x10	1059	**1059**	0.0	1108.6	4.7	8.4
ORB02	10x10	888	889	0.1	899.4	1.3	7.1
ORB03	10x10	1005	**1005**	0.0	1073.0	6.8	5.8
ORB04	10x10	1005	**1005**	0.0	1023.7	1.9	6.3
ORB05	10x10	887	889	0.2	926.2	4.4	7.4
LA01	10x5	666	**666**	0.0	666.1	0.0	0.1
LA06	15x5	926	**926**	0.0	926.0	0.0	0.1
LA16	10x10	945	**945**	0.0	954.8	1.0	5.6
LA21	15x10	1046	1047	0.1	1078.9	3.1	7.4
LA26	20x10	1218	**1218**	0.0	1219.3	0.1	6.0
PROMEDIO				0.8		2.8	149.1

3.7. Recocido Simulado paralelo con OpenMP

Como ya se expresó anteriormente, OpenMP permite implementar código paralelo en arquitecturas de memoria compartida. Se realiza una versión paralela del algoritmo recocido simulado, utilizando 2 y 4 hilos. Se aplica el modelo *fork-join*, característico de OpenMP, donde una tarea muy pesada se divide en *k* hilos (*fork*) con menor peso para después "recolectar" sus resultados al final y unirlos en un solo resultado (*join*).

En la figura 3.1 se muestra la implementación realizada.

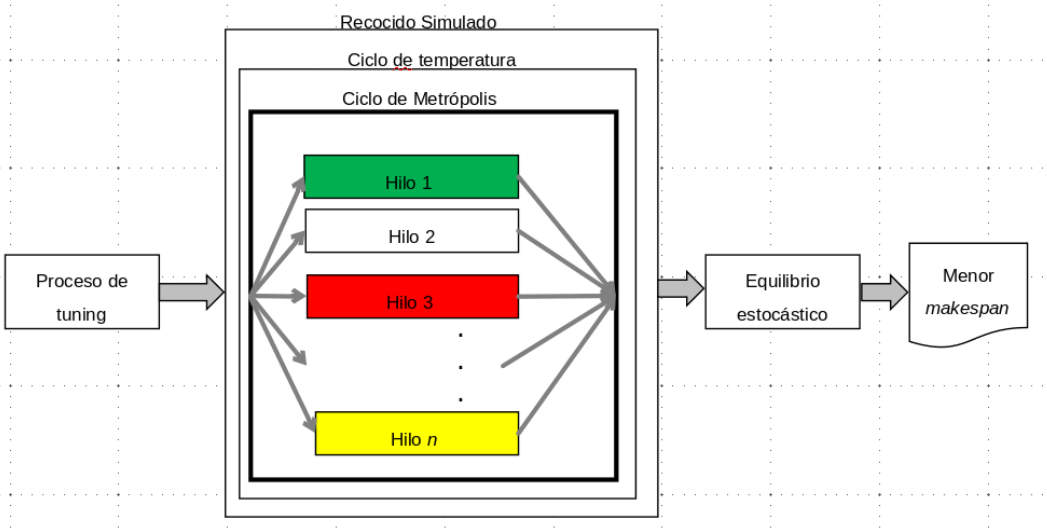

Figura 3.1: *Recocido simulado paralelo con OpenMP*

Como primer paso se realiza un proceso de sintonización de parámetros, posteriormente inician los ciclos de temperatura y de Metrópolis. Las iteraciones del ciclo de Metrópolis se reparten en *n* hilos utilizando directivas de OpenMP. Alfinal se aplica una fase de equilibrio estocástico y como salida del proccesamiento se imprime la solución con el menor makespan encontrado.

3.7.1. Resultados Obtenidos utilizando 2 hilos (PSA-2)

La Tabla 3.11 muestra los resultados de la experimentación realizada con 2 hilos y tiene la misma estructura que las presentadas en las anteriores tablas de la presente sección.

Tabla 3.11: *Resultados de Recocido Simulado Paralelo con OpenMP (2 hilos)*

			OpenMP con 2 hilos				
Instancia	Tamaño	MSC	Mejor	ER	Promedio	ER	Tiempo de ejecución (s)
FT06-MT06	6x6	55	**55**	0.0	55.6	1.0	0.3
FT10-MT10	10x10	930	**930**	0.0	982.6	5.7	2.4
FT20-MT05	20x5	1165	1173	0.7	1194.6	2.5	1.5
YN1	20x20	884	906	2.5	933.5	5.6	51.1
YN2	20x20	904	938	3.8	962.6	6.5	51.9
YN3	20x20	892	920	3.1	948.7	6.4	51.8
YN4	20x20	968	1008	4.1	1043.2	7.8	47.0
TA01	15x15	1231	1275	3.6	1327.5	7.8	11.4
TA11	20x15	1357	1438	6.0	1464.4	7.9	24.5
TA21	20x20	1642	1742	6.1	1787.7	8.9	53.9
TA31	30x15	1764	1798	1.9	1843.9	4.5	50.1
TA41	30x20	2005	2149	7.2	2209.3	10.2	120.3
TA51	50x15	2760	**2760**	0.0	2762.1	0.1	134.7
TA61	50x20	2868	2879	0.4	2925.1	2.0	286.1
TA71	100x20	5464	**5464**	0.0	5464.0	0.0	1081.1
ORB01	10x10	1059	1083	2.3	1121.7	5.9	2.6
ORB02	10x10	888	892	0.5	917.9	3.4	2.5
ORB03	10x10	1005	1030	2.5	1095.6	9.0	2.6
ORB04	10x10	1005	1012	0.7	1038.9	3.4	2.3
ORB05	10x10	887	899	1.4	949.1	7.0	2.8
LA01	10x5	666	**666**	0.0	668.7	0.4	0.4
LA06	15x5	926	**926**	0.0	926.0	0.0	0.9
LA16	10x10	945	946	0.1	979.0	3.6	2.9
LA21	15x10	1046	1058	1.1	1087.3	3.9	5.4
LA26	20x10	1218	**1218**	0.0	1224.0	0.5	8.2
PROMEDIO				1.9		4.6	80.0

3.7.2. Resultados Obtenidos utilizando 4 hilos (PSA-4)

La Tabla 3.12 muestra los resultados de la experimentación realizada con 4 hilos de procesameinto. Esta tabla tiene la misma estructura que la anterior.

De esta versión paralela se observa que el tiempo de procesamiento se eleva en la versión con 4 hilos, debido a que el tiempo de comunicación que existe entre los hilos aumenta conforme más hilos compiten para actualizar variables compartidas. En el algoritmo recocido simulado el resultado actual depente del resultado anterior. Otra cosa que se observa es que la calidad de las soluciones practicamente no tiene variación entre la versión de 2 hilos y la de 4 para este algoritmo en particular.

Tabla 3.12: *Resultados de Recocido Simulado Paralelo con OpenMP (4 hilos)*

			OpenMP con 4 hilos				
Instancia	Tamaño	MSC	Mejor	ER	Promedio	ER	Tiempo de ejecución (s)
FT06-MT06	6x6	55	**55**	0.0	55.5	0.9	0.6
FT10-MT10	10x10	930	**930**	0.0	980.8	5.5	5.0
FT20-MT05	20x5	1165	**1165**	0.0	1192.9	2.4	3.3
YN1	20x20	884	917	3.7	936.8	6.0	97.8
YN2	20x20	904	934	3.3	963.5	6.6	103.5
YN3	20x20	892	924	3.6	941.7	5.6	104.7
YN4	20x20	968	1017	5.1	1042.2	7.7	96.9
TA01	15x15	1231	1261	2.4	1313.6	6.7	23.7
TA11	20x15	1357	1440	6.1	1475.6	8.7	50.5
TA21	20x20	1642	1714	4.4	1783.1	8.6	103.1
TA31	30x15	1764	1840	4.3	1876.8	6.4	90.5
TA41	30x20	2005	2154	7.4	2203.0	9.9	237.8
TA51	50x15	2760	**2760**	0.0	2762.1	0.1	267.1
TA61	50x20	2868	2973	3.7	3002.5	4.7	515.7
TA71	100x20	5464	**5464**	0.0	5464.0	0.0	1842.0
ORB01	10x10	1059	1083	2.3	1115.8	5.4	5.1
ORB02	10x10	888	889	0.1	918.5	3.4	5.0
ORB03	10x10	1005	1049	4.4	1097.6	9.2	5.7
ORB04	10x10	1005	1017	1.2	1040.2	3.5	4.9
ORB05	10x10	887	890	0.3	934.5	5.3	6.0
LA01	10x5	666	**666**	0.0	667.9	0.3	0.9
LA06	15x5	926	**926**	0.0	926.0	0.0	1.8
LA16	10x10	945	946	0.1	974.8	3.2	5.8
LA21	15x10	1046	1060	1.3	1085.3	3.8	9.9
LA26	20x10	1218	**1218**	0.0	1222.6	0.4	16.5
PROMEDIO				2.2		4.6	144.2

CAPÍTULO 4

Métodos propuestos para Job Shop Scheduling Problem Multi-Objetivo

En esta sección se describen los algoritmos multi-objetivo desarrollados y el algoritmo principal que ejecuta los métodos propuestos. Además se describen las instancias y las métricas utilizadas para evaluar el desempeño de estos algoritmos.

4.1. Chaotic Multi-Objective Simulated Annealing (CMOSA)

El algoritmo CMOSA propuesto es un poco más complejo que los vistos anteriormente ya que tiene mayor número de subprocesos (cálculo de due date, dos objetivos más y métricas de desempeño, etc) que implementan algunos de los métodos ya descritos. En nuestra versión CMOSA para JSSP, el algoritmo recocido simulado se aplica para mejorar cada solución inicial con los tres objetivos.

Algorithm 4 Chaotic Multi-Objective Simulated Annealing (CMOSA)

procedure CMOSA($T_{initial}, T_{final}, \alpha, \beta, L_k, s_{current}$)
  $MAXSTAGNANT \leftarrow 10, counterTrapped \leftarrow 0, isCaught \leftarrow FALSE$
  $iterationsLocalSearch \leftarrow tasks \times timesLS, verifyCaught \leftarrow TRUE, countCaught \leftarrow 0$
  $mks_{current}, tds_{current}, flt_{current} \leftarrow calculateValues(s_{current})$ ▷ $mks: makespan, tds: tardiness, flt: flowtime$
  $T_k \leftarrow T_{initial}$
  while $T_k \geq T_{final}$ **do**
    for $i \leftarrow 0$ to L_k **do**
      if $isCaught = TRUE$ **then**
        $isCaught \leftarrow FALSE$
        for $k \leftarrow 0$ to $iterationsLocalSearch$ **do**
          if $k = 0$ **then**
            $s_{new} \leftarrow chaoticPerturbation(s_{current})$ ▷ See Algorithm 6
          else
            $s_{new} \leftarrow regularPerturbation(s_{current})$ ▷ Exchange of two operations
          end if
          $mks_{new}, tds_{new}, flt_{new} \leftarrow calculateValues(s_{new})$
          if $(mks_{new} < mks_{current})$ AND $(tds_{new} < tds_{current})$ AND $(flt_{new} < flt_{current})$ **then**
            $s_{current} \leftarrow s_{new}$
          end if
        end for
      else
        $s_{new} \leftarrow regularPerturbation(s_{current})$
        $mks_{new}, tds_{new}, flt_{new} \leftarrow calculateValues(s_{new})$
      end if
      if $(mks_{new} \neq mks_{current})$ AND $(tds_{new} \neq tds_{current})$ AND $(flt_{new} \neq flt_{current})$ **then**
        $verifyDominanceCMOSA(T_k, s_{new}, s_{current})$ ▷ See Algorithm 7
      end if
    end for
    if $verifyCaught = TRUE$ **then**
      if $caught(s_{current}, counterTrapped) = TRUE$ **then** ▷ See Algorithm 5
        $countCaught = countCaught + 1$
        if $countCaught = MAXSTAGNANT$ **then**
          $verifyCaught \leftarrow FALSE$
        end if
      end if
    end if
    $L_k \leftarrow \beta \times L_k$
    $T_k \leftarrow \alpha \times T_k$
  end while
  return $s_{current}$
end procedure

El Algoritmo 4 recibe sus parámetros en la línea 1: temperatura inicial ($T_{initial}$), temperatura final (T_{final}), alfa (α), beta (β), iteraciones del ciclo de Metrópolis (L_k) y la solución inicial ($s_{current}$) a mejorar. En las líneas 2 y 3 se inicializan las variables del algoritmo. En la línea 4, se procesa el $s_{current}$ para obtener los valores para cada uno de los tres objetivos como salida. En la línea 5, la temperatura inicial se establece como la temperatura actual (T_k). Luego, el ciclo principal comienza en la línea 6. Este ciclo se repite siempre que la temperatura actual sea mayor o igual que la temperatura final. En la línea 7, comienza el ciclo Metrópolis. Posteriormente, el algoritmo verifica si está estancado en la línea 8. Si ese es el caso, se ejecutan las líneas 9 a 20. El número de iteraciones para realizar una búsqueda local se establece en la línea 10; este valor se basa en el número de tareas de la instancia multiplicado por un parámetro que es sintonizado experimentalmente (en este caso, este parámetro es $timesLS = 10$). En la línea 11, da inicio un proceso de búsqueda local. En la primera iteración de esta búsqueda, se aplica una perturbación caótica (explicada en Algoritmo 6) a $s_{current}$ (línea 12), esto se realiza para reiniciar el proceso de búsqueda desde otro punto en el espacio de la solución. En posteriores iteraciones se aplica una perturbación regular (línea 14) que consiste únicamente en intercambiar la posición de dos operaciones en la solución, verificando siempre que la solución generada sea factible. En la línea 16, se procesa el s_{new} para obtener los valores para cada uno de los tres objetivos. Posteriormente, y solo si la nueva solución domina la solución actual en los tres objetivos, se utiliza la nueva solución para continuar el proceso de búsqueda (líneas 17 y 18). Cuando el algoritmo no está estancado, se aplica una perturbación regular y el flujo continúa (línea 22). Si la solución actual y la nueva son diferentes, procedemos con el proceso de verificación de dominancia para determinar qué solución se usa para continuar la búsqueda (línea 26); este proceso se explica en Algoritmo 7. Finalmente, de la línea 29 a la 36, se aplica un proceso para establecer un límite al número de veces que el algoritmo está estancado (Ver Algoritmo 5). Se determina que el algoritmo está estancado si, después de algunas iteraciones, no genera una nueva solución no dominada. En este algoritmo, el estancamiento se limita a diez iteraciones. Al final del algoritmo, en la línea 37, el número de repeticiones del ciclo Metrópolis (L_k) se incrementa multiplicando su valor anterior por el valor del parámetro β. Además, en la línea 38, la temperatura actual (T_k) se reduce multiplicándola por el valor α. Al final de la línea 40, la solución almacenada (s_{actual}) se genera como la salida del algoritmo.

El Algoritmo 5 muestra el proceso que se realiza para verificar el estancamiento mencionado en la línea 30 del algoritmo 4. En este algoritmo, la solución actual ($s_{current}$) y el contador de veces que se ha estancado ($counterTrapped$) se reciben como entrada. En la línea 2 se inicializan las variables utilizadas. Luego, se cuentan las veces que la solución actual está dominada por al menos una solución del frente no dominado (línea 3). Si la solución actual es no dominada (línea 4), se almacena al frente de las soluciones no dominadas (línea 5). Si la solución actual está dominada por al menos una solución (línea 7), entonces $counterTrapped$ se incrementa (línea 8). Cuando $counterTrapped$ es igual al número máximo de atrapados permitidos (línea 10), el valor de *isCaught* se establece en $TRUE$ (línea 11) y el contador de estancados se restablece a cero en la línea 12.

Algorithm 5 Atrapado

```
1:  procedure CAUGHT(s_current, counterTrapped)
2:      isCaught ← FALSE, timesDominated ← 0, maxTrapped ← 10
3:      timesDominated ← countTimesDominated(s_current)
4:      if timesDominated = 0 then
5:          F ← s_current
6:      end if
7:      if timesDominated ≥ 1 then
8:          counterTrapped ← counterTrapped + 1
9:      end if
10:     if counterTrapped = maxTrapped then
11:         isCaught ← TRUE
12:         counterTrapped ← 0
13:     end if
14:     return isCaught
15: end procedure
```

La función caótica utilizada es muy sensible a los cambios en las condiciones iniciales, y esta característica se utiliza para generar una perturbación a la solución para escapar de los óptimos locales. Entonces el caos o perturbación caótica es un proceso que se lleva a cabo para reiniciar la búsqueda desde otro punto del espacio de soluciones.

El algoritmo 6 se puede explicar en tres pasos. En primer lugar, se buscan las operaciones factibles, es decir aquellas operaciones que se pueden realizar sin violar ninguna restricción (línea 4). En segundo lugar, si solo hay una operación factible (línea 5) significa que es la última operación y es seleccionada (línea 6). Cuando hay más de una operación factible, se aplica una función caótica para seleccionar la que se agregará a la solución. En este caso, se utiliza la función logística (líneas 8-19), que aplica un

límite en el rango [0,5 a 1]. Finalmente, la operación seleccionada se agrega a la nueva solución (línea 21). Este proceso se aplica hasta que se seleccionan todas las operaciones.

Algorithm 6 Perturbación caótica

```
 1: procedure CHAOTICPERTURBATION(s_current)
 2:     feasibleTasksNumber ← 0, r ← 4, repeat ← TRUE, X_n ← 0, X_n1 ← 0
 3:     while counter < tasks do
 4:         feasibleTasksNumber ← searchFeasibleTasks()
 5:         if feasibleTasksNumber = 1 then
 6:             index ← 0
 7:         else
 8:             while repeat = TRUE do
 9:                 X_n ← random(0,1)
10:                 for i ← 0 to feasibleTasksNumber do
11:                     X_n1 ← (r × X_n) × (1.0 − X_n)
12:                     if X_n1 > 0.5 then
13:                         index ← i
14:                         repeat ← FALSE
15:                         break
16:                     end if
17:                     X_n ← X_n1
18:                 end for
19:             end while
20:         end if
21:         s_new ← addTask(index)
22:         counter ← counter + 1
23:     end while
24:     return s_new
25: end procedure
```

En el Algoritmo 7, la solución actual ($s_{current}$) se compara con la nueva solución (s_{new}) para determinar qué solución se usa para continuar la búsqueda. En esta comparación, hay tres casos:

1. Si s_{new} domina $s_{current}$, entonces s_{new} se usa para continuar la búsqueda (líneas 3 a 6).

2. Si s_{new} está dominado por $s_{current}$ entonces las diferencias de cada objetivo se calculan por separado de las dos soluciones comparadas para obtener el parámetro decremento (Δ) y usarlo para determinar si s_{new} continúa con la búsqueda de acuerdo con la condición en la línea 12. En este caso, $s_{current}$ se agrega al frente no dominado (F) y s_{new} reemplaza $s_{current}$ (líneas 13 y 14).

3. Si las dos soluciones no están dominadas entre sí, entonces la solución actual $s_{current}$ se agrega al frente no dominado (F), y la búsqueda continúa con s_{new} (líneas 18 a 21).

Algorithm 7 Verificación de Dominancia en CMOSA

1: **procedure** VERIFYDOMINANCECMOSA($T_k, s_{new}, s_{current}, mks_{new}, tds_{new}, flt_{new}, mks_{current}, tds_{current}, flt_{current}$)
2: $newDominateCurrent \leftarrow FALSE, currentDominateNew \leftarrow FALSE$
3: **if** $s_{new} \prec s_{current}$ **then**
4: $s_{current} \leftarrow s_{new}$
5: $newDominateCurrent \leftarrow TRUE$
6: **end if**
7: **if** $s_{current} \prec s_{new}$ **then**
8: $\Delta_{MKS} \leftarrow mks_{new} - mks_{current}$
9: $\Delta_{TDS} \leftarrow tds_{new} - tds_{current}$
10: $\Delta_{FLT} \leftarrow flt_{new} - flt_{current}$
11: $\Delta \leftarrow \Delta_{MKS} + \Delta_{TDS} + \Delta_{FLT}$
12: **if** $random(0,1) < e^{-\Delta/T_k}$ **then**
13: $F \leftarrow s_{current}$
14: $s_{current} \leftarrow s_{new}$
15: **end if**
16: $currentDominateNew \leftarrow TRUE$
17: **end if**
18: **if** $(newDominateCurrent = FALSE)$ AND $(currentDominateNew = FALSE)$ **then**
19: $F \leftarrow s_{current}$
20: $s_{current} \leftarrow s_{new}$
21: **end if**
22: **return** $s_{current}$
23: **end procedure**

4.2. Chaotic Multi-Objective Threshold Accepting (CMOTA)

El Algoritmo 8 muestra el algoritmo CMOTA, en el cual se observa que tiene una estructura similar que el algoritmo CMOSA.

Estos dos algoritmos tienen un ciclo de temperatura y, dentro de él, un ciclo de Metrópolis en el cual se aplica una perturbación a la solución actual. Después, se verifica el dominio de las dos soluciones con el objetivo de determinar cuál de ellas se usa para continuar el proceso de búsqueda (Algoritmo 9). Finalmente, se realiza el incremento de la variable que controla las iteraciones del ciclo de Metrópolis, la reducción de la temperatura y el incremento del contador para el número de temperaturas (línea 39).

En el algoritmo 9, se verifica el dominio de las dos soluciones para determinar cuál continúa con la búsqueda. Tiene los mismos tres casos usados en CMOSA (Algoritmo 7). Las principales diferencias

Algorithm 8 Chaotic Multi-Objective Threshold Accepting (CMOTA)

procedure CMOTA($T_{initial}, T_{final}, \alpha, \beta, L_k, s_{current}$)
  $counter \leftarrow 1, MAXSTAGNANT \leftarrow 10, counterTrapped \leftarrow 0, isCaught \leftarrow FALSE$
  $iterationsLocalSearch \leftarrow tasks \times timesLS, verifyCaught \leftarrow TRUE, countCaught \leftarrow 0$
  $mks_{current}, tds_{current}, flt_{current} \leftarrow calculateValues(s_{current})$ ▷ $mks: makespan, tds: tardiness, flt: flowtime$
  $T_k \leftarrow T_{initial}$
  while $T_k \geq T_{final}$ **do**
    for $i \leftarrow 0$ to L_k **do**
      if $isCaught = TRUE$ **then**
        $isCaught = FALSE$
        for $k \leftarrow 0$ to $iterationsLocalSearch$ **do**
          if $k = 0$ **then**
            $s_{new} \leftarrow chaoticPerturbation(s_{current})$ ▷ Ver Algoritmo 6
          else
            $s_{new} \leftarrow regularPerturbation(s_{current})$ ▷ Intercambio de dos operaciones
          end if
          $mks_{new}, tds_{new}, flt_{new} \leftarrow calculateValues(s_{new})$
          if $(mks_{new} < mks_{current})$ AND $(tds_{new} < tds_{current})$ AND $(flt_{new} < flt_{current})$ **then**
            $s_{current} \leftarrow s_{new}$
          end if
        end for
      else
        $s_{new} \leftarrow regularPerturbation(s_{current})$
        $mks_{new}, tds_{new}, flt_{new} \leftarrow calculateValues(s_{new})$
      end if
      if $(mks_{new} \neq mks_{current})$ AND $(tds_{new} \neq tds_{current})$ AND $(flt_{new} \neq flt_{current})$ **then**
        $verifyDominanceCMOTA(counter, T_k, s_{new}, s_{current})$ ▷ Ver Algoritmo 9
      end if
    end for
    if $verifyCaught = TRUE$ **then**
      if $caught(s_{current}, counterTrapped) = TRUE$ **then** ▷ Ver Algoritmo 5
        $countCaught = countCaught + 1$
        if $countCaught = MAXSTAGNANT$ **then**
          $verifyCaught \leftarrow FALSE$
        end if
      end if
    end if
    $L_k \leftarrow \beta \times L_k$
    $T_k \leftarrow \alpha \times T_k$
    $counter \leftarrow counter + 1$
  end while
  return $s_{current}$
end procedure

son las siguientes:

- Al principio, mientras que el contador de temperatura (*contador*) es menor que el valor del threshold (línea 4) T tiene un valor igual a T_k (línea 5), que es demasiado grande, lo que implica que a alta temperatura, la nueva solución (s_{new}) a menudo será aceptada para continuar la búsqueda. Es decir, durante el procesamiento del 95 % de temperaturas (parámetro $limite = 0.95$, cuyo valor se obtiene con la Ecuación 2.5.8 en el proceso de sintonización), se utiliza el parámetro γ para obtener el valor T (threshold), y dado que γ es igual a 1, entonces significa que T tiene el valor de T_k. Para el 5 % por ciento de las temperaturas restantes, γ toma el valor de $\gamma_{reduced}$ (0.978). Este parámetro se sintoniza experimentalmente (línea 12), y se establece para controlar el criterio de aceptación y hacerlo más restrictivo como parte del proceso.

- CMOTA incluye un proceso de verificación para aceptar una mala solución un poco diferente a CMOSA. Para determinar si el proceso de búsqueda continúa usando una solución dominada, CMOTA no usa el criterio de Boltzmann para aceptarla como la solución actual. En cambio, usa un threshold definido como el valor del parámetro T (línea 19), que se actualiza en la línea 29. En otras palabras, ya no es necesario calcular la disminución de las funciones objetivo. Esta modificación hace que CMOTA sea mucho más sencillo que CMOSA o cualquier otro algoritmo AMOSA. Además, debido a que el parámetro γ suele estar muy cerca de 1, no es necesario calcular las probabilidades para la distribución de Boltzmann o realizar un proceso de decisión aleatorio para las malas soluciones.

Algorithm 9 Verificación de Dominancia en CMOTA

procedure VERIFYDOMINANCECMOTA($counter, T_k, s_{new}, s_{current}$)
  $\gamma \leftarrow 1, \gamma_{reduced} \leftarrow 0.978, setT \leftarrow 1, bound \leftarrow NumberOfTemperatures \times limit$
  $newDominateCurrent \leftarrow FALSE$, $currentDominateNew \leftarrow FALSE$
  if $counter < bound$ **then**
    $T \leftarrow T_k$
  end if
  if $(counter = bound)$ AND $(setT = 1)$ **then**
    $setT \leftarrow 0$
    $T \leftarrow T_k$
  end if
  if $setT = 0$ **then**
    $\gamma \leftarrow \gamma_{reduced}$
  end if
  if $s_{new} \prec s_{current}$ **then**
    $s_{current} \leftarrow s_{new}$
    $newDominateCurrent \leftarrow TRUE$
  end if
  if $s_{current} \prec s_{new}$ **then**
    if $random(0,1) < T$ **then**
      $F \leftarrow s_{current}$
      $s_{current} \leftarrow s_{new}$
    end if
    $currentDominateNew \leftarrow TRUE$
  end if
  if $(newDominateCurrent = FALSE)$ AND $(currentDominateNew = FALSE)$ **then**
    $F \leftarrow s_{current}$
    $s_{current} \leftarrow s_{new}$
  end if
  $T \leftarrow T \times \gamma$
end procedure

4.3. Algoritmos Híbridos

Tres algoritmos híbridos basados en Scatter Search (Figura 4.1) son propuestos para la solución de MO-JSSP. Scatter Search es un algoritmo propuesto por Glover [52], y está compuesto por los siguientes métodos:

- ***Generador de soluciones diversas***. En este método se genera un conjunto P de soluciones aleatorias y diversas.

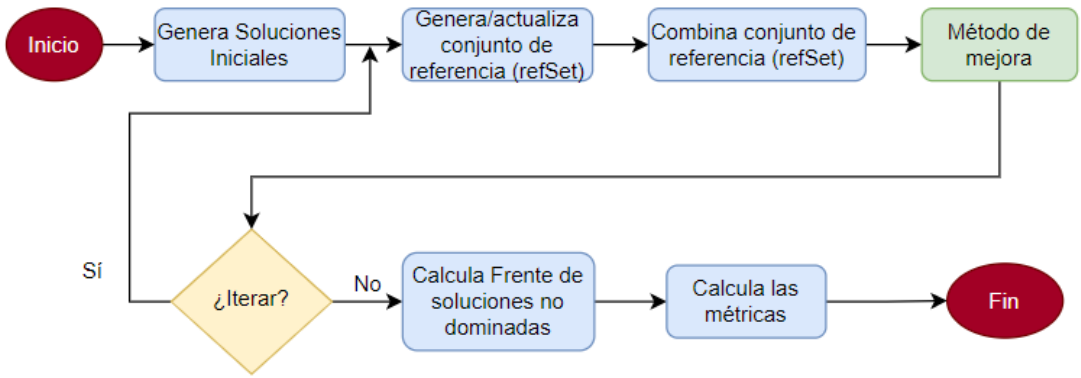

Figura 4.1: *Algoritmo Scatter Search*

- ***Actualizador y creador del conjunto de referencia***. Del total de soluciones P se seleccionan las tres primeras no dominadas y las tres más diversas, para formar un conjunto de referencia (refset) reducido. Para obtener estas últimas son elegidas las soluciones que tienen una distancia Euclidiana mayor.

- ***Combinación del conjunto de referencia***. Las soluciones que forman el conjunto de referencia se mezclan para obtener nuevas soluciones. En este proceso, todas las combinaciones posibles se generan. Para generar una nueva solución se toma la primera mitad de una solución del conjunto de referencia y la segunda mitad de otra.

- ***Método de mejora***. Es el proceso mediante el cual se intenta mejorar cada nueva solución creada por el método de combinación del conjunto de referencia. En este trabajo se implementan 3 métodos de mejora diferentes:

    1. Una Búsqueda Local (LS, por sus siglas en inglés de Local Search), que consiste en realizar un conjunto de iteraciones en las que se aplica una perturbación regular, que es un intercambio de dos operaciones en la solución actual para generar una nueva. En cada iteración se verifica la dominancia entre la solución actual y la nueva. Las nuevas soluciones no dominadas se almacenan en un arreglo.

    2. Un algoritmo Threshold Accepting Caótico (CMOTA, por sus siglas en inglés de Chaotic Multi-Objective Threshold Accepting). Threshold Accepting (TA) es un algoritmo propuesto en [63], en este método de mejora se utiliza una versión adaptada a JSSP [70].

3. Un algoritmo Chaotic Simulated Annealing (CMOSA, de Chaotic Multi-Objective Simulated Annealing). SA se propuso originalmente en [10] y en 2021 se implementa una nueva versión bajo el enfoque multiobjetivo en [70].

En los procesos de mejora 2 y 3 se realiza un proceso de sintonización analítica de los parámetros del algoritmo, este proceso está basado en [53]. También se aplica una perturbación regular para generar una nueva solución que se compara con la actual. De la comparación anterior se selecciona la solución no dominada y se descarta la dominada. En el caso de que ambas sean no dominadas, una se guarda en el conjunto de soluciones no dominadas y la otra continúa la búsqueda. En ambos algoritmos, si no se encuentran nuevas soluciones no dominadas, se aplica una perturbación caótica, que consiste en utilizar la ecuación de los mapas logísticos [55] como estrategia para escapar del estancamiento y buscar diversidad en las soluciones. Además también se aplica un proceso de recalentamiento, este consiste en elevar el valor del parámetro de temperatura actual para poder realizar una nueva exploración desde otro punto en el espacio de soluciones.

Los algoritmos de mejora implementados se describen con más detalle a continuación.

4.3.1. Algoritmo Híbrido Scatter Search con Local Search (SS/LS)

En este algoritmo mostrado en la Figura 4.2, se aplica una búsqueda local (LS), la cual consiste en realizar un conjunto de iteraciones, en cada una de ellas se aplica una perturbación regular a la solución actual, que consiste en el intercambio de dos operaciones, para generar una nueva. En cada una de las iteraciones se verifica la dominancia entre la solución actual y la solución nueva generada con la perturbación. En esta verificación se aplican los siguientes criterios de acuerdo al caso que ocurra:

1. Caso 1. Si la nueva solución domina la actual, entonces la nueva solución se guarda en un arreglo y reemplaza a la actual para continuar el proceso de búsqueda.

2. Caso 2. Si la solución actual domina a la nueva, se descarta la nueva solución.

3. Caso 3. Si la solución actual y la nueva son no dominadas, la nueva reemplaza a la actual para continuar la búsqueda. Ya no se guarda porque ya fue guardada en el caso 1.

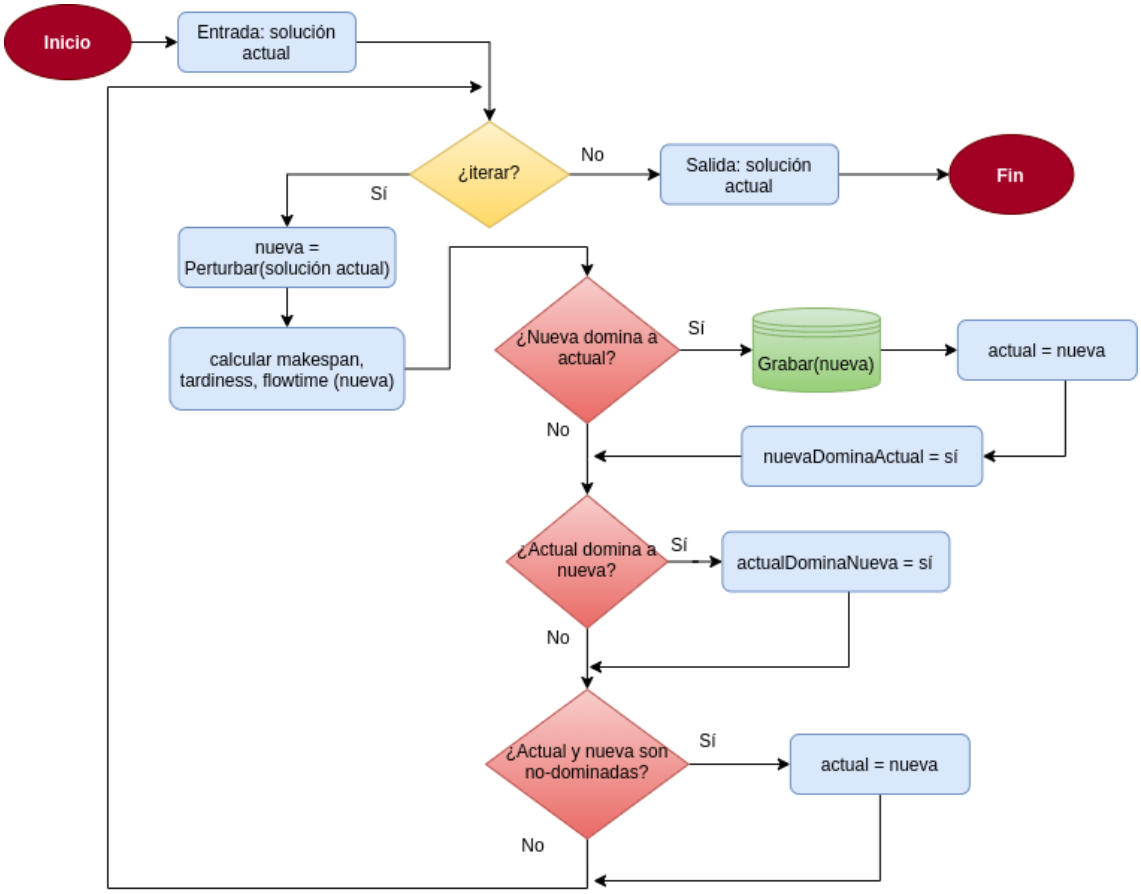

Figura 4.2: *Algoritmo híbrido Scatter Search/Local Search*

4.3.2. Algoritmo Híbrido Scatter Search con Chaotic Multi-Objective Simulated Annealing (SS/CMOSA)

Esta versión utiliza el algoritmo de Chaotic Multi-Objective Simulated Annealing (CMOSA) como método de mejora. Este algoritmo recibe como entrada una de las soluciones generadas en el método de combinación de las soluciones del refSet. En CMOSA existen dos ciclos, uno que controla la condición

de paro del algoritmo (ciclo de temperatura) y el otro ciclo interno (ciclo Metrópolis) que controla el número de iteraciones que se realizan para cada valor del parámetro de temperatura.

En el ciclo de Metrópolis, se hace una perturbación a la solución actual para generar una nueva. En cada iteración se verifica la dominancia entre la solución actual y la nueva. En esta verificación se aplican los mismos criterios para los 3 posibles casos previamente descritos. Los casos 1 y 3 se evalúan de la misma forma que la versión SS/LS. Para el caso 2, si la solución actual domina a la nueva, entonces la nueva solución tiene la oportunidad de reemplazar la solución actual y continuar la búsqueda. Este proceso se realiza utilizando la probabilidad de Boltzmann.

Adicionalmente, cuando se produce un estancamiento en la búsqueda; el cual se detecta al producirse una serie de iteraciones sin que se encuentren nuevas soluciones no dominadas, se aplica una perturbación caótica y un proceso de recalentamiento. La perturbación caótica consiste en utilizar la ecuación de los mapas logísticos [55], cuya principal característica es que incluso a pequeños cambios en sus datos de entrada genera salidas totalmente diferentes. Entonces el caos o perturbación caótica es un proceso que se lleva a cabo para reiniciar la búsqueda desde otro punto en el espacio de soluciones produciendo diversificación en el algoritmo. El recalentamiento es el proceso por el cual el parámetro de temperatura actual del algoritmo SA es elevado, esto ayuda a realizar un reprocesamiento en la búsqueda.

4.3.3. Algoritmo Híbrido Scatter Search con Chaotic Multi-Objective Threshold Accepting (SS/CMOTA)

En esta versión, el algoritmo Chaotic Multi-Objective Threshold Accepting (CMOTA) se utiliza como método de mejora. En CMOTA también existen dos ciclos como en CMOSA, uno que controla la condición de parada (ciclo de temperatura) y el otro interno (ciclo Metrópolis) que controla el número de iteraciones que se realizan para cada valor del parámetro de temperatura.

Se aplican los mismos criterios para los casos 1 y 3 descritos en el algoritmo SS/CMOSA. Mientras que para el caso 2, si la solución actual domina a la nueva, la nueva tiene la oportunidad de reemplazar la actual en la búsqueda utilizando un umbral establecido en el algoritmo.

Al igual que CMOSA, este algoritmo también aplica perturbaciones caóticas y un proceso de recalentamiento para escapar de óptimos locales.

4.4. Algoritmos Híbridos Paralelos con OpenMP

Los algoritmos híbridos que se realizaron en cómputo paralelo se implementaron utilizando lenguaje C con la tecnología Open Multi-Processing (OpenMP). Se detecta que en estos algoritmos el método de mejora requiere el mayor poder de cómputo ya que trata de realizar la mejora de cada una de las soluciones generadas. Tomando en cuenta esta consideración se decide realizar la paralelización de este método para tratar de reducir el tiempo de procesamiento de los algoritmos y evaluar los resultados.

4.4.1. Algoritmo Paralelo Scatter Search/Local Search (SS/LS)

El algoritmo que se toma como base para esta versión híbrida es Scatter Search (SS), y este tiene una fase en la cual se busca mejorar cada una de las soluciones. En esta versión la fase de mejora esta implementada con el algoritmo Local Search (algoritmo 4.2), el cual consta de un conjunto de iteraciones predefinidas. En base a que se tienen que realizar una serie de iteraciones, entonces se generan n cantidad de hilos con el objetivo de que procesen una solución a mejorar por cada hilo generado. En otras palabras, lo que se hace es que se generan n cantidad de algoritmos LS, uno por cada hilo generado.

La versión paralela del algoritmo Scatter Search/Local Search (SS/LS) se muestra en la Figura 4.3.

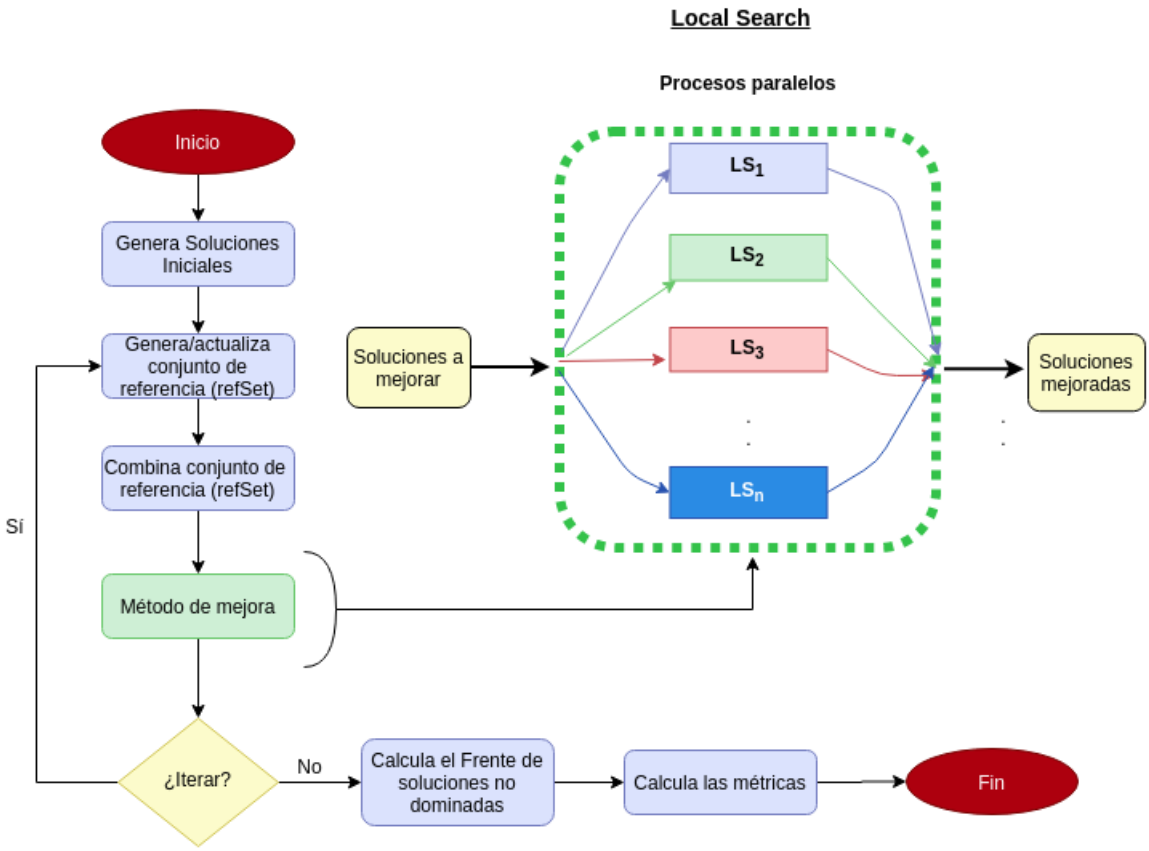

Figura 4.3: *Algoritmo Híbrido Paralelo Scatter Search/Local Search*

Como ya se ha explicado, las soluciones generadas en la fase de mejora se van almacenando en una arreglo de acuerdo a la dominancia que tiene. Al término del total de iteraciones que se realizan en la LS se aplica el Algoritmo 11 para encontrar las soluciones no dominadas, a las cuales se les calcularán las métricas para medir el desempeño del algoritmo.

4.4.2. Algoritmo Paralelo Scatter Search/Chaotic Multi-Objective Simulated Annealing (SS/CMOSA)

La versión paralela de Scatter Search/Chaotic Multi-Objective Simulated Annealing (SS/CMOSA) se muestra en la Figura 4.4.

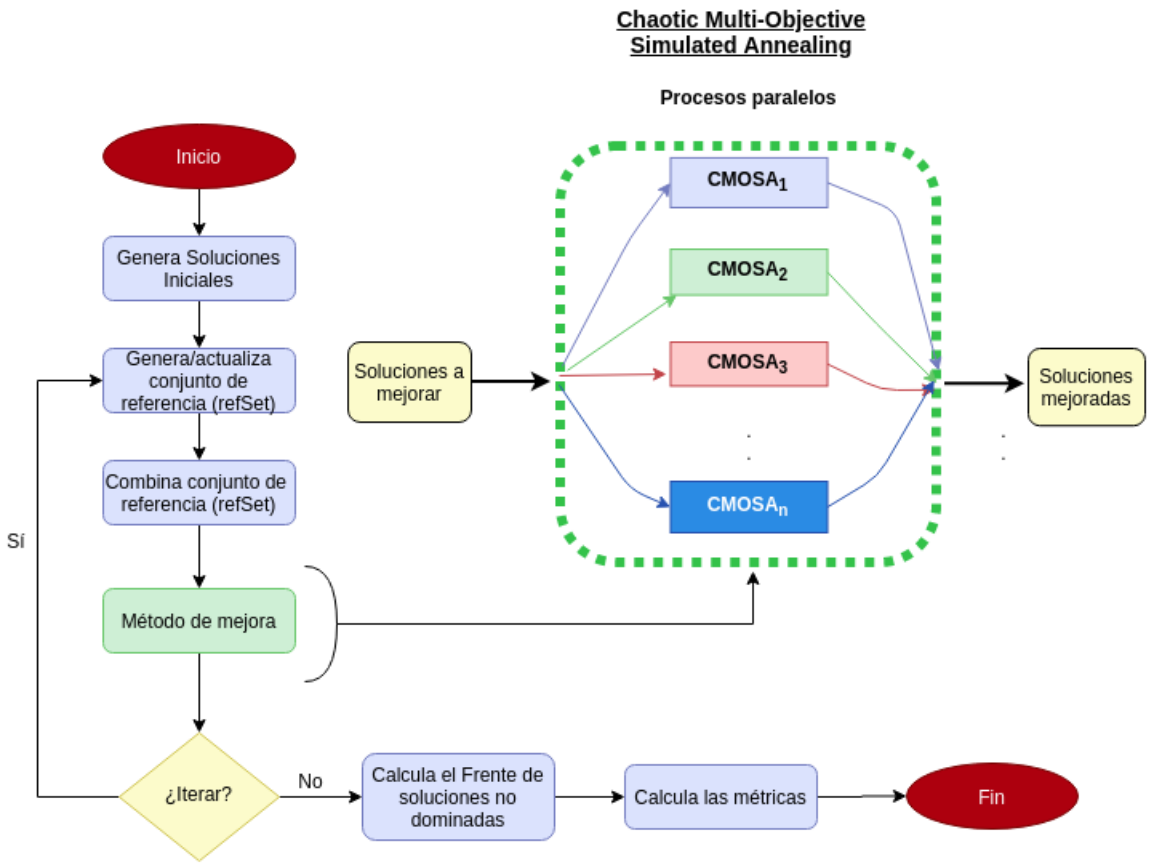

Figura 4.4: *Algoritmo Híbrido Paralelo Scatter Search/Local Search*

En este algoritmo al igual que el anterior se paraleliza el método de mejora. Cada solución seleccionada se procesa en un algoritmo CMOSA (ver Algoritmo 4), en otras palabras se generan *n* instancias del algoritmo CMOSA, una para cada solución a mejorar. Al finalizar el procesamiento de las soluciones se aplica el Algoritmo 11 para encontrar al frente de soluciones no dominadas y poder aplicar las métricas para medir el desempeño.

4.4.3. Instancias y parámetros utilizados

Para realizar la evaluación experimental de los algoritmos multi-objetivo a las 25 instancias que se utilizaron en mono-objetivo se le agregaron 45 instancias más para poder tener una mejor evaluación

con un total de 70 instancias. Se utilizaron 3 instancias (ft6, ft10 y ft20) de [3], 4 (yn1, yn2, yn3 y yn4) de [69], 5 (abz5 - abz9) de [1], 8 (ta01- ta71) de [12], 10 (orb1 - orb10) de [11] y 40 (la01 - la40) de [29].

Como ya se ha explicado, para realizar la sintonización analítica son necesarias algunas ejecuciones previas del algoritmo. Los parámetros usados para esas ejecuciones previas se muestran en la Tabla 4.1, y los parámetros usados en la experimentación final para cada instancia son se muestra en la Tabla 4.2.

Tabla 4.1: *Parámetros para las ejecuciones de sintonización de CMOSA/CMOTA*

Número de ejecuciones	**Temperatura inicial**	**Temperatura final**	α	L_k
50	100	0.1	0.98	100

Tabla 4.2: *Parámetros para la ejecución final de CMOSA/CMOTA*

Número de ejecuciones	**Soluciones iniciales**	α	**Núm. de estancamientos**
30	30	0.98	10

4.4.4. Algoritmo principal que ejecuta los métodos desarrollados

En el Algoritmo 10 se muestra el módulo principal desde el cual los dos algoritmos multi-objetivo (CMOSA y CMOTA) son llamados.

Algorithm 10 Algoritmo principal

```
1: S: Arreglo que guarda las soluciones iniciales
2: F: Arreglo que guarda las soluciones finales
3: leerInstancia()
4: sintonizacion_Analitica()
5: calcula_Duedate()
6: S ← CreaPoblacionInicial()
7: for each element i in S do
8:     s_old ← AlgoritmoAplicado(T_inicial, T_final, α, β, L_k, S_i)
9:     F_i ← s_old
10: end for
11: f_0 ← nonDominatedSorting(F)
12: calculaMetricas(f_0)
```

En este módulo principal, primero se lee la instancia a resolver (línea 3), luego se realiza el proceso de

sintonización analítica de parámetros (línea 4), se calcula la fecha de vencimiento o due date (línea 5), el cual es un elemento esencial para calcular la tardanza (*tardiness*), se genera cada una de las soluciones iniciales y se almacenan en el conjunto de soluciones iniciales (S) (línea 6), cada solución inicial se genera aleatoriamente de la siguiente manera: 1) se determina un conjunto de operaciones factibles, 2) luego una de ellas se selecciona aleatoriamente y 3) se agrega a la nueva solución. Este proceso se continúa hasta que todas las operaciones del trabajo son agregadas a la nueva solución generada.

Una vez que se ha generado el conjunto de soluciones iniciales, se aplica el algoritmo con el que se esté experimentando en ese momento (CMOSA o CMOTA) para mejorar cada solución inicial (línea 8), las soluciones finales generadas se almacenan en un arreglo de soluciones finales (F). A este conjunto de soluciones finales F se le aplica el algoritmo *fast non-dominated sorting* (línea 11) publicado en [58] (Algoritmo 11) para obtener el conjunto de soluciones no dominadas, también llamado frente cero (f_0).

Algorithm 11 Fast non-dominated Sort (P)

```
1:  for each element p ∈ P do
2:      for each element q ∈ P do
3:          if (p ≺ q) then                          ▷ If p dominates to q
4:              S_p ← S_p ∪ {q}                      ▷ Add q in S_p
5:          elseIf (q ≺ p)                           ▷ If p is dominated byq
6:              n_p ← n_p + 1                        ▷ Increase n_p
7:          end if
8:      end for
9:      if (n_p = 0) then                            ▷ If none dominates p
10:         F_1 = F_1 ∪ {p}                          ▷ p is a member of the front 1
11:     end if
12:     i = 1
13: end for
14: while F_i ≠ 0 do
15:     H = 0
16:     for each p ∈ F_i do                          ▷ For each member p in F_i
17:         for each q ∈ S_p do                      ▷ for each member of S_p
18:             n_q = n_q − 1                        ▷ Decrements n_q
19:             if n_q = 0 then
20:                 H = H ∪ {q}                      ▷ q is in the list H
21:             end if
22:         end for
23:         i = i + 1
24:         F_i = H                                  ▷ The members of H are the front i
25:     end for
26: end while
```

Para conocer la calidad del conjunto de soluciones no dominadas obtenido se aplican las métricas de desempeño MID, spacing, hipervolumen, spread, distancia generacional invertida (DGI) y coverage (línea 12).

Para realizar el cálculo de las métricas spread e IGD, es necesario conocer el frente de Pareto real (FP_{real}). Sin embargo, para las instancias utilizadas no se conoce este FP_{real}; por esta razón, el cálculo se realizó utilizando un frente de Pareto aproximado (FP_{approx}), que es obtenido de la unión de los frentes calculados con ejecuciones previas de los dos algoritmos aquí presentados (CMOSA y CMOTA).

CAPÍTULO 5

Análisis y Resultados

En este capítulo se presentan los resultados obtenidos en la evaluación experimental de cada uno de los algoritmos multi-objetivos implementados. Para cada tabla se presenta una descripción de la misma, así como un análisis detallado de los resultados obtenidos.

5.1. Recocido Simulado Multi-Objetivo Caótico (CMOSA)

La Tabla 5.1 muestra los resultados obtenidos para cada una de las 70 instancias utilizadas con el algoritmo CMOSA de las métricas MID, Spacing, Spread, HV y DGI. Adicionalmente en la última fila de la tabla se muestra el resultado promedio obtenido para cada métrica.

Tabla 5.1: *Métricas obtenidas con los resultados del algoritmo CMOSA con 70 instancias*

-	CMOSA				
Instance	MID	Spacing	Spread	HV	DGI
MT06	302.49	283.51	264.83	0.7694	1.81
MT10	8968.94	8013.63	6882.02	0.515	176.11
MT05	18599.1	16523.19	13850.46	0.1551	1086.3
ORB1	9423.07	8631.52	7506.09	0.5215	315
ORB2	8129.62	7302.86	6484.97	0.0757	58.31
ORB3	9443.81	8811.21	8008.17	0.3396	215.58
ORB4	9205.69	8684.37	8107.36	0.3019	102.46
ORB5	7983.66	7631.44	7160.68	0.7722	190.14
ORB6	9489.54	8992.36	8334	0.6094	268.78
ORB7	3630.48	3457.53	3259.98	0.4625	39.48
ORB8	8268.95	7450	6570.9	0.3608	190.56
ORB9	8778.61	8181.12	7523.24	0.3863	160.34
ORB10	8672.67	7905.48	7105.78	0.3066	99.46
LA01	5412.62	5185.85	4935.68	0.7205	43.7
LA02	5031.77	4796.26	4545.65	0.7237	50.36
LA03	4814.76	4671.64	4498.78	0.506	23.82
LA04	4861.93	4721.93	4568.1	0.4608	33.59
LA05	4562.81	4207.89	3830.65	0.6004	63.32
LA06	10838.09	9825.05	8500.62	0.1572	370.68
LA07	10116.11	9403.63	8492.96	0.8145	281.37
LA08	10042.56	9400.77	8591.76	0.1512	378.37
LA09	11036.3	9534.05	7664.43	0.2407	460.16
LA10	11202.9	10168.45	8836.71	0.6113	345.61
LA11	19027.38	17822.14	16036.39	0.4063	1043.36
LA12	15911.86	14478.37	12307.3	0.7123	872.88
LA13	17928.81	17028.97	15813.07	0.394	425.64
LA14	20538.67	18317.24	14785.44	0.0202	2073.23
LA15	19316.53	18535.29	17521.86	0.4315	675.91
LA16	8471.45	8130.28	7769.64	0.5119	79.86
LA17	7360.64	7180.51	6992.07	0.4976	30.61
LA18	7799.15	7548.58	7269.67	0.3886	75.14
LA19	7886.52	7327.72	6707.22	0.4576	129.53
LA20	8223.4	7924.39	7610.55	0.5595	61.18
LA21	14660.24	14258.83	13771.57	0.6037	267.58
LA22	13791.82	13255.87	12589.96	0.3012	201.05
LA23	14332.13	13337.73	12141.51	0.131	281.22
LA24	13621.56	12311.51	10795.91	0.077	248.51
LA25	14072.76	13593.37	13043.24	0.372	368.47
LA26	23328.49	21997.87	20257.12	0.229	699.68
LA27	23562.7	20950.37	18001.02	0.3868	548.52
LA28	23470.98	22503.14	21173.67	0.4783	631.41
LA29	23693.39	22837.5	21581.92	0.2558	635.35
LA30	25644.07	22372.92	18169.69	0.9407	1274.46
LA31	47688.21	43286	36916.42	0.1622	2058.47
LA32	49824.88	45377.73	39456.58	0.2456	2759.99
LA33	45505.39	39517.26	32519.4	0.8072	2552.7
LA34	48515.97	46689.13	44230.39	0.6089	1745.72
LA35	51334.25	48621.43	44009.29	0.8587	2773.3
LA36	20064.09	19364.09	18363.96	0.3586	767.11
LA37	20914.82	20389.58	19605.69	0.564	742.55
LA38	18259.69	16843.64	14733.81	0.2999	851.28
LA39	18883.47	17177.13	14912.4	0.6931	556.91
LA40	18713.13	16658.15	13996.54	0.6702	603.64
ABZ5	11065.38	9874.59	8721.64	0.1769	128.88
ABZ6	8562.26	8245.76	7901.07	0.4466	43.59
ABZ7	13456.23	10953.49	8177.11	0.0156	483.15
ABZ8	13876.7	12637.72	10869.79	0.0662	582.66
ABZ9	14196.03	13608.31	12669.93	0.6762	789.14
YN01	19600.96	18536.35	16769.17	0.6926	863.36
YN02	19478.97	17754.4	15269.85	0.4691	787.7
YN03	19373.68	17872.37	15509.6	0.0611	714.06
YN04	20795.84	17923.77	13887.37	0.6295	1005.78
TA01	18854.25	17145.36	14656.25	0.2636	722.21
TA11	28456.37	26229.2	23056.89	0.5516	1414.59
TA21	36784.31	33753.51	29020.96	0.3959	2026.32
TA31	57276.28	54786.08	50318.03	0.1867	2786.37
TA41	67727.19	63276.62	55584.15	0.1938	3400.56
TA51	149060.96	140602.59	126018.51	0.5074	9183.15
TA61	163794.79	141020.36	105655.14	0.3122	9838.71
TA71	634090.21	597552.21	525159.37	0.1104	50876.36
PROMEDIO	30680.19	28445.62	24969.31	0.42	1666.25

5.2. Threshold Accepting Multi-Objetivo Caótico (CMOTA)

La Tabla 5.2 muestra los resultados obtenidos para cada una de las 70 instancias utilizadas con el algoritmo CMOTA de las métricas MID, Spacing, Spread, HV y DGI. Adicionalmente en la última fila de la tabla se muestra el resultado promedio obtenido para cada métrica.

En la Tabla 5.3 se muestran solo los resultados promedio de los algoritmos CMOSA y CMOTA para una mejor observación de los mismos. Se observa que CMOSA obtiene los mejores valores para las métricas MID e IGD. Por otro lado para Spacing y Spread el algoritmo CMOTA obtiene un mejor desempeño. Y finalmente para la métrica HV, ambos algoritmos logran el mismo resultado (0,42). Se observa en la Tabla 5.4 que el algoritmo CMOSA obtiene un mejor resultado para la métrica coverage que CMOTA. Adicionalmente se realizó una prueba de Wilcoxon de dos colas con un nivel de significancia de 5 % cuyo resultado se muestra en la última columna de la Tabla 5.3 que demuestra que hay diferencias significativas entre CMOSA y CMOTA únicamente para las métricas MID e IGD.

Tabla 5.3: *Resumen de resultados obtenidos con CMOSA y CMOTA*

Métrica	**CMOSA**	**CMOTA**	**Diferencia significativa CMOSA-CMOTA**
MID	30680.19*	31233.15	Sí
SPACING	28445.62	28183.17*	No
SPREAD	24969.31	23401.88*	No
HV	0.42*	0.42*	No
IGD	1666.25*	1870.94	Sí

5.3. CMOSA vs CMOTA vs AMOSA

En la Tabla 5.5 se muestra la comparación de los resultados entre los algoritmos CMOSA y AMOSA. En esta tabla se muestran los valores de los resultados promedio para las 70 instancias resueltas. Se observa que CMOSA obtiene un mejor desempeño para todas las métricas evaluadas. Además, la

Tabla 5.2: *Métricas obtenidas con los resultados del algoritmo CMOTA con 70 instancias*

-	CMOTA				
Instance	MID	Spacing	Spread	HV	DGI
MT06	302.76	277.68	252.61	0.3777	2.18
MT10	9316.19	8755.98	7915.06	0.3059	301.95
MT05	18789.48	17739.36	16229.92	0.8048	1212.34
ORB1	9707.17	9180.13	8393.1	0.2747	328.3
ORB2	8526.49	8123.73	7624.59	0.6586	172.8
ORB3	9940.47	9111.26	8028.23	0.3623	315.26
ORB4	9874.42	9566.95	9158.45	0.5113	209.3
ORB5	8088.73	7176.28	5961.18	0.5062	250.13
ORB6	9326.38	8515.34	7529.9	0.4958	225.14
ORB7	3758.8	3378.73	2941.73	0.1496	58.53
ORB8	8562.67	7770.63	6673.16	0.6641	470.7
ORB9	9276.51	8809.53	8168.91	0.4169	346.15
ORB10	9001.91	7994.49	6874.12	0.7694	177.82
LA01	5597.42	5319.14	4996.53	0.5803	58.3
LA02	5184.33	4110.92	3146.39	0.0107	60.94
LA03	5134.95	4660.7	4163.96	0.5912	72.29
LA04	5031.3	4588.27	4085.77	0.7817	69.29
LA05	4714.16	4551.57	4349.3	0.499	103.39
LA06	10892.8	9660.2	7995.8	0.4286	463.07
LA07	10667.92	9002.08	7018.68	0.3067	400.5
LA08	10009.01	8044.14	5990.05	0.0495	404.68
LA09	11411.96	9788.51	7713.8	0.0142	560.81
LA10	11365.93	10108.79	8447.86	0.7471	415.17
LA11	19802.15	15870.98	10962.35	0.1347	1417.7
LA12	16330.94	14099.14	10900.69	0.0399	985.85
LA13	18139.11	16446.49	14433.28	0.1931	351.38
LA14	20433.48	17631.83	13506.69	0.0509	1989.23
LA15	20217.82	18346.98	15768.77	0.0473	1011.78
LA16	8504.98	7526.93	6536.56	0.2266	134.79
LA17	7642.17	7353.04	7017.62	0.8034	93.06
LA18	7970.16	7394.99	6705.28	0.2977	119.49
LA19	8018.64	7376.36	6656.98	0.6235	146.92
LA20	8504.94	7996.42	7431.91	0.3388	106.52
LA21	15048.78	14457.33	13647.14	0.5932	410.3
LA22	14273.02	12875.98	11072.54	0.1695	320.01
LA23	14681.08	13875.84	12713.71	0.379	512.16
LA24	14220.91	13419.12	12272.33	0.4233	416.13
LA25	14339.68	13188.22	11819.89	0.4927	543.33
LA26	23931.83	22264.13	19821.01	0.6662	1074.32
LA27	24858.52	24019.5	22789.89	0.701	969.43
LA28	24011.52	22843.64	21132.67	0.394	826.36
LA29	24403.19	23508.56	22051.6	0.4593	827.97
LA30	25928.94	24625.34	22669.67	0.5402	1215.65
LA31	49005.79	43461.59	35340.2	0.194	2482.67
LA32	51503.92	49408.53	46060.5	0.6294	3773.47
LA33	48241.29	46873.23	44690.77	0.2373	3673.72
LA34	50594.96	48113.22	44152.41	0.3961	2668.97
LA35	52684.38	48642.88	41461.6	0.8904	3244.22
LA36	20496.35	19567.32	18118.3	0.1703	986.53
LA37	21563.64	20552.28	18883.76	0.5058	1075.89
LA38	18899.71	16219.84	12597.32	0.3385	964.25
LA39	19664.49	17874.61	15082.6	0.4429	782.28
LA40	19548.43	15827.01	10869.3	0.1123	944.43
ABZ5	11716.44	11287.22	10776.43	0.5024	319.09
ABZ6	8777.87	8140.27	7409.23	0.5505	101.42
ABZ7	14117.93	12416.68	10162.27	0.3996	661.08
ABZ8	14964.84	13616.29	11477.82	0.4527	830.89
ABZ9	14482.01	12794.37	10240.33	0.3847	931.42
YN01	19081.94	17202.55	14571.94	0.2167	735.6
YN02	29387.6	27094.14	23716.57	0.6778	1553.09
YN03	37958.48	36336.82	33395.43	0.5299	2382.03
YN04	20085.05	18289.67	15477	0.3315	1037.34
TA01	21077.84	19800.18	17571.85	0.6004	1219.32
TA11	19829.68	17957.25	14965.13	0.4053	860.2
TA21	21706.29	20446.68	18328.94	0.6616	1174.04
TA31	58910.97	55471	49003.81	0.2449	3379.9
TA41	70740.02	64443.42	53119.54	0.1073	4276.5
TA51	147166.01	131401.13	106469.11	0.4512	8838.17
TA61	165040.63	150244.07	124877.89	0.3191	9514.26
TA71	633330.29	547984.53	405741.83	0.5441	52403.3
PROMEDIO	31233.15	28183.17	23401.88	0.42	1870.94

Tabla 5.4: *Resultados obtenidos por la métrica coverage*

Coverage(CMOSA,CMOTA)	Coverage(CMOTA,CMOSA)
0.854*	0.063

prueba de Wilcoxon indica que en 3 métricas existe diferencia significativa entre los algoritmos en favor de CMOSA.

Tabla 5.5: *Comparación entre CMOSA y AMOSA*

Metrica	**CMOSA**	**AMOSA** [62]	**Diferencia significativa CMOSA-AMOSA**
MID	30680.19*	32138.19	Yes
SPACING	28445.62*	30129.36	Yes
SPREAD	24969.31*	26625.04	No
HV	0.42*	0.37	No
IGD	1666.25*	2209.96	Yes

De igual forma se comparan los resultados promedio de los algoritmos CMOTA y AMOSA. Los resultados se muestran en la Tabla 5.6. En este caso, CMOTA también obtiene los mejores resultados promedio en todas las métricas empleadas; sin embargo, de acuerdo con la prueba de Wilcoxon, existen diferencias significativas en solo dos, spacing e IGD.

Tabla 5.6: *Comparación entre CMOTA y AMOSA*

Métrica	**CMOTA**	**AMOSA** [62]	**Diferencia significativa CMOTA-AMOSA**
MID	31233.15*	32138.19	No
SPACING	28183.17*	30129.36	Sí
SPREAD	23401.88*	26625.04	No
HV	0.42*	0.37	No
IGD	1870.94*	2209.96	Sí

5.4. CMOSA vs CMOTA vs IMOEA/D

En la Tabla 5.7 se comparan los resultados de CMOSA y CMOTA con los resultados obtenidos con algoritmo IMOEA/D usando las 58 instancias comunes publicadas en [23], en este trabajo se utiliza la métrica MID para medir la calidad de las soluciones. Esta tabla muestra el valor promedio de la métrica MID para el conjunto no dominado de soluciones de CMOSA y CMOTA. Los resultados muestran que CMOSA y CMOTA obtienen mejor desempeño que IMOEA/D, pues ambos algoritmos, CMOSA y CMOTA, lograron valores MID más pequeños, lo que indica que los frentes de Pareto de

los algoritmos propuestos se encuentran más cerca del punto de referencia (0,0,0). Adicionalmente la prueba de Wilcoxon confirma los resultados ya descritos.

Tabla 5.7: *Resultados de CMOSA, CMOTA e IMOEA/D para la métrica MID*

CMOSA	**CMOTA**	**IMOEA/D** [23]	**Diferencia significativa CMOSA-IMOEA/D**	**Diferencia significativa CMOTA-IMOEA/D**
15729.65*	16567.07	18727.04	Sí	Sí

5.5. CMOSA Vs CMOTA vs SPEA/CMOEA/MOPSO y MOMARLA

Finalmente, los algoritmos propuestos se adaptaron y ejecutaron para solo dos funciones objetivo (makespan y total tardiness) con el fin de evaluar su desempeño y comparar los resultados con los algoritmos SPEA, CMOEA, MOPSO y MOMARLA; para esto se utiliza la métrica HV, y se experimenta con las mismas 15 instancias de referencia usadas en [38].

Los resultados se muestran en la Tabla 5.8, en ella se observa que MOMARLA supera a SPEA, CMOEA y MOPSO. Se observa también que el algoritmo CMOSA tiene un mejor desempeño que MOMARLA y los demása algoritmos. Al comparar CMOTA y MOMARLA, notamos que ambos algoritmos obtuvieron los mismos resultados promedio para la métrica HV utilizada.

5.6. Algoritmos Híbridos SS/LS vs SS/CMOSA vs SS/CMOTA

Estos algoritmos híbridos propuestos se ejecutan 30 veces. Los resultados promedio para las 30 ejecuciones se muestran en la Tabla 5.9. En esta tabla, la primera fila presenta los resultados para el algoritmo SS usando Local Search, después los resultados de CMOSA y CMOTA posteriormente. Las dos últimas filas muestran los mejores resultados del estado del arte para estos mismos objetivos. Los mejores valores encontrados para cada métrica están marcados con un asterisco (*). Para aquellas

Tabla 5.8: *Comparación entre SPEA, CMOEA, MOPSO, MOMARALA, CMOSA y CMOTA con la métrica HV*

	Instancia	**SPEA** [38]	**CMOEA** [38]	**MOPSO** [38]	**MOMARLA** [38]	**CMOSA**	**CMOTA**
1	FT06	0.07	0.07	0.50	0.65	0.64	0.75*
2	FT10	0.17	0.26	0.87	0.96	0.71	0.69
3	FT20	0.20	0.20	0.21	0.25	0.57*	0.77*
4	ABZ5	0.34	0.33	0.36	0.40	0.85*	0.56*
5	ABZ6	0.22	0.36	0.31	0.42	0.60*	0.81*
6	ABZ7	0.51	0.45	1.00	1.00	0.79	0.51
7	ABZ8	0.88	0.36	0.99	0.99	0.69	0.66
8	LA26	0.33	0.39	0.47	0.47	0.91*	0.70*
9	LA27	0.58	0.56	0.41	0.60	0.71*	0.93*
10	LA28	0.48	0.42	0.48	0.54	0.92*	0.44
11	ORB01	0.62	0.74	0.59	0.80	0.87*	0.63
12	ORB02	0.20	0.04	0.30	0.53	0.88*	0.77*
13	ORB03	0.69	0.31	0.85	0.86	0.76	0.80
14	ORB04	0.63	0.28	0.52	0.79	0.76	0.81*
15	ORB05	0.00	0.023	0.22	0.90	0.74	0.32
	Promedio HV	0.39	0.32	0.54	0.68	0.76*	0.68

métricas en las que es necesario utilizar un frente real (PF_{true}), se usa un frente aproximado (PF_{aprox}). Este frente es generado por todos algoritmos implementados a lo largo del estudio de esta problema.

Tabla 5.9: *Comparación algoritmos híbridos*

Algoritmo	**MID**	**SPACING**	**HV**	**SPREAD**	**DGI**	**No. Soluciones**	**Tiempo ejecución**
SS/LS	29727.69 *	27345.38	0.42 *	23935.95	1381.14*	9.51	133.65*
SS/CMOSA	29986.85	27165.00*	0.42*	22533.70*	1453.10	8.79	1210.57
SS/CMOTA	29899.32	27577.10	0.41	24330.25	1430.52	8.23	1243.90
CMOSA	30680.19	28445.62	0.42*	24969.31	1666.25	10.57*	495.22
CMOTA	31233.15	28183.17	0.42*	23401.88	1870.94	8.66	229.42

Se observa que SS/LOCAL SEARCH obtiene el mejor resultado para la métrica MID y DG, además utilza el menor tiempo de procesamiento. En el caso de SS/CMOSA obtiene el mejor resultado en la métrica spacing y spread. El algoritmo SS/CMOTA no obtiene el mejor resultado en ninguna métrica sin embargo en MID y DGI supera a SS/CMOSA. Por otro lado para el número promedio de soluciones generadas, nuestros algoritmos propuestos no lograron superar al CMOSA del estado de la arte. Finalmente, en la métrica HV todos los algoritmos, a excepción de SS/CMOTA obtuvieron el

mismo valor con 0.42.

5.7. Algoritmos Híbridos Paralelos SS/LS y SS/CMOSA

La Tabla 5.10 muestra los resultados obtenidos en la experimentación del algoritmo SS/LS. La primer fila muestra los resultados para la versión secuencial y la segunda para la versión paralela.

Tabla 5.10: *Comparación algoritmos híbridos paralelos SS/LS*

Algoritmo	MID	SPACING	HV	SPREAD	DGI	No. Soluciones	Tiempo ejecución
SS/LS	29727.69	27345.38	0.42*	23935.95	1381.14*	9.51*	133.65
SS/LS paralelo	29345.82*	27239.56*	0.40	22036.49*	1644.30	7.81	77.21*

Se observa que la versión paralela obtiene mejores resultados en MID, spacing, spread mientras que la versión secuencial lo hace en HV, DGI y obtiene la mayor cantidad promedio de soluciones en el frente.

Por otro lado, la Tabla 5.11 muestra los resultados obtenidos en la experimentación del algoritmo SS/CMOSA. La primer fila muestra los resultados para la versión secuencial y la segunda para la versión paralela.

Tabla 5.11: *Comparación algoritmos híbridos paralelos SS/CMOSA*

Algoritmo	MID	SPACING	HV	SPREAD	DGI	No. Soluciones	Tiempo ejecución
SS/CMOSA	29986.85	27165.00*	0.42*	22533.70*	1453.10*	8.79	1210.57
SS/CMOSA paralelo	29295.81*	27484.86	0.41	24695.37	1728.36	9.54*	813.46*

En este algoritmo la versión secuencial obtiene mejor desempeño en spacing, HV spread y DGI, mientras que la versión paralela obtiene mejores resultados en la métrica MID y cantidad de soluciones promedio.

El objetivo de la implementación paralela es reducir el tiempo de ejecución de los algoritmos y como se observa en la Tabla 5.12 en el algoritmo híbrido SS/LS el tiempo de procesamiento se redujo en un

42.23 % mientras que en algoritmo híbrido SS/CMOSA se redujo en 32.80 %.

Tabla 5.12: *Tiempo de procesamiento de los Algoritmos Híbridos Paralelos SS/LS y SS/CMOSA*

Algoritmo	**Secuencial (s)**	**Paralelo (s)**	**Reducción**
SS/LS	133.65	77.21	42.23 %
SS/CMOSA	1210.57	813.36	32.80 %

Lo anterior demuestra que aplicar cómputo paralelo en los procesos que mayor cantidad de operaciones realizan logra reducir el tiempo de procesamiento de los algoritmos.

CAPÍTULO 6

Conclusiones y Trabajos Futuros

6.1. Conclusiones

En el presente trabajo se soluciona el problema Job Shop Scheduling mono-objetivo (makespan) con 25 instancias utilizando el algoritmo Simulated Annealing. Se desarrollan versiones con sintonización manual, sintonización analítica, usando equilibrio estocástico, aplicando reheat (recalentamiento) y procesos de perturbación caótica. La versión que aplica recalentamiento obtiene logra reducir el ER promedio a 3.8 empleando un tiempo de procesamiento promedio de 58.6 segundos, mientras que al añadir la perturbación caótica el ER promedio promedio se reduce a 2.8 sin embargo su tiempo de procesamiento se eleva a 149.1 segundos.

Se desarrollan 2 algoritmos (CMOSA y CMOTA) para la versión multi-objetivo y se experimenta con 70 instancias. CMOSA obtiene los mejores valores para las métricas MID e IGD, sin embargo

para Spacing y Spread el algoritmo CMOTA lo supera. En la métrica HV, ambos algoritmos logran el mismo resultado (0,42). En la prueba de Wilcoxon de dos colas con un nivel de significancia de 5 % el resultado muestra que hay diferencias significativas entre CMOSA y CMOTA únicamente para las métricas MID e IGD.

En la comparación que se realiza con el único algoritmo del estado del arte que aplica métricas para medir su desempeño y que utiliza los mismos 3 objetivos (IMOEA/D) se demuestra que CMOSA y CMOTA obtienen mejor desempeño que el IMOEA/D, pues ambos algoritmos, lograron valores MID más pequeños. Además la prueba de Wilcoxon confirmó estos resultados.

Se presenta también una solución a la versión multi-objetivo con 2 objetivos (makespan, total tardiness) aplicando CMOSA y CMOTA. Esta versión se desarrolla para hacer una comparación con algoritmos del estado del arte desarrollados recientemente (SPEA, CMOEA, MOPSO y MOMARLA). Se emplean 15 instancias y se compara el resultado de la métrica HV. Los resultados demuestran que el algoritmo CMOSA tiene un mejor desempeño que SPEA, CMOE, MOPSO y MOMARLA. Al comparar CMOTA y MOMARLA, notamos que ambos algoritmos obtuvieron los mismos resultados promedio.

Los resultados obtenidos superan a los mejores algoritmos del estado del arte y tienen una complejidad menor en algunos casos.

Además se proponen 3 versiones híbridas utilizando el algoritmo Scatter Search como base y como método de mejora se aplican los algoritmos Local Search, CMOSA y CMOTA. Estas versiones híbridas propuestas obtienen resultados que superan a los algoritmos antes mostrados CMOSA y CMOTA.

Por último se implementan dos versiones en cómputo paralelo SS/LS y SS/CMOSA. Se observa que el desempeño de las métricas es muy similar en casi todas y que las versiones paralelas reducen el tiempo de procesamiento de la versión secuencial. Para el caso de SS/LS el tiempo de procesamiento se logra reducir un 42.23 % y para la SS/CMOSA un 32.80 %.

6.2. Trabajos Futuros

Como trabajo futuro se planea la escritura de un artículo con los resultados de estos algoritmos híbridos propuestos, así como la realización de otro artículo con los resultados de los algoritmos paralelos aquí aplicados.

6.3. Publicaciones derivadas del presente trabajo

- Chaotic Multi-Objective Simulated Annealing and Threshold Accepting for Job Shop Scheduling Problem.

Bibliografía

[1] J. Adams, E. Balas, and D. Zawack, "The shifting bottleneck procedure for job shop scheduling," *Management Science*, vol. 34, no. 3, pp. 391–401, 1988.

[2] J. Carlier and E. Pinson, "An algorithm for solving job shop problem," *Management Science*, vol. 35, pp. 164–176, 02 1989.

[3] H. Fisher and G. L. Thompson, "Probabilistic learning combinations of local job-shop scheduling rules," *Industrial Scheduling*, vol. 1, no. 2, pp. 225–251, 1963.

[4] E. Demirkol, S. Mehta, and R. Uzsoy, "A computational study of shifting bottleneck procedures for shop scheduling problems," *J. Heuristics*, vol. 137, pp. 111–137, 1997.

[5] C. Y. Zhang, P. Li, Y. Rao, and Z. Guan, "A very fast TS/SA algorithm for the job shop scheduling problem," *Computers & Operations Research*, vol. 35, no. 1, p. 282–294, 2008.

[6] M. Pinedo, *Scheduling: Theory, Algorithms, And Systems*, 01 2008.

[7] M. R. Garey, D. S. Johnson, and R. Sethi, "PageRank: the complexity of flowshop and jobshop scheduling," *Mathematics of Operations & Research*, vol. 1, no. 2, pp. 117–129, 1976.

[8] B. Giffler and G. L. Thompson, "Algorithms for solving production-scheduling problems," *Oper. Res.*, vol. 8, no. 4, p. 487–503, aug 1960. [Online]. Available: https://doi.org/10.1287/opre.8.4.487

[9] J. Holland, *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. University of Michigan Press, 1975. [Online]. Available: https://books.google.com.mx/books?id=JE5RAAAAMAAJ

[10] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *American Association for the Advancement of Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[11] D. Applegate and W. Cook, "A computational study of the job-shop scheduling problem," *ORSA Journal on Computing*, vol. 3, no. 2, pp. 149–156, 1991.

[12] E. Taillard, "Benchmarks for basic scheduling problems," *European Journal of Operational Research*, vol. 64, pp. 278–285, 1993.

[13] P. Brucker, B. Jurisch, and B. Sievers, "A branch and bound algorithm for the job-shop scheduling problem," *Discret. Appl. Math.*, vol. 49, pp. 107–127, 1994.

[14] P. Martin and D. Shmoys, "A new approach to computing optimal schedules for the job-shop scheduling problem," 1996.

[15] J. Carlier and E. Pinson, "Adjustment of heads and tails for the job-shop problem," *European Journal of Operational Research*, vol. 78, no. 2, pp. 146–161, 1994. [Online]. Available: https://EconPapers.repec.org/RePEc:eee:ejores:v:78:y:1994:i:2:p:146-161

[16] W. Bożejko, *A new class of parallel scheduling algorithms*. Oficyna Wydawnicza Politechniki Wrocławskiej, 2010.

[17] W. Bożejko, M. Uchronski, and M. Wodecki, *Parallel Neuro-Tabu Search Algorithm for the Job Shop Scheduling Problem*. Springer Berlin Heidelberg, 2013.

[18] M. Dorigo and T. Stützle, *Ant colony optimization*. MIT Press, 2004.

[19] S. C. Lin, E. D. Goodman, and W. F. Punch, *Investigating parallel genetic algorithms on job shop scheduling problems*. Springer Berlin Heidelberg, 1997, vol. 1213.

[20] Y. Sawaragi, H. Nakagama, and T. Tanino, "Theory of multi-objective optimization," *Academic Press, Inc.*, 1985.

[21] D. L. Bakuli, "A survey of multi-objective scheduling techniques applied to the job shop problem (jsp)," *Applications of Management Science: In Productivity, Finance, and Operations*, pp. 51–62, 2015.

[22] K. R. Baker, "Sequencing rules and due-date assignments in job shop," *Management Science*, vol. 30(9), pp. 1093–1104, 1984.

[23] F. Zhao, Z. Chen, J. Wang, and C. Zhang, "An improved MOEA/D for multi-objective job shop scheduling problem," *International Journal of Computer Integrated Manufacturing*, 2016.

[24] J. Lenstra and A. Rinnooy, "Computational complexity of discrete optimization problems," *Annals of Discrete Mathematics*, vol. 4, pp. 121–140, 12 1979.

[25] G. J. F. and M. G. C. Resende, "A biased random-key genetic algorithm for job shop scheduling," 2011.

[26] S. B. Akers, "A graphical approach to production scheduling problems," *Operations Research*, vol. 4, no. 2, pp. 244–245, 1956. [Online]. Available: https://EconPapers.repec.org/RePEc:inm:oropre:v:4:y:1956:i:2:p:244-245

[27] B. Peng, Z. Lü, and T. Cheng, "A tabu search/path relinking algorithm to solve the job shop scheduling problem," *Computers & Operations Research*, vol. 53, pp. 154 – 164, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0305054814002160

[28] K. Akram, K. Kamal, and A. Zeb, "Fast simulated annealing hybridized with quenching for solving job shop scheduling problem," *Appl. Soft Comput. J*, vol. 49, pp. 510–523, 2016.

[29] S. Lawrence, *Resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques (supplement)*, 1984.

[30] I. Sabuncuoglu and M. Bayiz, "Theory and methodology job shop scheduling with beam search," *European Journal of Operational Research*, 1998.

[31] S. G. Ponnambalam, V. Ramkumar, and N. Jawahar, "A multiobjective genetic algorithm for job shop scheduling," *Production Planning and Control*, vol. 12:8, pp. 764–774, 2001.

[32] R. K. Suresh and M. Mohanasundaram, "Pareto archived simulated annealing for job shop scheduling with multiple objectives," *The International Journal of Advanced Manufacturing Technology*, vol. 29, pp. 184–196, 2006.

[33] V. Kachitvichyanukul and S. Sitthitham, "A two-stage genetic algorithm for multi-objective job shop scheduling problems," *Journal of Intelligent Manufacturing*, vol. 22, pp. 355–365, 2011.

[34] E. Zitzler, K. Deb, and L. Thiele, "Comparison of multiobjective evolutionary algorithms: Empirical results," *Evolutionary computation*, vol. 8, pp. 173–95, 02 2000.

[35] N. Karimi, M. Zandieh, and H. Karamooz, "Bi-objective group scheduling in hybrid flexible flowshop: A multi-phase approach," *Expert Systems with Applications*, vol. 37, no. 6, pp. 4024 – 4032, 2010.

[36] A. Scaria, K. George, and J. Sebastian, "An artificial bee colony approach for multi-objective job shop scheduling," *Procedia Technology*, vol. 25, pp. 1030–1037, 2016.

[37] M. González, A. Oddi, and R. Rasconi, "Multi-objective optimization in a job shop with energy costs through hybrid evolutionary techniques," *Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling*, pp. 140–148, 2017.

[38] B. Méndez-Hernández, E. D. Rodriguez Bazan, Y. Martinez, P. Libin, and A. Nowe, *A Multi-objective Reinforcement Learning Algorithm for JSSP*, 09 2019, pp. 567–584.

[39] P. Serafini, "Simulated annealing for multi objective optimization problems," *Proceedings of the Tenth International Conference on Multiple Criteria Decision Making*, vol. 1, 01 1992.

[40] E. H. L. Aarts, P. J. M. van Laarhoven, J. K. Lenstra, and N. L. J. Ulder, "A Computational Study of Local Search Algorithms for Job Shop Scheduling," *INFORMS Journal on Computing*, vol. 6, no. 2, pp. 118–125, May 1994.

[41] J. Blazewicz, K. Ecker, E. Pesch, G. Schmidt, and J. Weglarz, *Handbook on Scheduling*. Springer, 2007.

[42] M. Iori and S. Martello, "Metaheuristics for scheduling in industrial and manufacturing applications," *Studies in Computational Intelligence*, vol. 128, 2008.

[43] W. Kubiak, C. Sriskandarajah, and K. Zaras, "A note on the complexity of openshop scheduling problems," *INFOR: Information Systems and Operational Research*, vol. 29(4), p. 284–294, 1991.

[44] C. Liu and R. L. Bulfin, "Scheduling ordered open shops," *Computers & Operations Research*, vol. 14(3), p. 257–264, 1987.

[45] C. Prins, "An overview of scheduling problems arising in satellite communications," *Journal of the Operational Research Society*, vol. 45 (6)(6), p. 611–623, 1994.

[46] M. Pinedo and M. Singer, "Shifting bottleneck heuristic for minimizing the total weighted tardiness in a job shop," *Naval Research Logistics*, vol. 46(1), p. 1–17, 1999.

[47] T. Yamada, "Studies on metaheuristics for jobshop and flowshop scheduling problems," Ph.D. dissertation, Kyoto University, 2003.

[48] K. Deb, *Multiobjective Optimization Using Evolutionary Algorithms. Wiley, New York*, 01 2001.

[49] A. Díaz, F. Glover, H. Ghaziri, J. Gonzalez, M. Laguna, P. Moscato, and F. Tseng, *Optimización Heurística y Redes Neuronales*. Paraninfo, Madrid, 1996.

[50] F. Glover, "Future paths for integer programming and links to artificial intelligence," *Computers & Operations Research*, vol. 13, no. 5, pp. 533–549, 1986, applications of Integer Programming. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0305054886900481

[51] J. Kelly and I. H. Osman, "Meta-heuristic: Theory and applications," 1996.

[52] F. Glover, "Heuristics for integer programming using surrogate constraints," *Decision Sciences*, vol. 8, pp. 156 – 166, 01 1977.

[53] S. H. Sanvicente and J. Frausto, "A method to establish the cooling scheme in simulated annealing like algorithms," *Comput. Sci. Its Appl. - ICCSA 2004*, pp. 755–763, 2004.

[54] J. F. Solís, H. S. Sánchez, and F. I. Valenzuela, "Andymark: An analytical method to establish dynamically the length of the markov chain in simulated annealing for the satisfiability problem," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 4247, pp. 269–276, 2006.

[55] R. May, "Simple mathematical models with very complicated dynamics," *Nature*, vol. 26, p. 457, 07 1976.

[56] C. Coello, G. Lamont, and D. van Veldhuizen, *Evolutionary Algorithms for Solving Multi-Objective Problems*, ser. Genetic and Evolutionary Computation. Springer US, 2007.

[57] J. R. Schott., "Fault Tolerant Design Using Single and Multicriteria Genetic Algorithm Optimization." Master's thesis, Department of Aeronautics and Astronautics, Massachusetts Institute of Technology, 1995.

[58] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan, "A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II," *Schoenauer M. et al. (eds) Parallel Problem Solving from Nature PPSN VI. PPSN 2000. Lecture Notes in Computer Science, Springer*, vol. 1917, 2000.

[59] D. A. V. Veldhuizen, "Multiobjective evolutionary algorithms: classifications, analyses, and new innovations," Ph.D. dissertation, Air Force Institute of Technology, Wright-Patterson AFB, Ohio, 1999.

[60] D. A. V. Veldhuizen and G. B. Lamont, "Multiobjective evolutionary algorithm research: a history and analysis," 1998.

[61] C. Coello and N. Cruz, "Solving multiobjective optimization problems using an artificial immune system," *Genetic Programming and Evolvable Machines*, vol. 6, pp. 163–190, 06 2005.

[62] S. Bandyopadhyay, S. Saha, U. Maulik, and K. Deb, "A simulated annealing-based multiobjective optimization algorithm: Amosa," *Evolutionary Computation, IEEE Transactions on*, vol. 12, pp. 269 – 283, 07 2008.

[63] G. Dueck and T. Scheuer, "Threshold accepting: A general purpose algorithm appearing superior to simulated annealing," *Journal of Computational Physics*, vol. 90, pp. 161–175, 09 1990.

[64] N. Matloff, *The art of R progprogram. A tour of statistical software design*. No Starch Press, 2011.

[65] R. Chandra, L. Dagum, D. Kohr, D. Maydan, J. McDonald, and R. Menon, *Parallel Programming in OpenMP*. Morgan Kaufmann Publishers, 2001.

[66] (2021). [Online]. Available: http://ferestrepoca.github.io/paradigmas-de-programacion/paralela/paralela_teoria/index.html#thirteen

[67] L. Clevenger, H. Eng, K. Khan, J. Maghsoudi, and M. Reid, "Parallel computing hardware and software architectures for high performance computing."

[68] (2020). [Online]. Available: https://developer.nvidia.com/cuda-zone

[69] T. Yamada and R. Nakano, "A genetic algorithm applicable to large-scale job-shop problems," *Proc. of The Second International Conference on Parallel Problem Solving from Nature*, pp. 281–290, 1992.

[70] J. Frausto-Solis, L. Hernández-Ramírez, G. Castilla-Valdez, J. González-Barbosa, and J. Sanchez, "Chaotic multi-objective simulated annealing and threshold accepting for job shop scheduling problem," *Mathematical and Computational Applications*, vol. 26, p. 8, 01 2021.