EDUCACIÓN
SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLÓGICO NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE CIUDAD MADERO

DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

MAESTRÍA EN CIENCIAS DE LA COMPUTACIÓN

"POR MI PATRIA Y POR MI BIEN"

TESIS

OPTIMIZACIÓN GENERALIZADA PARA PROBLEMAS DE AGRUPACIÓN: UN ENFOQUE COEVOLUTIVO AUTO-ADAPTATIVO

Que para obtener el Grado de

Maestro en Ciencias de la Computación

Presenta

Ing. Jordan Michelt Aran Perez

G16070192

1154241

Director de Tesis

Dra. Laura Cruz Reyez

122925

Co-director de Tesis

Dr. Bernabé Dorronsoro

Cd. Madero, Tamaulipas — Febrero, 2024

Ciudad Madero, Tamaulipas, 15/junio/2023

Oficio No.: U.091/2023
Asunto: Autorización de impresión de tesis

C. JORDÁN MICHELT ARAN PÉREZ
No. DE CONTROL G16070192
P R E S E N T E

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

"OPTIMIZACIÓN GENERALIZADA PARA PROBLEMAS DE AGRUPACIÓN: UN ENFOQUE COEVOLUTIVO AUTO-ADAPTATIVO"

El Jurado está integrado por los siguientes catedráticos:

PRESIDENTA:	DR.	NELSON RANGEL VALDEZ
SECRETARIO:	DRA.	CLAUDIA GUADALUPE GÓMEZ SANTILLÁN
VOCAL:	DRA.	LAURA CRUZ REYES
SUPLENTE:	DR.	HÉCTOR JOAQUÍN FRAIRE HUACUJA
DIRECTORA DE TESIS:	DRA.	LAURA CRUZ REYES
CO-DIRECTOR:	DR.	BERNABÉ DORRONSORO DÍAZ

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

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

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

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

ccp. Archivo
MACG 'MLMR'

Declaración de originalidad

Yo, Jordan Michelt Aran Perez, en mi calidad de autor manifiesto que este documento de tesis es producto original de mi trabajo y que no infringe derechos de terceros, tales como derechos de publicación, derechos de autor, patentes y similares. Por lo tanto, la obra es de mi exclusiva autoría y soy titular de los derechos que surgen de la misma.

Asimismo, declaro que en las citas textuales que he incluido y en los resúmenes que he realizado de publicaciones ajenas, indico explícitamente los datos de los autores y las publicaciones.

En caso de presentarse cualquier reclamación o acción por parte de un tercero en cuanto a los derechos de autor sobre la obra en cuestión, asumiré toda la responsabilidad y relevo de ésta a mi director de tesis, así como al Tecnológico Nacional de México, Instituto Tecnológico de Ciudad Madero y a sus respectivas autoridades.

Cd. Madero, Tamaulipas. Junio 2023.

_____

ISC. Jordan Michelt Aran Perez

Resumen

Este estudio introduce un Algoritmo Coevolutivo de Agrupación con Cooperación Auto-Adaptativa (GCA-AC), diseñado como una solución general a problemas de agrupación. En particular, se centra en dos problemas específicos: el Problema de Empaquetado de Contenedores de una Dimensión (BPP-1D) y el Problema de Programación de Máquinas Paralelas (PMS).

Existe un vasto abanico de algoritmos genéticos y estrategias evolutivas que han sido propuestas como soluciones a la familia de los problemas de agrupación. Sin embargo, estos métodos pueden tener limitaciones en cuanto a adaptabilidad y rendimiento, lo cual motiva la búsqueda de nuevas estrategias que combinen y mejoren las existentes.

El GCA-AC se presenta como una innovadora solución que combina diversas estrategias evolutivas para mejorar su adaptabilidad y rendimiento. Una de las principales contribuciones de este estudio es el mecanismo de auto-adaptación del GCA-AC, que permite ajustar dinámicamente sus parámetros, ofreciendo una respuesta más eficiente a los cambios en el ambiente del problema.

Los resultados de la investigación indican que el GCA-AC es competente en la búsqueda de soluciones de calidad para los problemas BPP-1D y PMS, comparables con las generadas por algoritmos especializados ya existentes. Este estudio propone que la combinación de algoritmos genéticos con estrategias de coevolución y autoadaptación puede ser un enfoque fructífero para el desarrollo de algoritmos de agrupación universales.

A pesar de los prometedores resultados que este estudio proporciona, se reconoce que aún hay mucho por descubrir. Por lo tanto, se sugieren múltiples oportunidades para futuras líneas de investigación.

Abstract

This study introduces a Grouping Coevolutionary Algorithm with Auto-Adaptive Cooperation (GCA-AC), designed as a general solution to clustering problems. In particular, it focuses on two specific problems: the One-Dimensional Bin Packing Problem (BPP-1D) and the Parallel Machine Scheduling Problem (PMS).

There is a vast array of genetic algorithms and evolutionary strategies that have been proposed as solutions to the family of clustering problems. However, these methods may have limitations in terms of adaptability and performance, which motivates the search for new strategies that combine and improve the existing ones.

The GCA-AC is presented as an innovative solution that combines various evolutionary strategies to enhance its adaptability and performance. One of the main contributions of this study is the GCA-AC's self-adaptation mechanism, which allows dynamic adjustment of its parameters, providing a more efficient response to changes in the problem environment.

The research results indicate that the GCA-AC is competent in searching for quality solutions for the BPP-1D and PMS problems, comparable to those generated by already existing specialized algorithms. This study suggests that the combination of genetic algorithms with coevolution and self-adaptation strategies can be a fruitful approach for the development of universal clustering algorithms.

Despite the promising results this study provides, it is acknowledged that there is still much to discover. Therefore, multiple opportunities are suggested for future lines of research.

Índice general

Capítulo 1

Introducción

Los problemas de agrupación representan una categoría crucial dentro del ámbito de la computación, en los que un conjunto de elementos se segmenta en diversos subconjuntos. Con una variedad de aplicaciones en situaciones prácticas, estos problemas han despertado un interés creciente por su nivel de complejidad y por el desafío que suponen para su resolución.

En la mayoría de los contextos, los problemas de agrupación requieren que cada elemento exista exclusivamente en un grupo, cumpliendo con un conjunto específico de restricciones. Dada su naturaleza, estos problemas de la vida real son altamente complejos y se clasifican dentro de la categoría de problemas NP-duros. Esta categoría incluye problemas para los cuales no se conocen algoritmos capaces de proporcionar una solución óptima en un tiempo polinomial.

Por esta razón, el diseño de algoritmos para resolver estos problemas puede ser una tarea considerablemente ardua. A pesar de que los algoritmos heurísticos han demostrado ser efectivos, la mayoría están diseñados para abordar un caso específico de un problema de agrupación. Hasta la fecha, pocos algoritmos han sido capaces de abordar más de dos tipos diferentes de problemas de agrupación.

Este trabajo se centra en la propuesta y desarrollo de un algoritmo que puede manejar eficientemente al menos dos tipos diferentes de problemas de agrupación. Se adaptará el Algoritmo Genético de Agrupación con Transmisión de Gen Controlada (GGA-CGT), una metodología desarrollada por Quiroz [2014], que ha demostrado su capacidad para abordar casi todas las instancias del problema de empaquetado de contenedores de una dimensión (Bin Packing Problem, BPP). Posteriormente, González [2021] introdujo mejoras a este algoritmo, incorporando nuevas estrategias para controlar los niveles de intensificación y diversificación a través del aprendizaje reforzado.

Entre las estrategias que han demostrado potencial para aumentar la capacidad de resolución de los metaheurísticos se destacan los enfoques hiperheurísticos, la

coevolución y la autoadaptabilidad de los parámetros. En este proyecto, se propone una estrategia de coevolución basada en el GGA-CGT, para mejorar su capacidad de resolución. Además, se propone un método de la auto-adaptación de algunos de sus parámetros basado en aprendizaje reforzado.

1.1. Antecedentes

Este trabajo se plantea en el contexto de un proyecto de investigación sobre algoritmia experimental llevado a cabo en la División de Estudios de Posgrado e Investigación del Instituto Tecnológico de Ciudad Madero (ITCM). En particular se tiene interés en estudiar el comportamiento de un algoritmo del estado del arte que da solución a un problema de agrupación mono-objetivo para ser adaptado con la finalidad de ampliar su capacidad de solución a dos problemas.

La línea de trabajo en problemas de agrupación en el ITCM lo lidera la Dra. Laura Cruz, e inicia con el desarrollo de algoritmos de solución al problema de empacado de contenedores (BPP, por su siglas en inglés. García incorpora retroalimentación a la arquitectura de selección y caracteriza el comportamiento algorítmico (García, 2004). Posteriormente, Álvarez incorpora nuevos índices de caracterización de BPP (Alvarez, 2006).

García muestra la aplicabilidad de la caracterización del comportamiento algorítmico al desarrollar dos índices de la intensidad del uso de heurísticas dentro de un algoritmo hiperheurístico, esto permitió identificar la necesidad de incluir un mayor uso de técnicas de exploración para mejorar el desempeño algorítmico (García, 2010).

Quiroz [2014] identificó factores que influyen en el desempeño y se cuantifican mediante índices. Además, identificó propiedades que definen la estructura de una instancia de BPP y explicó el desempeño de los algoritmos. Esto quedó plasmado en el algoritmo denominado GGA-CGT. Este algoritmo fue modificado posteriormente por González [2021] quien agrego nuevas estrategias para mejorarlo.

Recientemente el trabajo de Ramos-Figueroa [2022] se menciona como una continuación de los esfuerzos en la misma línea de investigación del equipo liderado por Quiroz en la Universidad Veracruzana. Al igual que los otros investigadores mencionados, Ramos-Figueroa está buscando mejorar la forma en que se resuelven los problemas de agrupación. En su caso, se centra en mejorar la programación de máquinas paralelas (Parallel Machine Scheduling, PMS). Para esto, desarrolló un algoritmo genético, el Final Grouping Genetic Algorithm (FGGA). Su objetivo es optimizar la asignación de tareas en máquinas con diferentes capacidades para minimizar el tiempo total de procesamiento, conocido como makespan.

Los algoritmos que mejor han resuelto a BPP y PMS son GGA-CGT y FGGA, respectivamente. Dado que estos algoritmos fueron especializados para un problema,

en este trabajo de tesis se agregan a GGA-CGT estrategias de generalización para dar solución a ambos problemas de agrupación.

1.2. Objetivos de la tesis

1.2.1. Objetivo General

Desarrollar un algoritmo coevolutivo auto-adaptable para la optimización de dos problemas de agrupación, con la finalidad de producir resultados competitivos en cada uno de los problemas en términos de generalidad y efectividad con respecto al estado-del-arte.

1.2.2. Objetivos Específicos

1. Analizar y seleccionar por lo menos una estrategia metaheurística del estado del arte aplicadas a la solución de los problemas de agrupación que se abordarán.
2. Desarrollar un algoritmo coevolutivo de optimización basado en las estrategias de solución seleccionadas.
3. Configurar de manera auto-adaptativa la interacción de los parámetros del algoritmo.
4. Analizar estadísticamente el desempeño en optimalidad y nivel de generalidad del algoritmo coevolutivo propuesto, contrastando su desempeño con algoritmos del estado del arte usados en la solución de los problemas de agrupación abordados en este proyecto.

1.3. Descripción del Problema

El problema de investigación que se aborda en esta tesis se centra en el diseño de un algoritmo coevolutivo que pueda resolver de manera eficiente los problemas Bin Packing (BPP) y Parallel Machine Scheduling (PMS). La descripción formal del problema se puede presentar de la siguiente manera:

Dados:

1. Un conjunto de problemas de agrupación $P = \{p_1, p_2\}$, donde p_1 se refiere al problema de Bin Packing (BPP) y p_2 al problema de Parallel Machine Scheduling (PMS).

2. Un conjunto de estrategias de solución $E = \{e_1, e_2, \ldots, e_m\}$, las cuales se denominan especies y están soportadas en propuestas del estado del arte para resolver de manera independiente estos tipos de problemas.

Se busca:

1. Desarrollar un algoritmo coevolutivo que combine dichas estrategias de solución, de manera que sean capaces de resolver los problemas contenidos en P de manera efectiva.
2. Identificar y determinar los parámetros auto-adaptativos a emplear en el algoritmo coevolutivo. Estos deben ser capaces de modificar su magnitud y carácter de manera tal que se logre mejorar la solución a cada problema con respecto a los métodos presentes en el estado del arte.

El desafío aquí radica en la creación de un algoritmo que no sólo sea capaz de manejar dos problemas diferentes de agrupación, sino que además logre auto-adaptarse para mejorar su rendimiento con el tiempo. Esto implica un análisis profundo de las estrategias existentes y un proceso de selección riguroso para determinar cuáles de ellas se integrarán al algoritmo propuesto. Asimismo, se debe realizar una investigación exhaustiva y una serie de experimentaciones para establecer los parámetros que permitirán al algoritmo autoadaptarse de forma eficiente. La meta final es que el algoritmo propuesto sea equiparable en rendimiento a los métodos actuales y ofrezca una mayor generalidad.

1.4. Justificación

El ritmo acelerado del desarrollo en las ciencias de la computación, en conjunto con la creciente demanda de soluciones más eficientes para problemas cada vez más complejos, han generado un diverso panorama de algoritmos de optimización. No obstante, la especialización inherente de estos algoritmos, diseñados a menudo para tratar un conjunto limitado de problemas específicos, limita su rendimiento y adaptabilidad al enfrentarse a escenarios diferentes a los que fueron originalmente concebidos.

La propuesta de un algoritmo que pueda generalizarse y adaptarse a un espectro más amplio de problemas de agrupación es de vital importancia, no sólo desde una perspectiva científica y técnica, sino también desde un enfoque económico. El desarrollo de un algoritmo capaz de resolver una variedad de problemas permitiría optimizar tiempo y recursos, eliminando la necesidad de diseñar o implementar soluciones específicas para cada escenario particular.

Además, este enfoque alienta la consolidación de las ciencias de la computación como una disciplina científica robusta. Como se evidencia en los casos anteriores, es

esencial adoptar técnicas científicas rigurosas para fortalecer teorías, algoritmos y sistemas. Una investigación orientada en esta dirección facilitará una mejor comprensión del comportamiento de los algoritmos, permitiendo identificar sus limitaciones y prever su desempeño futuro.

Este trabajo se focaliza en el diseño y desarrollo de un algoritmo coevolutivo con un enfoque generalizado, capaz de abordar más de un tipo de problema de agrupación. Se busca así, contribuir a cerrar la brecha existente en la literatura respecto a la generalización de algoritmos, y presentar una solución que se pueda aplicar a una gama más amplia de instancias, contribuyendo significativamente a un campo en constante evolución como es la algoritmia.

Al ampliar el marco de aplicación de los algoritmos, se aspira a inaugurar nuevas rutas de investigación y permitir que estas herramientas tecnológicas sean aún más versátiles y efectivas. En consecuencia, este trabajo tiene el potencial de aportar significativamente al avance de las ciencias de la computación y a la resolución eficiente de problemas de optimización, fortaleciendo así la base para futuras exploraciones en este campo.

1.5. Alcances y limitaciones

1.5.1. Alcances

1 Este estudio se aplica a la solución de dos problemas de agrupación relacionados, expandiendo la capacidad de resolver múltiples problemas de agrupación.

2 Se consideran dos enfoques para la generalización: la coevolución y la autoadaptación de parámetros, lo que proporciona un mayor grado de flexibilidad y adaptabilidad.

3 La estrategia de solución se basa en el algoritmo genético GGA-CGT, uno de los métodos más relevantes para este tipo de problemas.

4 Se configuran adaptativamente al menos dos parámetros, mejorando la capacidad del algoritmo para ajustarse a diferentes tipos de problemas y entornos.

5 La evaluación de la propuesta incluye varios indicadores de desempeño, incluyendo el número de óptimos encontrados, el error de solución, el tiempo de ejecución y el nivel de generalidad.

1.5.2. Limitaciones

1 A pesar de su amplio alcance, este estudio se limita a la solución de dos problemas de agrupación específicos. Por lo tanto, los resultados pueden no ser aplicables a todos los problemas de agrupación.

2 Solo se consideran instancias disponibles en la literatura para la experimentación, lo que podría limitar la diversidad y complejidad de los problemas abordados.

3 El número de parámetros que se pueden configurar adaptativamente está limitado a dos, lo que podría no ser suficiente para todos los casos de uso.

4 Los resultados y conclusiones de este estudio están sujetos a las limitaciones inherentes a los métodos de medición utilizados. Específicamente, la medida en número de óptimos encontrados y error de solución podrían no capturar todos los aspectos relevantes del rendimiento del algoritmo.

5 Aunque se realizan esfuerzos para generalizar el algoritmo, la verdadera generalidad de la solución solo puede evaluarse en la práctica y puede estar limitada por una serie de factores desconocidos.

1.6. Organización del documento

Este documento se estructura de manera lógica y sistemática, proporcionando un flujo coherente de información y análisis. Comenzando con este capítulo, donde se ofrece un panorama general de la tesis, presentando los temas clave que se tratarán en el texto.

A continuación, el Capítulo 2 sirve como base teórica de la investigación, introduciendo conceptos fundamentales para la comprensión de los problemas de agrupación que se tratan y evidenciando la complejidad inherente en su resolución.

El recorrido prosigue con el Capítulo 3, que aborda el estado del arte, proporcionando un vistazo al panorama actual de la investigación en esta área. Este capítulo compara algoritmos para los problemas tratados y pone de relieve el papel de los algoritmos coevolutivos en el contexto general.

En el corazón del estudio, el Capítulo 4 desglosa los problemas de agrupación abordados y presenta el modelo unificado para los problemas tratados. Este capítulo también explica la adaptación de las instancias de problemas y la modificación y adaptación del algoritmo GGA-CGT, una pieza central de la investigación. La sección final de este capítulo introduce la propuesta del algoritmo coevolutivo de agrupacion con coperacion auto-adaptativa (Grouping Coevolution Algorithm with Adaptive Cooperation, GCA-AC), detallando su diseño e implementación.

El Capítulo 5 ofrece una revisión exhaustiva de los resultados obtenidos. Mediante pruebas rigurosas y análisis detallados, se evalúa el rendimiento del algoritmo propuesto GCA-AC y se compara con otros algoritmos relevantes del estado del arte.

Finalmente, el Capítulo 6 concluye la tesis, recopilando los hallazgos, reflexiones y conclusiones derivadas del estudio. Este último capítulo también sugiere posibles

direcciones para futuras investigaciones, explorando nuevas formas de abordar los problemas de agrupación y proponiendo mejoras a la propuesta actual.

En cada etapa de este trabajo, se dedica el tiempo necesario para desglosar y explicar cada proceso, elección y resultado, siempre con el objetivo de proporcionar una comprensión completa y rica de la investigación desarrollada en esta tesis.

Capítulo 2

Marco Teórico

En este capitulo se discuten los fundamentos necesarios para comprender las técnicas y metodologías empleadas en este estudio. Los conceptos principales están relacionados con los problemas de agrupación y su resolución asi como en las diferentes estrategias de coevolución.

Además de discutir estos conceptos, se presentan y comparan diversas técnicas y algoritmos relevantes para la solución de problemas de agrupación. La comprensión de estas técnicas y algoritmos ayuda a justificar las elecciones metodológicas en este estudio y proporciona el contexto para su discusión y análisis.

2.1. Problemas de optimización

La optimización se entiende como el proceso mediante el cual se busca la mejor solución a un problema dado en un tiempo determinado (Duarte et al., 2007). En este contexto, un problema de optimización se puede definir como aquel en el que existen múltiples soluciones posibles, y donde se pueden comparar estas soluciones en términos de su calidad.

Formalmente, un problema de optimización P puede ser representado como $P = (f, SS, F)$ tal como se indica en la 2.1. En esta representación, f es la función objetivo que se busca maximizar o minimizar, F es el conjunto de soluciones factibles y SS es el espacio de soluciones (Duarte et al., 2007).

$$P = \begin{cases} \text{opt} : f(x) & \text{Función Objetivo} \\ x \in F \in SS & \text{Restricciones} \end{cases} \tag{2.1}$$

Los problemas de optimización pueden categorizarse de diversas maneras, por

ejemplo, por la naturaleza de la solución, que puede ser real o entera. En este trabajo, nos enfocaremos en los problemas de agrupación, los cuales se encuentran en la categoría de optimización combinatoria y cuyas soluciones se representan mediante conjuntos enteros.

Un gran número de problemas en el ámbito de la optimización combinatoria son considerados difíciles de resolver. En otras palabras, hasta el día de hoy no se ha descubierto un algoritmo que ofrezca una solución óptima en tiempo polinómico (es decir, que el tiempo de ejecución aumente en un orden polinómico con el tamaño de la entrada). Debido a esta complejidad, estos problemas son clasificados como NP-completos o NP-duros.

Los problemas NP-Completo y NP-duro, tal como los define Sipser [2006], son problemas de optimización que son al menos tan difíciles como cualquier otro problema en la clase NP. Si pudiéramos encontrar un algoritmo eficiente para resolver uno de estos problemas, podríamos resolver todos los problemas en NP de manera eficiente. Sin embargo, hasta ahora, no se ha encontrado un algoritmo que proporcione una solución óptima a estos problemas en tiempo polinómico. Según Arora y Barak [2009], el tiempo necesario para resolver estos problemas crece exponencialmente o incluso más rápido a medida que crece el tamaño del problema, lo que hace que estos problemas sean particularmente difíciles de resolver cuando son grandes.

Estos problemas, a pesar de su dificultad, tienen un gran valor en la ciencia de la computación y otras disciplinas debido a la importancia de las preguntas que plantean y a la amplia gama de aplicaciones que tienen. Por lo tanto, se han desarrollado numerosos métodos y técnicas, incluyendo algoritmos heurísticos y metaheurísticos, para encontrar soluciones "buenas" si no óptimas, en tiempos de ejecución razonables. En este trabajo, exploramos y aplicamos algunas de estas técnicas para resolver problemas de agrupación.

2.2. Problemas de agrupación

Los problemas de agrupación son un subconjunto significativo de problemas de optimización combinatoria. Estos problemas implican la organización de un conjunto de elementos en subconjuntos de acuerdo a ciertos criterios de agrupación. Se enfocan en optimizar una función de costo determinada sobre todas las posibles agrupaciones válidas de los elementos, y a menudo, vienen con ciertas restricciones que deben cumplirse para que una agrupación sea válida (Falkenauer, 1994).

Falkenauer [1994] describe a los problemas de agrupación de la siguiente manera: un conjunto U de objetos se divide en una colección de subconjuntos mutuamente disjuntos U_i de U de modo que:

$$U = \bigcup_i U_i \tag{2.2}$$

$$U_i \cap U_j = \emptyset, \quad i \neq j \tag{2.3}$$

La ecuación 2.2 dice que el conjunto U es la unión de todos los subconjuntos U_i. Es decir, si juntamos todos los subconjuntos U_i, obtenemos el conjunto U. Esta ecuación se cumple siempre que cada elemento en U pertenezca a algún subconjunto U_i.

La ecuación 2.3 afirma que cualquier par de subconjuntos diferentes U_i y U_j (donde $i \neq j$) son disjuntos. Es decir, no tienen ningún elemento en común. Esto es fundamental en la definición de un problema de agrupación: si cada elemento de U debe pertenecer a un subconjunto U_i, entonces no debe haber ningún elemento que pertenezca a más de un subconjunto.

Existen varios tipos de problemas de agrupación que varían dependiendo de las restricciones y la función de costo que se esté optimizando. Algunos de los ejemplos más conocidos de problemas de agrupación son: el problema de empaquetamiento (Bin Packing), el problema de diseño de taller (Workshop Layouting), y el problema de coloreo de grafos (Graph Coloring) (Falkenauer, 1994). La tabla 2.1 describe estos y otros problemas de agrupación.

En general, los problemas de agrupamiento presentan una gran complejidad y muchos de ellos son NP-difíciles. Esto significa que hasta la fecha, no se ha encontrado un algoritmo que pueda resolver estos problemas de manera óptima en tiempo polinómico (Ramos-Figueroa et al., 2020). Otros problemas de agrupamiento destacados son: el balanceo de carga, el enrutamiento de vehículos, el diseño de líneas de ensamble, la calendarización, la formación de equipos, entre otros (Ramos-Figueroa et al., 2020).

En el contexto de este trabajo, se exploraron los problemas de agrupación en el ámbito de la optimización combinatoria con soluciones representadas con conjuntos enteros. El objetivo fue buscar la mejor distribución posible de los elementos del conjunto en subconjuntos, mientras se satisfacían un conjunto de restricciones y se optimizaba una función objetivo determinada. Esta fue una tarea compleja, ya que involucró la exploración de un espacio de soluciones muy grande, especialmente cuando se trabajaba con grandes volúmenes de datos. A pesar de su complejidad, los problemas de agrupación fueron muy relevantes, ya que modelaron una gran cantidad de situaciones prácticas en diversas disciplinas y campos de aplicación.

Tabla 2.1: Problemas de Agrupación en la literatura

Problema	**Descripción y Referencias**
Problema de Empaquetado de Contenedores (Bin Packing)	Problema de asignar objetos a contenedores con el objetivo de minimizar el número total de contenedores utilizados Achterberg and Hoogeveen (2008).
Problema de Coloración de Grafos (Graph Coloring Problem)	Problema de asignar colores a los vértices de un grafo de manera que dos vértices adyacentes no compartan el mismo color Garey et al. (1976); Johnson (1975).
Problema de Diseño de Distribución (Workshop Layout Problem)	Problema de asignar ubicaciones a máquinas y estaciones de trabajo en un taller para minimizar los tiempos de recorrido y mejorar la eficiencia del proceso Agarwal and Ergun (2008); Pinedo (2012).
Problema de Balanceo de Carga (Load Balancing Problem)	Problema de distribuir de manera equitativa la carga de trabajo entre los recursos disponibles para mejorar la eficiencia y minimizar los tiempos de ejecución Awerbuch (1987); Baruah and Buyya (2009).
Problema de Formación de Equipos (Team Formation Problem)	Problema de asignar tareas o roles a un conjunto de agentes de manera óptima para maximizar la eficiencia del equipo o cumplir con ciertos objetivos Ahuja et al. (1993); Hall and Smith (1989).
Problema de Agrupamiento de Datos (Data Clustering Problem)	Problema de agrupar objetos o elementos similares en clusters o grupos basados en características o atributos compartidos para descubrir patrones o estructuras subyacentes en los datos Aggarwal and Reddy (2014); Kaufman and Rousseeuw (2009).
Problema de Asignación de Tareas (Task Assignment Problem)	Problema de asignar tareas a agentes o recursos de manera óptima para maximizar la eficiencia y minimizar los costos Hall and Smith (1989); Lawler (1976).

2.2.1. Problema del empaquetado de objetos en contenedores (Bin Packing, 1-D BPP)

El problema de empaquetado de contenedores de una dimensión (BPP) es un problema de optimización combinatoria que se ocupa de empacar un conjunto de objetos con diferentes tamaños en un número finito de contenedores sin exceder la capacidad máxima especificada de los contenedores. Esto debe hacerse de manera que se minimice el número total de contenedores requeridos (Achterberg and Hoogeveen, 2008). Por simplicidad, en lo sucesivo, este problema será refererido como BPP.

El modelo matemático del BPP se puede expresar de la siguiente manera:

El problema de Bin Packing (BPP) consiste en asignar cada objeto a un contenedor de tal forma que la suma de los pesos de los objetos en cada contenedor

no exceda c y el número de contenedores m utilizado sea mínimo (Martello and Toth, 1990). Es decir, se busca encontrar el menor número de subconjuntos B_j, para $1 \leq j \leq m$, de una partición del conjunto N, es decir, $\bigcup_{j=1}^{m} B_j = N$, tal que:

Dado un conjunto $N = \{1, \ldots, n\}$ de objetos a distribuir en contenedores del mismo tamaño (capacidad), sea C la capacidad de cada contenedor y w_i el peso del objeto i, tal que $0 < w_i \leq C$ para $1 \leq i \leq n$,

$$\bigcup_{j=1}^{m} B_j = N \tag{2.4}$$

se busca:

$$\min: m \tag{2.5}$$

sujeto a:

$$\sum_{i=1}^{n} w_i b_{ij} \leq C, \quad \forall j \in \{1, 2, \ldots, m\} \tag{2.6}$$

$$b_{ij} \in \{0, 1\}, \quad \forall i \in \{1, 2, \ldots, n\} \tag{2.7}$$

donde:

- m es el número de contenedores usados.
- n es el número de objetos.
- C es la capacidad del contenedor.
- b_{ij} es una variable binaria que indica si el objeto i está empacado en un contenedor j (1) o no (0).
- w_i es el peso del objeto i, tal que $0 < w_i \leq C$ para $1 \leq i \leq n$

La función objetivo 2.5 minimiza el número total de bins requeridos. Las restricciones aseguran que el peso de cada bin no exceda su capacidad. Las variables binarias aseguran que cada elemento esté empacado o no en un contenedor. En la ecuación, $\sum_{i=1}^{n} b_i$ representa la suma de las variables binarias b_i para todos los objetos i. Cada variable b_i toma el valor de 1 si el objeto i está empacado en algún contenedor, y 0 en caso contrario.

La primera restricción (2.6), $\sum_{i=1}^{n} w_i \leq C, \quad \forall j \in 1, 2, \ldots, m$, garantiza que la suma de los pesos de los objetos empacados en un contenedor j no exceda su capacidad C. Aquí, $\sum_{i=1}^{n} w_i$ representa la suma de los pesos w_i de todos los objetos empacados en un contenedor j, y C es la capacidad máxima permitida para ese contenedor. La restricción se aplica a todos los contenedores j en el rango de 1 a m.

La segunda restricción (2.7), $b_i \in 0, 1, \quad \forall i \in 1, 2, \ldots, n$, indica que las variables binarias b_i solo pueden tomar los valores 0 o 1. Esto significa que cada objeto i está asociado con una variable binaria b_i, que indica si ese objeto está empacado (valor 1) o no empacado (valor 0) en algún contenedor.

Estas restricciones aseguran que el peso total de los objetos empacados en un contenedor no exceda su capacidad, y que cada objeto esté correctamente asignado a un contenedor o no esté asignado en absoluto. El objetivo es encontrar una asignación de objetos a contenedores que minimice el número total de contenedores utilizados.

El problema de empaquetamiento de contenedores (BPP) es un problema NP-duro (Johnson and Garey, 1974), lo que significa que presenta una complejidad combinatoria considerable. En busca de una solución óptima, es necesario examinar todas las posibles formas de dividir el conjunto de n objetos en n o menos subconjuntos. Sin embargo, esta tarea se vuelve problemática a medida que aumenta el tamaño del problema, ya que el número de particiones posibles crece de manera exponencial. De hecho, la cantidad de particiones supera ampliamente $(2^n)/n$ (Basse, 1998), lo que implica que realizar una búsqueda exhaustiva de todas las combinaciones se vuelve impracticable para problemas de gran envergadura. Por consiguiente, resolver el BPP de manera óptima se convierte en un desafío y se requiere recurrir a enfoques heurísticos o algoritmos aproximados para obtener soluciones eficientes y cercanas a la óptima.

Aquí tenemos algunas heurísticas simples para el problema de empaquetado en una dimensión:

- **Primero en Ajustar (FF)**: Esta heurística simplemente empaqueta el primer objeto que encuentra en el primer contenedor que tiene suficiente espacio para él. Si el primer objeto no cabe en ninguno de los contenedores, se crea un nuevo contenedor. Esta heurística es muy sencilla de implementar, pero no siempre produce soluciones óptimas (Johnson and Garey, 1974; Martello and Toth, 1990).
- **Mejor Ajuste (BF)**: Esta heurística empaqueta el primer objeto que encuentra en el contenedor que tiene el espacio restante más grande. Esta heurística es más probable que produzca soluciones óptimas que FF, pero también es más compleja de implementar (Garey et al., 1976; Ibarra and Kim, 1975).
- **Primero en Ajustar con ñ objetos preasignados (FF-ñ)**: Esta heurística es una variación de FF que primero empaqueta ñ objetos grandes, que no se podrían combinar con otros del mismo subconjunto, en contenedores separados. Los objetos restantes se empacan luego utilizando FF. Esta heurística puede ser más efectiva que FF para problemas con un gran número de objetos grandes (Quiroz Castellanos, 2014).

Otra estrategia es el uso de un límite inferior de una instancia de BPP es un número de contenedores menor o igual que el número de contenedores necesarios

para empaquetar todos los objetos (Pérez-Ortega et al., 2016). Los límites inferiores son útiles por dos razones. En primer lugar, se pueden utilizar para demostrar que una solución es óptima. Si una solución es igual a un límite inferior, entonces debe ser óptima. En segundo lugar, se pueden utilizar límites inferiores para mejorar la calidad de las soluciones aproximadas. Si se conoce un límite inferior, se puede usar un algoritmo aproximado para encontrar una solución que sea al menos tan buena como el límite inferior. Aquí se presentan algunos algoritmos y métodos utilizados para calcular el límite inferior del problema BPP:

- **Bola y Viga (Ball and Beam)**: Este algoritmo funciona colocando cada objeto en una bola de radio igual a su tamaño. El número de contenedores requeridos es entonces igual al número de bolas que no caben en un solo contenedor(Martello and Toth, 1990).
- **Codicioso Determinístico (Deterministic Greedy)**: Este algoritmo funciona al empacar codiciosamente los objetos en los contenedores, uno a la vez. El algoritmo siempre elige el contenedor que tiene más espacio restante (Ibarra and Kim, 1975).
- **Rama y Cota Heurística (Heuristic Branch and Bound)**: Este algoritmo es un algoritmo de ramificación y cota que utiliza una heurística para estimar el límite inferior de la solución óptima (Pinedo, 2008).
- **L1-Límite (L1-Bound)**: Este límite se da por la siguiente fórmula:

  $$L1 = \lceil \frac{\sum_{i=1}^{n} w_i}{C} \rceil$$

  donde w_i es el tamaño del objeto i y C es la capacidad de un contenedor (Scholl et al., 1997).
- **Programación Lineal**: Este método se puede usar para calcular un límite inferior para el problema BPP formulando el problema como un problema de programación lineal entera (Grötschel et al., 1988).

A continuación, se proporciona una explicación más detallada del límite L1:

El límite L1 se calcula sumando primero los tamaños de todos los objetos. Esta suma se divide entonces por la capacidad de un contenedor. El resultado es el número de contenedores que se requieren para contener todos los objetos, asumiendo que los objetos se pueden fragmentar. Esta es una subestimación del número de contenedores que realmente se requieren, pero es un buen límite inferior (Quiroz Castellanos, 2014).

2.2.2. Problema de programación de máquinas paralelas

La programación de máquinas paralelas (PMS) es una familia de problemas en la que un conjunto de trabajos debe ser programado en un conjunto de máquinas. Cada trabajo tiene un tiempo de procesamiento que depende de la máquina en la que está programado. El objetivo es encontrar un programa que minimice el makespan, que es el tiempo que se tarda en completar todos los trabajos.

El problema de programación de máquinas paralelas puede formularse de la siguiente manera (Ramos-Figueroa, 2022):

$$\min C_{\max} \tag{2.8}$$

sujeto a:

$$C_{\max} = \text{máx}(C_1, C_2, C_3, ..., C_M) \tag{2.9}$$

$$\sum_{i=1}^{m} x_{ij} = 1 \quad \forall j \in N \tag{2.10}$$

$$\sum_{j=1}^{n} p_{ij} \cdot x_{ij} \leq C_{\max} \quad \forall i \in M \tag{2.11}$$

$$x_{ij} \in \{0, 1\} \quad \forall j \in N, \forall i \in M \tag{2.12}$$

donde:

- $C_{\max} = \text{máx}(C_i)$: Indica el tiempo de finalización máximo en todas las máquinas.
- C_i: Es el tiempo de finalización que cada máquina i necesita para procesar sus trabajos asignados.
- p_{ij}: Es el tiempo de procesamiento del trabajo j en lamáquina i.
- $x_{ij} = 1$ si el trabajo j está asignado a la máquina i, de lo contrario $x_{ij} = 0$.

La ecuación 2.8 representa la función objetivo del problema que es minimizar el makespan. La ecuación 2.8 define el objetivo, que es minimizar $C_{\max}$, el makespan o tiempo total de finalización. La ecuación 2.9 especifica cómo se calcula $C_{\max}$, que es el máximo de los tiempos de finalización en todas las máquinas. La ecuación 2.10 garantiza que cada trabajo se asigne a una máquina (Ramos-Figueroa, 2022).

La variante más general del problema de programación de máquinas paralelas es el problema $R||C_{\max}$, en el que las máquinas son idénticas y los trabajos pueden procesarse en cualquier orden. En este trabajo, de manera equivalente se usara PMS para referir esta variante. Este problema es NP-difícil, lo que significa que no existe un algoritmo de tiempo polinomial que pueda encontrar siempre la solución óptima (Graham et al., 1979). Sin embargo, existen una serie de heurísticas que pueden utilizarse para encontrar buenas soluciones.

Además de las heurísticas, existen una serie de algoritmos exactos que pueden utilizarse para resolver el problema $R||C_{\max}$. Los algoritmos exactos son técnicas programáticas que buscan la solución más precisa y óptima a un problema, a diferencia de las heurísticas que buscan soluciones "suficientemente buenas". Su diseño permite resolver problemas de manera exacta, aunque a veces su ejecución puede ser costosa en tiempo y recursos. Estos algoritmos se basan generalmente en la técnica de branch-and-bound o branch-and-cut. Sin embargo, estos algoritmos pueden ser muy consumidores de tiempo, y no siempre son capaces de encontrar la solución óptima (Brucker, 1984). Algunas de las heurísticas más comunes incluyen (Coello Coello et al., 2002):

- **Mínimo aleatorio (Random min, RM):** Esta heurística asigna trabajos a las máquinas de forma aleatoria, comenzando con el trabajo que tiene el tiempo de procesamiento más corto.
- **Mínimo aleatorio de límite inferior (Random lowest min, RLM):** Esta heurística es una extensión de la heurística Random min que incorpora el uso del límite inferior lb. La heurística primero calcula el límite inferior para cada trabajo. Luego, asigna trabajos a las máquinas de forma aleatoria, comenzando con el trabajo que tiene el límite inferior más bajo.
- **Primero en llegar, primero en ser atendido (FCFS)**: Los trabajos se programan en el orden en que llegan.
- **EDD:** Esta heurística es común para el problema $R||C_{\max}$ es la regla de la fecha de vencimiento más temprana (EDD). La regla EDD programa los trabajos en orden de fecha de vencimiento creciente. Esta heurística tampoco siempre es óptima, pero puede ser efectiva en situaciones en las que los trabajos tienen diferentes fechas de vencimiento (Brucker, 1984).
- **SPT:** Una de las heurísticas más comunes para el problema $R||C_{\max}$ es la regla del tiempo de procesamiento más corto (SPT). La regla SPT programa los trabajos en orden de tiempo de procesamiento creciente. Esta heurística no siempre es óptima, pero a menudo está muy cerca de serlo (Brucker, 1984).
- **LPT:** Esta heurística asigna trabajos a las máquinas en orden de tiempo de procesamiento decreciente.
- **Camino crítico (CP)**: Los trabajos se programan a lo largo del camino crítico, que es el camino más largo a través de la red de trabajos.

- **HEFT:** Esta es una heurística híbrida que combina las heurísticas EDD y LPT. La heurística primero calcula la fecha de vencimiento más temprana para cada trabajo. Luego, asigna trabajos a las máquinas en orden de fecha de vencimiento decreciente, rompiendo empates al asignar trabajos con tiempos de procesamiento más cortos primero.
- **NEH:** Esta es una heurística constructiva que comienza con un horario vacío e iterativamente agrega trabajos al horario, tratando de minimizar el makespan.
- **GRASP:** Esta es una metaheurística que combina una búsqueda greedy con una búsqueda local. La búsqueda greedy comienza con un horario vacío e iterativamente agrega trabajos al horario, tratando de minimizar el makespan. La búsqueda local luego intenta mejorar el horario intercambiando trabajos.
- **Busqueda Local Iterada (Iterated Local Search, ILS):** Esta es una metaheurística que comienza con una solución e iterativamente trata de mejorarla haciendo una secuencia de cambios locales. Los cambios locales se hacen en un orden aleatorio, y la heurística termina cuando no se pueden hacer más mejoras.

La elección de la heurística depende de las características específicas de la instancia del problema. Por ejemplo, FCFS suele ser una buena elección cuando los trabajos tienen tiempos de procesamiento similares. SPT suele ser una buena elección cuando los trabajos tienen diferentes tiempos de procesamiento y el objetivo es minimizar el makespan. LPT suele ser una buena elección cuando los trabajos tienen diferentes tiempos de procesamiento y el objetivo es maximizar la utilización de las máquinas. CP suele ser una buena elección cuando los trabajos tienen restricciones de precedencia (Hall et al., 2003).

La heurística RLM, mencionada anteriormente, destaca por su naturaleza aleatoria y mínima. Esta heurística incorpora el uso de un límite inferior lb.

$$lb = \frac{1}{m} \sum_{j=1}^{n} \text{mín}(p_{ij}) \tag{2.13}$$

Donde lb consiste en sumar los tiempos de procesamiento p_{ij} requeridos por las máquinas más rápidas para procesar cada trabajo j y dividir el resultado por el número de máquinas disponibles m como se muesta en la ecuación 2.13. En este orden de ideas, el procedimiento de mínima aleatoria del límite inferior es el siguiente. Para cada trabajo j seleccionado de forma aleatoria con una distribución uniforme, esta estrategia intenta asignar j a la máquina i que lo procesa en el menor tiempo p_{ij}, es decir, utilizando la propiedad del mínimo, considerando la siguiente regla. Si $C_i + p_{ij} \leq lb$, el trabajo j se asigna a la máquina más rápida i. De lo contrario, el trabajo j permanece sin asignar. C_i representa el tiempo que la máquina i necesita para procesar sus trabajos asignados. Finalmente, la heurística de mínima aleatoria del límite inferior asigna los trabajos no asignados.

El problema de programación de máquinas paralelas es un problema desafiante, pero es un problema importante en una variedad de aplicaciones. Por ejemplo, el problema puede utilizarse para programar trabajos en una planta de fabricación, para programar tareas en un sistema informático, o para programar vuelos en una aerolínea (Pinedo, 2008).

2.3. Heurísticas y metaheurísticas

Los algoritmos heurísticos utilizan estrategias llamadas heurísticas, para obtener soluciones de calidad (no necesariamente óptimas) a problemas complejos de manera eficiente. Las heurísticas son procedimientos basados en el sentido común, que ofrecen una buena solución a problemas (particulares) difíciles, de un modo fácil y rápido (Díaz et al., 1996).

Por otro lado, un algoritmo metaheurístico es un algoritmo heurístico no determinista de propósito general, que combina diferentes heurísticas definidas de acuerdo al problema. Estos algoritmos parten de soluciones iniciales y, mediante alteraciones inteligentes, exploran soluciones vecinas de su entorno en búsqueda de mejores soluciones (Michalewicz and Fogel, 2004; Dréo et al., 2006).

Algunas de las metaheurísticas más comunes que se pueden utilizar para resolver problemas de empaquetado de contenedores incluyen: Algoritmos genéticos, recocido simulado y búsqueda Tabú.

Las heurísticas y metaheurísticas pueden utilizarse para resolver tanto el problema de empaquetado de contenedores (BPP) como el problema de programación de máquinas paralelas (PMS), que son problemas de optimización combinatoria que se encuentran en la intersección de la informática y la investigación de operaciones. En estos problemas, el objetivo es minimizar la cantidad de contenedores o máquinas utilizadas para empaquetar o programar un conjunto de artículos o tareas. Las heurísticas y metaheurísticas se pueden usar para encontrar buenas soluciones a ambos problemas, pero no garantizan encontrar soluciones óptimas.

2.4. Metaheurísticas evolutivas

Las metaheurísticas evolutivas son una clase de algoritmos de optimización inspirados en el proceso de evolución biológica. Trabajan generando iterativamente nuevas soluciones, llamadas individuos, y seleccionando los mejores individuos para continuar el proceso. Los mejores individuos se utilizan para generar nuevos individuos y el proceso se repite hasta que se encuentra una solución satisfactoria (Coello Coello et al., 2007).

Las metaheurísticas evolutivas son una herramienta poderosa para resolver una

amplia variedad de problemas de optimización. Son particularmente adecuados para problemas que son difíciles de resolver con técnicas de optimización tradicionales, como problemas con un gran número de variables o problemas con funciones objetivas no convexas (Blum and Roli, 2003).

Algunas de las metaheurísticas evolutivas más comunes incluyen:

- **Algoritmos genéticos:** Los algoritmos genéticos se inspiran en el proceso de selección natural. Trabajan generando iterativamente nuevas soluciones, llamadas cromosomas, y seleccionando los mejores cromosomas para continuar el proceso. Los mejores cromosomas se utilizan para generar nuevos cromosomas y el proceso se repite hasta que se encuentra una solución satisfactoria (Coello Coello et al., 2007).
- **Optimización por Enjambre de Partículas (PSO):** Se basa en el comportamiento de un enjambre de partículas. Cada partícula representa una solución y se mueve en el espacio de búsqueda utilizando información de su experiencia pasada y de las partículas vecinas (Talbi, 2009).

Las metaheurísticas evolutivas se han aplicado con éxito a una amplia variedad de problemas de optimización, incluyendo:

- **Problemas de programación:** Los problemas de programación implican encontrar un horario que minimice el costo o maximice el beneficio de un conjunto de tareas. Las metaheurísticas evolutivas se han aplicado con éxito a una variedad de problemas de programación, como la programación de talleres y la programación de flujo de trabajo.
- **Problemas de rutas de vehículos:** Los problemas de rutas de vehículos implican encontrar la ruta más corta para una flota de vehículos para visitar un conjunto de clientes. Las metaheurísticas evolutivas se han aplicado con éxito a una variedad de problemas de rutas de vehículos, como el problema del vendedor viajero y el problema de dial-a-ride.
- **Problemas de aprendizaje automático:** Los problemas de aprendizaje automático implican encontrar un modelo que pueda predecir la salida de un sistema dado un conjunto de entradas. Las metaheurísticas evolutivas se han aplicado con éxito a una variedad de problemas de aprendizaje automático, como la clasificación y la regresión.

Las metaheurísticas evolutivas son una herramienta poderosa para resolver una amplia variedad de problemas de optimización. Son especialmente adecuadas para problemas que son difíciles de resolver utilizando técnicas de optimización tradicionales (Liang and Suganthan, 2009).

2.5. Algoritmos Coevolutivos

Los algoritmos coevolutivos son un tipo de algoritmo evolutivo que utiliza múltiples poblaciones de individuos para resolver un problema (Goldberg, 1993). Cada población representa un aspecto diferente del problema, y los individuos en cada población se evalúan en función de su capacidad para interactuar con los individuos en otras poblaciones. Este enfoque puede utilizarse para resolver problemas que son difíciles o imposibles de resolver con una sola población de individuos.

Los algoritmos coevolutivos se pueden dividir en dos tipos principales: coevolución cooperativa y coevolución competitiva(Simões and Costa, 2012):

- **En la coevolución cooperativa**, los individuos de cada población trabajan juntos para resolver el problema. Las poblaciones colaboran para resolver problemas. El enfoque tradicional asume que los problemas son separables, y en cada iteración ensamblan subcomponentes de especies para obtener una solución global (Talbi, 2009).
- **En la coevolución competitiva**, las poblaciones compiten entre sí para resolver el problema. Por ejemplo, el modelo depredador-presa se basa en manadas que se agrupan para encontrar lugares seguros y de calidad. El éxito de una población es a expensas de otra población en términos de aptitud y restricciones (Talbi, 2009).

Los algoritmos coevolutivos han sido utilizados para resolver una variedad de problemas, incluyendo juegos, programación y optimización (Goldberg, 1989; Holland, 1991). Se ha demostrado que son efectivos en la resolución de problemas que son difíciles o imposibles de resolver con otros tipos de algoritmos evolutivos (Rosin and Belew, 2011).

Además, los algoritmos coevolutivos tienen varias ventajas. Por ejemplo, pueden resolver problemas que son difíciles o imposibles de resolver con una sola población de individuos. También pueden utilizarse para resolver problemas que son dinámicos o cambiantes, y pueden utilizarse para resolver problemas que son multiobjetivo (Goldberg, 1989; Deb, 2002).

No obstante, estos algoritmos también presentan desventajas. Por ejemplo, la coevolución competitiva puede ser más eficiente que la coevolución cooperativa, ya que sólo necesita concentrarse en encontrar el mejor individuo de cada población. Sin embargo, también puede ser más difícil encontrar una buena solución, ya que los individuos de cada población están constantemente tratando de superarse mutuamente (Holland, 1991). A pesar de estos desafíos, recientes avances en los algoritmos coevolutivos han permitido superar algunas de estas dificultades, mejorando su eficiencia y eficacia (Fogel, 2006; Mahdavi et al., 2015; Ramos-Figueroa, 2022).

2.6. Control Auto-Adaptativo de Parámetros

En el contexto de los algoritmos, la auto-adaptación se refiere a la capacidad de un algoritmo para ajustar automáticamente sus parámetros con el fin de mejorar su rendimiento. Esto contrasta con los algoritmos tradicionales, que requieren que el usuario ajuste manualmente los parámetros para alcanzar un rendimiento óptimo (Rechenberg, 1994; Greferstette, 1995; Marques-Silva and Cruz, 2001).

Existen diferentes formas de implementar la auto-adaptación. Un enfoque común es usar un ciclo de retroalimentación. En este enfoque, el algoritmo mide su propio rendimiento y utiliza esta información para ajustar sus parámetros (Rechenberg, 1994). Otro enfoque de auto-adaptación es utilizar un algoritmo genético, en el cual se seleccionan y se cruzan los individuos más aptos de una población, heredando los genes de sus progenitores, y el algoritmo repite este proceso hasta encontrar una solución suficientemente buena (Yang, 2009; Marques-Silva and Cruz, 2005).

Los algoritmos auto-adaptativos pueden ser muy efectivos en una variedad de configuraciones. Pueden usarse para resolver problemas que son difíciles de solucionar con algoritmos tradicionales y pueden usarse para mejorar el rendimiento de algoritmos tradicionales.

Existen diferentes tipos de auto-adaptación. Un tipo de autoadaptación es la **adaptación de parámetros**, donde el algoritmo ajusta automáticamente sus parámetros para mejorar su rendimiento. Otro tipo de auto-adaptación es la **adaptación de estructura**, donde el algoritmo cambia automáticamente su estructura para mejorar su rendimiento (Marques-Silva and Cruz, 2005, 2013).

Aunque la auto-adaptación puede ser una herramienta poderosa para mejorar el rendimiento de los algoritmos, también puede hacer que los algoritmos sean más complejos y difíciles de entender (Marques-Silva and Cruz, 2001).

Existen diferentes tipos de adaptividad de parámetros:

- **Estático:** Los parámetros son fijos y no cambian durante la ejecución del algoritmo.
- **Adaptativo:** Los parámetros se ajustan durante la ejecución del algoritmo, basándose en el comportamiento observado del capturado generalmente en reglas predefinidas.
- **Auto-adaptativo:** Los parámetros son ajustados por el propio algoritmo, sin ninguna intervención del usuario.

Los algoritmos auto-adaptativos son más complejos que los algoritmos estáticos o adaptativos, pero a menudo pueden alcanzar un mejor rendimiento (Kramer, 2008).

2.7. Generalización de Algoritmos

Los algoritmos pueden generalizarse para trabajar con una variedad de problemas utilizando varias técnicas. Una técnica común es utilizar una representación independiente del problema. Esto significa que el algoritmo no necesita conocer los detalles específicos del problema para resolverlo. En lugar de eso, el algoritmo trabaja con una representación genérica del problema, lo que hace al algoritmo más general y fácil de aplicar a nuevos problemas (Russell and Norvig, 2020).

Otra técnica común para generalizar algoritmos es el uso de heurísticas específicas del problema. Las heurísticas son reglas generales que se pueden utilizar para mejorar el rendimiento de un algoritmo. A menudo, las heurísticas son específicas del problema, lo que significa que son efectivas solo para un tipo particular de problema. Sin embargo, las heurísticas se pueden usar para mejorar el rendimiento de los algoritmos para una variedad de problemas (Talbi, 2009).

El rendimiento de los algoritmos se puede medir de varias formas. Una medida común es el tiempo hasta la solución. Esto es la cantidad de tiempo que lleva a un algoritmo encontrar una solución a un problema. Otra medida común es la calidad de la solución, que es la medida de cuán cerca está la solución a la solución óptima (Sutton and Barto, 2018). También existe el desafío de evaluar la generalidad, debido a la dificultad de obtener un conjunto lo suficientemente grande de instancias de problemas para entrenar y probar un algoritmo (Talbi, 2009).

2.7.1. Evaluación de la generalidad

En la literatura especializada, solo se han identificado trabajos que analizan la generalidad en el contexto de los hiperheurísticos. Para fines prácticos, en esta tesis se considera que los mismos principios también se pueden aplicar a los algoritmos coevolutivos, ya que debido a su diseño, tienen el potencial de aumentar su generalidad.

En su trabajo, Pillay y Qu proponen una métrica para evaluar la generalidad de los hiperheurísticos, que definen como la capacidad de un hiperheurístico para desempeñarse bien en una amplia variedad de instancias de problemas. Proponen la desviación estándar de las diferencias (SDD, por sus siglas en inglés) como medida de generalidad. La SDD se calcula de la siguiente manera (Pillay and Qu, 2021):

$$SDD = \sqrt{\frac{\sum (f(x) - f(y))^2}{n}} \tag{2.14}$$

Donde $f(x)$ es el valor objetivo de una solución a la instancia del problema x, $f(y)$ es el valor objetivo de una solución a la instancia del problema y, y n es el número de instancias del problema.

La SDD observada en la ecuación 2.14 es una medida de cuánto varían los valores objetivos de las soluciones a diferentes instancias de problemas. Una SDD baja indica que los valores objetivos de las soluciones a diferentes instancias de problemas son similares, lo que significa que el hiperheurístico puede encontrar buenas soluciones para una amplia variedad de instancias de problemas (Pillay and Qu, 2021).

2.7.2. Integración de la medida de generalidad con otros criterios de evaluación

Generalmente se requiere evaluar el desempeño algorítmico en términos de otros criterios además de la generalidad (SDD). Por ejemplo, puede ser de interés evaluar un conjunto de algoritmos en los siguientes términos:

- calidad de la optimización y
- tiempo de ejecución.

Pillay and Qu (2021) sugieren un enfoque sencillo basado en tomar la suma ponderada de los diferentes criterios. Sin embargo, debido a que los valores de los indicadores establecidos para cada criterio pudieran estar en diferentes dominios, es necesario que estos valores esten normalizados en el intervalo de 0 a 1.

Capítulo 3

Estado del Arte

Esta sección proporciona una visión detallada de los desarrollos más significativos y relevantes en el campo, presentando un panorama completo del estado del arte actual. En particular, se enfoca en el análisis de las contribuciones de Quiroz Castellanos (2014) y González San Martín (2021) a la agrupación genética y la adaptabilidad de los parámetros, respectivamente, estableciendo así un contexto para el trabajo de investigación propuesto. Además, se exploran los recientes avances en la resolución del Bin Packing Problem (BPP) y Parallel Machine Scheduling (PMS), dos problemas de agrupación críticos. Este análisis exhaustivo permite identificar brechas y oportunidades, proporcionando un punto de partida sólido para la propuesta y desarrollo de un enfoque coevolutivo auto-adaptativo para problemas de agrupación generalizados.

3.1. Comparación de estrategias de algoritmos para problemas de agrupación

Se presenta un análisis comparativo detallado de las diversas estrategias de algoritmos aplicadas a los problemas de agrupación. Este análisis se basa en una serie de criterios clave, como el tipo de algoritmo, el número de objetivos, las restricciones y la estrategia principal utilizada. Tambein permite un entendimiento más profundo de las fortalezas y debilidades de las diferentes estrategias algorítmicas y da base sólida para el desarrollo y mejora de nuevas estrategias algorítmicas en este campo.

3.1.1. Problema del empaquetado de objetos en contenedores

En los últimos años, ha habido avances significativos en el ámbito de los porblemas de agrupación y empaquetado de objetos en contenedores (BPP). Varias

estrategias de algoritmos han sido propuestas, explorando desde enfoques de aprendizaje profundo hasta heurísticas y algoritmos genéticos.

La tabla 3.1 presenta una comparación de trabajos de la literatura en términos del problema y algoritmo de solución. Estos algoritmos se describen enseguida.

Mohiuddin et al. (2019) propusieron un algoritmo seguro distribuido adaptativo para el empaquetado en el almacenamiento en la nube. Este algoritmo está diseñado para ser seguro contra los proveedores de la nube malintencionados y es adaptativo, lo que significa que se puede usar para empacar una amplia variedad de tamaños y formas de datos.

Laterre et al. (2019) propusieron un nuevo método para entrenar agentes de aprendizaje por refuerzo para el empaquetado de objetos en contenedores. El método, llamado Ranked Reward", utiliza una función de recompensa clasificada para entrenar a los agentes para empacar objetos en contenedores de una manera óptima.

Kundu et al. (2019) propusieron un nuevo algoritmo de aprendizaje profundo por refuerzo para el empaquetado de objetos en contenedores. El algoritmo, llamado Deep-Pack, usa un enfoque basado en la visión para aprender a empacar objetos en contenedores de manera óptima.

Gupta et al. (2020) propusieron un nuevo enfoque heurístico para el empaquetado de objetos en contenedores con objetos superpuestos. El enfoque se basa en una novedosa descomposición del problema en subproblemas más pequeños.

Borgulya (2021) presenta un algoritmo evolutivo híbrido que utiliza soluciones factibles como individuos y contiene descripciones de los contenedores (bins). Este algoritmo híbrido implementa dos nuevos operadores de mutación y procedimientos de búsqueda local para mejorar la calidad de las soluciones. Además, se realizó un análisis de sensibilidad para evaluar el impacto de los diferentes parámetros en el rendimiento del algoritmo, y se encontró que el algoritmo es robusto frente a los cambios en los valores de los parámetros.

Grouping Genetic Algorithm with Controlled Gene Transmission: El algoritmo Grouping Genetic Algorithm with Controlled Gene Transmission (GGA-CGT) es una metaheurística de empacado inteligente que fue desarrollada por **Quiroz Castellanos (2014)**. Se basa en el uso de un procedimiento de reacomodo que permite la exploración y explotación del espacio de búsqueda, junto con un conjunto de operadores genéticos de agrupación que promueven la transmisión controlada de los mejores genes en los cromosomas sin perder el equilibrio entre la presión selectiva y la diversidad de la población.

El algoritmo GGA-CGT comienza generando una población inicial de individuos a través de la heurística de empacado FF-ñ. Esta heurística es una mejora de la heurística First Fit con ñ objetos asignados, donde el valor de ñ representa el número total de objetos grandes (pesos superiores al 50 % de la capacidad del contenedor)

que no pueden combinarse con otros del mismo subconjunto.

Una vez generada la población inicial, el algoritmo pasa a una fase de selección, cruzamiento y mutación. Durante la fase de selección, se seleccionan algunos individuos de la población para ser recombinados y mutados. La selección se realiza utilizando una estrategia de selección controlada, que se asegura de que los mejores individuos de la población tengan una mayor probabilidad de ser seleccionados.

La fase de cruzamiento se realiza utilizando un operador de cruzamiento genético. El operador de cruzamiento toma dos individuos de la po-blación e intercambia sus genes. Esto tiene como objetivo crear nuevos individuos que hereden las características de los individuos mejores.

La fase de mutación se realiza utilizando un operador de mutación genético. El operador de mutación toma un individuo de la población y cambia aleatoriamente uno de sus genes. Esto tiene como objetivo crear nuevos individuos que sean diferentes de los individuos existentes.

El proceso de selección, cruzamiento y mutación se repite durante un número determinado de generaciones. Al final del proceso, el algoritmo guarda el individuo con la mejor aptitud.

En GGA-CGT, la "transmisión genética controlada"se refiere a un mecanismo específico que controla cómo se transmiten los genes entre las generaciones. Los operadores genéticos son regulados por estrategias controladas. Entre las estrategias controladas se incluyen reglas especiales para la selección de cromosomas, la forma en que los genes se transmiten durante el cruce y reglas para aplicar las mutaciones.

El algoritmo GGA-CGT ha demostrado ser efectivo en el problema de empacado de objetos. Se ha demostrado que supera a otros algoritmos heurísticos y metaheurísticos en una variedad de instancias de prueba.

En 2021, **González San Martín** propuso una nueva variante de GGA-CGT, denominada GGA-CGT/D, para mejorar el rendimiento en la solución del problema del empaquetado de objetos unidimensionales (1D-BPP). Esta variante incorpora tres estrategias principales:

- Un método para seleccionar el límite inferior más adecuado para cada problema específico, lo que permite al algoritmo enfocarse en soluciones potencialmente factibles desde el principio.
- Un método para reducir el tamaño del problema, simplificando así el espacio de soluciones y permitiendo un cálculo más eficiente.
- Un método para diversificar las soluciones, evitando así el estancamiento en soluciones locales subóptimas y explorando un espectro más amplio de soluciones posibles.

Algoritmo 1: GGA-CGT

Data: $\mathcal{N}$ (Número de objetos), $\mathcal{C}$ (capacidad del contenedor), $\vec{w}_i$ (vector de pesos de los objetos), $\mathcal{P}$ (tamaño de la población), parámetros: max_gen (máximo de generaciones), n_c (porcentaje de cruza), n_m (porcentaje de mutación), k_1 (tasa de cambio para soluciones no clonadas), k_2 (tasa de cambio para soluciones clonadas), $|\mathcal{B}|$ (tamaño del grupo elite), life_span (tiempo de vida del individuo).

Result: Mejor solución global

```
Inicializar parámetros de GGA-CGT;
Generar una población inicial con FF-ñ;
while generación < max_gen ∧ Tamaño(mejor_solución) > L2 do
    Seleccionar n_c individuos para cruzar por medio de
     Selección_Controlada();
    Aplicar cruzamiento a los n_c individuos seleccionados;
    Aplicar Reemplazo_Controlado() para introducir las crías;
    Seleccionar n_m individuos y clonar soluciones élite por
     Selección_Controlada();
    Aplicar mutación a los mejores n_m individuos;
    Aplicar Reemplazo_Controlado() para introducir los clones;
    Actualizar la mejor solución;
end
```

Los resultados presentados por González San Martín et al. muestran que GGA-CGT/D supera en desempeño a los algoritmos del estado del arte en una serie de pruebas. De hecho, este algoritmo ha sido capaz de resolver óptimamente todas las instancias seleccionadas de BPPLIB, así como un número considerable de instancias de BPPv_u_c (Delorme et al., 2018).

Tabla 3.1: Comparación de diferentes algoritmos para el problema de empaquetamiento de contedores

Trabajo	Tipo	Obj.	Rest	Estrategia Principal
Mohiuddin et al. (2019)	Mono-objetivo	1	2	Algoritmo Adaptativo Distribuido
Laterre et al. (2019)	Mono-objetivo	1	0	Aprendizaje por Refuerzo
Kundu et al. (2019)	Mono-objetivo	1	2	Aprendizaje Profundo por Refuerzo
Gupta et al. (2020)	Mono-objetivo	1	0	Descomposición Heurística
Quiroz Castellanos (2014)	Mono-objetivo	1	2	GGA-CGT
González San Martín (2021)	Mono-objetivo	1	2	GGA-CGT/D

3.1.2. Problema de programación de máquinas paralelas

El problema de programación de máquinas paralelas no relacionadas con el objetivo de minimizar el makespan ha sido abordado en varios trabajos desde diversas perspectivas y metodologías.

Ghirardi and Potts (2005) propusieron una implementación de la búsqueda tabú para resolver el problema de programación de máquinas paralelas no relacionadas con tiempos de configuración dependientes de la secuencia y la máquina. Presentaron una nueva estrategia de búsqueda, llamada Recovering Beam Search, que supera las limitaciones de la Búsqueda Beam tradicional y permite la recuperación y corrección de decisiones erróneas tomadas durante el proceso de búsqueda.

Posteriormente, **Helal et al. (2006)** propusieron una implementación de búsqueda tabú para resolver el problema de programación de máquinas paralelas no relacionadas con tiempos de configuración dependientes de la secuencia y de la máquina. Su algoritmo utiliza dos fases de esquemas de perturbación, intra-máquina e inter-máquina, y los resultados muestran que generalmente supera a otros heurísticos existentes para problemas de tamaño pequeño y grande.

Vallada and Ruiz (2011) propusieron un algoritmo genético para el problema de programación de máquinas paralelas no relacionadas con tiempos de configuración dependientes de la secuencia de máquinas y trabajos. Su enfoque incluye una búsqueda local rápida y un operador de cruce mejorado por búsqueda local. Después de calibraciones extensas utilizando el enfoque de diseño de experimentos (DOE), obtuvieron dos versiones del algoritmo. Sus resultados muestran un excelente rendimiento en un conjunto completo de instancias de referencia, superando a otros métodos evaluados.

Ghirardi and Potts (2013) presentaron un algoritmo de búsqueda por haces de recuperación para resolver el problema de minimización del makespan en sistemas de máquinas paralelas no relacionadas. El algoritmo incorpora un mecanismo de recuperación que mejora la calidad de las soluciones encontradas por el algoritmo de búsqueda por haces.

Lee and Jang (2019) abordaron el problema desde la perspectiva de la programación de máquinas paralelas uniformes con máquinas dedicadas, división de trabajos y recursos de configuración limitados. Desarrollaron un algoritmo heurístico constructivo que proporcionó soluciones de buena calidad para problemas de gran tamaño, lo que sugiere la eficiencia de este enfoque en escenarios de la vida real.

Ramos-Figueroa et al. (2022) propusieron una mejora significativa a través de la aplicación del Algoritmo Genético de Agrupación (GGA) en sus diferentes versiones (EGGA y FGGA) para el problema R||Cmax. Sus contribuciones resaltaron la importancia de comprender a fondo las propiedades del problema para crear soluciones más eficientes y efectivas. Esta mejora significativa sugiere un enfoque prometedor para la resolución de problemas complejos de agrupación.

Kaid et al. (2022) propusieron un modelo de programación lineal entera mixta (MILP) para abordar el problema de programación de máquinas paralelas no relacionadas con recursos no confiables, además de introducir dos metaheurísticas, la búsqueda tabú y el recocido simulado, para resolver problemas de gran escala. Este enfoque ha demostrado ser efectivo y eficiente, especialmente para instancias de problemas más grandes.

La tabla 3.2 presenta los algoritmos antes descritos, que, diversas estrategias han sido desarrolladas para resolver el problema de programación de máquinas paralelas no relacionadas con el objetivo de minimizar el makespan. Estos enfoques han demostrado ser efectivos y eficientes, lo que sugiere que se está progresando en la resolución de este problema NP-hard.

Tabla 3.2: Estrategias para el problema de programación de máquinas paralelas no relacionadas.

Estudio	**Estrategia**
Ghirardi and Potts (2005)	Beam Search
Helal et al. (2006)	Búsqueda Tabú
Vallada and Ruiz (2011)	Algoritmo Genético
Ghirardi and Potts (2013)	Beam Search
Lee and Jang (2019)	Planificación con Máquinas Dedicadas
Ramos-Figueroa et al. (2022)	Algoritmo Genético de Agrupación (GGA) y Heurísticas Eficaces
Kaid et al. (2022)	Programación Lineal Entera Mixta y Metaheurísticas

3.2. Revisión de trabajos que abordan varios problemas de agrupación

En el campo de la optimización combinatoria, existen múltiples problemas que incorporan una fase de agrupación. Un estudio interesante es "A particle swarm optimizer for grouping problems" de Husseinzadeh Kashan et al. (2013), donde adaptaron la estructura del algoritmo de optimización de enjambre de partículas (PSO) para resolver problemas de agrupación, creando así una versión de agrupación del algoritmo PSO, que denominaron algoritmo GPSO.

Otra contribución importante en este campo es la tesis doctoral "Efficient local search for several combinatorial optimization problems" de Buljubasic (2015), que se centra en la implementación de algoritmos basados en búsqueda local para tres problemas diferentes en el campo de la optimización combinatoria, incluyendo el problema de empaque en una dimensión, el problema de reasignación de máquinas y el problema del material rodante.

Por otro lado, en el estudio "Ant colony optimization and local search for bin

packing and cutting stock problems" de Levine y Ducatelle (Levine and Ducatelle, 2004), se empleó un enfoque de optimización de colonias de hormigas (ACO) para resolver tanto el problema de empaque como el problema de corte de stock. Se demostró que su enfoque híbrido ACO, mejorado con un algoritmo de búsqueda local simple pero efectivo, puede superar los mejores métodos de solución evolutivos híbridos conocidos para ciertas clases de problemas.

Finalmente, en el trabajo "A grouping hyper-heuristic framework based on linear linkage encoding for graph coloring" de Elhag and Ozcan (2013), se propuso un marco de hiperheurística de agrupamiento basado en codificación de enlace lineal para problemas de coloreado de grafos. Este marco proporciona la implementación de un conjunto fijo de heurísticas de bajo nivel que pueden funcionar en todos los problemas de agrupamiento en los que se busca un compromiso entre un objetivo dado y el número de grupos. La tabla 3.3 presenta un cuadro comparativo de estos trabajos.

Tabla 3.3: Comparación de diferentes algoritmos de solución de múltiples de problemas de agrupación

Referencia	**Algoritmo**	**Estrategia Principal**	**Problema que aborda**	**Estrategia**	**Intensifica y/o diversifica**
Husseinzadeh Kashan et al., 2013	GPSO	Particle Swarm Optimizer	Single batch machine-scheduling problem and Bin packing problem	Group-based particle position and velocity updating equations	-
Buljubasic, 2015	Local Search	GRASP and Tabu Search	One-dimensional Bin Packing, Machine Reassignment Problem, Rolling Stock Problem	Local Search-based heuristic algorithm	Si
Levine and Ducatelle, 2004	ACO	Ant Colony Optimization	Bin Packing and Cutting Stock Problems	Pure ACO approach and ACO augmented with a local search algorithm	Si
Elhag and Özcan, 2013	Grouping Hyper-heuristic	Linear Linkage Encoding	Graph Coloring	Fixed set of low level heuristics	-

3.3. Estudio de trabajos que usan coevolución en problemas de agrupación

El trabajo de Bożejko et al. (2015) se enfoca en el problema de empaquetamiento tridimensional (3D-BPP), donde un volumen máximo se coloca en un único contenedor. Para resolver el problema mencionado, se utilizó un algoritmo coevolutivo paralelo basado en la evolución separada de subpoblaciones cooperantes de posibles soluciones. Este enfoque permite una búsqueda más eficiente de las soluciones óptimas al explorar múltiples soluciones potenciales simultáneamente. Los experimentos computacionales realizados en este trabajo apuntan a examinar el impacto de la paralelización en el tiempo de cómputo y la calidad de las soluciones obtenidas. El problema 3D-BPP es un problema industrial importante con aplicaciones prácticas en la reducción de costos asociados con el almacenamiento y transporte de mercancías. Sin embargo, es también un difícil problema de optimización que pertenece a la clase de problemas NP-hard.

En el trabajo de Wang et al. (2021) se investiga el problema de programar un conjunto de trabajos en máquinas de procesamiento en lote paralelas con diferentes capacidades y potencias de procesamiento no idénticas para minimizar el makespan y el consumo total de energía. Para abordar este problema de optimización bi-objetivo, se propone un algoritmo coevolutivo de tres poblaciones, basado en búsquedas de exploración y coordinación entre las tres colonias. Aunque el problema estudiado en este documento se abstrae de la fabricación de semiconductores, existen muchos escenarios similares en otros entornos de producción reales, como el computo en la nube, el transporte portuario, entre otros. En consecuencia se pueden abstraer algunos otros modelos de problemas relacionados y el algoritmo propuesto en este documento se puede aplicar para abordar estos problemas relacionados. Este algoritmo propuesto es cooperativo, ya que utiliza un enfoque co-evolutivo que optimiza tanto la programación de trabajos como la asignación de máquinas simultáneamente. El algoritmo también utiliza una matriz de feromonas para compartir información importante entre las poblaciones, lo que ayuda a mejorar el rendimiento general del algoritmo.

3.4. Comentarios finales

Para dar continuidad a los trabajos del grupo de investigación al que se pertenece, dos problemas de agrupación son abordados en esta tesis: *BPP* y *PMS*. La revisión de la literatura ha mostrado lo siguiente:

1. A pesar del trabajo existente, los problemas *BPP* y *PMS* siguen vigente por los retos que presentan y su aplicabilidad (ver sección 3.1).

2. Los mejores algoritmos del estado del arte para *BPP* de una dimensión son los

algoritmos *GGA_CGT* y *GGA- CGT/D*. Estos resultados están reportados en Quiroz Castellanos (2014) y González San Martín (2021). Dado que se busca desarrollar un algoritmo con mayor nivel de generalidad, se seleccionó el *GGA_CGT* por ser menos especializado que *GGA- CGT/D*.

3. El mejor algoritmo del estado del arte para *PMS*, en su variante *R||Cmax*, es el algoritmo *FGGA* Ramos-Figueroa (2022). Debido a que *GGA_CGT* es un producto del grupo de trabajo, se decidió incorporar estrategias de *FGGA* en *GGA_CGT* para incrementar su nivel de generalidad.

4. Hasta nuestro conocimiento, existe un número pequeño de trabajos reportados en la literatura especializada que resuelvan más de un problema de agrupación con el mismo algoritmo (ver Sección 3.2), a lo más se han resuelto tres. Ninguno de estos abordan los dos problemas de interés.

5. Hasta nuestro conocimiento, existe un número pequeño de trabajos reportados en la literatura especializada que utilizan coevolución en problemas de agrupación; la Sección 3.3 presenta una pequeña muestra.

Capítulo 4

Propuesta de Solución: Un enfoque generalizado basado en coevolución auto-adaptativa para dos problemas de Optimización

En este capitulo se presenta en detalle una propuesta basada en coevolución y auto-adptación para abordar los problemas de agrupación bajo estudio. Primero se da una descripcion de las adaptaciones realizadas tanto en las instancias de los problemas como en el algoritmo genético de agrupación con transmisión de gen controlada (GGA-CGT) y en una busqueda local iterada (ILS). Posteriormente se presenta la propuesta del algoritmo coevolutivo de agrupación con cooperación auto-adaptativa (GCA-AC). El algoritmo propuessto incorpora como especies a dos versiones de GCA-AC y una ILS para abordar eficientemente los problemas de Bin Packing (BPP) y Parallel Machine Scheduling (PMS). El capítulo también aborda la implementación del algoritmo propuesto y se detalla el proceso de configuración y adaptación de sus parámetros para lograr una solución generalizada y adaptable. Con un enfoque riguroso y exhaustivo, este capítulo sienta las bases para la implementación y evaluación de la propuesta de solución presentada en esta tesis.

4.1. Modelo Unificado de Problemas de Agrupación

Para facilitar la solución de más de un problema de agrupación por un mismo algoritmo, se propone abordar un modelo unificado de los problemas de Bin Packing (BPP) y Programación de Máquinas Paralelas (PMS). Dado que el algoritmo propuesto debe trabajar con ambos problemas de una forma auto-adaptativa y sin intervención del usuario, entonces las instancias de ambos problemas deberán cumplir con un formato similar.

El formato propuesto permite modelar las instancias de BPP y PMS que pertenecen a la familia de problemas de agrupación, por lo cual comparten características. De igual forma, este formato es un punto de partida para la integración de otros problemas de agrupación en futuros trabajos.

- Conjuntos: Elementos $I = \{i, |, 1 \leq i \leq n\}$, Grupos $J = \{j, |, 1 \leq j \leq m\}$.
- Número de elementos (n): Representa la cantidad de objetos en el BPP-1D y la cantidad de trabajos en el PMS.
- Número de grupos (m): Equivale al número de contenedores en el BPP-1D y al número de máquinas paralelas en el PMS. Para el BPP-1D, se establece igual al número de objetos ($m = n$). Para el PMS es el numero de maquinas (m).
- Capacidad de los grupos (C): En el caso del BPP-1D, es la capacidad máxima que puede soportar cada contenedor. Para el PMS, se establece igual al limite lb de la heuristica $Mnimo\ aleatorio\ del\ lmite\ inferior$.
- Atributo tamaño de los elementos (a_i): En el BPP-1D, corresponde al peso de cada objeto i. En el PMS, representa el tiempo de trabajo requerido para completar cada tarea i cuyo tiempo de procesamiento varía en cada máquina j.
- Asignación de elementos (x_{ij}): Es una variable binaria que indica si el elemento i está asignado al grupo j.

El nuevo modelo unificado utiliza estos parámetros para representar ambos problemas de agrupación de manera efectiva y simplificar el algoritmo propuesto. Al adaptar estos parámetros según las características específicas de cada problema, el modelo unificado puede abordar eficientemente tanto el BPP-1D como el PMS.

Tabla 4.1: Comparación de las características principales del BPP, el Modelo Unificado y el PMS

BPP	**Modelo Unificado**	**PMS**
Objetos (n)	Elementos (n)	Trabajos (n)
Contenedores (m)	Grupos (m)	Máquinas Paralelas (m)
Capacidad de los contenedores (C)	Capacidad de los grupos (C)	Limite de la heurística (C)
Tamaño de los objetos (a_i)	Atributo de los elementos (a_i)	Tiempo de trabajo requerido para cada tarea (a_i)
Asignación de objetos a contenedores (x_{ij})	Asignación de elementos a grupos (x_{ij})	Asignación de trabajos a máquinas (x_{ij})

4.1.1. Conversíon de instancias

Para este trabajo, se utilizan instancias de prueba tanto del problema de Bin Packing (BPP) como del problema de Programación de Máquinas Paralelas (PMS). Estas instancias son transformadas al modelo unificado descrito anteriormente, con el objetivo de que puedan ser resueltas de manera eficiente por el algoritmo propuesto.

Las instancias de prueba fueron seleccionadas de fuentes de referencia establecidas en la literatura académica, las cuales ofrecen una variedad de instancias desafiantes y representativas de los problemas de agrupación considerados. Se buscó incluir instancias de diferentes tamaños y niveles de complejidad para evaluar y comparar el desempeño del algoritmo propuesto en diferentes escenarios.

Para garantizar la consistencia y la compatibilidad con el algoritmo propuesto se convirtieron, las instancias de ambos problemas usando un formato derivado del modelo unificado. En este formato, cada instancia esta representada por una matriz, donde cada fila representa un elemento, y cada columna representa un grupo. Cada celda de la matriz representará el tamaño del elento o peso del objeto (en el caso de BPP) o el tiempo de procesamiento del trabajo (en el caso de PMS) para un contenedor o máquina específica. Este formato se ha diseñado para permitir al algoritmo propuesto trabajar de manera fluida y eficiente con instancias de ambos problemas.

4.2. Instancias de prueba para BPP

Para el problema de empaquetado de contenedores de una dimensión (BPP), se utilizaron varias instancias de prueba de referencia que se utilizan en general en publicaciones. Estas instancias conforman un conjunto de 1615 casos estándar reconocidos por la comunidad científica, las son una porción del del conjunto BPPLIB (Delorme et al., 2018). Los conjuntos de prueba son los siguientes: los conjuntos de prueba U y T de Falkenauer (1996a); tres conjuntos (conjunto-1, conjunto-2, conjunto-3) de Klein and Scholl (2009); el conjunto de pruebas gau de Schwerin and Wäscher (1997); dos conjuntos de prueba (was-1, was-2) de Schwerin and Wäscher (1997) y el conjunto de prueba hard28 de CaP (2014). La descripción de las instancias del problema está disponible en la tabla 4.2.

Tabla 4.2: Grupos de Instancias para BPP

Grupo	Conjunto	Total	Capacidad	Objetos	Pesos	Ident
1	Uniform (Falkenauer, 1996a)	80	150	120, 250, 500, 1000	20-100	u_120, u_250, u_500, u_1000
1	Triplets (Falkenauer, 1996a)	80	100	60, 120, 249, 501	25-50	t_60, t_120, t_249, t_501
2	Data Set 1 (Klein and Scholl, 2009)	720	100, 120, 150	50, 100, 200, 500	1-100, 20-100, 30-100	n-c-w
2	Data Set 2 (Klein and Scholl, 2009)	480	1000	50, 100, 200, 500	Variado	n-w-b-r
2	Data Set 3 (Klein and Scholl, 2009)	10	100000	200	20000-35000	Hard
3	Was 1 (Schwerin and Wäscher, 1997)	100	1000	100	150-200	Was 1
3	Was 2 (Schwerin and Wäscher, 1997)	100	1000	120	150-200	Was 2
3	Gau 1 (Schwerin and Wäscher, 1997)	17	10000	57-239	2-7332	Gau 1
4	Hard28 (CaP, 2014)	28	1000	160-200	1-800	Hard28

4.3. Instancias de prueba para PMS

En lo que respecta al problema de programación de máquinas paralelas (PMS), las instancias de prueba que se utilizan fueron generadas con una distribución uniforme en intervalos diferentes, así como con trabajos y máquinas correlacionados (JobsCorr y MacsCorr). Estas pertenecen a un conjunto de 1400 instancias de prueba (Fanjul-Peyro and Ruiz, 2010). Estas instancias se agrupan en siete clases en función del criterio empleado para generar los tiempos de procesamiento p_{ij} en cada instancia. Las características de estas clases son las siguientes:

- Los primeros cinco grupos incluyen instancias con valores de p_{ij} distribuidos uniformemente en los intervalos U(1, 100), U(10, 100), U(100, 120), U(100, 200), y U(1000, 1100), respectivamente.
- El sexto y séptimo grupo incluyen instancias con trabajos correlacionados (JobsCorr) y máquinas correlacionadas (MacsCorr), respectivamente.

Cada subconjunto (clase) contiene diez instancias para cada una de las veinte combinaciones posibles generadas con $m = 10, 20, 30, 40, 50$ y $n = 100, 200, 500, 1000$. Por lo tanto, el tamaño de las instancias más pequeñas es 100×10, y el tamaño de las más grandes es 1000×50.

El uso de instancias de prueba de fuentes reconocidas y utilizadas en la literatura académica permite evaluar el rendimiento del algoritmo propuesto de manera más precisa y comparativa. Al proporcionar una gama de instancias de diferentes tamaños y niveles de dificultad, se puede examinar cómo el algoritmo se desempeña en una variedad de escenarios, lo que permite una evaluación más completa de su efectividad.

4.4. GGA-CGT2: una adaptación del algoritmo GGA-CGT para BBP y PMS

El Grouping Genetic Algorithm with Controlled Gene Transmission (GGA-CGT), originalmente diseñado por Quiroz Castellanos (2014), maneja instancias del problema Bin Packing (BPP). Sin embargo, el objetivo de este trabajo es extender la aplicación del algoritmo GGA-CGT a otra clase de problemas de agrupación. Debido a esto fue adaptado en este trabajo para manejar las instancias generadas para los problemas de Bin Packing (BPP) y Parallel Machine Scheduling (PMS).

Para realizar esta extensión, el algoritmo GGA-CGT original ha sido sometido a una serie de adaptaciones y modificaciones a nivel de código. Es importante señalar que estas modificaciones se han orientado principalmente a permitir que el algoritmo gestione las instancias generadas tanto para el BPP como para el PMS.

A pesar de estas modificaciones, se ha tenido un especial cuidado en preservar el rendimiento y la eficacia del algoritmo original para los problemas BPP. En otras palabras, las modificaciones implementadas no han alterado la capacidad del GGA-CGT para proporcionar soluciones de alta calidad para las instancias del BPP, asegurando así la continuidad y compatibilidad con los trabajos previos realizados en este campo.

Aunque los problemas BPP y PMS pueden considerarse similares en términos de que ambos implican la asignación de elementos a grupos (objetos a bins en el caso de BPP, trabajos a máquinas en el caso de PMS), existen diferencias fundamentales que requerían la adaptación del algoritmo.

Además, la representación de una solución y los operadores genéticos (cruce y mutación) en el GGA-CGT original estaban diseñados para trabajar con el enfoque de "grupo" del problema BPP. Para el problema PMS, estos elementos tuvieron que ser modificados para manejar el enfoque de "máquina" del problema. Por lo tanto, los cambios necesarios implicaron adaptar la representación de la solución, los operadores genéticos, y los criterios de evaluación y selección para manejar las instancias del problema PMS.

Con esta adaptación, el GGA-CGT2 ahora puede manejar instancias del problema PMS, demostrando su flexibilidad y aplicabilidad a una gama más amplia de problemas.

4.4.1. Codificación genética

En GGA-CGT, el esquema de codificación se basa en grupos (Quiroz Castellanos, 2014). Los cromosomas pueden variar en longitud dependiendo del número de contenedores usados en cada solución. Cada gen en el cromosoma representa un contenedor que almacena un grupo de objetos. Sin embargo, dado que en el problema de PMS se maneja el número de máquinas, el cromosoma maneja una longitud fija igual al número de contenedores/máquina. Asimismo, se agrega un identificador único a cada gen del cromosoma que define a qué máquina corresponde, facilitando los procesos subsecuentes.

En la Figura 4.1, cada caja representa un gen en el cromosoma, es decir, un grupo, donde dependiendo el problema se interpreta como un contenedor o una maquina. Los elementos dentro de cada caja representan los elementos que están almacenados en ese contenedor en la solución correspondiente del problema. Por lo tanto, los operadores genéticos operarán con estos grupos y la información sobre los elementos que pertenecen a cada grupo.

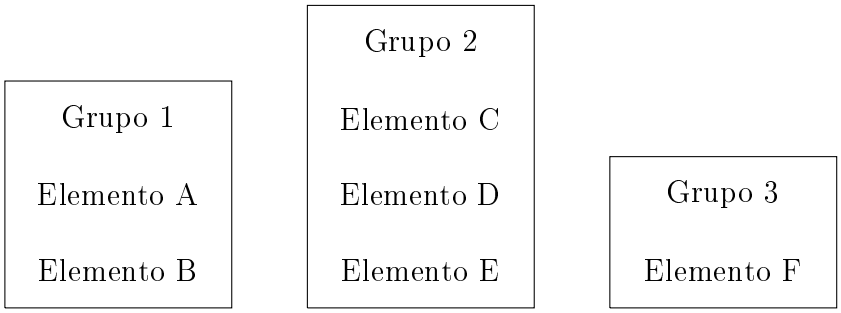

Figura 4.1: Codificación basada en grupos. El cromosoma se representa como un gen que almacena un grupo de elementos.

4.4.2. Función de aptitud

La función de aptitud tiene por objetivo primordial operar a nivel de genes para promover la transmisión de los mejores contenedores en GGA-CGT. La evaluación de la aptitud se realiza mediante la función definida por Falkenauer (1992). Para el modelo unificado m representa el número de grupos utilizados en la solución, S_i es la suma de los tamaños de los objetos en el contenedor i y c es la capacidad de los grupos.

$$F_{BPP} = \frac{1}{m} \sum_{i=1}^{m} \left(\frac{S_i}{c} \right)^2 \tag{4.1}$$

En esta versión, hemos dividido la suma total por m para obtener el promedio de los términos individuales de la suma en función de m. Esto coloca todo el lado derecho de la ecuación en relación a m.

El objetivo de esta función 4.1 es tener algunos grupos casi completos y algunos casi vacíos, en lugar de varios grupos igualmente llenos. Esto se debe a que los grupos casi vacíos permiten acomodar más fácilmente objetos adicionales, que de otro modo podrían ser demasiado grandes para caber en cualquiera de los grupos medio llenos. A pesar de no funcionar directamente para PMS, esta función se mantiene para su uso en problemas de BPP.

Caso 1: Tenemos 4 contenedores, todos ellos con la siguiente distribución de tamaños de objetos:

- Contenedor 1: 6 unidades
- Contenedor 2: 7 unidades
- Contenedor 3: 8 unidades
- Contenedor 4: 9 unidades

Usando la función de aptitud F_{BPP}, la puntuación para este caso sería:

$$F_{BPP} = \frac{1}{4}\left[\left(\frac{6}{10}\right)^2 + \left(\frac{7}{10}\right)^2 + \left(\frac{8}{10}\right)^2 + \left(\frac{9}{10}\right)^2\right]$$
$$= 0{,}155 + 0{,}196 + 0{,}256 + 0{,}324 = 0{,}23275.$$

Caso 2: Tenemos 4 contenedores, pero la distribución de tamaños de objetos es diferente:

- Contenedor 1: 2 unidades
- Contenedor 2: 7 unidades
- Contenedor 3: 9 unidades
- Contenedor 4: 10 unidades

La puntuación de aptitud para este caso sería:

$$F_{BPP} = \frac{1}{4}\left[\left(\frac{2}{10}\right)^2 + \left(\frac{7}{10}\right)^2 + \left(\frac{9}{10}\right)^2 + \left(\frac{10}{10}\right)^2\right]$$
$$= 0{,}01 + 0{,}196 + 0{,}324 + 0{,}4 = 0{,}2325.$$

En estos dos ejemplos, la función de aptitud favorece el segundo caso sobre el primero. Este es un resultado deseado ya que en el segundo caso hay un contenedor casi vacío (el de 2 unidades) que puede acomodar más objetos si es necesario, en lugar de tener todos los contenedores igualmente llenos. De este modo, la función de aptitud promueve la flexibilidad en la solución, lo que puede ser beneficioso en situaciones donde se necesita adaptarse a nuevos objetos que se van añadiendo.

4.4.3. Población inicial

GGA-CGT propone la heurística FF-ñ, que reduce el número de pasos en el proceso de empaquetado y obtiene grupos con un mejor llenado. En primer lugar, los ñ objetos más grandes son empacados en contenedores separados. Posteriormente, los $n-$ objetos restantes son empacados usando la heurística de acomodo FF sobre una permutación aleatoria de los objetos.

Para el PMS, se propone una modificación basada en el "best fit". Aquí, se busca encontrar el contenedor con suficiente espacio para acomodar el elemento, pero que

también sea el contenedor más cercano al llenarse. Esto implica revisar todos los contenedores disponibles y seleccionar el que mejor se ajuste al elemento en términos de espacio disponible.

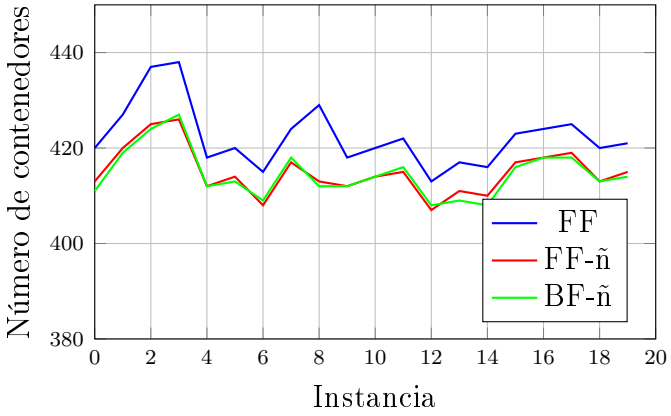

Figura 4.2: Resultados experimentales obtenidos por las heurísticas FF, FF-ñ y BF-ñ sobre las instancias u1000.

A partir de FF-ñ, se realiza una búsqueda adicional para encontrar un grupo con un mejor ajuste. La búsqueda se realiza comparando la capacidad ocupada del grupo actual más el tamaño del elemento con la capacidad ocupada de los grupos restantes más el tamaño del elemento, buscando el grupo que tenga un menor tamaño al agregar el artículo. Si se encuentra un grupo con un mejor ajuste, se actualiza el índice del grupo para seleccionar ese grupo en lugar del grupo actual.

La figura 4.2 presenta una comparación de tres heurísticas para el Problema del Bin Packing (BPP). Se observa que FF-ñ y BF-ñ muestran resultados de desempeño similares, pero mejores que sin la estrategia ñ, y su posición relativa cambia ligeramente con el número de instancia. En la figura 4.3 se observa que, para el Problema de la Máquina de Programación (PMS), como es de esperarse, la heurística propuesta BF-ñ no supera a FGGA, pero obtiene valores satisfactorios como heurística de inicialización. Estas figuras resaltan el potencial de BF-ñ para generar la población inicial. se mantiene el número máximo de máquinas definido por m.

4.4.4. Operador de cruzamiento por agrupación

En GGA-CGT, el operador de cruzamiento por agrupación toma dos padres y produce un descendiente. El proceso de cruzamiento empieza seleccionando un gen del primer padre y copiándolo al descendiente. Posteriormente, selecciona un gen del segundo padre, pero si el grupo de este gen ya está presente en el descendiente, selecciona el siguiente gen. Este proceso continúa hasta que todos los grupos del descendiente están llenos.

Para el problema PMS, se propone un método que recalcula la capacidad ocupada

de un grupo sumando los pesos de todos sus elementos cada vez que se asigna un grupo a la nueva solución. Este ajuste en el operador de cruzamiento se debe a la naturaleza única de las máquinas y los trabajos en el problema PMS como se muestra en la figura 4.3.

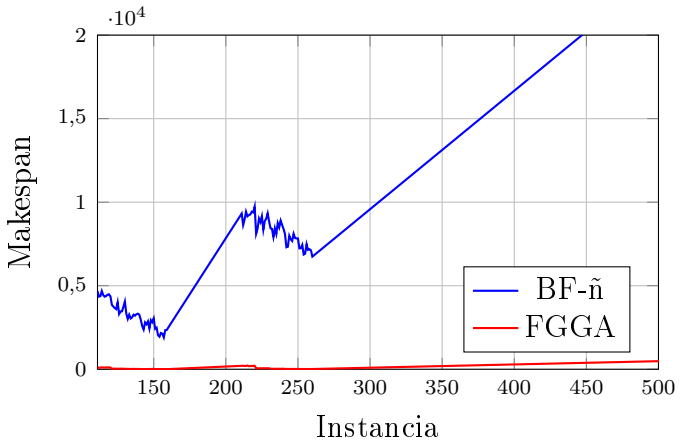

Figura 4.3: Resultados experimentales obtenidos por las heurísticas FGGA y BF-ñ sobre las instancias 1a100[8] de PMS.

4.4.5. Operador de mutación por agrupación

En el caso de BPP, el operador de mutación adaptativa se encarga de cambiar los objetos de un contenedor a otro. El operador de mutación adaptativa del GGA-CGT se mantiene con modificaciones para manejar la naturaleza del problema PMS. Para el problema PMS, se realiza una mutación por agrupamiento de trabajos.

Los trabajos son considerados en orden descendente de su tiempo de procesamiento, y se realiza un reacomodo por pares para mejorar el llenado de las máquinas. En esta heurística, cada par de trabajos en las máquinas es considerado hasta que un reemplazo es posible, asegurando un mejor aprovechamiento de las capacidades de las máquinas.

4.4.6. Comparación de tiempos de ejecución

Se probó tanto la implementación del algoritmo GGA-CGT original como la versión modificada para analizar sus tiempos de ejecución. Ambos algoritmos fueron probados utilizando las instacias de prueba para el BPP proporcionar una representación precisa de su eficiencia en diferentes situaciones.

Las pruebas para la comparación de tiempos de ejecución entre el algoritmo GGA-CGT original y su adaptación se realizaron siguiendo las configuraciones sugeridas por (Quiroz Castellanos, 2014):

- Tamaño de la población (|P|) = 100
- Número máximo de generaciones (max_gen) = 500
- Número de individuos a cruzar $n_c = 20$
- Número de individuos a mutar $n_m = 83$
- El número de contenedores a eliminar (nb) en el operador de mutación para las soluciones no clonadas se calcula utilizando una tasa de cambio (k) = 1.3
- Número de individuos en el grupo élite (|B|) = 10
- El número de contenedores a eliminar (nb) en el operador de mutación cuando la solución fue clonada se calcula utilizando una tasa de cambio (k) = 4
- Edad máxima para que un individuo pueda ser clonado (max_edad) = 10

Las pruebas se realizaron en una máquina con sistema operativo Windows 10, equipada con un procesador Intel i7-10750H a 2.60GHz y 16GB de memoria RAM. Esta configuración permitió realizar simulaciones de alta fidelidad y obtener resultados precisos y replicables.

Los resultados mostraron que, en términos generales, la versión adaptada del GGA-CGT registró tiempos de ejecución más rápidos que el algoritmo original en instancias más pequeñas del problema. Este mejor rendimiento puede atribuirse a las optimizaciones específicas incorporadas en la adaptación, como el uso de nuevas estructuras de datos más detalladas a nivel de programación y la implementación de funciones mejoradas para la generación de la población inicial y los operadores de cruzamiento y mutación.

Sin embargo, para algunas instancias mayores, la versión modificada del GGA-CGT mostró tiempos de ejecución más lentos. Esto podría deberse a la creciente complejidad del problema a medida que aumenta el tamaño de la instancia, lo que puede requerir un mayor número de operaciones y por lo tanto, un mayor tiempo de ejecución.

Es importante recordar que el tiempo de ejecución es solo una medida de la eficiencia de un algoritmo. Mientras que la versión modificada de GGA-CGT fue más rápida para instancias más pequeñas (58 %) y más lenta para algunas instancias más grandes (42 %), esto no necesariamente se traduce en una mejor calidad de las soluciones. Otros aspectos, como la precisión de las soluciones y la robustez del algoritmo ante cambios en los parámetros del problema, también son fundamentales al evaluar el rendimiento de un algoritmo de optimización.

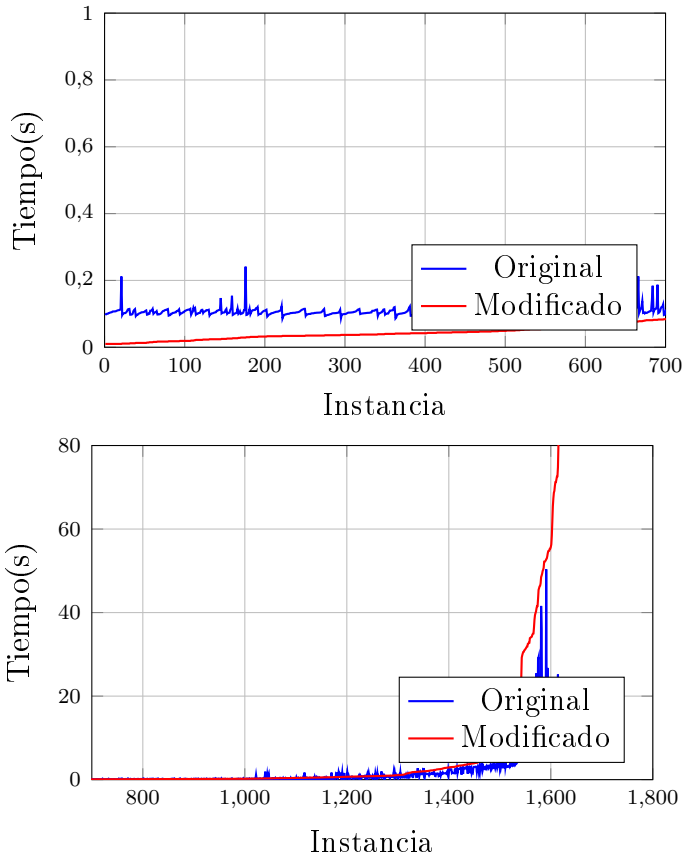

Figura 4.4: Comparación del tiempo de ejecución de la modificación del GGA-CGT.

Para corroborar los resultados obtenidos a través de los análisis temporales vistos en la figura 4.4, se llevó a cabo la prueba de Wilcoxon. Esta prueba no paramétrica se utiliza para determinar si existen diferencias significativas entre dos grupos de datos. En este caso, los grupos de datos son los tiempos de ejecución registrados para el algoritmo GGA-CGT original y su adaptación.

La hipótesis nula (H0) de la prueba es que las medianas de las diferencias entre las muestras de los dos grupos son iguales. Esto implicaría que no hay una diferencia significativa en los tiempos de ejecución entre el algoritmo original y su adaptación.

Los resultados de la prueba de Wilcoxon se resumen en la tabla 4.3:

Estadístico	p-valor	Resultado
569771	0.103 E-4	H0 es rechazada

Tabla 4.3: Resultados de la prueba de Wilcoxon para la comparación entre el algoritmo GGA-CGT original y su adaptación para el PMS.

El p-valor obtenido fuede 0.103E-4. Dado que este p-valor es muy pequeño (menor que 0.05),se rechaza la hipótesis nula. Esto sugiere que existe una diferencia significativaen los tiempos de ejecución entre el algoritmo GGA-CGT original y su adaptaciónpara el PMS.

Este resultado estadísticamente demuestra que las modificaciones realizadas en la versión adaptada del GGA-CGT incrementa su tiempo de ejecución en los casos de prueba estándar.

4.4.7. PMS con el GGA-CGT modifcado

Durante el desarrollo y las pruebas iniciales de las modificaciones implementadas para adaptar el GGA-CGT al modelo de instancias propuesto para ambos problemas BPP y PMS, se observó que las soluciones generadas tienden a acercarse más al límite inferior *lb*. Este límite, definido en la adaptación de las instancias para el PMS, se considera como la capacidad del contenedor, lo que sugiere que el GGA-CGT busca maximizar el volumen de cada grupo al intentar acercarse a dicho límite.

Es importante señalar que, a pesar de que las soluciones obtenidas se aproximan al límite inferior, todavía se sitúan relativamente lejos de las soluciones de referencia para este problema. Este resultado sugiere que, aunque la adaptación del GGA-CGT al PMS ha demostrado tener potencial, no es comparable por sí sola a las soluciones de mejor rendimiento conocidas en la literatura. Por esta razón, se agregó la búsqueda local iterada descrita en la siguiente sección.

4.5. Adaptación de la Búsqueda Local Iterada (ILS)

La Búsqueda Local Iterada es una metaheurística diseñada para abordar problemas de optimización combinatoria como el PMS. Esta técnica combina estrategias de exploración y explotación para encontrar soluciones óptimas o cercanas al óptimo. La exploración del espacio de soluciones se lleva a cabo mediante la perturbación aleatoria de la solución actual, mientras que la explotación se realiza mediante la mejora de la solución perturbada utilizando técnicas de búsqueda local Santos et al. (2019).

4.5.1. Aplicación de Vecindarios

En el algoritmo de ILS empleado en este estudio, las perturbaciones de las soluciones actuales se realizan mediante la implementación de los vecindarios propuestos por Santos et al. (2019). Estos vecindarios representan diferentes maneras de modificar la solución actual y se describen a continuación:

Vecindario de intercambio: Realiza un intercambio de dos trabajos entre dos máquinas diferentes. Este movimiento puede ayudar a encontrar un mejor balance entre las máquinas y, por tanto, una solución de mayor calidad.

Vecindario de eliminación e inserción: Implica la eliminación de un trabajo de una máquina y su posterior inserción en otra máquina diferente. Esta perturbación puede ser útil para equilibrar la carga entre las máquinas y mejorar la eficiencia total.

Vecindario de permutación: Altera el orden de los trabajos dentro de la misma máquina. Al reorganizar los trabajos, se puede encontrar un orden más eficiente, lo que podría reducir el tiempo total de finalización.

4.5.2. Algoritmo de ILS

En la implementación de la ILS como una especie, se realizan una serie de movimientos y evaluaciones. Se recorren los grupos y los elementos dentro de cada grupo en la especie. Este paso aplica el concepto de "vecindario de intercambio" y "vecindario de inserción":

a. Vecindario de inserción: Se evalúa si un elemento en un grupo puede ser movido a otro grupo (llamado targetGroup) sin exceder la capacidad del targetGroup. Si es factible, se realiza el movimiento.

b. Vecindario de intercambio: Si no es posible mover el elemento directamente, se intenta un intercambio entre un elemento en el grupo actual y otro elemento en el targetGroup. Este intercambio solo se realiza si ambos grupos resultan con un volumen reducido después del intercambio.

Estas perturbaciones permiten explorar el espacio de soluciones vecinas y potencialmente encontrar una solución de mayor calidad. La aplicación iterada de estos movimientos a lo largo de las 30 evaluaciones permitidas puede llevar a mejoras significativas en la solución inicial, esto para limitar la sobrecarga debido a la complejidad de la ILS. La Figura 4.5 muestra el desempeño (makespan) de la ILS y la heurística FGGA obtenidos al aplicar un conjunto de instancias del PMS. FGGA es una heurística ampliamente utilizada para resolver PMS; sin embargo, en este experimento preliminar, se observa que la ILS parece tener un mejor desempeño.

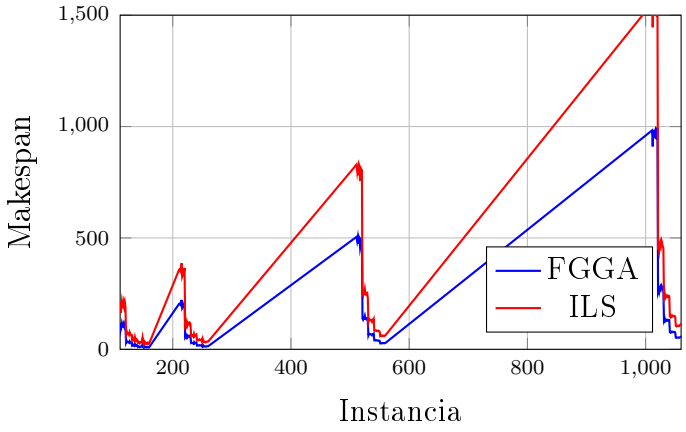

Figura 4.5: Resultados experimentales obtenidos por las heurísticas FGGA y ILS sobre las instancias 1a100 de PMS.

4.6. Diseño del algoritmo coevolutivo de agrupación con cooperación auto-adaptativa (GCA-AC)

El algoritmo coevolutivo de agrupación con cooperación auto-adaptativa (GCA-AC por sus siglas en ingles Grouping Coevolution Algorithm with Adaptive Cooperation) utiliza la estrategia de coevolución para abordar de manera efectiva tanto el problema del Bin Packing (BPP) como el problema de Parallel Machine Scheduling (PMS).

Ambos son problemas complejos y desafiantes de optimización combinatoria, que requieren de estrategias de resolución robustas y versátiles. La coevolución, como principio central del GCA-AC, permite que diferentes "especies" o estrategias de solución evolucionen conjuntamente y se especialicen en diferentes aspectos de estos problemas.

En el contexto del BPP, algunas especies pueden centrarse en minimizar el número de contenedores utilizados, mientras que otras pueden concentrarse en maximizar el uso de espacio dentro de cada contenedor. Para el PMS, algunas especies pueden enfocarse en equilibrar la carga de trabajo entre las máquinas, mientras que otras pueden buscar minimizar el tiempo total de finalización.

La cooperación entre estas especies especializadas, junto con la adaptación de la población en cada iteración, permite al GCA-AC explotar las fortalezas de diferentes estrategias y abordar de manera efectiva ambos problemas. Esta capacidad para adaptarse y especializarse en diferentes aspectos de los problemas es lo que distingue al GCA-AC y lo convierte en una herramienta poderosa para resolver tanto el BPP como el PMS. En este algoritmo se utilizan cuatro especies, cada una con su propio enfoque y estrategia de búsqueda y optimización.

Especie 1: Optimización

La Especie 1 (s1) utiliza el GGA-CGT modificado (GGA-CGT2) y sigue la configuración original de los parámetros propuesta por Quiroz et al. (2014). Esta especie se centra en la optimización, buscando mejorar las soluciones existentes durante el proceso evolutivo.

Especie 2: Diversificación Controlada

La Especie 2 (s2) utiliza también el algoritmo GGA-CGT, pero se centra más en la diversificación de las soluciones. Para lograr esto, aumenta el porcentaje de soluciones de élite en su población, permitiendo trabajar con un mayor número de soluciones de menor fitness. Este enfoque promueve una exploración más amplia del espacio de búsqueda y evita la convergencia prematura a óptimos locales.

Especie 3: Reinicio

La Especie 3 (s3) se enfoca en introducir nuevas soluciones al espacio de búsqueda para mantener la diversidad de la población. Se utiliza la técnica de reinicio, reemplazando algunos individuos en la población con soluciones generadas de manera aleatoria. Esta técnica permite la exploración de regiones no visitadas del espacio de búsqueda, lo que potencialmente puede conducir al descubrimiento de soluciones de mejor calidad.

Especie 4: Búsqueda Local Iterada (ILS)

La Especie 4 (s4) implementa el enfoque de la Búsqueda Local Iterada (ILS), que se basa en el trabajo de Santos et al. (2019). El ILS propuesto fue diseñado para abordar principalmente PMS, pero también puede resolver BPP. Este enfoque combina la exploración del espacio de búsqueda mediante perturbaciones aleatorias con la búsqueda local para mejorar las soluciones existentes. Las perturbaciones de las soluciones actuales se realizan mediante la implementación de los vecindarios de inserción e intercambio La Especie 4 realiza hasta 30 evaluaciones por solución, enfocándose en refinar soluciones prometedoras y mejorar su calidad.

El GCA-AC asigna de manera auto adaptiva una proporción de la población global a cada especie, asegurando una representación en base a su rendimiento y una y participación activa de todas las especies; en la en base a su rendimiento y una siguiente sección se detalla esta estrategia.. Además, la distribución aleatoria de la población en cada iteración promueve la diversidad genética y una exploración amplia del espacio de búsqueda, aumentando así la probabilidad de encontrar soluciones óptimas y evitando óptimos locales

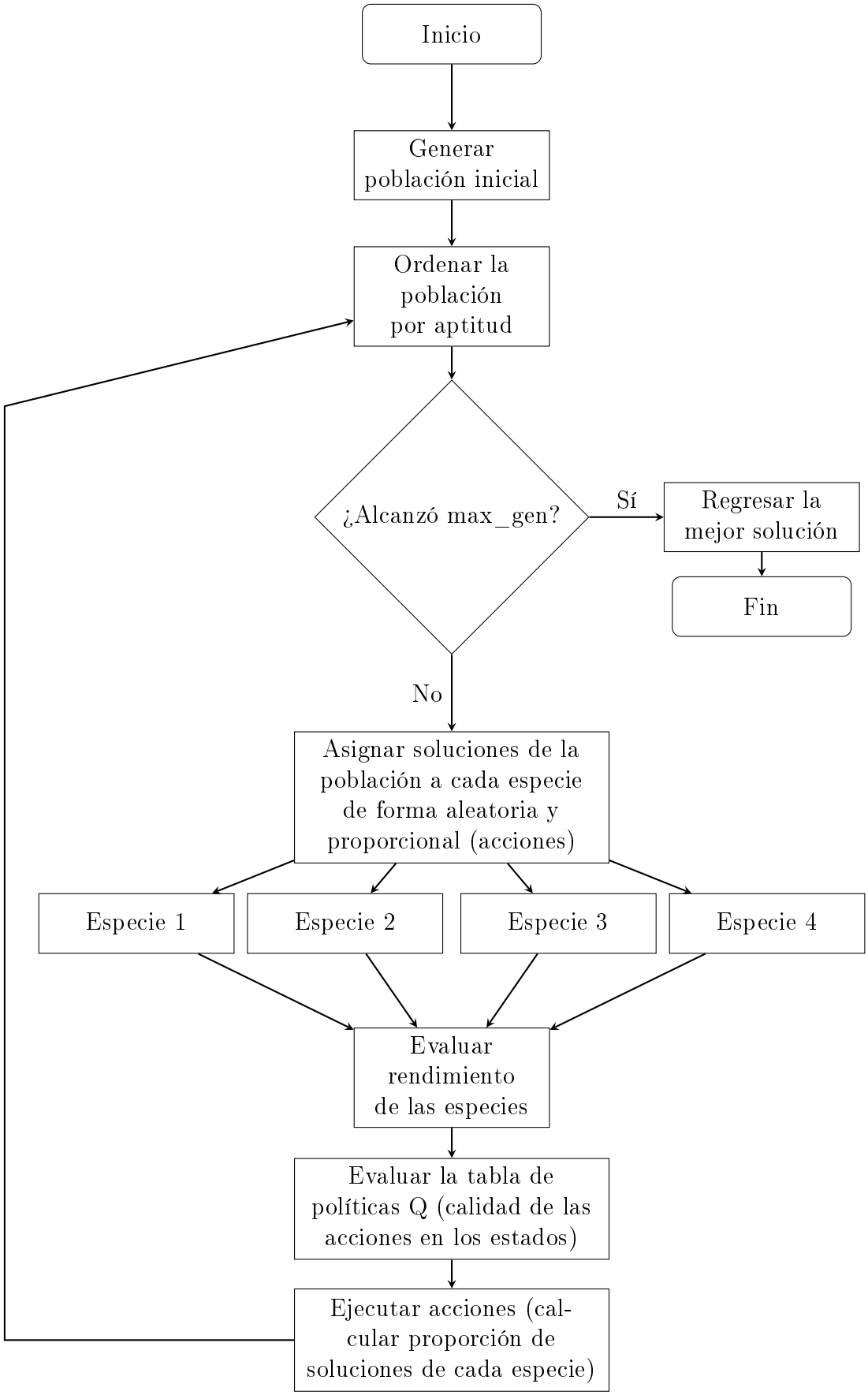

Figura 4.6: Algoritmo coevolutivo de agrupación con cooperación auto-adaptativa (GCA-AC)

El diagrama de flujo de la Figura 4.6 ilustra los pasos en el Algoritmo coevolutivo de agrupación con cooperación auto-adaptativa (GCA-AC). Este algoritmo comienza con la generación de una población inicial de soluciones candidatas, que son luego ordenadas por aptitud o calidad (*Generar población inicial* y *Ordenar la población por aptitud*).

Posteriormente, se realiza una verificación para determinar si se ha alcanzado la cantidad máxima de generaciones (*¿Alcanzó max_ gen?*). Si no se ha alcanzado el límite, el algoritmo procede a asignar soluciones de la población a cada una de las especies de manera aleatoria (*Asignar soluciones a la población a cada especie de forma aleatoria*). Estas especies, representadas por *Especie 1*, *Especie 2*, *Especie 3*, y *Especie 4* en el diagrama, son estrategias de solución distintas que reciben una porción de la población y realizan su propia optimización.

Una vez que cada especie ha efectuado su estrategia de optimización, el algoritmo evalúa la calidad de las soluciones de cada una de ellas (*Evaluar especies*). Seguidamente, se evalúa la tabla de políticas Q (*Evaluar la tabla de políticas Q*), que guía las decisiones sobre qué acciones tomar a continuación (*Ejecutar acciones*).

Este ciclo se repite hasta que se alcanza el número máximo de generaciones predefinido. En ese punto, el algoritmo devuelve la mejor solución encontrada (*Regresar la mejor solución*) y concluye su ejecución (*Fin*). Este enfoque permite que GCA-AC explore el espacio de soluciones de manera sistemática, con cada especie aportando su propia estrategia de optimización y cooperando a través del intercambio de soluciones de alta calidad.

Es importante resaltar que, en contraste con el algoritmo en el GGA-CGT2 y en el GCA-AC, en el GCA-AC no se establecieron límites como criterio de parada del algoritmo. En el GGA-CGT, se utiliza el límite L1 para el BPP como criterio de parada. Sin embargo, esta estrategia no fue implementable en el GCA-AC debido a las diferencias en los objetivos de los problemas del BPP y del PMS. Mientras que el BPP busca minimizar el número de contenedores utilizados, el PMS apunta a minimizar el tiempo total de finalización. Estos objetivos divergentes requieren enfoques distintos para determinar cuándo detener el algoritmo. Por lo tanto, en el GCA-AC, se optó por no establecer límites como criterio de parada, permitiendo una mayor flexibilidad y adaptabilidad a la naturaleza de los problemas que se abordan.

4.6.1. Desarrollo de estrategia auto-adaptativa de parámetros

En el caso del Algoritmo Coevolutivo de Agrupación con Cooperación Auto-Adaptativa (GCA-AC), se requiere que el algoritmo pueda adaptarse automáticamente y elegir la mejor estrategia para resolver el problema sin necesidad de intervención manual del usuario. Esto es especialmente relevante cuando el algoritmo trata problemas de agrupación, que son problemas complejos y requieren estrategias

de búsqueda especializadas.

Para el desarrollo de la estrategia auto-adaptativa de parámetros para las cuatro especies propuestas en el algoritmo coevolutivo. Además de los ajustes individuales para cada especie, es importante gestionar la cooperación entre ellas. Cada especie tiene asignado un porcentaje de la población global que participa en cada iteración. Durante cada iteración, la población global se distribuye aleatoriamente de acuerdo con el porcentaje asignado a cada especie. Esto permite que cada especie tenga una representación proporcional en la solución del problema en cuestión. Esta distribución proporcional fomenta una exploración diversa del espacio de soluciones, lo que aumenta la probabilidad de encontrar soluciones óptimas.

La asignación de porcentaje a cada especie también juega un papel crucial en la regulación de la cooperación entre especies. Para regular esta cooperación, se emplea un algoritmo de Q-learning, una forma de aprendizaje por refuerzo. El porcentaje asignado a cada especie define su participación en la solución del problema, y el algoritmo de Q-learning ajusta estos porcentajes en función del rendimiento de las especies. Se da una ventana de aprendizaje de una cantidad dada de iteraciones para realizar las evaluaciones y actualizar la tabla Q , la cual almacena los valores de calidad o utilidad de las acciones (porcentaje) en cada estado (especies), y es utilizada por el agente para guiar su comportamiento y maximizar sus recompensas en un entorno de aprendizaje por refuerzo. Esta estrategia asegura que el algoritmo pueda adaptarse de forma dinámica a los problemas y seguir mejorando las soluciones a lo largo del tiempo.

4.6.2. Aplicación del Q-learning en el Algoritmo Coevolutivo de Agrupación con Cooperación Auto-Adaptativa (GCA-AC)

El Q-learning es un algoritmo de aprendizaje por refuerzo que es independiente del modelo. Este algoritmo aprende el valor de una acción en un estado particular, lo que significa que no necesita un modelo del entorno para su funcionamiento. Esto se traduce en que puede manejar problemas con transiciones y recompensas estocásticas sin necesidad de adaptaciones (Sutton and Barto, 2018).

Para cualquier proceso de decisión de Markov finito (FMDP, por sus siglas en inglés), el Q-learning encuentra una política óptima en el sentido de maximizar el valor esperado de la recompensa total en todos y cada uno de los pasos sucesivos, partiendo del estado actual.

La función Q que el algoritmo calcula se refiere a las recompensas esperadas para una acción tomada en un estado dado. El algoritmo Q-learning funciona actualizando iterativamente una tabla de valores Q, la cual mapea los estados a acciones y recompensas esperadas. Los valores Q se actualizan utilizando la siguiente fórmula:

$$Q(s,a) = Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)] \tag{4.2}$$

donde:

- α es la tasa de aprendizaje.
- r es la recompensa recibida al tomar la acción a en el estado s.
- γ es el factor de descuento.
- s' es el próximo estado después de tomar la acción a en el estado s.
- a' es la acción óptima en el estado s'.

En el contexto del Algoritmo Coevolutivo de Agrupación con Cooperación Auto-Adaptativa (GCA-AC), el Q-learning se emplea para regular la cooperación entre especies. Aquí, los estados corresponden a las especies con el mejor fitness en las últimas 10 iteraciones. Las acciones, por otro lado, están relacionadas con la modificación de los porcentajes de cada especie, y las recompensas son otorgadas en función de si el estado se mantiene o cambia.

En el GCA-AC, los parámetros α, γ y r son definidos de la siguiente manera:

- α (tasa de aprendizaje) se establece en 0.1, lo que indica cuánta influencia tiene la nueva información en la tabla Q.
- γ (factor de descuento) se fija en 0.9, determinando la importancia de las recompensas futuras en relación a las inmediatas.
- r (recompensa) se asigna un valor de +1 si el estado se mantiene y -1 si el estado cambia.

El proceso de Q-learning en el GCA-AC permite la cooperación eficaz entre las especies, llevando a soluciones de mayor calidad y permitiendo que cada especie se especialice en diferentes aspectos del problema (Pizzuti, 2008).

4.6.3. Diseño de especies

En esta sección, abordaremos el diseño de cada una de las especies y su comportamiento, tanto de forma individual como en su interacción con otras especies.Las pruebas del algoritmo GCA-AC se llevaron a cabo en una máquina con sistema operativo Windows 10, equipada con un procesador Intel i7-10750H a 2.60GHz y 16GB de memoria RAM.

Tabla 4.4: Conjunto de instancias BPP $\nu - c$ (Carmona-Arroyo et al., 2021)

Clase	Inst.	Especie 1	Especie 2	GCA-AC with s1 y s2
Uniform	80	80	80	80
Triplets	80	80	80	80
Data Set 1	720	720	720	720
Data Set 2	480	480	480	480
Data Set 3	10	10	5	10
Was 1	100	100	68	100
Was 2	100	100	98	100
Gau 1	17	16	12	16
Hard28	28	16	5	16
Total	1615	1602	1548	1602

Especie 1 y Especie 2: GGA-CGT

Tanto la Especie 1 como la Especie 2 implementan una adaptación que se realizó para el algoritmo GGA-CGT para el modelo de instancias propuesto para trabajar tanto para el problema del Bin Packing (BPP) como para el problema de Parallel Machine Scheduling (PMS).

Ambas especies fueron diseñadas para complementarse entre sí. Mientras que la Especie 1 se centra en la optimización de soluciones, utilizando la configuración original de los parámetros propuesta por Quiroz et al. (2014), la Especie 2, en cambio, se centra en la diversificación controlada de las soluciones.

La Especie 2 utiliza un grupo de élite de mayor tamaño, un subconjunto de individuos seleccionados de la población en función de su fitness. Al aumentar el tamaño del grupo de élite se pretende mantener la diversidad en la población ya que se permite trabajar con soluciones de un rango de fitness mayor que a su vez se toman en los operadores de cruza y muta en la generación de nuevas soluciónes.

La primera evaluación se realizó utilizando la instancia BPP10, que pertenece al conjunto BPP $\nu - c$ (Carmona-Arroyo et al., 2021). Esta instancia consta de 100 objetos y tiene una capacidad de contenedor de 1000.

En el uso conjunto de la Especie 1 y la Especie 2 en el GCA-AC a lo largo de 500 generaciones. Se usó un grupo élite de 10 % en la Especie 1 y 90 % en la Especie 2. A pesar de la aparente ventaja en tamaño del grupo élite de la Especie 2, los resultados demostraron que la Especie 1 obtuvo una mayor participación en las soluciones finales. Esta tendencia se observó en todas las instancias probadas, lo que indica que la Especie 1, a pesar de tener un menor tamaño de grupo élite, demostró una mayor efectividad al explorar y optimizar el espacio de soluciones.

La figura 4.7 muestra la evolución de la participación de las Especies 1 y 2 a lo largo de las 500 generaciones en la instancia BPP10.

La Especie 1, con su enfoque en la optimización, demostró ser eficaz al buscar y mejorar soluciones de alta calidad. Por otro lado, la Especie 2, a pesar de su mayor tamaño de grupo élite y su enfoque en la diversificación, no logró superar a la Especie 1 en términos de contribución a las soluciones finales.

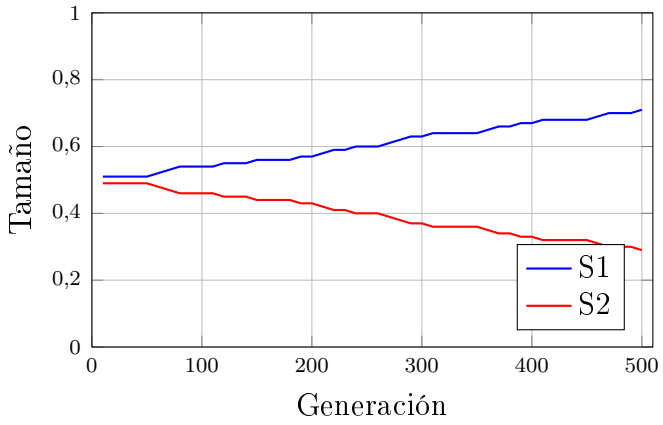

Figura 4.7: Instancia BPP10 con S1 y S2.

La figura 5.13 presenta el estudio con el problema de Parallel Machine Scheduling (PMS), los resultados obtenidos utilizando la Especie 1 y la Especie 2 se mantienen consistentes con los mencionados anteriormente para la adaptación del GGA-CGT.

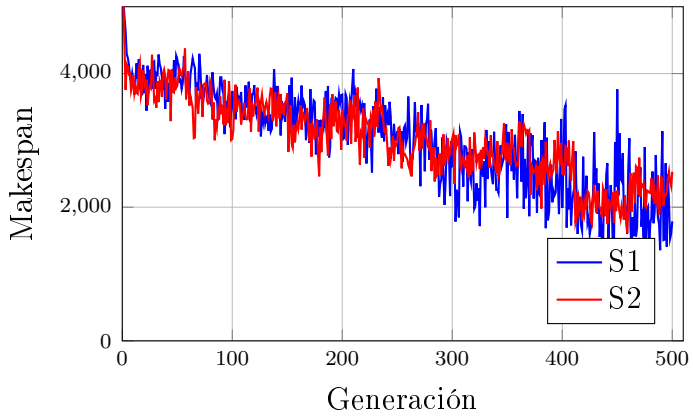

Figura 4.8: Makespan de las especies (s1 y s2) en la instancia 120-10a100.

Especie 3: Método de Reinicio

La Especie 3 en el algoritmo coevolutivo de agrupación con cooperación auto-adaptativa (GCA-AC) utiliza el método de reinicio para introducir nuevas soluciones al espacio de búsqueda. Este método se basa en técnicas heurísticas y metaheurísticas para reiniciar el proceso de búsqueda desde cero. La idea detrás del reinicio es evitar que el algoritmo quede atrapado en mínimos locales o cuando no logra encontrar una buena solución en un tiempo razonable.

La heurística BF-ñ para el problema de bin packing se basa en la idea de ordenar los objetos de mayor a menor tamaño antes de proceder a su agrupación en contenedores. Una vez ordenados, la heurística intenta insertar cada objeto en el contenedor que tenga la menor cantidad de espacio libre pero que aún sea suficiente para albergar el objeto. Si no se encuentra un contenedor adecuado, se abre un nuevo contenedor para el objeto.

En el caso de la Especie 3, se utiliza la heurística BF-ñ para generar soluciones completamente nuevas al reemplazar toda la población. Este método permite explorar nuevas regiones del espacio de búsqueda y evitar estancamientos en mínimos locales. Al reemplazar completamente la población correspondiete a la Especie 3, se proporciona una oportunidad para descubrir soluciones potencialmente mejores. Esto contribuye a mantener la diversidad y a encontrar soluciones óptimas en el problema de agrupación.

Especie 4: Búsqueda Local Iterada

La inclusión de la Especie 4 en el algoritmo coevolutivo de agrupación con cooperación auto-adaptativa (GCA-AC) complementa y mejora el desempeño del algoritmo en el problema de la Programación de Máquinas Paralelas (PMS). Mientras que las especies anteriores (Especies 1, 2 y 3) tienen enfoques más generales y no están especializadas específicamente en el PMS, la Especie 4 se centra exclusivamente en este problema.

La Especie 4 utiliza el enfoque de búsqueda local iterada (ILS) específicamente diseñado para el PMS. Esta estrategia de búsqueda se adapta y optimiza la asignación de tareas a las máquinas paralelas, mejorando el tiempo de finalización total y buscando soluciones de alta calidad. Al estar especializada en el PMS, la Especie 4 puede aprovechar las características únicas de este problema y aplicar estrategias de búsqueda altamente efectivas y específicas.

La incorporación de la Especie 4 en el GCA-AC proporciona una mayor capacidad de explotación del espacio de búsqueda del PMS, lo que se traduce en una mejora significativa en la calidad de las soluciones encontradas. Al combinar las fortalezas de las especies anteriores, que se centran en la optimización, la diversificación y la introducción de nuevas soluciones, con la especialización y capacidad de búsqueda local iterada de la Especie 4, el GCA-AC se convierte en un enfoque altamente efectivo y completo para resolver el problema de la Programación de Máquinas Paralelas.

Esta sinergia entre las diferentes especies del GCA-AC, donde cada una aporta su enfoque único y especialización, permite abordar eficientemente tanto el problema del Bin Packing (BPP) como el problema de la Programación de Máquinas Paralelas (PMS), proporcionando soluciones de alta calidad y superando los resultados obtenidos por las especies individuales.

Capítulo 5

Experimentación

Este capítulo se centra en el diseño y los resultados de las pruebas experimentales realizadas para evaluar la eficacia y el rendimiento del algoritmo GCA-AC propuesto. Los experimentos se llevaron a cabo de manera rigurosa y sistemática, siguiendo principios de reproducibilidad y robustez, para garantizar la validez y confiabilidad de los resultados obtenidos.

5.1. Analisis del comportameniento de las especies

En esta sección presenta un análisis exhaustivo del comportamiento de las cuatro especies (s1, s2, s3 y s4) en la resolución de instancias de prueba. El propósito principal de este análisis es evaluar el rendimiento y la eficacia de estas especies en diversos escenarios, así como determinar la efectividad de la estrategia de cooperación implementada en la búsqueda de soluciones óptimas.

En el marco de esta investigación, se ha establecido la siguiente configuración para el algoritmo:

Tamaño de la población: Se ha establecido un tamaño de población de 100 individuos.

Número de iteraciones: Se realizarán 500 iteraciones para el algoritmo.

Ventana de aprendizaje: La ventana de aprendizaje, que define el número de iteraciones pasadas que se consideran para la actualización de los valores Q, se ha establecido en 10 iteraciones.

Tasa de elitismo: Se ha definido una tasa de elitismo del 10 %. Esto implica que el 10 % de los individuos más aptos de la población se mantendrán de generación en generación.

Tasa de disminución del tamaño de las especies: Se ha establecido una tasa de disminución del tamaño de las especies del 1 %.

Semilla para el generador de números aleatorios: Se ha seleccionado la semilla número 1 para el generador de números aleatorios, con el fin de reproducir y controlar la aleatoriedad en los experimentos.

5.1.1. Porcentaje de especies

En esta sección, se realizo un análisis detallado de la evolución del tamaño de las especies en el GCA-AC. El objetivo principal es examinar cómo las especies evolucionan y se adaptan a lo largo del tiempo, aprovechando el poder del aprendizaje por refuerzo.

BPP

En primer lugar, se se resuelven algunas instancias de ejemplo del BPP para analizar la interacción entre las especies. Con este fin se generarán graficas para mostrar el porcentaje de especies dominantes en cada generación. Estos resultados proporcionan una visión detallada de cómo las especies evolucionan y se mantienen en el proceso de búsqueda de soluciones óptimas para el BPP.

Las Figuras 5.9 a la 5.14 muestran el grado de dominación de las especies, medida en proporción del tamaño poblacional, durante la evolución del algoritmo propuesto GGA-CA. En algunos casos se analizan las cuatro especies y en otros pares de especies.

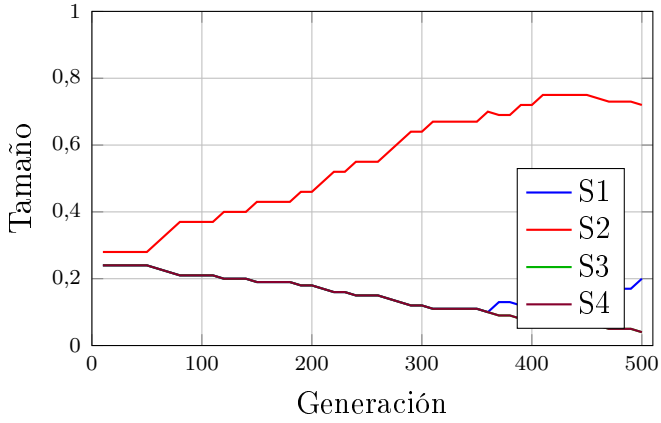

Figura 5.1: Instancia BPP10 analisis participación de las especies.

La instancia BPP_10 pertenece a un conjunto instancias sugeridas por Schwerin, Wäscher y Gau. Las primeras dos clases was 1 y was 2 se han definido como ffd-

duras (difíciles de resolver por la heurística FFD) Schwerin and Wäscher (1997) y consta de 120 objetos con una capacidad de contenedor de 1000. Las Figuras 6-9 corresponden a la instancia BPP_10.

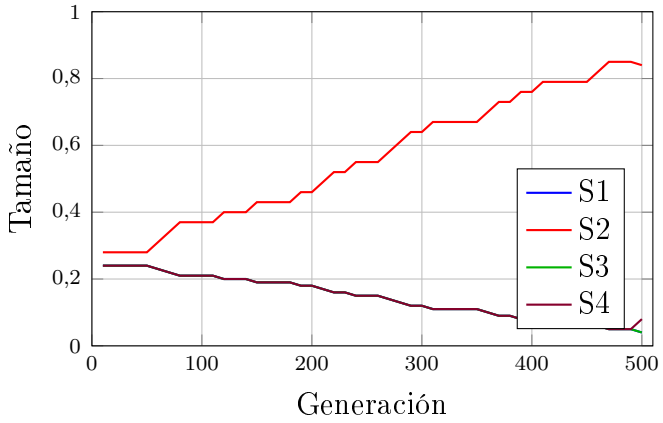

Figura 5.2: Instancia BPP_10 analisis participación de las especies.

La instancia Hard9 perteneciente a un conjunto formado por una única clase con diez instancias consideradas difíciles, mejor conocidas como hard Klein and Scholl (2009). Consta de 200 objetos con una capacidad de contenedor de 100000. Las Figuras 10-13 corresponden a la instancia Hard9.

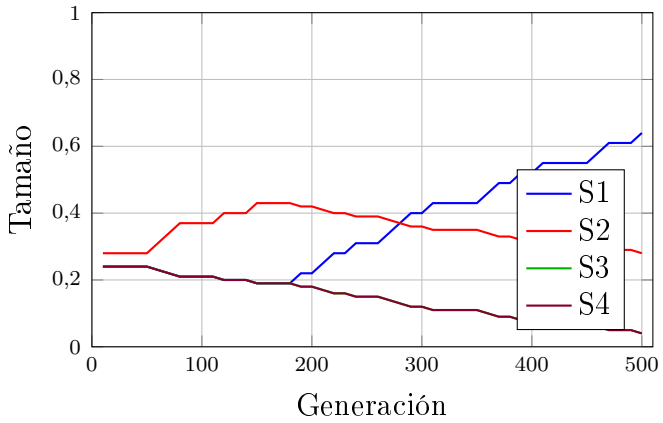

Figura 5.3: Instancia Hard9 analisis participación de las especies.

La instancia hBPP640 perteneciente a un conjunto de 28 instancias difíciles Schoenfield (2002). Consta de 180 objetos con una capacidad de contenedor de 1000. Las Figuras 14-17 corresponden a la instancia hBPP640.

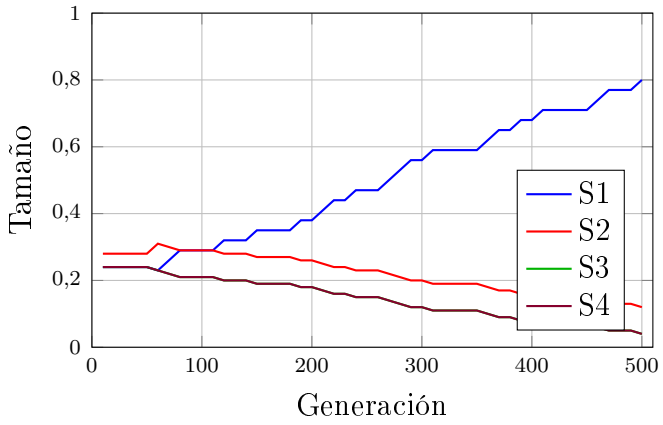

Figura 5.4: Instancia hBPP640 analisis participación de las especies.

A partir de los experimentos realizados en diversas instancias de BPP, se pudo apreciar que los comportamientos de las especies respecto a la participación en la población total varían significativamente.

En algunas instancias, se encontró que las especies exhiben comportamientos similares, lo que indica que las características de estas instancias son apropiadas para la combinación de estrategias empleadas por las especies. Por otro lado, hubo instancias en las que los comportamientos de las especies divergen notablemente, lo que sugiere que las peculiaridades de estas instancias requieren estrategias específicas para obtener resultados óptimos.

Finalmente, se encontró que, en la mayoría de los casos, la especie 1 basada en el GGA-CGT tiende a tener una mayor participación. Esto puede ser debido a que las técnicas empleadas por esta especie se adecuan bien a la naturaleza del problema BPP.

PMS

Las Figuras 5.12 a 5.14 muestran el grado de dominación de las especies en GGA-CA. Para cada problema, primero se analizan las cuatro especies y posteriormente por pares.

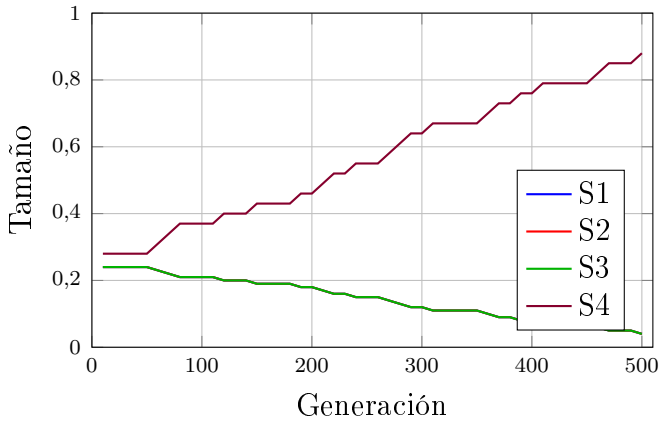

Figura 5.5: Instancia 111-1a100 analisis participación de las especies.

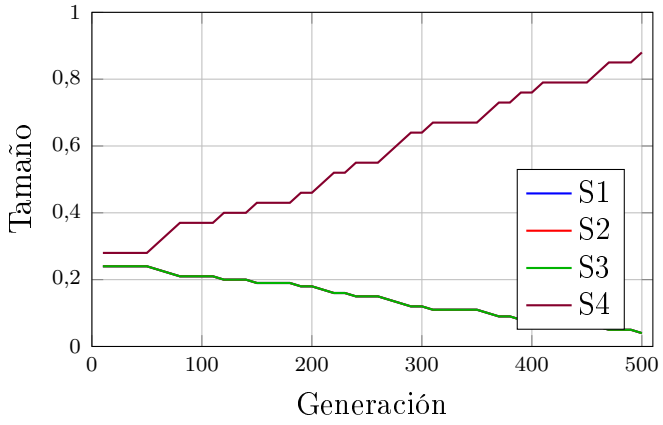

Figura 5.6: Instancia 120-10a100 analisis participación de las especies.

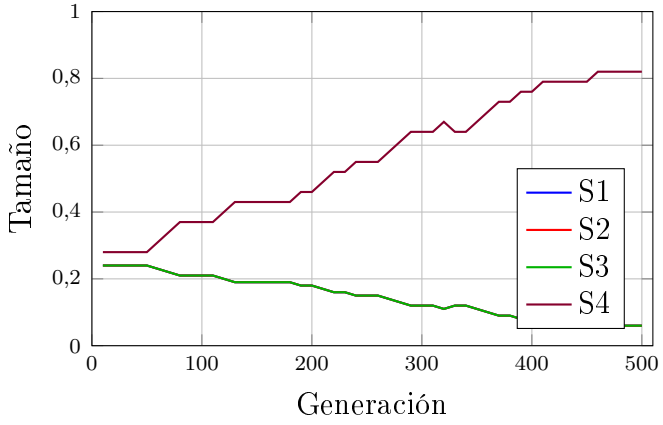

Figura 5.7: Instancia 130-100a120 analisis participación de las especies.

Los experimentos realizados en el Problema de Programación de Máquinas Paralelas (Parallel Machine Scheduling, PMS) demostraron resultados y comportamientos notables distintos a los del Problema de Empacado de Bins (BPP). En particular,

se observó que en este tipo de problemas, la especie 4, basada en la búsqueda local iterada, tiende a dominar en términos de encontrar mejores soluciones.

En resumen, en el problema PMS, la adaptación del tamaño de la población de cada especie por el algoritmo Q-Learning demostró ser eficaz para maximizar el rendimiento de las técnicas de optimización más efectivas. Estos hallazgos subrayan la importancia de la adaptabilidad y la cooperación entre diferentes técnicas de optimización en los algoritmos evolutivos para resolver problemas complejos y variados como el BPP y el PMS.

5.1.2. Fitness de especies

En esta sección, analizaremos graficamente (ver 5.8 a 5.14) la evolución del fitness de las especies en los distintos problemas. Observaremos cómo se adaptan y mejoran en respuesta a los desafíos de cada problema, brindando información valiosa sobre los procesos de adaptación y selección en sistemas coevolutivos.

BPP

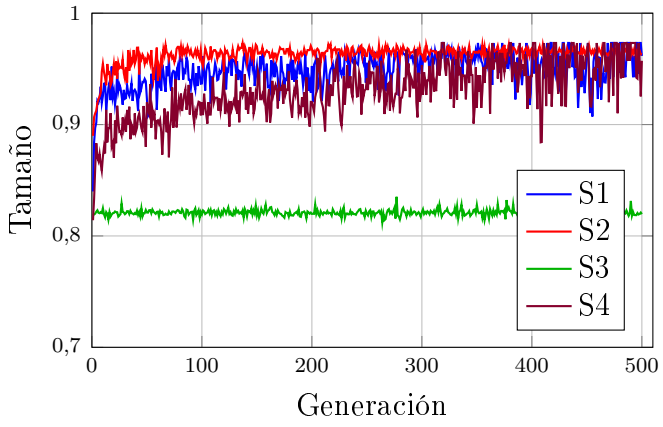

Figura 5.8: Fitness de las especies en la instancia BPP_10.

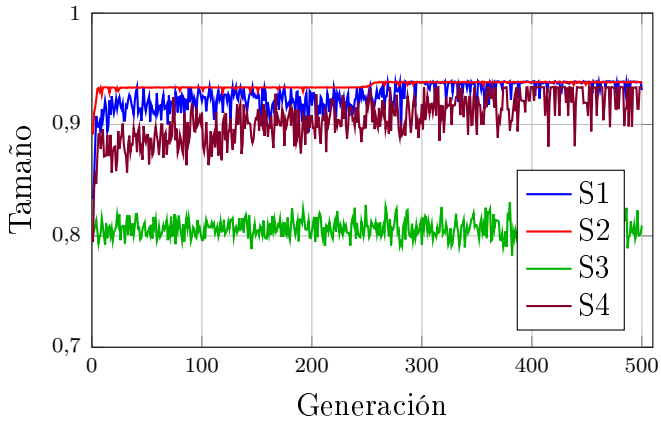

Figura 5.9: Fitness de las especies en la instancia BPP10.

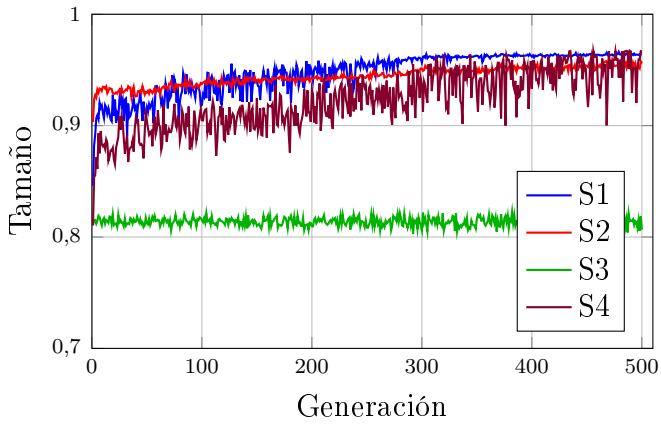

Figura 5.10: Fitness de las especies en la instancia Hard9.

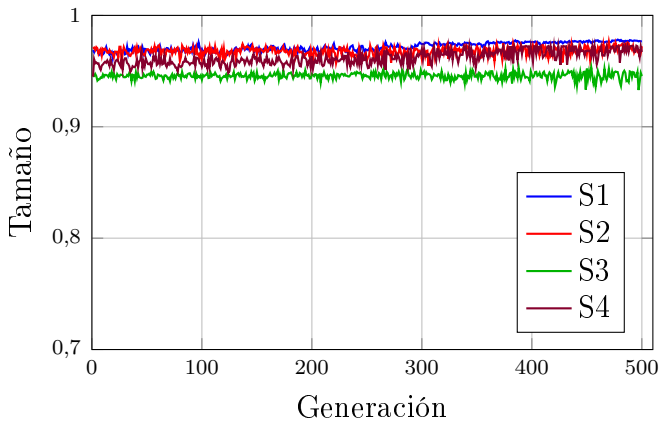

Figura 5.11: Fitness de las especies en la instancia hBPP640.

PMS

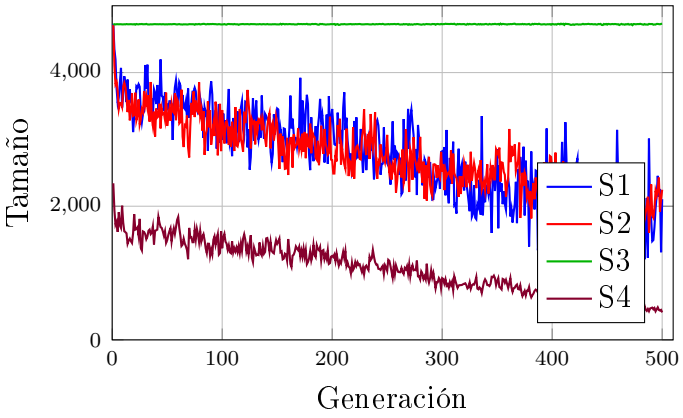

Figura 5.12: Fitness de las especies en la instancia 111-1a100.

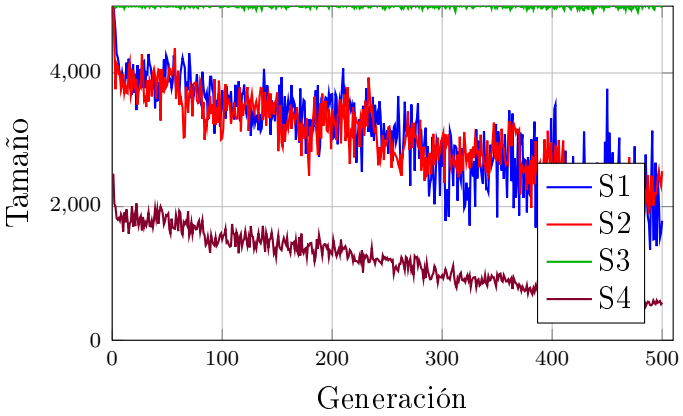

Figura 5.13: Fitness de las especies en la instancia 120-10a100.

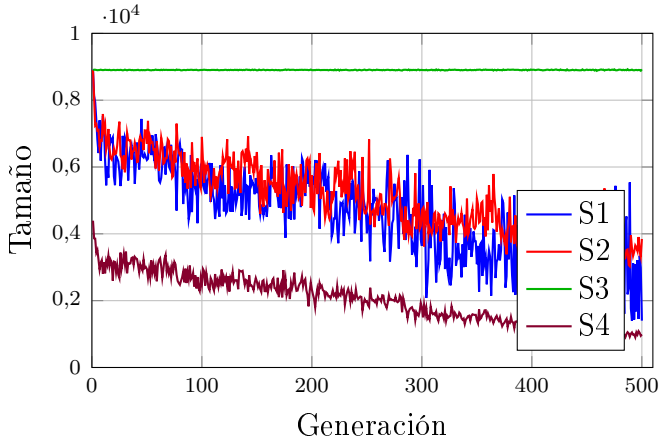

Figura 5.14: Fitness de las especies en la instancia 130-100a120.

Los resultados de los experimentos realizados para analizar el fitness de las diferentes especies proporcionan una visión detallada de su comportamiento y rendimiento en los problemas BPP y PMS. A lo largo de estos experimentos, la Especie 3, que introduce nuevas soluciones en la población general mediante un reinicio, demostró mantener un nivel de fitness bastante constante, independientemente de la instancia o el tipo de problema que se aborda. Esta estabilidad puede ser atribuida a la capacidad de esta especie de introducir diversidad en la población, lo que puede ayudar a prevenir la convergencia prematura y a mantener un buen nivel de exploración del espacio de búsqueda.

En el caso del problema BPP, se observó una variabilidad notable en los niveles de fitness entre las especies. La Especie 1, en particular, demostró tener una tendencia constante de mejorar su fitness a lo largo del tiempo. Esta tendencia puede estar vinculada a la eficacia de la estrategia de agrupación utilizada por esta especie, que puede ser especialmente adecuada para el problema BPP.

Para la instancia hBPP640, que pertenece al conjunto Hard28, se observó un comportamiento muy similar entre las cuatro especies. Esto sugiere que para algunas instancias particularmente desafiantes, la combinación de diferentes estrategias de búsqueda puede resultar en un comportamiento y un rendimiento similares.

En el problema PMS, por otro lado, la Especie 4, que utiliza una estrategia de búsqueda local, fue la que logró minimizar de manera más efectiva las soluciones a los problemas desde las primeras etapas, manteniendo esta tendencia a lo largo de la evolución. Esto contrasta con las demás especies, que en general mantuvieron peores resultados en términos de fitness para el PMS.

5.2. Diseño experimental

Para cada problema, se seleccionó un conjunto representativo de instancias de diferentes niveles de dificultad. Las instancias de BPP se seleccionaron del conjunto de benchmark propuesto por Falkenauer (1996b), que incluye tanto instancias uniformes (U) como instancias binarias (B). Para el problema PMS, se utilizaron las instancias propuestas por Graham et al. (1979), que cubren una amplia gama de configuraciones de tareas y máquinas.

Las pruebas se llevaron a cabo en una máquina con sistema operativo Windows 10, equipada con un procesador Intel i7-10750H a 2.60GHz y 16GB de memoria RAM. Este equipo permitió la realización de simulaciones de alta fidelidad, proporcionando resultados precisos y replicables.

Para la ejecución de los experimentos, se establecieron los siguientes parámetros:

- **Tamaño de la población:** Se fijó un tamaño de población de 100 individuos.

- **Número de iteraciones:** Se realizaron 500 iteraciones para el algoritmo.
- **Ventana de aprendizaje:** La ventana de aprendizaje, que define el número de iteraciones pasadas que se consideran para la actualización de los valores Q, se estableció en 10 iteraciones.
- **Tasa de elitismo:** Se definió una tasa de elitismo del 10 %. Esto implica que el 10 % de los individuos más aptos de la población se mantendrán de generación en generación.
- **Tasa de disminución del tamaño de las especies:** Se estableció una tasa de disminución del tamaño de las especies del 1 %.
- **Semilla para el generador de números aleatorios:** Para reproducir y controlar la aleatoriedad en los experimentos, se seleccionó la semilla número 1 para el generador de números aleatorios.

El tamaño de la población y el número de iteraciones para el algoritmo GGA-CGT se basan en el trabajo de Quiroz Castellanos (2014).

El tamaño de la población se estableció en 100 individuos, y el número de iteraciones se fijó en 500. Estos parámetros son consistentes con los utilizados en el estudio de Quiroz et al., y por lo tanto, brindan una base sólida para la comparación del rendimiento de nuestro algoritmo con los resultados previamente publicados.

Para la especie 1, basada en el algoritmo GGA-CGT, se optó por mantener los parámetros propuestos por Quiroz Castellanos (2014). Por lo tanto, la especie 1 fue guiada estrictamente de acuerdo con los parámetros y reglas propuestas en este estudio previo. Específicamente, se decidió aplicar la tasa de elitismo a la especie 2. Esto significa que el 10 % de los individuos más aptos de la especie 2 se mantendrán de generación en generación.

Dos parámetros fundamentales son la ventana de aprendizaje, que define cuántas iteraciones pasadas se consideran para la actualización de los valores Q, y la tasa de disminución del tamaño de las especies, que regula la proporción de la población total que conforma cada especie. Estos parámetros determinan la contribución de cada especie al proceso del algoritmo.

Inicialmente se fijaron estos valores en 10 iteraciones y el 1 % respectivamente, pero para comprender mejor su impacto, se realizaron experimentos con diferentes valores. El análisis de estos experimentos permitirá entender cómo estos parámetros pueden afectar la habilidad del algoritmo para encontrar soluciones óptimas y cómo ajustarlos para mejorar su rendimiento.

Los resultados de estos experimentos y análisis detallados se presentarán en las siguientes secciones, buscando optimizar la configuración de estos parámetros para diferentes problemas o instancias.

5.3. BPP

En este apartado, analizaremos los resultados obtenidos al aplicar el algoritmo GCA-AC a problemas de Bin Packing (BPP). Se llevo cabo experimentos con diferentes instancias del problema pertenecientes a BPPLIB* Delorme et al. (2018) y presentaremos el rendimiento de nuestra propuesta, comparándola con el algortimo de referencia GGA-CGT por Quiroz Castellanos (2014) y HEA de Borgulya (2021) . La implementación de las modificaciones y las mejoras propuestas se realizó en C++, y las pruebas se llevaron a cabo en un sistema de alto rendimiento para garantizar la fiabilidad y precisión de los resultados.

Tabla 5.1: Comparación de los resultados obtenidos por los algoritmos HEA, GGA-CGT y GCA-AC para diferentes clases de problemas.

Clase	Inst.	HEA		CGA-CGT		GCA-AC	
		opt_f	T(s)	opt_f	T(s)	opt_f	T(s)
Uniform	80	80	3.66	80	1.92	80	3.40
Triplets	80	80	4.67	80	3.55	80	3.47
Data Set 1	720	718	7.63	720	25.49	720	25.07
Data Set 2	480	480	0.44	480	6.61	480	6.48
Data Set 3	10	8	4.82	10	65.96	10	71.74
Was 1	100	100	0.84	100	0.07	100	0.26
Was 2	100	100	1.16	100	12.72	100	17.59
Gau 1	17	17	0.96	16	5.57	16	6.71
Hard28	28	28	46.98	16	32.71	18	18.78
Total	1615	1611	-	1602	154.64	1604	153.54

En todas las clases de instancias, los algoritmos lograron resultados óptimos similares, indicados por la columna “Opt_f”. En la mayoría de los casos, los dos algoritmos alcanzaron la solución óptima. Sin embargo, en las instancias “Gau 1” y “Hard28”, ambos algoritmos obtuvieron resultados subóptimos, aunque el GCA-AC superó ligeramente al GGA-CGT.

En términos de tiempo de ejecución “T(s)”, se observa que el GCA-AC tiende a ser más eficiente en general, obteniendo tiempos más cortos en la mayoría de las instancias, lo que se puede ver en las columnas “T(s)”. La excepción notable es la instancia “Data Set 3”, donde el GGA-CGT es ligeramente más rápido.

5.4. PMS

Para la comparación de GCA-AC y el algortimo de la literatura FGGA se consideran cuatro escenarios en los valores del número máximo de generaciones (max_gen): 500, 1000, 2000, y 10000. La eficiencia se mide en función del número

de instancias en las que encuentra una solución mejor que CPLEX.

	GCA-AC		FGGA	
Instance set	500	1000	500	1000
U(1, 100)	0	0	0	0
U(10, 100)	0	0	0	0
U(100, 120)	2	3	1	1
U(100, 200)	3	9	5	6
U(1000, 1100)	6	10	7	14
JobsCorr	4	9	3	10
MacsCorr	0	0	0	0
Total	15	27	16	31

Tabla 5.2: Resultados obtenidos por el algoritmo propuesto GCA-AC y el algoritmo FGGA (Ramos-Figueroa, 2022) en el problema PMS.

La efectividad del GCA-AC se encuentra mejor representada en el escenario de 1000 generaciones, donde el algoritmo logra 27 instancias mejores que CPLEX. En comparación, el algoritmo FGGA alcanza 31 instancias en el mismo escenario. En este sentido, se puede concluir que el desempeño del GCA-AC se acerca al del FGGA en aproximadamente un 87 % ($\frac{27}{31} = 0{,}87$, que multiplicado por 100 para convertir a porcentaje da 87 %).

En los conjuntos de instancias más pequeños, específicamente aquellos con rangos menores como U(1, 100) y U(10, 100), GCA-AC logró obtener una ventaja marginal en términos de rendimiento. Este comportamiento puede estar vinculado al hecho de que GCA-AC puede explorar de manera más eficiente el espacio de soluciones en instancias menores, debido a su diseño algorítmico y a las estrategias de cooperación implementadas entre las diferentes especies de su población.

Sin embargo, esta ventaja tiende a disminuir a medida que aumenta el tamaño de las instancias, lo que sugiere que GCA-AC puede tener dificultades para mantener su eficacia en instancias más grandes.

Capítulo 6

Conclusiones y Trabajos Futuros

En esta investigación, se presentó el desarrollo de un algoritmo coevolutivo de agrupación con cooperación auto-adaptativa (GCA-AC). Este algoritmo busca ser una solución general para los problemas de agrupación, enfocándose específicamente en el problema de empaquetado de contenedores de una dimensión (BPP) y en el problema de programación de máquinas paralelas (PMS).

El GCA-AC combina diversas estrategias evolutivas en su enfoque, lo que le permite adaptarse a los diferentes problemas y condiciones. El algoritmo se apoya en el mecanismo de autoadaptación para ajustar sus parámetros de manera dinámica, incrementando su capacidad para responder a cambios en el ambiente del problema.

Los resultados obtenidos, aunque preliminares, son prometedores. El algoritmo GCA-AC ha demostrado su capacidad para encontrar soluciones de calidad para las instancias de prueba utilizadas en BPP y PMS, comparables con las obtenidas por los algoritmos especializados actuales. Estos resultados sugieren que la adaptación de algoritmos genéticos y su combinación con estrategias de coevolución y autoadaptación podría ser un enfoque fructífero para el desarrollo de algoritmos generales para los problemas de agrupación.

6.1. Aportaciones

La investigación y el desarrollo del Algoritmo Coevolutivo de Agrupación con Cooperación Auto-adaptativa (GCA-AC) han aportado significativamente al campo de la optimización combinatoria. Estas aportaciones se destacan en varios aspectos clave:

1. Innovación en Algoritmos de Optimización: El GCA-AC representa un avance notable en la resolución de problemas de optimización. Este algoritmo, al inte-

grar técnicas de coevolución y auto-adaptación, ofrece un enfoque novedoso y eficaz, distinguiéndose de las soluciones convencionales centradas en problemas específicos.

2. Flexibilidad y Adaptabilidad en la Optimización: Una de las mayores contribuciones del GCA-AC es su capacidad para adaptar dinámicamente sus parámetros en respuesta a las características y desafíos específicos de diferentes problemas de optimización. Esta flexibilidad lo convierte en una herramienta valiosa para una amplia gama de aplicaciones prácticas.

3. Eficiencia en Problemas Complejos de Agrupación: El algoritmo ha demostrado ser eficaz en problemas complejos como el BPP y el PMS, ofreciendo resultados comparables a los de algoritmos especializados. Su rendimiento en estas aplicaciones subraya su potencial como solución generalizable para diversos problemas de optimización.

4. Contribución a la Investigación en Optimización Combinatoria: El desarrollo del GCA-AC aporta a la base de conocimiento existente en el campo de la optimización combinatoria, proporcionando nuevas perspectivas y métodos para abordar problemas de optimización. Esta investigación amplía las fronteras actuales y ofrece nuevas vías para futuros desarrollos en el área.

5. Potencial para Aplicaciones Futuras: La investigación proporciona una base sólida para futuras aplicaciones prácticas y teóricas en optimización. La capacidad del GCA-AC para adaptarse y rendir en diversos contextos lo convierte en una herramienta prometedora para exploraciones futuras en diferentes campos, desde la logística hasta la planificación de recursos.

La contribución del GCA-AC al campo de la optimización combinatoria es multifacética, marcando un hito en la búsqueda de soluciones más flexibles, adaptables y eficientes para problemas de optimización complejos y variados. Su desarrollo no solo aborda desafíos actuales, sino que también establece una base para futuras investigaciones y aplicaciones en una variedad de dominios industriales y académicos.

Los resultados obtenidos en esta investigación son testimonio de la capacidad del GCA-AC para generar soluciones competitivas en comparación con algoritmos especializados. En particular:

1. Rendimiento en Problemas BPP y PMS: El GCA-AC ha mostrado un desempeño destacado en las instancias de prueba para BPP y PMS. Su habilidad para ofrecer resultados comparables a los algoritmos especializados subraya su potencial como una solución general para problemas de optimización.

2. Flexibilidad y Adaptabilidad: La capacidad de autoadaptación del GCA-AC, que permite ajustes dinámicos en sus parámetros, ha sido fundamental para su éxito. Esta característica asegura que el algoritmo se mantenga eficiente y efectivo frente a diversas condiciones y tipos de problemas.

3. Integración de Estrategias Evolutivas: La combinación de coevolución, auto-adaptación y algoritmos genéticos ha resultado ser una estrategia potente, facilitando un enfoque más holístico y robusto para la resolución de problemas de agrupación.

Estos resultados corroboran que el GCA-AC no solo es un algoritmo versátil y eficiente, sino que también representa un avance significativo en la investigación de algoritmos de optimización. Su capacidad para adaptarse y generar soluciones de calidad en problemas complejos lo posiciona como una herramienta valiosa en el campo de la optimización combinatoria.

6.2. Trabajos Futuros

Los hallazgos de este estudio abren varias líneas de investigación para el futuro:

1. **Ampliar la generalidad del algoritmo:** El algoritmo desarrollado aquí se centra en dos problemas específicos de agrupación. Sería interesante investigar cómo se puede adaptar y extender este enfoque para abordar una gama más amplia de problemas de agrupación.
2. **Experimentación con más parámetros auto-adaptativos:** En este estudio, se consideraron dos parámetros para la auto-adaptación. En trabajos futuros, se podría considerar la posibilidad de auto-adaptar un mayor número de parámetros, para permitir una mayor flexibilidad y capacidad de adaptación del algoritmo.
3. **Combinación con otras técnicas metaheurísticas:** El GCA-AC combina la coevolución y la auto-adaptación con los algoritmos genéticos. Podría ser útil explorar cómo la integración de otras técnicas metaheurísticas (como la optimización de enjambre de partículas, búsqueda de armonía, entre otras) podría mejorar el rendimiento del algoritmo.
4. **Adaptación a problemas dinámicos:** Finalmente, sería relevante investigar cómo este enfoque se puede adaptar a problemas de agrupación dinámicos, donde las restricciones o los datos del problema pueden cambiar con el tiempo.

En conclusión, el GCA-AC emerge como una solución prometedora en el ámbito de la optimización combinatoria, mostrando gran potencial para avanzar en la resolución de problemas complejos de agrupación. La flexibilidad y adaptabilidad inherentes a este algoritmo abren un campo amplio para futuras investigaciones y aplicaciones prácticas en diversas áreas.

Bibliografía

Capad 2014. cutting and packing at dresden university, benchmark data sets. `http://www.math.tu-dresden.de/~capad/cpd-ti.html#pmp`, 2014.

T. Achterberg and J. A. Hoogeveen. A survey of bin packing heuristics. *European Journal of Operational Research*, 184(2):528–542, 2008.

A. Agarwal and O. Ergun. A survey of machine scheduling problems with sequence-dependent setup times. *European Journal of Operational Research*, 187(2):659–681, 2008.

C. C. Aggarwal and P. Reddy. *Data clustering: Algorithms and applications*. CRC Press, 2014.

R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network flows: Theory, algorithms, and applications*. Prentice Hall, 1993.

V. Alvarez. Modelo para representar la complejidad del problema y el desempeño de algoritmos. Master's thesis, Instituto Tecnológico de Cd. Madero, Tamaulipas, México, 2006.

Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, illustrated edition, abril 2009. ISBN 0521424267.

B. Awerbuch. Load balancing in homogeneous distributed systems. *Journal of the ACM*, 34(2):204–225, 1987.

S. Baruah and R. Buyya. Load balancing in cloud computing: A survey. *ACM Computing Surveys (CSUR)*, 41(5):1–58, 2009.

Sara Basse. *Computer Algorithms: Introduction to Design and Analysis*. Addison-Wesley Publishing Company, 1998.

Christian Blum and Andrea Roli. *Metaheuristics in Combinatorial Optimization*. Wiley, 2003.

I. Borgulya. A hybrid evolutionary algorithm for the offline bin packing problem. *Central European Journal of Operations Research*, 29(2):425–445, 2021.

W. Bożejko, Ł. Kacprzak, and M. Wodecki. Parallel coevolutionary algorithm for three-dimensional bin packing problem. In *Artificial Intelligence and Soft Computing: 14th International Conference, ICAISC 2015, Zakopane, Poland, June 14-18, 2015, Proceedings, Part I*, pages 319–328. Springer International Publishing, 2015.

P. Brucker. *Scheduling algorithms*. Springer-Verlag, 1984.

Mirsad Buljubasic. *Efficient local search for several combinatorial optimization problems*. PhD thesis, Université Montpellier, 2015.

G. Carmona-Arroyo, J. B. Vázquez-Aguirre, and M. Quiroz-Castellanos. One-dimensional bin packing problem: An experimental study of instances difficulty and algorithms performance. In O. Castillo and P. Melin, editors, *Fuzzy Logic Hybrid Extensions of Neural and Optimization Algorithms: Theory and Applications*, volume 940 of *Studies in Computational Intelligence*. Springer, 2021. doi: 10.1007/978-3-030-68776-2_10. URL `https://doi.org/10.1007/978-3-030-68776-2_10`.

Carlos A. Coello Coello, Juan A. Lozano, and Gary R. Raidl. Parallel-machine scheduling problem: An experimental study of instances difficulty and algorithms performance. 2002.

Carlos A. Coello Coello, Gary B. Lamont, and David A. Van Veldhuizen. *Evolutionary Algorithms for Solving Multi-Objective Optimization Problems*. Springer, 2007.

K. Deb. *Multi-objective optimization using evolutionary algorithms*. John Wiley and Sons, 2002.

M. Delorme, M. Iori, and S. Martello. Bpplib: a library for bin packing and cutting stock problems. *Optim Lett*, 12:235–250, 2018. doi: 10.1007/s11590-017-1192-z. URL `https://doi.org/10.1007/s11590-017-1192-z`.

Johann Dréo, Andrzej Pétrowski, Patrick Siarry, and Éric Taillard. *Metaheuristics for Hard Optimization Methods and Case Studies*. Springer-Verlag, Berlin Heidelberg, 2006.

A. Duarte, J.J. Pantrigo, and M. Gallego. *Metaheuristicas*. Universidad Rey Juan Carlos, 2007.

A. Díaz, J. González, M. Laguna, P. Mascato, T. Tseng, F. Glover, and M. Ghaziri. *Optimización Heurística y Redes Neuronales*. Editorial Parainfo, España, 1996.

A. Elhag and E. Ozcan. A grouping hyper-heuristic framework based on linear linkage encoding for graph coloring. In *2013 13th UK Workshop on Computational Intelligence (UKCI)*, pages 321–326. IEEE, 2013.

E Falkenauer. A hybrid grouping genetic algorithm for bin packing. *Journal of Heuristics*, 2:5–30, 1996a.

Emanuel Falkenauer. A new representation and operators for genetic algorithms applied to grouping problems. *Evolutionary Computation*, 2:123–144, 1994.

Emanuel Falkenauer. A hybrid grouping genetic algorithm for bin packing. *Journal of Heuristics*, 2:5–30, 1996b.

Eric Falkenauer. The grouping genetic algorithm—widening the scope of the gas. *Belgian Journal of Operations Research, Statistics and Computer Science*, 33:79–102, 1992.

Luis Fanjul-Peyro and Rubén Ruiz. Iterated greedy local search methods for unrelated parallel machine scheduling. *European Journal of Operational Research*, 207 (1):55–69, 2010.

D. B. Fogel. *Evolutionary computation: toward a new philosophy of machine intelligence*. John Wiley and Sons, 2006.

N. García. Análisis de las propiedades del problema de la distribución de bases de datos y su influencia en el desempeño de algoritmos heurísticos. Master's thesis, Instituto Tecnológico de Cd. Madero, Tamaulipas, México, 2004.

R. García. Hiper-heurístico para resolver el problema de cartera de proyectos sociales. Master's thesis, Instituto Tecnológico de Cd. Madero, Tamaulipas, México, 2010.

M. R. Garey, D. S. Johnson, and L. J. Stockmeyer. Strong np-completeness results. *Journal of the Association for Computing Machinery*, 23(4):473–485, 1976.

M. Ghirardi and C. N. Potts. Makespan minimization for scheduling unrelated parallel machines: A recovering beam search approach. *European Journal of Operational Research*, 165(2):457–467, 2005. doi: 10.1016/j.ejor.2004.04.015.

M. Ghirardi and C. N. Potts. A recovering beam search approach for makespan minimization in unrelated parallel machine systems. 2013.

D. E. Goldberg. Coevolutionary algorithms: A new approach to complex systems. In *Proceedings of the 2nd International Conference on Genetic Algorithms*, pages 10–17, 1989.

D. E. Goldberg. Coevolutionary algorithms: A useful tool for the system designer. *International Journal of Systems Science*, 24(8):1281–1295, 1993.

Jessica Elena González San Martín. *Un estudio formal de heurísticas para el problema de empacado de objetos de una dimensión*. PhD thesis, Instituto Tecnológico de Cuidad Madero, 2021.

R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. *Optimization and approximation in deterministic sequencing and scheduling: A survey*, volume 5. Annals of Discrete Mathematics, 1979.

John J. Greferstette. *Self-Adaptive Systems: An Engineering Approach*. 1995.

M. Grötschel, L. Lovász, and A. Schrijver. *Geometric algorithms and combinatorial optimization*. Springer Science and Business Media, 1988.

A. Gupta, A. Kumar, and R. Singh. A heuristic approach for bin packing with overlapping items. *International Journal of Innovative Research in Computer and Communication Engineering*, 8(5):1380–1386, 2020.

D. J. Hall and B. Smith. Allocation of tasks to agents: An experimental study of the effects of agent selection and task characteristics. *Organizational Behavior and Human Decision Processes*, 44(1):1–19, 1989.

Nick Hall, Chris Potts, and D. J. Smith. *Scheduling in manufacturing and services*. John Wiley & Sons, 2003.

M. Helal, G. Rabadi, and A. Al-Salem. A tabu search algorithm to minimize the makespan for the unrelated parallel machines scheduling problem with setup times. *International Journal of Operations Research*, 3(3):182–192, 2006. URL `http://www.orstw.org.tw/ijor/vol3no3/Paper-3--IJOR-Vol3_3_-Ghaith%20Rabadi.pdf`.

J. H. Holland. Coevolutionary algorithms: The next step in evolutionary computation. *Evolutionary Computation*, 1(1):39–44, 1991.

A. Husseinzadeh Kashan et al. A particle swarm optimizer for grouping problems. *Inform. Sci.*, 2013. doi: http://dx.doi.org/10.1016/j.ins.2012.10.0326.

O. H. Ibarra and C. Kim. Fast approximation algorithms for bin packing problems. *Journal of the ACM*, 22(4):593–605, 1975.

D. S. Johnson. The np-completeness column: A survey. *Journal of the Association for Computing Machinery*, 22(1):1–16, 1975.

D. S. Johnson and M. R. Garey. Np-completeness of some bin-packing problems. *SIAM Journal on Applied Mathematics*, 29(1):85–93, 1974.

H. Kaid, A. Al-Ahmari, A. Al-Shayea, E. Abouel Nasr, A. K. Kamrani, and H. A. Mahmoud. Metaheuristics for optimizing unrelated parallel machines scheduling with unreliable resources to minimize makespan. 2022. doi: 10.1177/16878132221097023.

L. Kaufman and P. J. Rousseeuw. *Finding groups in data: An introduction to cluster analysis*. John Wiley & Sons, 2009.

R. Klein and A. Scholl. Bin packing. `http://www.wiwi.uni-jena.de/Entscheidung/binpp/`, 2009.

O. Kramer. Self-adaptive optimization: A survey. In *Parallel Problem Solving from Nature (PPSN X)*, pages 853–862, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

S. Kundu, A. Das, and S. Mukherjee. Deep-pack: A vision-based 2d online bin packing algorithm with deep reinforcement learning. *arXiv preprint arXiv:1905.08655*, 2019.

K. Laterre, B. De Backer, and B. D'Hondt. Ranked reward: Enabling self-play reinforcement learning for bin packing. *arXiv preprint arXiv:1904.02686*, 2019.

E. L. Lawler. Combinatorial optimization: Networks and algorithms. *Holt, Rinehart and Winston*, 1976.

J. H. Lee and H. Jang. Uniform parallel machine scheduling with dedicated machines, job splitting and setup resources. *Sustainability*, 11(24):7137, 2019. doi: 10.3390/su11247137.

J Levine and F Ducatelle. Ant colony optimization and local search for bin packing and cutting stock problems. *Journal of the Operational Research Society*, pages 705–716, 2004.

J.J. Liang and P.N. Suganthan. Survey of evolutionary-based metaheuristics. *IEEE Transactions on Evolutionary Computation*, 13(2):187–218, 2009.

S. Mahdavi, M. E. Shiri, and S. Rahnamayan. Metaheuristics in large-scale global continuous optimization: A survey. *Information Sciences*, 295:407–428, 2015.

João P. Marques-Silva and Luís Cruz. *Self-Adaptive Software: Concepts, Architectures and Algorithms*. 2001.

João P. Marques-Silva and Luís Cruz. *Self-Adaptive Metaheuristics*. 2005.

João P. Marques-Silva and Luís Cruz. *Self-Adaptive Search: Applications in Optimization, Machine Learning, and Data Mining*. 2013.

G. Martello and P. Toth. *Knapsack problems: Algorithms and applications*. John Wiley & Sons, 1990.

Z. Michalewicz and D. B. Fogel. *How to Solve It: Modern Heuristics*. Springer-Verlang, 2nd edition, 2004.

M. Mohiuddin, M. U. Khan, and A. Ullah. Secure distributed adaptive bin packing algorithm for cloud storage. *IEEE Access*, 7:119491–119501, 2019.

N. Pillay and R. Qu. A metric for evaluating the generality of hyperheuristics. In *Automated Design of Machine Learning and Search Algorithms*, pages 2–14. Springer, Cham, 2021.

M. Pinedo. *Scheduling: Theory, algorithms, and systems*. Prentice Hall, 2008.

M. L. Pinedo. *Scheduling: Theory, algorithms, and applications with emphasis on manufacturing*. Springer Science & Business Media, 2012.

Clara Pizzuti. Cooperative coevolution and its application to the design of hyperheuristics. 2008.

J. Pérez-Ortega, H. Castillo-Zacatelco, D. Vilariño-Ayala, A. Mexicano-Santoyo, J. C. Zavala-Díaz, A. Martínez-Rebollar, and H. Estrada-Esquivel. A new heuristic strategy for the bin packing problem. *Ing. invest. y tecnol.*, 17(2), June 2016.

Marcela Quiroz Castellanos. *Caracterización del proceso de optimización de algoritmos heurísticos aplicados del problema de empacado de objetos en contenedores*. PhD thesis, Instituto Tecnológico de Tijuana, 2014.

M. A. I. Ramos-Figueroa, M. Quiroz-Castellanos, and E. Mezura-Montes. Efficient heuristic strategies for the parallel-machine scheduling problem with unrelated machines and makespan minimization. *Universidad Veracruzana*, 2022.

O. Ramos-Figueroa, M. Quiroz-Castellanos, E. Mezura-Montes, and O. Schütze. Metaheuristics to solve grouping problems: A review and a case study. *Swarm and Evolutionary Computation*, 53:100643, 2020.

Octavio Ramos-Figueroa. *Efficient Heuristic Strategies for the Parallel-Machine Scheduling Problem with Unrelated Machines and Makespan Minimization*. Phd thesis, Xalapa, Veracruz, México, 2022.

Ingo Rechenberg. *Self-Adaptive Evolutionary Computation*. 1994.

C. D. Rosin and R. K. Belew. New methods for competitive coevolution. *Evolutionary Computation*, 19(1):1–29, 2011.

S. J. Russell and P. Norvig. *Artificial intelligence: A modern approach*. Pearson Education, 4 edition, 2020.

H. G. Santos, T. A. Toffolo, C. L. Silva, and G. Vanden Berghe. Analysis of stochastic local search methods for the unrelated parallel machine scheduling problem. *International Transactions in Operational Research*, 26(2):707–724, 2019.

J. E Schoenfield. Fast, exact solution of open bin packing problems without linear programming. *IEEE Access*, 7:119491–119501, 2002.

A. Scholl, R. Klein, and U. Jurgens. A new lower bound for the bin packing problem. *Operations Research Letters*, 21(3):189–193, 1997.

P. Schwerin and G. Wäscher. The bin-packing problem: A problem generator and some numerical experiments with ffd packing and mtp, 1997.

A. Simões and E. Costa. Cooperative coevolutionary algorithms in optimization problems. *Progress in Artificial Intelligence*, 1(2):153–173, 2012.

Michael Sipser. *Introduction to the Theory of Computation*. Cengage India, 3rd edition, noviembre 2006. ISBN 8131525295.

R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT Press, 2 edition, 2018.

E. G. Talbi. *Metaheuristics: From design to implementation*. John Wiley & Sons, 2009.

E. Vallada and R. Ruiz. A genetic algorithm for the unrelated parallel machine scheduling problem with sequence dependent setup times. *European Journal of Operational Research*, 211(3):612–622, 2011. doi: 10.1016/j.ejor.2011.01.011.

Y. Wang, Z. H. Jia, and K. Li. A multi-objective co-evolutionary algorithm of scheduling on parallel non-identical batch machines. *Expert Systems with Applications*, 167:114145, 2021.

Xin-She Yang. *Self-Adaptive Heuristics for Optimization*. 2009.