Universidad Autónoma de Baja California

Facultad de Ingeniería, Arquitectura y Diseño

Maestría y Doctorado en Ciencias e Ingeniería

Algoritmo metaheurístico para el problema de programación de tareas tipo Job Shop con restricción de no espera

TESIS

que para cubrir parcialmente los requisitos necesarios para obtener el grado de

DOCTOR EN CIENCIAS

Presenta

VÍCTOR MANUEL VALENZUELA ALCARAZ

Ensenada, Baja California, Febrero del 2022.

UNIVERSIDAD AUTÓNOMA DE BAJA CALIFORNIA

FACULTAD DE INGENIERÍA, ARQUITECTURA Y DISEÑO

MAESTRÍA Y DOCTORADO EN CIENCIAS E INGENIERÍA

Algoritmo metaheurístico para el problema de programación de tareas tipo Job Shop con restricción de no espera.

TESIS

Que para obtener el grado de Doctorado en Ciencias presenta:

Víctor Manuel Valenzuela Alcaraz

Aprobada por:

Dra. María de los Ángeles Cosío León
Directora de tesis

Dr. Carlos Alberto Brizuela Rodríguez
Co-director de tesis

Dra. Anabel Martínez Vargas
Miembro del comité

Dr. Miguel Enrique Martínez Rosas
Miembro del comité

Dr. Jesús Everardo Olguín Tiznado
Miembro del comité

Ensenada Baja California, México. Febrero 2022

Resumen de la tesis de **Víctor Manuel Valenzuela Alcaraz**, presentada como requisito parcial para la obtención del grado de DOCTOR EN CIENCIAS del programa de Maestría y Doctorado en Ciencias e Ingeniería (MYDCI) de la Universidad Autónoma de Baja California (UABC). Ensenada Baja California, México, Febrero 2022.

Algoritmo metaheurístico para el problema de programación de tareas tipo Job Shop con restricción de no espera.

El problema de programación del tipo *no-wait Job Shop* es una extensión del conocido problema *Job Shop* clásico, sujeta a la restricción de que las operaciones de cualquier trabajo, una vez iniciadas, deben ser procesadas inmediatamente, una tras otra, hasta la finalización del trabajo. El problema es NP-difícil y los métodos exactos únicamente pueden resolver casos de tamaño pequeño. Avances reportados sobre este problema en las últimas dos décadas, muestran que un aspecto fundamental para el éxito delas metaheurísticas que lo abordan, ha sido la descomposición de la solución en dos partes: la secuenciación y la programación. En este sentido, la mayoría de estas metaheurísticas utilizan una permutación para representar la secuenciación, mientras que para la programación utiliza reglas específicas que diferencian un enfoque de otro. La principal contribución de este trabajo es la propuesta de un algoritmo coevolutivo cooperativo en el que la secuenciación y programación interactúan entre sí para hacer evolucionar las decisiones de secuenciación y de programación de forma casi óptima. Para ello, el algoritmo coevoluciona una población de permutaciones con una población de cadenas binarias. La permutación decide la secuenciación, mientras que cada bit de la cadena binaria decide si un trabajo se desplaza o no a la posición más a la izquierda en su correspondiente máquina. Por lo tanto, toda la cadena binaria define una regla de programación que se optimiza automáticamente durante el proceso de evolución. El algoritmo también incluye mecanismos de perturbación de un solo paso que ayudan a mejorar la calidad de la solución. El algoritmo propuesto se prueba en un conjunto de casos de referencia para compararlo con siete métodos del estado del arte. Los experimentos computacionales muestran que el algoritmo propuesto produce resultados competitivos, además, se obtienen los mejores valores para cuatro casos de referencia.

Palabras Clave: Scheduling, no-wait, job shop, makespan, timetabling, coevolutivo, algoritmo genético.

Dra. Ma. de los Ángeles Cosío León
Director de Tesis

Dr. Carlos A. Brizuela Rodríguez
Co-director de Tesis

Dedicatoria

A Dios por haberme guiado por el camino de la felicidad, brindarme salud, bienestar y una excelente familia.

A mi esposa **Alma Danisa**, por brindarme su apoyo y motivación día con día para alcanzar nuevas metas, tanto profesionales como personales, pero sobre todo por el amor y cariño que siempre me ha manifestado durante estos 23 años, gracias por estar a mi lado y ser parte de mi vida.

A mis hijas **Danissa y Arlenne**, quienes me prestaron el tiempo que les pertenecía para cumplir con esta meta, siempre las cuidaré para verlas hechas personas capaces y que puedan valerse por si mismas.

A mis padres quienes me dieron vida, apoyo, educación y consejos, a ustedes por siempre mi corazón y mi agradecimiento.

A mis hermanas, cuñadas, cuñados y sobrinos por estar conmigo y ser parte de mi vida, los quiero mucho.

Agradecimientos

Quiero expresar mi agradecimiento:

A mis directores de tesis **Dra. María de los Ángeles Cosío León** y **Dr. Carlos Alberto Brizuela Rodríguez**, a mis sinodales **Dra. Anabel Martínez Vargas, Dra. Larisa Burtseva, Dr. Miguel Enrique Martínez Rosas y Dr. Jesús Everardo Olguín Tiznado** por ofrecerme sus consejos, criticas, su paciencia, apoyo e interés en el desarrollo de esta tesis.

A mis compañeros y amigos por su amistad y los buenos momentos.

A mis maestros, gracias por su tiempo, por su apoyo, así como por el conocimiento que me transmitieron en el desarrollo de mi formación profesional.

Al personal de la Facultad de Ingeniería, Arquitectura y Diseño de la Universidad Autónoma de Baja California, por sus atenciones, amabilidad y disposición para ayudarme.

Al Tecnológico Nacional de México y al Instituto Tecnológico de Agua Prieta, por otorgar la posibilidad de desarrollarme de manera profesional y académica.

Al Centro de Investigación Científica y de Educación Superior de Ensenada (CICESE) por sus atenciones y apoyo en el uso de las instalaciones y facilidades otorgadas durante la realización del doctorado.

Al Consejo Nacional de Ciencia y Tecnología (CONACyT) por el apoyo económico brindado y a la Facultad de Ingeniería, Arquitectura y Diseño de la Universidad Autónoma de Baja California por las facilidades otorgadas para la realización de éste trabajo.

Índice general

Índice de Figuras

Índice de tablas

Capítulo 1

Introducción

Muchos de los problemas de decisión existentes en el mundo real, como la programación de operaciones en máquinas, el despacho de vuelos en aeropuertos, la planeación de la atención a pacientes en hospitales, la distribución de actividades de estudiantes y profesores en instituciones educativas, la decisión de qué automóvil reparar primero en un taller mecánico, entre otros, pueden formularse como problemas de programación de tareas [1, 2]. Este problema consiste en determinar la asignación de un conjunto de tareas a un conjunto de recursos, minimizando algún criterio de optimización [3]. La asignación de las tareas a los recursos se llama programa (schedule).

Los problemas de programación de tareas pueden ser formulados como problemas de optimización combinatoria [4], es decir, hay que elegir una solución entre un conjunto de opciones posibles. Este tipo de problemas se presentan de forma tangible en la optimización de procesos de producción industrial, en donde un conjunto de tareas (pasos para la elaboración de cierto producto) deben ser asignadas para ser procesadas por un conjunto de máquinas [5] y lo que se busca es seleccionar una programación que minimice el tiempo necesario para completar todas las tareas.

La importancia en los problemas de programación de tareas para diseñar programas eficientes, efectivos y precisos en procesos industriales ha aumentado en los últimos años, debido por un lado a la creciente demanda de una gran variedad de productos por parte de los clientes, por otro lado, a la reducción de los ciclos de vida de los productos y al rápido desarrollo de nuevos procesos y tecnologías. Además, a la necesidad de mantener la satisfacción de clientes en la calidad de los productos y su fecha de entrega.

En general, el problema de programación de tareas consiste en asignar un conjunto de trabajos, compuesto por un conjunto de operaciones, para ser procesados en un conjunto de máquinas. El conjunto de características asociados a estos problemas de programación se listan a continuación:

- Las máquinas son entes que realizan alguna tarea y se dispone de un grupo de ellas (cada una con una función específica).
- Una máquina cualquiera no puede realizar más de una operación al mimso tiempo.

- Las operaciones son los pasos necesarios para elaborar un producto terminado y una operación consiste en utilizar una máquina determinada durante un periodo determinado de tiempo ininterrumpido.

- Un trabajo es un producto en particular e indica el conjunto de operaciones para lograr dicho producto.

En un problema de programación de tareas, múltiples trabajos deben ser procesados en varias máquinas. Cada trabajo consiste en una secuencia de operaciones, que deben realizarse en un orden determinado, y cada operación debe procesarse en una máquina específica.

El problema consiste en programar los trabajos en las máquinas para minimizar la duración del programa (Makespan, C_{max}), es decir, el tiempo que tardan en completarse todos los trabajos. Sujeto a las siguientes restricciones, ninguna operación de un trabajo puede iniciarse hasta que la operación anterior de ese trabajo se haya completado, una máquina sólo puede trabajar en una operación a la vez y una vez iniciada una operación, debe ejecutarse hasta su finalización.

Los problemas de programación pueden ser del tipo *Single machine* (Figura 1.1.i) donde todos los trabajos requieren la misma máquina para ser procesados, *Open shop* (Figura 1.1.ii) aqui las operaciones de los trabajos no tienen ninguna restricción en cuanto al orden en que visitan a las máquinas [6, 7], *Flow Shop* (Figura 1.1.iii) el orden por el que cada trabajo visita a las máquinas es siempre el mismo y finalmente el *Job Shop* (Figura 1.1.iv) donde el orden de procesamiento de cada trabajo en las máquinas tiene una secuencia diferente [5]. En la presente tesis se aborda el problema de programación del tipo *Job Shop*.

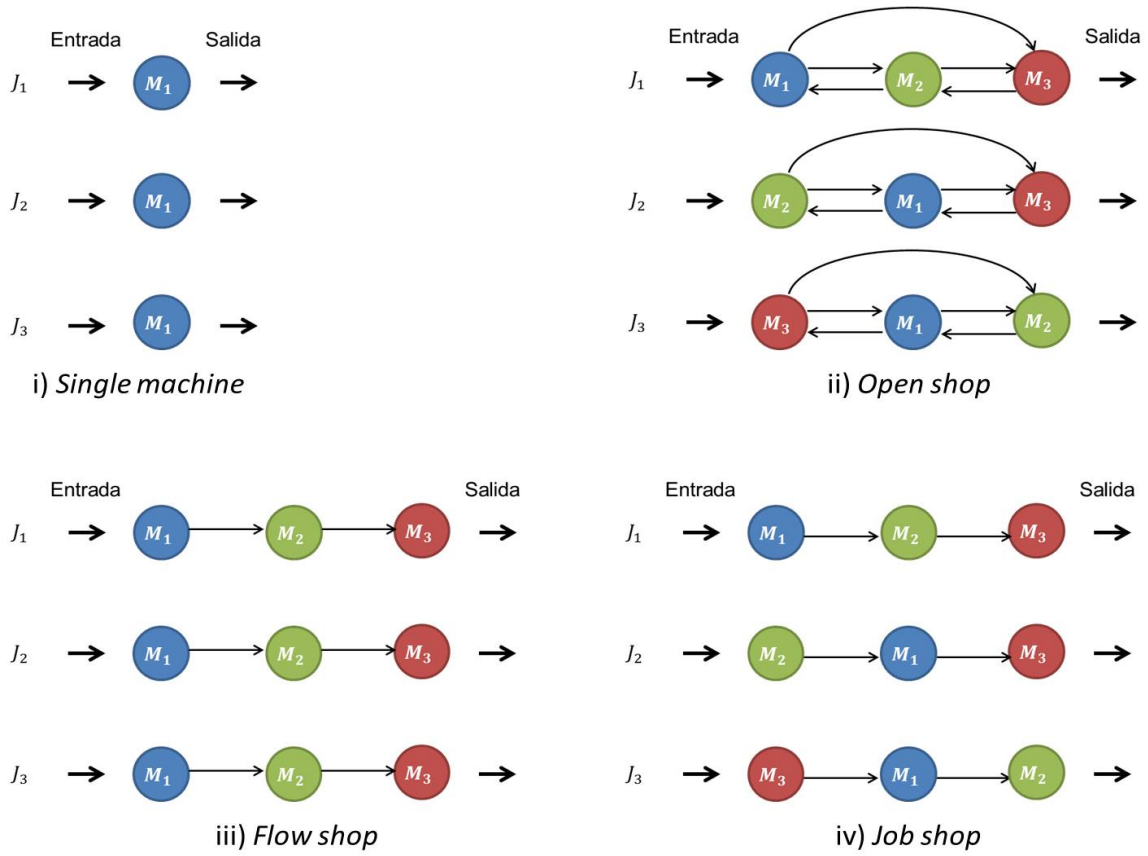

Figura 1.1: Clasificación de problemas de programación. Ejemplo con 3 Trabajos J_1, J_2 y J_3 y 3 Máquinas M_1, M_2 y M_3.

1.1. Planteamiento del problema

En el *Job Shop* clásico, para la generación de un programa factible, es necesario satisfacer las siguientes restricciones: la relación de precedencia entre las operaciones de un mismo trabajo, no se permite procesar dos operaciones simultáneamente en la misma máquina, ninguna operación tiene prioridad sobre las demás, una vez iniciada una operación, debe ejecutarse hasta su finalización, ningún trabajo puede ser procesado más de una vez en la misma máquina, un trabajo tiene que esperar a que la siguiente máquina esté disponible para que este sea procesado y hay solo una máquina de cada tipo [8]. Estas restricciones tecnológicas provocan conflictos entre las operaciones durante la generación de la planificación, lo que causa retrasos (tiempos de espera) de las operaciones en las máquinas con cuello de botella. Por lo anterior, el *Job Shop* clásico es uno de los problemas más complejos de resolver en ambientes de manufactura [9] y [10].

El problema de programación de tareas del tipo *Job Shop* es clasificado como NP-difícil [11], lo que significa que encontrar una solución óptima es posible si se usa un algoritmo enumerativo, y el tiempo de cómputo se incrementa exponencialmente a medida que el tamaño del problema crece.

Un ejemplo de un caso del problema de programación de tareas del tipo *Job Shop* con cinco trabajos y tres máquinas es presentado en la Tabla 1.1. Para cada fila, la primer columna indica el trabajo, las siguientes columnas, muestran la secuencia en la que pasarán sus operaciones por las máquinas. Asociada a cada operación está el tiempo en que debe ser procesada por la máquina. En este ejemplo, para completar el trabajo J_1, las operaciones se ejecutan de la siguiete manera: primero la máquina M_3, después la máquina M_2 y finalmente la máquina M_1.

Tabla 1.1: Problema con 5 trabajos y 3 máquinas.

Trabajo	Máquina (tiempo de procesamiento)		
J_1	$M_3(2)$	$M_2(2)$	$M_1(3)$
J_2	$M_2(3)$	$M_1(2)$	$M_3(3)$
J_3	$M_2(2)$	$M_1(5)$	$M_3(2)$
J_4	$M_1(2)$	$M_3(2)$	$M_2(2)$
J_5	$M_2(2)$	$M_1(1)$	$M_3(3)$

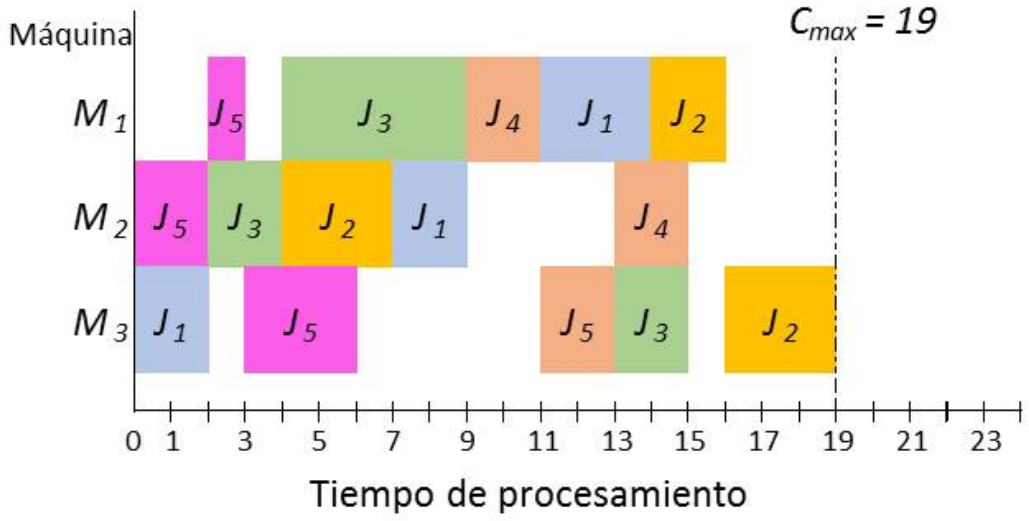

Figura 1.2: Diagrama de Gantt de una posible solución del problema de la Tabla 1.1.

La forma más común de mostrar una solución (programa final) para los problemas de programación de tareas es mediante el diagrama de Gantt. En estos diagramas el eje

horizontal representa el tiempo de procesamiento, en el eje vertical se ubican las máquinas, y los bloques rectangulares (un color diferente para cada trabajo) representan las operaciones de cada trabajo indicando el tiempo de su ejecución como se observa en la Figura 1.2.

El diagrama de Gantt de la Figura 1.2 se interpreta de la siguiente forma: la máquina M_1 inicia procesando la segunda operación del trabajo J_5 en el tiempo 2, después en el tiempo 4, procesará la operación 2 del trabajo J_3, luego en el tiempo 9, procesara la primera operación del trabajo J_4, posteriormente en el tiempo 11 procesará la operación 3 del trabajo J_1 y finalmente, en el tiempo 14 procesara la segunda operación del trabajo J_2. Análogamente se interpretan las máquinas M_2 y M_3. El tiempo necesario para completar todos los trabajos (C_{max}) es de 19 unidades de tiempo.

Por otro lado, un mismo problema de programación de tareas puede tener varias soluciones, por ejemplo la Figura 1.3 muestra cuatro posibles soluciones adicionales para el problema de la Tabla 1.1. En esta figura, se puede observar que si se hace una combinación diferente en el orden en que se procesan las operaciones de las máquinas (programación), se obtienen mejores o peores valores en el C_{max} comparados con el obtenidos en la Figura 1.2.

En realidad, para este problema con 5 trabajos y 3 máquinas existen un total de 1,728,000 soluciones posibles. En general, si n trabajos son procesados en m máquinas, las soluciones posibles está dado por la expresión $(n!)^m$ (Leung 2004), donde m es el número de máquinas y n es el número de trabajos.

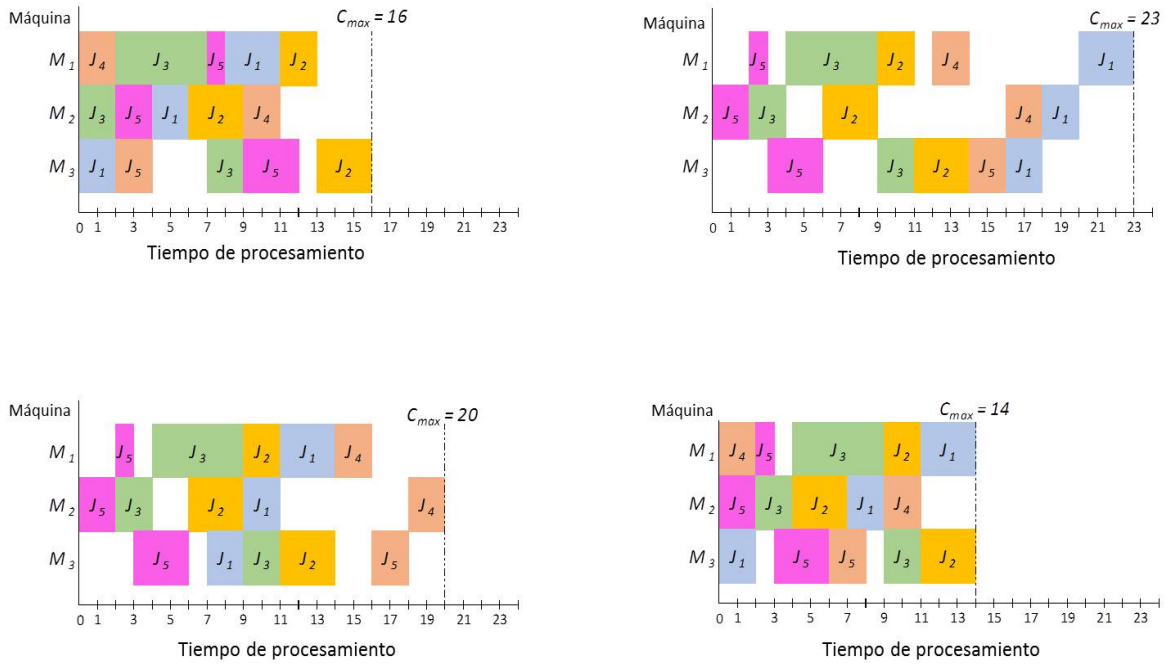

Figura 1.3: Diagrama de Gantt de 4 posibles soluciones del problema de la Tabla 1.1.

Los problemas de programación suelen tener una o más restricciones que hacen más difícil el problema. Algunas de las restricciones más representativas son:

1. *Release date*, indica el instante de tiempo en el que un trabajo puede comenzar a procesarse.
2. *Due date*, indica el instante de tiempo en el cual un trabajo debe ser completado.
3. *Weight*, representa la importancia de un trabajo con respecto a los demás.
4. *Set-up*, indica el tiempo necesario para reconfigurar una máquina antes de iniciar con algun trabajo.
5. *Preemption*, esta restricción se da en aquellos problemas que se permite la interrupción de un trabajo para procesar uno que es más urgente.
6. *Blocking*, esta restricción aparece cuando no tenemos capacidad de almacenamiento entre dos máquinas y los trabajos deben esperar en su máquina actual hasta que se libere la siguiente.
7. *No-wait*, una vez iniciado la primera operación de cualquier trabajo, debe ser procesada hasta que todas sus operaciones hayan finalizado.

El problema que se aborda en esta tesis es el problema de programación de tareas del tipo *Job Shop* con la restricción de no-wait (no-wait job shop) y con el criterio de minimizar el C_{max} como función objetivo.

1.2. Representación de soluciones

Como en todos los problemas que se enfrentan en el mundo real, para poder resolverlos se tiene que encontrar una forma de abstraerlos y poder representar sus posibles soluciones. Cheng et al. [12] han clasificado la representación de soluciones en 9 categorías:

Representación basada en operaciones [13], en donde cada elemento en el vector representa la operación. **Representación basada en trabajos** [14], en este caso, cada posición del vector representa la secuencia y cada elemento en la posición indica el trabajo. **Representación basada en máquinas** [15], cada elemento y su posición en el vector representan la máquina y su orden en ser programada respectivamente. **Representación basada en lista de preferencia** [16], en este caso, cada elemento en el vector representa el orden de preferencia que tienen las operaciones en ser procesadas en una máquina dada. **Representación basada en reglas de prioridad** [15], en este caso, cada elemento en un vector representa una regla de un conjunto de reglas de prioridad de despacho. **Representación basada en números aleatorios** [17], los números aleatorios en el vector son utilizados como claves de ordenamiento para decodificar la solución. **Representación basada en tiempo de finalización** [18], un vector es una lista ordenada de tiempos de finalización de las operaciones. **Representación basada entre pares de trabajos** [19], esta representación está basada en la relación de precedencia entre pares de trabajos sobre las máquinas correspondientes. **Representación basada en grafo disyuntivo** [20], esta representación codifica un programa basada en grafos disyuntivos.

1.3. Motivación

En la presente tesis, se abordará el problema de programación de tareas del tipo no-wait job shop, un problema de la clase NP-difícil [11].

Dado que en la literatura [21–25] solo se ha dado solución a casos de tamaño menor o igual a 20 máquinas con 20 trabajos, surge el interés de la presente investigación en diseñar una representación/decodificación de soluciones al problema de programación de tareas con restricción de no-espera para optimizar el desempeño de una metaheurística.

Por lo anterior, nos hemos planteado las siguientes preguntas de investigación:

1.- ¿Existe una técnica de decodificación diferente a las existentes actualmente para explotar la representación basada en trabajos?

2.- Dada una secuencia de trabajos, ¿Qué trabajos se deben empujar a la izquierda lo más cerca posible al tiempo cero y cuáles no?

Entonces, la motivación principal de esta tesis es diseñar una representación/decodificación de soluciones novedosa que sea capaz de generar resultados competitivos con aquellos generados por los mejores métodos conocidos en la literatura.

1.4. Objetivo general

Desarrollar un algoritmo para ser usado en una metaheurística y su correspondiente representación/decodificación de soluciones para el problema de programación de tareas del tipo job shop con restricción de no-espera que sea competitivo con los métodos del estado del arte.

1.5. Objetivos específicos

- Diseñar una estrategia de decodificación de soluciones para la representación basada en trabajos para abordar el problema el problema de programación de tareas del tipo *Job Shop* con restricción de no-espera (*No-Wait Job Shop Scheduling Problem, NWJSSP*).
- Desarrollar un método de empuje para establecer los tiempos de inicio dada una secuencia de trabajos para el *NWJSSP*.
- Diseñar un algoritmo genético coevolutivo-cooperativo con la decodificación de soluciones y el método de empuje propuestos en los objetivos anteriores.
- Determinar el desempeño relativo del algoritmo diseñado en el objetivo anterior con respecto a los algoritmos del estado del arte para el mismo problema.

1.6. Propuesta de solución

En este documento la metodología para abordar el *NWJSSP* se desarrolla en dos etapas:

Etapa 1 En esta estapa se desarrolla una nueva representación/decodificación soluciones llamada "Decodificación mixta" (*mix*-D/AG). Esta estrategia de decodificación mixta utiliza simultáneamente los procedimientos de decodificación en sentido estándar y en sentido inverso, la cual trabaja junto con un algoritmo genético para encontrar soluciones de buena calidad. La decodificación propuesta, denominada Decodificación Reversa (DR), es una variante de la decodificación estándar para la representación basada en trabajos.

Etapa 2 En esta etapa se desarrolla una nueva estrategia para establecer los tiempos de inicio dada una secuencia de trabajo denominada Algoritmo Decodificación de Empujes (ADE). ADE combina dos estrategias de empuje, una empuja los trabajos a la izquierda lo más cerca posible del tiempo cero y la otra empuja los trabajos a la izquierda lo más cerca posible del tiempo de inactividad más reciente en la máquina correspondiente. ADE trabaja junto con un algoritmo genético coevolutivo-cooperativo (AGCC), ambos algoritmos determinan qué trabajos se empujan con una estrategia o con la otra.

Todos los experimentos computacionales se llevaron a cabo en un conjunto de casos de prueba propuestos en la literatura. Los resultados experimentales se comparan estadísticamente con pruebas no paramétricas.

Los resultados obtenidos de cada una de las metodologías se compararon con los de los enfoques más avanzados en la literatura: *Mix*-D/AG se comparó con CLLM [21], MCLM [22], HABC [23], HH-EA/G [24] y Ozolins [26]. Mientras que AGCC-ADE se compara con CLLM [21], MCLM [22], HABC [23], MSA-BST [25], GASA [27], TS [28], y HTS [29].

1.7. Secuencia de la tesis

La presente tesis está organizada en cinco capítulos, los cuales son descritos brevemente a continuación:

En el Capítulo 1, se da una introducción al problema de programación de tareas y se describe el planteamiento del problema.

En el Capítulo 2, se describe el problema de programación de tareas del tipo no-wait job shop así como su formulación matemática. Se presenta una revisión de la literatura sobre los métodos de solución propuestos para abordar este problema. Además se hace una descripción detallada de las estrategias de empuje propuestas en la literatura para establecer tiempos de inicio.

En el Capítulo 3, se describen las dos metodologías propuestas para resolver el problema de programación de tareas del tipo no-wait job shop.

En el Capítulo 4, se presentan los resultado obtenidos y el análisis de los mismos, en cada una los algoritmos propuestos. Además se presentan las pruebas no paramétricas aplicadas a los resultados obtenidos.

En el Capítulo 5, se presentan las discusiones y conclusiones de los resultados obtenidos, finalmente, se detallan cuáles podrían ser los trabajos futuros derivados de la presente investigación.

Capítulo 2

Revisión de Literatura

En este capítulo se hace una revisión de la literatura relacionada con el *NWJSSP*. Además, se hace una revisión de los métodos de solución para este tipo de problema. Finalmente se hace una revisión de las estrategias que se han propuesto en la literatura para establecer tiempos de inicio de un trabajo dado.

2.1. El problema de programación de tareas tipo no-wait job shop (*NWJSSP*)

En el clásico problema de programación de tareas tipo job shop, se permite la espera de operaciones en un trabajo, esto significa que cuando un trabajo completa una operación, la operación subsiguiente no tiene que ser procesada inmediatamente. Por el contrario, el *NWJSSP* está sujeto a la restricción de que, una vez iniciadas, las operaciones de cualquier trabajo tienen que ser procesadas inmediatamente una tras otra hasta el final del trabajo, es decir, no se permite el tiempo de espera entre dos operaciones consecutivas de un trabajo.

La Figura 2.1 muestra una solución que cumple con la restricción de no-espera para un problema de 6 máquinas con 4 trabajos, por ejemplo, el trabajo J_1 (bloques verdes): en el tiempo 3 termina su primer operación en la máquina M_1, inmediatamente en el mismo tiempo 3 inicia su segunda operación en la máquina M_4, la cual se termina de procesar en el tiempo 6, e inmediatamente en este mismo tiempo inicia su última operación en la máquina M_5. La mismo sucede con el resto de los trabajos.

Esta restricción se encuentra en procesos de la vida real como los sistemas de producción de acero [30, 31], donde el acero que se calienta a temperaturas elevadas tiene que pasar por una serie de operaciones continas antes de ser enfriado para evitar defectos en la calidad del producto. Otros ejemplos incluyen la industria farmacéutica y química, [2, 32, 33], la producción de hormigón, [34, 35], los sistemas informáticos [36], la industria alimentaria [2], entre otros.

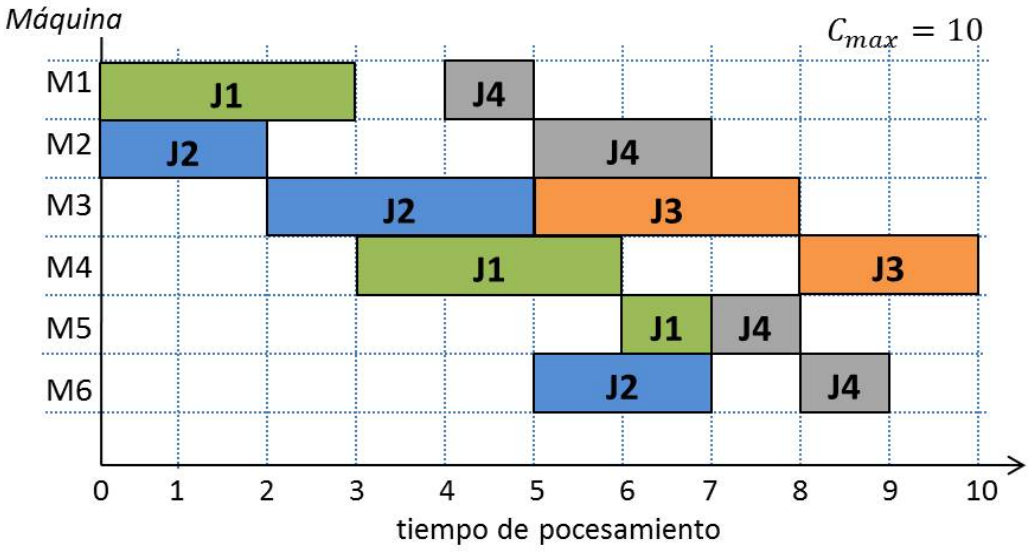

Figura 2.1: Diagrama de Gantt para un *NWJSSP* de 4 trabajos y 6 máquinas.

En términos de complejidad, el *NWJSSP* es un problema $NP\text{-}dificil$ [37], lo que implica que no se conoce algoritmo determinista alguno que encuentre una solución óptima en tiempo de polinomial.

Formalmente el *NWJSSP*, consiste básicamente en programar n trabajos $J = \{J_1, J_2, \ldots, J_n\}$ en m máquinas $M = \{M_1, M_2, \ldots, M_m\}$. Cada trabajo J_j tiene n_j operaciones $O_{j\mu_{j1}}, O_{j\mu_{j2}}, \ldots, O_{j\mu_{jn_j}}$, donde μ_{jk} indica el número de la máquina que procesara la k-ésima operación del trabajo J_j ($1 \leq j \leq n$, $1 \leq \mu_{jk} \leq m$, $k \in K_{n_j} := \{1, 2, \ldots, n_j\}$). Cada operación $O_{j\mu_{jk}}$ es previamente asignada para ser procesada en la máquina μ_{jk}, además, cada operación tiene asignado $t_{j\mu_{jk}}$ unidades de tiempo, que indica el tiempo de procesamiento de la operación $O_{j\mu_{jk}}$.

Adicionalmente, todos los trabajos están disponibles en el tiempo cero, ninguna máquina puede procesar dos o más operaciones al mismo tiempo, y ninguna operación puede ser procesada por dos o más máquinas al mismo tiempo, cada operación debe ser procesada en el orden de su ruta de procesamiento, y una vez que un trabajo comienza su primera operación, continuará sin ninguna interrupción o espera, es decir, dos operaciones consecutivas del mismo trabajo deben ser procesadas, sin tiempo de espera.

El reto en este problema es encontrar una programación de los trabajos en las máquinas, que garantice la restricción de no espera y minimice el makespan (C_{max}). Un calendario es la asignación de los tiempos de inicio $S_{1\mu_{11}}, S_{2\mu_{21}}, \ldots, S_{n\mu_{n1}}$ a las ope-

raciones $O_{1\mu_{11}}$, $O_{2\mu_{21}}$, ..., $O_{n\mu_{n1}}$, respectivamente. En la notación de Graham [38], el *NWJSSP* con makespan como función objetivo se define como: $J|no-wait|C_{max}$.

El *NWJSSP* puede ser formulado matemáticamente de la siguiente manera [39]:

$$Minimizar\ C_{max} = \max_{j \in J}\{S_{j\mu_{jn_j}} + t_{j\mu_{jn_j}}\} \tag{2.1}$$

sujeto a

$$S_{j\mu_{jk}} \geq 0 \quad \forall k \in K_{n_j},\ j \in J, \tag{2.2}$$

$$S_{j\mu_{jk}} + t_{j\mu_{jk}} = S_{j\mu_{j(k+1)}} \quad \forall k \in K_{n_j-1},\ j \in J, \tag{2.3}$$

$$S_{ja} + t_{ja} \leq S_{ia} \quad or \quad S_{ia} + t_{ia} \leq S_{ja} \quad \forall i, j \in J;\ a \in M, \tag{2.4}$$

donde (2.1) es la función objetivo; (2.2) requiere que todas las operaciones estén disponibles a tiempo 0; (2.3) asegura la restricción de no-espera; (2.4) requiere que ninguna máquina pueda procesar dos o más operaciones al mismo tiempo.

2.2. Métodos de solución para el *NWJSSP*

Desde que se planteó el problema, en los años 50, han sido muchos los métodos que se han aplicado para su resolución. En general, estos métodos se pueden clasificar en cuatro categorías: métodos exactos, métodos aproximados, metaheurísticas basadas en una solución y metaheurísticas basadas en una población.

Los métodos exactos se basan en la enumeración de las soluciones del problema y usan un modelo matemático [40]. Los métodos aproximados resuelven el problema encontrando una solución aproximada, aunque no es la óptima, hay una garantía sobre un límite superior de la solución obtenida del óptimo global [40].

Las metaheurísticas basadas en una solución [40, 41], también llamada métodos de trayectoria, aplican un procedimiento de generación y reemplazo a una única solución inicial. En la etapa de generación, se construye un conjunto de soluciones candidatas a partir de la solución actual. En la etapa de reemplazo, se elige una solución adecuada, del conjunto generado, para reemplazar la solución actual. Este proceso continúa hasta que se obtiene una solución de buena calidad. Las metaheurísticas basadas en una población [40, 41] aplican un procedimiento de generación y reemplazo a un conjunto de soluciones dispersas en el espacio de búsqueda. Se genera el conjunto inicial de soluciones. Luego se itera y manipula para generar un nuevo conjunto. El nuevo conjunto de soluciones se selecciona de entre el conjunto de soluciones existente y el recién generado utilizando una estrategia de reemplazo. Este proceso continúa hasta que se cumpla un criterio de parada. La mayoría de este tipo de metaheurísticas están inspiradas en la naturaleza.

Un estudio detallado sobre los *NWJSSP* es proporcionado por Hall y Sriskandaraja [2], además los autores describen diferentes aplicaciones donde se encuentra la restricción de no-espera, así mismo se examina la complejidad del problema. Otro estudio presentado en [42] mostró que el *NWJSSP* es *NP-difícil* incluso cuando el problema se restringe a sólo dos máquinas.

2.2.1. Métodos exactos y aproximados

De los métodos aproximados utilizados para resolver el *NWJSSP* podemos mencionar un algoritmo de tiempo pseudo-polinomial diseñado para problemas con dos máquinas [43, 44], posteriormente, se demostró que el problema es *APX-difícil* para el caso de dos máquinas con un máximo de cinco operaciones por trabajo, y para el caso de tres máquinas con un máximo de tres operaciones por trabajo [45]. Después un esquema de aproximación de tiempo polinómico fue propuesto para este problema con makespan como función objetivo y exactamente dos operaciones por trabajo [46].

En relación a los métodos exactos que han sido utilizados para resolver el problema en casos de tamaño pequeño en un tiempo razonable, son el algoritmo branch-and-bound [47,48], un modelo de programación lineal entera y una combinación de búsqueda binaria con programación de restricciones [49], un algoritmo basado programación dinámica [26]. Adicionalmente, se mostró que el problema es solucionable en un tiempo $O(nlog\ n)$ si cada trabajo tiene exactamente dos operaciones de igual duración [50].

2.2.2. Metaheurísticas

Las metaheurísticas que han sido utilizadas para abordar el problema de programación de tareas del tipo no-wait job shop son: un algoritmo de búsqueda de tabú (TS, por sus siglas en inglés) de dos fases, donde el problema se descompone en el sub-problema de secuenciación y el sub-problema de programación de horarios para resolverlo [51]. Asimismo, un algoritmo de búsqueda de vecindario variable (VNS, por sus siglas en inglés) y un algoritmo híbrido recocido simulado/algoritmo genético (GASA, por sus siglas en inglés) son propuestos para dar solución al problema basándose en su descomposición en el sub-problema de secuenciación y un sub-problema de programación de horarios [27].

Posteriormente, se propone un algoritmo de búsqueda local completa con memoria (CLM, por sus siglas en inglés) que utiliza una estrategia de empuje mejorada para generar programas basado en la estrategia de empuje sin retraso [52]. Poco tiempo después, se propone un algoritmo TS que utiliza una nueva estrategia de programación sin retardo para generar los programas [28]. Años más tarde se desarrolló una heurística de dos fases para resolver el problema con dos máquinas, la fase 1 transforma el problema en un problema de tipo no-wait flow shop, después de que el problema es resuelto por el algoritmo de Gilmore y Gomory, luego en la fase 2, se aplica un algoritmo TS para mejorar la solución obtenida en la fase 1 [53].

Después, se propone un algoritmo TS híbrido [29] que encuentra soluciones mejores (en términos de makespan) que el algoritmo GASA y el algoritmo TS propuesto en [28]. Más tarde, se propone una búsqueda local completa con memoria limitada (CLLM, por sus siglas en inglés) [21] para resolver el problema, los resultados mostraron que este enfoque es superior a los algoritmos VNS, GASA, y CLM. Posteriormente, se desarrolla un algoritmo modificado de memoria de búsqueda local completa (MCLM, por sus siglas en inglés) [22], sus resultados mostraron ser superiores sobre los enfoques CLM y CLLM. Más tarde, se propusieron tres algoritmos de secuenciación: a) TS, b) híbrido de TS con búsqueda de vecindario variable, y c) híbrido de TS con optimización del enjambre de partículas [54], donde se analizó el efecto de estos tres algoritmos combinándolos con cuatro estrategias de empuje para resolver el *NWJSSP*.

Posteriormente, un algoritmo de estimación de distribuciones [55] es propuesto, sus resultados mostraron que el algoritmo es mejor que el CLLM con respecto al makespan. Después, se presenta un algoritmo de búsqueda local completa con memoria y estructura de vecindad variable (CLMMV, por sus siglas en inglés) [56], los resultados muestran que el CLMMV es mucho más eficiente que el CLLM. Después se plantea un algoritmo híbrido de optimización de búsqueda de grupos discretos [57]. Finalmente, se sugiere un algoritmo de recocido simulado junto con una estrategia de empuje bidireccional [25], los resultados mostraron que el algoritmo superó a los tres mejores metaheurísticos (CLLM, MCLM y HABC) disponibles hasta la fecha.

Por otro lado, las metaheurísticas basadas en la población que han sido empleadas para solucionar el *NWJSSP* se describen a continuación. Algoritmo genético híbrido [58], en el que es seccionada aleatoriamente una sección secuencial de un cromosoma, esta sección se resuelve como un problema del agente viajero [59] y la mejor secuencia obtenida se coloca de nuevo en el cromosoma. Después se presenta un algoritmo genético con ajuste automático de sus parámetros [60]. Más adelante, se desarrolla un híbrido entre el algoritmo de entropía cruzada y el algoritmo genético (CEGA, por sus siglas en inglés) [61], en general, el rendimiento de CEGA es mejor que GASA especialmente para casos de prueba de tamaño pequeño.

Más adelante, se plantea un algoritmo Hibrido de Colonia de Abejas Artificiales (HABC, por sus siglas en inglés) [23], los resultados mostraron que el HABC es mejor que los algoritmos CLLM y MCLM. Años más tarde, se propone una hyper-heurística basada en algoritmos genéticos y algoritmos de estimación de distribuciones [24], sus resultados en casos de prueba pequeños muestran que la heurística es capaz de lograr resultados mejores o iguales en menos tiempo que los enfoques CLLM, MCLM y HABC. Más tarde se presento un algoritmo genético que utiliza un procedimiento que combina una codificación y decodificación de soluciones entre estándar e inversa [62]. Seguidamente, se formula un algoritmo iterativo basado en una población [63], donde el problema es abordado bajo el esquema de descomposición en el sub-problema de secuenciación y el sub-problema de horario.

Muchas de las metaheurísticas anteriores se basaban en la idea de descomponer el problema en el sub-problema de secuenciación y el sub-problema de horarios propuesto por Macchiaroli et al. [51]. El **sub-problema de secuenciación** consiste en encontrar una secuencia óptima de trabajos, mientras que el **sub-problema de horarios** consiste en determinar un conjunto factible de tiempos de inicio para la secuencia de trabajos obtenida por el sub-problema de secuenciación. Las diferencias entre estas metaheurísticas residen principalmente en el paradigma de búsqueda de la técnica utilizada en el sub-problema de secuenciación y en las reglas de despacho empleadas en el sub-problema de horarios.

2.3. Métodos de programación de horarios

Algunas metaheurísticas que han abordado el problema de programación de tareas tipo no-wait job shop han propuesto diferentes estrategias para establecer los tiempos de inicio dada una secuencia de trabajos. A estas estrategias se les conocen como métodos de programación de horarios [21, 22, 25, 56, 63]. A continuación se presenta una muestra representativa de estos métodos.

1. Non-delay timetabling [27]: en esta estrategia, el tiempo de inicio de cada trabajo en la secuencia de procesamiento se establece lo antes posible sin violar la restricción ordinal (es decir, la definida por la secuencia de procesamiento). Por ejemplo, supongamos la siguiente secuencia de 5 trabajos (J_2, J_3, J_1, J_5, J_4), la restricción de ordinal implica que los trabajo J_2, J_3 y J_1 no pueden comenzar antes que los trabajos J_5 y J_4.

2. Enhanced timetabling [52]: esta estrategia genera dos programas. Uno con el método non-delay timetabling y el otro también con el método non-delay timetabling pero utilizando la secuencia de trabajos y la secuencia de sus operaciones de forma inversa (non-delay reverse timetabling). Después, el programa con el mejor makespan se selecciona como el programa final.

3. Left-shift timetabling [21, 29, 39]: esta estrategia es similar a la de la non-delay timetabling, con la excepción de que se puede violar la restricción ordinal impuesta por la secuencia de procesamiento. Esta estrategia empuja a la izquierda todos los trabajos lo más cerca posible del tiempo cero.

4. Shift timetabling procedure [21]: esta estrategia divide la secuencia de trabajos en sub-secuencias, luego para cada sub-secuencia se generan dos sub-programas, uno aplica la estrategia left-shift timetabling a todos los trabajos de la sub-secuencia, y el otro, aplica una estrategia right-shift timetabling al primer trabajo de la sub-secuencia. El mejor de los dos horarios generados se reasigna como el horario actual. El programa actual es el programa final después de que todas las sub-secuencias hayan sido probadas.

5. Right-shift + reverse timetabling [54]: primeramente se generan dos programas, uno con non-delay timetabling y el otro con non-delay reverse timetabling, luego, se aplica una estrategia de empuje a la derecha a ambos programas, generando de esta manera dos nuevos programas, finalmente, la estrategia de empuje a la izquierda se aplica a los cuatro programas, y el mejor de ellos será elegido como el programa que se asocia con la secuencia de trabajos dada.

6. Alternative to delay timetabling [56]: este método adopta las reglas, "lo más temprano posible" o "lo más tarde posible" alternativamente para generar la programación de una secuencia. El método de programación utilizado en esta estrategia es el mismo que el del left-shift timetabling.

7. Shift penalty-based timetabling [22]: esta estrategia se basa en la diferencia de tiempo entre pares de trabajos. En primer lugar, se generan dos programas iniciales con una estrategia basada en penalización de empujes, uno bajo la restricción ordinal con respecto al tiempo de inicio y el otro bajo la restricción ordinal con respecto al tiempo de finalización, luego, estos programas se mejoran con un método de ajuste, finalmente, se selecciona el programa con la mejor makespan como el programa final.

8. Bi-directional shift timetabling [25]: primeramente de forma aleatoria se decide si genera un programa hacia adelante o hacia atrás, luego selecciona al azar un trabajo de la secuencia, y luego determina si el trabajo seleccionado se retrasa una unidad de tiempo o no. Un programa hacia adelante decodifica los tiempos de inicio de las operaciones de izquierda a derecha en la secuencia de trabajos, mientras que un programa hacia atrás decodifica los tiempos de inicio de las operaciones de derecha a izquierda de la secuencia de trabajos. En ambos casos, los trabajos de la secuencia se procesan uno por uno.

Las diferencias entre los métodos anteriores radica mayormente en la estrategia de empuje utilizada para decidir el tiempo de inicio de los trabajos. En este contexto, una pregunta central que las metaheurísticas tienen que afrontar dentro de las estrategias de empuje es, **¿qué trabajos se deben empujar a la izquierda lo más cerca posible al tiempo cero y cuáles no?** Nuestra contribución se centra en la forma de enfocar esta decisión, proponemos que un algoritmo genético coevolutivo cooperativo tome esta decisión coevolucionando el componente de programación de horarios con la parte de secuenciación.

El algoritmo genético coevolutivo cooperativo (AGCC) [64] es una extensión del algoritmo genético (AG) clásico [65]. El AGCC se basa en el paradigma de "divide y vencerás", el problema se descompone en sub-problemas, en el que cada sub-problema es tratado por una sub-población que es manejada por un AG estándar. En el AGCC, se genera una solución completa combinando individuos en cada sub-población. Es decir, la aptitud de un individuo de una sub-población particular se calcula estimando

lo bien que "coopera" con otras sub-poblaciones para producir buenas soluciones. Un estudio detallado de las aplicaciones del AGCC a los problemas de programación de tareas puede ser encontrado en [66].

En el AGCC propuesto, una población con cromosomas basados en permutaciones de trabajos para tratar la parte de la secuenciación, y una población de cromosomas binarios para tratar la parte de la programación de horarios coevolucionan y cooperan para buscar una solución que minimice el makespan.

En esta tesis se proponen:

- Una estrategia de decodificación combinada que utiliza simultáneamente un procedimiento de decodificación estándar con un nuevo procedimiento de decodificación en sentido inverso. La decodificación propuesta, denominada decodificación reversa, es una variante de la decodificación estándar para la representación basada en trabajos.
- Una estrategia de empuje de trabajos que combina dos estrategias de empuje para determinar el tiempo de inicio de un trabajo dado: una de ellas empuja a la izquierda el trabajo lo más cerca posible del tiempo cero, y la otra empuja a la izquierda el trabajo lo más cerca posible del tiempo de inactividad más reciente en la máquina correspondiente.

Capítulo 3

Metodología implementada

En el presente capítulo se describe la metodología seguida para abordar el NWJSSP, con el criterio de el minimizar makespan como función objetivo. La metodología se desarrolla en dos etapas:

Primeramente, se describe un algoritmo genético con una nueva estrategia de decodificación denominada decodificación reversa. Mediante ejemplos numéricos se muestra que al utilizar la decodificación reversa se obtienen mejores soluciones que cuando se utiliza la decodificación estándar.

Posteriormente, se detalla un algoritmo genético evolutivo cooperativo junto con una nueva estrategia de empuje de trabajos llamada algoritmo de decodificación de empujes. Ambos algoritmos determinan qué trabajos se empujan a la izquierda lo más cerca posible del tiempo cero y qué trabajos se desplazan a la izquierda lo más cerca posible del tiempo de inactividad más reciente en la máquina correspondiente.

3.1. Algoritmo genético para el problema de programación de tareas tipo no-wait job shop

Esta sección describe cómo resolver el problema de programación de tareas tipo no-wait job shop usando un algoritmo genético con una novedosa estrategia de decodificación denominada decodificación reversa.

3.1.1. Representación de la solución

La representacion basada en trabajos [14] es utilizada para codificar un programa. Dado que el NWJSSP es un problema de permutación, cada solución (programa) se representa como una permutación de trabajos que describe el orden en que los trabajos deben ser decodificados en su diagrama de Gantt. Por ejemplo, una permutación de trabajos (4, 5, 1, 2, 3) representa una solución para el problema que se muestra en la

Tabla 3.1. Esto indica que el trabajo J_4 debe programarse primero, seguido del trabajo J_5, J_1, J_2 y al final el trabajo J_3. Las restricciones de precedencia para cada trabajo y sus tiempos de procesamiento de sus operaciones se muestran en la Tabla 3.1.

Tabla 3.1: Ejemplo un problema de tamaño $n = 5$, $m = 3$.

Trabajo	Máquina (Tiempo de procesamiento)		
J_1	$M_2(3)$	$M_2(2)$	$M_3(1)$
J_2	$M_1(1)$	$M_1(4)$	$M_3(2)$
J_3	$M_2(2)$	$M_3(2)$	$M_1(5)$
J_4	$M_1(2)$	$M_3(2)$	$M_2(1)$
J_5	$M_2(2)$	$M_1(1)$	$M_3(4)$

3.1.2. Decodificación de la solución

Se aplica el prodecimiento de decodificación de soluciones propuesto por Brizuela et al. [39]. Esta decodificación asegura la factibilidad de las soluciones. El procedimiento de decodificación funciona de la siguiente manera:

Algoritmo decodificación estándar [39]:

- Paso 1. Asignar intervalos de tiempo muerto de cero a infinito a cada máquina.
- Paso 2. Seleccionar el primer trabajo de la permutación y encontrar todas las coincidencias entre los intervalos de tiempo muerto de las máquinas y el tiempo de procesamiento del trabajo actual, de manera que no se viole la restricción de no-espera.
- Paso 3. Actualizar los intervalos de tiempo muerto de todas las máquinas y eliminar el trabajo seleccionado de la permutación.
- Paso 4. Si todos los trabajos de la permutación han sido programados, entonces el procedimiento se detiene. De lo contrario, volver al Paso 2.

La Figura 3.1 muestra una solución mediante la gráfica de Gantt de la secuencia de trabajos (4, 5, 1, 2, 3) para el problema descrito en la Tabla 3.1, donde el trabajo J_4 debe procesar su primer operación en la máquina M_1, después su segunda operación es procesada por la máquina M_3 y su última operación se procesa en la máquina M_2. Por otro lado, el trabajo J_5 debe ser procesado primeramente en la máquina M_2, luego en la máquina M_1 y al final en la máquina M_3. Análogamente para los trabajos J_1, J_2 y J_3. Este orden en el procesamiento de las operaciones de un trabajo determinado, que denominaremos decodificación estándar (DE), es el que se utiliza comúnmente para decodificar la precedencia de las operaciones en el problema.

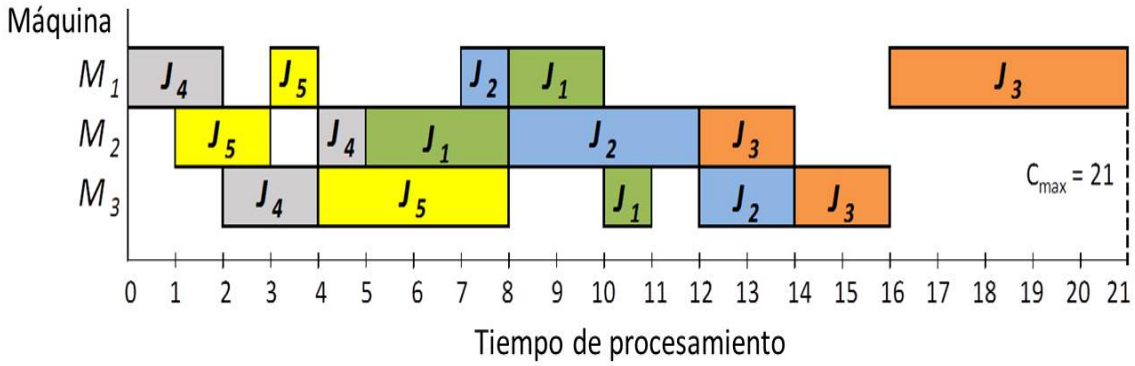

Figura 3.1: Gráfica de Gantt con la solución obtenida al utilizar la decodificación estándar (DE) para la permutación (4,5,1,2,3).

Decodificación Reversa (DR)

A diferencia de la estrategia DE, DR decodifica el orden de las operaciones en una secuencia completamente inversa. Esto significa que las operaciones de un trabajo serán procesadas empezando por la última operación, seguida por la operación anterior, y así sucesivamente.

Regresando al ejemplo de la Tabla 3.1 y aplicando la estrategia DR, la decodificación de las operaciones para la permutación (4, 5, 1, 2, 3) es la siguiente, el trabajo J_4 debe ser procesado primeramente en la máquina M_2, luego en M_3 y al último en M_1, igualmente el trabajo J_5 debe ser procesado primero en la máquina M_3, luego en la máquina M_1 y en la máquina M_2. Del mismo modo, el trabajo J_1 debe ser ejecutado primero por M_3, seguido por M_1 y M_2, análogamente para los trabajos 2 y 3.

En general, para decodificar el orden de las operaciones de un trabajo dado con la estrategia DR se define como $\{O_{j\mu_{jn_j}}, O_{j\mu_{jn_j-1}}, O_{j\mu_{jn_j-2}}, ..., O_{j\mu_{j1}}\}$, donde n_j es el número de operaciones del trabajo J_j. Cada operación $O_{j\mu_{jk}}$ del trabajo J_j se asigna previamnte para ser procesada en la máquina μ_{jk} ($1 \leq j \leq n$, $1 \leq \mu_{jk} \leq m$, $k \in K_{n_j} := \{1, 2, \ldots, n_j\}$).

La Figura 3.2 muestra un gráfico de Gantt para la secuencia (4, 5, 1, 2, 3) decodificada con la estrategia DR descrita anteriormente. Sin embargo, esta solución no cumple con la restricción de precedencia de que cada trabajo debe ser procesado en el orden de sus operaciones (ver la Tabla 3.1). Para resolver esto, se añade el paso 5 al algoritmo propuesto en [39]. La nueva versión del algoritmo se describe en la siguiente sección.

Algoritmo decodificación con estrategia DR:

- Paso 1. Asignar intervalos de tiempo muerto de cero a infinito a cada máquina.
- Paso 2. Seleccionar el primer trabajo de la permutación y encontrar todas las coincidencias entre los intervalos de tiempo muerto de las máquinas y el tiempo

de procesamiento del trabajo actual, de manera que no se viole la restricción de no-espera.

- Paso 3. Actualizar los intervalos de tiempo muerto de todas las máquinas y eliminar el trabajo seleccionado de la permutación.
- Paso 4. Si todos los trabajos de la permutación han sido programados, entonces el procedimiento se detiene. De lo contrario, volver al Paso 2.
- Paso 5. Para todas las operaciones de un trabajo determinado, se recalculan los tiempos de inicio y fin, que se definen mediante la siguiente ecuación:

$$S_{j\mu_{j1}} = C_{max} - S'_{j\mu_{j1}} \tag{3.1}$$

donde:
$S'_{j\mu_{j1}}$ = *tiempo de inicio actual de* $O_j\mu_{j1}$.

Después de aplicar el algoritmo de decodificación con estrategia DR, la nueva solución será como se muestra en la Figura 3.3, donde se observa que se satisfacen todas las limitaciones de precedencia del ejemplo de la Tabla 3.1. Con esta propuesta el C_{max} se reduce de 21 (estrategia DE) a 17 (estrategia DR). Obsérvese que esta reducción de C_{max} no garantiza que ocurra siempre. Es importante aclarar que la estrategia RD no equivale a invertir la permutación, es decir, DR (4, 5, 1, 2, 3) $\neq$ DE (3, 2, 1, 5, 4), aunque en algunos casos pueden coincidir.

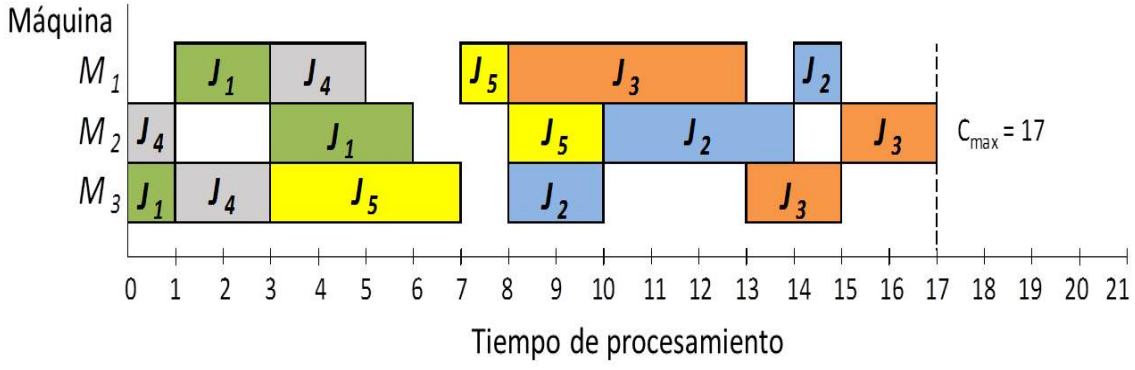

Figura 3.2: Gráfica de Gantt que muestra una solución infactible (no cumple con la restricción de precedencia) con la estrategia DR para la permutación (4,5,1,2,3).

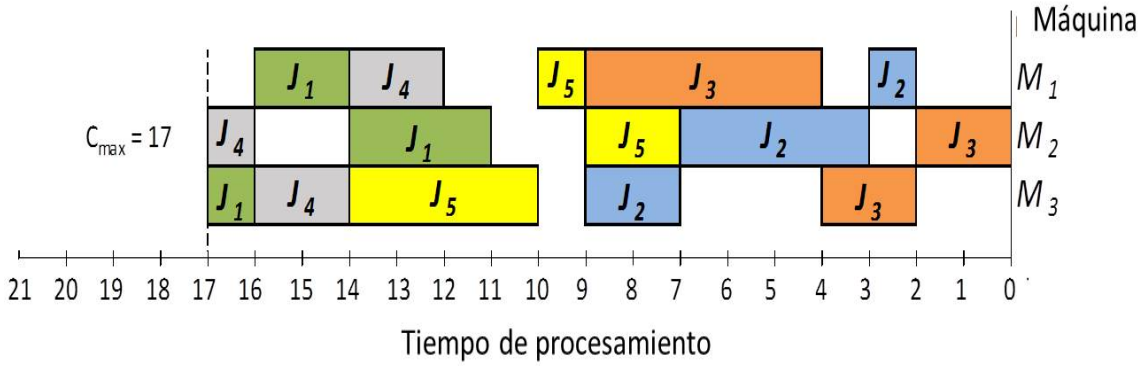

Figura 3.3: Gráfica de Gantt con la solución obtenida al utilizar la decodificación reversa (DR) para la permutación (4,5,1,2,3).

3.1.3. Función Objetivo

La función objetivo a minimizar es la longitud del programa (C_{max}), dada por la ecuación (2.1). La función objetivo se calcula en el algoritmo de decodificación SD o RD.
El valor del C_{max} se define de la siguiente manera: cada trabajo tiene un número de operaciones y cada operación tiene un tiempo de procesamiento particular. Cuando estas operaciones se procesan en la secuencia del programa, cada máquina obtiene un tiempo final de fabricación particular. El máximo tiempo de finalizacion de todas las máquinas es el valor de C_{max}.

3.1.4. Generación de la población inicial y operador de selección

Cada cromosoma en la población representa una permutación, que es decodificado en una solución (programa) para el problema. Cada gen en el cromosoma representa un trabajo y el orden en que aparecen representa la secuencia de los trabajos. Cada cromosoma se genera de manera aleatoria y se aplica el algoritmo de decodificación SD o RD (Sección 3.1.2) para generar una solución factible y calcular el valor de aptitud correspondiente.

La selección de la ruleta [67] se utiliza para la selección de los padres. La selección de la ruleta es un método de selección estocástica, donde la probabilidad de selección de un individuo es proporcional a su aptitud.

El método se inspira en las ruletas del mundo real pero posee una importante diferencia con respecto a ellas. Note que las ruletas siempre tienen segmentos del mismo tamaño como se observa en la Figura 3.4a. Eso significa, que todos los segmentos tienen la misma probabilidad de ser seleccionados.

Por el contrario, en la versión utilizada en los algoritmos genéticos, cada segmento representa un individuo de la población y su tamaño está en función de su valor de aptitud tal como se muestra en la Figura 3.4b, es decir, entre mayor sea la aptitud de un individuo, más grande será el tamaño de su segmento en la ruleta. En otras palabras, los padres son seleccionados con una probabilidad directamente proporcional a sus valores de aptitud (una porción de la rueda de la ruleta). El individuo con la mayor aptitud (es decir, los mayores tamaños de los segmentos en la ruleta) tienen más probabilidad de ser elegidos. El individuo más apto ocupa el segmento más grande, mientras que el menos apto tiene un segmento correspondientemente más pequeño dentro de la rueda de la ruleta.

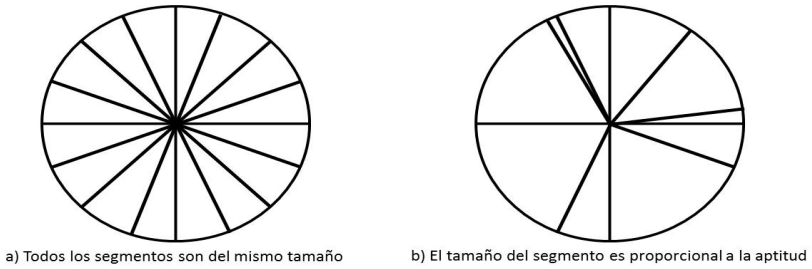

Figura 3.4: La selección de la rueda de la ruleta.

El área de la ruleta es la suma de todos los valores de aptitud de los individuos de una población. La probabilidad de selección ρ_i para el individuo i se define como:

$$\rho_i = \frac{f_i}{\sum_{j \in g} f_j} \tag{3.2}$$

donde f_1, f_2, ... f_g son valores de la aptitud del individuo $1, 2, ..., g$, respectivamente.

3.1.5. Operadores de cruzamiento

La Figura 3.17 muestra los dos operadores de cruzamiento implementados en el algoritmo propuesto, los cuales son descritos a continuación:

- **Cruzamiento ordenado de 1-punto** [68]: Este operador es una adaptación para los problemas de permutación basada en el cruzamiento de un punto [69]. El punto de cruce se selecciona aleatoriamente dentro del rango [1, l-1], donde l es la longitud del individuo. Después de que cada padre es dividido en este punto, cada hijo tiene la primera participación del primer padre y la segunda participación del segundo, eliminando los trabajos duplicados (esto asegura soluciones factibles). Se crean dos hijos combinando los padres en el punto de cruce.

- **Cruzamiento de mapeado parcial** [70]: Este operador trasmite tanto los valores de los genes como el orden en que aparecen en el cromosoma de los padres

a los hijos. Para crear nuevos cromosomas para la próxima generación, se seleccionan aleatoriamente dos puntos de cruce, la cadena de genes entre estos puntos se intercambian para formar una nueva desendencia. Luego, los extremos de los puntos de cruce de cada hijo se crean añadiendo los cromosomas del otro padre, eliminando los trabajos duplicados.

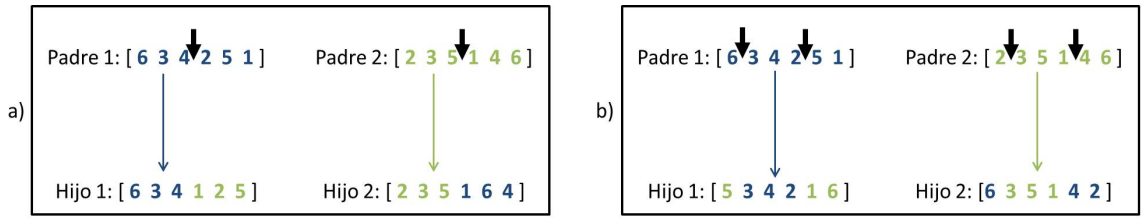

Figura 3.5: Operadores de cruzamiento: a) ordenado de 1-punto. b) mapeado parcial.

3.1.6. Operadores de mutación

La Figura 3.6 muestra los cuatro operadores de mutación implementados en el algoritmo propuesto, los cuales son descritos a continuación:

- **Intercambio 2-posiciones** [71]: Se seleccionan aleatoriamente dos posiciones de un cromosoma y se intercambian sus valores (Figura 3.6a).
- **Intercambio 3-posiciones** [71]: Se eligen aleatoriamente tres posiciones del cromosoma y se intercambian los valores correspondientes a esas posiciones. El elemento de la primera posición seleccionada se mueve a la segunda posición, el elemento de la segunda posición se mueve a la tercera posición y finalmente el elemento de la tercera posición se mueva a la primera posición (Figura 3.6b).
- **Inserción** [71]: Se seleccionan aleatoriamente dos posiciones en un cromosoma, después, el elemento de la primera posición se inserta en la segunda posición, el elemento de la segunda posición se mueve a la siguiente posición con respecto al primer elemento, los elementos anteriores se desplazan (Figura 3.6c).
- **Inversión** [71]: Se seleccionan de forma totalmente aleatoria dos posiciones en un cromosoma, estas posiciones definen un intervalo. Los elementos dentro de este intervalo se re-ordenan en sentido opuesto (Figura 3.6d).

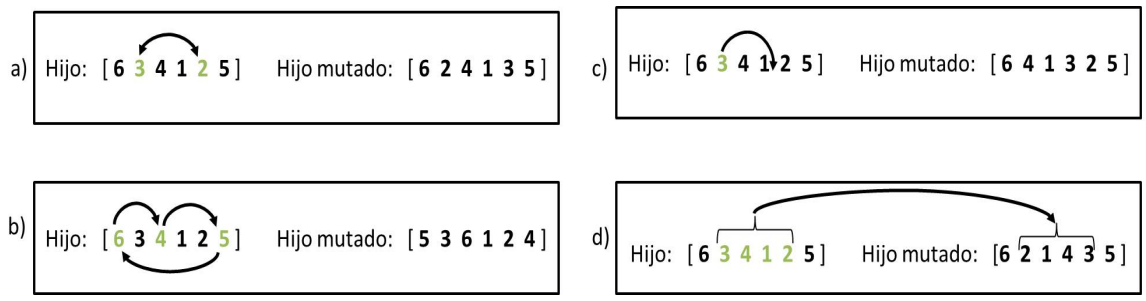

Figura 3.6: Operadores de mutación: a) Mutación de intercambio 2-posiciones, b) Mutación de intercambio 3-posiciones, c) Mutación de inserción y d) Mutación de inversión.

El algoritmo genético selecciona aleatoriamente un operador de cruzamiento y un operador de mutación de los mencionados anteriormente. Tanto el operador de cruzamiento como el de mutación se seleccionan siguiendo una distribución de probabilidad uniforme.

3.1.7. Estrategia para mejorar soluciones

Después de aplicar cualquiera de los operadores de mutación mencionados, se aplican dos estrategias para tratar de encontrar una mejora en la descendencia.
Antes de aplicar ambas estrategias, la secuencia de trabajos se divide en k sub-secuencias. Cada sub-secuencia tiene $\lfloor n/k \rfloor$ trabajos, $k \in \{2, \lfloor n/3 \rfloor\}$. En algunos casos, la última sub-secuencia podría tener un número diferente de trabajos que el resto de las sub-secuencias (por ejemplo con $k = 3$ y $n = 20$, la última sub-secuencia tendrá seis trabajos). El primer trabajo de una sub-secuencia se llama el trabajo principal y al resto se les llama trabajos miembros. Después, el procedimiento comienza a intercambiar el trabajo principal con sus trabajos miembros uno por uno para cada sub-secuencia..

- **La estrategia Partial pair-wise exchange** [21] genera vecinos con sólo intercambiar el trabajo principal y sus trabajos miembros respectivamente para cada sub-secuencia. La Figura 3.7 muestra los 4 vecinos generados al aplicar esta estrategia al hijo obtenido en la Figura 3.18a. En este ejemplo la secuencia es dividida en dos sub-secuencias.

- **La estrategia Simple 1-insertion** [21] genera vecinos moviendo cada sub-secuencia a cada posible combinación respecto a las otras sub-secuencias una a una. La Figura 3.8 muestra los 5 vecinos generados al aplicar esta estrategia al hijo obtenido en la Figura 3.18a. En este ejemplo la secuencia es dividida en tres sub-secuencias.

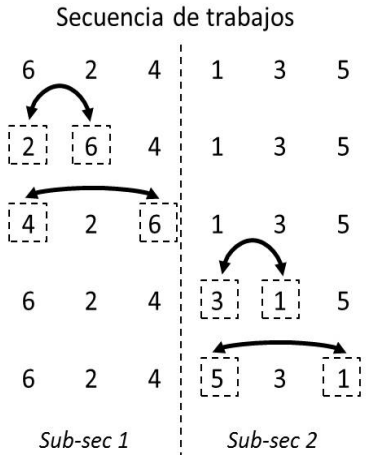

Figura 3.7: Aplicación de la estrategia Partial Pair-Wise Exchange [21] a la secuencia de trabajos (6,2,4,1,3,5).

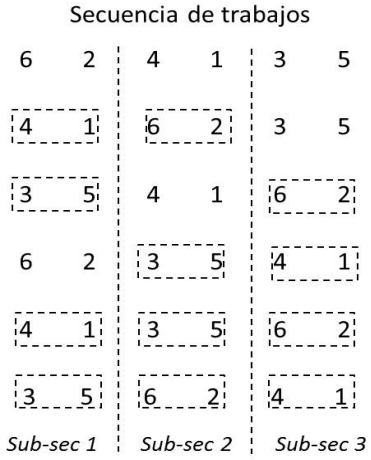

Figura 3.8: Ejemplo de la estrategia Simple 1-insertion [21] al ser aplicada a la secuencia de trabajos (6,2,4,1,3,5).

3.1.8. Criterio de terminación

Cuando el algoritmo genético alcanza un número predefinido de generaciones, este se detiene y devuelve como salida el mejor programa junto con el correspondiente makespan.

3.2. Algoritmo genético coevolutivo-cooperativo para el problema de programación de tareas tipo no-wait job shop

En esta sección se describe el algoritmo genético coevolutivo-cooperativo (AGCC) que trabaja junto con un nuevo método de programación de horarios. Este método combina dos estrategias de empuje para determinar el tiempo de inicio de un trabajo dado: una estrategia empuja el trabajo a la izquierda lo más cercano posible al tiempo cero, y la otra estrategia empuja el trabajo a la izquierda lo más cercano posible al intervalo de tiempo muerto más reciente de la máquina que procesará su primera operación.

Para ilustrar estas estrategias, consideremos un problema con 5 trabajos y 3 máquinas que se presenta en la Tabla 3.2. Dada la secuencia de trabajos (J_5,J_3,J_1,J_4,J_2), el programa parcial A de la Figura 3.9 muestra el diagrama de Gantt después de programar los trabajos J_5 y J_3. Según la secuencia el próximo trabajo a programar es J_1 y su primera operación debe ser procesada en la máquina M_3:

- Si se utiliza la estrategia "empujar el trabajo lo más cerca posible del tiempo cero", significa que se debe de buscar a partir del tiempo cero, un intervalo de tiempo en que la máquina esté disponible y que coincida con los tiempos de procesamiento de todas las operaciones del trabajo J_1. A estos intervalos de tiempo le llamaremos intervalos de tiempo muerto, por ejemplo la máquina M_3 del programa parcial A tiene tres intervalos de tiempo muerto: de 0 a 3, de 6 a 9 y de 11 a ∞. En este sentido, el tiempo de inicio más cercano al tiempo cero y que coincide con los tiempos de procesamiento de las operaciones y sin violar la restricción de no-espera para el trabajo J_1 será el tiempo 6 (segundo intervalo de tiempo muerto de la máquina M_3), tal como se muestra en el programa parcial B de la Figura 3.9.

- Por el contrario, si se utiliza la estrategia "empujar el trabajo a la izquierda lo más cerca posible al tiempo de inactividad más reciente de su máquina correspondiente", significa que se debe de buscar a partir del último intervalo de tiempo muerto, un intervalo de tiempo que coincida con los tiempos de procesamiento de las operaciones del trabajo J_1. En este caso el último intervalo de tiempo de la máquina M_3 es de 11 a ∞, por lo tanto se buscaría a partir del tiempo 11, que a la vez resulta ser un tiempo de inicio factible para el trabajo J_1, así como se observa en el programa parcial C de la Figura 3.9. La motivaión de utilizar la segunda estrategia es que puede permitir que los trabajos futuros se acomoden mejor en la programación.

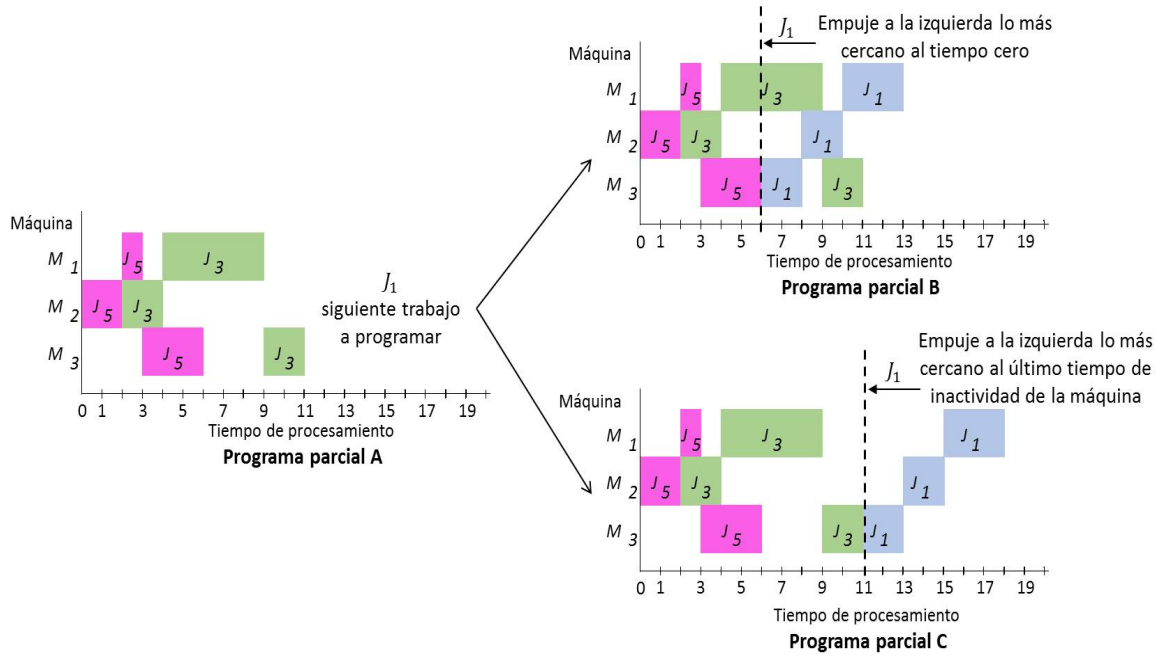

Figura 3.9: Programas parciales de la secuencia de trabajos (J_5,J_3,J_1,J_4,J_2): En el programa parcial A, los trabajos J_5 y J_3 se empujan a la izquierda lo más cerca posible al tiempo cero. En el Programa B, el trabajo J_1 se empuja a la izquierda lo más cerca posible al tiempo cero. En el programa parcial C, el trabajo J_1 se empuja a la izquierda lo más cerca posible al tiempo de inactividad más reciente en la máquina M_3.

Una vez explicadas las dos estrategias, y retomado el problema de la Tabla 3.2 y la secuencia de trabajos (J_5,J_3,J_1,J_4,J_2).

Si todos los trabajos se empujan a la izquierda lo más cerca posible del tiempo cero y siguiendo la secuencia mencionada, la programación obtenida se muestra en la Figura 3.10a con un $C_{max} = 20$.

Mientras que si dejamos que el trabajo J_1 sea empujado a la izquierda lo más cerca posible al tiempo de inactividad más reciente de su máquina correspondiente, y el resto de los trabajos J_5,J_3,J_2 y J_4 se empujan a la izquierda lo más cerca posible al tiempo cero se logra una mejora al reducir el C_{max}= 18, como se muestra en la Figura 3.10b.

Es importante aclarar que esta mejora en el C_{max} no se garantiza que siempre ocurra. Tal es el caso que se muestra en la Figura 3.11b, donde se observa que si el trabajo J_2 se empuja a la izquierda tan cerca como sea posible al tiempo de inactividad más reciente en la máquina M_2, se obtiene un peor programa con un valor $C_{max} = 27$, que es superior al obtenido en Figura 3.11a.

Tabla 3.2: NWJSSP instance, $n = 5$, $m = 3$.

Job	Machine route (Processing time)		
J_1	$M_3(2)$	$M_2(2)$	$M_1(3)$
J_2	$M_2(3)$	$M_1(2)$	$M_3(3)$
J_3	$M_2(2)$	$M_1(5)$	$M_3(2)$
J_4	$M_1(2)$	$M_3(2)$	$M_2(2)$
J_5	$M_2(2)$	$M_1(1)$	$M_3(3)$

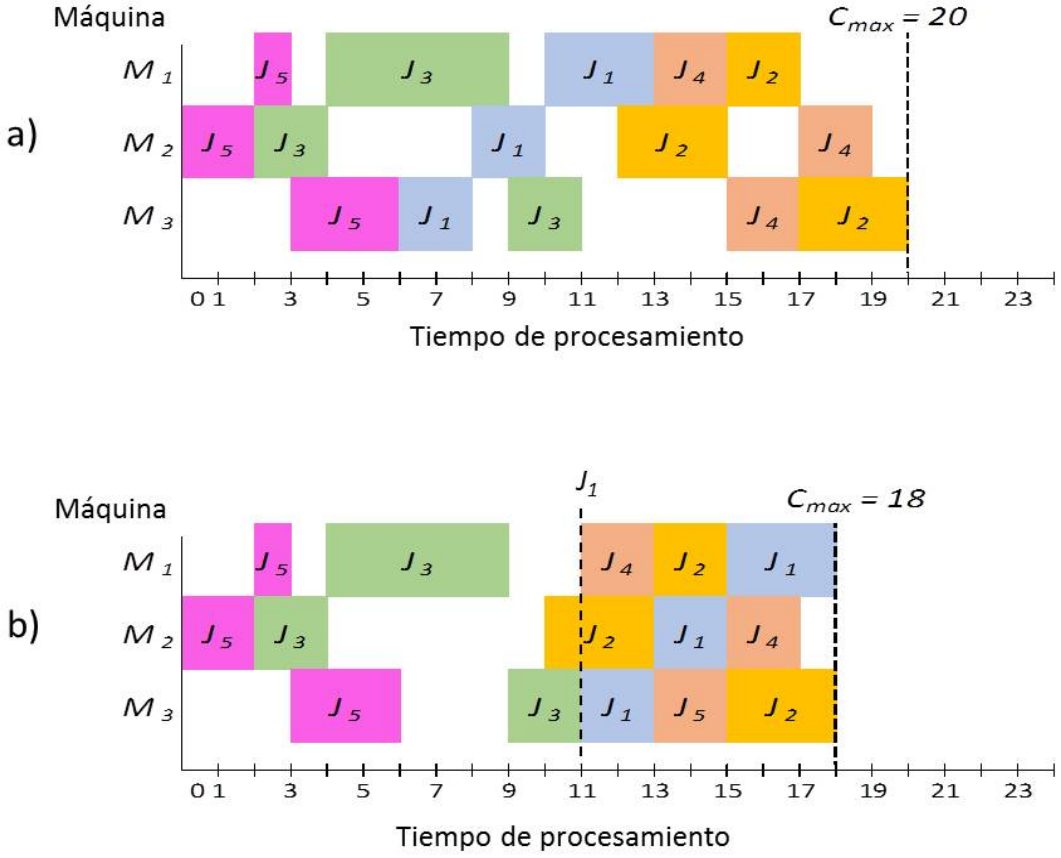

Figura 3.10: Gantt obtenido para la secuencia de trabajos (J_5,J_3,J_1,J_4,J_2). En a) todos los trabajos fueron empujados a la izquierda lo más cerca posible al tiempo cero, en b) todos los trabajos se empujan a la izquierda lo más cerca posible al tiempo cero excepto el trabajo J_1, que se empuja a la izquierda lo más cerca posible al tiempo de inactividad más reciente en la máquina M_3.

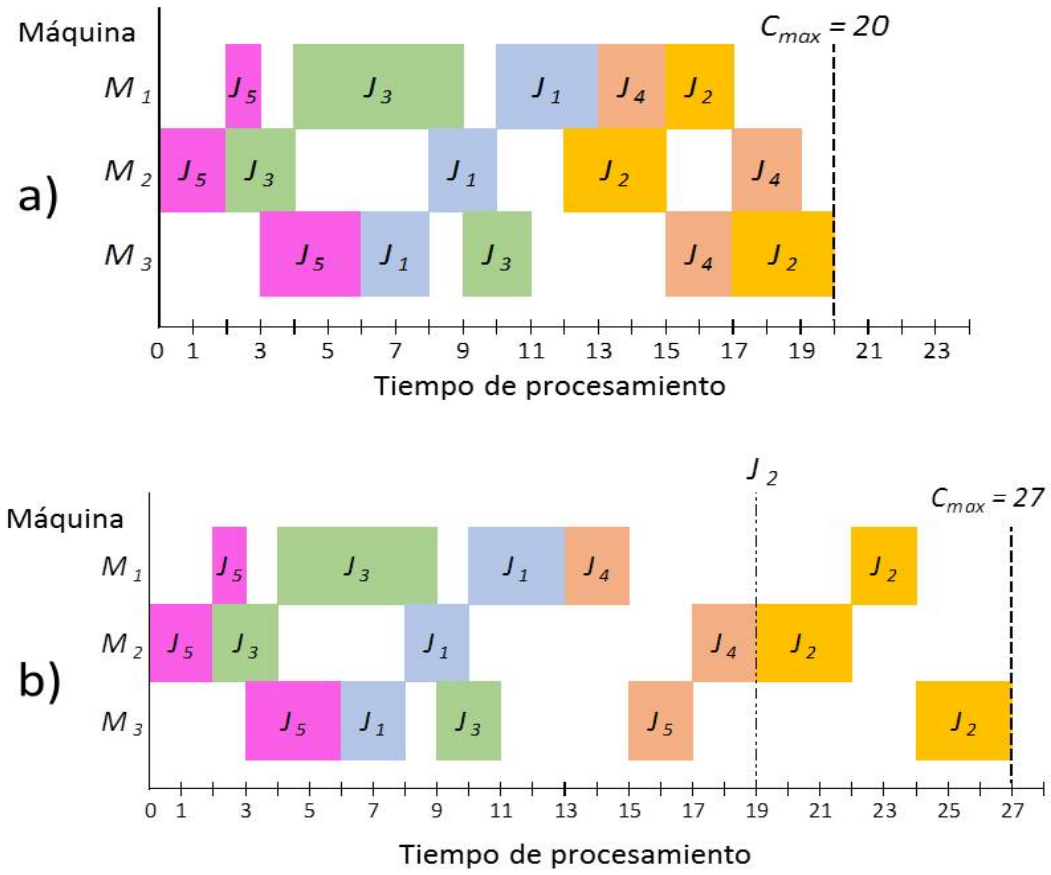

Figura 3.11: Gantt obtenido para la secuencia (J_5,J_3,J_1,J_4,J_2). En a) todos los trabajos se empujan la izquierda lo más cerca posible del tiempo cero, en b) todos los trabajos se empujan a la izquierda lo más cerca posible del tiempo cero excepto el trabajo J_2, que se empuja a la izquierda lo más cerca posible del tiempo de inactividad más reciente en la máquina M_2.

El reto en este problema es decidir qué trabajos deben ser empujados a la izquierda lo más cerca al tiempo cero, y qué trabajos deben ser empujados a la izquierda lo más cerca al tiempo ocioso más reciente en la máquina correspondiente. Para ello, hemos integrado ambas estrategias con un AGCC, que determinará por sí mismo qué trabajos se empujan a la izquierda lo más cerca posible del tiempo cero, y qué trabajos se empujan a la izquierda lo más cerca posible del tiempo de inactividad más reciente en la máquina correspondiente.

La mayoría de las metaheurísticas se basan en la descomposición del NWJSSP en dos sub-problemas: el sub-problema de secuenciación y el sub-problema de programación de horarios. Estas metaheurísticas utilizan una permutación para representar la parte de secuenciación de trabajos, mientras que la en parte de programación de hora-

rios se utilizan reglas específicas para decidir qué trabajos deben programarse lo antes posible y cuáles deben retrasarse. El AGCC que se propone en esta tesis aborda esta decisión co-evolucionando el componente de programación de horarios (representado por una población de cromosomas binarios) con la parte de secuenciación (representada por una población de permutaciones). Ambas poblaciones cooperan para encontrar una solución que minimice el C_{max}. A continuación se describe en detalle el algoritmo genético coevolutivo propuesto.

Como se describe en el Algoritmo 1, el AGCC propuesto comienza con la generación aleatoria de dos poblaciones P_1 y P_2, cada una de tamaño g, una población se forma con cromosomas de permutación, mientras que la otra población consiste en cromosomas binarios. En cada generación, dos padres, Padre 1 y Padre 2 son seleccionados de cada población P_1 y P_2 utilizando la estratgia de selección de la ruleta [67]. Después, los dos padres se utilizan para generar dos hijos mediante los operadores de cruzamiento y mutación, esto se repite hasta generar una población de descendencia de tamaño g. Posteriormente, ambas poblaciones P_1 y P_2 son reemplazadas por sus respectivas poblaciones de descendencia. Tenga en cuenta que los individuos en una población en AGCC solamente codifican un segmento de la solución; por lo que cada individuo tiene que colaborar con un indviduo de la otra población para construir una solución completa y asi evaluar su aptitud utilizando un mecanismo cooperativo, el cual se describe en la sección 3.2.2.

3.2.1. Representación de la solución

Se introduce una representación novedosa, que codifica un programa del NWJSSP combinando un vector de permutación de enteros y un vector binario. Donde, los individuos asociados a una población son codificados como una permutación $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$, $\pi_i \in J$ y los individuos correspondientes a la otra población son codificados como un vector de unos y ceros $b = (b_1, b_2, \ldots, b_n)$, $b_i \in \{0, 1\}$. Todos los individuos son de tamaño igual al número de trabajos del problema.

- Para el individuo π, se utiliza la representación basada en trabajos [14] para codificar una secuencia de trabajos. Cada π es una permutación de trabajos, π_i es el número de trabajo que debe ser programado en la i-ésima posición en su gráfica de Gantt. Por ejemplo, π = (4,5,1,2,3) indica que el trabajo 4 deber ser programado primeramente, posteriormente el trabajo 5, luego el trabajo 1, después el trabajo 2 y finalmente el trabajo 3.

- Para el individuo b, se utiliza la representación binaria para codificar la estrategia de empuje que se aplicará para establecer los tiempos de inicio para cada π_i, donde b_i corresponde a π_i. Si el elemento b_i is igual a "1", significa que el trabajo π_i debe ser empujado a la izquierda lo más cerca posible al tiempo cero; de otra forma, si b_i es igual a "0" el trabajo π_i debe ser empujado a la izquierda pero lo más cercano posible al intervalo de tiempo muerto más reciente de su máquina correspondiente. Por ejemplo, si tenemos b = (1,0,1,1,1) significa que el trabajo correspondiente

Algoritmo 1 : Algoritmo Genético Coevolutivo

Notacion g: tamaño de la población, G_{max}: número máximo de generaciones, Sol_{best}: mejor solución, f_{best}: Aptitud de la mejor solución.

1: $Sol_{best} \leftarrow$ Null, $f_{best} \leftarrow \infty$
2: **for** $i = 1$ to 2 **do**
3: Inicializa la i-ésima población P_i aleatoriamente, $P_i \leftarrow \{P_{i,1}, P_{i,2}, ..., P_{i,g}\}$
4: **end for**
5: **for** $i = 1$ to 2 **do**
6: Evaluación de la población P_i mediante mecanismo de cooperación (Algoritmo 2)
7: **end for**
8: **while** $gen \leq G_{max}$ **do**
9: gen++;
10: **for** $i = 1$ to 2 **do**
11: **for** $j = 1$ to $g/2$ **do**
12: Reproducción de descendencia:
13: $Padre1$, $Padre2 \leftarrow$ (Selección *rueda-ruleta* (P_i))
14: $Hijo1$, $Hijo2 \leftarrow$ (cruzamiento y mutación ($Padre1$, $Padre2$))
15: Los hijos se agregan a la población de descendencia: $P_i^{'} \leftarrow (Hijo1, Hijo2)$
16: **end for**
17: **for** $i = 1$ to 2 **do**
18: Evaluar población $P_i^{'}$ mediante mecanismo de cooperación (Algoritmo 2).
19: $P_i \leftarrow P_i^{'}$
20: $P_i^{'} \leftarrow \{\}$
21: **end for**
22: **end for**
23: **end while**
24: **regresa** f_{best}, Sol_{best}

Algoritmo 2 : Mecanismo de cooperación para AGCC

Entrada: g: tamaño de la población, población P_1 y P_2, Sol_{best}: mejor solución, f_{best}: aptitud de la mejor solución, d: umbral para aplicar los operadores de perturbación.
Salida: Sol_{best}: mejor solución, f_{best}: aptitud de la mejor solución.

```
1: for i = 1 to 2 do
2:     for j = 1 to g do
3:         Sol_j ← colaboración (P_{i,j}, P_{y,z}), P_{y,z} es el z-ésimo individuo
4:         de la población y, z es seleccionado aleatoriamente en {1, ..., g}, i ≠ y, y = {1, 2}.
5:         f_j ←evaluar la aptitud (Sol_j) (ver Figura 3.14)
6:         if (f_j − f_best) < (f_best ∗ d%) then
7:             Aplicar MBPL (f_j, Sol_j) (ver Algoritmo 3)
8:         end if
9:         if (f_j < f_best) then
10:            f_best ← f_j,
11:            Sol_best ← Sol_j
12:        end if
13:    end for
14: end for
15: regresa f_best, Sol_best
```

a π_1 será empujado a la izquierda lo más cerca posible al tiempo cero, luego el trabajo correspondiente a π_2 debe ser empujado a la izquierda lo más cerca posible al intervalo de tiempo muerto más reciente de su máquina correspondiente y los trabajos correspondientes a π_3, π_4 y π_5 deberán ser empujados a la izquierda lo más cerca posible al tiempo cero.

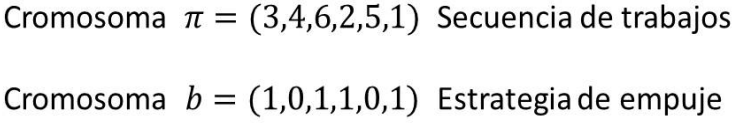

Figura 3.12: Ilustración de la representación de la solución para π = (3,4,6,2,5,1) y b = (1,0,1,1,0,1).

La Figura 3.12 representa una solución de una posible configuración para el caso de prueba Ft06 [72] de la Tabla 3.3. Este caso de prueba tiene 6 trabajos (J_1,J_2,J_3,J_4,J_5,J_6), cada trabajo tiene 6 operaciones que deben ser procesadas en 6 máquinas (M_1,M_2,M_3,M_4, M_5,M_6), ($n = 6$, $m = 6$).

En la Figura 3.12, el primer cromosoma π = (3,4,6,2,5,1) indica que el trabajo 3 debe ser programado en primer lugar, después el trabajo 4, posteriormente el trabajo 6, luego el trabajo 2, seguido el trabajo 5 y al final el trabajo 1. El segundo cromosoma

b = (1,0,1,1,0,1) indica que todos los trabajos deben ser empujados a la izquierda lo más cerca posible al tiempo cero con excepción de los trabajos 4 y 5 (son los trabajos que tienen asociado un cero en el cromosoma b). En la siguiente sección se muestra cómo decodificar esta solución en un programa final (gráfica de Gantt).

Tabla 3.3: NWJSSP Ft06 Instance, $n = 6$, $m = 6$.

Job	Secuencia en la máquina (Tiempo de procesamiento)					
J_1	$M_3(1)$	$M_1(3)$	$M_2(6)$	$M_4(7)$	$M_6(3)$	$M_5(6)$
J_2	$M_2(8)$	$M_3(5)$	$M_5(10)$	$M_6(10)$	$M_1(10)$	$M_4(4)$
J_3	$M_3(5)$	$M_4(4)$	$M_6(8)$	$M_1(9)$	$M_2(1)$	$M_5(7)$
J_4	$M_2(5)$	$M_1(5)$	$M_3(5)$	$M_4(3)$	$M_5(8)$	$M_6(9)$
J_5	$M_3(9)$	$M_2(3)$	$M_5(5)$	$M_6(4)$	$M_1(3)$	$M_4(1)$
J_6	$M_2(3)$	$M_4(3)$	$M_6(9)$	$M_1(10)$	$M_5(4)$	$M_3(1)$

3.2.2. Mecanismo de cooperación para el AGCC

La evaluación de un individuo de una población requiere cooperación con un individuo de la otra población. Como se presenta en el Algoritmo 2, el mecanismo de cooperación utilizado en el AGCC propuesto funciona de la siguiente manera: para cada individuo de la población P_1, un individuo de la población P_2 es seleccionado aleatoriamente, luego ambos individuos se combinan, la combinación resultante es evaluada, y su valor de aptitud es calculado. De manera análoga, cada individuo de la población P_2 se combina y evalúa. Como se ilustra en la Figura 3.13, las poblaciones en AGCC evolucionan por separado y solo comparten información cuando los individuos se evalúan como se explica a continuación.

La Figura 3.14 muestra un ejemplo donde una solución completa proviene de individuos de las poblaciones P_1 y P_2, el individuo evaluado en cada población se combina con un individuo seleccionado aleatoriamente de la otra población. En cuanto al individuo "41523" de la población P_1, su evaluación respecto a la aptitud se realiza combinándolo con un colaborador de la otra población P_2 ("00101") seleccionado aleatoriamente. En consecuencia, el valor de aptitud de "41523" es 21.

3.2.3. Decodificación propuesta

Una versión modificada del algoritmo de decodificación propuesto en [39] se aplica para decodificar una solución. Esta decodificación asegura la factibilidad de la solución, para mostrar cómo funciona denotaremos la programación o permutación de los trabajos a procesar como $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$, $\pi_i \in J$ y un vector binario $b = (b_1, b_2, \ldots, b_n)$, $b_i \in \{0, 1\}$. Con esta notación, el enfoque de decodificación de la solución se describe en la Figura 3.15. Los detalles se dan en la siguiente sección.

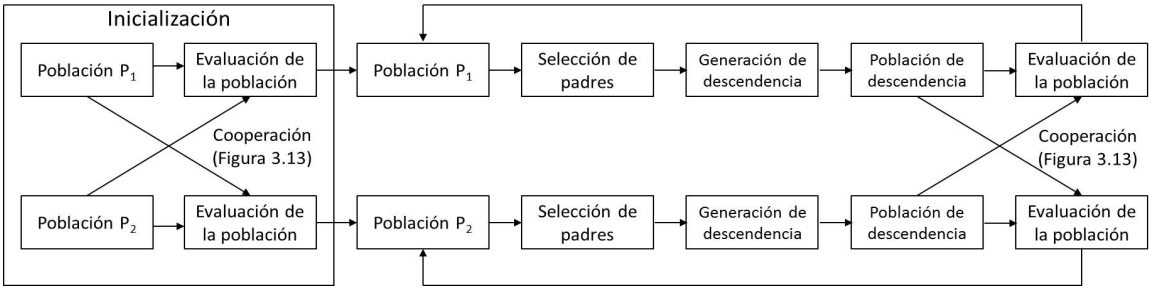

Figura 3.13: Componentes y mecanismo de cooperación del AGCC propuesto.

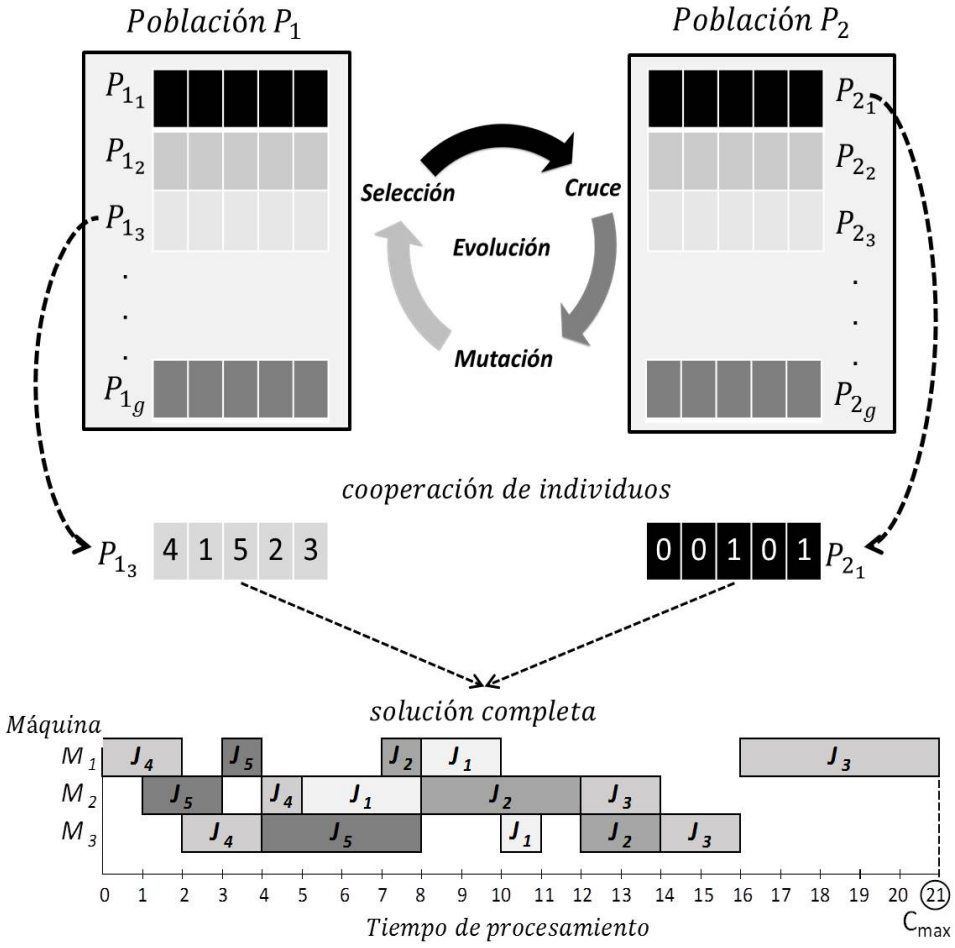

Figura 3.14: Ilustración de una solución completa en el AGCC.

Algoritmo Decodificación de Empujes (ADE)

La estructura general del ADE propuesto puede describirse como sigue:

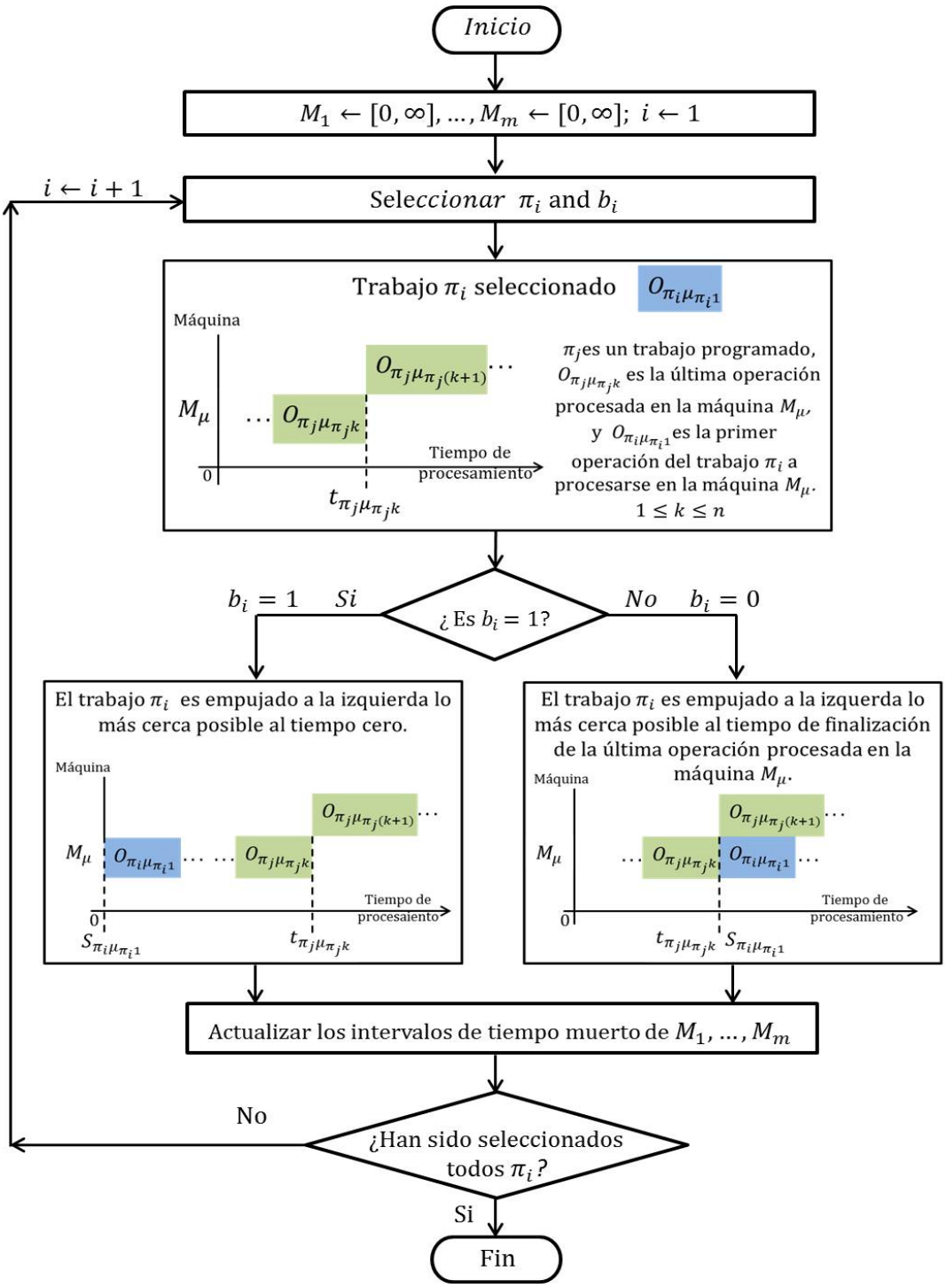

Figura 3.15: Diagrama de flujo del Algoritmo Decodificación de Empujes (ADE).

Paso 1 Asignar intervalos de tiempo muerto de cero a infinito a cada máquina, donde los intervalos de tiempo muerto son diferencias de tiempo entre dos operaciones consecutivas en la misma máquina. Inicializar $i = 1$.

Paso 2 Seleccione π_i del vector de permutación y b_i del vector binario.

Paso 3 Aplicar la siguiente estrategia de empuje:

- Si el elemento b_i seleccionado es igual a "1", entonces:
  Empezando desde el tiempo cero, encontrar todas las coincidencias entre los intervalos de tiempo de inactividad de las máquinas, y el tiempo de procesamiento de todas las operaciones del trabajo π_i, de tal manera que no se viole la restricción de no-espera.
- Si el elemento b_i seleccionado es igual a "0", entonces:
  Empezando desde el intervalo de tiempo muerto más reciente de la máquina correspondiente, encontrar todas las coincidencias entre los intervalos de tiempo muerto de las máquinas, y el tiempo de procesamiento de todas las operaciones del trabajo π_i de tal manera que no se viole la restricción de no-espera. En ambos casos "1" o "0", si el tiempo de procesamiento de cualquier operación no coincide con el intervalo de tiempo muerto de su máquina correspondiente, se retrasa el tiempo de inicio del trabajo π_i, y vuelve a comprobar que el tiempo de procesamiento de todas sus operaciones coincida con los intervalos de tiempo muerto de sus máquinas correspondientes y se respete la restricción de no-espera.

Paso 4 Una vez que el tiempo de inicio del trabajo π_i ha sido fijada, se actualizan los intervalos de tiempo muerto de todas las máquinas.

Paso 5 Si todos los trabajos de la permutación π han sido programados, el procedimiento termina. En caso contario, $i = i + 1$ y volver al Paso 2.

La Figura 3.16 muestra el programa obtenido al aplicar el ADE de la solución mostrada en la Figura 3.12. Se calcula la aptitud de la solución en función del makespan (C_{max}). En este caso el C_{max} de la solución mostrado en la Figura 3.16 es 87.

3.2.4. Función objetivo

En general, el objetivo del NWJSSP es encontrar una secuencia de trabajos en la que sus tiempos de inicio correspondientes satisfagan la restricción de no espera y, al mismo tiempo, optimicen una función objetivo.

En esta tesis, la función objetivo es el tiempo máximo de finalización entre todos los trabajos (C_{max} definido por la ecuación 2.1), y el algoritmo busca una solución que la minimice.

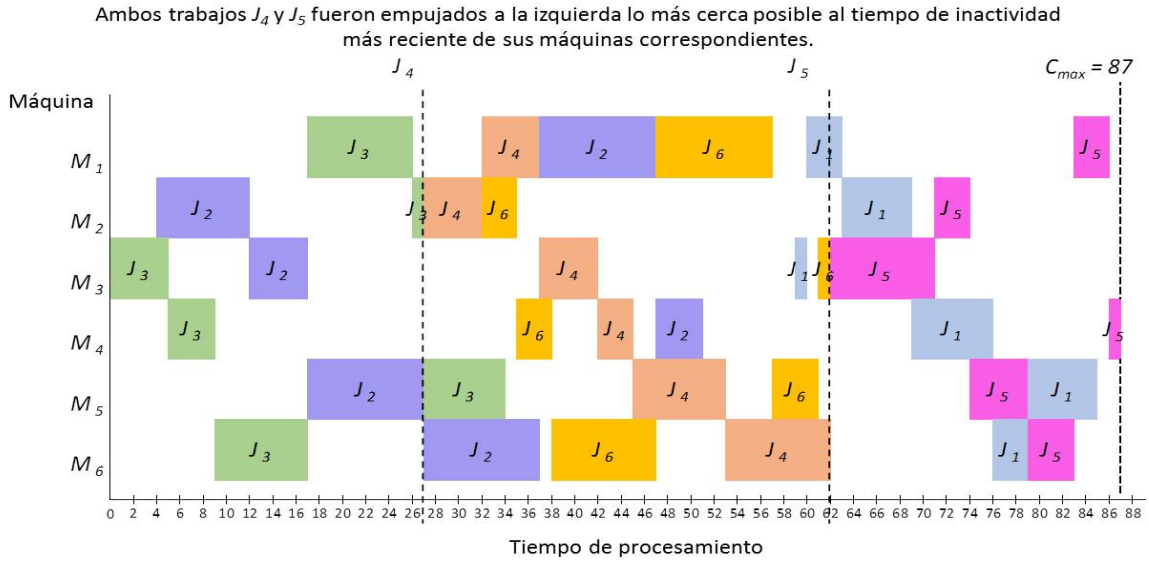

Figura 3.16: Programa final después de aplicar ADE a la permutación $\pi = (3, 4, 6, 2, 5, 1)$ y vector binario $b = (1, 0, 1, 1, 0, 1)$.

3.2.5. Generación de la población inicial y selección de padres

En ambas poblaciones P_1 y P_2, los g individuos son generados aleatoriamente, $P_1 = \{P_{1,1}, P_{1,2}, ..., P_{1,g}\}$ y $P_2 = \{P_{2,1}, P_{2,2}, ...P_{2,g}\}$, donde el individuo $P_{1,i}$ es el cromosoma de la permutación y el individuo $P_{2,j}$ es el cromosoma binario. La combinación de $P_{1,i}$ y $P_{2,j}$ representa una solución (programa) para el problema de programación tipo no-wait job shop y el ADE descrito anteriormente se aplica para generar una solución factible y calcular el valor de aptitud correspondiente.

Por otra parte, la técnica de selección por ruleta [67] se aplica para la selección de los padres. Como se mencionó en la sección 3.1.4, los padres son seleccionados con una probabilidad directamente proporcional a sus valores de aptitud. Los individuos con la mayor aptitud tienen más probabilidad de ser elegidos, es decir, el individuo más apto ocupa el segmento más grande, mientras que el menos apto tiene un segmento correspondientemente más pequeño dentro de la rueda de la ruleta.

3.2.6. Operador de cruzamiento

- **Cruzamiento ordenado de 1-punto** [69] es utilizado para la población P_1. Este operador es una adaptación para los problemas de permutación basada en el cruzamiento de un punto [68]. En este operador, primeramente se selecciona un punto de cruce de forma totalmente aleatoria, después el Padre 1 y el Padre 2 se dividen en este punto, el Hijo 1, tiene la primera participación del Padre 1 (man-

teniendo su posición y orden) y la segunda participación del Padre 2, eliminando los trabajos duplicados y seleccionando sólo los restantes (esto asegura soluciones factibles). De igual manera el Hijo 2, se forma con la primera partición del Padre 2 y la segunda partición del Padre 1. Un ejemplo al respecto se muestra en la Figura 3.17a.

- **Cruzamiento de 1-punto** [65] es considerado para la población binaria P_2. Primero, se selecciona aleatoriamente un punto de cruce. Después cada padre es dividido en ese punto, el Hijo 1 se forma con la primera mitad del Padre 1 y la segunda mitad del Padre 2, así mismo, el Hijo 2 se genera con la primera mitad del Padre 2 y la segunda mitad del Padre 1. La Figura 3.17b ilustra un ejemplo de este operador.

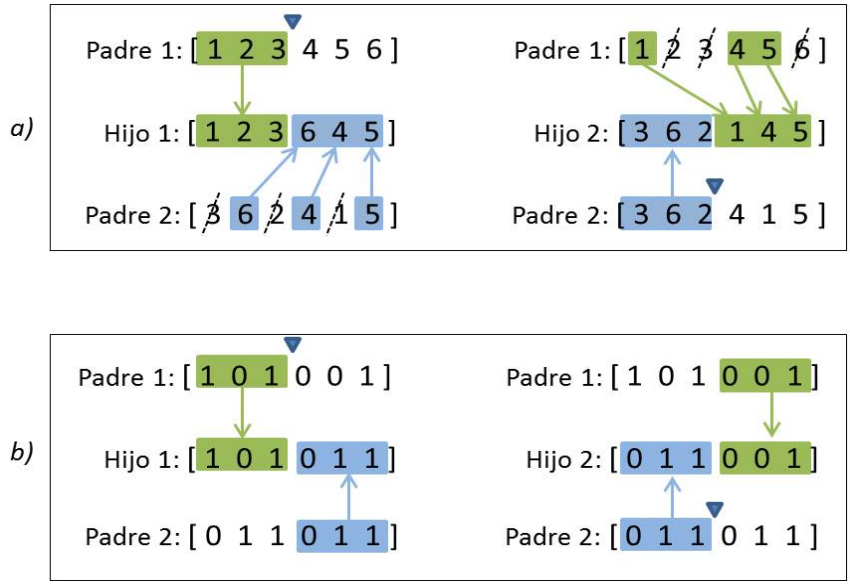

Figura 3.17: Operadores de cruzamiento: a) cruzamiento ordenado de 1-punto para la población de permutaciones P_1. b) Cruzamiento de 1-punto para la población binaria P_2. El símbolo "▼" señala el punto de cruce.

3.2.7. Operadores de mutación

- **Mutación de intercambio 2-posiciones** [73] Este operador se aplica a las permutaciones en la población P_1, que consiste en seleccionar aleatoriamente dos posiciones diferentes en el cromosoma para después intercambiar sus genes (trabajos). La Figura 3.18a presenta un ejemplo de este operador de mutación.

- **Mutación** $Bit-wise$ [71] Este operador es considerado para la población binaria P_2, que consiste en cambiar aleatoriamente (de 0 a 1 o de 1 a 0) cada gen del cromosoma binario con una probabilidad de mutación p_m. Un ejemplo de este operador de mutación se muestra en la Figura 3.18b.

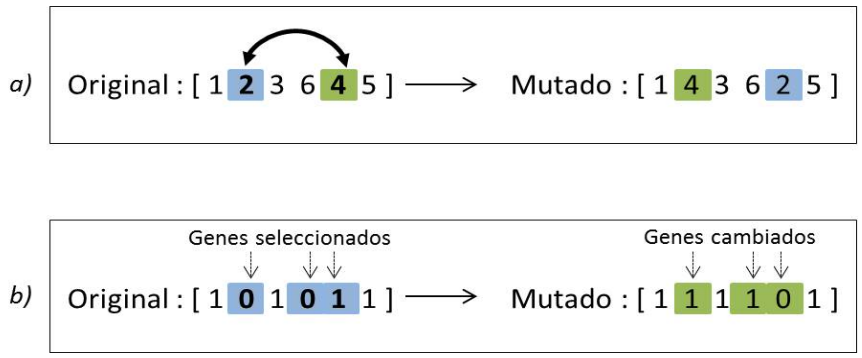

Figura 3.18: Operadores de mutación: a) Mutación de intercambio 2-posiciones para la población de permutaciones P_1. b) Mutación $Bit-wise$ para la población binaria P_2.

3.2.8. Criterio de terminación

Cuando se alcanza un número predefinido de generaciones G_{max}, el algoritmo se detiene y devuelve como salida el mejor programa encontrado con su correspondiente C_{max}.

3.2.9. Mejoramiento mediante Perturbación Local

La mayoría de los algoritmos de la literatura [24, 27, 54, 55, 57, 58, 62, 63] incluyendo dos de los mejores enfoques [23, 25] aplicó un algoritmo de búsqueda local (BL) al problema de programación del tipo no-wait job shop como estrategia para mejorar las soluciones, por esta razón exploramos si agregar o no un enfoque de perturbación local a nuestro AGCC-ADE puede ayudar o no a mejorar la calidad de las soluciones.

Para ello, se implementa una estrategia de perturbación local denominada estrategia de perturbación local múltiple (EPLM), en el que un enfoque de perturbación local de 1-paso es seleccionado de una lista de procedimientos de perturbación local (LPPL) que luego se aplica a la solución. Los pasos se describen con más detalle en el Algoritmo 3.

EPLM es aplicado a la solución actual sólo cuando la diferencia entre su aptitud, y la aptitud de la mejor solución encontrada hasta el momento es menos que $d\,\%$ de la aptitud de la mejor solución actual [23], $d\,\%$ es un parámetro definido por el usuario. Esta estrategia evita aplicar la perturbación local a cada solución, lo que permite evitar un alto costo de cómputo.

Algoritmo 3 : Estrategia de perturbación local múltiple (EPLM)

Notación Sol^o: solucion actual, f^o: valor en la función objetivo de Sol^o, Sol^{best}: mejor solucion, f^{best}: valor de apitud de Sol^{best}, $V = \{Sol^1, Sol^2, \ldots, Sol^{|V|}\}$ vecinos de Sol^o.

```
1: procedure EPLM(Sol^o, f^o)
2:     set Sol^best ← Sol^o, f^best ← f^o, mejora ← verdadero
3:     while mejora do
4:         mejora ← falso
5:         seleccionar aleatoriamente una PL de LPPL
6:         V ← generar un vecindario PL(Sol^o)
7:         for i = 1 to |V| do
8:             f^i ← evaluarAptitud(Sol^i)
9:             if (f^i < f^best) then
10:                f^best ← f^i
11:                Sol^best ← Sol^i
12:                Sol^o ← Sol^i
13:                mejora ← true
14:                break
15:            end if
16:        end for
17:    end while
18: output Sol^best, f^best
```

Cada una de las perturbaciones locales (PL) incluidas en el $LPPL$ funciona de la siguiente manera:

1) *Partial pair-wise exchange* [21]: primeramente, la secuencia de trabajos se divide en k sub-secuencias. Cada sub-secuencia tiene $\lfloor n/k \rfloor$ trabajos, $k \in \{2, \lfloor n/3 \rfloor\}$. En algunos casos, la última sub-secuencia podría tener un número diferente de trabajos que el resto de las sub-secuencias (por ejemplo con $k = 3$ y $n = 20$, la última sub-secuencia tendrá seis trabajos). El primer trabajo de una sub-secuencia se llama *trabajo principal* y al resto se les llama *trabajos miembros*. Después, el procedimiento comienza a intercambiar el *trabajo principal* con sus *trabajos miembros* uno por uno para cada sub-secuencia. Si el valor C_{max} mejora después de intercambiar los trabajos, la nueva configuración se incluye en la solución, y devuelve la nueva solución. De lo contrario, el *trabajo principal* se intercambia con el siguiente *trabajo miembro* de la sub-secuencia. La Figura 3.19 muestra un ejemplo de la secuencia de trabajos (1,2,3,4,2,5,7,8,9), donde la búsqueda local genera 6 permutaciones al intercambiar el *trabajo principal* y sus *trabajos miembros* en cada sub-secuencia.

2) *Simple 1-insertion* [21]: Como en el procedimiento anterior, la secuencia de trabajo se divide en k sub-secuencias, luego el procedimiento comienza moviendo la primera sub-secuencia a cada posible posición una vez a la vez. Si la calidad de

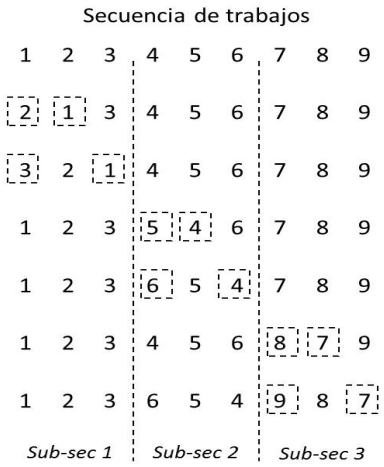

Figura 3.19: Partial Pair-Wise Exchange [21].

la solución se mejora después de mover la sub-secuencia, este cambio se incluye en la secuencia de trabajo, y se devuelve la nueva solución. De lo contrario, el procedimiento se repite para la siguiente sub-secuencia. La Figura 3.20 ilustra un ejemplo en el que se generan 5 vecinos al aplicar este procedimiento.

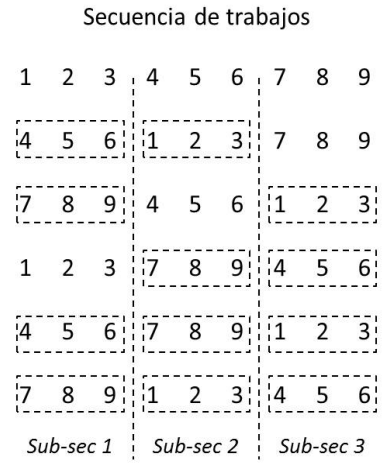

Figura 3.20: Simple 1-insertion [21].

3) *Intercambio adyacente y no adyacente* [74]: este procedimiento comienza con una posición i seleccionada aleatoriamente de la secuencia de trabajo, luego comprueba todas las $(n-1)$ posiciones de intercambio posibles. Si la calidad de la solución mejora después de intercambiar las posiciones, este resultado se incluye en la solución, y la nueva solución es devuelta. De lo contrario, este procedimiento

se repite para la siguiente posición de intercambio. La Figura 3.21 muestra un ejemplo en el que esta estructura de vecindad genera 9 permutaciones diferentes para la secuencia de trabajo (1,2,3,4,2,5,7,8,9).

Figura 3.21: Intercambio adyacente y no adyacente [74].

4) *Inserción adyacente y no adyacente* [74]: este procedimiento es una variante de la perturbación local anterior, comienza con una posición i seleccionada aleatoriamente de la secuencia de trabajo y considera todas las $(n-1)$ posiciones de inserción posibles en la secuencia de trabajo. Si la calidad de la solución mejora después de insertar el trabajo π_i, entonces este cambio se incluye en la solución, y la nueva solución se devuelve. De lo contrario, este procedimiento se repite para la siguiente posición de inserción en la secuencia de trabajo.

5) *Procedimiento cambio de bits* [73]: este procedimiento comienza con la posición inicial del vector binario y su correspondiente bit se cambia de 0 a 1 o de 1 a 0 según sea el caso. Si la calidad de la solución mejora después de cambiar el bit, entonces este cambio se incluye en la solución, y se regresa la nueva solución. De lo contrario, este procedimiento se repite para la siguiente posición en el vector binario.

Figura 3.22: Inserción adyacente y no adyacente [74].

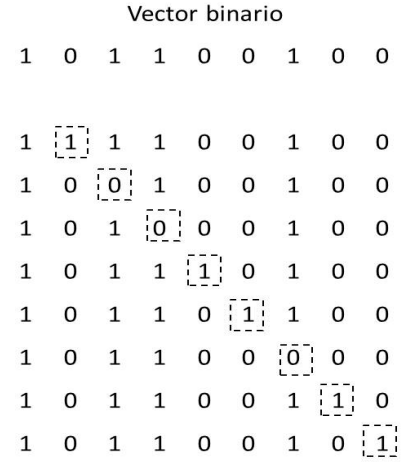

Figura 3.23: Procedimiento cambio de bits [73].

Capítulo 4

Pruebas y Resultados

En este capítulo se detallan y analizan los resultados obtenidos por los algoritmos Mix-D/AG (sección 3.1) y AGCC-ADE (sección 3.2) descritos previamente.

4.1. Preparación de las pruebas experimentales

Los algoritmos propuestos fueron programados en C++, compilado con la versión 7.3.0 de gcc y ejecutado en un sistema operativo basado en Ubuntu 18.10 con un procesador Intel® Core™ i7-4900MQ @ 2.80GHz con 16 GB de RAM. Los algoritmos Mix-D/AG y AGCC-ADE fueron evaluados en 62 y 90 casos de prueba respectivamente. Estos casos de prueba que son ampliamente conocidos en la literatura son de Lawrence (1984), Fisher y Thompson (1963), Applegate y Cook (1991), Storer y otros (1992), Yamada y Nakano (1992), Adams y otros (1988) y Tailard (1988). El total de casos de prueba se agrupan en 4 conjuntos: tamaño pequeño, tamaño grande-A, tamaño grande-B1 y tamaño grande-B2 que tienen 21, 40, 9 y 20 casos de prueba, respectivamente.

Para cada caso de prueba, los algoritmos fueron ejecutados 20 veces con semillas aleatorias separadas. Los resultados obtenidos por Mix-D/GA se compararon con CLLM [21], MCLM [22], HABC [23], HH-EA/G [24] y Ozolins [26]. Mientras que, los resultados del AGCC-ADE fueron comparados con los resultados obtenidos por: CLLM [21], MCLM [22], HABC [23], MSA-BST [25], GASA [27], TS [28], y HTS [29].

Los valores de los parámetros principales para los algoritmos propuestos, se establecieron de la siguiente manera:

- Para Mix-D/AG: 1500 generaciones, tamaño de la población 300, tasa de cruzamiento 1.0 y tasa de mutación 0.6.
- Para AGCC-ADE: 100 generaciones, tamaño de la población de 600 individuos y tasa de cruzamiento 1.0 para ambas poblaciones P_1 y P_2, tasa de mutación 0.06 para la población P_1, tasa de mutación 0.03 para la población P_2 y d=10 % para EPLM.

El rendimiento de los algoritmos se midió utilizando las métricas:

1. Porcentaje de Desviación Relativa (PRD, por sus siglas en inglés Percentage Relative Deviation):
$$PRD = \frac{Best_{Alg} - BKS}{BKS} \times 100\,\% \tag{4.1}$$
donde $Best_{Alg}$ es el mejor valor obtenido por el algoritmo y BKS es la solución (C_{max}) mejor conocida para el caso de prueba.

2. Porcentaje de Desviación Relativa Promedio (APRD, por sus siglas en inglés Average Percentage Relative Deviation):
$$APRD = \frac{Avg_{Alg} - BKS}{BKS} \times 100\,\% \tag{4.2}$$
donde Avg_{Alg} es el valor promedio obtenido por el algoritmo en las 20 ejecuciones.

3. Tiempo de ejecución (T): es el promedio del tiempo de ejecución en segundos sobre las 20 ejecuciones para cada caso de prueba obtenida por el algoritmo.

Teniendo en cuenta que los algoritmos CLLM, MCLM, HABC y MSA-BST se ejecutaron con diferentes recursos computacionales, es difícil comparar con precisión las eficiencias de los algoritmos comparados. Para resolver esto, Ying and Lin (MSA-BST) [25] ajustaron los tiempos de ejecución de estos algoritmos, usando la siguiente ecuación:

$$T\ original \times (puntuación\ de\ computadora\ A / puntuación\ de\ computadora\ B) \tag{4.3}$$

donde la computadora A (PC con un procesador Pentium 4) fue la que se utilizó para ejecutar los algoritmos CLLM, MCLM y HABC, dando una puntuación de 818. La computadora B (Intel® Core™ i7-920 @ 2.67GHz processor) fue usada para ejecutar el algoritmo MSA-BST, dando una puntuación de 1165. Estos puntajes fueron tomados de los puntos de referencia de la CPU (http://www.cpubenchmark.net/singleThread.html).

De igual forma, el algoritmo ACC-ADE propuesto en esta tesis se ejecutó con un procesador que es 7 % más rápido (Intel® Core™ i7-4900MQ @ 2.80GHz) que la del procesador utilizado por MSA-BTS [25] (Intel® Core™ i7-920 @ 2.67GHz). Esta información fue tomada del sitio WEB CPU UserBenchmark (https://cpu.user benchmark.com/Compare/Intel-Core-i7-920-vs-Intel-Core-i7-4900MQ/1981vs1994).

Para tener en cuenta esta diferencia en la velocidad del procesador, hemos modificado la ecuación 4.3 para ajustar el tiempo de ejecución de nuestro algoritmo, de la siguiente manera:

$$[T\ original \times (puntuación\ de\ computadora\ A / puntuación\ de\ computadora\ B] \times r \tag{4.4}$$

donde r es un factor de ajuste que depende de la velocidad del procesador. Dada la velocidad del procesador utilizado en este estudio, $r = 1.07$.

4.2. Resultados y discuciones del algoritmo Mix-D/AG

Los experimentos computacionales se componen de tres partes. En la primera parte se comparan los resultados obtenidos por el algoritmo genético utilizando de forma independiente cada una de las decodificaciones, primeramente la decodificación estándar que llamaremos DE/AG y posteriormente la decodificación reversa que indentificaremos como DR/AG. En la segunda parte se comparan los resultados obtenidos por DE/AG y DR/AG ambos con las estrategias de mejora descritos en la sección 3.1.7 (Partial pair-wise exange y Simple 1-insertion). En la tercera parte del experimento se muestran los resultados obtenidos por el algoritmo genético al usar simultáneamente las estrategias DE y DR, llamado mix-D/AG.

Las Tablas 4.1, 4.3, y 4.5 informan de los resultados en 21 casos de tamaño pequeño, mientras que las Tablas 4.2, 4.4, y 4.6 informan de los resultados en 40 casos de tamaño grande-A.

Para cada tabla, las etiquetas de las columnas "Caso" corresponden al nombre del caso de prueba, (n, m) describe el tamaño del caso, y el "BKS" representa el valor de la mejor solución conocida para cada caso. Además, las tablas informan para cada caso las métricas "PRD" y "APRD" definidas por las ecuaciones (4.1) y (4.2), respectivamente. También la última fila de cada tabla reporta el promedio de las métricas de PRD y APRD de cada estrategia.

4.2.1. Comparación de DE/AG y DR/AG

Tanto en la Tabla 4.1 y como en la Tabla 4.2 se observa que cuando se utiliza la estrategia DE para algunos casos de prueba, el valor del PRD obtenido es alto en comparación con DR, mientras que si se utiliza la estrategia DR este valor es mejorado. Por ejemplo, para el caso de prueba La02 (ver Tabla 4.1), el resultado obtenido en el PRD es 2.56 usando la estrategia DE y empleando la estrategia DR el PRD es 0.00.

De la misma manera, en los casos de prueba donde se utiliza la estrategia DR, el valor del PRD obtenido es bajo en comparación con el obtenido por DE, mientras que si se utiliza la estrategia DE el valor obtenido es alto en comparación con DR. Por ejemplo, para el caso La29 (ver Tabla 4.2), el valor obtenido del PRD es 0.20 usando la estrategia DS y al utilizar la estrategia DR el valor que se obtiene es 2.61.

Este comportamiento se presenta en 12 casos de tamaño pequeño de la Tabla 4.1 y en todas los casos de tamaño grande-A de la Tabla 4.2.

Tabla 4.1: Comparación de resultados en casos de prueba de tamaño pequeño

			DE/AG		*DR/AG*	
Caso	(n, m)	BKS	PRD	APRD	PRD	APRD
Ft06	(6,6)	73	0.00	0.00	0.00	0.00
La01	(10,5)	971	0.41	0.95	2.37	4.02
La02	(10,5)	937	2.56	3.35	0.00	0.00
La03	(10,5)	820	3.29	3.29	1.10	2.00
La04	(10,5)	887	0.00	0.02	0.00	0.00
La05	(10,5)	777	0.51	0.51	0.00	0.00
Ft10	(10,10)	1607	0.00	0.00	0.00	0.53
Orb01	(10,10)	1615	0.00	0.00	0.00	0.68
Orb02	(10,10)	1485	0.00	0.89	2.22	2.22
Orb03	(10,10)	1599	0.00	0.00	0.00	0.00
Orb04	(10,10)	1653	0.00	0.00	1.88	3.06
Orb05	(10,10)	1365	0.44	2.17	1.47	1.47
Orb06	(10,10)	1555	0.00	0.00	0.13	0.13
Orb08	(10,10)	1319	0.00	0.00	0.00	0.00
Orb09	(10,10)	1445	0.00	0.75	0.00	0.00
Orb10	(10,10)	1557	4.24	4.69	3.15	3.15
La16	(10,10)	1575	0.00	0.00	0.00	0.00
La17	(10,10)	1371	0.95	2.03	0.00	0.00
La18	(10,10)	1417	6.35	6.35	0.00	3.74
La19	(10,10)	1482	0.61	1.46	0.61	0.61
La20	(10,10)	1526	0.00	0.39	3.54	4.64
Promedio			0.92	1.28	0.78	1.25

Tabla 4.2: Comparación de resultados en casos de prueba de tamaño grande-A

			DE/AG		*DR/AG*	
Caso	(n, m)	BKS	PRD	APRD	PRD	APRD
la06	(15,5)	1248	4.09	5.05	0.00	3.60
la07	(15,5)	1172	3.33	4.40	0.00	3.26
la08	(15,5)	1244	2.01	3.86	4.26	3.47
la09	(15,5)	1358	3.02	5.10	0.74	2.42
la10	(15,5)	1287	2.41	2.61	1.94	4.01
la11	(20,5)	1671	2.87	3.35	0.72	3.68
la12	(20,5)	1452	4.34	6.03	3.17	5.80
la13	(20,5)	1624	1.72	5.16	0.12	4.43
la14	(20,5)	1691	-2.13	3.89	0.59	3.02
la15	(20,5)	1694	4.31	5.11	4.60	6.21
la21	(15,10)	2048	2.44	5.23	6.25	4.79
la22	(15,10)	1887	3.39	4.17	1.32	2.60
la23	(15,10)	2032	2.51	3.78	4.53	4.76
la24	(15,10)	2015	3.03	4.08	3.57	4.41
la25	(15,10)	1917	4.02	4.33	3.55	4.12
la26	(20,10)	2553	4.74	6.35	2.78	5.28
la27	(20,10)	2747	3.68	4.68	3.17	4.17
la28	(20,10)	2624	4.38	6.59	3.28	5.99
la29	(20,10)	2489	0.20	2.39	2.61	3.75
la30	(20,10)	2665	1.46	3.34	5.97	4.39
la31	(30,10)	3745	3.20	5.02	-0.64	3.41
la32	(30,10)	4028	5.78	6.81	4.62	5.16
la33	(30,10)	3749	-1.39	0.98	2.56	3.30
la34	(30,10)	3824	0.47	3.61	0.89	3.31
la35	(30,10)	3760	3.62	6.12	4.60	5.81
la36	(15,15)	2685	6.37	7.21	0.00	5.82
la37	(15,15)	2962	2.33	5.16	5.37	6.69
la38	(15,15)	2617	4.24	4.42	3.78	5.46
la39	(15,15)	2697	1.22	4.95	4.82	5.35
la40	(15,15)	2594	4.70	4.70	5.13	6.11
swv01	(20,10)	2328	2.19	3.54	1.76	4.47
swv02	(20,10)	2418	3.27	4.43	3.06	3.48
swv03	(20,10)	2415	3.73	5.36	2.11	3.77
swv04	(20,10)	2506	1.14	2.71	3.11	4.80
swv05	(20,10)	2333	6.86	7.43	7.93	8.23
swv06	(20,15)	3291	6.35	6.82	4.44	6.74
swv07	(20,15)	3271	2.17	4.61	2.38	4.64
swv08	(20,15)	3530	2.41	3.63	2.27	2.65
swv09	(20,15)	3307	1.54	2.71	3.63	4.32
swv10	(20,15)	3488	4.39	5.44	0.97	4.82
Promedio			3.01	4.63	2.90	4.56

4.2.2. Comparación de DE/AG con DR/AG ambos con estrategia de mejora

Comparando los resultados de los casos de tamaño pequeño obtenidos por el AG sin estrategia de mejora (EM) (descrito en la sección 3.1.7), el AG con EM es mejor en 2 casos usando la decodificación DE y también en 2 casos usando la decodificación DR (ver Tabla 4.3). Además el promedio de los indicadores PRD y APRD obtenidos por el AG (ver Tabla 4.3) con EM son más bajos (mejores) que el AG sin EM (ver Tabla 4.1).

Por otro lado, la comparación de los resultados en casos de tamaño grande-A obtenidos por AG sin EM, el AG con EM es mejor en la mayoría de los casos usando DE o DR (Tabla 4.4). Asimismo, el promedio de los indicadores PRD y APRD obtenidos por AG con EM son mejores que los obtenidos sin EM (ver Tabla 4.4). Además se observa que cuando se utiliza la decodificación DE en un caso dado y el rendimiento del PRD es bajo, este rendimiento mejora utilizando la decodificación DR y viceversa.

Tabla 4.3: Comparación de resultados en casos de tamaño pequeño del AG con estrategia de mejora (EM)

			con EM			
			DE/AG		*DR/AG*	
Caso	(n, m)	BKS	PRD	APRD	PRD	APRD
Ft06	(6,6)	73	0.00	0.00	0.00	0.00
La01	(10,5)	971	0.41	0.41	2.37	2.37
La02	(10,5)	937	2.56	2.56	0.00	0.00
La03	(10,5)	820	3.29	3.29	1.10	2.00
La04	(10,5)	887	0.00	0.02	0.00	0.00
La05	(10,5)	777	0.51	0.51	0.00	0.00
Ft10	(10,10)	1607	0.00	0.00	0.00	0.53
Orb01	(10,10)	1615	0.00	0.00	0.00	0.68
Orb02	(10,10)	1485	0.00	0.89	2.22	2.22
Orb03	(10,10)	1599	0.00	0.00	0.00	0.00
Orb04	(10,10)	1653	0.00	0.00	0.00	2.00
Orb05	(10,10)	1365	0.44	1.00	1.47	1.47
Orb06	(10,10)	1555	0.00	0.00	0.13	0.13
Orb08	(10,10)	1319	0.00	0.00	0.00	0.00
Orb09	(10,10)	1445	0.00	0.75	0.00	0.00
Orb10	(10,10)	1557	1.48	1.48	2.06	2.10
La16	(10,10)	1575	0.00	0.00	0.00	0.00
La17	(10,10)	1371	0.95	0.95	0.00	0.00
La18	(10,10)	1417	0.00	5.50	0.00	3.74
La19	(10,10)	1482	0.61	1.46	0.61	0.61
La20	(10,10)	1526	0.00	0.39	3.54	3.54
Promedio			0.49	0.91	0.64	1.02

Tabla 4.4: Comparación de resultados en casos de tamaño grande-A del AG con estrategia de mejora (EM)

			con EM			
			DE/AG		*DR/AG*	
Caso	(n,m)	BKS	PRD	APRD	PRD	APRD
la06	(15,5)	1248	2.01	6.30	0.00	3.49
la07	(15,5)	1172	3.33	4.22	0.00	0.58
la08	(15,5)	1244	2.01	3.71	0.00	2.44
la09	(15,5)	1358	1.25	3.58	0.74	1.95
la10	(15,5)	1287	2.10	3.83	0.47	2.77
la11	(20,5)	1671	-2.27	1.13	-1.32	0.67
la12	(20,5)	1452	0.34	2.89	3.10	5.69
la13	(20,5)	1624	0.86	4.13	0.12	3.71
la14	(20,5)	1691	-1.77	2.01	-0.59	3.19
la15	(20,5)	1694	2.24	5.50	1.59	4.19
la21	(15,10)	2048	0.29	1.15	0.00	4.60
la22	(15,10)	1887	1.75	1.15	0.95	1.46
la23	(15,10)	2032	2.02	3.67	2.12	3.29
la24	(15,10)	2015	1.89	4.40	1.04	2.18
la25	(15,10)	1917	-0.57	3.86	2.40	3.10
la26	(20,10)	2553	1.80	5.48	1.45	3.86
la27	(20,10)	2747	-2.37	0.82	1.57	5.23
la28	(20,10)	2624	0.34	5.56	1.41	5.55
la29	(20,10)	2489	-0.52	1.05	-0.08	2.64
la30	(20,10)	2665	1.99	3.43	0.60	3.40
la31	(30,10)	3747	-1.36	2.13	-1.09	1.40
la32	(30,10)	4028	2.51	4.68	0.92	4.83
la33	(30,10)	3749	-0.16	2.68	-1.49	1.78
la34	(30,10)	3824	1.28	2.49	-1.26	2.38
la35	(30,10)	3760	0.77	4.26	-0.82	4.03
la36	(15,15)	2685	6.48	6.75	0.00	1.65
la37	(15,15)	2962	1.82	3.25	0.84	3.50
la38	(15,15)	2617	0.00	0.00	1.68	3.69
la39	(15,15)	2697	0.00	3.46	0.63	1.77
la40	(15,15)	2594	0.00	0.37	0.62	6.48
swv01	(20,10)	2328	0.52	3.10	0.52	4.21
swv02	(20,10)	2418	0.54	3.25	-0.04	2.63
swv03	(20,10)	2415	1.20	3.57	1.61	3.54
swv04	(20,10)	2506	0.36	1.61	2.08	3.80
swv05	(20,10)	2333	0.00	7.09	4.29	6.63
swv06	(20,15)	3291	3.37	5.83	3.04	5.68
swv07	(20,15)	3271	-2.54	2.31	-0.18	2.87
swv08	(20,15)	3530	-0.71	2.99	0.85	2.37
swv09	(20,15)	3307	0.00	2.09	0.18	2.53
swv10	(20,15)	3488	0.00	3.94	0.97	2.91
Promedio			0.77	3.34	0.72	3.32

4.2.3. Pruebas no paramétricas de Mann-Whitney-Wilcoxon

Para asegurar que el desempeño del algoritmo Mix-D/AG no es igual que si se ejecutara tanto el algoritmo DE/AG como DR/AG con el doble de tiempo de cómputo utilizado por Mix-D/AG, se realizaron pruebas no-paramétricas mediante la prueba de Mann-Whitney-Wilcoxon.

En primer lugar, la prueba no paramétrica de Wilcoxon (también conocida como Wilcoxon-Mann-Whitney) se llevó a cabo con el fin de probar si existen diferencias estadísticas significativas entre los resultados obtenidos por DE/AG (con el doble de tiempo de cómputo) y la estrategia Mix-D/AG. La prueba se realizó para un nivel de confianza α=0.05 con respecto a PRD. La Figura 4.1 presenta los resultados de esta prueba, la cual indica que hay una diferencia significativa (p-$value$=0.0286 <0.05) en todos los casos de prueba con respecto a PRD, lo que significa que el rendimiento del Mix-D/AG es estadísticamente mejor que la estrategia DE/AG (con el doble de tiempo de cómputo).

Mann-Whitney Test and CI: *mix, SD*

```
     N  Median
mix  40   0.000
SD   40   0.530

Point estimate for η1 - η2 is -0.655
95.1 Percent CI for η1 - η2 is (-1.320,-0.000)
W = 1392.0
Test of η1 = η2 vs η1 ≠ η2 is significant at 0.0286
The test is significant at 0.0278 (adjusted for ties)
```

Figura 4.1: Pruebas de Mann-Whitney-Wilcoxon entre mix-D/AG y DE/AG.

De manera similar, se realizó la prueba no paramétrica de Wilcoxon (para un nivel de confianza α=0.05) entre los resultados obtenidos por DR/AG (con el doble de tiempo de cómputo) y la estrategia Mix-D/AG. La figura 4.2 muestra el resultado de esta prueba, la cual indica que existe una diferencia significativa (p-$value$=0.0047 <0.05) en todos los casos de prueba con respecto a PRD, lo que significa que el rendimiento del Mix-D/AG es estadísticamente mejor que la estrategia DR/AG (con el doble de tiempo de cómputo).

Mann-Whitney Test and CI: *mix-D, RD*

```
       N  Median
Mix-D  40  0.0000
RD     40  0.7900

Point estimate for η1 - η2 is -0.7650
95.1 Percent CI for η1 - η2 is (-1.3798,-0.1798)
W = 1325.5
Test of η1 = η2 vs η1 ≠ η2 is significant at 0.0047
The test is significant at 0.0045 (adjusted for ties)
```

Figura 4.2: Pruebas de Mann-Whitney-Wilcoxon entre mix-D/AG y DR/AG.

4.2.4. Comparación de *mix*-D/AG con enfoques de la literatura

Después de observar los resultados de las Tablas 4.1, 4.2, 4.3, 4.4 es claro que el mejor desempeño del algoritmo genético, es cuando utiliza simultáneamente las estrategias DE y DR, llamamos a esta estrategia *mix*-D/AG y la comparamos con enfoques de la literatura.

En los casos de tamaño pequeño (ver Tabla 4.5), comparando con CLLM, *mix*-D/AG es mejor en 1 caso, peor en 5 casos e igual en 15 casos en términos de PRD. Por otro lado, el promedio de PRD y APRD obtenido por *mix*-D/AG es peor que el de CLLM. De manera similar, comparando con MCLM, *mix*-D/AG es mejor en 5 casos, peor en 5 casos e igual en 11 casos en términos de PRD. El promedio de PRD obtenido por *mix*-D/AG es mejor que el de MCLM. El promedio de APRD obtenido por *mix*-D/AG es peor que el del MCLM. De la misma manera, comparando con HABC, *mix*-D/AG es mejor en 3 casos, peor en 4 casos e igual en 14 casos en términos de PRD. El promedio de PRD y APRD obtenido por *mix*-D/AG es mejor que el del HABC. De forma similar, comparando con HH-AE/G, *mix*-D/AG es mejor en 3 casos, peor en 5 e igual en 13 casos en términos de PRD. El promedio de PRD y APRD obtenido por *mix*-D/AG es mejor que el de HH-EA/G. De manera similar, comparando con los obtenidos por Ozolins [26], *mix*-D/AG es mejor en 3, peor en 3 e igual en 5 casos en términos de PRD. El promedio de PRD obtenido por *mix*-D/AG es mejor que el de Ozolins [26].

Tabla 4.5: Comparación de resultados del *mix*-D/AG en casos de tamaño pequeño

	CLLM		MCLM		HABC		HH-EA/G		Ozolins	*mix*-D/AG	
Caso	PRD	APRD	PRD	APRD	PRD	APRD	PRD	APRD	PRD	PRD	APRD
Ft06	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	—	0.00	0.00
La01	0.00	0.10	0.00	0.02	0.41	0.41	0.00	0.41	0.41	0.41	0.41
La02	0.01	0.21	0.00	0.00	2.56	2.56	2.57	2.57	0.00	0.00	0.00
La03	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.50	0.00	1.10	2.00
La04	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.04	0.11	0.00	0.00
La05	0.00	0.50	0.51	0.90	0.51	0.51	0.51	0.51	0.51	0.00	0.00
Ft10	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Orb01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.04	0.00	0.00	0.00
Orb02	0.00	1.89	2.16	2.16	0.00	0.00	0.00	0.78	0.00	0.00	0.89
Orb03	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.14	0.00	0.00	0.00
Orb04	0.00	0.42	0.00	0.12	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Orb05	0.15	0.15	0.00	0.00	0.37	0.37	0.37	0.37	0.00	0.44	1.00
Orb06	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Orb08	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.04	2.65	0.00	0.00
Orb09	0.00	0.83	0.00	0.00	0.00	0.28	0.00	0.87	0.00	0.00	0.00
Orb10	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.38	0.00	1.48	1.48
La16	0.00	0.00	1.84	1.84	0.00	0.00	0.00	0.00	0.00	0.00	0.00
La17	0.00	0.22	0.00	0.12	0.95	0.95	0.95	0.95	0.00	0.00	0.00
La18	0.00	1.13	2.82	2.82	0.00	5.22	0.00	3.56	0.00	0.00	3.74
La19	0.00	0.14	0.00	0.61	0.00	0.43	0.00	1.01	0.61	0.61	0.61
La20	0.00	0.13	1.31	1.31	0.00	0.00	0.00	0.00	0.00	0.00	0.39
Promedio	0.01	0.27	0.41	0.47	0.23	0.51	0.23	0.58	0.21	0.19	0.50

Además, en los casos de prueba de tamaño grande-A (ver la Tabla 4.6), en comparación con CLLM, *mix*-D/AG es mejor en 13 casos, peor en 16 e igual en 12 en términos de PRD. El promedio de PRD obtenido por *mix*-D/AG es mejor que el de CLLM. El promedio de APRD obtenido por *mix*-D/AG es peor que el de CLLM. De manera similar, comparando con MCLM, *mix*-D/AG es mejor en 8 casos, peor en 28 e igual en 5 en términos de PRD. El promedio de PRD obtenido por *mix*-D/AG es mejor que el de MCLM. El promedio de APRD obtenido por *mix*-D/AG es peor que el del MCLM. De la misma manera, comparando con HABC, *mix*-D/AG es mejor en 2 casos, peor en 32 casos e igual en 7 casos en términos de PRD. El promedio de PRD y APRD obtenido por *mix*-D/AG es peor que el del HABC.

Tabla 4.6: Comparación de resultados del *mix*-D/AG en casos de tamaño grande-A

	CLLM		MCLM		HABC		*mix*-D/AG	
Caso	PRD	APRD	PRD	APRD	PRD	APRD	PRD	APRD
la06	0.00	2.80	0.00	0.75	0.00	0.00	0.00	3.49
la07	0.00	2.47	0.51	2.46	0.00	0.92	0.00	0.58
la08	0.00	2.25	0.00	0.47	0.00	0.15	0.00	2.44
la09	0.00	1.47	0.52	0.73	0.29	0.67	0.74	1.95
la10	0.00	0.70	0.00	0.04	0.54	0.70	0.47	2.77
la11	0.00	2.69	-2.15	-0.58	-2.63	-1.97	-2.27	0.67
la12	0.00	3.72	-1.58	0.57	-1.24	0.00	0.34	2.89
la13	0.00	2.28	-1.17	-0.15	-2.71	-1.47	0.12	3.71
la14	0.00	1.77	-2.54	-1.16	-3.02	-2.24	-1.77	2.01
la15	0.00	3.25	-0.53	1.09	-0.89	-0.09	1.59	4.19
la21	0.00	2.73	0.00	0.11	-0.24	0.27	0.00	1.15
la22	0.00	1.33	0.80	0.99	-1.85	-1.12	0.95	1.15
la23	0.00	3.25	-0.49	1.47	0.00	0.71	2.02	3.29
la24	0.00	1.64	0.00	0.77	-1.04	-0.02	1.04	2.18
la25	0.00	2.82	0.68	2.07	-0.57	-0.57	-0.57	3.10
la26	0.00	6.03	-0.82	1.91	-1.84	0.30	1.45	3.86
la27	0.00	3.31	-1.17	0.28	-2.66	-2.62	-2.37	0.82
la28	0.00	4.88	-2.44	1.77	-2.44	0.62	0.34	5.55
la29	0.00	2.01	-4.90	-2.48	-4.02	-2.70	-0.52	1.05
la30	0.00	2.93	-4.54	-1.51	-7.99	-3.00	0.60	3.40
la31	0.00	3.71	-4.54	-1.40	-4.09	-2.60	-1.36	1.40
la32	0.00	5.74	-4.79	0.56	-2.86	-0.35	0.92	4.68
la33	0.00	2.48	-4.67	-1.52	-5.87	-3.37	-1.49	1.78
la34	0.00	2.82	-3.66	-0.88	-5.60	-2.76	-1.26	2.38
la35	0.00	5.96	-1.65	1.27	-4.44	-0.41	-0.82	4.03
la36	0.00	3.46	1.90	4.98	0.00	0.29	0.00	1.65
la37	0.00	1.92	0.00	0.11	-0.81	0.98	0.84	3.25
la38	0.00	2.25	-3.52	-1.73	-3.52	-1.08	0.00	0.00
la39	0.00	2.93	1.19	2.43	0.22	0.88	0.00	1.77
la40	0.00	4.43	-0.54	-0.54	0.00	0.00	0.00	0.37
swv01	0.00	2.62	0.22	0.37	-0.43	-0.02	0.52	3.10
swv02	0.00	3.10	0.00	0.38	-0.41	-0.02	-0.04	2.63
swv03	0.00	2.82	-1.41	-0.41	-1.41	-0.92	1.20	3.54
swv04	1.44	2.24	-1.76	0.30	0.00	0.29	0.36	1.61
swv05	0.00	6.94	0.00	0.00	0.00	0.00	0.00	6.63
swv06	2.58	5.04	0.00	1.69	0.00	0.40	3.04	5.68
swv07	0.00	1.53	-0.95	-1.59	-2.54	-2.54	-2.54	2.31
swv08	0.00	2.95	-3.04	-2.21	-3.03	-1.78	-0.71	2.37
swv09	0.00	1.66	-1.12	-0.28	-1.84	-0.86	0.00	2.09
swv10	0.00	2.18	-1.06	0.12	-0.75	-0.22	0.00	2.91
Promedio	0.10	3.03	-1.23	0.28	-1.74	-0.64	0.02	2.61

4.3. Resultados y discuciones del algoritmo AGCC-ADE

Esta sección se compone de cuatro partes. En la primera parte se presenta el análisis de la calibración de parámetros del AGCC-ADE. En la segunda parte se comparan los resultados obtenidos por cada componente del AGCC-ADE. En la tercera parte se comparan los resultados obtenidos por el AGCC-ADE con y sin la estrategia de perturbación local múltiple (EPLM). La cuarta parte compara los resultados obtenidos por el AGCC-ADE con los generados por los algoritmos CLLM [21], MCLM [22], HABC [23], MSA-BST [25], GASA [27], TS [28], y HTS [29].

4.3.1. Calibración de parámetros

En el AGCC-ADE se utilizan cinco parámetros clave: la probabilidad de cruzamiento (P_c), la probabilidad de mutación en la población 1 ($P_m(P_1)$), la probabilidad de mutación en la población 2 ($P_m(P_2)$), un umbral para aplicar los operadores de perturbación (d) y el tamaño de la población (g). Para investigar el efecto de estos parámetros en el rendimiento del AGCC-ADE, se realizó un diseño de experimentos con el método Taguchi [75] en 6 casos de prueba seleccionadas aleatoriamente: La18, La34, La37, Ta05, Yn1 y Swv09.

Tabla 4.7: Niveles para cada parámetro en la calibración

Level	P_c	P_m P_1	P_m P_2	d	g
1	0.70	0.04	0.01	0.10	300
2	0.80	0.06	0.02	0.20	400
3	0.90	0.08	0.03	0.30	500
4	1.00	0.10	0.04	0.40	600

Como se muestra en la Tabla 4.7, cada uno de los cinco parámetros tiene cuatro niveles, lo que resulta en una matriz ortogonal L_{16}. Por lo tanto, hay 16 combinaciones de valores de parámetros en total. Para cada combinación, el AGCC-ADE se ejecutó 10 veces de forma independiente y se estableció $120000n$ milisegundos (n es el número de trabajos) como criterio de terminación [25], además, consideramos el promedio de APRD como la variable de respuesta. Obviamente, cuanto menor sea la APRD, mejor será la combinación de los valores de los parámetros. La matriz ortogonal y los valores de respuesta se enumeran en la Tabla 4.8, el rango de significancia de cada parámetro se enumera en la Tabla 4.9, y la tendencia de cada nivel de factor se muestra en la Figura 4.3.

Tabla 4.8: Arreglo ortogonal y valores de APRD.

No.	P_c	P_m P_1	P_m P_2	d	g	APRD
1	0.7	0.04	0.01	0.1	300	11.89
2	0.7	0.06	0.02	0.2	400	12.20
3	0.7	0.08	0.03	0.3	500	12.60
4	0.7	0.10	0.04	0.4	600	11.49
5	0.8	0.04	0.02	0.3	600	12.25
6	0.8	0.06	0.01	0.4	500	12.98
7	0.8	0.08	0.04	0.1	400	11.61
8	0.8	0.10	0.03	0.2	300	11.69
9	0.9	0.04	0.03	0.4	400	12.82
10	0.9	0.06	0.04	0.3	300	12.79
11	0.9	0.08	0.01	0.2	600	10.40
12	0.9	0.10	0.02	0.1	500	11.36
13	1.0	0.04	0.04	0.2	500	11.79
14	1.0	0.06	0.03	0.1	600	6.65
15	1.0	0.08	0.02	0.4	300	12.66
16	1.0	0.10	0.01	0.3	400	12.58

Tabla 4.9: APRDs obtenidos y posición de cada parámetro.

Level	P_c	P_m P_1	P_m P_2	d	g
1	12.04	12.19	11.96	10.38	12.26
2	12.13	11.15	12.12	11.52	12.30
3	11.84	11.82	10.94	12.55	12.18
4	10.92	11.78	11.92	12.49	10.20
Rango	1.21	1.03	1.18	2.18	2.11
Posición	3	5	4	1	2

La Tabla 4.9 presenta el valor de significancia de cada parámetro donde se puede ver que el umbral para la aplicación de los operadores de perturbación d es el parámetro más significativo. Si d es demasiado grande, el algoritmo se vuelve voraz y hace que el algoritmo converja rápidamente a un óptimo local. El segundo lugar lo ocupa el tamaño de la población (g). Si g es demasiado pequeño, el espacio de soluciones no se puede muestrear suficientemente; por el contrario, si g es demasiado grande, el algoritmo convergerá lentamente. Mientras que la probabilidad de cruzamiento (P_c) y la probabilidad de mutación de la población 2 ($P_m(P_2)$) ocupan el tercer y cuarto lugar, respectivamente. En cuanto a $P_m(P_1)$, es el parámetro que tiene la mínima influencia en el rendimiento del AGCC-ADE.

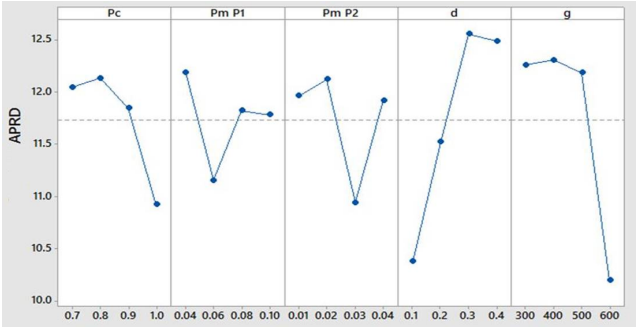

Figura 4.3: Tendencia del nivel de los factores para el AGCC-ADE.

La Figura 4.3 muestra que el parámetro P_c obtiene el valor mínimo de la variable de respuesta en el cuarto nivel. En cuanto al parámetro $P_m(P_1)$, se observa que el valor mínimo de la variable de respuesta se obtiene en el segundo nivel. Una tendencia similar se observa para el parámetro $P_m(P_2)$, el valor mínimo de la variable de respuesta se obtiene en el tercer nivel. En cuanto al parámetro d, el valor mínimo de la variable de respuesta se obtiene en el primer nivel. Por último, en el parámetro g, se observa que el valor mínimo de la variable de respuesta se obtiene en el cuarto nivel.

De acuerdo con el análisis anterior, los parámetros AGCC-ADE se sugieren como $P_c = 1.0$, $P_m(P_1) = 0.06$, $P_m(P_2) = 0.03$, $d = 0.10$, y $g = 600$, que se utilizarán en los siguientes experimentos. Junto con estos parámetros se utilizó un $G_{max} = 100$ para todas las comparaciones siguientes; este número de generaciones requiere, en la mayoría de los casos, más de $120000n$ milisegundos.

4.3.2. Análisis del rendimiento de cada componente del algoritmo AGCC-ADE

En esta sección investigamos la efectividad de los componentes del algoritmo propuesto, primero comparamos cinco mecanismos de colaboración presentados por [76], y luego investigamos la contribución de la estrategia de perturbación local múltiple propuesta.

Efectividad del mecanismo de colaboración

Muchos de los algoritmos coevolutivos cooperativos [64, 76–82] del estado del arte que han abordado el problema de programación han utilizado diferentes mecanismos de cooperación para generar una solución completa. En este trabajo hemos implementado la *selección aleatoria de colaboradores* [79] como mecanismo de cooperación.

Para investigar el impacto que tiene este mecanismo de cooperación (*selección aleatoria de colaboradores* [79]) en el algoritmo propuesto, el cual denotamos AC, fue comparado con otros cuatro mecanismos de colaboración: *selección del mejor colaborador* (MC), *selección del peor colaborador* (PC), *selección del colaborador relacionado* (RC) y *selección del colaborador basado en el vecindario* (VC). Estos mecanismos de colaboración funcionan de la siguiente forma:

- AC selecciona aleatoriamente individuos de la otra población para formar la solución completa [79].
- En MC, cada individuo de la población colabora con el mejor individuo de la otra población para formar la solución completa del problema [83].
- PC, al contrario de MC, cada individuo de la población colabora con el peor individuo de la otra población [79].

- RC combina los individuos relacionados en ambas poblaciones para formar una solución completa [79]. Es decir, el primer individuo de la población P_1 colabora con el primer individuo de la población P_2 y así sucesivamente.
- VC construye la solución completa de un individuo de la población con sus individuos vecinos de la otra población [80, 81].

Para más detalles sobre cada uno de los mecanismos de colaboración, véase [76].

Para evaluar los cinco mecanismos de colaboración descritos anteriormente, ejecutamos cada uno de ellos 20 veces de forma independiente para cada caso de prueba. Los resultados se presentan en la Tabla 4.10. Además, se realizan pruebas no paramétricas (pruebas de Wilcoxon) con un intervalo de confianza del 95 % (o un nivel de significancia del 5 %) para cada par de mecanismos de colaboración. Si el *valor-p* es inferior a 0.05, significa que la diferencia entre los mecanismos de colaboración comparados es significativa.

La Tabla 4.10 muestra que, aunque los valores-p indican que, a un nivel de confianza del 95 %, no hay diferencias significativas entre el AC, RC, MC y VC, el AGCC-ADE con AC obtuvo los valores más bajos tanto en PRD como en APRD en la mayoría de los casos en comparación con sus variantes. Con respecto al AGCC-ADE con PC, la Tabla 4.10 muestra que hay una diferencia significativa en todos los casos de prueba con respecto tanto a PRD como a APRD, lo que significa que el rendimiento del AGCC-ADE con AC es estadísticamente mejor que el AGCC-ADE con PC.

Tabla 4.10: Comparación de los distintos mecanismos de colaboración con respecto al PRD y al APRD.

Caso de prueba			AC		RC		AC vs RC	MC		AC vs MC	VC		AC vs VC	PC		AC vs PC
Caso	(n,m)	BKS	PRD	APRD	PRD	APRD	*p*-value	PRD	APRD	*p*-value	PRD	APRD	*p*-value	PRD	APRD	*p*-value
Orb05	(10,10)	1365	0.15	0.30	0.15	0.46	0.6108	0.15	1.22	0.0654	0.15	0.86	0.5928	5.64	8.64	0.0000
la09	(15,5)	1358	0.52	0.94	0.52	1.35	0.6407	0.52	1.19	0.1159	0.52	1.57	0.6984	12.37	20.46	0.0000
la11	(20,5)	1621	0.68	1.75	1.54	2.57	0.3571	0.86	2.26	0.4649	0.80	2.89	0.1556	7.65	21.25	0.0000
la12	(20,5)	1425	1.54	3.45	2.67	4.76	0.1133	2.25	4.97	0.0986	1.54	5.02	0.0581	17.47	22.00	0.0000
la13	(20,5)	1580	1.14	2.77	2.03	2.48	0.4008	3.48	3.40	0.4484	2.03	2.82	1.0000	11.96	18.61	0.0000
la14	(20,5)	1610	1.06	3.01	2.80	2.86	0.6261	4.16	2.70	0.1894	2.80	3.20	0.5517	17.08	24.26	0.0000
la15	(20,5)	1679	0.12	2.45	1.31	3.09	0.2224	0.12	3.08	0.1074	1.31	2.09	0.5500	14.59	17.99	0.0000
la28	(20,10)	2552	2.12	3.66	3.80	4.28	0.1573	2.86	4.63	0.0514	2.78	5.18	0.0903	17.99	27.45	0.0000
la32	(30,10)	3835	0.78	3.75	2.63	3.63	0.6039	2.56	3.25	0.0935	0.78	2.29	0.0791	20.37	25.60	0.0000
la33	(30,10)	3413	1.49	5.55	2.23	5.10	0.3359	5.54	5.42	0.1595	3.52	5.26	0.0914	23.03	27.40	0.0000
la34	(30,10)	3551	1.94	5.49	1.94	5.06	0.3301	2.08	4.70	0.0699	2.11	4.34	0.0725	22.89	25.78	0.0000
la37	(15,15)	2831	3.74	4.32	3.74	4.91	0.1742	3.74	5.07	0.1226	3.74	5.43	0.0742	23.03	30.82	0.0000
swv04	(20,10)	2462	1.91	2.92	2.15	3.25	0.7071	2.15	3.90	0.1081	1.91	3.54	0.5063	6.05	11.72	0.0000
swv06	(20,15)	3278	0.40	2.76	2.38	2.91	0.5986	0.40	4.17	0.0865	2.56	3.94	0.0607	11.50	15.35	0.0000

Efectividad de la estrategia de perturbación local múltiple

Para evaluar la efectividad de la Estrategia de Perturbación Local Múltiple (EPLM) propuesta, consideramos el AGCC-ADE con esta estrategia ($AGCC_0$) y lo comparamos con cada una de las perturbaciones locales que componen la EPLM, las cuales se denotan como sigue AGCC-ADE con *partial pair-wise exhange* ($AGCC_1$), AGCC-ADE con *simple 1-insertion* ($AGCC_2$), AGCC-ADE con *intercambio adyacente y no adyacente* ($AGCC_3$), AGCC-ADE con *inserción adyacente y no adyacente* ($AGCC_4$), y AGCC-ADE con *procedimiento de cambio de bits* ($AGCC_5$).

La Tabla 4.11 muestra los valores del PRD y del APRD obtenidos en 20 ejecuciones independientes para 14 casos de prueba, donde se observa que $AGCC_0$ supera al resto de perturbaciones locales en todos los casos de prueba tanto para el PRD como para el APRD.

En cuanto a la prueba no paramétrica, se observa que $AGCC_0$ también es significativamente mejor que el resto de estrategias en los 14 casos. En general, se puede concluir que el EPLM es un componente clave en el enfoque propuesto y contribuye significativamente al rendimiento de AGCC-ADE. Además, se observa que cada componente del EPLM por separado no ayuda a mejorar los resultados, mientras que el trabajo conjunto de todos ellos sí lo hace.

Tabla 4.11: Comparación de las estrategias de perturbación local aislada contra la perturbación múltiple, en relación con el PRD y el APRD.

Caso de prueba			$AGCC_0$			$AGCC_0$ vs $AGCC_1$			$AGCC_0$ vs $AGCC_2$			$AGCC_0$ vs $AGCC_3$			$AGCC_0$ vs $AGCC_4$			$AGCC_0$ vs $AGCC_5$
Caso	(n,m)	BKS	PRD	APRD	PRD	APRD	valor-p	PRD	APRD	valor-p	PRD	APRD	valor-p	PRD	APRD	valor-p	PRD	APRD	valor-p
Orb05	(10,10)	1365	0.15	0.30	0.44	3.52	0.0000	0.44	2.92	0.0001	0.44	3.33	0.0000	0.44	2.79	0.0000	0.44	3.12	0.0000
la09	(15,5)	1358	0.52	0.94	0.74	5.91	0.0011	3.76	6.57	0.0000	1.47	5.06	0.0003	3.02	6.17	0.0000	0.74	4.51	0.0181
la11	(20,5)	1621	0.68	1.75	4.57	6.84	0.0000	4.38	7.61	0.0000	2.96	7.04	0.0000	2.28	6.67	0.0001	2.28	6.87	0.0003
la12	(20,5)	1425	1.54	3.45	5.89	9.79	0.0000	2.25	10.18	0.0000	7.30	10.28	0.0000	4.91	9.57	0.0000	2.25	8.60	0.0003
la13	(20,5)	1580	1.14	2.77	5.19	9.31	0.0000	4.68	9.97	0.0000	4.24	8.29	0.0000	2.91	8.59	0.0002	3.29	8.53	0.0001
la14	(20,5)	1610	1.06	3.01	3.04	10.03	0.0000	7.52	11.60	0.0000	4.72	10.00	0.0001	5.22	10.03	0.0000	3.04	10.09	0.0003
la15	(20,5)	1679	0.12	2.45	3.16	8.16	0.0000	4.71	7.40	0.0000	3.81	8.02	0.0000	2.74	7.17	0.0000	6.13	6.78	0.0000
la28	(20,10)	2552	2.12	3.66	6.23	10.90	0.0001	5.68	11.29	0.0000	7.56	11.89	0.0000	4.98	12.02	0.0000	6.66	10.04	0.0002
la32	(30,10)	3835	0.78	3.75	7.54	11.34	0.0000	8.58	12.75	0.0000	4.56	10.53	0.0001	3.39	10.03	0.0001	6.18	11.39	0.0000
la33	(30,10)	3413	1.49	5.55	6.42	12.38	0.0000	6.50	13.68	0.0000	4.22	11.14	0.0000	7.68	11.83	0.0000	6.74	11.43	0.0000
la34	(30,10)	3551	1.94	5.49	7.63	11.01	0.0000	8.22	12.30	0.0000	3.41	10.35	0.0001	5.29	10.98	0.0000	5.04	10.67	0.0000
la37	(15,15)	2831	3.74	4.32	6.53	11.07	0.0000	8.90	12.90	0.0000	6.53	10.45	0.0000	6.53	10.81	0.0000	6.53	8.62	0.0021
swv04	(20,10)	2462	1.91	2.92	3.66	7.36	0.0000	3.05	7.66	0.0000	2.23	7.07	0.0000	3.05	6.88	0.0000	3.53	7.17	0.0000
swv06	(20,15)	3278	0.40	2.76	4.85	8.71	0.0001	5.58	9.97	0.0000	4.45	7.99	0.0000	4.94	8.29	0.0003	5.74	8.39	0.0000

Para demostrar que todas las variantes utilizan los mismos recursos computacionales, las Tablas 4.12 y 4.13 presentan el promedio de los tiempos de cálculo en todos los casos de prueba de todos los algoritmos comparados. Se puede observar que todos los algoritmos emplean tiempos de cálculo similares. Además, los valores-p indican que, con un nivel de confianza de 0.95, no hay diferencias significativas entre los algoritmos comparados con respecto al tiempo de cálculo.

Tabla 4.12: Promedio de tiempo computacional y prueba de Wilcoxon para los mecanismos de colaboración en 20 ejecuciones.

Casos de prueba		AC	RC	AC vs RC	MC	AC vs MC	VC	AC vs VC	PC	AC vs PC
Caso	(n,m)	Tiempo	Tiempo	valor-p	Tiempo	valor-p	Tiempo	valor-p	Tiempo	valor-p
Orb05	(10,10)	41.72	41.60	0.6603	41.55	0.3986	40.93	0.4052	41.57	0.7345
la09	(15,5)	120.74	120.14	0.4782	120.62	0.4364	119.86	0.4572	120.98	0.7499
la11	(20,5)	230.89	229.74	0.3022	230.43	0.5733	229.66	0.7880	228.35	0.4273
la12	(20,5)	224.55	225.22	0.4568	224.77	0.3420	225.67	0.7110	225.00	0.4265
la13	(20,5)	249.07	248.57	0.8336	249.82	0.5581	248.82	0.5103	249.57	0.7484
la14	(20,5)	240.40	239.20	0.7780	241.60	0.6331	240.64	0.5463	238.88	0.3403
la15	(20,5)	252.40	253.41	0.3804	252.90	0.3580	252.63	0.2663	252.90	0.5418
la28	(20,10)	736.94	737.68	0.5137	738.41	0.2527	736.20	0.5345	738.41	0.2157
la32	(30,10)	2381.36	2380.65	0.5431	2380.17	0.5975	2379.89	0.2361	2380.44	0.4244
la33	(30,10)	2458.40	2457.66	0.2367	2460.86	0.5315	2459.07	0.3610	2459.32	0.4480
la34	(30,10)	2490.69	2489.94	0.5617	2495.67	0.5703	2488.20	0.4490	2488.74	0.6770
la37	(15,15)	372.63	372.93	0.7630	372.26	0.3526	373.49	0.2807	373.38	0.4542
swv04	(20,10)	649.44	648.79	0.2573	647.50	0.4806	647.79	0.5059	650.74	0.4343
swv06	(20,15)	1079.42	1079.10	0.3276	1078.34	0.5269	1080.50	0.5293	1078.58	0.5336

Tabla 4.13: Promedio de tiempo computacional y prueba de Wilcoxon para las estrategias de perturbación local en 20 ejecuciones.

Casos de prueba				$AGCC_0$ vs		$AGCC_0$ vs		$AGCC_0$ vs		$AGCC_0$ vs		$AGCC_0$ vs
Caso	(n,m)	$AGCC_0$ Time	$AGCC_1$ Time	$AGCC_1$ p-value	$AGCC_2$ Time	$AGCC_2$ p-value	$AGCC_3$ Time	$AGCC_3$ p-value	$AGCC_4$ Time	$AGCC_4$ p-value	$AGCC_5$ Time	$AGCC_5$ p-value
Orb05	(10,10)	41.72	41.82	0.1403	42.71	0.1046	42.45	0.2134	41.57	0.7868	40.94	0.2733
la09	(15,5)	120.74	120.52	0.2392	119.83	0.2183	122.10	0.2287	121.09	0.6949	120.42	0.0742
la11	(20,5)	230.89	229.94	0.0583	230.23	0.7868	230.65	0.3942	231.26	0.5250	229.94	0.2085
la12	(20,5)	224.55	224.05	0.5428	223.98	0.8711	223.71	0.7557	224.92	0.9246	224.78	0.3720
la13	(20,5)	249.07	249.18	0.4170	250.59	0.5792	247.79	0.6553	248.47	0.3941	248.05	0.0935
la14	(20,5)	240.40	241.60	0.9892	240.51	0.6167	241.74	0.2733	240.71	0.5792	240.46	0.6073
la15	(20,5)	252.40	252.14	0.4903	251.07	0.5791	254.05	0.1332	249.33	0.2853	253.45	0.2853
la28	(20,10)	736.94	735.78	0.4570	733.97	0.1264	734.76	0.2674	734.83	0.1136	735.27	0.2393
la32	(30,10)	2381.36	2382.87	0.1717	2381.95	0.4989	2380.79	0.5181	2380.44	0.2235	2381.16	0.8604
la33	(30,10)	2458.40	2459.46	0.2184	2458.23	0.7659	2457.17	0.1806	2458.89	0.6359	2457.96	0.4249
la34	(30,10)	2490.69	2489.69	0.0810	2491.18	0.2853	2490.69	0.9246	2489.58	0.0565	2490.44	0.5789
la37	(15,15)	372.63	372.51	0.8817	370.99	0.0764	371.27	0.1404	372.88	0.8392	371.27	0.1404
swv04	(20,10)	649.44	648.15	0.2287	647.50	0.1404	646.21	0.0531	648.15	0.2287	650.09	0.3369
swv06	(20,15)	1079.42	1076.19	0.0639	1077.27	0.1636	1078.34	0.3648	1076.41	0.0810	1077.81	0.2616

4.3.3. Comparación de AGCC-ADE con y sin estrategia de perturbación local múltiple (EPLM)

Para evaluar el rendimiento del AGCC-ADE, cuando se utiliza con y sin EPLM, el algoritmo se ejecutó en 20 casos de prueba seleccionados aleatoriamente (10 tamaño pequeño y 10 tamaño grande). Primeramente AGCC-ADE sin EPLM, luego la EPLM aislada y finalmente, el AGCC-ADE con EPLM. Los experimentos entre estas variantes se ejecutan con el mismo equipo de cómputo y con el mismo tiempo de ejecución.

Adicionalmente y con el fin de probar si existen diferencias estadísticas significativas entre AGCC-ADE y sus variantes, se ha utilizado la prueba de significancia estadística no paramétrica de Wilcoxon con un nivel de confianza $\alpha = 0.05$ con respecto a PRD y APRD.

Los resultados se presentan en la Tabla 4.14, el AGCC-ADE con EPLM obtiene el valor del BKS en 15 casos de prueba, mientras que el AGCC-ADE sin EPLM y con EPLM obtienen el BKS en 5 y 3 casos, respectivamente. En cuanto al promedio del PRD, el AGCC-ADE con EPLM alcanza un valor del 0.33, mientras que el AGCC-ADE sin EPLM y solo la EPLM obtienen el 3.71 y el 5.43, respectivamente. Además, el promedio de APRD de AGCC-ADE con EPLM es del 1.52, mientras que AGCC-ADE sin EPLM y EPLM aislada obtienen un 8.06 y un 8.87, respectivamente. Los resultados mostraron que el valor del BKS es alcanzado por el AGCC-ADE con EPLM en casi todos los casos (15 de 20). Por otro lado, para casos de tamaño $n >10$ los valores PRD y APRD son mayores tanto en AGCC-ADE sin EPLM como en solo EPLM.

En cuanto a la prueba no paramétrica, la Tabla 4.14 muestra que hay una diferencia

significativa en todos los casos de prueba con respecto tanto a PRD como a APRD, lo que significa que el rendimiento del AGCC-ADE con EPLM es estadísticamente mejor que el de ambos AGCC-ADE sin EPLM y EPLM aislada. Esto demuestra claramente que hay un efecto sinérgico cuando se utilizan juntos estos enfoques (AGCC-ADE y EPLM).

Con base en los resultados anteriores, el AGCC-ADE se ejecutó con EPLM en el resto de los casos de prueba y los resultados obtenidos se muestran y analizan en las siguientes secciones.

Tabla 4.14: Comparación del AGCC-ADE sin/con EPLM

Caso de prueba			AGCC-ADE		AGCC-ADE sin EPLM		AGCC-ADE vs AGCC-ADE sin EPLM	EPLM		AGCC-ADE vs MLS
Caso	(n,m)	BKS	PRD	APRD	PRD	APRD	valor-p	PRD	APRD	valor-p
La01	(10,5)	971	0.00	0.91	0.00	3.86	0.0000	0.41	2.48	0.0000
La04	(10,5)	887	0.00	0.06	0.11	2.54	0.0000	0.11	1.60	0.0001
Orb01	(10,10)	1615	0.00	0.00	0.99	3.12	0.0000	1.71	2.65	0.0000
Orb02	(10,10)	1485	0.00	0.89	0.00	3.45	0.0000	0.27	2.72	0.0000
Orb03	(10,10)	1599	0.00	0.00	0.00	1.76	0.0000	0.00	1.11	0.0000
Orb05	(10,10)	1365	0.15	0.30	0.15	3.30	0.0000	1.47	2.25	0.0000
Orb08	(10,10)	1319	0.00	0.00	0.00	1.07	0.0000	0.00	0.82	0.0000
Orb09	(10,10)	1445	0.00	0.00	0.00	3.81	0.0000	0.00	1.59	0.0000
La17	(10,10)	1371	0.00	0.95	0.95	3.04	0.0000	1.55	2.30	0.0000
La20	(10,10)	1526	0.00	0.39	0.98	3.53	0.0000	2.10	4.44	0.0000
La06	(15,5)	1248	0.00	1.20	6.13	11.03	0.0000	5.93	12.76	0.0000
La07	(15,5)	1172	0.00	2.47	5.03	10.47	0.0000	8.11	12.97	0.0000
La13	(20,5)	1580	1.14	2.77	8.67	14.04	0.0000	8.80	16.25	0.0000
La21	(15,10)	2030	0.89	2.32	4.98	13.30	0.0000	7.93	12.47	0.0000
La25	(15,10)	1906	0.00	2.23	3.74	10.35	0.0000	4.14	11.18	0.0000
La29	(20,10)	2300	0.00	5.61	10.35	16.85	0.0000	17.09	21.83	0.0000
La32	(30,10)	3835	0.78	3.75	11.06	15.17	0.0000	18.19	22.91	0.0000
La38	(15,15)	2525	0.00	2.49	8.28	14.74	0.0000	9.23	14.76	0.0000
swv02	(20,10)	2417	0.00	1.20	4.30	11.57	0.0000	9.97	14.35	0.0000
swv06	(20,15)	3278	0.40	2.76	8.48	14.24	0.0000	11.53	15.96	0.0000

4.3.4. Comparación de AGCC-ADE con enfoques de la literatura

En esta sección se comparan los resultados obtenidos por AGCC-ADE con los generados por los algoritmos CLLM, MCLM, HABC, MSA-BST, GASA, TS y HTS. Las Tablas 4.15, 4.16, 4.17 y 4.18 informan de los resultados en 21 casos de prueba de tamaño pequeño, 40 de tamaño grande-A, 9 de tamaño grande-B1 y 20 de tamaño grande-B2, respectivamente.

Los BKSs de 21 casos de prueba de tamaño pequeño mostrados en la Tabla 4.15 son los valores óptimos obtenidos por [47], los BKSs de los 40 casos de tamaño grande-

A presentados en la Tabla 4.16 no se sabe que sean valores óptimos, sino los mejores valores obtenidos por el CLLM, MCLM, HABC o MSA-BST, los BKSs de los 9 casos de tamaño grande-B1 y 20 casos de tamaño grande-B2 presentados en las Tablas 4.17 y 4.18, respectivamente, son valores obtenidos por Ying y Lin [25].

Para cada caso de prueba, AGCC-ADE fue ejecutado 20 veces, además, se realizan pruebas no paramétricas con un intervalo de confianza del 95 %. El resultado de la prueba de Wilcoxon se indica en la última fila de las tablas, de la siguiente manera, si existe una diferencia estadística entre el AGCC-ADE y cualquiera de las variantes de comparación, el valor correspondiente al promedio de PRD o al promedio de APRD se indica con " * ", de lo contrario no se indica.

Comparación con casos de prueba de tamaño pequeño

En los casos de prueba de tamaño pequeño, la Tabla 4.15 muestra que AGCC-ADE obtiene el valor del BKS en 20 (de 21) casos de prueba, mientras que CLLM, MCLM, HABC y MSA-BST obtienen el BKS en 16, 20, 16 y 21 casos, respectivamente.

En cuanto al promedio del PRD, AGCC-ADE obtiene un valor de 0.01 %, mientras que CLLM, MCLM, HABC y MSA-BST obtienen 0.01 %, 0.41 %, 0.23 % y 0.00 %, respectivamente. Además, el promedio del APRD de AGCC-ADE es del 0.42 %, mientras que CLLM, MCLM, HABC y MSA-BST obtienen el 0.27 %, 0.47 %, 0.51 % y 0.08 %, respectivamente.

Los resultados muestran que el algoritmo AGCC-ADE encontró 20 de 21 soluciones óptimas para los casos de pruebas de tamaño pequeño. Además, el AGCC-ADE obtiene valores más bajos en el promedio de PRD y APRD que los algoritmos MCLM y HABC. Sin embargo, el AGCC-ADE requiere más tiempo de cómputo que los algoritmos comparados, excepto el CLLM, para resolver los casos de prueba de tamaño pequeño. Para estos casos de prueba de tamaño pequeño, los promedios tanto de PRD como de APRD son menores que 0.52 para todos los algoritmos comparados, lo que implica que los casos de prueba de tamaño pequeño no son difíciles de resolver.

Las pruebas de significancia estadística no paramétricas para los casos de prueba de tamaño pequeño (Tabla 4.15) indican que, no hay diferencias significativas entre la AGCC-ADE y los demás enfoques, lo que evidencia que el rendimiento de la AGCC-ADE es estadísticamente equivalente al de los demás algoritmos en cuanto al PRD. Por otra parte, con respecto a la APRD, la Tabla 4.15 indica que hay una diferencia significativa entre AGCC-ADE y MSA-BST, lo que significa que el rendimiento de MSA-BST es estadísticamente mejor que el de AGCC-ADE para los casos de prueba de tamaño pequeño.

Tabla 4.15: Resultados de PRD, APRD y T en casos de tamaño pequeño

Casos de prueba			CLLM			MCLM			HABC			MSA-BST			AGCC-ADE		
Caso	(n,m)	BKS	PRD	APRD	T(seg)	PRD	APRD	T(seg)	PRD	APRD	T(seg)	PRD	APRD	T(seg)	PRD	APRD	T(seg)
Ft06	(6,6)	73	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.11	0.00	0.00	0.54	0.00	0.00	0.67
La01	(10,5)	971	0.00	0.10	9.83	0.00	0.02	3.19	0.41	0.41	0.55	0.00	0.00	2.00	0.00	0.91	12.78
La02	(10,5)	937	0.00	0.21	23.17	0.00	0.00	5.34	2.56	2.56	0.99	0.00	0.00	1.82	0.00	0.49	15.27
La03	(10,5)	820	0.00	0.00	8.43	0.00	0.00	2.18	0.00	0.00	0.49	0.00	0.00	1.80	0.00	1.21	15.95
La04	(10,5)	887	0.00	0.00	23.87	0.00	0.00	4.39	0.00	0.00	0.61	0.00	0.00	1.71	0.00	0.06	14.50
La05	(10,5)	777	0.00	0.50	5.62	0.51	0.90	2.74	0.51	0.51	0.80	0.00	0.00	1.56	0.00	0.40	10.78
Ft10	(10,10)	1607	0.00	0.00	64.60	0.00	0.00	5.51	0.00	0.00	6.92	0.00	0.00	3.46	0.00	0.00	36.55
Orb01	(10,10)	1615	0.00	0.00	50.55	0.00	0.00	4.67	0.00	0.00	5.62	0.00	0.00	3.37	0.00	0.00	26.72
Orb02	(10,10)	1485	0.00	1.89	33.70	2.16	2.16	4.70	0.00	0.00	4.82	0.00	0.22	3.78	0.00	0.89	33.34
Orb03	(10,10)	1599	0.00	0.00	49.85	0.00	0.00	9.65	0.00	0.00	7.07	0.00	0.00	3.44	0.00	0.00	11.36
Orb04	(10,10)	1653	0.00	0.42	28.09	0.00	0.12	5.51	0.00	0.00	3.71	0.00	0.00	3.88	0.00	0.00	20.28
Orb05	(10,10)	1365	0.15	0.15	9.83	0.00	0.00	5.97	0.37	0.37	3.35	0.00	0.14	3.75	0.15	0.30	31.34
Orb06	(10,10)	1555	0.00	0.00	30.89	0.00	0.00	2.49	0.00	0.00	4.04	0.00	0.00	3.62	0.00	0.13	25.95
Orb08	(10,10)	1319	0.00	0.00	42.13	0.00	0.00	4.49	0.00	0.00	6.31	0.00	0.00	3.20	0.00	0.00	20.71
Orb09	(10,10)	1445	0.00	0.83	13.34	0.00	0.00	2.98	0.00	0.28	2.88	0.00	0.00	3.59	0.00	0.00	33.58
Orb10	(10,10)	1557	0.00	0.00	48.45	0.00	0.00	8.32	0.00	0.00	3.87	0.00	0.09	4.36	0.00	0.79	34.12
La16	(10,10)	1575	0.00	0.00	28.79	1.84	1.84	3.97	0.00	0.00	3.81	0.00	0.00	3.39	0.00	0.00	31.88
La17	(10,10)	1371	0.00	0.22	37.21	0.00	0.12	8.32	0.95	0.95	3.24	0.00	0.00	3.56	0.00	0.95	32.31
La18	(10,10)	1417	0.00	1.13	35.85	2.82	2.82	4.32	0.00	5.22	5.01	0.00	0.79	3.97	0.00	1.74	27.66
La19	(10,10)	1482	0.00	0.14	11.23	0.00	0.61	3.51	0.00	0.43	2.42	0.00	0.48	3.78	0.00	0.55	38.47
La20	(10,10)	1526	0.00	0.13	4.21	1.31	1.31	3.79	0.00	0.00	2.11	0.00	0.00	3.45	0.00	0.39	19.15
Promedio			0.01	0.27	26.65	0.41	0.47	4.57	0.23	0.51	3.27	0.00	0.08*	3.13	0.01	0.42	23.49

Comparación en casos de prueba de tamaño grande-A

Los resultados de los 40 casos de tamaño grande-A que se indican en la Tabla 4.16 muestran que CLLM, MCLM, HABC y MSA-BST obtuvieron 8, 11, 18 y 34 BKS, respectivamente, mientras que el AGCC-ADE obtuvo el valor del BKS en 16 casos. Uno de los 16 valores del BKS no se ha encontrado anteriormente en la literatura, este nuevo BKS se logra en La39 con un C_{max}=2660.

Con respecto a todas los casos de prueba de tamaño grande-A, el promedio de PRD de AGCC-ADE es de 0.66 %, mientras que los de CLLM, MCLM, HABC y MSA-BST son de 2.74 %, 1.33 %, 0.81 % y 0.08 %, respectivamente. Por otro lado, el resultado promedio de APRD de AGCC-ADE fue de 2.87 %, mientras que los de CLLM, MCLM, HABC y MSA-BST fueron de 5.74 %, 2.90 %, 1.95 % y 1.76 %, respectivamente. El AGCC-ADE obtiene valores más bajos en el promedio de PRD que CLLM, MCLM y HABC. Por el contrario, el AGCC-ADE obtiene valores más altos en el promedio de PRD y APRD que el MSA-BST. Los resultados revelan que el AGCC-ADE muestra un buen rendimiento en la mayoría de los casos de tamaño grande-A. Sin embargo, para el grupo La26-35, el AGCC-ADE obtiene los valores más altos de APRD.

Las pruebas de significancia estadística no paramétrica para los casos de prueba de tamaño grande-A (Tabla 4.16) muestran que hay diferencias significativas, tanto para PRD como para APRD, entre AGCC-ADE y CLLM y también con MSA-BST, lo que significa que AGCC-ADE muestra una mejora sobre el CLLM, por el contrario, AGCC-ADE muestra un peor rendimiento que MSA-BST tanto en PRD como en APRD. Por otra parte, no hay diferencias significativas entre AGCC-ADE con MCLM y con HABC, lo que implica que, el rendimiento de AGCC-ADE es estadísticamente equivalente a MCLM y HABC en lo que respecta a PRD para los casos de tamaño grande-A.

Tabla 4.16: Resultados de PRD, APRD y T en casos de tamaño grande-A

Casos de prueba			CLLM			MCLM			HABC			MSA-BST			AGCC-ADE			
Caso	(n, m)	BKS	PRD	APRD	T(seg)	PRD	APRD	T(seg)	PRD	APRD	T(seg)	PRD	APRD	T(seg)	Best	PRD	APRD	T(seg)
La06	(15, 5)	1248	0.00	2.80	185	0.00	0.75	63	0.00	0.00	28	0.00	0.00	49	1248	0.00	1.20	77
La07	(15, 5)	1172	0.00	2.47	153	0.51	2.46	58	0.00	0.92	63	0.00	0.00	47	1172	0.00	2.47	78
La08	(15, 5)	1244	0.00	2.25	176	0.00	0.47	56	0.00	0.15	60	0.00	1.03	65	1244	0.00	1.85	79
La09	(15, 5)	1358	0.00	1.47	163	0.52	0.73	74	0.29	0.67	24	0.00	0.46	51	1365	0.52	0.94	91
La10	(15, 5)	1287	0.00	0.70	141	0.00	0.04	48	0.54	0.70	46	0.00	0.22	57	1287	0.00	1.86	89
La11	(20, 5)	1621	3.08	5.86	314	0.86	2.49	308	0.37	1.05	182	0.00	1.53	191	1632	0.68	1.75	173
La12	(20, 5)	1425	1.89	5.68	350	0.28	2.48	416	0.63	1.89	119	0.00	2.26	184	1457	1.54	3.45	169
La13	(20, 5)	1580	2.78	5.13	449	1.58	2.63	213	0.00	1.27	156	1.14	2.74	199	1598	1.14	2.77	187
La14	(20, 5)	1610	5.03	6.89	326	2.36	3.81	220	1.86	2.68	110	0.00	1.67	167	1627	1.06	3.01	181
La15	(20, 5)	1679	0.89	4.17	340	0.36	1.99	298	0.00	0.80	169	0.36	1.20	206	1689	0.12	2.45	190
La21	(15, 10)	2030	0.89	3.65	215	0.89	1.00	55	0.64	1.16	50	0.00	0.52	110	2048	0.89	2.32	193
La22	(15, 10)	1852	1.89	3.24	249	2.70	2.90	100	0.00	0.75	64	0.00	1.11	111	1863	0.59	3.02	191
La23	(15, 10)	2021	0.54	3.81	216	0.05	2.02	35	0.54	1.26	85	0.00	1.43	129	2032	0.54	2.28	174
La24	(15, 10)	1972	2.18	3.85	296	2.18	2.97	69	1.12	2.16	69	0.00	1.89	110	1991	0.96	2.38	185
La25	(15, 10)	1906	0.58	3.41	209	1.26	2.66	50	0.00	0.00	65	0.00	0.49	120	1906	0.00	2.62	192
La26	(20, 10)	2506	1.88	8.02	570	1.04	3.82	245	0.00	2.18	157	0.00	1.60	315	2506	0.00	2.23	482
La27	(20, 10)	2666	3.04	6.45	586	1.84	3.33	272	0.30	0.34	109	0.00	0.49	289	2682	0.34	3.46	508
La28	(20, 10)	2552	2.82	7.84	569	0.31	4.64	220	0.31	3.46	139	0.00	3.03	332	2621	2.12	3.66	554
La29	(20, 10)	2300	8.22	10.39	546	2.91	5.53	312	3.87	5.30	373	0.00	3.42	332	2300	0.00	5.61	559
La30	(20, 10)	2452	8.69	11.87	577	3.75	7.05	264	0.00	5.43	174	0.00	5.32	338	2452	0.00	5.28	460
La31	(30, 10)	3369	11.16	15.29	1817	6.11	9.60	2176	6.62	8.27	1205	0.00	6.11	1316	3432	1.87	5.35	1797
La32	(30, 10)	3835	5.03	11.06	1894	0.00	5.62	2327	2.03	4.66	1117	0.63	4.00	1304	3951	0.78	3.75	1789
La33	(30, 10)	3413	9.84	12.57	1816	4.72	8.18	2109	3.40	6.14	1084	0.00	3.93	1352	3487	1.49	5.55	1847
La34	(30, 10)	3551	7.69	10.73	1934	3.75	6.74	2370	1.66	4.72	987	0.00	3.17	1276	3665	1.94	5.49	1871
La35	(30, 10)	3556	5.74	12.04	1836	3.99	7.08	2165	1.04	5.30	1262	0.00	3.42	1562	3569	0.37	5.18	1821
La36	(15, 15)	2685	0.00	3.46	376	1.90	4.98	133	0.00	0.29	54	0.00	0.50	166	2685	0.00	1.82	298
La37	(15, 15)	2831	4.63	6.64	355	4.63	4.74	65	3.78	5.65	129	0.00	4.65	179	2937	3.74	4.32	280
La38	(15, 15)	2525	3.64	5.98	349	0.00	1.85	64	0.00	2.52	219	0.00	2.31	159	2525	0.00	2.49	313
La39	(15, 15)	2697	0.00	2.93	392	1.19	2.43	81	0.22	0.88	103	0.00	0.37	160	2660	-1.37	0.85	370
La40	(15, 15)	2580	0.54	5.00	199	0.00	0.00	43	0.54	0.54	134	0.00	0.52	135	2594	0.54	0.97	260
Swv01	(20, 10)	2318	0.43	3.06	450	0.65	0.80	362	0.00	0.41	614	0.43	0.80	376	2318	0.00	1.86	461
Swv02	(20, 10)	2417	0.04	3.14	546	0.04	0.42	343	0.00	0.02	675	0.00	0.00	358	2417	0.00	1.20	469
Swv03	(20, 10)	2381	1.43	4.28	532	0.00	1.01	363	0.00	0.49	715	0.00	1.01	448	2393	0.50	3.07	459
Swv04	(20, 10)	2462	3.25	4.06	474	0.00	2.09	299	1.79	2.08	1214	0.04	1.98	391	2515	1.91	2.92	488
Swv05	(20, 10)	2333	0.00	6.94	500	0.00	0.00	200	0.00	0.00	376	0.00	1.72	371	2333	0.00	3.29	494
Swv06	(20, 15)	3278	2.99	5.46	798	0.40	2.09	525	0.40	0.80	622	0.00	0.91	562	3327	0.40	2.76	811
Swv07	(20, 15)	3188	2.60	4.17	826	1.63	0.97	410	0.00	0.00	582	0.00	0.28	564	3188	0.00	1.05	784
Swv08	(20, 15)	3423	3.13	6.16	807	0.00	0.85	290	0.00	1.29	808	0.00	2.09	577	3474	1.49	3.59	776
Swv09	(20, 15)	3246	1.88	3.57	739	0.74	1.59	315	0.00	1.00	1075	0.00	0.87	517	3307	1.88	2.68	895
Swv10	(20, 15)	3451	1.07	3.27	802	0.00	1.19	365	0.32	0.85	1001	0.41	1.55	474	3462	0.32	2.06	888
Promedio			2.74*	5.74*	602	1.33	2.90	460	0.81	1.95*	405	0.08*	1.76*	391		0.66	2.87	550

Comparación con casos de prueba de tamaño grande-B1

Los resultados de los 9 casos de tamaño grande-B1 se indican en la Tabla 4.17, los cuales se compararon con los resultados de PRD sólo de GASA, TS, HTS y MSA-BST porque los métodos GASA, TS, HTS no incluyen los resultados de APRD. La Tabla 4.17 muestra que el promedio del PRD obtenido por AGCC-ADE fue del 0.55 %, mientras que la de GASA, TS, HTS y MSA-BST fue del 13.38 %, 11.39 %, 1.35 % y 0.21 %, respectivamente. Asimismo, los resultados muestran que AGCC-ADE obtiene el valor BKS en 4 casos, mientras que HTS y MSA-BST obtuvieron 2 y 5 BKSs, respectivamente.

Las pruebas no paramétricas para los casos de prueba de tamaño grande-B1 (Tabla 4.17) muestran que existen diferencias significativas entre AGCC-ADE y GASA y TS, lo que expresa que el rendimiento de AGCC-ADE es estadísticamente mejor que GASA y TS para casos de tamaño grande B1. Por otro lado, no hay diferencias significativas entre AGCC-ADE con HTS y con MSA-BST, lo que manifiesta que el rendimiento de AGCC-ADE es estadísticamente equivalente al rendimiento de HTS y MSA-BST en términos de PRD para los casos de tamaño grande-B1. Aunque el rendimiento de AGCC-ADE es estadísticamente equivalente al de HTS y MSA-BST, AGCC-ADE fue capaz de encontrar una mejor solución, que no fue encontrada por HTS o MSA-BST, esta solución se obtuvo en Abz05 con C_{max}=2150.

Tabla 4.17: Resultados de PRD, APRD y T en casos de tamaño grande-B1

Casos de prueba			GASA	TS	HTS	MSA-BST	AGCC-ADE	
Caso	(n,m)	BKS	PRD	PRD	PRD	PRD	Best	PRD
yn1	(20,20)	2360	16.57	12.46	0.00	0.08	2407	1.99
yn2	(20,20)	2312	14.36	17.00	2.51	0.00	2323	0.48
yn3	(20,20)	2299	17.62	15.01	0.91	0.00	2299	0.00
yn4	(20,20)	2458	15.42	10.05	2.24	0.81	2500	1.71
abz5	(10,10)	2170	3.00	2.90	0.55	0.00	2150	-0.92
abz6	(10,10)	1718	2.33	2.33	2.44	0.00	1718	0.00
abz7	(20,15)	1563	15.23	16.44	1.86	0.06	1563	0.00
abz8	(20,15)	1616	13.99	12.31	1.61	0.93	1634	1.11
abz9	(20,15)	1562	21.96	14.02	0.00	0.00	1571	0.58
Promedio			13.38*	11.39*	1.35	0.21		0.55

Comparación con casos de prueba de tamaño grande-B2

En la Tabla 4.18 se presentan los resultados de 20 casos de prueba de tamaño grande-B2, los cuales se compararon sólo con los resultados de MSA-BST porque los otros métodos no incluyen estos casos de prueba. La Tabla 4.18 muestra que el pomedio del PRD obtenido por AGCC-ADE fue de 0.91 %, mientras que MSA-BST fue

de 0.33 %. Asimismo, el valor promedio del APRD para AGCC-ADE fue de 3.65 %, mientras que para MSA-BST fue de 2.34 %. Los resultados muestran que AGCC-ADE obtiene los valores más altos tanto en PRD como en APRD para los casos de tamaño $n = 20$.

Las pruebas estadísticas no paramétricas para casos de tamaño grande-B2 (Tabla 4.18) muestran que existen diferencias significativas entre AGCC-ADE y MSA-BST, lo que significa que el rendimiento de MSA-BST es estadísticamente mejor que el de AGCC-ADE tanto en PRD como en APRD para este tipo de casos.

Aunque el rendimiento de AGCC-ADE es peor que el de MSA-BST, AGCC-ADE fue capaz de encontrar dos mejores soluciones, que no son encontradas por MSA-BST, estos nuevos valores de BKSs se obtuvieron en Ta02 con C_{max}=2561, y Ta07 con C_{max}=2635, las BKSs anteriores para estos casos fueron 2646 y 2684, respectivamente.

Las soluciones de los nuevos valores obtenidos por el AGCC-ADE en relación a BKS son mostradas en la siguiente sección. Además, las mejores soluciones encontradas para cada caso de prueba se pueden consultar en https:// figshare.com/s/9904937d8e22b47 f4940.

Tabla 4.18: Resultados de PRD, APRD y T en casos de tamaño grande-B2

Casos de prueba			MSA-BST			AGCC-ADE			
Caso	(n,m)	BKS	PRD	APRD	T(seg)	Mejor	PRD	APRD	T(seg)
ta01	(15,15)	2674	0.00	0.61	257.8	2674	0.00	1.25	317.6
ta02	(15,15)	2646	0.64	0.90	221.7	2561	-3.21	1.17	372.2
ta03	(15,15)	2600	0.00	0.36	212.1	2613	0.50	2.10	353.6
ta04	(15,15)	2500	0.00	1.71	246.9	2501	0.04	1.84	378.0
ta05	(15,15)	2600	0.00	2.22	281.6	2643	1.65	3.58	344.4
ta06	(15,15)	2549	0.00	1.03	189.3	2549	0.00	3.08	370.4
ta07	(15,15)	2684	0.00	1.45	254.7	2635	-1.83	2.73	336.0
ta08	(15,15)	2634	0.00	1.99	216.8	2658	0.91	3.67	329.4
ta09	(15,15)	2637	0.00	0.06	184.0	2667	1.14	3.68	339.6
ta10	(15,15)	2614	0.57	1.81	212.2	2667	2.03	3.94	377.5
ta11	(20,15)	3041	0.00	3.46	647.1	3041	0.00	4.37	789.3
ta12	(20,15)	3350	0.00	3.04	626.3	3427	2.30	5.49	779.0
ta13	(20,15)	3028	0.00	3.87	596.4	3075	1.55	4.23	720.8
ta14	(20,15)	3126	0.00	2.85	626.9	3133	0.22	3.79	721.8
ta15	(20,15)	3064	0.88	2.50	663.7	3131	2.19	4.77	732.4
ta16	(20,15)	3288	2.62	4.01	611.3	3374	2.62	4.76	767.4
ta17	(20,15)	3247	1.82	5.97	738.1	3393	4.50	6.93	761.3
ta18	(20,15)	3240	0.00	2.25	671.1	3279	1.20	3.51	770.7
ta19	(20,15)	3099	0.00	4.04	626.5	3176	2.48	4.10	741.7
ta20	(20,15)	3254	0.00	2.66	639.8	3254	0.00	4.06	768.2
Promedio			0.33^*	2.34^*	436.22		0.91	3.65	553.56

4.3.5. Discusión

Basándose en el análisis experimental descrito anteriormente, se puede concluir que para los casos de tamaño pequeño, AGCC-ADE es mejor que MCLM y HABC en lo que respecta al promedio de PRD y al promedio de APRD. En comparación con CLLM, AGCC-ADE obtiene el mismo resultado en lo que respecta al promedio de PRD.

Para los casos de prueba de tamaño grande-A, los resultados muestran que AGCC-ADE es mejor que CLLM, MCLM y HABC con respecto al promedio de PRD. En comparación con MSA-BST, AGCC-ADE es peor en términos del promedio de PRD. En cuanto al promedio de APRD, AGCC-ADE es mejor que CLLM y MCLM y peor que HABC y MSA-BST.

Por otro lado, en los casos de prueba de tamaño grande-B1, se observa que AGCC-ADE es mejor que GASA y TS con respecto al promedio de PRD. En comparación con HTS y MSA-BST, AGCC-ADE tiene un desempeño equivalente en términos del

promedio de PRD. En relación a los casos de prueba de tamaño grande-B2, se puede evidenciar que el rendimiento de AGCC-ADE es inferior que el del MSA-BST.

En general, el algoritmo propuesto no es el más eficiente en cuanto a tiempo de ejecución, véanse las Tablas 4.15 a 4.18, sin embargo, su tiempo de cálculo está en el mismo orden de magnitud que el enfoque más eficaz MSA-BST.

Por el contrario, en lo que respecta a la calidad de la solución, el enfoque propuesto ocupa el tercer lugar en los casos de prueba de tamaño pequeño, (después del CLLM que es el quinto en casos de tamaño grande A), el tercero en casos de tamaño grande A (después del algoritmo HABC que es el quinto en casos de tamaño pequeño), y sólo es mejorado por el MSA-BST en casos de tamaño grande-B2. Es decir, MSA-BST es el ganador actual para este problema en la mayoría de los casos de prueba y nuestro enfoque lo sigue.

En cuanto a los componentes del algoritmo AGCC-ADE, especialmente el EPLM, está claro que el uso conjunto de todos ellos produce un efecto sinérgico mientras que cada uno por separado no produce una mejora significativa. El análisis de los cinco mecanismos de colaboración muestra que cuatro de ellos son indistintos entre sí y sólo el conocido como peor colaborador (PC) tiene un rendimiento significativamente inferior al de los demás mecanismos.

Los resultados son alentadores, ya que la flexibilidad del enfoque propuesto permite muchas vías de mejora e incluso en este primer intento el enfoque fue capaz de encontrar cuatro nuevas mejores soluciones para casos difíciles y logró un segundo puesto general. Lo que nuestro trabajo muestra es que la evolución de las estrategias de retraso es posible para el NWJSSP, esta es la primera prueba de concepto y la novedad de nuestro enfoque.

Una oprtunidad de mejora tiene que ver con la estrategia de empuje, recordemos que, aunque AGCC-ADE evoluciona la combinación de sólo dos reglas de empuje, no es una limitación del método. El método aún puede incorporar otras estrategias de retraso como la utilizada en el MSA-BST [25].

Por ello, los resultados obtenidos aquí son motivadores, es decir, todavía hay margen de mejora y nuestra propuesta es sólo el primer paso en la coevolución de las reglas de "timetabling" y las permutaciones de trabajos. Además, la eficiencia es otro aspecto que puede mejorarse, en los algoritmos coevolutivos la paralelización de la evaluación de la función objetivo suele ser un medio para lograr este objetivo.

4.3.6. Soluciones de los nuevos BKS encontrados por AGCC-ADE

La Figura 4.4 muestra la gráfica de Gantt del nuevo BKS encontrado para el caso de prueba La39 por AGCC-ADE, donde se observa que el record lo tenia CLLM (2009) con un valor de 2697, sin embargo, un nuevo valor del C_{max}= 2660 es obtenido por AGCC-ADE.

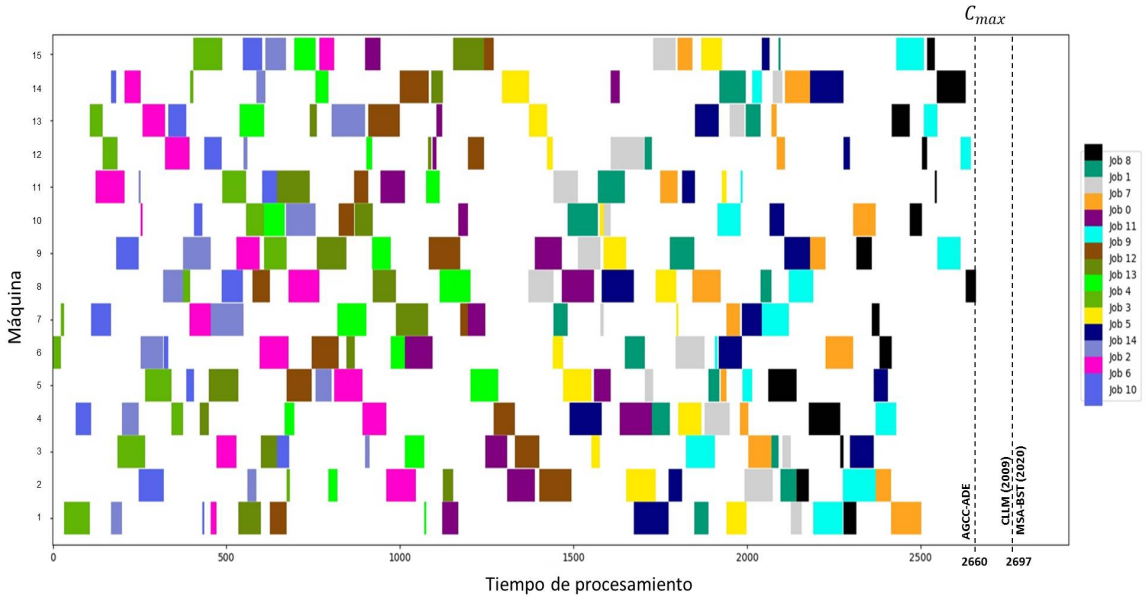

Figura 4.4: Nueva solución encontrada con AGCC-ADE para el caso de prueba La39 (n=15, m=15).

En la Figura 4.5 muestra una gráfica de Gantt para el caso de prueba Abz5, donde se observa que el AGCC-ADE logra mejorar el valor del C_{max}= 2170 establecido como la mejor solución por MSA-BST en el año 2020, el valor de C_{max} que obtiene AGCC-ADE es 2170.

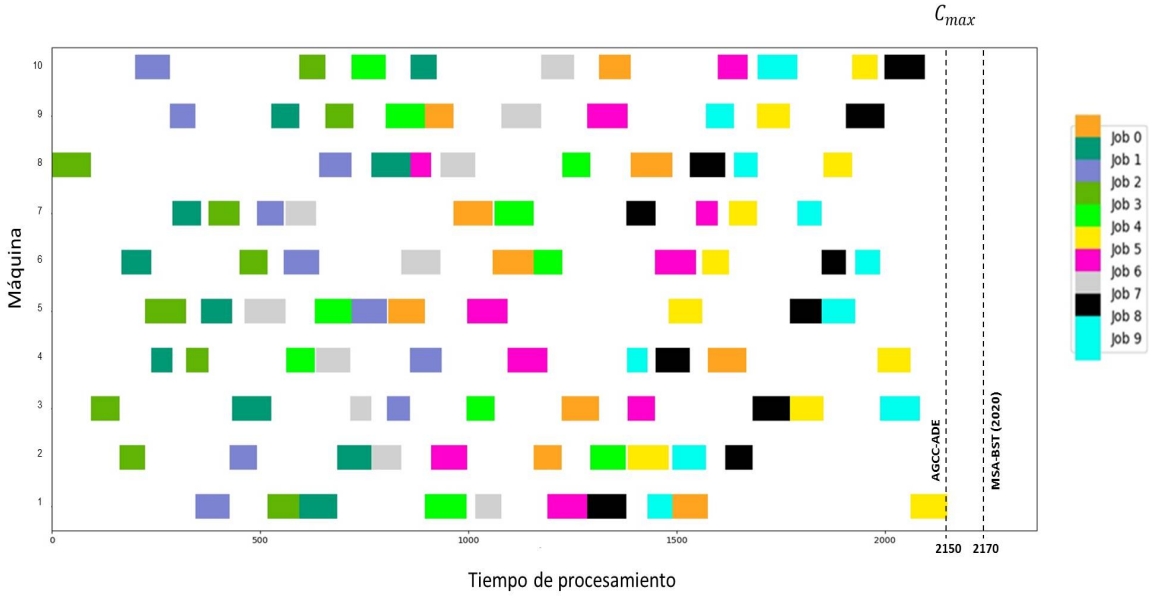

Figura 4.5: Nueva solución encontrada con AGCC-ADE para el caso de prueba Abz5 (n=10, m=10).

La Figura 4.6 muestra que AGCC-ADE encuentra un nuevo valor del C_{max}= 2633 para el caso de prueba Ta02, el cual supera a la mejor solución conocida (C_{max}= 2646) hasta hoy, obtenida por MSA-BST en el año 2020.

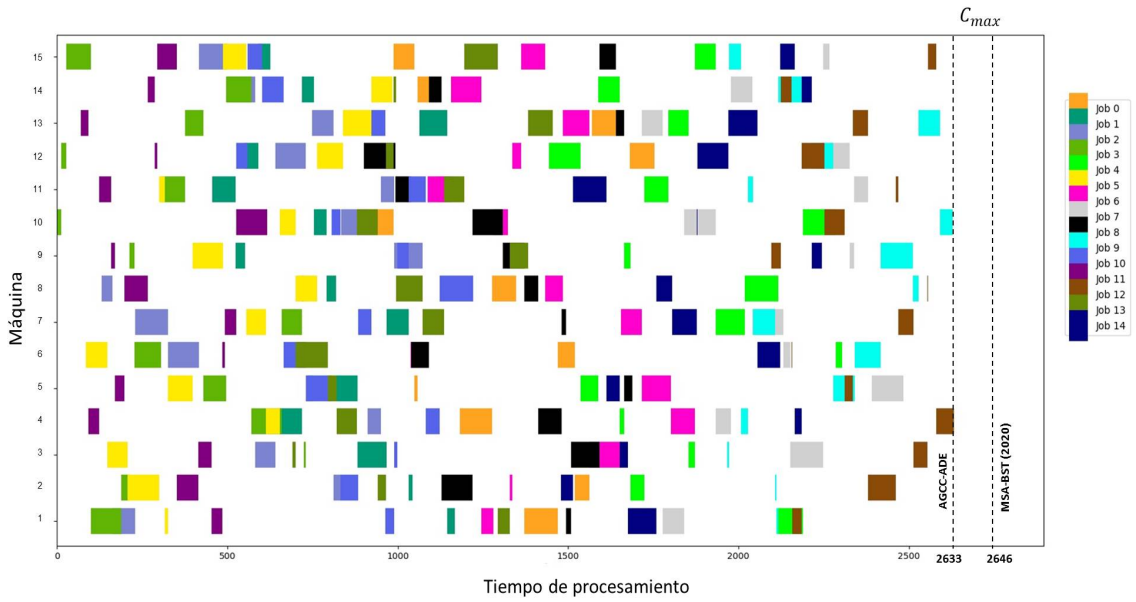

Figura 4.6: Nueva solución encontrada con AGCC-ADE para el caso de prueba Ta02 (n=15, m=15).

La Figura 4.7 presenta la gráfica de Gantt para el caso de prueba Ta07, se observa que AGCC-ADE obtiene un valor del C_{max}= 2635, el cual es menor que el C_{max}= 2684 (mejor solución conocida) obtenido previamente por MSA-BST.

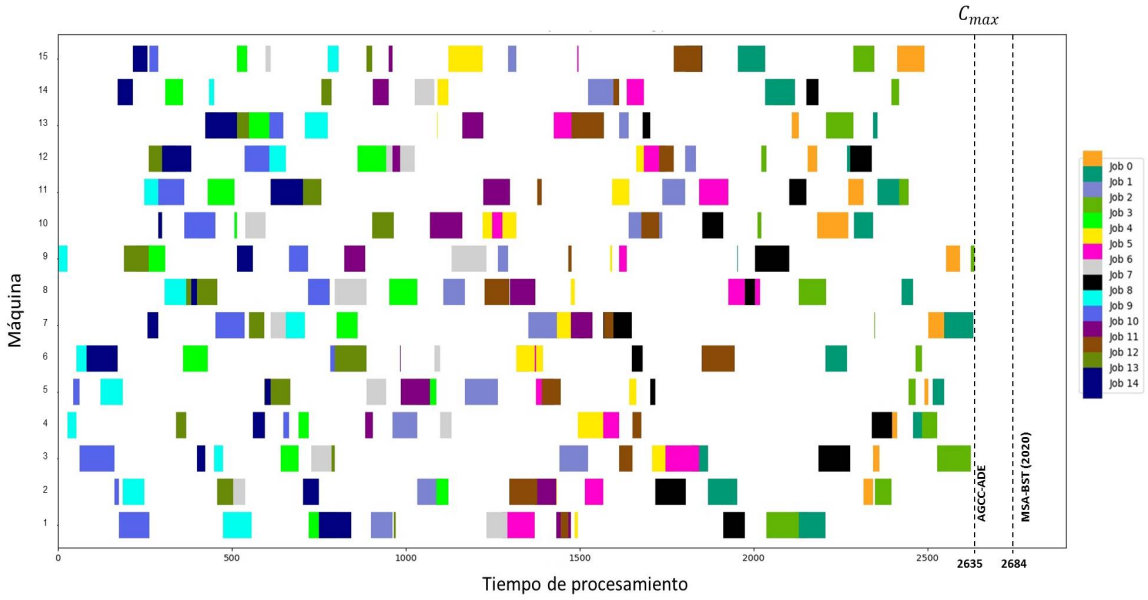

Figura 4.7: Nueva solución encontrada con AGCC-ADE para el caso de prueba Ta07 (n=15, m=15).

Capítulo 5

Conclusiones y trabajo futuro

En este capítulo se presentan las conclusiones basadas en los resultados obtenidos, las observaciones y experiencias. Además, se plantea el trabajo futuro derivado de esta investigación.

5.1. Conclusiones

Un factor clave para el éxito de las metaheurísticas empleadas en la literatura para resolver el NWJSSP depende por una parte, de la representación/decodificación de soluciones factibles y por otra parte de la estrategia de programación empleada para establecer los tiempos de inicio.

En esta tesis se proponen una estrategia de decodificación combinada que utiliza simultáneamente un procedimiento de decodificación estándar con un nuevo procedimiento de decodificación en sentido inverso. Esta decodificación es utilizada en el proceso de un algoritmo genético (*mix*-D/AG) para encontrar soluciones de alta calidad. El procedimiento de decodificación propuesto, llamado decodificación reversa, es una variante simple de la decodificación estándar para la representación basada en trabajos. Los experimentos computacionales muestran que aunque el procedimiento de decodificación reversa por sí solo no siempre mejora el procedimiento de decodificación estándar, la selección de la mejor de las dos soluciones siempre mejora la solución de ambas, incluso cuando el esfuerzo computacional de una sola decodificación y de la combinación se igualan.

Además, se propone por primera vez un AG coevolutivo cooperativo (AGCC) que evoluciona una estrategia de programación de horarios. Para ello se propone el Algoritmo de Decodificación de Empuje (ADE). El ADE combina dos estrategias de empuje para determinar el tiempo de inicio de cada trabajo en una secuencia determinada. El AGCC trabaja junto con el ADE (AGCC-ADE), ambas estrategias determinan qué

trabajos se empujan a la izquierda lo más cerca posible del tiempo cero y qué trabajos se empujan a la izquierda lo más cerca posible del tiempo de inactividad más reciente en la máquina correspondiente.

Los resultados obtenidos por el AGCC-ADE se compararon con los resultados obtenidos por los algoritmos más competitivos de la literatura (CLLM, MCLM, HABC, MSA-BST, GASA, TS y HTS) en un conjunto de casos de prueba. Estos resultados mostraron que la propuesta AGCC-ADE produce resultados competitivos para resolver el NWJSSP. Además, AGCC-ADE es capaz de encontrar nuevos valores para cuatro casos de prueba de tamaño grande en comparación con las mejores soluciones conocidas en la literatura. Estos resultados prometedores regresan a los algoritmos evolutivos a la competencia como un serio contendiente para el desafiante NWJSSP.

5.2. Trabajo futuro

Como trabajo futuro, se propone incluir unidades de retraso en el tiempo inicio de algunos trabajos como parte de los cromosomas en la población binaria, de manera que las opciones de empuje sean empuje a la izquierda lo más cerca posible del tiempo cero o retrasar k unidades de tiempo desde el último tiempo de inactividad de la máquina de inicio del trabajo. Esta idea está motivada por los resultados obtenidos por el enfoque más competitivo recientemente propuesto, MSA-BST [25].

5.3. Productos de investigación

1. Publicación de artículo de conferencia:
   Víctor M Valenzuela, Carlos A Brizuela, MA Cosío-León, and A Danisa Romero-Ocaño. A combination of two simple decoding strategies for the no-wait job shop scheduling problem. In Proceedings of the Genetic and Evolutionary Computation Conference, pages 864–871, 2019. DOI: 10.1145/3321707.3321870.

2. Publicación de artículo de revista:
   Víctor M Valenzuela-Alcaraz, MA Cosío-León, A Danisa Romero-Ocaño, and Carlos A Brizuela. A cooperative coevolutionary algorithm approach to the no-wait job shop scheduling problem. Expert Systems with Applications, page 116498, 2022. DOI: 10.1016/j.eswa.2022.116498.

Bibliografía

[1] Ali Allahverdi. A survey of scheduling problems with no-wait in process. *European Journal of Operational Research*, 255(3):665–686, 2016.

[2] Nicholas G Hall and Chelliah Sriskandarajah. A survey of machine scheduling problems with blocking and no-wait in process. *Operations research*, 44(3):510–525, 1996.

[3] Michael Pinedo. *Scheduling*, volume 5. Springer, 2012.

[4] Stefan Voß. Meta-heuristics: The state of the art. In *Workshop on Local Search for Planning and Scheduling*, pages 1–23. Springer, 2000.

[5] John F Muth and Gerald Luther Thompson. *Industrial scheduling*. Prentice-Hall, 1963.

[6] Hsiao-Lan Fang, Peter Ross, and David Corne. *A promising genetic algorithm approach to job-shop scheduling, rescheduling, and open-shop scheduling problems*. University of Edinburgh, Department of Artificial Intelligence, 1993.

[7] Ulrich Dorndorf, Erwin Pesch, and Toàn Phan-Huy. Solving the open shop scheduling problem. *Journal of Scheduling*, 4(3):157–174, 2001.

[8] JR King. The theory-practice gap in job-shop scheduling. *Production Engineer*, 55(3):137–143, 1976.

[9] Yailen Martínez. A generic multi-agent reinforcement learning approach for scheduling problems. *Vrije Universiteit Brussel*, 2012.

[10] Weiming Shen. Distributed manufacturing scheduling using intelligent agents. *IEEE intelligent systems*, 17(1):88–94, 2002.

[11] Michael R Garey, David S Johnson, and Ravi Sethi. The complexity of flowshop and jobshop scheduling. *Mathematics of operations research*, 1(2):117–129, 1976.

[12] Runwei Cheng, Mitsuo Gen, and Yasuhiro Tsujimura. A tutorial survey of job-shop scheduling problems using genetic algorithms—i. representation. *Computers & industrial engineering*, 30(4):983–997, 1996.

[13] Mitsuo Gen. Solving job-shop scheduling problem using genetic algorithm. In *Proc. of 16th International Conference on Computers & Industrial Engineering*, pages 576–579, 1994.

[14] Clyde W Holsapple, Varghese S Jacob, Ramakrishnan Pakath, and Jigish S Zaveri. A genetics-based hybrid scheduler for generating static schedules in flexible manufacturing contexts. *IEEE Transactions on Systems, Man, and Cybernetics*, 23(4):953–972, 1993.

[15] Ulrich Dorndorf and Erwin Pesch. Evolution based learning in a job shop scheduling environment. *Computers & Operations Research*, 22(1):25–40, 1995.

[16] Lawrence Davis. Job shop scheduling with genetic algorithms. In *Proceedings of an international conference on genetic algorithms and their applications*, volume 140. Carnegie-Mellon University Pittsburgh, Pennsylvania, 1985.

[17] James C Bean. Genetic algorithms and random keys for sequencing and optimization. *ORSA journal on computing*, 6(2):154–160, 1994.

[18] Takeshi Yamada and Ryohei Nakano. A genetic algorithm applicable to large-scale job-shop problems. In *PPSN*, volume 2, pages 281–290, 1992.

[19] Ryohei Nakano and Takeshi Yamada. Conventional genetic algorithm for job shop problems. In *ICGA*, volume 91, pages 474–479, 1991.

[20] Hisashi Tamaki. A paralleled genetic algorithm based on a neighborhood model and its application to the jobshop scheduling. *Parallel Problem Solving from Nature 2*, pages 573–582, 1992.

[21] Jie Zhu, Xiaoping Li, and Qian Wang. Complete local search with limited memory algorithm for no-wait job shops to minimize makespan. *European Journal of Operational Research*, 198(2):378–386, 2009.

[22] Jie Zhu and Xiaoping Li. An effective meta-heuristic for no-wait job shops to minimize makespan. *IEEE Transactions on Automation Science and Engineering*, 9(1):189–198, 2012.

[23] Shyam Sundar, Ponnuthurai N Suganthan, Chua Tay Jin, Cai Tian Xiang, and Chong Chin Soon. A hybrid artificial bee colony algorithm for the job-shop scheduling problem with no-wait constraint. *Soft Computing*, 21(5):1193–1202, 2017.

[24] Sachchida Nand Chaurasia, Shyam Sundar, Donghwi Jung, Ho Min Lee, and Joong Hoon Kim. An evolutionary algorithm based hyper-heuristic for the job-shop scheduling problem with no-wait constraint. In *Harmony Search and Nature Inspired Optimization Algorithms*, pages 249–257. Springer, 2019.

[25] Kuo-Ching Ying and Shih-Wei Lin. Solving no-wait job-shop scheduling problems using a multi-start simulated annealing with bi-directional shift timetabling algorithm. *Computers & Industrial Engineering*, page 106615, 2020.

[26] A Ozolins. A new exact algorithm for no-wait job shop problem to minimize makespan. *Operational Research*, pages 1–31, 2018.

[27] Christoph J Schuster and Jose M Framinan. Approximative procedures for no-wait job shop scheduling. *Operations Research Letters*, 31(4):308–318, 2003.

[28] Christoph J Schuster. No-wait job shop scheduling: Tabu search and complexity of subproblems. *Mathematical Methods of Operations Research*, 63(3):473–491, 2006.

[29] Wojciech Bożejko and Mariusz Makuchowski. A fast hybrid tabu search algorithm for the no-wait job shop problem. *Computers & Industrial Engineering*, 56(4):1502–1509, 2009.

[30] Jürgen Dorn and Reza Shams. Scheduling high-grade steelmaking. *IEEE Expert*, 11(1):28–35, 1996.

[31] Lixin Tang, Jiyin Liu, Aiying Rong, and Zihou Yang. A mathematical programming model for scheduling steelmaking-continuous casting production. *European Journal of Operational Research*, 120(2):423–435, 2000.

[32] Wenny HM Raaymakers and JA Hoogeveen. Scheduling multipurpose batch process industries with no-wait restrictions by simulated annealing. *European Journal of Operational Research*, 126(1):131–151, 2000.

[33] Chandrasekharan Rajendran. A no-wait flowshop scheduling heuristic to minimize makespan. *Journal of the Operational Research Society*, 45(4):472–478, 1994.

[34] Józef Grabowski, Jarosław Pempera, and Czesław Smutnicki. Scheduling in production of concrete wares. In *Operations Research Proceedings 1996*, pages 192–196. Springer, 1997.

[35] Jozef Grabowski and Jaroslaw Pempera. Sequencing of jobs in some production system. *European journal of operational research*, 125(3):535–550, 2000.

[36] SS Reddi and CV Ramamoorthy. A scheduling problem. *Journal of the Operational Research Society*, 24(3):441–446, 1973.

[37] Jan Karel Lenstra, AHG Rinnooy Kan, and Peter Brucker. Complexity of machine scheduling problems. *Studies in integer programming*, 1:343–362, 1977.

[38] Ronald L Graham, Eugene L Lawler, Jan Karel Lenstra, and AHG Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. In *Annals of discrete mathematics*, volume 5, pages 287–326. Elsevier, 1979.

[39] Carlos A Brizuela, Yong Zhao, and Nobuo Sannomiya. No-wait and blocking job-shops: Challenging problems for ga's. In *Systems, Man, and Cybernetics, 2001 IEEE International Conference on*, volume 4, pages 2349–2354. IEEE, 2001.

[40] El-Ghazali Talbi. *Metaheuristics: from design to implementation*, volume 74. John Wiley & Sons, 2009.

[41] Anupriya Gogna and Akash Tayal. Metaheuristics: review and application. *Journal of Experimental & Theoretical Artificial Intelligence*, 25(4):503–526, 2013.

[42] Sartaj Sahni and Yookun Cho. Complexity of scheduling shops with no wait in process. *Mathematics of Operations Research*, 4(4):448–457, 1979.

[43] Wieslaw Kubiak. A pseudo-polynomial algorithm for a two-machine no-wait job-shop scheduling problem. *European journal of operational research*, 43(3):267–270, 1989.

[44] Svetlana A Kravchenko. A polynomial algorithm for a two-machine no-wait job-shop scheduling problem. *European Journal of Operational Research*, 106(1):101–107, 1998.

[45] Gerhard J Woeginger. Inapproximability results for no-wait job shop scheduling. *Operations Research Letters*, 32(4):320–325, 2004.

[46] Nikhil Bansal, Mohammad Mahdian, and Maxim Sviridenko. Minimizing makespan in no-wait job shops. *Mathematics of Operations Research*, 30(4):817–831, 2005.

[47] Alessandro Mascis and Dario Pacciarelli. Job-shop scheduling with blocking and no-wait constraints. *European Journal of Operational Research*, 143(3):498–517, 2002.

[48] Abdelhakim AitZai, Brahim Benmedjdoub, and Mourad Boudhar. Branch-and-bound and pso algorithms for no-wait job shop scheduling. *Journal of Intelligent Manufacturing*, 27(3):679–688, 2016.

[49] HM Vermeulen, JA Hoogeveen, and JM van den Akker. Solving the no-wait job shop problem: an ilp and cp approach. *CPAIOR 2011 Late Breaking Abstracts*, page 44, 2011.

[50] Christos Koulamas and SS Panwalkar. The proportionate two-machine no-wait job shop scheduling problem. *European Journal of Operational Research*, 252(1):131–135, 2016.

[51] Roberto Macchiaroli, S Mole, and S Riemma. Modelling and optimization of industrial manufacturing processes subject to no-wait constraints. *International Journal of Production Research*, 37(11):2585–2607, 1999.

[52] Jose M Framinan and Christoph Schuster. An enhanced timetabling procedure for the no-wait job shop problem: a complete local search approach. *Computers & Operations Research*, 33(5):1200–1213, 2006.

[53] Ching-Fang Liaw. An efficient simple metaheuristic for minimizing the makespan in two-machine no-wait job shops. *Computers & operations research*, 35(10):3276–3283, 2008.

[54] Hamed Samarghandi, Tarek Y ElMekkawy, and Al-Mehdi M Ibrahem. Studying the effect of different combinations of timetabling with sequencing algorithms to solve the no-wait job shop scheduling problem. *International Journal of Production Research*, 51(16):4942–4965, 2013.

[55] Shao-Feng Chen, Bin Qian, Rong Hu, and Zuo-Cheng Li. An enhanced estimation of distribution algorithm for no-wait job shop scheduling problem with makespan criterion. In *International Conference on Intelligent Computing*, pages 675–685. Springer, 2014.

[56] Xiaoping Li, Haiyan Xu, and Minmin Li. A memory-based complete local search method with variable neighborhood structures for no-wait job shops. *The International Journal of Advanced Manufacturing Technology*, 87(5-8):1401–1408, 2016.

[57] Guanlong Deng, Zhiwang Zhang, Tianhua Jiang, and Shuning Zhang. Total flow time minimization in no-wait job shop using a hybrid discrete group search optimizer. *Applied Soft Computing*, 81:105480, 2019.

[58] Jason Chao-Hsien Pan and Han-Chiang Huang. A hybrid genetic algorithm for no-wait job shop scheduling problems. *Expert Systems with Applications*, 36(3):5800–5806, 2009.

[59] Merrill M Flood. The traveling-salesman problem. *Operations research*, 4(1):61–75, 1956.

[60] Wojciech Bożejko and Mariusz Makuchowski. Solving the no-wait job-shop problem by using genetic algorithm with automatic adjustment. *The International Journal of Advanced Manufacturing Technology*, 57(5-8):735–752, 2011.

[61] Budi Santosa, Muhammad Arif Budiman, Stefanus Eko Wiratno, et al. A cross entropy-genetic algorithm for m-machines no-wait job-shopscheduling problem. *Journal of Intelligent Learning Systems and Applications*, 3(03):171, 2011.

[62] Víctor M Valenzuela, Carlos A Brizuela, MA Cosío-León, and A Danisa Romero-Ocaño. A combination of two simple decoding strategies for the no-wait job shop scheduling problem. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 864–871, 2019.

[63] Guanlong Deng, Qingtang Su, Zhiwang Zhang, Huixia Liu, Shuning Zhang, and Tianhua Jiang. A population-based iterated greedy algorithm for no-wait job shop scheduling with total flow time criterion. *Engineering Applications of Artificial Intelligence*, 88:103369, 2020.

[64] Mitchell A Potter and Kenneth A De Jong. A cooperative coevolutionary approach to function optimization. In *International Conference on Parallel Problem Solving from Nature*, pages 249–257. Springer, 1994.

[65] John Holland. Adaptation in natural and artificial systems: an introductory analysis with application to biology. *Control and artificial intelligence*, 1975.

[66] Haiping Ma, Shigen Shen, Mei Yu, Zhile Yang, Minrui Fei, and Huiyu Zhou. Multi-population techniques in nature inspired optimization algorithms: A comprehensive survey. *Swarm and evolutionary computation*, 44:365–387, 2019.

[67] Kenneth Alan De Jong. Analysis of the behavior of a class of genetic adaptive systems. 1975.

[68] Tadahiko Murata and Hisao Ishibuchi. Positive and negative combination effects of crossover and mutation operators in sequencing problems. In *Proceedings of IEEE International Conference on Evolutionary Computation*, pages 170–175. IEEE, 1996.

[69] Riccardo Poli and William B Langdon. Genetic programming with one-point crossover. In *Soft Computing in Engineering Design and Manufacturing*, pages 180–189. Springer, 1998.

[70] David E Goldberg, Robert Lingle, et al. Alleles, loci, and the traveling salesman problem. In *Proceedings of an international conference on genetic algorithms and their applications*, volume 154, pages 154–159. Lawrence Erlbaum, Hillsdale, NJ, 1985.

[71] Agoston E Eiben, James E Smith, et al. *Introduction to evolutionary computing*, volume 53. Springer, 2003.

[72] Henry Fisher. Probabilistic learning combinations of local job-shop scheduling rules. *Industrial scheduling*, pages 225–251, 1963.

[73] Thomas Bäck, David B Fogel, and Zbigniew Michalewicz. *Evolutionary computation 1: Basic algorithms and operators*. CRC press, 2018.

[74] Jorge MS Valente, José Fernando Gonçalves, and Rui AFS Alves. A hybrid genetic algorithm for the early/tardy scheduling problem. *Asia-Pacific Journal of operational research*, 23(03):393–405, 2006.

[75] Douglas C Montgomery. *Design and analysis of experiments*. John wiley & sons, 2017.

[76] Xiaoliang Ma, Xiaodong Li, Qingfu Zhang, Ke Tang, Zhengping Liang, Weixin Xie, and Zexuan Zhu. A survey on cooperative co-evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 23(3):421–441, 2018.

[77] Jie Zheng, Ling Wang, and Jing-jing Wang. A cooperative coevolution algorithm for multi-objective fuzzy distributed hybrid flow shop. *Knowledge-Based Systems*, page 105536, 2020.

[78] Quan-Ke Pan, Liang Gao, and Ling Wang. An effective cooperative co-evolutionary algorithm for distributed flowshop group scheduling problems. *IEEE Transactions on Cybernetics*, 2020.

[79] R Paul Wiegand, William C Liles, and Kenneth A De Jong. An empirical analysis of collaboration methods in cooperative coevolutionary algorithms. In *Proceedings of the genetic and evolutionary computation conference (GECCO)*, volume 2611, pages 1235–1245, 2001.

[80] Xiaoliang Ma, Fang Liu, Yutao Qi, Xiaodong Wang, Lingling Li, Licheng Jiao, Minglei Yin, and Maoguo Gong. A multiobjective evolutionary algorithm based on decision variable analyses for multiobjective optimization problems with large-scale variables. *IEEE Transactions on Evolutionary Computation*, 20(2):275–298, 2015.

[81] Franciszek Seredynski. Loosely coupled distributed genetic algorithms. In *International Conference on Parallel Problem Solving from Nature*, pages 514–523. Springer, 1994.

[82] Maximilian Schiffer and Grit Walther. The electric location routing problem with time windows and partial recharging. *European Journal of Operational Research*, 260(3):995–1013, 2017.

[83] M. A. Potter and K. Jong. A cooperative coevolutionary approach to function optimization. In *PPSN*, 1994.