UNIVERSIDAD JUÁREZ AUTÓNOMA DE TABASCO

DIVISIÓN ACADÉMICA DE CIENCIAS Y TECNOLOGÍAS DE LA INFORMACIÓN

FRAMEWORK BIO-INSPIRADO PARA PROBLEMAS DE OPTIMIZACIÓN GLOBAL MULTI-OBJETIVO

TESIS PARA OBTENER EL GRADO DE:

DOCTOR EN CIENCIAS DE LA COMPUTACIÓN

PRESENTA:

JOSÉ ADRIAN GARCÍA LÓPEZ

BAJO LA DIRECCIÓN DE:

DRA. BETANIA HERNÁNDEZ OCAÑA

EN CODIRECCIÓN:

DR. OSCAR ALBERTO CHÁVEZ BOSQUEZ

CUNDUACÁN, TABASCO, A, SEPTIEMBRE 2025

UNIVERSIDAD JUÁREZ AUTÓNOMA DE TABASCO

DIVISIÓN ACADÉMICA DE CIENCIAS Y TECNOLOGÍAS DE LA INFORMACIÓN

FRAMEWORK BIO-INSPIRADO PARA PROBLEMAS DE OPTIMIZACIÓN GLOBAL MULTI-OBJETIVO

TESIS PARA OBTENER EL GRADO DE:

DOCTOR EN CIENCIAS DE LA COMPUTACIÓN

PRESENTA:

JOSÉ ADRIAN GARCÍA LÓPEZ

BAJO LA DIRECCIÓN DE:

DRA. BETANIA HERNÁNDEZ OCAÑA

EN CODIRECCIÓN:

DR. OSCAR ALBERTO CHÁVEZ BOSQUEZ

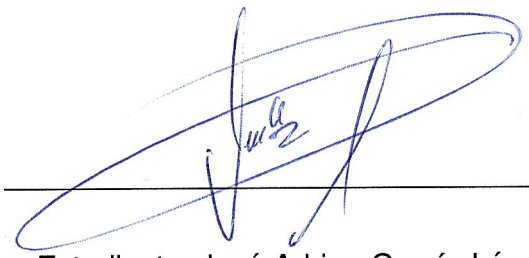
CUNDUACÁN, TABASCO, A, SEPTIEMBRE 2025

Declaración de Autoría y Originalidad

En la Ciudad de Cunduacán el día dos del mes de Septiembre del año 2025, el que suscribe **José Adrian García López**, alumno del Programa de la **Doctor en Ciencias de la Computación** con número de matrícula **221h18002**, adscrito a la **División Académica de Ciencias y Tecnologías de la Información**, de la Universidad Juárez Autónoma de Tabasco, como autor de la Tesis presentada para la obtención de Grado de Doctorado y titulada **Framework bio-inspirado para problemas de optimización global multi-objetivo**, dirigida por la Dra. Betania Hernández Ocaña y el Dr. Oscar Alberto Chávez Bosquez.

DECLARO QUE: La Tesis es una obra original que no infringe los derechos de propiedad intelectual ni los derechos de propiedad industrial u otros, de acuerdo con el ordenamiento jurídico vigente, en particular, la LEY FEDERAL DEL DERECHO DE AUTOR (Decreto por el que se reforman y adicionan diversas disposiciones de la Ley Federal del Derecho de Autor del 01 de Julio de 2020 regularizando, aclarando y armonizando las disposiciones legales vigentes sobre la materia), en particular, las disposiciones referidas al derecho de cita. Del mismo modo, asumo frente a la Universidad cualquier responsabilidad que pudiera derivarse de la autoría o falta de originalidad o contenido de la Tesis presentada de conformidad con el ordenamiento jurídico vigente.

Cunduacán, Tabasco a 02 de Septiembre de 2025.



Estudiante: José Adrian García López

Cunduacán, Tabasco, a 29 de agosto de 2025
Oficio No. 1529/2025/DACYTI/D

Asunto: Autorización de impresión de Tesis

C. José Adrián García López
Egresado del Doctorado en Ciencias de la Computación

En virtud de que cumple satisfactoriamente los requisitos establecidos en el Reglamento General de Estudios de Posgrado vigente en la Universidad, informo a Usted que se autoriza la impresión del trabajo recepcional ***"Framework bio-inspirado para problemas de Optimización global multi-objetivo"***, para presentar examen y obtener el Grado de Doctor en Ciencias de la Computación.

Sin otro particular, aprovecho la ocasión para enviarle un afectuoso saludo.

Atentamente

Dr. Óscar Alberto González González
Director

UNIVERSIDAD JUÁREZ AUTÓNOMA DE TABASCO
DIVISIÓN ACADÉMICA DE CIENCIAS Y TECNOLOGÍAS DE LA INFORMACIÓN

C.c.p. Mtra. Yenny Lorena Dussán Rojas. – Encargada del despacho de la Coordinación de Posgrado.
Archivo.
Consecutivo.

DR.*OAGG/YLDR

Carretera Cunduacán-Jalpa Km. 1, Colonia Esmeralda, C.P. 86690.
Cunduacán, Tabasco, México.
Tel: (993) 358 1500 ext. 6727; (914) 336 0616; Fax: (914) 336 0870
E-mail: direccion.dacyti@ujat.mx

www.ujat.mx

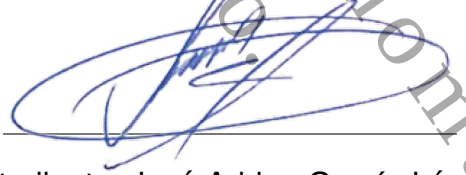
Carta de Cesión de Derechos

Villahermosa, Tabasco a 02 de Septiembre de 2025.

Por medio de la presente manifiesto haber colaborado como AUTOR en la producción, creación y/o realización de la obra denominada: **Framework bio-inspirado para problemas de optimización global multi-objetivo**.

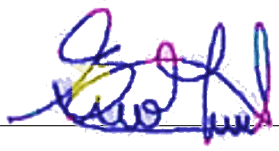
Con fundamento en el artículo 83 de la Ley Federal del Derecho de Autor y toda vez que, la creación y/o realización de la obra antes mencionada se realizó bajo la comisión de la Universidad Juárez Autónoma de Tabasco; entendemos y aceptamos el alcance del artículo en mención de que tenemos el derecho al reconocimiento como autores de la obra, y a la Universidad Juárez Autónoma de Tabasco mantendrá en un 100 % la titularidad de los derechos patrimoniales por un período de 20 años sobre la obra en la que colaboramos, por lo anterior, cedemos el derecho patrimonial exclusivo en favor de la Universidad.

COLABORADOR



Estudiante: José Adrian García López

TESTIGOS



Dra. Betania Hernández Ocaña

Dr. Oscar Alberto Chávez Bosquez

Índice general

Índice de figuras

Índice de tablas

Resumen

Problemas del mundo real implican la optimización simultánea de múltiples objetivos con restricciones complejas, conocidos como Problemas de Optimización Multi-objetivo con Restricciones (POMRs). Estos problemas, comúnmente NP-completos, exigen soluciones que equilibren diversos criterios en conflicto bajo condiciones de factibilidad, lo cual complica su resolución mediante métodos exactos.

En esta tesis se propone el desarrollo de un framework híbrido que integra dos algoritmos bio-inspirados: el algoritmo de Optimización de Enjambre de Partículas (PSO, por sus siglas en inglés) y el Algoritmo de Optimización Basado en el Forrajeo de Bacterias de doble Nado (TS-MBFOA, por sus siglas en inglés), ambos adaptados al criterio de dominancia de Pareto. El framework fue diseñado para ser modular, extensible y de libre distribución, permitiendo al usuario resolver POMRs de referencia y definidos dinámicamente. Su arquitectura, implementada en Java, incorpora un analizador matemático (*parser*) que posibilita la evaluación de funciones y restricciones descritas como expresiones algebraicas.

Se evaluó el desempeño del framework utilizando 10 problemas del benchmark CEC 2021, todos con dos objetivos y restricciones no lineales. Las soluciones fueron analizadas mediante métricas estándar como Hipervolumen y *Two Set Cover*. Los resultados muestran una tasa de finalización del 90 % y una tasa de factibilidad promedio superior al 20 %. Ambas metaheurísticas ofrecieron soluciones competitivas, con comportamientos diferenciados según el problema abordado. Esta herramienta contribuye a la toma de decisiones informada en escenarios complejos de optimización.

Keywords: Framework computacional, Metaheurísticas bio-inspiradas, Optimización multiobjetivo.

Capítulo 1

Generalidades

1.1. Introducción

Muchas aplicaciones del mundo real tienen más de un objetivo a optimizar y se modelan como Problemas de Optimización Multi-objetivo (POM). Generalmente, implican restricciones tanto en el espacio de decisión como en el objetivo (Z.-Z. Liu y Wang, 2019). Los POM se encuentran con frecuencia en distintas disciplinas o áreas tales como: medicina, ingeniería, economía, entre otras, donde buscan optimizar sus procesos o técnicas, causando al tomador de decisiones una tarea complicada (Wong y Ming, 2019).

Durante las últimas décadas, los Problemas de Optimización Multi-objetivo con Restricciones (POMR) han ganado mucha atención, ya que, en la mayoría de las aplicaciones o decisiones de la vida cotidiana, existen múltiples objetivos en conflicto con una o más restricciones donde se exigen optimizar estos objetivos mientras se satisfacen las restricciones simultáneamente (Kumar et al., 2021a). Por ejemplo, si alguien quiere comprar una casa, puede tener dos cuestiones esenciales: el costo mínimo y al mismo tiempo la ubicación en donde se encuentra la casa. Además, puede tener otros requisitos como: el número de cuartos deben ser mayor a 2, la casa debe ser de color azul y su precio debe ser menor a 500,000 pesos. Este problema de selección de casa se puede considerar un POMR, ya que contiene dos objetivos en conflicto, es decir, el costo mínimo y la ubicación, y tres restricciones: $Cuartos > 2$, $Color = Azul$ y $Precio < 50,000$.

Para el tomador de decisiones es difícil elegir entre las mejores opciones disponibles, por ello, se debe tener una medida que permita identificar qué tan buenas son las soluciones encontradas.

La dominancia de Pareto es una métrica utilizada para evaluar la eficacia de las soluciones e identificar el conjunto de estas soluciones Pareto-óptimas. Donde una solución va a pertenecer a dicho conjunto, si no puede encontrarse una solución tal que mejore uno de los objetivos sin empeorar al menos uno de los otros (López, 2013).

Los altos costos computacionales de los POMR plantean grandes desafíos para los algoritmos usados en la búsqueda de las mejores soluciones (He et al., 2020). En general, los POMR reales son muy complicados, por eso se necesita hacer uso de técnicas que generen muchas soluciones en tiempos considerados, como las metaheurísticas. Estas técnicas han ganando popularidad como métodos de inteligencia computacional, muy útiles para resolver POMR.

Las metaheurísticas son algoritmos computacionales que permiten encontrar soluciones aproximadas en un tiempo razonable a POMR. Dentro de las metaheurísticas existe una rama de algoritmos bio-inspirados eficientes en la búsqueda de soluciones óptimas. En general, estos algoritmos se clasifican en dos grupos: Algoritmos Evolutivos (AE), que operan emulando el proceso de evolución natural y la supervivencia del más apto (Eiben y Smith, 2003), y los Algoritmos de Inteligencia de Colectiva (AIC), que basan su funcionamiento en los comportamientos cooperativos de ciertos organismos simples, pero inteligentes, como abejas (Karaboga y Basturk, 2007), hormigas (Dorigo et al., 1996), bacterias (Passino, 2002), aves (Kennedy y Eberhart, 1995), entre otros.

Los AEs son algoritmos de búsqueda y en el campo de la optimización de POMR son el paradigma más aplicado (Huang et al., 2019), siendo uno de lo más estudiados el Algoritmo Genético. Los AEs se adaptan para resolver este tipo de problemas y proporcionan un conjunto de soluciones. Las características de los AEs se basan en la búsqueda poblacional y el intercambio de información entre individuos. Los AEs se han aplicado con éxito en la optimización de procesos multi-objetivo orientada a satisfacer las necesidades e intereses de las partes interesadas involucradas (Su et al., 2020), en la actualización de predicciones durante excavación (Jin et al., 2019), o en la optimización de señales de tráfico basada en el control difuso (Lin et al., 2022) y muchas más.

Algunos AEs para obtener múltiples soluciones utilizan la clasificación rápida para reducir la complejidad del tiempo, provocando comparaciones entre soluciones, redundantes o innecesarias. También, a medida que aumenta el número de objetivos, reduce la presión de selección y

ralentiza el proceso evolutivo, la diversidad y la convergencia se ven afectadas. La mayor presión hace que las soluciones prefieran algunas regiones específicas, lo que provoca que algunas soluciones desaparezcan durante el proceso evolutivo, es posible que las soluciones finales no se distribuyan uniformemente a lo largo del espacio de búsqueda (Huang et al., 2019).

En los últimos años, los AICs se han popularizado entre disciplinas y/o campos de investigación. Estos algoritmos cuentan con un amplio número de aplicaciones en diversos dominios para resolver problemas de optimización mono-objetivo y multi-objetivo, por ejemplo, problemas de programación, robots, sistemas de energía, optimización de parámetros, identificación de sistemas, procesamiento de imágenes, entre muchas más (Tang et al., 2021). Los AIC manejan un comportamiento colectivo donde procesan información de forma grupal y toman decisiones conjuntas, generando mejores decisiones (Sasaki y Biro, 2017).

Un AIC muy popular para resolver POMR es la técnica de optimización basada en enjambre de partículas (PSO, por sus siglas en inglés) (Kennedy y Eberhart, 1995). Igual que otros algoritmos de inteligencia colectiva, PSO es un algoritmo de búsqueda basado en población llamadas partículas. A diferencia de los AEs, cada partícula está asociada con una velocidad ajustada dinámicamente de acuerdo a sus comportamientos históricos. Por lo tanto, las partículas tienden a volar hacia el área de búsqueda cada vez mejor en el transcurso del proceso de búsqueda, permitiendo encontrar mejores soluciones (Shi, 2004).

PSO es aplicado en una variedad de trabajos donde cada autor integra nuevos procesos o mecanismos al algoritmo buscando mejoras en el rendimiento o resultados. Por mencionar algunos, el PSO se aplicó en la elaboración de un *framework* de optimización de peso para problemas de optimización multi-objetivo (R. Liu et al., 2020), para la optimización multi-objetivo a gran escala mejorada cuánticamente (Cao et al., 2020), en la optimización multi-objetivo para la selección de características en la calificación crediticia (Kozodoi y Lessmann, 2021), entre otras.

En 2002 Kevin Passino (Passino, 2002) propuso un nuevo AIC llamado Algoritmo de Optimización basado en el Forrajeo de Bacterias (BFOA, por sus siglas en inglés). En los procesos del BFOA las bacterias pueden comunicarse entre ellas. Este comportamiento puede resumirse en cuatro procesos: quimiotaxis (nadar y girar), agrupamiento, reproducción y eliminación-dispersión. BFOA se utilizó inicialmente para resolver problemas de optimización sin restricciones. Años más tarde, se realizaron importantes modificaciones: se redujo el número de parámetros, se incor-

poró un operador para el manejo de restricciones (Mezura-Montes y Hernández-Ocaña, 2008), y se adaptó un operador de mutación para mejorar la capacidad de exploración y explotación (Hernández-Ocaña, Pozos-Parra y Mezura-Montes, 2016) (este operador es similar a un algoritmo evolutivo). Estas modificaciones se propusieron en el algoritmo Two-Swim Modified-BFOA (TS-MBFOA).

El BFOA original está pensado para resolver problemas de optimización mono-objetivos, sin embargo, diferentes autores lo han utilizado y modificado para POM, tal como, la creación de un método de selección de características multi-objetivo (Niu et al., 2021), o en la propuesta de un algoritmo híbrido de forrajeo de bacteria basada en multi-objetivos (Y. Liu et al., 2020). TS-MBFOA se ha aplicado con éxito a problemas del mundo real y problemas *benchmarks*, pero en problemas de optimización mono-objetivos, como el problema de prueba de resorte de tensión/compresión (Hernández-Ocaña, Hernández-Torruco, Chávez-Bosquez, Canul-Reich y Montané-Jiménez, 2019), la optimización de un smart grid (Hernández-Ocaña, Hernández-Torruco, Chávez-Bosquez, Calva-Yáñez y Portilla-Flores, 2019), la generación de menús nutritivos (Hernández-Ocaña et al., 2018) y un front-end para TS-MBFOA en Matlab (Hernández-Ocaña et al., 2022).

El TS-MBFOA es un algoritmo mejorado de la versión original BFOA. Este algoritmo está escasamente estudiado y no ha sido implementado para resolver POMR, sin embargo, ha demostrado tener buen rendimiento y genera resultados óptimos y factibles en tiempos razonables, a demás que su versión original tiene aplicaciones exitosas en el mundo real, esto lo convierte en un algoritmo de interés para resolver POMR. El PSO es un algoritmo ampliamente estudiado y desarrollado para resolver POMR, sin embargo, los investigadores que utilizan este algoritmo deben de codificarlo o adaptarlo a su problema que desean resolver, esto es difícil para investigadores con poco conocimiento de programación y dificulta la obtención de resultados en tiempos oportunos.

En esta de tesis se implementó el TS-MBFOA y el PSO para simplificar los procesos de solución y brindar una manera resumida para la toma de decisiones. Esta implementación se basa en los principios de un framework, donde se reutilizan bloques de códigos que permita resolver diferentes POMR. TS-MBFOA y PSO son codificados en base a la dominancia de Pareto. El framework es diseñado con ayuda del Lenguaje Unificado de Modelado (UML) y por medio de una

Interfaz de Usuario (UI, por sus siglas en inglés) permite al tomador de decisiones con o sin conocimientos de programación, incluir su propio POMR de manera fácil. También, el framework incluye problemas del Benchmark para hacer una evaluación de los resultados generados por los AIC y hacer los análisis estadísticos. El framework es desarrollado en un lenguaje de programación de libre distribución y el código fuente es distribuido en una plataforma de alojamiento gratuita para que más investigadores puedan tener acceso.

1.2. Planteamiento del problema

1.2.1. Definición del problema

En la actualidad, muchas áreas y disciplinas cuentan con sistemas que automatizan y optimizan sus procesos. Sin embargo, una gran parte de estas tareas se encuentran dentro de la categoría de problemas NP-Completos (Garey et al., 1976) debido a la complejidad que estos procesos representan. Esto se refiere al número de variables que un experto define para representar una tarea o proceso en un problema complejo.

Un modelo matemático de un problema complejo, en ciertos casos, requiere optimizar varios criterios u objetivos que generalmente están en conflicto, es decir, uno maximizar o minimizar una función.

Un modelo matemático puede representarse como un problema de programación lineal o no lineal, y cuando se tienen más de una función objetivo se conoce como un POM. Cabe mencionar, que la mayoría de estos problemas consideran restricciones en sus modelados, descritas por el experto, las cuales pueden ser de igualdad o desigualdad, mismas que se satisfacen antes que las funciones objetivo; dando como resultado a un POMR. Este tipo de problemas pueden ser dinámicos o estáticos dependiendo del área en la que se encuentre. La optimización de un POM o POMR es simultánea y normalmente contrapuestos, ya que a cada objetivo le corresponde una solución óptima diferente, pero ninguna de estas soluciones es óptima para todos los objetivos, por ende, se busca un equilibrio entre todas las soluciones lo más cercano posible a la Frente óptima de Pareto que representa un conjunto de soluciones que satisface una función objetivo sin perjudicar las otras. Las dificultades que tiene un POMR se puede visualizar en el siguiente

ejemplo:

Se requiere comprar un auto nuevo de cuatro modelos disponibles: un Volkswagen Golf (VW), Opel Astra, Ford Focus o un Toyota Corolla. La decisión depende del precio del auto, el consumo de combustible y la potencia. Se busca un auto con un precio económico, con mayor potencia y un menor consumo de combustible. En este caso, hay un problema con tres funciones criterio y cuatro opciones posibles. Las características de los autos se muestran en la Tabla 1.1.

Criterios	**VW**	**Opel Astra**	**Ford Focus**	**Toyota Corolla**
Precio (USD)	16K	14K	13K	15K
Consumo (1 / 100km)	7.2	7.0	7.5	8.2
Potencia (kW)	66.0	62.0	55.0	71.0

Tabla 1.1. Ejemplo de un POMR: Características de los modelos de auto.

¿Cómo decidir cuál de los cuatro autos es el *"mejor"*?, si el más potente es al mismo tiempo el que más combustible consume, entonces no podemos conseguir un auto que tenga un precio económico, el más potente y el que menos combustible consume al mismo tiempo. Sin embargo, también podrían entrar en juego otros objetivos para la compra del auto como: número de cilindros del motor, número de asientos o el tipo de transmisión. Es así como en la vida diaria las empresas, áreas de investigación y las personas se enfrentan a la toma de decisiones de problemas que envuelven muchos objetivos.

El desafío principal para solucionar un POMR es encontrar el conjunto de soluciones para ofrecer al tomador de decisiones las mejores alternativas para la selección objetiva entre las disponibles.

Las metaheurísticas son técnicas de búsqueda que generan soluciones aproximadas en tiempos razonables a los POMR (Durillo y Nebro, 2011a). Están pensadas para resolver este tipo de problemas de NP-Complejidad, donde aplican algún conjunto de reglas que se basan en alguna fuente de conocimiento, a fin de explorar el espacio de búsqueda de manera más eficiente.

Dentro las metaheurísticas se encuentran los algoritmos bio-inspirados y se clasifican en los AEs, el paradigma más usado para resolver POMRs. Algunos de estos algoritmos exhiben un mal rendimiento de convergencia o diversidad en POMR con regiones factibles pequeñas (Tian et al.,

2021). Por otra parte, los AIC son técnicas que han sido probadas con éxito en problemas de optimización mono-objetivos y pocas veces implementadas para resolver POMRs.

El TS-MBFOA es una metaheurística basada en forrajeo de bacterias, que gracias a una buena calibración de sus parámetros, genera un conjunto de resultados competitivos con costo computacional moderado y en tiempos razonables. PSO es una técnica de optimización basada en enjambre de partículas, tiene una amplia variedad de optimización de tareas e inclusive pocas implementaciones multi-objetivo. Ambas metaheurísticas tienen participación en la optimización de problemas numéricos, en el caso del TS-MBFOA existe su versión original (BFOA) que ha sido aplicada en optimización multi-objetivo al igual que PSO en problemas particulares, sin embargo, no hay un framework de la versión TS-BMFOA (algoritmo modificado para incrementar su eficiencia) junto al PSO como una herramienta integral para resolver POMRs del benchmark y definidos por el usuario.

Para tener una alternativa viable para resolver POMRs con metaheurísticas de IC, se propuso la implementación de TS-MBFOA y PSO. Estos algoritmos serán desarrollados como framework de libre distribución donde serán adaptados al criterio de dominancia de Pareto para la optimización multi-objetivo. En el framework serán incluidos problemas del benchmark para visualizar a primera instancia la calidad de los resultados en comparación con los existentes en la literatura. También se codificó un modelo de inclusión de problemas definidos por el usuario, para permitir la inserción de cualquier problema multi-objetivo modelado como un POMR. La calidad de los resultados puede compararse con algunos de los indicadores clásicos como Hipervolumen (Guerreiro et al., 2021). El experto que utilice esta herramienta para resolver un problema particular, le facilitará la toma de decisiones objetiva por medio del análisis de los resultados presentados en tablas y gráficas.

1.2.2. Delimitación de la investigación

Alcances

- La investigación se enfocará en el uso de TS-MBFOA y PSO como framework para la optimización de POMRs.
- Se ajustó el algoritmo TS-MBFOA para resolver POMRs.

- El framework está compuesto por el algoritmo TS-MBFOA y PSO, el criterio Pareto, problemas del benchmark, y un módulo para la inserción de POMRs definidos por el usuario.
- Los resultados obtenidos a través del framework serán valores no dominados por otros resultados, de acuerdo con la optimalidad de Pareto.
- El framework está disponible para su libre distribución.

Limitaciones

- No se realizará estudios de frameworks de algoritmos evolutivos.
- El framework no aceptara problemas de optimización que no estén modelados como POMR.
- No se realizará estudios sobre sensibilidad de parámetros del TS-MBFOA y PSO.

1.3. Pregunta de investigación e hipótesis

Las preguntas de investigación de esta tesis son:

1. ¿Las soluciones generadas por el framework son competitivos a los existentes en la literatura con los algoritmos evolutivos clásicos?
2. ¿El framework basado en algoritmos de inteligencia colectiva facilita la inclusión de nuevos POMRs?
3. ¿El framework genera resultados competitivos para la mejor toma de decisiones objetiva?

En esta tesis se plantea la siguiente hipótesis:

Es posible que los algoritmos de inteligencia colectiva pueden resolver POMRs de prueba y particulares con una tasa de factibilidad promedio mayor al $20\,\%$ integrados en un framework que facilite la configuración, ejecución y generación de resultados para cualquier usuario final con una tasa de finalización superior o igual a $90\,\%$.

1.4. Objetivo general

Implementar los algoritmos bio-inspirados TS-MBFOA y PSO como framework de libre distribución para simplificar los procesos de solución de POMRs y facilitar la toma de decisiones al tomador de decisiones.

1.5. Objetivos específicos

- Ajustar el TS-MBFOA para resolver POMRs.
- Incorporar la dominancia de Pareto a los procesos de selección de mejores resultados del TS-MBFOA y PSO.
- Desarrollar el framework.
- Comparar los resultados de problemas del benchmark generados por el framework con los algoritmos evolutivos clásicos de la literatura.

1.6. Justificación

Existen diversos problemas de optimización que son clasificados como NP-completos, entre ellos se encuentran los POMR, los cuales se presentan en distintas áreas de estudio. Cada uno de estos problemas, se resumen a expresiones matemáticas que deben ser evaluadas bajo ciertas restricciones (otras expresiones matemáticas) y rangos de variables. Una gran parte de los problemas complejos cumplen estas características, pero todos tienen propiedades diferentes como la dimensión de las variables, los rangos distintos de cada variable, el número y tipo de restricciones que pueden ser de desigualdad o de igualdad lineal o no lineal, y el número de funciones objetivo, que pueden ser lineal, no lineal, cuadrática, entre otras. Además, cada problema tiene propiedades únicas que lo distinguen entre sí, como el número de objetivos o criterios que se deben cumplir.

Debido a la complejidad de los problemas de optimización, se requieren estrategias adecuadas para resolverlos, las cuales varían según la familia de problemas a optimizar. Para problemas

simples, como los lineales y sin restricciones, funcionan bien los algoritmos heurísticos, mientras que para los problemas más complejos con múltiples objetivos y restricciones, se utilizan comúnmente los algoritmos metaheurísticos. Estos últimos son técnicas de búsqueda basadas en principios más generales que ofrecen una exploración más amplia del espacio de soluciones en tiempos razonables.

El TS-MBFOA es un algoritmo popular por su capacidad de resolver problemas complejos de la vida real en tiempos razonables, lo que permite ahorrar costos computacionales y económicos. Sin embargo, actualmente solo es capaz de resolver problemas mono-objetivos y específicos. La manipulación de los parámetros del algoritmo requiere una interacción directa con el código, lo que implica que se necesitan conocimientos de programación para su uso. Esto puede representar una desventaja para quienes no los tienen. PSO es una técnica de optimización basada en enjambre de partículas que ha sido ampliamente estudiada y modificada. Tiene una amplia variedad de aplicaciones para la optimización de tareas, e incluso hay algunas implementaciones multi-objetivo.

Desarrollar un framework de libre distribución como solución alternativa para abordar los POMRs con AIC, consiste en la implementación de dos algoritmos: TS-MBFOA y PSO. Este framework incluye el criterio de dominancia de Pareto para seleccionar el mejor vector de soluciones del POMR, y se incluyó problemas del benchmark para comparar la calidad de los resultados obtenidos con los existentes en la literatura. Además, se creará un modelo que permita la inclusión de problemas definidos por el usuario para que puedan ser insertados como POMRs y resueltos con estos algoritmos.

La implementación de esta herramienta proporciona varios beneficios, como la reducción del tiempo y esfuerzo necesarios para configurar los algoritmos y seleccionar o insertar un POMR. Además, el código está disponible de forma gratuita para la comunidad científica, lo que permitirá realizar pruebas con problemas del benchmark y utilizar las herramientas necesarias para ejecutar el código. Esto hará que el código sea de fácil acceso y esté disponible para cualquiera que desee utilizarlo.

Para concluir, el framework es incluido en una interfaz de usuario amigable, lo que permitirá a aquellos usuarios sin conocimientos de programación interactuar visualmente con las funcionalidades del framework, incluyendo la calibración de los parámetros del algoritmo, la visualización

de resultados, estadísticas básicas y gráficos de convergencia.

1.7. Metodología utilizada

Una investigación se puede abordar desde dos enfoques metodológicos distintos (Paitán et al., 2014). El primer enfoque es cuantitativo y se caracteriza por ser descriptivo, comparativo o relacional. El segundo enfoque es cualitativo y se enfoca en descubrir, explicar o explorar atributos o propiedades no cuantificables. En esta investigación, se ha elegido el enfoque cuantitativo, debido a que se están tratando problemas de optimización y se necesitan comparar los resultados generados entre dos o más grupos y evaluarlos utilizando medidas estadísticas.

Por otra parte, para el desarrollo de la UI del framework, se deben visualizar, especificar, construir y documentar los artefactos que componen al sistema. Para ayuda de estas tareas, se utiliza UML para facilitar el trabajo gracias a su notación de modelado en diagramas.

Para lograr el objetivo de construir un framework del TS-MBFOA y el PSO integrando el criterio de la dominancia de Pareto para resolver POMRs, se realizaron las siguientes tareas:

- Revisión de algoritmos de inteligencia colectiva TS-MBFOA y PSO en problemas Multi-Objetivo.
- Redacción de marco conceptual, referencial y tecnológico del proyecto de investigación.
- Diseño de diagramas con el modelo UML.
- Codificación del algoritmo TS-MBFOA y PSO en un lenguaje de programación de libre distribución.
- Codificar e incluir la técnica de dominancia de Pareto al TS-MBFOA y PSO.
- Codificación de los problemas del benchmark Multi-Objetivo.
- Codificación de la métrica de evaluación Hipervolumen.
- Adaptación del evaluador de expresiones matemáticas.
- Codificación del modelo de inclusión de POMRs.

- Pruebas preliminares del desarrollo (comparación de los resultados del framework con los algoritmos evolutivos clásicos de la literatura).
- Reporte de las pruebas preliminares.
- Tesis: Experimentos y resultados.
- Tesis: Terminar el documento final.
- Registro del software en INDAUTOR.

Para dar una representación visual del flujo de trabajo de la propuesta, en la Figura 1.1, se muestra el diagrama donde los círculos en azul son los POMRs del benchmark integrados en el framework, el círculo en celeste es un POMR ingresado manualmente por el usuario y los círculos en verde son las metaheurísticas TS-MBFOA y PSO.

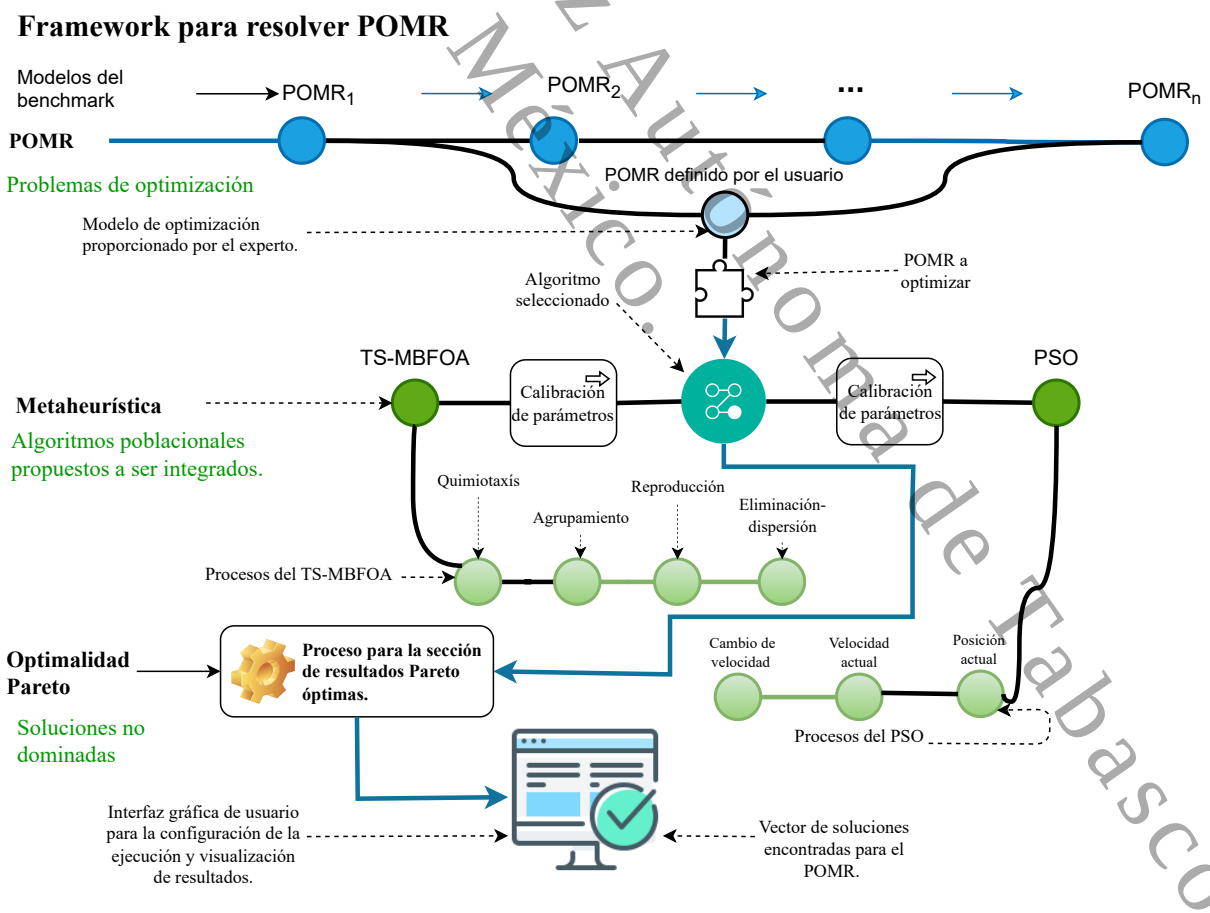


Figura 1.1. Diagrama del flujo de trabajo del framework para la optimización de un POMRs.

Capítulo 2

Marco teórico

En este capítulo, se presentan las teorías fundamentales integradas en la investigación, los modelos de optimización empleados, la clasificación de las metaheurísticas utilizadas, una descripción detallada de los algoritmos TS-MBFOA y PSO, la importancia de la optimalidad de Pareto, las métricas de rendimiento para evaluar la competitividad de los resultados, los trabajos relacionados que existen en la literatura especializada, así como las tecnologías usadas para desarrollar el framework.

2.1. Conceptos y teorías fundamentales de la investigación

2.1.1. Modelo de optimización

La optimización es parte de las matemáticas, consiste en la aplicación y formulación de métodos para ejecutar técnicas de toma de decisión. Existen problemas del mundo real no lineales y de gran dimensión; para resolver esta clase de problemas NP-Completos (Garey et al., 1976) se requiere encontrar soluciones óptimas, por ende, la optimización es un proceso de búsqueda de la mejor solución posible a un problema bajo ciertas circunstancias (Velázquez-Reyes, 2006) y con ella se pretende encontrar una solución que maximice o minimice un determinado valor de un problema de optimización.

Un problema de optimización puede definirse como (S, f), donde S representa un conjunto de soluciones factibles o espacio de búsqueda y $f : S \longrightarrow \mathbb{R}$ la función objetivo a optimizar. La

función objetivo f asigna a cada solución $s \in S$ del espacio de búsqueda un número real que cuantifica el valor de dicha solución. La función objetivo f permite definir una relación de orden entre cualquier par de soluciones en el espacio de búsqueda (El-Ghazali, 2009).

El objetivo principal al resolver un problema de optimización es encontrar una solución óptima global s^* , donde $s^* \in S$ y es un óptimo global si tiene una función objetivo mejor que todas las soluciones del espacio de búsqueda, es decir $\forall s \in S, f(s^*) \leq f(s)$.

Los problemas de optimización se clasifican de acuerdo al número de funciones objetivo, la existencia de restricciones, tipos de variables y clasificación de la función objetivo (Sarker y Newton, 2007), tal como se muestra en la Figura 2.1.

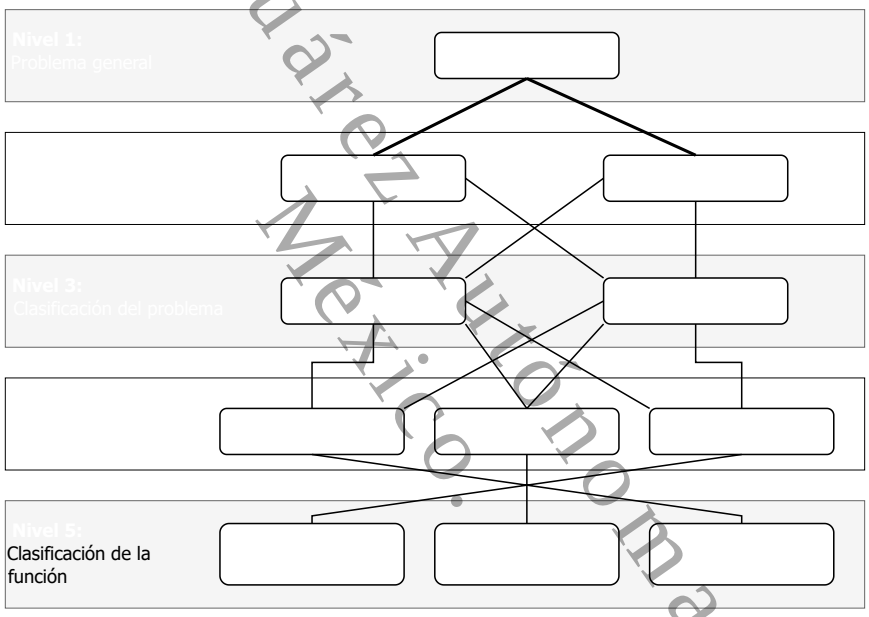


Figura 2.1. Clasificación de los problemas de optimización.

Los problemas de optimización complejos cuentan con características particulares como el manejo de restricciones, ya sea de igualdad o desigualdad lineales o no lineales. En este tipo de problemas las soluciones encontradas no todas son óptimas, las S soluciones válidas se llaman soluciones factibles, mientras que el resto soluciones no factibles.

Existen casos donde hay una función objetivo (por lo general, contiene varias variables) y un conjunto de restricciones que se deben de cumplir, a este tipo de problemas se llaman problemas de optimización mono-objetivo. Si embargo, hay casos donde se tiene más de una función objetivo y un conjunto de restricciones, a este tipo de problemas se les conoce como POMR. Muchos de

los problemas del mundo real tienen restricciones y es desafiante lidiar de forma simultánea con las restricciones y las función objetivo.

Problemas de optimización mono-objetivo

En problemas con una sola función objetivo a optimizar se le conoce como problemas de optimización mono-objetivo. Una estructura general del modelo matemático (también conocido como modelo de programación matemática) (Sarker y Newton, 2007) puede representarse del siguiente modo:

$$\text{minimizar (o maximizar): } f(\vec{x})$$

sujeto a:

$$g_i(\vec{x}) \leq 0, \quad i = 1, 2, ..., m$$
$$h_j(\vec{x}) = 0, \quad j = 1, 2, ..., p$$

donde, f denota la única función objetivo, donde $\vec{x} \in \mathbb{R}^n$. $\vec{x}$ es el vector de soluciones n-dimensional $\vec{x} = [x_1, x_2, x_3, ..., x_n]^T$, donde cada $x_i, i = 1, 2, 3, ..., n$ está delimitada por el límite inferior y superior $L_k \leq x_i \leq U_k, k = 1, 2, ..., D$; D es el número de variables de diseño, m es el número de restricciones de desigualdad y p es el número de restricciones de igualdad (para ambos casos podrían ser restricciones lineales o no lineales). Observe que $\mathbb{R}^n$ contiene todas las posibles x que pueden utilizarse para satisfacer una evaluación de $f(x)$ y sus restricciones. Si denotamos con F la región factible (donde se encuentran todas las soluciones que satisfacen al problema) y con S a todo el espacio de búsqueda, entonces debe ser claro que $F \subseteq S$ (Hernández-Ocaña, Pozos-Parra, Mezura-Montes et al., 2016). A este modelo matemático se le conoce como Problemas de Optimización Numérico con Restricciones (PONR)(Mezura-Montes y Coello, 2011).

Problemas de optimización multi-objetivo

Hoy en día, los problemas de optimización del mundo real implican la optimización simultánea de varios objetivos que compiten entre sí. Este tipo de problemas difiere de un problema mono-

objetivo, dado que los primeros no tienen una única función objetivo, sino 2 o más funciones. En estos problemas no existe una única solución óptima, sino un conjunto de soluciones alternativas. Estas soluciones son óptimas en el sentido que ninguna otra solución del espacio de búsqueda es superior a ellas y se le conocen como soluciones Pareto-óptimas (Coello et al., 2007). De manera general, este tipo de problemas se conocen como POM (también llamado optimización multi-criterio, multi-resultado o problema de optimización vectorial). Como los problemas mono-objetivos, los multi-objetivos cuentan con restricciones donde un vector de variables de decisión que satisface las restricciones y optimiza una función vectorial cuyos elementos representan las funciones objetivo.

Las **variables de decisión** son informaciones numéricas cuyos valores deben elegirse en un problema de optimización. Estas variables se denotan como $x_j, j = 1, 2, ..., n$. El vector $\mathbf{x}$ de n variables de decisión está representado por:

$$\mathbf{x} = [x_1, x_2, ..., x_n]^T,$$

donde T indica la traspuesta del vector en forma columna a un vector fila. En la mayoría de los POM siempre hay restricciones fijadas por las particularidades de la naturaleza del problema. Estas restricciones deben satisfacerse para considerar viable una determinada solución. Todas restricciones describen dependencias entre las variables de decisión y las constantes que intervienen en el problema. Las restricciones se expresan en forma de desigualdad:

$$g_i(\mathbf{x}) \leq 0, \quad i = 1, 2, ..., m,$$

o igualdad:

$$h_j(\mathbf{x}) = 0, \quad j = 1, 2, ..., p,$$

donde, p es el número de restricciones de igualdad, este debe ser menor que n (variables de decisión), ya que si $p \geq n$ entonces el problema está sobrecargado y no tiene grados de libertad para optimizar. El número de grados de libertad viene dado por $n - p$.

En los problemas del mundo real, los objetivos múltiples que se optimizan casi siempre entran en conflicto, y algunas deben minimizarse, mientras que otras se maximizan. Estos criterios de

evaluación se expresan como funciones computables denominadas funciones objetivo. Las funciones objetivo de un POM se denotan como: $f_1(\mathbf{x}), f_2(\mathbf{x}), ..., f_k(\mathbf{x})$, donde k representa el número de funciones objetivo. Las funciones objetivo forman una función vectorial $\mathbf{f}(\mathbf{x})$ definida como:

$$\mathbf{f}(\mathbf{x}) = [f_1(\mathbf{x}), f_2(\mathbf{x}), ..., f_k(\mathbf{x})]^T$$

El conjunto de todas las n-tuplas de números reales denotadas por $\mathbb{R}^n$ se denomina n-**espacio** Euclidiano (Coello et al., 2007). En los POMs se consideran dos espacios:

- El espacio n-dimensional de las variables de decisión en el que cada eje de coordenadas corresponde a un componente del vector $\mathbf{x}$.
- El espacio k-dimensional de las funciones objetivo en el que cada eje de coordenadas corresponde a un vector componente $\mathbf{f}_k(\mathbf{x})$.

En general, los POMRs requieren de la optimización de k funciones objetivo simultáneamente. Esto implica la maximización, minimización o combinación de todas las k funciones. Un POMR se define formalmente como:

$$\text{minimizar (o maximizar): } F(\mathbf{x}) = (f_1(\mathbf{x}), f_2(\mathbf{x}), ..., f_k(\mathbf{x}))$$

sujeto a:

$$g_i(\mathbf{x}) \leq 0, \quad i = 1, 2, ..., m$$
$$h_j(\mathbf{x}) = 0, \quad j = 1, 2, ..., p$$

Una solución de un POMR minimiza (o maximiza) los componentes de un vector $F(\mathbf{x})$, donde $\mathbf{x}$ es un vector variable de decisión n-dimensional, tal que $\mathbf{x} \in \Omega$ donde $\mathbf{x} = (x_1, x_2, ..., x_n)$. m representa las restricciones de desigualdad, p las restricciones de igualdad (ambas pueden ser lineales o no lineales). Las restricciones deben cumplirse al minimizar (o maximizar) $F(\mathbf{x})$ y Ω contiene todos los $\mathbf{x}$ posibles que pueden usarse para satisfacer una evaluación de $F(\mathbf{x})$.

Así, un POMR tiene de k objetivos reflejados en las k funciones objetivo lineales o no lineales, m y p son las restricciones en las funciones objetivo y n variables de decisión.

Existen métodos para resolver problemas de optimización, estos se clasifican en dos categorías: (1) los métodos exactos que obtienen soluciones óptimas y garantizan su optimalidad. Los métodos exactos son una opción para resolver los problemas de optimización simples y sin restricciones, mientras que generalmente no resuelven problemas complejos. (2) Los métodos metaheurísticos son técnicas para resolver los problemas de optimización complejos, generalmente para PONR y POMR, ya que generan soluciones de alta calidad en tiempos razonables, pero no hay garantía de encontrar una solución óptima global. Sin embargo, son las técnicas más efectivas para la búsqueda de soluciones de problemas NP-Completos (El-Ghazali, 2009).

2.1.2. Metaheurísticas

En distintas áreas o disciplinas existen problemas no lineales y de alta dimensión (NP-completos) que son difíciles de encontrar una solución óptima en un tiempo razonable y aceptable. Los métodos de optimización convencionales y clásicos no son eficientes en la búsqueda de soluciones para este tipo de problemas. En la actualidad, existen muchos métodos de optimización que se aplican a diferentes tipos de problemas. En la Figura 2.2, se presenta la clasificación de los métodos de optimización.

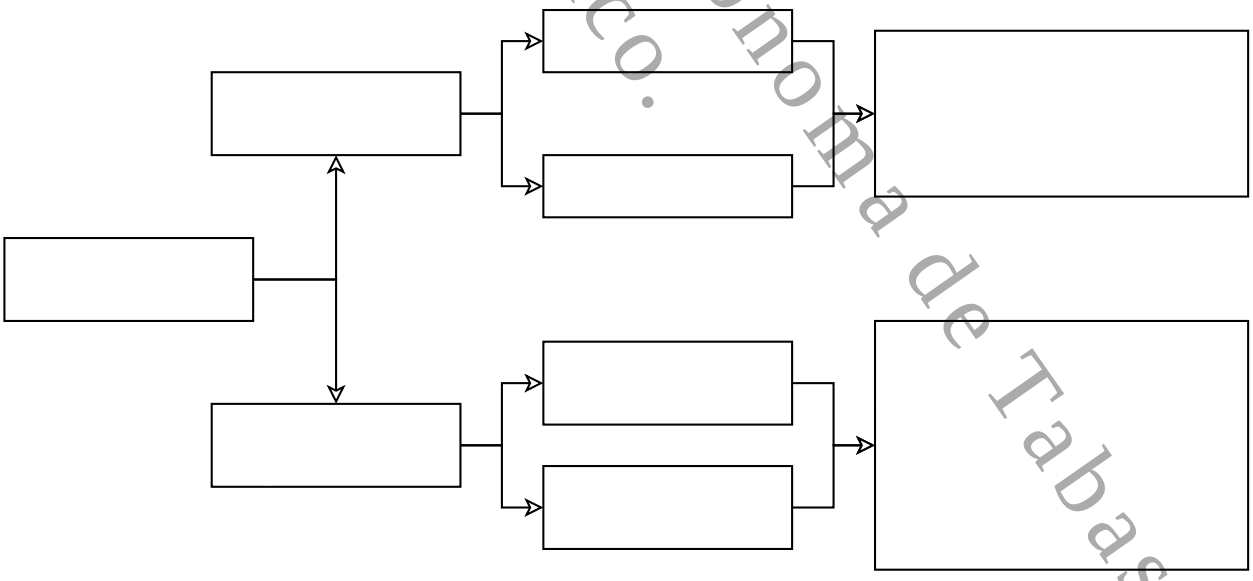


Figura 2.2. Clasificación de métodos de optimización (Rajabi Moshtaghi et al., 2021).

Los métodos exactos son técnicas capaces de alcanzar el óptimo global para problemas lineales, de baja escala y sin restricciones. Sin embargo, al tratarse de problemas complejos de

gran tamaño, aumenta exponencialmente el espacio de búsqueda y resulta imposible encontrar soluciones en un tiempo razonable (Beheshti y Shamsuddin, 2013).

Para la optimización de problemas complicados como los no lineales, los métodos aproximados generan una solución aceptable, pero no garantizan la optimalidad, sin embargo, son técnicas capaces de generar resultados de buena calidad y en tiempos aceptables. Estos algoritmos se basan en búsqueda poblacional y se clasifican en dos grupos:

1. Los algoritmos heurísticos, que su procedimiento de resolver un problema de optimización lo realiza mediante una aproximación intuitiva (R. Martí, 2003); muchos de ellos fueron diseñados para problemas específicos sin posibilidad de generalización o aplicación a otros problemas similares.

2. Los métodos metaheurísticos son algoritmos que se aplican a diversos problemas de optimización, y se conocen como técnicas de aproximación. Estas técnicas buscan soluciones aceptables en el espacio de búsqueda utilizando el mecanismo de combinación de exploración y explotación (Blum y Roli, 2003). La exploración se centra en buscar de mejores soluciones en regiones del espacio de búsqueda, lo que incluye aquellas zonas cercanas a la mejor solución encontrada hasta el momento. Por otro lado, la explotación utiliza la información disponible para buscar soluciones que estén más cerca de la óptima, lo que significa explorar regiones del espacio de búsqueda que no han sido exploradas.

   Las metaheurísticas incluyen varios mecanismos para evitar la convergencia prematura y pueden utilizar desde técnicas de búsqueda local hasta métodos de aprendizaje avanzados (Abdel-Basset et al., 2018). En general, las metaheurísticas son una estructura algorítmica flexible que se adapta bien a una variedad de problemas de optimización y proporciona soluciones aceptables en un tiempo razonable.

La mayoría de las metaheurísticas son inspiradas en la naturaleza y se basan en principios de evolución biológica (Fister Jr et al., 2013). Existen dos paradigmas más utilizados para la optimización, estos son:

1. **Los algoritmos evolutivos (EA):** simulan la progresión biológica de la evolución de las especies manejando operadores de selección, cruce, mutación y reproducción para generar

mejores soluciones candidatas. Algunos de los algoritmos históricos de AE son: la programación genética (Koza, 1990), los algoritmos genéticos (Sampson, 1976), las estrategias evolutivas (Rechenberg, 1973) y la programación evolutiva (Fogel, 1998).

2. **Los Algoritmos de Inteligencia Colectiva (AIC):** imitan el comportamiento colectivo de diferentes especies de una comunidad. La inteligencia colectiva se basa principalmente en actualizar las soluciones candidatas mediante la interacción local entre sí y su entorno. Los AIC emulan el comportamiento colaborativo de especies simples e inteligentes como: un enjambre de partículas (PSO) (Kennedy y Eberhart, 1995), abejas (ABC) (Karaboga y Basturk, 2007), hormigas (ACO) (Dorigo et al., 1996), bacterias (BFOA) (Passino, 2002), entre otros.

Las metaheurísticas generan soluciones buenas en tiempos razonables, sin embargo, no todos los valores generados son factibles a la solución del problema. Por lo tanto, es importante conocer que existen dos regiones dentro el espacio de búsqueda (Figura 2.3), el óptimo local (x^0) y el óptimo global (x^1).

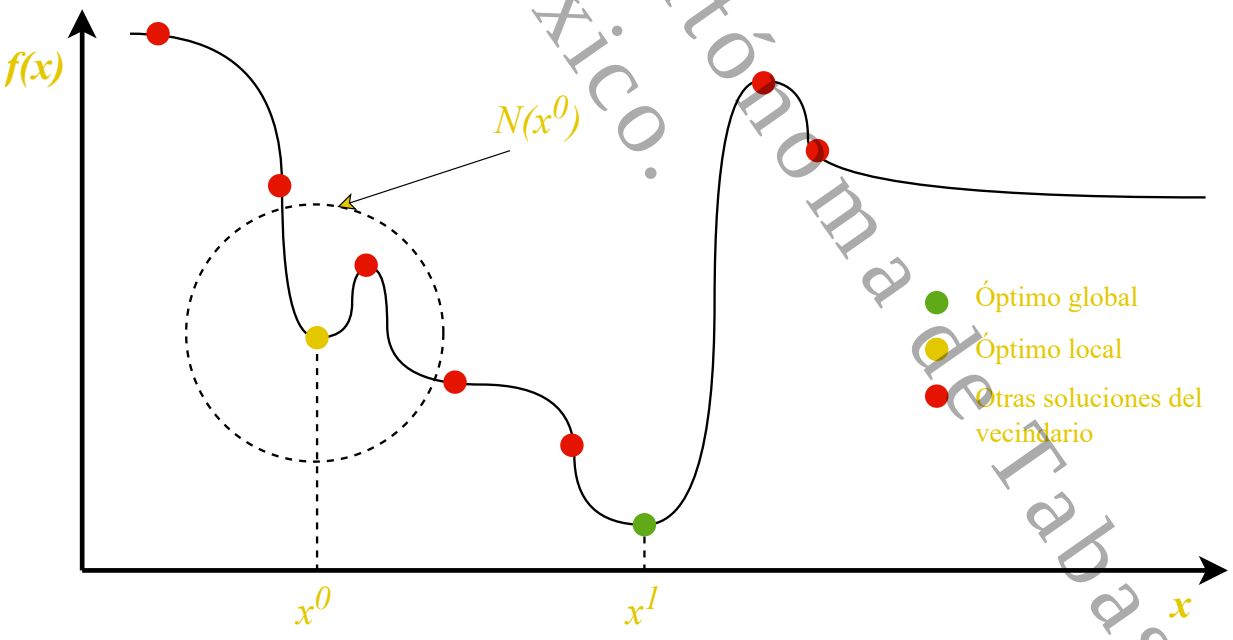


Figura 2.3. Óptimo local y óptimo global. Nota: fuente extraída de (Vélez y Montoya, 2007).

En la Figura 2.3, se muestra la búsqueda del valor x que minimiza la función $f(x)$, donde $N(x)$ representa el vecindario de x. El método de optimización durante la búsqueda llega a x^0

y el método no acepta soluciones de inferior calidad en $N(x^0)$, por ende, se queda atrapado en el óptimo local. Las metaheurísticas tratan de evitar esta situación al aceptar eventualmente por diversos mecanismos soluciones de menor calidad, para mejorar la probabilidad de llegar a el óptimo global $x1$.

En los últimos años, la implementación de algoritmos metaheurísticos ha aumentado para generar soluciones a este tipo de problemas y son utilizados ampliamente en diversos campos de estudio. A continuación, se describen dos AIC implementados en este trabajo para la solución de POMR.

2.2. Algoritmo basado en el forrajeo de bacterias

En los años 70, surgen ideas de optimización a través del comportamiento de quimiotaxis de las bacterias (H. Bremermann, 1974) y a finales de los años 80 (H. J. Bremermann y Anderson, 1989), fueron aplicadas las primeras implementaciones en el entrenamiento de una red neuronal. Años más tarde, Passino, 2002 propone la metaheurística que simula el proceso completo del forrajeo de las bacterias Escherichia coli (Figura 2.4), este proceso es:

1. **Quimiotáxis:** Las bacterias realizan movimientos de nado y giro de manera aleatoria evadiendo sustancias nocivas.

2. **Agrupamiento:** Las bacterias encuentran un pozo de nutrientes y se comunican entre las bacterias mediante segregación de sustancia.

3. **Reproducción:** Las mejores bacterias se reproducen de manera asexual, donde una parte de la bacteria se separa y crece una bacteria idéntica.

4. **Eliminación-dispersión:** Las peores bacterias se eliminan, si embargo una parte de ella se dispersa y se integra en el mapa de nutrientes.

La propuesta de esta metaheurísticas fue llamada Algoritmo de Optimización Basado en el Forrajeo de Bacterias (BFOA, por sus siglas en inglés) y como muchos AIC, tiene sus parámetros propios e independientes del problema a optimizar, donde los parámetros de entrada son presentados en la Tabla 2.1.

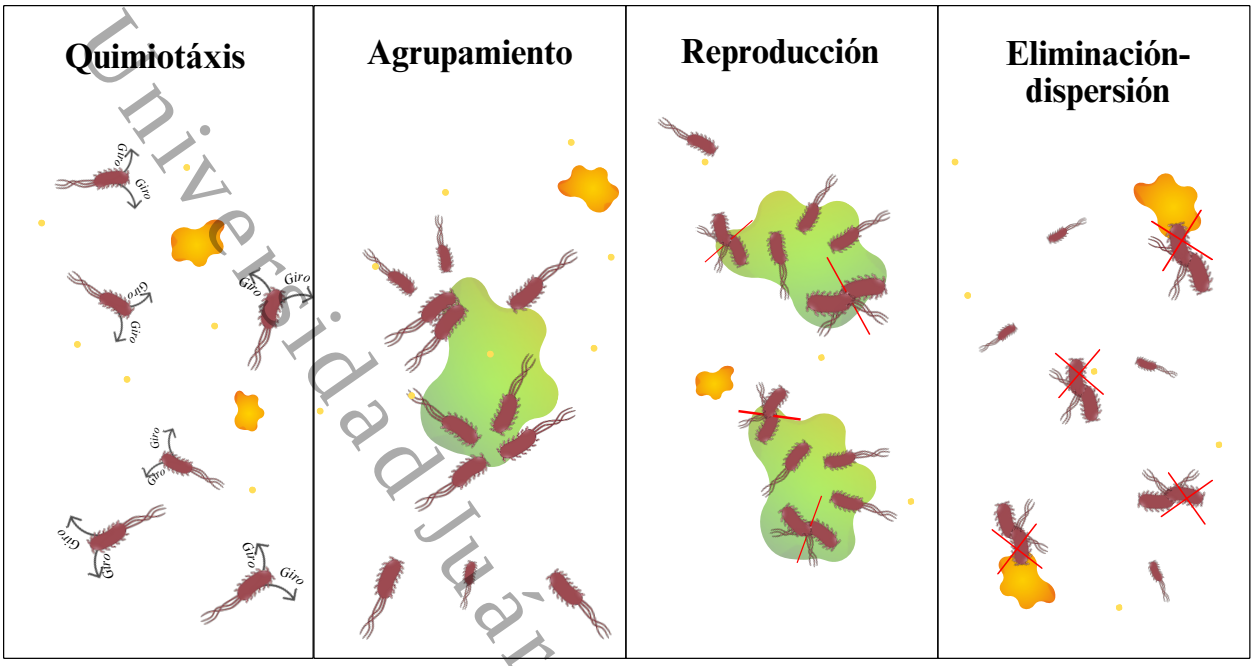


Figura 2.4. Búsqueda de alimento o forrajeo de las bacterias Escherichia coli.

Parámetro	Descripción
S_b	Número de bacterias.
N_c	Límite del paso de quimiotaxis.
N_s	Límite del paso de nado.
N_{re}	Límite del ciclo de reproducción.
S_r	Número de bacterias a reproducir.
N_{ed}	Límite del ciclo de eliminación-dispersión.
C_i	Tamaño de paso.
P_{ed}	Probabilidad de eliminación-dispersión.

Tabla 2.1. Parámetros del BFOA.

Los parámetros y ciclos que utiliza el BFOA son numerosos, esto hace que sea un algoritmo complejo, produciendo un número elevado de evaluaciones. Existen modificaciones implementadas al BFOA original, el cual adaptan mecanismos que permiten al algoritmo resolver problemas de optimización con restricciones (Mezura-Montes y Hernández-Ocaña, 2008, 2009). Esta modificación es llamada Modified-BFOA, y sus mejoras son las siguientes:

1. Un solo ciclo que incluye el proceso de quimiotaxis, reproducción y eliminación-dispersión.
2. Tamaño de paso diferente para cada variable de diseño.
3. Manejo de restricciones del problema de optimización empleando reglas de factibilidad

(Deb, 2000).

4. Comunicación entre las bacterias y la mejor bacteria de la población de una generación.

En la Tabla 2.2, se presenta los parámetros de entrada de MBFOA.

Parámetro	Descripción
S_b	Número de bacterias.
N_c	Límite del paso de quimiotaxis.
S_r	Número de bacterias a reproducir.
β	Factor de escalamiento.
R	Porcentaje del tamaño de paso.
$GMAX$	Número de generaciones.

Tabla 2.2. Parámetros del MBFOA.

Años más tarde, MBFOA sufre modificaciones de sus procesos con la finalidad de mejorar el rendimiento del algoritmo; la propuesta fue llamada Two-Swing-MBFOA (Hernández-Ocaña, Pozos-Parra et al., 2016). En su formulación, cada bacteria i representa una solución potencial y se denota como $\theta^i(j, G)$, donde j es el ciclo quimiotáxico y G es el ciclo de generación completo de los cuatro procesos del forrajeo de bacterias detalladas a continuación:

1. **Proceso de quimiotaxis:** En este proceso se alternan dos nados diferentes, el nado de explotación y el nado de exploración. En cada ciclo se realiza un nado, comenzando con el nado de explotación (nado clásico). Sin embargo, una bacteria no necesariamente intercalará exploración y explotación en los nados, ya que si la nueva posición de un nado dado, $\theta^i(j+1, G)$ tiene una mejor aptitud (basada en las reglas de factibilidad) que la posición original $\theta^i(j, G)$, otro nado en la misma dirección se llevará a cabo en el siguiente ciclo. De lo contrario, un nuevo giro será calculado. Este proceso se detiene después de N_c intentos.

   El nado de exploración usa la mutación entre bacterias y es calculado con la Ecuación 2.1:

$$\theta^i(j+1, G) = \theta^i(j, G) + (\beta)(\theta^r_1(j, G) - \theta^r_2(j, G)) \tag{2.1}$$

   donde $\theta^r_1(j, G)$ y $\theta^r_2(j, G)$ son dos bacterias seleccionadas aleatoriamente de la población S_b. β define la nueva posición de una bacteria respecto a la posición de la mejor bacteria,

es decir, representa la zona donde una bacteria puede moverse. Este parámetro lo define el usuario con valor en $[0, 2]$ y se utiliza en el operador de agrupamiento.

El nado de explotación es calculado con la Ecuación 2.2:

$$\theta^i(j+1, G) = \theta^i(j, G) + C(i, G)\phi(i) \tag{2.2}$$

donde $\phi(i)$ es un giro para la bacteria y es calculado con el operador de giro original de BFOA definido en la Ecuación 2.3:

$$\phi(i) = \frac{\Delta(i)}{\sqrt{\Delta(i)^T \Delta(i)}} \tag{2.3}$$

donde $\Delta(i)^T$ es un vector aleatorio generado con elementos entre $[-1, 1]$. $C(i, G)$ es el tamaño de paso aleatorio de cada bacteria actualizado con la Ecuación 2.4:

$$C(i, G) = R * \Theta(i) \tag{2.4}$$

donde $\Theta(i)$ es un vector aleatorio de tamaño n con elementos dentro del rango de cada variable de decisión: $[L_k, U_k]$, $k = 1, ..., n$. R es un parámetro para escalar el tamaño de paso definido por el usuario con valor cercano a cero (por ejemplo $5.00e-04$). La inicial $C(i, 0)$ se genera utilizando $\theta(i)$. Este tamaño de paso aleatorio permite que las bacterias se muevan en diferentes direcciones dentro del espacio de búsqueda y evita la convergencia prematura, como sugiere Kasaiezadeh et al., 2014.

2. **Proceso de agrupamiento:** En el ciclo medio del proceso de quimiotaxis es aplicado el operador de agrupamiento con la Ecuación 2.5:

$$\theta^i(j+1, G) = \theta^i(j, G) + \beta(\theta^B(G) - \theta^i(j, G)) \tag{2.5}$$

donde $\theta^i(j+1, G)$ es la nueva posición de la bacteria i. $\theta^B(G)$ es la posición de la mejor bacteria de la generación. β el parámetro llamado factor de escalamiento que regula la cercanía de la bacteria i de la mejor bacteria θ^B. Sin embargo, si una solución viola el

límite de las variables de decisión $L_i \leq x_i \leq U_i$, una nueva solución de x_i es generada aleatoriamente entre los límites inferior y superior de cada variable de decisión.

3. **Proceso de reproducción:** En este proceso se realiza lo siguiente:
   - Se ordena las bacterias con base a las reglas de factibilidad.
   - Se eliminan las peores bacterias $S_b - S_r$.
   - Se duplican las mejores bacterias cada cierto número de generación. La frecuencia de reproducción es un parámetro definido por el usuario $RepCycle$.
4. **Proceso de eliminación-dispersión:** Se elimina la peor bacteria de la población $\theta^w(j, G)$ (basado en las reglas de factibilidad) y se genera una nueva bacteria de manera aleatoria.

El pseudocódigo de TS-MBFOA es presentado en el Algoritmo 1.

Algoritmo 1: Pseudocódigo del TS-MBFOA.

```
Generar una población inicial (vector aleatorio) de bacterias θ^i(j,0) ∀i, i = 1,...,S_b.
Evaluar f(θ^i(j,0)) ∀i, i = 1,...,S_b.
for G = 1 to GMAX do
    for i = 1 to S_b do
        for j = 1 to N_c do
            En el proceso de quimiotaxis intercalar los nados con las Ecuaciones 2.1 y 2.2.
            Aplicar el operador de agrupamiento con la Ecuación 2.5 usando β para la bacteria θ^i(j,G).
        end
    end
    if G mod RepCycle == 0 then
        Ordenar la población con base a las reglas de factibilidad.
        Realizar el proceso de reproducción.
        Eliminar las peores bacterias S_b − S_r.
        Duplicar las mejores bacterias.
    end
    Realizar el proceso de eliminación-dispersión eliminando a la peor bacteria θ^w(j,G) de la población actual
    considerando las reglas de factibilidad.
    Actualizar el vector de tamaño de paso usando la Ecuación 2.4.
end
```

En la Tabla 2.3, se presenta los parámetros de entrada del TS-MBFOA.

Parámetro	Descripción	Límites
S_b	Número de bacterias.	[10,500]
N_c	Límite del paso de quimiotaxis.	[1,S_b]
S_r	Número de bacterias a reproducir.	[1,$\frac{S_b}{2}$]
β	Factor de escalamiento.	[0,2]
R	Porcentaje del tamaño de paso.	[0,1]
$RepCycle$	Frecuencia de reproducción.	[1,100]
$GMAX$	Número de generaciones.	[100, -]

Tabla 2.3. Parámetros del TS-MBFOA y sus intervalos de valores.

2.3. Algoritmo de Optimización de Enjambre de Partículas

El algoritmo de Optimización de Enjambre de Partículas (PSO, por sus siglas en inglés) fue propuesto por Kennedy y Eberhart, 1995 convirtiéndose en una técnica ampliamente estudiada para resolver problemas de optimización global. PSO es inspirado en el comportamiento cooperativo y social de diversas especies como aves, peces e incluso seres humanos.

Los mecanismos del algoritmo PSO es el siguiente:

1. La población de enjambre de NP partículas son llamadas soluciones potenciales y se encuentran distribuidas al azar en un espacio de búsqueda.
2. Las partículas actualizan su posición en el espacio de búsqueda con una velocidad determinada, buscando una mejor fuente de alimento (solución óptima).
3. Las partículas mantienen una memoria que les permite recordar su posición anterior.
4. Cada posición de las partículas se distinguen como mejor posición personal y mejor posición global.

El proceso de PSO es de naturaleza estocástica y utiliza la memoria de cada partícula, así como los conocimientos obtenidos por el enjambre en su población para encontrar la mejor solución óptima. Según Abdel-Basset et al., 2018, PSO mediante el procedimiento de búsqueda, las soluciones candidatas (partículas) se actualizan en el espacio n-dimensional según una posición x_i y una velocidad v_i. La nueva velocidad se calcula de acuerdo a la Ecuación 2.6:

$$v_i^{t+1} = wv_i^t + \alpha_1\epsilon_1[pbest_i^t - x_i^t] + \alpha_2\epsilon_2[gbest^t - x_i^t] \tag{2.6}$$

donde i es un índice que representa una partícula y t es un índice de tiempo discreto. El resto de la Ecuación se detalla a continuación:

- v_i^{t+1} es la velocidad actualizada de la partícula i en el siguiente paso de tiempo.
- w es una constante positiva definida por el usuario que representa la inercia y controla el impacto de velocidad de una partícula al encontrar una nueva posición.
- α_1 es una constante cognitiva definida por el usuario que controla cuánto afecta la mejor posición personal al movimiento de la partícula.
- α_2 es una constante social definida por el usuario que controla el impacto de lo mejor global en el movimiento de la partícula.
- ϵ_1 y ϵ_2 son vectores aleatorios uniformes entre $[0, 1]$ que permiten mejorar la exploración y pueden ayudar a prevenir la convergencia prematura.
- v_i^t es la velocidad de la partícula i en el tiempo t.
- $pbest_i^t$ es un mejor valor personal de la partícula i en el tiempo t.
- $gbest^t$ es un mejor valor global en el tiempo t.
- x_i^t es la posición actual de la partícula i.

La actualización de posición de la partícula se define con la ecuación 2.7:

$$x_i^{t+1} = x_i^t + v_i^{t+1} \tag{2.7}$$

La Figura 2.5 resume como una partícula x_i^t (resaltada con color azul) recalcula su nueva posición tomando en cuenta los siguientes componentes: la mejor posición personal que ha tenido en su recorrido (posición en amarilla), la mejor posición global que tiene la mejor partícula (partícula en verde) y el efecto de inercia w. Estos tres componentes vectoriales se combinan para generar la nueva posición de la partícula de la siguiente iteración.

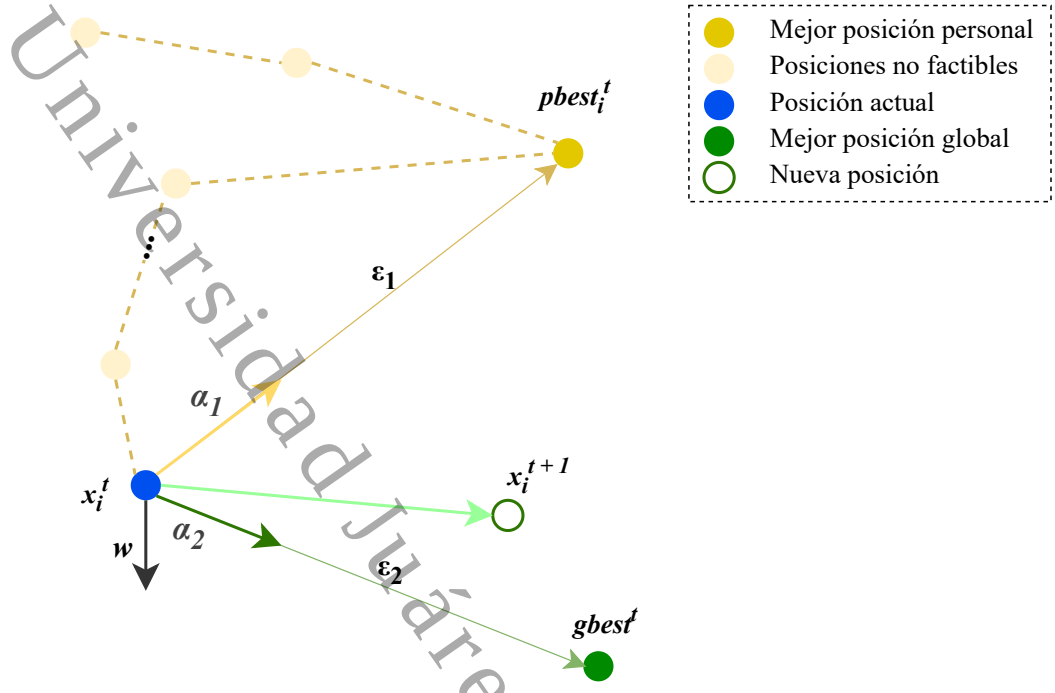


Figura 2.5. Movimiento de una Partícula.

El pseudocódigo de PSO es presentado en el Algoritmo 2.

Algoritmo 2: Pseudocódigo del PSO (Abdel-Basset et al., 2018).

```
Generar una población de partículas, i = 1, 2, ..., particulas.
Generar un vector de velocidades de tamaño particulas.
while criterio de parada do
    for i = 1 to particulas do
        Calcular el valor de aptitud f(x_i).
        if valor de aptitud es mejor que pbest_i. Ecuación 2.9 then
            Set pbest_i = valor de aptitud actual.
        end
        if pbest_i es mejor que gbest then
            Set gbest = pbest_i.
        end
    end
    for i = 1 to particulas do
        Calcular la velocidad usando la Ecuación 2.6.
        Actualizar la posición de la partícula i usando la Ecuación 2.7.
    end
end
```

En la Tabla 2.4, se presenta los parámetros de entrada del algoritmo PSO.

Parámetro	Descripción	Límites
NP	Número de partículas en el enjambre.	[20-100]
α_1	Coeficiente cognitivo.	[0.5-2]
α_2	Coeficiente social.	[0.5-2]
w	Coeficiente de inercia.	[0-1]

Tabla 2.4. Intervalos de los parámetros del PSO según Zielinski y Laur, 2006.

2.4. Optimalidad Pareto

En un POMR existen varias funciones objetivo, el cual, el propósito es encontrar soluciones de equilibrio en lugar de una única solución. En la búsqueda de soluciones se llega a un conjunto de las soluciones no dominadas en el espacio de los objetivos. La optimalidad Pareto se ocupa de combinar la búsqueda y la toma de decisiones para encontrar las soluciones equilibradas para varios objetivos donde la solución de un objetivo no empeore a otro. El concepto de óptimo adoptado habitualmente en la optimización multi-objetivo es el propuesto por Vilfredo Pareto en 1986 y denominada optimalidad de Pareto definida como (Cagnina et al., 2005):

Un vector de decisión $\vec{x}^* \in \Omega$ es Pareto óptimo si $\forall \vec{x} \in \Omega$ para todo $I = \{1, 2, ..., k\}$ o bien:

$$\forall i \in I(f_i(\vec{x}^*) \leq f_i(\vec{x})) \tag{2.8}$$

y, hay al menos un $i \in I$ tal que

$$f_i(\vec{x}^*) < f_i(\vec{x}) \tag{2.9}$$

El vector objetivo $\vec{x}^*$ es óptimo de Pareto si no existe otro vector factible $\vec{x}$ que reduzca algún objetivo sin causar un aumento simultáneo en al menos otro objetivo.

Las soluciones encontradas son un vector de variables y para evaluar la calidad de las soluciones se utiliza el criterio de dominancia de Pareto.

Un vector $\vec{x} = (x_1, x_2, ..., x_k)$ domina a $\vec{y} = (y_1, y_2, ..., y_k)$ denotado por $\vec{x} \preceq \vec{y}$, si y solo si $\vec{x}$ es parcialmente inferior a $\vec{y}$, es decir, $\forall i \in \{1, 2, ..., k\} : x_i \leq y_i$ y, al menos para una i, $x_i < y_i$. Por ejemplo, un vector A pertenece al conjunto de soluciones no dominadas (frente de Pareto) si no existe ninguna solución B que mejore algún objetivo sin empeorar al menos otro (Figura 2.6).

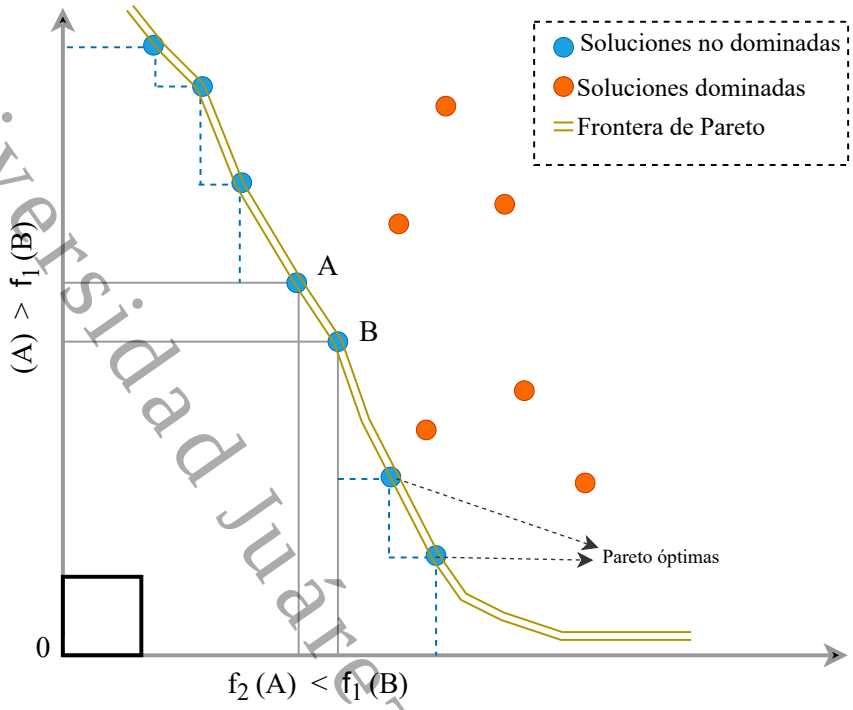


Figura 2.6. Dominancia y frontera de Pareto (López, 2013).

En la Figura 2.6 se visualiza que, entre dos soluciones A y B en el frente de Pareto, no se puede afirmar que una sea estrictamente mejor que la otra, ya que pueden ocurrir los siguientes casos:

- A es mejor que B en algunos objetivos,
- B es mejor que A en otros,
- A y B son equivalentes en ciertos aspectos, o
- A y B resultan ser incomparables.

Para un problema multi-objetivo: $\vec{f}(x)$, el conjunto óptimo de Pareto, denotado por $\mathcal{P}^*$ o $\mathcal{P}_{true}$ se define como la Ecuación 2.10:

$$\mathcal{P}^* = \{x \in \Omega | \nexists' \in \Omega \vec{f}(x') \preceq \vec{f}(x)\} \tag{2.10}$$

El frente óptimo de Pareto dado $\vec{f}(x)$ y el conjunto óptimo de Pareto $\mathcal{P}^*$ o $\mathcal{PF}_{true}$ es definida por la Ecuación 2.11 y se visualiza en la Figura 2.6:

$$\mathcal{PF}^* = \{\vec{y} = \vec{f} = (f_1(x), f_2(x), ..., f_k(x)) | x \in \mathcal{P}^*\} \tag{2.11}$$

2.5. Métrica para medir la calidad de soluciones

Las métricas de rendimiento son técnicas de comparación para medir la calidad de los resultados generados por diferentes algoritmos. Las métricas son muy importantes de aplicar en el campo de optimización, existiendo un número considerable de métricas (Riquelme et al., 2015).

Para comparar el desempeño entre las salidas generadas por TS-MBFOA y PSO, se debe seleccionar la métrica apropiada para realizar las comparaciones. Tomando en cuenta que las métricas consideran tres aspectos de un conjunto de soluciones (Riquelme et al., 2015) como:

- La cercanía al frente óptimo de Pareto (convergencia).
- La distribución y extensión (diversidad).
- El número de soluciones.

Considerando los tres aspectos, y con base a la literatura, se propone utilizar la métrica: Hipervolumen, el cual es la métrica más usadas de acuerdo a la comparación realizada en (Riquelme et al., 2015). A continuación, se detalla brevemente.

Hipervolumen (HV) es una métrica de rendimiento ampliamente usada en la comparación de resultados de algoritmos multi-objetivo. Esta métrica mide el tamaño del espacio objetivo dominado por las soluciones, integrando de forma implícita la precisión, diversidad y cardinalidad del conjunto. Como indica Guerreiro et al., 2021, es la única métrica unaria que incorpora estos tres aspectos en un único valor cuantitativo.

2.6. Literatura relacionada

En la literatura especializada, se ha explorado la implementación de PSO y BFOA original para abordar POMR, como parte integral de un framework. Entre las propuestas más recientes, se encuentran:

El estudio realizado por Mejía-de-Dios y Mezura-Montes, 2022, se enfoca en el desarrollo del framework Metaheuristics, que implementa diversos algoritmos metaheurísticos, como Evolución Diferencial, PSO, Algoritmo Genético, NSGA-II, entre otros. Metaheuristics está diseñado para resolver problemas de optimización global con restricciones. Destacando el *Evolutionary Centers Algorithm* (ECA) como uno de los optimizadores implementados. La codificación se llevó a cabo en el lenguaje de programación Julia, simplificando el proceso de configuración de los algoritmos para facilitar su manipulación. Aunque se ejemplifica con un problema de un solo objetivo de diez dimensiones, se señala que Metaheuristics fue diseñado para abordar problemas de optimización de un objetivo o múltiples. Metaheuristics es una propuesta atractiva para la comunidad evolutiva, sin embargo, su limitación radica en su dependencia de conocimiento de programación en Julia, un lenguaje que se encuentra fuera de los primeros treinta lugares más utilizados según el *TIOBE Index*[1]. En el presente trabajo, se propone la codificación del framework en un lenguaje de programación común, orientado a objetos y de distribución libre, para permitir ampliar el acceso y la utilidad del framework.

El trabajo realizado por Blank y Deb, 2020 consiste en el desarrollo del framework de optimización multiobjetivo pymoo. La investigación aborda la falta de frameworks completos en el ámbito de la ciencia de datos y el aprendizaje automático, presentando pymoo como una propuesta de solución. Los autores desarrollaron el framework y presentan un escenario de optimización multi-objetivo restringido, destacando la flexibilidad de pymoo al estar codificado en el lenguaje de programación Python. Utilizaron algoritmos como GA, DE, NSGA-II, NSGA-III, MOEAD, entre otros, mostrando la capacidad de adaptación de pymoo a diferentes problemas de optimización, incluido un conjunto de problemas de prueba integrados para validar su implementación. En la investigación actual, se propone el desarrollo de una interfaz gráfica de usuario que permita la exploración visual del conjunto de problemas, los resultados, la configuración de los algoritmos y la exportación de los resultados. Esta diferencia ofrece a la comunidad científica y académica una herramienta accesible para sus investigaciones y aplicaciones.

La investigación presentada en Biscani y Izzo, 2020, detalla el desarrollo del framework pagmo, una herramienta científica diseñada para abordar la optimización mul-tiobjetivo paralelo. Este framework incorpora implementaciones de algoritmos evolutivos, así como métodos matemáticos

[1] https://www.tiobe.com/tiobe-index/

como el método simplex, SQP, y métodos de puntos interiores, entre otros. La implementación del framework se llevó a cabo en el lenguaje de programación C++. Además, los autores presentan una extensión llamada pygmo para aquellos usuarios que prefieren utilizar Python. En su investigación, los autores realizaron pruebas de pagmo y pygmo utilizando el problema de optimización de la esfera unidimensional, configurando una población de 20 individuos. Además, destacan la importancia de codificar el problema específico que se va a optimizar para utilizar eficazmente los algoritmos implementados en los frameworks. En el estudio actual, se propone la incorporación de nuevos problemas de optimización a través de una interfaz gráfica de usuario interactiva y amigable. Para usuarios que prefieran utilizar el framework sin la interfaz, se permite la entrada de problemas a través de cadenas de texto, eliminando la necesidad de programar el problema, lo cual reduce el esfuerzo a usuarios con pocos conocimientos en programación.

Una investigación más es la presentada en Benítez-Hidalgo et al., 2019, donde los autores se enfocaron en el desarrollo del framework jMetalPy para abordar problemas de optimización multi-objetivo mediante técnicas metaheurísticas. jMetalPy es una adaptación de las funciones de jMetal (Durillo y Nebro, 2011b) al lenguaje de programación Python, incluyendo la implementación de PSO, NSGA-II y sus variantes, así como técnicas para abordar problemas restringidos y dinámicos. Este framework incluye un conjunto de problemas de uno y múltiples objetivos. Al estar codificado en Python, resalta por características como la visualización de gráficas, pruebas estadísticas y soporte para optimización dinámica mediante el uso de bibliotecas como Matplotlib. Aunque jMetalPy está disponible en GitHub bajo la licencia MIT y representa una propuesta interesante en la comunidad científica, la falta de una interfaz amigable que simplifique su utilización y acelere las pruebas y configuraciones de los algoritmos genera una limitación significativa.

Otras implementaciones de frameworks diseñados para abordar problemas de optimización multi-objetivo se pueden observar en la Tabla 2.5, donde se incluyen trabajos recientes relacionados con la propuesta de esta tesis.

Framework	Algoritmos	Lenguaje	Mono-obj.	Multi-obj.	Distribuido	Visual
MultiObjective Algorithms (Gandibleux et al., 2023)	MOMIP, MOLP, MOIP, MOCO	Julia	×	✓	✓	×
NLopt (Ypma et al., 2018)	Algoritmos heurísticos como: CRS, MLSL, StoGO, AGS, ISRES, ESCH, etc.	C, C++, Fortran, MATLAB, GNU Octave, Python, etc.	✓	×	✓	×
BlackBox Optim (Feldt, 2022)	NES, DE, RS, SPSA, BorgMOEA	Julia	✓	✓	✓	×
CMAEvolution Strategy (Brea, 2022)	ECA, DE, PSO, GA, MOEA/D-DE, NSGA-II, NSGA-III, CCMO, etc.	Julia	✓	✓	✓	×
Metaheuristics (Mejía-de-Dios y Mezura-Montes, 2022)	ECA, DE, PSO, GA, MOEA/D-DE, NSGA-II, NSGA-III, CCMO, etc.	Julia	✓	✓	✓	×
DESDEO (Misitano et al., 2021)	NIMBUS síncrono, RVEA, NSGA-III, E-NAUTILUS	Python	×	✓	✓	×
pymoo (Blank y Deb, 2020)	GA, DE, BRKGA, NSGA-II, R-NSGA-II, NSGA-III, MOEAD, etc.	Python	✓	✓	✓	✓
Inspyred (Tonda, 2020)	GA, ES, PSO, ACO, SA, PAES, NSGA-II	Python	✓	✓	✓	×
Geatpy (Jazzbin, 2020)	DE, EE, GA, awGA, MOEA/D, NSGA-II, PPS-MOEA/D-DE, RVEA, NSGA-III, RVEA*	Python	✓	✓	✓	×
pagmo, pygmo (Biscani y Izzo, 2020)	DE, Simplex, SQP	C++, Python	✓	✓	×	×
JMetalPy (Benítez-Hidalgo et al., 2019)	GA, EA, SPEA2, NSGA-II, NSGA-III, SMPSO, GDE3, OMOPSO, etc.	Python	×	✓	✓	✓
DEAP (Kim y Yoo, 2019)	GP, GA, NSGA-II, SPEA2	Python	×	✓	✓	×
JMetalSP (Barba-González et al., 2018)	NSGA-II, MOCell	Java	×	✓	✓	×
PlatEMO (Tian et al., 2017)	SPEA2, PSEA-II, NSGA-I, NSGA-II, NSGA-III, MOEA/D, SMS-EMOA, MSOPS-II, AGE-II, etc.	MATLAB	×	✓	✓	✓
ECJ (Luke, 2017)	GA, PSO, DE, NSGAII y SPEA2, CMA-ES, etc.	Java	✓	✓	✓	×

Tabla 2.5. Frameworks existentes para la optimización multi-objetivo.

Existen varios frameworks implementados para la optimización multi-objetivo, sin embargo, la mayoría implementan los algoritmos evolutivos. Algunos incorporan la codificación del PSO, pero los AIC son poco explorados en estas implementaciones. La mayoría de los frameworks mencionados no implementan una interfaz amigable que facilite la exploración de las diversas utilidades que ofrecen. Además, algunos están desarrollados en lenguajes poco comunes o lenguajes de

programación comercial.

La mayoría de estos frameworks son gratuitos, pero algunos no comparten su código o tienen instalaciones complicadas. Además, los algoritmos basados en el forrajeo de bacterias aún no se incluyen en ninguno de estos frameworks. Sin embargo, existen frameworks específicos diseñados para abordar problemas particulares, como es el caso de la optimización multi-objetivo en la entropía celular (Yi et al., 2016), la optimización multi-objetivo para mejorar la presión de selección hacia el frente de Pareto (Chen et al., 2018), o la optimización multi-objetivo con aprendizaje profundo (Zhou et al., 2023).

2.7. Marco tecnológico

El desarrollo de un sistema de software está sujeto a utilizar herramientas de Ingeniería de Software Asistidas por Computadora, que ayudan a realizar tareas específicas para la implementación del framework para resolver POMR. Las herramientas a utilizar son de fácil acceso y gratuitas. A continuación, se describe las tecnologías usadas y su importancia para este desarrollo.

2.7.1. Framework y Lenguaje de programación

Las metaheurísticas se han convertido en un paradigma importante para resolver POMR. Cada metaheurísticas tiene sus propias características como la calibración de parámetros propios que se deben configurar para una mejor obtención de resultados. Generalmente, las metaheurísticas se desarrollan de forma individual con poca o ninguna reutilización de código. Esto provoca esfuerzos excesivos en la codificación, ya que se debe de ingresar en cada desarrollo el problema que se necesita optimizar.

El uso de framework de alto nivel es una herramienta valiosa para ayudar en la implementación de nuevas ideas, facilitando la reutilización de código de algoritmos existentes y para comprender el comportamiento de las técnicas (Lau et al., 2004). Además, permite al diseñador de algoritmos centrarse en la investigación o experimentación algorítmica, reduciendo el esfuerzo de tiempo en la implementación del código (Durillo y Nebro, 2011b).

Por otra parte, existe una larga lista de lenguajes de programación libres, multiplataforma y orientados a objetos como: Java, C++, Python, JavaScript, PHP, C#, entre otras. Cada lenguaje

tiene sus características como el uso del paradigma de la Programación Orientada a Objetos (POO) que ayuda a dominar la complejidad de un problema que se desee programar. Está basada en varias técnicas como herencia, abstracción, polimorfismo, acoplamiento y encapsulamiento, permitiendo agrupar bibliotecas o librerías llamadas framework. En la POO se crean clases para agrupar objetos que son un conjunto de variables (datos) y métodos (funciones) relacionados entre sí (Álvarez-Díaz, 2011).

La codificación del framework y la interfaz gráfica de usuario se implementó en un lenguaje de programación libre y multiplataforma, que permita el paradigma de POO con el fin separar por clases y funciones todos los procesos requeridos de cada algoritmo a implementar. Por lo tanto, se propone utilizar el lenguaje de programación de Java en la versión gratuita OpenJDK 17.0.2, gracias a sus características como: ser un lenguaje de uso general, simple, orientado a objetos, distribuido, robusto, seguro, de arquitectura neutra, portable, multiplataforma, multitarea, dinámico y cuenta con una amplia comunidad que día a día lanzan mejoras al lenguaje.

2.7.2. Lenguaje de Modelado Unificado

Para el desarrollo del framework y la UI se debe visualizar, especificar, construir y documentar los artefactos que componen al sistema de software. Hacer esto es una tarea complicada, debido a la dimensión objetiva que tiene el desarrollo del sistema integral. Por ello, se propone utilizar UML para identificar los requerimientos y necesidades del framework por medio de la notación de modelado con reglas sintácticas, semánticas y prácticas (Pressman, 2010).

UML proporciona trece diagramas diferentes para modelar un sistema, sin embargo, en el desarrollo del framework y la UI de esta investigación, se implementaron seis diagramas señalados en (Pressman, 2010), el cual son los siguientes:

Diagrama de casos de uso: Mediante actores y casos de uso que se asocian al usuario como el elemento de interacción, ayudan a determinar la funcionalidad y características del sistema desde la perspectiva del usuario. En particular, un caso de uso describe la interacción del sistema con el usuario que lo utiliza, definiendo los pasos requeridos para lograr una meta específica. Por lo tanto, el diagrama de casos de uso es un panorama de todos los casos de uso y sus relaciones que proporciona un gran panorama de la funcionalidad del sistema.

Diagrama de clases: Es una técnica útil para modelar las clases que contenga el sistema, debido a que la POO encapsula todo sus atributos, operaciones, relaciones y asociaciones con otras clases. Este diagrama aporta una visión estática o de estructura del sistema sin mostrar la naturaleza dinámica de las comunicaciones entre los objetos de las clases.

Diagrama de secuencias: Se utiliza para mostrar las comunicaciones dinámicas entre objetos durante la ejecución de una tarea. Este tipo de diagrama muestra el orden temporal de la ejecución entre los objetos para lograr una tarea específica.

Diagrama de actividades: Muestra el comportamiento dinámico de una parte o todo el sistema a través del flujo de control entre acciones que realiza. Es similar a un diagrama de flujo, excepto porque un diagrama de actividad puede mostrar flujos que coinciden en varias tareas.

Diagrama de comunicación: Para mostrar las relaciones entre los objetos y visualizar la comunicación que existe entre ellos, los diagramas de comunicación son una opción, puesto que, proporciona otro indicio del orden temporal de las comunicaciones, pero enfatiza las relaciones entre los objetos y clases en lugar del orden temporal.

Diagrama de implementación: Para la estructura de un sistema y para mostrar la distribución física entre diferentes plataformas de hardware o software, se necesita crear diagramas de implementación, debido a que programar un sistema en un lenguaje de programación multiplataforma requiere de la visualización estática de cada entorno de ejecución.

Capítulo 3

Diseño y desarrollo del framework

3.1. Modelado UML

Para el desarrollo del framework, se diseñaron diagramas UML con el propósito de visualizar los diversos componentes del sistema. Estos diagramas facilitan la identificación de la estructura, el funcionamiento y las interacciones presentes en el sistema. A continuación, se describen los diagramas UML utilizados para visualizar los requisitos del framework en la optimización multi-objetivo con restricciones.

Diagrama de Caso de Uso

En la Figura 3.1, se presenta el diagrama de caso de uso que visualiza la interacción entre el usuario y el sistema durante el proceso de una nueva ejecución en el framework para resolver problemas de optimización.

El actor principal de este caso de uso es el tomador de decisiones (usuario final), quien inicia la ejecución del sistema con el propósito de resolver un problema multi-objetivo utilizando el framework. El actor puede realizar las siguientes acciones:

Elegir el tipo de problema: El usuario tiene la opción de seleccionar un problema de optimización mono-objetivo o multi-objetivo de los predefinidos en el framework o definir un problema personalizado.

Seleccionar optimizador: El usuario puede elegir entre dos algoritmos de optimización: El PSO

o el TS-MBFOA. Al elegir un algoritmo, también podrá configurar los parámetros propios del algoritmo seleccionado.

Seleccionar un número de iteraciones independientes: El usuario puede especificar el número de iteraciones independientes del algoritmo durante la ejecución del framework en la optimización de un problema seleccionado o ingresado.

Visualizar resultados: Una vez terminada la ejecución del algoritmo, el usuario puede acceder a visualizar los resultados, entre ellos las soluciones de Pareto óptimas, estadísticas, gráficas boxplot, y métricas de rendimiento como hipervolumen y epsilon.

Exportación de resultados: El usuario final tiene la opción de exportar el reporte de resultados en una hoja de cálculo.

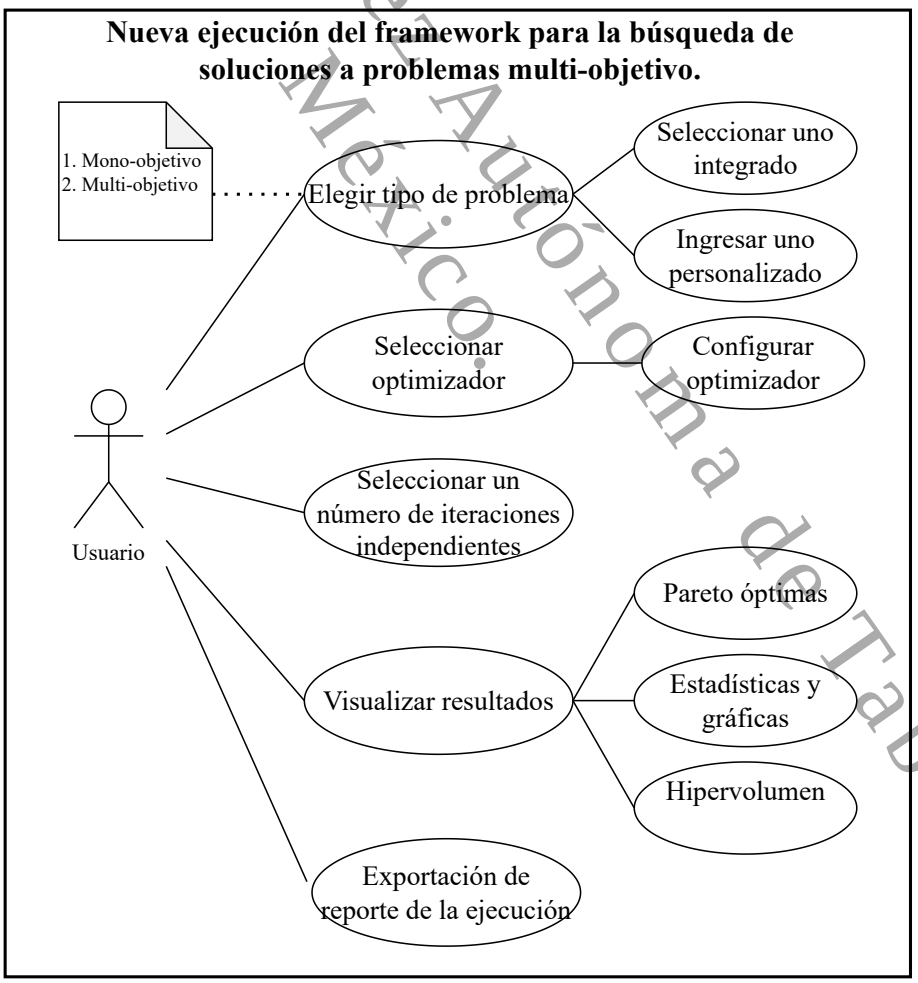


Figura 3.1. Diagrama de Caso de Uso del framework.

Diagrama de Clases

La Figura 3.2, muestra un diagrama de clases que sirve como base del framework diseñado para la optimización multi-objetivo. Este diagrama presenta las clases que proporcionan funcionalidad y estructura al sistema. Entre las clases representadas se incluyen:

TSMBFOA y PSO: Representan la codificación del algoritmo optimizador y todos sus procesos.

Pareto: Esta clase funciona para mantener y gestionar el conjunto de soluciones de Pareto-óptimas encontradas durante la ejecución del framework.

Crowding: Esta clase gestiona el cálculo de la distancia de agrupamiento de las soluciones.

CNOP: Representa el modelo matemático de un problema mono-objetivo y multi-objetivo con restricciones.

MetaheuristicBase: Es una clase padre que incluye los atributos genéricos de una metaheurística, esta clase esta diseñada para poder ser heredada en nuevas implementaciones de algoritmos.

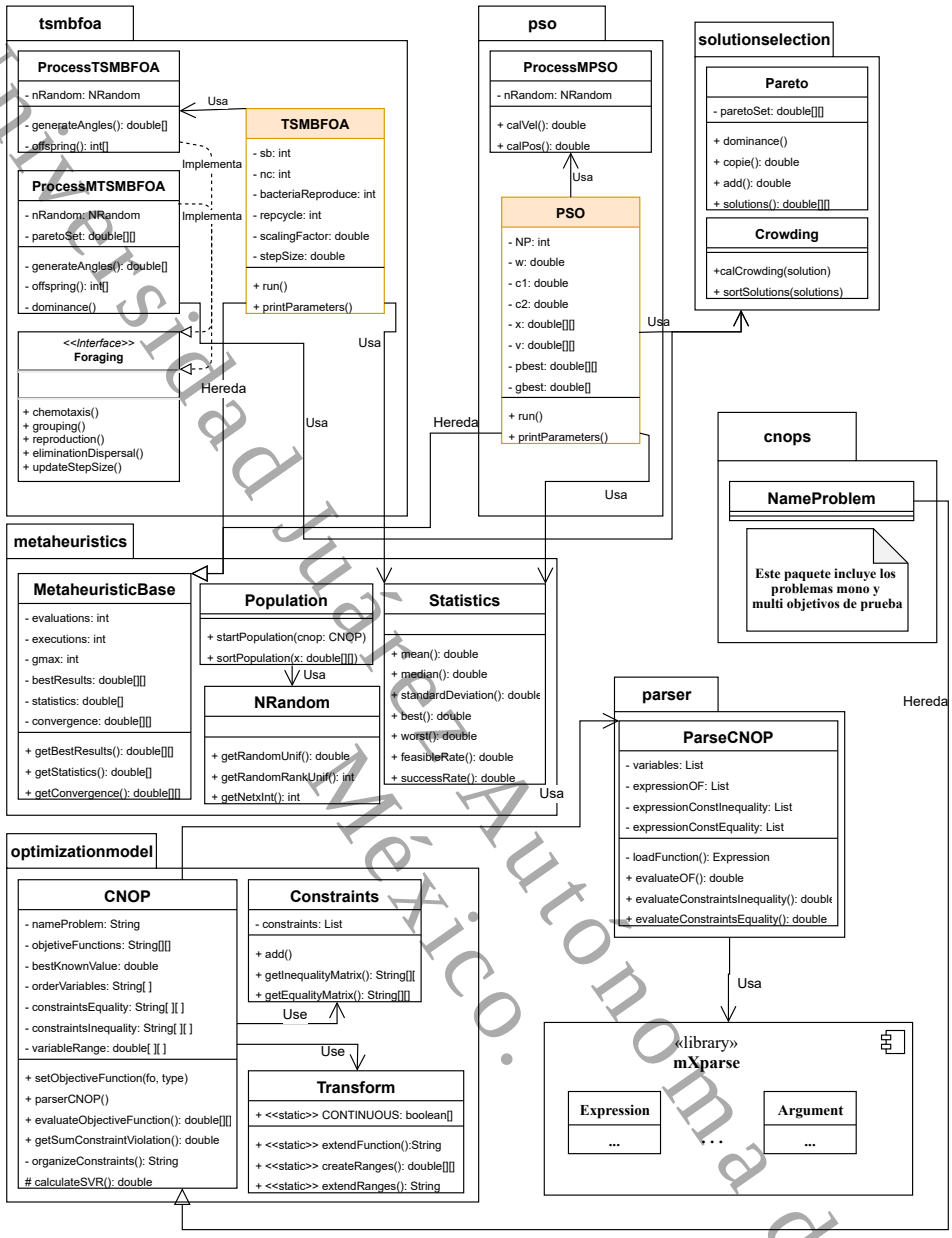


Figura 3.2. Diagrama de Clases del framework.

Diagrama de Secuencias

En La Figura 3.3, se muestra el diagrama de secuencia donde el actor principal es el usuario final que busca optimizar un problema de optimización.

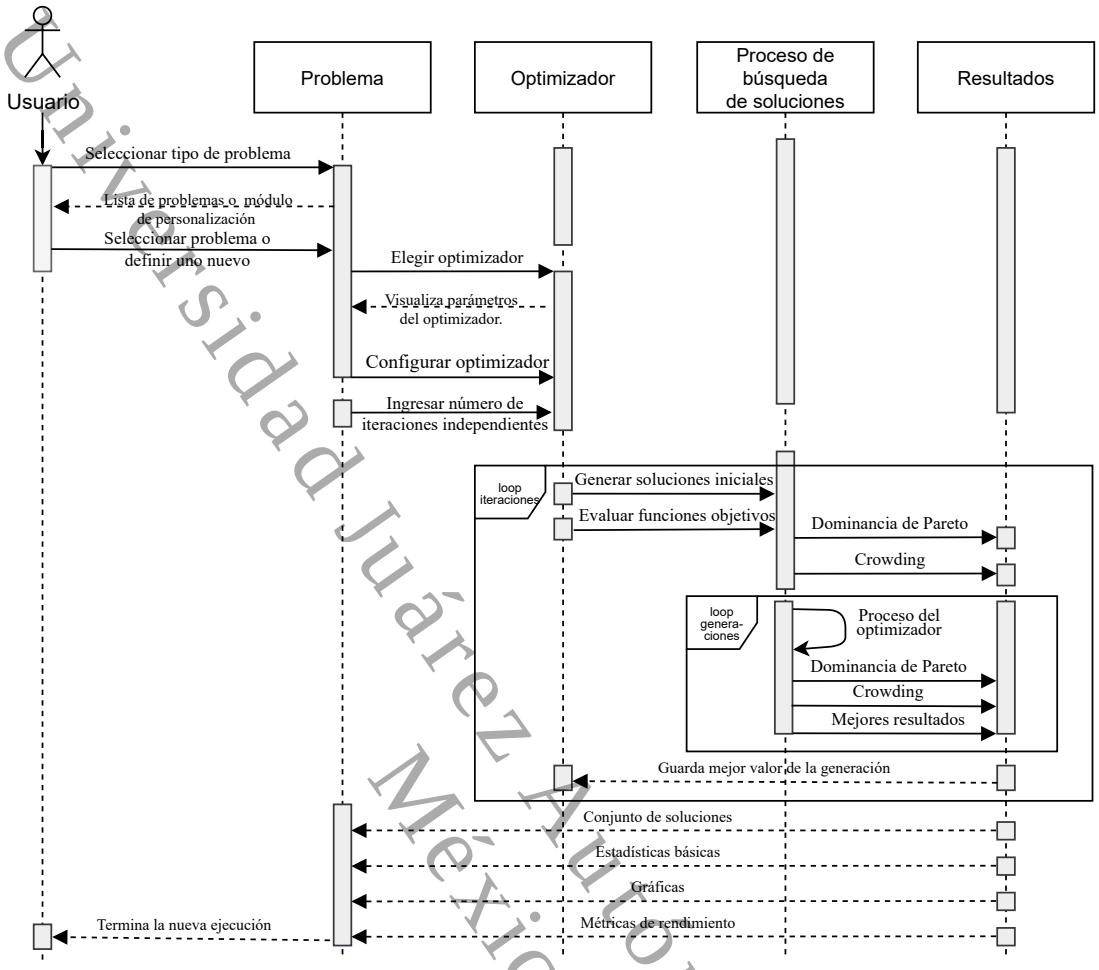


Figura 3.3. Diagrama de Secuencias del framework.

El proceso comienza con la interacción del usuario al seleccionar el tipo de problema de optimización, ya sea mono-objetivo o multi-objetivo. El usuario puede elegir un problema predefinido o ingresar uno personalizado. Además, puede seleccionar un algoritmo de optimización y ajustar sus parámetros. También, el usuario puede ingresar el número de iteraciones independientes que se ejecutarán el algoritmo.

Después de configurar los detalles iniciales, el algoritmo optimizador elegido inicia su proceso. Para cada iteración ingresada se generan soluciones iniciales, que son evaluadas en la o las funciones objetivo del problema. En caso de encontrar soluciones factibles inicialmente, se realiza el proceso de dominancia de Pareto y el cálculo de distancia utilizando la medida Crowding.

Posteriormente, se inicia un bucle generacional donde el algoritmo realiza sus operaciones, generando nuevas soluciones óptimas y aplicando nuevamente los procesos de dominancia de

Pareto y cálculo de distancia. Al finalizar cada generación, se selecciona la mejor solución obtenida. Al completar todas las iteraciones, el usuario tiene la opción de visualizar los resultados.

3.2. Procesos adicionales para la optimización multi-objetivo

Los problemas multi-objetivos tienen diferentes características, el cual se destacan, los no restringidos y los restringidos. Existen algoritmos inicialmente diseñados para la búsqueda de soluciones de los primeros, sin embargo, en la vida real los problemas de optimización restringidos son los más estudiados. Esto se debe a que las restricciones añaden complejidad al problema al limitar el espacio de búsqueda de soluciones. Para el manejo de las restricciones, con la metaheurística 1 y 2, integrados en el framework, se incorpora el mecanismo de manejo de restricciones propuesto por Deb, 2000. A continuación, el Algoritmo 3, visualiza el proceso utilizado para la Suma de Violación de Restricciones (SVR).

Algoritmo 3: Proceso para el manejo de las restricciones. El valor de eps es un número pequeño de tolerancia 10^{-4}.

```
Evaluar g(x⃗) ∀i, i = 1, ..., m. y h(x⃗) ∀j, j = 1, ..., p.
svr = 0
for i = 1 to m do
    if g_i > 0 then
        svr = svr + g_i
    end
end
if p < n then
    for j = 1 to p do
        h_j = |h_j| − eps
        if h_j > 0 then
            svr = svr + h_j
        end
    end
end
else
    El problema está sobre cargado y no tiene grados de libertad para optimizar (Coello et al., 2007).
end
if svr == 0 then
    x⃗ es factible para todas las restricciones.
end
```

Resolver problemas multi-objetivo implica cambios en diferentes mecanismos de las meta-heurísticas, y uno de ellos es el criterio de selección. Para los algoritmos implementados en este trabajo, se incorpora la Dominancia de Pareto como un elemento del proceso de selección descrito en la Sección 2.4. La Dominancia de Pareto se encarga de identificar el frente de Pareto, que representa las soluciones no dominadas en el espacio de los objetivos. El proceso de selección del frente de Pareto se presenta en el Algoritmo 4.

Algoritmo 4: Proceso de selección de soluciones no dominadas. La entrada de datos es el conjunto de soluciones $\mathcal{P}$ con vectores de objetivos $\vec{f}(\vec{x})$ y la salida es el Frente de Pareto $\mathcal{PF}^*$ con soluciones no dominadas.

```
1  PF* = {}
2  for Cada solución x en P do
3      x es no dominado ← 1 //Verdadero
4      for Cada solución y en P do
5          //Ecuación 2.8
6          if y domina a x para todas las f then
7              x es no dominado ← 0 //Falso
8              break bucle interno
9          end
10     end
11     if x es no dominado then
12         Agregar x a PF*
13     end
14 end
```

En el Algoritmo 4, se evalúa cada solución en el conjunto $\mathcal{P}$, donde cada solución está asociada a un vector de objetivos $\vec{f}(\vec{x})$. La Dominancia de Pareto se verifica mediante la comparación de cada par de soluciones en un bucle anidado.

Si una solución $\vec{x}$ no es dominada por ninguna otra solución en el conjunto $\mathcal{P}$, se determina como una solución no dominada y se agrega al conjunto $\mathcal{PF}^*$, que representa el frente de Pareto de soluciones no dominadas. Este proceso asegura que las soluciones en $\mathcal{PF}^*$ son óptimos de Pareto, cumpliendo con la condición de que ninguna otra solución en $\mathcal{P}$ puede mejorar un objetivo sin empeorar al menos uno de los otros.

En la optimización multi-objetivo, el elitismo es un proceso muy utilizado para mantener a los mejores resultados de una población en las generaciones futuras y para asegurar que las

soluciones de alta calidad no se pierdan y puedan mejorar en las generaciones siguientes. En la literatura especializada, se encuentran varios enfoques de elitismo, entre los más usados son detallados en la Tabla 3.1.

Enfoque	Ventajas	Desventajas
Uso de archivo externo	Almacenamiento de soluciones no dominadas, manteniendo diversidad y calidad.	Costos de almacenamiento.
Selección de mejores individuos	Mejora la calidad de las soluciones.	Riesgo de disminuir diversidad y convergencia prematura.
Técnicas de reemplazo	Efectivas para mantener a los mejores individuos de la población.	Riesgo de disminuir diversidad y estancamiento en óptimos locales.

Tabla 3.1. Enfoques de elitismo utilizados en la optimización multi-objetivo (Groşan et al., 2003).

En este trabajo, se implementa el enfoque de archivo externo para gestionar las soluciones no dominadas en la optimización multi-objetivo. Este enfoque ha sido seleccionado debido a su capacidad de mantener los mejores resultados a lo largo de las generaciones y no afectar la convergencia del algoritmo. El archivo externo almacenará solo las soluciones factibles no dominadas, y los resultados están basados en este conjunto (Figura 3.4).

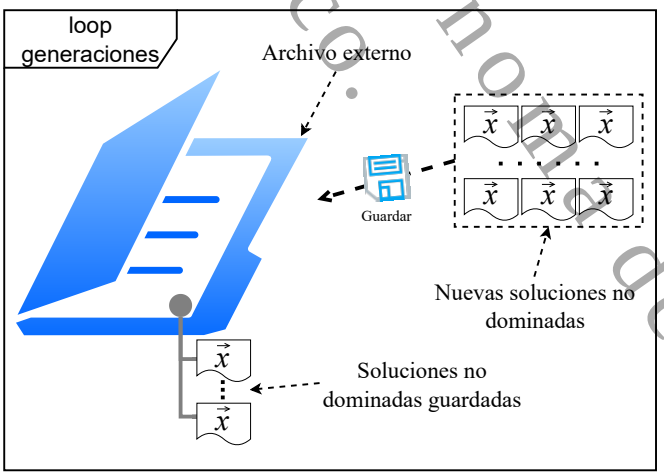


Figura 3.4. Archivo externo para guardar las soluciones no dominadas.

Las metaheurísticas empleadas en la optimización multi-objetivo destacan su capacidad para identificar diferencias entre individuos asignados al mismo frente. Sin embargo, las soluciones cercanas en el espacio de búsqueda tienden a ser similares y, por ende, no contribuyen signi-

ficativamente a la diversidad poblacional. Entonces, para mantener la diversidad en el espacio objetivo, se introduce el Estimador de Densidad basado en la Distancia de Crowding, una técnica utilizada para mantener la diversidad en una población de soluciones no dominadas (L. Martí et al., 2018).

Cabe mencionar que hay alternativas como la Distancia Euclidiana, la cual implica calcular la distancia geométrica entre dos puntos en el espacio objetivo. Por otro lado, Crowding determina la distancia en función de la clasificación de las soluciones según su proximidad en el espacio objetivo. La operación de Crowding generalmente presenta una complejidad computacional lineal y es eficiente en términos de tiempo de ejecución y en problemas de mayor dimensionalidad (Deb et al., 2003). Por esta razón, Crowding se ha incluido en este trabajo como una estrategia para mantener la diversidad en la población. En el Algoritmo 5, se presenta el procedimiento para calcular la distancia de Crowding, a partir de las soluciones no dominadas almacenadas en el archivo externo.

Algoritmo 5: Cálculo de la distancia de Crowding. Donde X es el archivo externo donde están guardadas las soluciones no dominadas.

```
1  l = tamaño(X)
2  if l > 2 then
3      for i = 1 to l do
4          X[i]_distancia = 0
5      end
6      for k = 1 to número de objetivos do
7          X = ordenar(X, k)
8          X[1]_distancia = ∞
9          X[l]_distancia = ∞
10         for j = 2 to l − 1 do
11             X[j]_distancia = X[j]_distancia + (X[j+1][k] − X[j−1][k]) / (X[l][k] − X[1][k])
12         end
13     end
14 end
```

3.3. Diagrama de flujo del framework

Para la búsqueda de soluciones a POMR se requiere la integración de procesos que aborden la complejidad del problema. Para cada algoritmo implementado en este trabajo, se incorpora la suma de violación de restricciones, la identificación de soluciones no dominadas para formar el conjunto óptimo de Pareto, la gestión de soluciones factibles no dominadas a través de un archivo externo, y la aplicación de la estrategia de cálculo de la distancia de Crowding para mantener la diversidad en poblaciones no dominadas.

En la Figura 3.5, se visualiza los diagramas de flujo para la solución de un problema de optimización desde el framework y el proceso general del optimizador, empleando los procesos adicionales integrados en cada algoritmo incluido en el framework. A continuación, se describe los diagramas presentados.

- **Diagrama de flujo A:** Proceso general de actividades para la solución de problemas de optimización.

  El diagrama de flujo A, presenta una visión paso a paso del proceso general para abordar un problema de optimización. Desde la elección del tipo de problema hasta la visualización de los resultados. Un proceso importante en este diagrama es el inicio del optimizador, el cual, se detalla en el diagrama de flujo B.

- **Diagrama de flujo B:** Flujo general de alto nivel del proceso de búsqueda de soluciones con un optimizador seleccionado.

  El diagrama de flujo B, abarca el flujo general de alto nivel del proceso que lleva a cabo la búsqueda de soluciones mediante un optimizador seleccionado. Incluye los mecanismos clave mencionados anteriormente para la gestión de las soluciones óptimas generadas por la metaheurística elegida. Este diagrama proporciona una perspectiva global del proceso de optimización desde la generación de soluciones iniciales, hasta la visualización de los resultados.

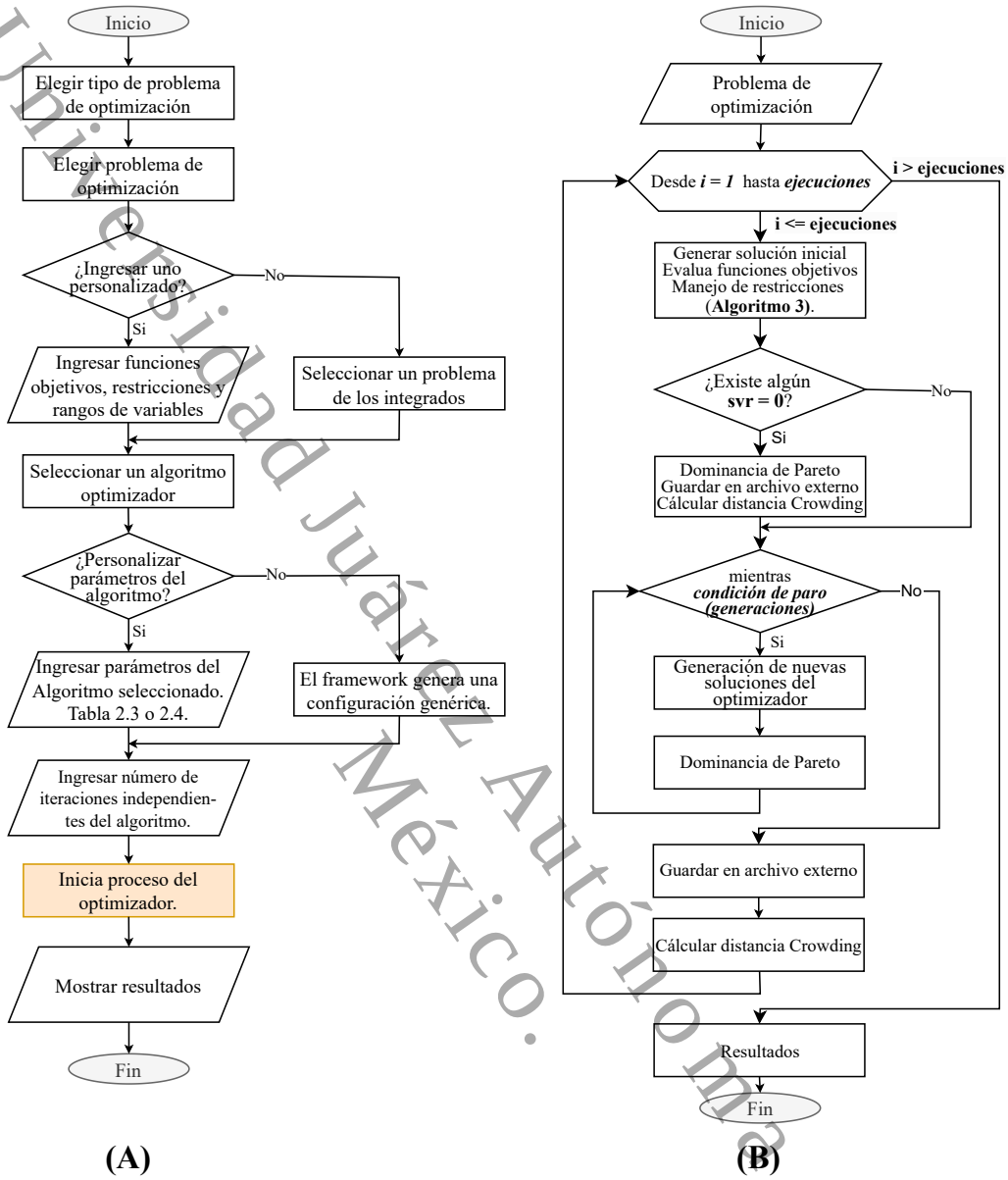


Figura 3.5. Diagramas de flujo del framework. El diagrama de flujo **A**, representa las actividades del proceso de selección de un problema de optimización y el diagrama de flujo **B**, visualiza el proceso general de búsqueda de soluciones de las metaheurísticas.

3.4. Algoritmo general para la solución de POMR

Una vez identificado los procesos necesarios para la optimización de problemas multi-objetivo, se define que, aunque cada metaheurística posee sus procedimientos particulares, existe una serie de actividades compartidas al buscar soluciones a POMR. Estas actividades pueden generalizarse y representarse de manera sistemática mediante un pseudocódigo.

El Algoritmo 6, visualiza de manera integral el proceso de optimización, abarcando aspectos fundamentales como el manejo de restricciones, la gestión de soluciones no dominadas, la utilización de un archivo externo y el cálculo de la distancia de Crowding. Esta representación proporciona una guía general para la búsqueda de soluciones óptimas a POMR, independientemente de la metaheurística seleccionada por el usuario final.

Algoritmo 6: Algoritmo General para la Solución de POMR.

```
for i = 0 hasta el número de iteraciones do
    x⃗ = generar soluciones iniciales dentro de los límites especificados para cada x_i.
    Evaluar F[f_1(x⃗), f_2(x⃗), ..., f_k(x⃗)]
    Gestionar restricciones (svr) usando el Algoritmo 3.
    if Existe alguna x⃗ con svr = 0 then
        Aplicar Algoritmo 4 para la selección de soluciones no dominadas
        Guardar en el archivo externo soluciones no dominadas según la Figura 3.4.
        Calcular las distancias Crowding con el Algoritmo 5.
    end
    while condición de parada do
        for j = 1 hasta n do
            Buscar nuevas soluciones para x_j con los procesos de la metaheurística seleccionada.
            Evaluar que la nueva solución x_j esté dentro del límite inferior y límite superior.
        end
        //Con el nuevo x⃗
        Evaluar F[f_1(x⃗), f_2(x⃗), ..., f_k(x⃗)]
        Gestionar restricciones (svr) usando el Algoritmo 3.
        Comparar si la nueva solución es factible usando las reglas de factibilidad (Deb, 2000).
        Si la metaheurística lo requiere, ordenar la población.
        Aplicar Algoritmo 4 para la selección de soluciones no dominadas
    end
    Guardar en el archivo externo soluciones no dominadas según la Figura 3.4.
    Calcular las distancias Crowding con el Algoritmo 5.
end
```

El proceso de cada generación para la búsqueda de soluciones óptimas depende de los mecanismos específicos de la metaheurística seleccionada. En este trabajo, el usuario final tiene la flexibilidad de elegir entre dos enfoques: el TS-MBFOA (ver Algoritmo 1) o el PSO (ver Algoritmo 2). La elección entre estas metaheurísticas permite adaptar el enfoque de optimización según las características y requisitos particulares del POMR.

3.5. Problemas de referencias integrados

Para evaluar los algoritmos integrados en el framework, se propone la incorporación de problemas estándar que sirvan como referencia. Estos problemas de referencia han sido diseñados para representar las características diversas en problemas del mundo real. En esta tesis se integran 10 problemas seleccionados del trabajo de Kumar et al., 2021a, ya que los autores destacan que estos representan desafíos complejos derivados de aplicaciones del mundo real. Además, estos problemas fueron utilizados en la competencia CEC 2021, lo que garantiza su relevancia y aceptación en la comunidad científica.

El *test-suite* de Kumar, A., et al. (2021) incluye 50 problemas provenientes de diversas áreas, como el diseño mecánico y los sistemas de potencia. Estos problemas están definidos por ecuaciones matemáticas simples, pero con distintos niveles de dificultad. La selección de los 10 problemas problemas integrados en el framework se restringió a problemas con las siguientes características:

- Problemas de dos objetivos para facilitar la visualización y el análisis de los resultados.
- Optimización continua, debido a que este tipo de problemas es común en aplicaciones de ingeniería, donde las variables de diseño suelen tener valores continuos.

A continuación, se presenta la Tabla 3.2, que detalla las características de los problemas seleccionados, donde k representa el número de objetivos, n el número de variables de diseño, m el número de restricciones de desigualdad y p el número de restricciones de igualdad.

Problema	Nombre	k	n	m	p
PMO1	Cantilever Beam Design	2	2	2	0
PMO2	Design of a Disc Brake	2	4	4	0
PMO3	Front Rail Design	2	3	3	0
PMO4	Vibrating Platform	2	5	5	0
PMO5	Two Bar Truss Design	2	3	3	0
PMO6	Welded Beam Design	2	4	4	0
PMO7	Two Bar Plane Truss	2	2	2	0
PMO8	Simply Supported I-Beam Design	2	4	1	0
PMO9	Multiple-disk Clutch Brake Design	2	5	8	0
PMO10	Hydro-static Thrust Bearing Design	2	4	7	0

Tabla 3.2. Características de los problemas de integrados en el framework.

3.6. Arquitectura del framework

El framework diseñado para la optimización de problemas encapsula de manera integral los procesos de cada algoritmo integrado, así como otros componentes importantes para el proceso de optimización. La Figura 3.6 presenta una visión encapsulada de los elementos esenciales en la arquitectura del framework.

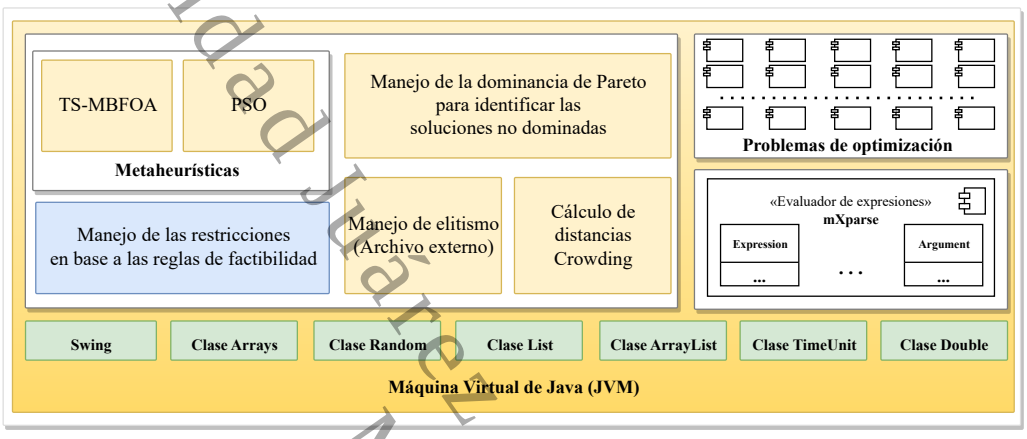


Figura 3.6. Arquitectura del framework para la solución de problemas de optimización.

La arquitectura presentada en la Figura 3.6 ilustra la encapsulación de los procesos clave dentro del framework. Este diseño de alto nivel, implementado en Java, requiere la ejecución en una máquina virtual. Cabe mencionar, que la arquitectura incorpora un componente externo, mxparse, un evaluador de expresiones matemáticas. Este parser es usado cuando el usuario final define un problema personalizado, permitiendo la evaluación de funciones y restricciones a partir de cadenas de texto.

Capítulo 4

Aplicación y pruebas del framework

En este capítulo se describe la aplicación del framework desarrollada. Se detallan las líneas de código necesarias para su uso, así como sus clases más importantes. Además, se presentan las interfaces gráficas diseñadas para la aplicación y se describen su funcionalidad. Por último, se presentan pruebas realizadas para verificar el correcto funcionamiento del sistema integral.

4.1. Fragmentos de código de muestra

El framework desarrollado se puede usar de dos maneras: directamente en el código fuente de alguna aplicación creada en Java o desde una interfaz gráfica. Para los usuarios que prefieran la primera opción, se detallan a continuación los fragmentos de código necesarios para el uso del framework.

El framework incluye una serie de problemas de optimización predefinidos detallados en la Tabla 3.2. En esta sección, para demostrar el código, se presentan dos problemas de optimización. El primero es el problema conocido como Cantilever, un problema de optimización multiobjetivo (Levi et al., 2005), que se utiliza como ejemplo de los problemas predefinidos en el framework. El segundo problema es el Diseño de un freno de disco (Yang y Deb, 2013), que es un problema multiobjetivo con dos objetivos, cinco restricciones y cuatro variables de decisión. Este se utiliza como ejemplo de los problemas que pueden ser ingresados por el usuario.

4.1.1. Selección y resolución del problema precargado Cantilever

El framework proporciona una manera sencilla para seleccionar y resolver problemas de optimización predefinidos. Para ello, se utilizan tres clases claves: `CNOP`, `TSMBFOA` o `PSO` y la clase del problema específico, en este caso, `Cantilever`. A continuación, se detalla cada una de estas clases.

- `CNOP`: La clase `CNOP` es una representación genérica de un problema de optimización. Cada problema específico, como Cantilever, implementa esta interfaz para definir su propio comportamiento con características específicas del problema.
- `TSMBFOA` o `PSO`: La clase `TSMBFOA` o `PSO` representa la metaheurística seleccionada para la búsqueda de soluciones. Esta clase toma una instancia de un problema CNOP y busca soluciones no dominadas utilizando el algoritmo seleccionado. Esta clase toma una instancia de un problema `CNOP`.
- `Cantilever`: Es una instancia de un problema específico predefinido en el framework.

El proceso de seleccionar y resolver un problema es simple y se puede lograr con solo unas pocas líneas de código. A continuación, se muestra un ejemplo de cómo se puede seleccionar y resolver el problema `Cantilever` dentro de un metodo `main` en Java:

```
CNOP cnop = new Cantilever();
TSMBFOA tsmbfoa = new TSMBFOA(cnop, false);
tsmbfoa.setExecutions(30);
tsmbfoa.run();
```

En este código, la primera línea crea una nueva instancia del problema `Cantilever`. La segunda línea crea una nueva instancia de la metaheurística `TSMBFOA`, pasando el problema de optimización. La tercera línea establece el número de ejecuciones independientes de la metaheurística a 30. Finalmente, la cuarta línea ejecuta la metaheurística para resolver el problema de optimización.

Para acceder a las soluciones no dominadas encontradas por la metaheurística, se utiliza el siguiente método:

```
tsmbfoa.getResultFinal();
```

Los resultados devueltos por este método son una lista de soluciones no dominadas encontradas por la metaheurística. Cada solución incluye los valores de las variables de decisión $\vec{x}$ y los valores de las funciones objetivo $f_1, f_2, ..., f_n$.

4.1.2. Resolución de un problema ingresado por el usuario

El framework también permite a los usuarios personalizar sus propios problemas de optimización. Para ello, se deben implementar las clases `CNOP` y `TSMBFOA` o `PSO`. A continuación, se muestra un ejemplo de cómo se puede definir un problema de optimización y resolverlo dentro de un método `main` en Java.

- En lugar de instanciar una `CNOP` integrada, hay que crear lo siguiente:

```
CNOP cnop = new CNOP();
```

- A continuación, se establece la información del `CNOP`, como el nombre, las funciones objetivos, el orden de las variables y los rangos de las variables.

```
cnop.setNameProblem("Design of a Disc Brake");
cnop.addObjective("4.9 * 10^(-5) * (x2^2 - x1^2) * (x4-1)", CNOP.MINIMIZATION);
cnop.addObjective("(9.82 * 10^6 * (x2^2 - x1^2)) / (x3 * x4 * (x2^3 - x1^3))"
          , CNOP.MINIMIZATION);
cnop.setOrderVariables("x1;x2;x3;x4");
cnop.setVariableRange("(55.0 , 80.0);(75.0 , 110.0);(1000.0 , 3000.0)"
          + ";(2.0 , 20.0)");
```

- Luego se hace la importación de una clase llamada `Constraints` que permite definir las restricciones del problema. A continuación, se muestra un ejemplo de cómo se pueden definir las restricciones.

```
Constraints constraints = new Constraints();


constraints.add("(x2 - x1) - 20 >= 0");
constraints.add("30 - 2.5 * (x4 + 1) >= 0");
constraints.add("0.4 - ( (x3) / (3.14 * (x2^2 - x1^2) ) ) >= 0");
constraints.add("1 - ( (2.22 * 10^(-3) * x3 * (x2^3 - x1^3)) / "
          + "( (x2^2 - x1^2)^2 ) ) >= 0");
constraints.add("( (2.66 * 10^(/2) * x3 * x4 * (x2^3 - x1^3)) / "
          + "(x2^2 - x1^2) ) - 900 >= 0");
```

```
cnop.setConstraints(constraints); // Se agregan las restricciones al problema
```

- Finalmente, se crea una instancia de la metaheurística y se ejecuta para resolver el problema de optimización.

```
//...
TSMBFOA tsmbfoa = new TSMBFOA(cnop, false);
tsmbfoa.setExecutions(30);
tsmbfoa.run();
```

Una de las características de las metaheurísticas es que sus parámetros pueden ser ajustados para mejorar su rendimiento. En el caso de la metaheurística `TSMBFOA`, se pueden ajustar los siguientes parámetros con unas simples líneas de código:

```
tsmbfoa.setSb(100);
tsmbfoa.setNc(50);
tsmbfoa.setStepSize(0.0005);
tsmbfoa.setScalingFactor(1.95);
tsmbfoa.setBacteriaReproduce(1);
tsmbfoa.setRepcycle(100);
tsmbfoa.setEvaluations(30000);
```

Los resultados generados por la metaheurística se obtienen utilizando el método `getResultFinal()`, de la misma manera que en el caso anterior. Estos resultados, visualizados desde la consola, consisten en una lista de soluciones no dominadas.

Para verificar que los valores obtenidos para cada variable de decisión cumplen con las restricciones y sus límites mínimos y máximos, se ha implementado una herramienta llamada *CNOP-solution Tester*. Esta herramienta, accesible en https://garcialopez.github.io/CNOPsolution-tester/ facilita la validación de los resultados generados por el framework.

En esta prueba, se utilizaron una solución no dominada resultantes del framework para cada variable de decisión. Los valores de las variables de decisión x y las funciones objetivo f son:

$$x = \begin{bmatrix} 79.072087445597 \\ 99.123123468757 \\ 2977.0651874116 \\ 9.3720126634401 \end{bmatrix}, \quad f = \begin{bmatrix} 1.4657462839668454 \\ 2.6224300796937237 \end{bmatrix}$$

Los valores de las variables x se ingresan en la herramienta *CNOPsolution Tester*. Los valores de las funciones objetivo f servirán como referencia para verificar que la evaluación sea correcta. Dado que *CNOPsolution Tester* fue inicialmente desarrollado para problemas mono-objetivo, para esta prueba, la segunda función objetivo se ingresa como una restricción. La Figura 4.1 muestra la configuración utilizada para la validación en *CNOPsolution Tester*.

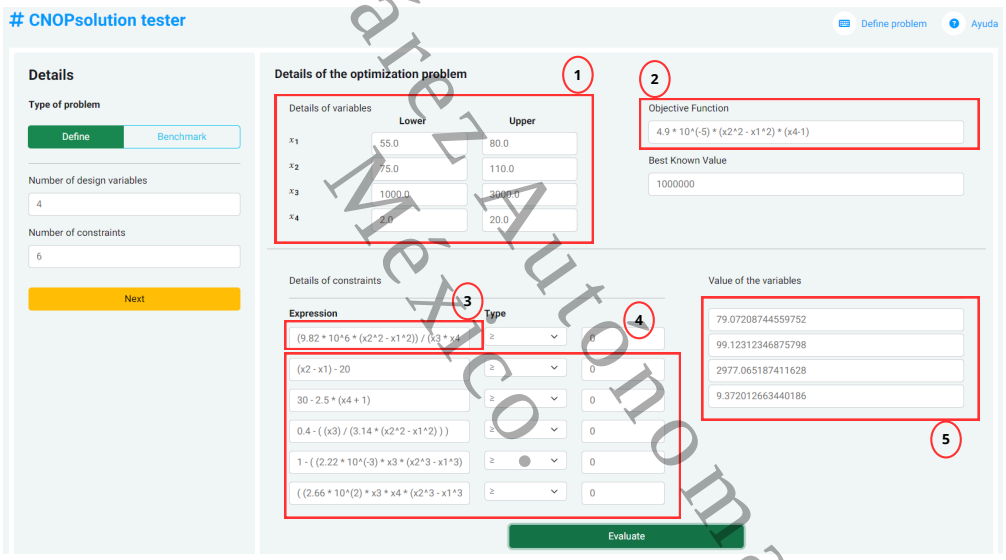


Figura 4.1. Configuración del problema en CNOPsolution Tester.

La Figura 4.1 muestra el proceso de ingreso de datos en la herramienta *CNOPsolution Tester*. En el panel 1, se ingresan los límites de las variables de decisión. La primera función objetivo se introduce en el panel 2, mientras que la segunda función objetivo, se ingresa como restricción en el panel 3. Las restricciones del problema se ingresan en el panel 4, y los valores de las variables de decisión generados por el framework se ingresan en el panel 5. El resultado de la validación se muestra en la Figura 4.2.

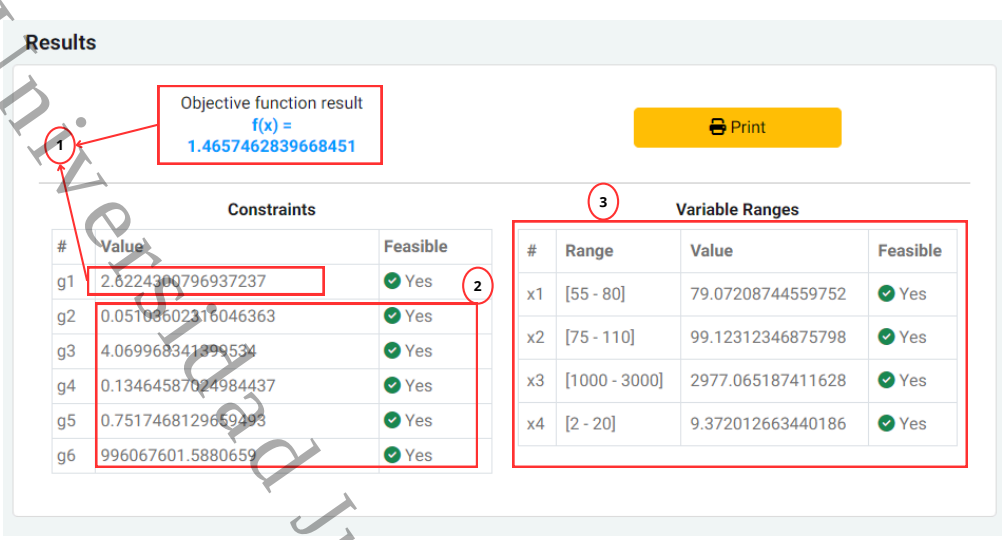


Figura 4.2. Resultado de la validación en CNOPsolution Tester.

La Figura 4.2 confirma que los valores generados por el framework cumplen con las restricciones y los límites de las variables de decisión, como se muestra en los paneles 2 y 3. Además, las funciones objetivo generadas por el framework coinciden con las generadas en la herramienta *CNOPsolution Tester*, como se puede observar en el panel 1.

Las soluciones generadas por el framework se visualizan en texto plano en la consola. Para facilitar la visualización de los resultados de los usuarios que opten por generar resultados desde esta opción, se ha desarrollado *ParetoPlot-MO*, una herramienta que permite visualizar el frente de Pareto generado por el framework. Esta herramienta, accesible en https://garcialopez.github.io/ParetoPlot-MO/pareto.html, recibe dos conjuntos de datos como entrada: el primero para las funciones objetivo 1 y el segundo para las funciones objetivo 2. La Figura 4.3 muestra la configuración utilizada (panel 1) y la visualización del frente de Pareto (panel 2) generado por el framework.

Figura 4.3. Visualización del frente de Pareto en ParetoPlot MO.

4.2. Interfaz de Usuario

Existen usuarios con poco o nulo conocimiento de programación, que realizar las tareas descritas en la sección anterior puede resultar complicado. Por ello, en esta sección se presenta un sistema integral desarrollado con el objetivo de facilitar el uso del framework. Este sistema permite que el usuario final pueda utilizar el framework con solo unos clics desde una interfaz gráfica, sin la necesidad de conocer el código fuente escrito en el lenguajes de programación.

Para utilizar el sistema integral, se deben seguir tres pasos que corresponden a las configuraciones de la ejecución a realizar. La interfaz gráfica inicial se puede visualizar en la Figura 4.4.

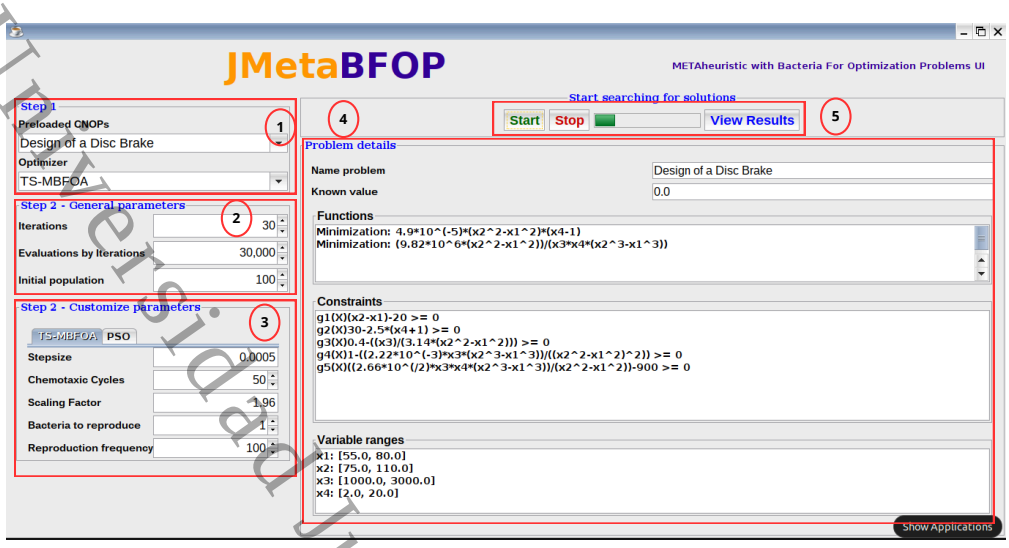


Figura 4.4. Interfaz gráfica inicial para la configuración de una nueva ejecución.

La Interfaz gráfica presentada en la Figura 4.4, esta estruturada en 5 partes, las cuales se describen a continuación:

- **Parte 1:** En esta sección, el usuario puede seleccionar un problema de optimización predefinido en el framework o ingresar un problema personalizado. Para seleccionar un problema predefinido, el usuario debe elegir el problema de la lista desplegable. También el usuario puede seleccionar la metaheurística a utilizar, en este caso TS-MBFOA o PSO.

- **Parte 2:** El usuario puede configurar los parámetros generales para la ejecución del algoritmo seleccionado. Estos incluyen el número de ejecuciones independientes, que es un valor entero entre 1 y 30. Otro parámetro es el tamaño de la población, que varía entre 10 y 500, donde cada individuo representa una solución inicial. Al tratarse de algoritmos poblacionales, estas soluciones iniciales sirven como base para buscar y mejorar iterativamente los resultados. Finalmente, se establece el número de evaluaciones, que determina la cantidad máxima de veces que las funciones objetivo serán evaluadas.

- **Parte 3:** En esta sección, el usuario puede configurar los parámetros específicos de la metaheurística seleccionada, los cuales varían según el algoritmo elegido, ya sea TS-MBFOA o PSO. Ajustar estos parámetros ayuda a optimizar el rendimiento del algoritmo, adaptándolo al problema a resolver y haciendo más eficiente la búsqueda de soluciones no dominadas.

- **Parte 4:** En esta sección, el usuario puede consultar información del problema, incluyendo las funciones objetivo, las restricciones y los límites de las variables.
- **Parte 5:** Esta sección contiene tres botones y una barra de progreso. El primer botón permite al usuario iniciar la ejecución del algoritmo. El segundo botón permite al usuario detener la ejecución en cualquier momento. La barra de progreso muestra el avance del algoritmo en tiempo real. Finalmente, el tercer botón permite al usuario ir a una nueva pantalla (Fig. 4.5) donde puede visualizar los resultados en tablas y gráficos.

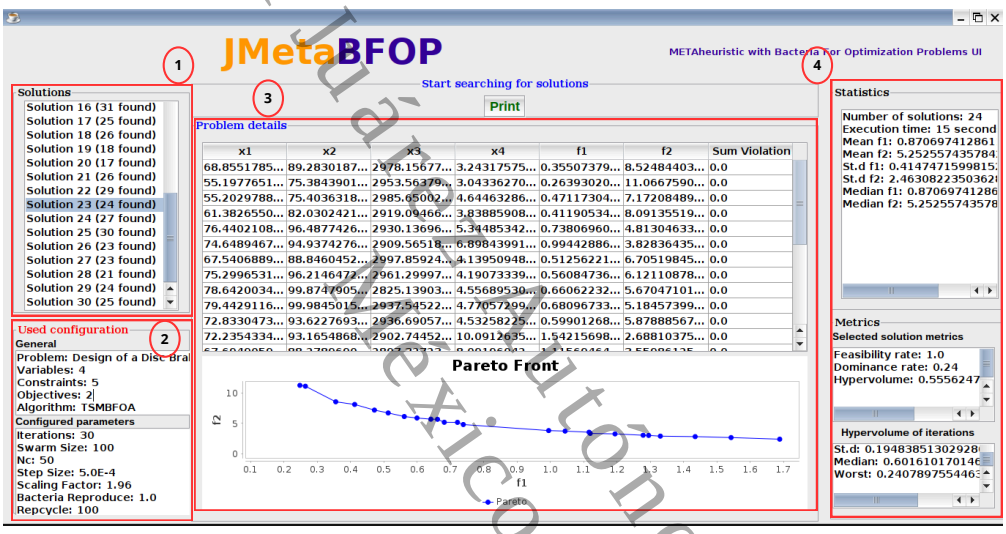


Figura 4.5. Interfaz gráfica de visualización de resultados.

La interfaz gráfica de resultados tiene 4 secciones para la visualización de elementos que facilitan al usuario obtener información en un tiempo menor en comparación con obtenerla manualmente por consola. Estas secciones se describen a continuación:

- **Sección 1:** Esta sección muestra una lista dinámica de soluciones, desde la solución 1 hasta la solución correspondiente al número de ejecuciones. Al seleccionar una solución de la lista, los elementos de la tabla y el gráfico del frente de Pareto cambian en función de la solución seleccionada.
- **Sección 2:** Esta sección proporciona detalles generales de la configuración realizada en la interfaz gráfica presentada en la Figura 4.4.

- **Sección 3:** Esta sección muestra la tabla de resultados y el gráfico del frente de Pareto de las soluciones no dominadas. Estos valores son dinámicos y cambian en función del número de solución seleccionado en la Sección 1.
- **Sección 4:** Esta sección presenta estadísticas generales para cada función objetivo, como la media, la mediana, la desviación estándar, el número de soluciones y el tiempo de ejecución. Además, se añaden métricas como la tasa de factibilidad, la tasa de dominancia y el hipervolumen. También se incluyen estadísticas básicas como el mejor, el peor, la media, la mediana y la desviación estándar del cálculo de todos los hipervolumen del total de soluciones.

Este sistema integral proporciona una interfaz gráfica de usuario que facilita a los usuarios sin conocimientos de programación la ejecución de las metaheurística incluidas. Los usuarios pueden seleccionar y configurar problemas, ajustar parámetros de las metaheurísticas, y visualizar los resultados. Además, la capacidad de ver detalles de la configuración, las estadísticas generales y las métricas de rendimiento para visualizar el rendimiento del algoritmo.

4.3. Prueba experimentales de la interfaz de usuario

Para evaluar la funcionalidad del sistema integral presentado en la sección anterior, se diseñaron tres Pruebas Experimentales (PE). Estas pruebas se centran en tres aspectos: la configuración de la ejecución, la visualización de los resultados y el dinamismo de los resultados. Cada prueba se describe en detalle en la Tabla 4.1, 4.2 y 4.3, incluyendo los pasos seguidos y los resultados de cada prueba.

Título de la prueba: Configuración de la ejecución	**CPF**	**CPF-01**
	¿Prueba de despliegue?	**No**
Descripción: Verificar que el sistema permita al usuario seleccionar y configurar problemas correctamente, así como ajustar los parámetros de las metaheurísticas de forma eficiente y sin errores.		
Pasos: 1. Seleccionar un problema. 2. Configurar el problema seleccionado. 3. Ajustar los parámetros de las metaheurísticas. 4. Probar diferentes combinaciones de problemas y parámetros.		
Resultados: El sistema permitió la selección y configuración de problemas, así como el ajuste de los parámetros de las metaheurísticas. Para verificar su funcionalidad, se realizaron 30 ejecuciones independientes, variando parámetros clave como el tamaño de la población (entre 10 y 500 individuos) y el número de evaluaciones. En todos los casos, el sistema permitió la configuración correcta de los problemas seleccionados. Además, como parte de las pruebas de validación, se intentó ingresar valores no válidos, como 0 en el tamaño de la población, y el sistema respondió correctamente al no permitir la configuración con estos valores erróneos.		

Tabla 4.1. Caso prueba funcional 01. Configuración de la ejecución.

Título de la prueba: Visualización de resultados	**CPF**	**CPF-02**
	¿Prueba de despliegue?	**No**
Descripción: Evaluar la capacidad del sistema para generar y mostrar correctamente los resultados de la ejecución de la metaheurística seleccionada. Se debe verificar si al ejecutar el algoritmo con los parámetros configurados, el sistema es capaz de generar una representación visual coherente de los resultados. Esto incluye la correcta creación de tablas y gráficos que presenten información relevante, como los valores de las funciones objetivos, las iteraciones y los parámetros utilizados durante la ejecución.		
Pasos: 1. Realizar una nueva ejecución. 2. Abrir la interfaz gráfica de resultados. 3. Verificar que la tabla y gráfico se generen correctamente. 4. Verificar que las tablas y gráficos muestren la información coherente.		
Resultados: El sistema genera y muestra correctamente los resultados de la ejecución de la metaheurística seleccionada. Al ejecutar el algoritmo con los parámetros configurados, el sistema produce una representación visual coherente de los resultados, incluyendo la correcta creación de tablas y gráficos que presentan información de los valores de las funciones objetivo, las iteraciones y los parámetros utilizados.		

Tabla 4.2. Caso prueba funcional 02. Visualización de resultados.

Título de la prueba: Dinamismo de los resultados	**CPF**	**CPF-03**
	¿Prueba de despliegue?	**No**
Descripción: Verificar que, al hacer la selección de un conjunto de soluciones de la lista, el sistema actualice en tiempo real la representación visual (gráfico de Pareto y tablas) y los valores de las funciones objetivo, las iteraciones y los parámetros utilizados. Además, asegurar que los datos presentados en los gráficos y tablas coincidan con los valores calculados internamente por el sistema, y que la visualización sea clara y fácil de interpretar para el usuario.		
Pasos: 1. Seleccionar una solución de la lista. 2. Verificar que los gráficos de Pareto y los resultados de la tabla se actualizan dinámicamente. 3. Seleccionar diferentes soluciones.		
Resultados: El sistema actualiza dinámicamente los gráficos de Pareto y los resultados de la tabla al seleccionar un conjunto de soluciones de la lista. El tamaño de la lista depende del número de ejecuciones que se haya configurado. La actualización se realiza en tiempo real, mostrando los valores de las funciones objetivo, las iteraciones y los parámetros correspondientes a la solución seleccionada. Además, los datos presentados en los gráficos y tablas coinciden con los valores calculados internamente por el sistema y son fácil de leer y visualizar.		

Tabla 4.3. Caso prueba funcional 03. Dinamismo de los resultados.

Las pruebas experimentales realizadas en el sistema demuestran que las funcionalidades claves se ejecutan correctamente. En la prueba visualizada en la Tabla 4.1, el sistema permitió la correcta selección y configuración de los problemas, así como el ajuste adecuado de los parámetros de las metaheurísticas. En cuanto a la visualización de resultados (ver Tabla 4.2), el sistema mostró de manera coherente y precisa los resultados de la ejecución de la metaheurística seleccionada, generando tablas y gráficos que reflejan correctamente la generación de los valores de las funciones objetivos y los parámetros utilizados. Por último, en la prueba presentada en la Tabla 4.3, el sistema actualiza dinámicamente los gráficos y las tablas al seleccionar diferentes soluciones, manteniendo una visualización clara y fácil de interpretar.

Capítulo 5

Experimentos y Resultados

Este capítulo se describe la fase experimental de la tesis, detallando la configuración de los algoritmos y los parámetros utilizados. Se comparan los resultados obtenidos por TS-MBFOA y PSO al resolver problemas de optimización multiobjetivo, evaluando su desempeño mediante métricas como hipervolumen, tasa de factibilidad y Two Set Coverage.

5.1. Parámetros y ejecución del algoritmo

La fase experimental del framework requiere ajustes iniciales para configurar los algoritmos utilizados en la búsqueda de soluciones no dominadas de los problemas de optimización. Tomando en cuenta los valores de referencia utilizados en el trabajo de Kumar et al., 2021a, los cuales sirven como punto de comparación para medir la competitividad de los algoritmos implementados en esta tesis, se establece que el número de ejecuciones independientes (Max_e) es de 25. Estos experimentos se realizaron en una PC con sistema operativo Linux (Ubuntu 22.04), procesador Intel Core i7 (8ª generación) y 16 GB de RAM.

Cada algoritmo tiene parámetros específicos ajustados al problema, por lo que TS-MBFOA y PSO presentan configuraciones distintas. Sin embargo, para garantizar una comparación justa, existen parámetros comunes, como el tamaño de la población y el número de evaluaciones, que se utilizan como condición de paro.

Según Kumar et al., 2021a, el número de evaluaciones (Max_{FEs}) se determina en función del número de variables de diseño (n) y número de objetivos (k) del problema de optimización. Estos

valores se representa de la siguiente manera:

$$Max_{FEs} = \begin{cases} 2 \times 10^4 & \text{si } k = 2 \text{ y } n \leq 10 \\ 8 \times 10^4 & \text{si } k = 2 \text{ y } n > 10 \end{cases}$$

De acuerdo con los problemas integrados descritos en la Tabla 3.2, solo se incluyen problemas con dos objetivos y con menos de 10 variables de diseño. Esto significa que la configuración para el número de evaluaciones se establece en 2×10^4. La configuración final se presenta en la Tabla 5.1.

Parámetro	**TS-MBFOA**	**PSO**
Población	S_b : 100	NP : 100
N_c	50	-
S_r	1	-
β	1.95	-
$RepCycle$	100	-
α_1	-	1.496180
α_2	-	1.496180
w	-	0.729844
Max_{FEs}	2×10^4	2×10^4
Max_e	25	25

Tabla 5.1. Configuración de parámetros de TS-MBFOA y PSO.

Se utilizaron los puntos nadir de las funciones objetivo, obtenidos según Kumar et al., 2021a, disponibles en el repositorio de GitHub: https://github.com/P-N-Suganthan/2021-RW-MOP. Estos puntos nadir representan los valores de referencia de las funciones objetivo $f_1(x)$ y $f_2(x)$ para los 10 problemas seleccionados. La Tabla 5.2 presenta estos valores, los cuales se utilizaron para evaluar la competitividad de los algoritmos implementados en esta tesis.

Problema	$f_1(x)$	$f_2(x)$
PMO1	3.06E+00	2.04E-03
PMO2	5.31E+00	3.03E+00
PMO3	9.34E-01	1.20E+00
PMO4	-1.27E-03	3.18E+02
PMO5	1.00E-01	1.00E+05
PMO6	3.67E+01	1.31E-02
PMO7	1.87E+02	6.77E-05
PMO8	4.38E+02	6.15E-02
PMO9	1.40E+00	1.49E-02
PMO10	2.67E+02	-2.77E-05

Tabla 5.2. Puntos nadir de las funciones objetivos de problemas integrados.

Los puntos nadir representan los valores más altos alcanzables por cada función objetivo $f_i(x)$ en el conjunto de soluciones Pareto óptimas $\mathcal{P}^*$ (Bechikh et al., 2010). El punto nadir para la función $f_i(x)$ es el valor máximo f_i^{nadir} tal que:

$$f_i^{\text{nadir}} = \max_{x \in \mathcal{P}} f_i(x)$$

Estos puntos sirven como referencia para comparar los resultados obtenidos por los algoritmos TS-MBFOA y PSO con los valores teóricos, permitiendo evaluar la competitividad de las soluciones generadas.

5.2. Resultados, métricas y análisis gráfico

Los resultados presentados a continuación permiten evaluar la competitividad de los algoritmos TS-MBFOA y PSO, en la resolución de los problemas multi-objetivo abordados en esta tesis. Los experimentos se realizaron utilizando las configuraciones descritas en la Tabla 5.1. Cada algoritmo fue ejecutado de manera independiente en 25 iteraciones para cada problema de optimización. Posteriormente, se seleccionó el mejor conjunto de soluciones obtenido en cada caso, registrando también el punto nadir correspondiente a los nuevos valores generados. En la Tabla 5.3 se presenta una comparación de los resultados generados por ambos algoritmos en cada uno de los problemas integrados.

Problema	TS-MBFOA			PSO		
	$f_1(x)$	$f_2(x)$	$\|\mathcal{P}^*\|$	$f_1(x)$	$f_2(x)$	$\|\mathcal{P}^*\|$
PMO1	3.0561	4.70E-05	36	3.0469	4.40E-05	88
PMO2	1.3542	3.0485	26	1.1501	3.5480	62
PMO3	0.9024	1.0259	33	0.9021	1.0277	81
PMO4	-0.0027	318.8033	8	-0.0027	315.6398	13
PMO5	0.0228	14109.1868	22	0.0179	12966.1596	14
PMO6	-	-	0	-	-	0
PMO7	181.2369	5.20E-07	48	64.00	1.47E-06	94
PMO8	446.2522	0.0095	27	484.4949	0.0088	56
PMO9	1.0253	0.0030	19	1.9537	0.0029	24
PMO10	6937.6991	4.33E-06	6	6290.00	3.66E-06	16

Tabla 5.3. Comparación de resultados entre TS-MBFOA y PSO para los problemas multiobjetivo.

Para cada problema, se registraron los valores de las funciones objetivo $f_1(x)$ y $f_2(x)$ correspondientes a las soluciones más representativas, así como el tamaño del conjunto de soluciones $\mathcal{P}^*$ (no dominadas) identificado en la mejor ejecución de las 25 realizadas por cada algoritmo.

Para el problema PMO1, los valores generados por ambos algoritmos son cercanos. TS-MBFOA genera un valor de $f_1(x) = 3.0561$, mientras que PSO un valor ligeramente más bajo de $f_1(x) = 3.0469$, lo que indica que ambos algoritmos presentan un desempeño equilibrado en la primera función objetivo. En cuanto a la $f_2(x)$, TS-MBFOA obtiene un valor de 4.70E-05, mientras que PSO un valor de 4.40E-05. Este comportamiento es característica de las metaheurísticas, que al estar basadas en la búsqueda de soluciones aproximadas y la exploración de grandes espacios de búsqueda, suelen mostrar resultados competitivos y cercanos entre sí, especialmente cuando se emplean en problemas restringidos como este. Ambos algoritmos se acercan al punto nadir en ambas funciones, lo que refleja la capacidad de estos métodos para explorar eficazmente el espacio de soluciones. Aunque TS-MBFOA tiene un valor de $f_1(x)$ ligeramente más alto que el punto nadir, la diferencia es mínima, lo cual puede atribuirse a la naturaleza de las metaheurísticas, que son métodos eficientes, aunque no garantizan la optimalidad. PSO, por su parte, también muestra un rendimiento similar, lo que refuerza la idea de que ambos algoritmos logran una optimización notable en $f_2(x)$, alcanzando valores muy cercanos al mínimo posible.

En el problema PMO2, PSO muestra un mejor desempeño que TS-MBFOA en ambas funciones objetivo. Para $f_1(x)$, PSO obtiene un valor de 1.1501 frente a 1.3542 de TS-MBFOA, y en $f_2(x)$, logra 3.5480 contra 3.0485. Aunque este último valor esté más cerca del punto nadir

($f_2(x) = 3.03$), el rendimiento global de PSO en ambas funciones sugiere una mayor eficiencia en la optimización del problema.

PMO2 es un problema más complejo que otros como PMO1 o PMO7, ya que cuenta con más variables de diseño y restricciones de desigualdad. Esto amplía el espacio de búsqueda y lo vuelve más desafiante, al reducir las regiones factibles y aumentar la dificultad para encontrar soluciones viables. PSO aprovecha su capacidad de exploración global mediante el intercambio constante de información entre partículas, lo que le permite cubrir mejor el espacio y concentrarse después en zonas prometedoras. TS-MBFOA también tiene mecanismos de búsqueda global, pero podría mostrar menor efectividad para intensificar en áreas óptimas bajo estas condiciones, afectando su rendimiento. Aun así, ambos algoritmos logran mejoras importantes con respecto al punto nadir, lo que refleja su capacidad como metaheurísticas para encontrar soluciones cercanas al óptimo, incluso en problemas con restricciones complejas.

En el problema PMO3, tanto TS-MBFOA como PSO obtienen resultados casi equivalentes. Para la $f_1(x)$, los valores son 0.9024 y 0.9021, mientras que en la $f_2(x)$ se obtiene 1.0259 para TS-MBFOA y 1.0277 para PSO. Estas pequeñas diferencias indican que ambos algoritmos fueron igualmente efectivos al enfrentar este problema. PMO3 cuenta con tres variables de diseño y tres restricciones de desigualdad, lo que representa una complejidad moderada dentro del conjunto de problemas evaluados. En este tipo de escenarios, donde el espacio de búsqueda no es excesivamente amplio pero sigue siendo lo suficientemente complejo, las metaheurísticas tienden a comportarse de forma similar, alcanzando soluciones comparables gracias a sus mecanismos de exploración y explotación. El hecho de que ambos algoritmos hayan generado soluciones tan cercanas al punto nadir ($f_1(x) = 0.934$, $f_2(x) = 1.20$) confirma su capacidad para adaptarse a este tipo de problemas con restricciones balanceadas y representan ser alternativas viables para obtener soluciones de calidad, sin mostrar diferencias significativas entre ellas.

En el problema PMO4, ambos algoritmos muestran un desempeño similar en la $f_1(x)$, con un valor de -0.0027. En la segunda función objetivo existe ligera diferencia: TS-MBFOA alcanza un valor de 318.8033, mientras que PSO obtiene 315.6398; aunque esta variación no es amplia, indica que TS-MBFOA logra una aproximación más cercana al punto nadir en $f_2(x)$. PMO4 se caracteriza por tener seis variables de diseño y cuatro restricciones de desigualdad, lo que lo convierte en un problema de mayor complejidad en comparación con casos anteriores como PMO3.

Estas características aumenta la dificultad del espacio de búsqueda, ya que implica regiones factibles más restringidas y una mayor interacción entre variables y restricciones. El que ambos algoritmos hayan llegado a valores tan cercanos al punto nadir demuestra que pueden adaptarse bien a problemas más complejos. Aun así, la pequeña ventaja que muestra TS-MBFOA en $f_2(x)$ podría deberse a que tiene una mejor capacidad para buscar soluciones en zonas del espacio de búsqueda que son más prometedoras.

En el problema PMO5, PSO muestra una ventaja clara sobre TS-MBFOA. El valor de $f_1(x)$ para PSO es 0.0179, mientras que TS-MBFOA obtiene un valor más alto de 0.0228. La diferencia en $f_2(x)$ es aún más significativa, con PSO alcanzando 12966.1596 y TS-MBFOA 14109.1868. Esto indica que PSO es más eficiente en la búsqueda de soluciones cercanas a las óptimas en este caso. PMO5 tiene un espacio de búsqueda más amplio y complejo, lo que puede haber influido en la diferencia de desempeño entre los algoritmos. PSO logra resultados más cercanos al punto nadir en ambas funciones: el valor de $f_1(x)$ es considerablemente menor que el punto nadir, y lo mismo ocurre con $f_2(x)$. Aunque TS-MBFOA también muestra buenos resultados, no alcanza el nivel de eficiencia de PSO en este problema específico.

En el caso de PMO6, ambos algoritmos no lograron encontrar soluciones válidas. Este problema tiene cuatro restricciones no lineales, todas las cuales limitan el espacio de búsqueda. Además, las variables tienen límites estrictos, lo que hace que el espacio factible sea aún más cerrado.

En el problema PMO7, ambos algoritmos mejoran el punto nadir en al menos una de las funciones. PSO logra un $f_1(x) = 64.00$, muy por debajo del nadir de 1.87E+02, lo que refleja su eficacia en la primera función. En cambio, TS-MBFOA alcanza un $f_2(x) = 5.20E-07$, frente al nadir de $6.77E-05$, demostrando su superioridad en la segunda función. Por su parte, TS-MBFOA obtiene un $f_1(x) = 181.2369$ y PSO un $f_2(x) = 1.47E-06$, ambos valores también por debajo del nadir, pero menos extremos. En conjunto, PSO explota mejor el espacio para optimizar f_1, mientras que TS-MBFOA intensifica con más precisión en las regiones óptimas de f_2.

En el problema PMO8, TS-MBFOA consigue un mejor valor en la primera función, $f_1(x) = 446.2522$, frente a 484.4949 de PSO, mientras que PSO alcanza un $f_2(x) = 0.0088$, ligeramente por debajo del 0.0095 de TS-MBFOA. Comparados con el punto nadir, ambos mejoran notablemente f_2, pero solo TS-MBFOA se acerca más al óptimo de f_1. Esto refleja cómo PSO, con su fuerte

exploración global, explota eficazmente regiones de bajo f_2, mientras que TS-MBFOA, gracias a su capacidad de intensificación, consigue soluciones superiores en f_1.

En el problema PMO9, con cinco variables de diseño y ocho restricciones de desigualdad, ambos algoritmos mejoran notablemente el punto nadir. TS-MBFOA alcanza un $f_1(x) = 1.0253$, muy por debajo del nadir de referencia, frente a los 1.9537 de PSO, lo que muestra su capacidad para intensificar con precisión en la primera función. Por su parte, PSO se mejora en la segunda función, con $f_2(x) = 0.0029$, ligeramente mejor que los 0.0030 de TS-MBFOA lo que evidencia su eficacia explorando zonas óptimas para f_2.

En PMO10, que involucra cuatro variables de diseño y siete restricciones de desigualdad, PSO vuelve a generar mejores soluciones que TS-MBFOA en ambos objetivos. Para $f_1(x)$, PSO alcanza 6290.00 frente a 6937.6991 de TS-MBFOA, mientras que en $f_2(x)$ logra 3.66E-06 contra 4.33E-06. Ninguno llega al punto nadir en $f_1(x)$, pero ambos se acercan notablemente en $f_2(x)$.

En la Figura 5.1 se visualiza que, en la mayoría de los problemas, las trayectorias de $f_1(x)$ generadas por TS-MBFOA, PSO y los valores nadir de referencia convergen casi de forma coincidente; únicamente en PMO10 PSO alcanza una disminución más pronunciada de $f_1(x)$.

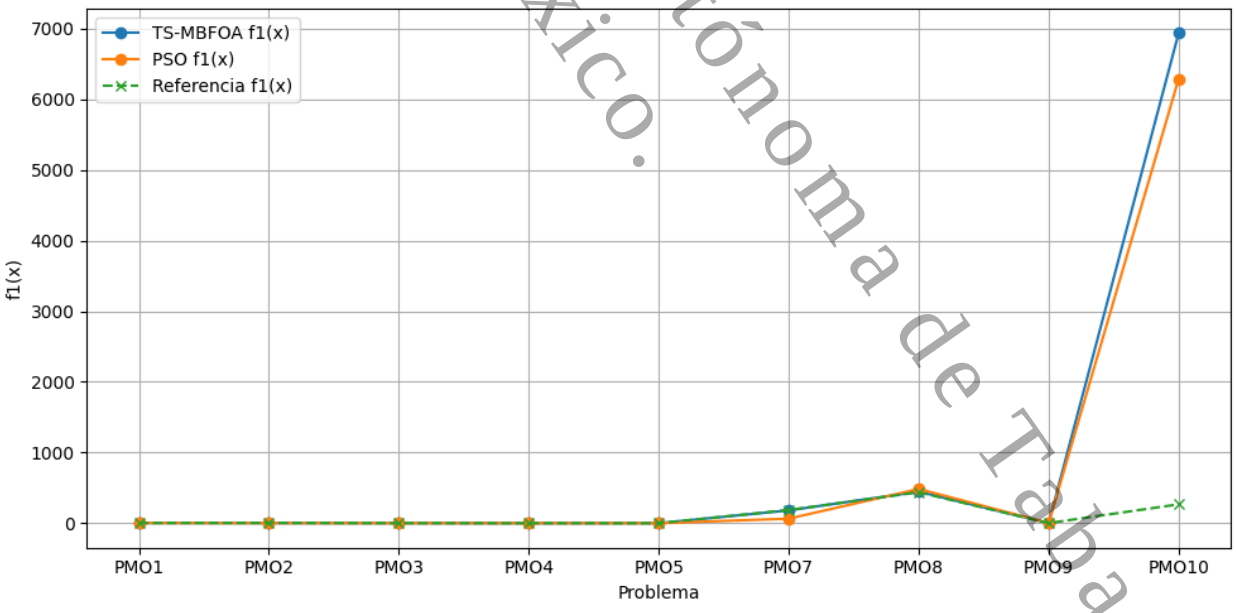


Figura 5.1. Comparación de $f_1(x)$ entre TS-MBFOA, PSO y los valores de referencia.

En la Figura 5.2 se observa que las curvas de $f_2(x)$ de TS-MBFOA, PSO y los valores nadir de referencia muestran una coincidencia en los problemas, lo que confirma que ambos algoritmos

alcanzan niveles comparables de optimización en la segunda función objetivo.

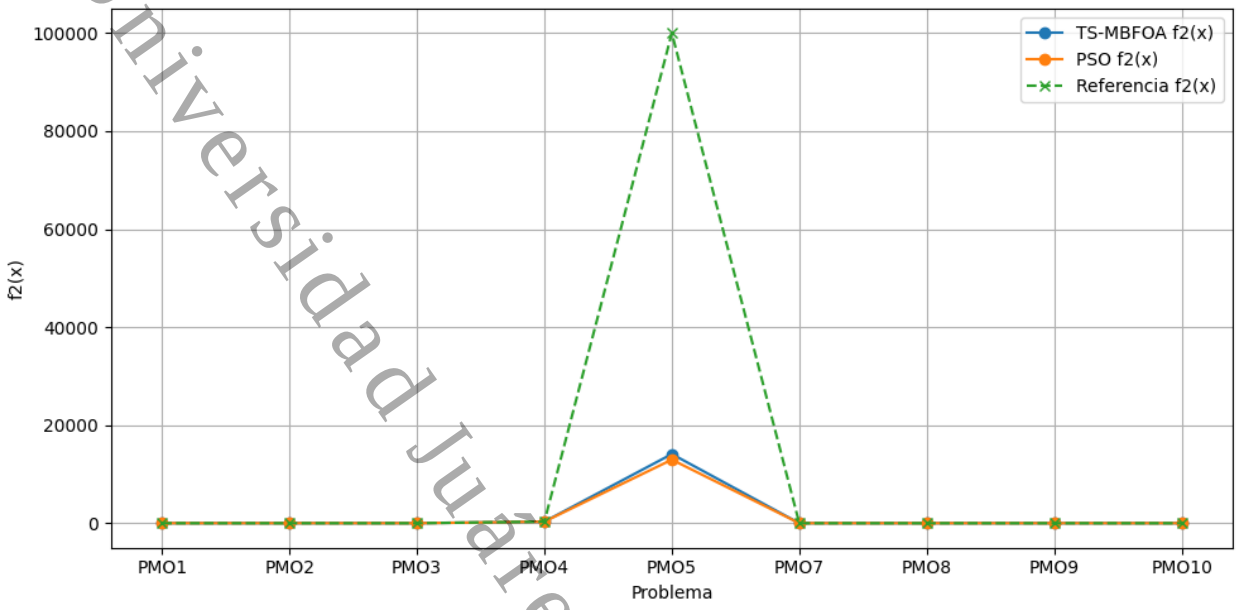


Figura 5.2. Comparación de $f_2(x)$ entre TS-MBFOA, PSO y los valores de referencia.

En las cantidades de soluciones no dominadas ($|\mathcal{P}^*|$) encontradas por TS-MBFOA y PSO (ver Figura 5.3), se observa que PSO tiende a generar un mayor número de soluciones no dominadas en la mayoría de los problemas. Esto se debe a su capacidad de exploración más amplia, lo que le permite abarcar un mayor rango del espacio de soluciones. Por otro lado, TS-MBFOA, con su enfoque más centrado en el doble nado para explorar y explotar, encuentra menos soluciones, pero igual de eficiente en mejorar las soluciones existentes.

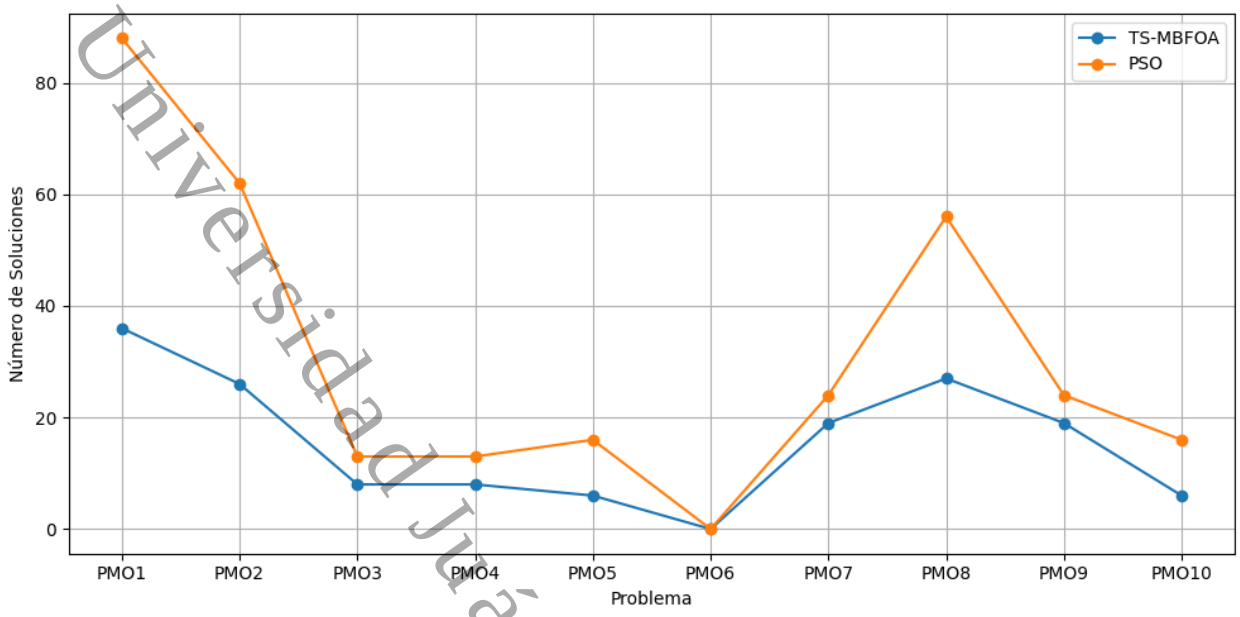


Figura 5.3. Comparación del número de soluciones no dominadas por TS-MBFOA y PSO.

Además del número de soluciones no dominadas, se evaluó el rendimiento de los algoritmos considerando la tasa de factibilidad (T_Fact), el hipervolumen (HV), su desviación estándar (HV_std), la media (HV_med) y el peor caso observado (HV_peor). Estos indicadores permiten comparar la calidad, estabilidad y robustez de las soluciones obtenidas por TS-MBFOA y PSO. La Tabla 5.4 presenta los resultados para los diez problemas PMO1–PMO10.

Problema	**TS-MBFOA**					**PSO**				
	T_Fact	HV	HV_std	HV_med	HV_peor	T_Fact	HV	HV_std	HV_med	HV_peor
PMO1	0.36	0.2084	**0.0948**	**0.3290**	0.1799	**0.88**	**0.7680**	0.2658	0.6234	**0.3359**
PMO2	0.26	**0.9011**	**0.2115**	0.4553	0.1571	**0.62**	0.8895	0.3483	**0.5940**	**0.1907**
PMO3	0.33	0.0303	**0.0061**	0.0167	0.0064	**0.81**	**0.0930**	0.0130	**0.0696**	**0.0443**
PMO4	0.08	0	0	0	0	**0.13**	0	0	0	0
PMO5	**0.22**	0.2000	**0.1020**	**0.2454**	0.0850	0.14	**0.7000**	0.1167	0.2137	**0.0987**
PMO6	-	-	-	-	-	-	-	-	-	-
PMO7	0.48	0.9154	0.3880	0.5934	0.0759	**0.94**	**0.9780**	**0.1984**	**0.8590**	**0.5952**
PMO8	0.27	**0.9710**	**0.2739**	**0.6226**	0.2000	**0.56**	0.9647	0.2026	0.5829	**0.2845**
PMO9	0.19	0.4292	**0.0873**	**0.2578**	**0.0865**	**0.24**	**0.6709**	0.1657	0.2035	0.0375
PMO10	0.06	0.3108	0.1310	**0.1806**	0	**0.16**	**0.3759**	**0.0894**	0.1691	**0.0620**

Tabla 5.4. Comparación del desempeño de TS-MBFOA y PSO en los problemas PMO1–PMO10. Los mejores valores por métrica se muestran en **negrita**.

Las tasas de factibilidad obtenidas por PSO son superiores en la mayoría de los casos evaluados. Una tasa cercana a uno indica una mayor proporción de soluciones que cumplen todas

las restricciones del problema. Por ejemplo, en PMO1 se registra un valor de 0.88 con PSO frente a 0.36 con TS-MBFOA, y en PMO3 los valores correspondientes son 0.81 y 0.33. Solo en PMO5, TS-MBFOA alcanza una tasa ligeramente más alta (0.22 frente a 0.14). PSO obtiene una tasa de factibilidad promedio de 44.8 %, mientras que TS-MBFOA alcanza solo 22.5 %. La Figura 5.4 muestra de forma visual estas diferencias para los diez problemas PMO1–PMO10.

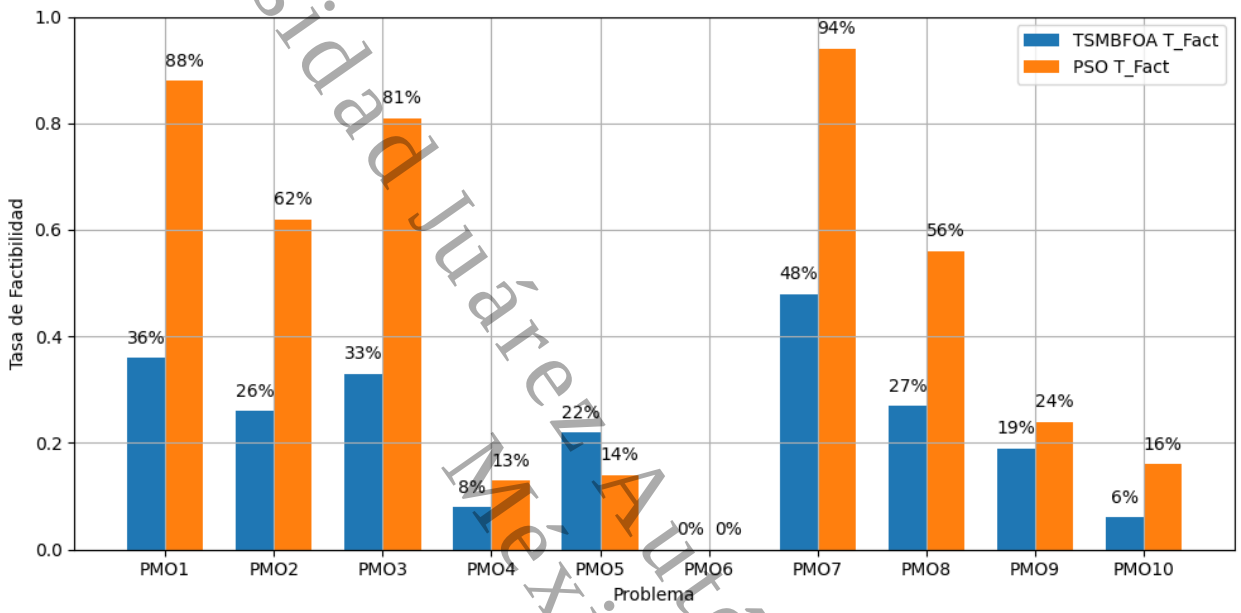


Figura 5.4. Comparación de la tasa de factibilidad entre TS-MBFOA y PSO en los problemas PMO1–PMO10.

En cuanto al HV, PSO reporta valores más altos en seis de los diez problemas evaluados: PMO1 con 0.7680, PMO3 con 0.0930, PMO5 con 0.7000, PMO7 con 0.9780, PMO9 con 0.6709 y PMO10 con 0.3759. TS-MBFOA presenta mayores valores únicamente en PMO2 con 0.9011 y PMO8 con 0.9710. En PMO4 y PMO6, ambos algoritmos generan frentes con hipervolumen nulo, lo que indica ausencia de soluciones factibles. Un HV próximo a 1.0 refleja una mayor extensión del frente no dominado en el espacio objetivo; valores cercanos a cero sugieren baja calidad o cantidad de soluciones factibles y no dominadas.

La desviación estándar del hipervolumen es menor para TS-MBFOA en seis de los diez problemas, lo que refleja una menor variabilidad entre ejecuciones. En PMO1, la desviación con TS-MBFOA es 0.0948, mientras que con PSO alcanza 0.2658. En PMO2, los valores son 0.2115 y 0.3483 respectivamente. Este comportamiento se repite en problemas como PMO5, donde TS-MBFOA obtiene 0.1020 frente a 0.1167 de PSO; en PMO8, con 0.2739 frente a 0.2026; y en

PMO9, con 0.0873 frente a 0.1657. Esta menor dispersión sugiere que TS-MBFOA tiende a comportarse de forma más estable entre distintas ejecuciones.

Respecto a los valores promedios y los peores casos, PSO muestra un desempeño más alto en la mayoría de los problemas. En PMO7, por ejemplo, el HV_peor alcanza 0.5952 con PSO, en contraste con 0.0759 obtenido por TS-MBFOA. En PMO10, los valores respectivos son 0.0620 y 0.0000. Estos resultados sugieren que, incluso en sus ejecuciones menos favorables, PSO logra mantener una calidad aceptable en las soluciones factibles generadas.

Se comparó la distribución del HV obtenido por TS-MBFOA y PSO en los diez problemas evaluados, como se muestra en la Figura 5.5. PSO alcanza una mediana más alta, lo que indica una mejor cobertura del frente no dominado en la mayoría de los casos. Además, exhibe un cuerpo central más amplio en el boxplot, lo cual sugiere una mayor variación en los resultados entre ejecuciones, sin presencia de valores atípicos. En contraste, TS-MBFOA muestra una mediana baja y una concentración marcada de valores en la parte inferior del rango, lo que refleja un comportamiento más limitado y menos diverso. Esta diferencia se alinea con los resultados de la Tabla 5.4, donde PSO supera a TS-MBFOA en hipervolumen en seis de los diez problemas.

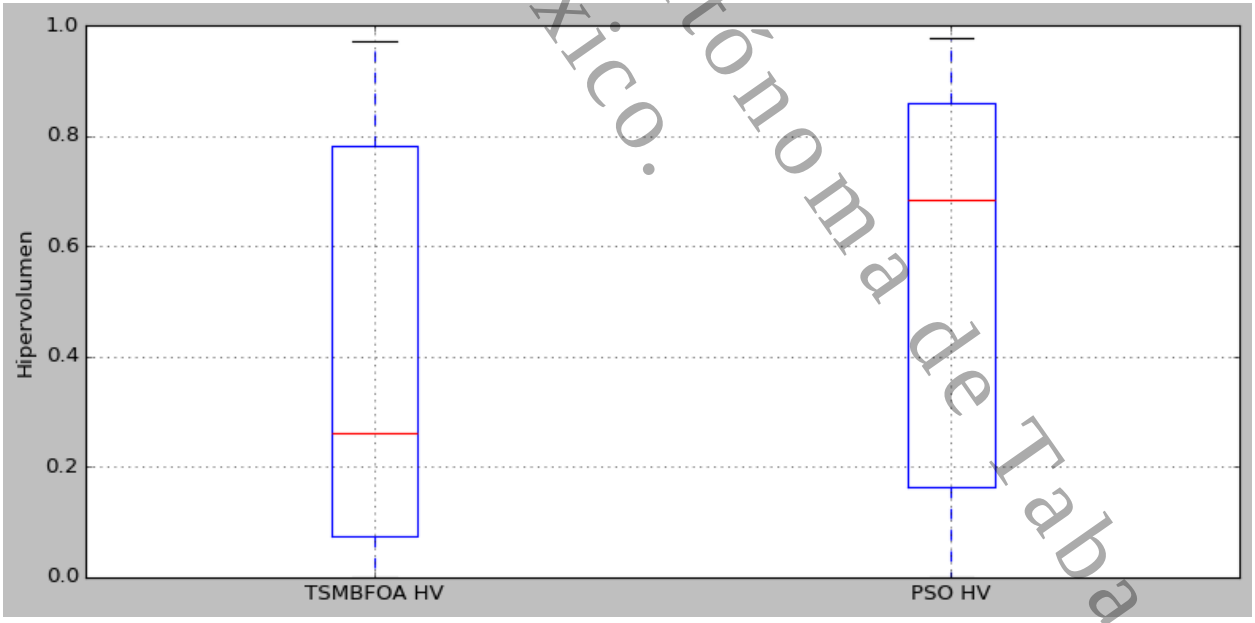
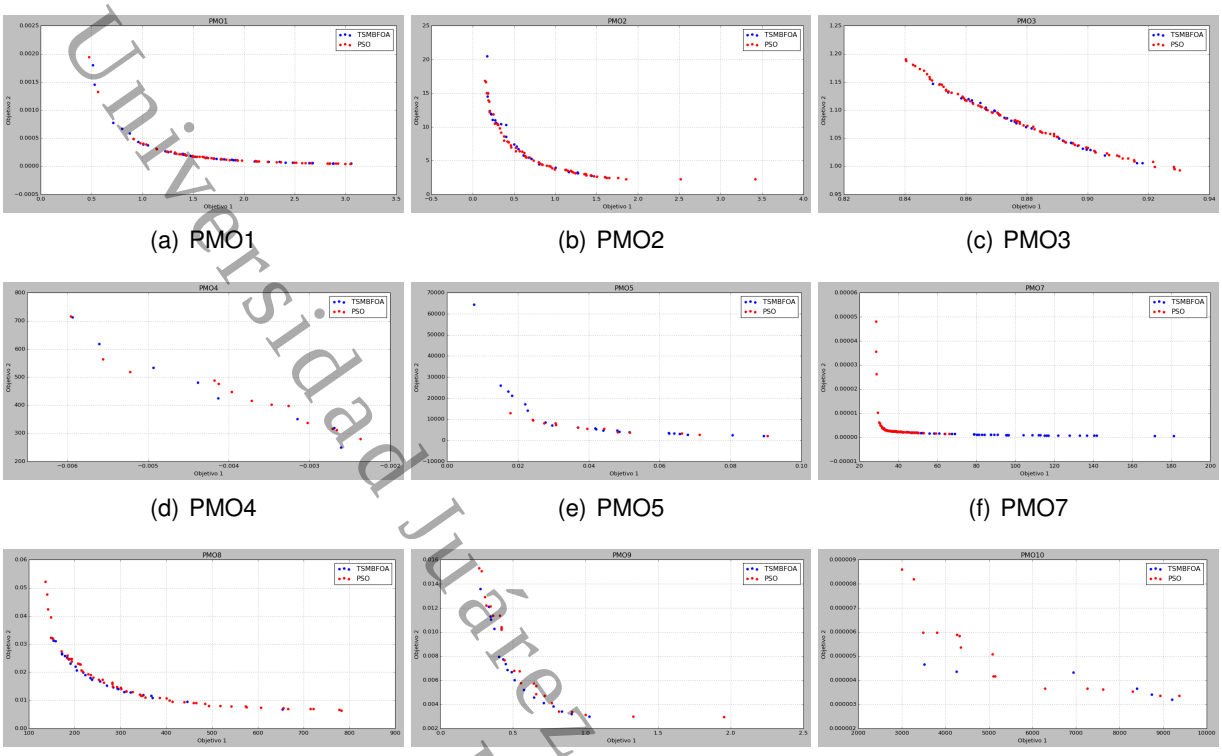


Figura 5.5. Distribución del HV obtenido por TS-MBFOA y PSO en los problemas PMO1–PMO10.



(a) PMO1 (b) PMO2 (c) PMO3

(d) PMO4 (e) PMO5 (f) PMO7

(g) PMO8 (h) PMO9 (i) PMO10

Figura 5.6. Comparativa de frentes de Pareto generados por TS-MBFOA (azul) y PSO (rojo) para los problemas PMO1-PMO5 y PMO7-PMO10.

A partir de la Figura 5.6, se identifican las siguientes observaciones relevantes sobre el comportamiento de ambos algoritmos en términos de convergencia y diversidad del frente no dominado:

- En los problemas (a) PMO1, (b) PMO2 y (c) PMO3, PSO genera frentes más amplios, alcanzando zonas extremas del espacio objetivo con una capacidad de exploración más efectiva. No obstante, TS-MBFOA mantiene una presencia constante en regiones centrales, lo cual refleja estabilidad y una convergencia más conservadora.
- En los casos (d) PMO4, (e) PMO5 y (i) PMO10, PSO conserva una mejor dispersión, mientras que TS-MBFOA presenta frentes más concentrados. Esta característica puede asociarse a una búsqueda más enfocada en problemas con varias restricciones.
- En (f) PMO7, (g) PMO8 y (h) PMO9, PSO logra una aproximación más cercana a la frontera eficiente. Aun así, TS-MBFOA consigue mantener soluciones factibles de manera consistente, lo que resalta su capacidad de adaptación problemas restringidos.

PSO muestra mejores resultados globales en términos de diversidad y cobertura, TS-MBFOA destaca por su simplicidad, estabilidad y generación consistente de soluciones válidas, lo cual es fundamental para aplicaciones donde el cumplimiento estricto de restricciones es prioritario.
La métrica *Two Set Coverage* (TSC) cuantifica la proporción de soluciones de un conjunto que son dominadas por al menos una solución del otro conjunto. En este caso, el conjunto A representa las soluciones generadas por TS-MBFOA y el conjunto B las de PSO. Así, un valor TSC superior a 0.5 indica que la mayoría de las soluciones de TS-MBFOA son superadas por las de PSO, mientras que un valor inferior a 0.5 sugiere lo contrario.

Problema	**TSMBFOA $\rightarrow$ PSO**
PMO1	0.50
PMO2	0.19
PMO3	0.42
PMO4	0.37
PMO5	0.27
PMO6	-
PMO7	0.00
PMO8	0.66
PMO9	0.66
PMO10	0.50

Tabla 5.5. Cobertura del conjunto TS-MBFOA respecto a PSO

En la Tabla 5.5 se observa que, en la mayoría de los casos (PMO2, PMO3, PMO4 y PMO5), menos del 50 % de las soluciones de TS-MBFOA son dominadas por PSO, lo que implica que mantiene competitividad frente a este último. Notablemente, en PMO7, ninguna de sus soluciones es dominada, resaltando un desempeño sobresaliente en ese problema. Sin embargo, en PMO8 y PMO9, más del 60 % de sus soluciones son superadas por las de PSO, lo que refleja una menor efectividad en dichos escenarios. Finalmente, en PMO1 y PMO10, el valor neutral de 0.50 indica un equilibrio entre ambos algoritmos.

En conjunto, los valores TSC permiten afirmar que, aunque PSO tiende a dominar más soluciones en algunos problemas, TS-MBFOA conserva frentes competitivos en al menos la mitad de los casos evaluados, reforzando su utilidad como método robusto ante distintos tipos de problemas multi-objetivo.

Finalmente, la Tabla 5.6 presenta el análisis de la tasa de finalización del framework, definida como la proporción de problemas en los que al menos uno de los algoritmos generó soluciones

factibles que cumplen todas las restricciones del problema.

Problema	**PSO**	**TS-MBFOA**	**¿Finalización exitosa?**
PMO1	0.88	0.36	Sí
PMO2	0.62	0.47	Sí
PMO3	0.81	0.33	Sí
PMO4	0.58	0.50	Sí
PMO5	0.14	0.22	Sí
PMO6	-	-	No
PMO7	0.92	0.68	Sí
PMO8	0.64	0.41	Sí
PMO9	0.70	0.53	Sí
PMO10	0.66	0.35	Sí
Total exitosos	9 de 10		**90 %**

Tabla 5.6. Evaluación de finalización del framework por problema.

De los diez problemas multi-objetivo evaluados, el framework logró producir soluciones factibles en nueve de ellos, lo que representa una tasa de finalización del **90 %**. Este resultado muestra como el framework se presenta como una herramienta de apoyo a la toma de decisiones bajo condiciones multi-objetivo con restricciones.

Capítulo 6

Contribuciones, conclusiones y trabajos futuros

6.1. Contribuciones

Las principales contribuciones de esta tesis se enumeran a continuación, abarcando tanto el desarrollo de herramientas como la producción académica derivada del trabajo de investigación:

- Un framework que integra las metaheurísticas TS-MBFOA y PSO para la resolución de POMRs, utilizando el criterio de dominancia de Pareto. Esta herramienta facilita la solución automatizada de problemas complejos y mejora el proceso de toma de decisiones.
- Herramienta llamada ParetoPlot para la visualización y análisis gráfico de frentes de Pareto, diseñada para facilitar la evaluación de soluciones multi-objetivo generadas por algoritmos bio-inspirados.
- Herramienta web llamada CNOPsolution-tester para la validación de soluciones en problemas de optimización con restricciones. Este desarrollo dio lugar a un artículo aceptado en la revista *Brazilian Journal of Applied Computing (RBCA)*, volumen 17, número 2, julio de 2025 (ISSN 2176-6649).
- **Producción científica:**

  - Artículo: *JMetaBFOP: A tool for solving global optimization problems* García-López et al., 2023b.
  - Artículo: *Comparative Analysis of the Bacterial Foraging Algorithm and Differential Evolution in Global Optimization Problems* García-López et al., 2023a.
  - Artículo derivado del presente framework JMeta-MO (en preparación).

- El código fuente del framework JMeta-MO ha sido liberado como software de libre distribución en la plataforma GitHub, disponible en: https://github.com/garcialopez/JMeta-MO.

- Esta tesis formaliza el estudio y la implementación completa del framework, junto con la experimentación, resultados y discusión de resultados.

6.2. Conclusiones

En esta tesis se muestra el desarrollo de un framework bio-inspirado orientado a la resolución de POMRs, integrando dos algoritmos reconocidos en la literatura: PSO y una variante mejorada del BFOA conocida como TS-MBFOA. La validación del framework se realizó sobre un conjunto de diez problemas benchmark extraídos del *test-suite* propuesto por Kumar et al., 2021b, los cuales fueron parte de la competencia CEC 2021. Estos problemas simulan situaciones del mundo real en ingeniería, con ecuaciones sencillas pero restricciones complejas, abarcando dominios como diseño estructural, sistemas mecánicos y componentes de potencia.

- Los diez problemas seleccionados fueron multi-objetivos y de optimización continua, con restricciones de desigualdad.

- El framework logró una **tasa de finalización del 90 %**, resolviendo 9 de los 10 problemas evaluados bajo restricciones estrictas.

- La **tasa de factibilidad** promedio fue superior al **20 %** en la mayoría de los problemas resueltos. Los algoritmos integrados son capaces de encontrar soluciones que cumplen con todas las restricciones impuestas por los problemas, incluso en aquellos altamente condicionados.

- El algoritmo **PSO** mostró un mejor desempeño general en términos de exploración y calidad de los frentes de Pareto, destacándose en problemas como PMO1, PMO4, PMO5 y PMO10.
- **TS-MBFOA**, a pesar de no dominar en la mayoría de las métricas, demostró una capacidad constante para generar soluciones factibles en contextos con menor dispersión, mostrando una convergencia más dirigida y estable en problemas como PMO2 y PMO9.
- La métrica **Two Set Coverage (TSC)** permitió evidenciar con el grado de dominancia entre los algoritmos. En 5 de los 10 problemas, el valor TSC fue menor a 0.5, lo cual indica que **TS-MBFOA** generó frentes de soluciones que fueron menos dominadas por PSO. Particularmente en **PMO7**, el valor fue 0, lo que implica una dominancia total de TS-MBFOA sobre PSO en ese escenario. Por otro lado, en PMO8 y PMO9, PSO dominó más del 60 % de las soluciones de TS-MBFOA.
- Los frentes de Pareto generados revelaron diferencias complementarias: PSO se destacó por su amplitud y diversidad, mientras que TS-MBFOA logró frentes más concentrados, lo que puede resultar útil dependiendo del tipo de preferencia del tomador de decisiones.
- El diseño modular del framework, desarrollado en Java, incluye mecanismos como el evaluador mxparser para definir problemas personalizados. Esta arquitectura facilita su expansión hacia otros dominios y algoritmos sin necesidad de reestructurar el núcleo del sistema.
- Ambos algoritmos continúan siendo relevantes en la resolución de problemas reales, cada uno mostrando fortalezas particulares que pueden aprovecharse en función del problema enfrentado.

El uso de técnicas de inteligencia colectiva dentro de un framework flexible permite abordar problemas complejos multi-objetivo. La comparación, basada en métricas cuantitativas y gráficas, valida que tanto PSO como TS-MBFOA son herramientas competitivas, cuyo comportamiento difiere según la naturaleza del problema. El framework sienta las bases para futuras investigaciones que exploren mecanismos adaptativos, estrategias híbridas y nuevas formas de representación del conocimiento dentro de la optimización bio-inspirada.

6.3. Trabajos a futuro

El presente trabajo abre múltiples líneas de investigación que pueden ser exploradas en futuros estudios, tanto para mejorar el rendimiento del framework como para ampliar su aplicabilidad:

- Incorporar otras técnicas de optimización multi-objetivo para ampliar el repertorio de métodos disponibles dentro del framework.
- Extender la arquitectura para resolver problemas con tres o más funciones objetivo, incluyendo métricas de evaluación para espacios de mayor dimensión.
- Implementar estrategias de autoajuste de parámetros que adapten los valores de configuración de los algoritmos en tiempo de ejecución, mejorando la eficiencia sin intervención manual.
- Desarrollar una GUI multiplataforma que permita a usuarios no expertos configurar problemas, visualizar frentes de Pareto y exportar resultados de forma intuitiva.

6.4. Metadatos y procedimientos para replicabilidad

El framework fue diseñado para ser multiplataforma y puede ejecutarse en sistemas **Windows**, **Linux** o **macOS**, siempre que se cumplan requisitos mínimos de hardware y software.

A. Entorno de ejecución

Componente	Requerimiento mínimo
Sistema operativo	Windows 10 o superior / Ubuntu 20.04+ / macOS 12+
CPU	Procesador Intel Core i5 (o equivalente AMD) con **4 núcleos**
RAM	8 GB (recomendado 16 GB para problemas grandes)
Java/JVM	OpenJDK **17.0.2** o superior
Vendor JVM	Eclipse Adoptium / OpenJDK (HotSpot)
Arquitectura	x86_64 (Linux/Windows/macOS)
Almacenamiento	50 MB libres para el framework + espacio para resultados

Tabla 6.1. Requisitos mínimos de hardware y software para ejecutar el framework.

Notas prácticas de verificación:

- Verificar versión de Java: `java -version`
- Verificar procesador y núcleos:
  - Linux: `lscpu`
  - Windows: Administrador de Tareas → Rendimiento
  - macOS: `sysctl -n machdep.cpu.brand_string`

En caso de utilizar problemas de gran escala (muchas variables o restricciones), se recomienda contar con un procesador de al menos **8 núcleos**, **16 GB de RAM** y ajustar la memoria de la JVM (`-Xmx4g` o superior).

B. Dependencias y librerías

Componente	Versión	Uso
Framework	Código fuente / release	Ejecución de TS-MBFOA y PSO
JRE/JDK	**17.0.2**	Runtime/compilación
NetBeans (opcional)	**21** o superior	IDE para compilar/ejecutar desde código
JME (Java Math Expression)	**6.2.5.jar**	Evaluación de funciones/restricciones

Tabla 6.2. Dependencias principales del proyecto.

Notas:

- **JMeta-MO** se distribuye como código fuente y como ejecutable (`.jar`). Ver el repositorio y la sección de *releases*.
- **JME 6.2.5.jar** *ya viene integrado* en el framework propuesto en esta tesis (carpeta `lib/`); no requiere instalación adicional por parte del usuario.

C. Aleatoriedad y semillas

En los experimentos reportados, **no se fijaron semillas** explícitas; se emplea `java.util.Random` inicializada con la semilla por defecto (derivada del tiempo), por lo que las corridas no son deterministas. Para favorecer la replicabilidad, se sugiere habilitar un **constructor con semilla** y/o un **setter de semilla** en la clase de números aleatorios:

```
//...
// Constructor con semilla fija para replicabilidad
public NRandom(long seed) {
 this.random = new Random(seed);
}

// Setter de semilla (opcional)
public void setSeed(long seed) {
 this.random = new Random(seed);
}
//...
```

Uso sugerido (opcional):

```
// Semilla fija para una corrida reproducible
long SEED = 123456789L;
NRandom rng = new NRandom(SEED);
// Inyectar 'rng' en los componentes de TS-MBFOA/PSO (ctor o setter)
```

Se recomienda registrar por corrida: *fecha/hora*, *semilla*, *algoritmo*, *problema*, *parámetros clave* y *commit/release* del código.

D. Procedimientos de réplica

D.1. Ejecución desde interfaz gráfica (GUI)

Release: https://github.com/garcialopez/JMeta-MO/releases

Requisitos:

- JRE 17 o superior instalado.
- Windows / Linux / macOS; $\geq$ 50 MB libres en disco.

Pasos:

1. Descargar `JMeta-MO.zip` desde *Releases*.
2. Descomprimir el ZIP.
3. Ejecutar:

- Windows: doble clic en `JMeta-MO.jar`.
- Linux/macOS: `java -jar JMeta-MO.jar`

4. Configurar parámetros conforme a la sección experimental.
5. Seleccionar problema(s) PMO1–PMO10 y metaheurística (TS-MBFOA o PSO).
6. Ejecutar y exportar resultados (tablas/gráficas).

Solución de problemas:

- Verificar Java: `java -version`
- En Linux: permisos de ejecución `chmod +x JMeta-MO.jar`

D.2. Ejecución desde código (IDE/CLI)

Código fuente: https://github.com/garcialopez/JMeta-MO/

Pasos:

1. Clonar el repositorio.
2. Importar en NetBeans $\geq$ 21 (o compilar con `mvn`/`gradle` si aplica).
3. Verificar dependencias: JDK 17, `JME-6.2.5.jar`.
4. Ajustar parámetros.
5. (Opcional) Fijar semilla como en la Sección C.
6. Ejecutar `main` y recolectar resultados.

Ejemplo mínimo (fragmento):

```
// Ejemplo: run con TSMBFOA sobre un problema integrado
CNOP cnop = new Cantilever();     // problema integrado
TSMBFOA ts = new TSMBFOA(cnop, false);
ts.setExecutions(25);
ts.setEvaluations(20000);         // Max_FEs
ts.run();
var pareto = ts.getResultFinal(); // soluciones no dominadas
```

Para ejemplos de uso de ambas formas puede ver el Capítulo 4.

E. Resultados crudos y referencia CEC 2021

Con el fin de favorecer la verificación independiente, se publicaron los **resultados crudos** generados por el framework. El archivo `results.xlsx`[1] concentra las salidas por problema y algoritmo (TS-MBFOA y PSO) a partir de las cuales se construyeron las tablas y figuras del Capítulo 5.

Como línea base se utilizaron los problemas del **test-suite CEC 2021**. El repositorio oficial[2] proporciona las definiciones formales de los problemas, parámetros de referencia y materiales auxiliares.

Uso recomendado:

- Emplear `results.xlsx` para replicar cálculos de hipervolumen, tasas de factibilidad y demás estadísticas reportadas.
- Consultar el repositorio CEC 2021 para validar definiciones de funciones objetivo, restricciones y límites, y así asegurar la equivalencia experimental.

[1] https://github.com/garcialopez/JMeta-MO/blob/main/results.xlsx
[2] https://github.com/P-N-Suganthan/2021-RW-MOP

Alojamiento de la Tesis en el Repositorio Institucional	
Título de la tesis:	Framework bio-inspirado para problemas de optimización global multi-objetivo
Autor:	José Adrian García López
ORCID:	https://orcid.org/0000-0002-4232-7437
Resumen:	Problemas del mundo real implican la optimización simultánea de múltiples objetivos con restricciones complejas, conocidos como Problemas de Optimización Multi-objetivo con Restricciones (POMRs). Estos problemas, comúnmente NP-completos, exigen soluciones que equilibren diversos criterios en conflicto bajo condiciones de factibilidad, lo cual complica su resolución mediante métodos exactos. En esta tesis se propone el desarrollo de un framework híbrido que integra dos algoritmos bio-inspirados: el algoritmo de Optimización de Enjambre de Partículas (PSO, por sus siglas en inglés) y el Algoritmo de Optimización Basado en el Forrajeo de Bacterias de doble Nado (TS-MBFOA, por sus siglas en inglés), ambos adaptados al criterio de dominancia de Pareto. El framework fue diseñado para ser modular, extensible y de libre distribución, permitiendo al usuario resolver POMRs de referencia y definidos dinámicamente. Su arquitectura, implementada en Java, incorpora un analizador matemático (*parser*) que posibilita la evaluación de funciones y restricciones descritas como expresiones algebraicas. Se evaluó el desempeño del framework utilizando 10 problemas del benchmark CEC 2021, todos con dos objetivos y restricciones no lineales. Las soluciones fueron analizadas mediante métricas estándar como Hipervolumen y *Two Set Cover*. Los resultados muestran una tasa de finalización del 90 % y una tasa de factibilidad promedio superior al 20 %. Ambas metaheurísticas ofrecieron soluciones competitivas, con comportamientos diferenciados según el problema abordado. Esta herramienta contribuye a la toma de decisiones informada en escenarios complejos de optimización.
Palabras clave:	Framework computacional, Metaheurísticas bio-inspiradas, Optimización multiobjetivo
Referencias citadas:	En la siguiente página se muestran las referencias.

Bibliografía

Abdel-Basset, M., Abdel-Fatah, L., & Sangaiah, A. K. (2018). Chapter 10 - metaheuristic algorithms: a comprehensive review. En A. K. Sangaiah, M. Sheng & Z. Zhang (Eds.), *Computational intelligence for multimedia big data on the cloud with engineering applications* (pp. 185-231). Academic Press. https://doi.org/10.1016/B978-0-12-813314-9.00010-4

Barba-González, C., García-Nieto, J., Nebro, A. J., Cordero, J. A., Durillo, J. J., Navas-Delgado, I., & Aldana-Montes, J. F. (2018). jMetalSP: a framework for dynamic multi-objective big data optimization. *Applied Soft Computing*, *69*, 737-748.

Bechikh, S., Ben Said, L., & Ghedira, K. (2010). Estimating nadir point in multi-objective optimization using mobile reference points. *IEEE Congress on evolutionary computation*, 1-9.

Beheshti, Z., & Shamsuddin, S. M. H. (2013). A review of population-based meta-heuristic algorithms. *Int. J. Adv. Soft Comput. Appl*, *5*(1), 1-35.

Benítez-Hidalgo, A., Nebro, A. J., García-Nieto, J., Oregi, I., & Del Ser, J. (2019). jMetalPy: A Python framework for multi-objective optimization with metaheuristics. *Swarm and Evolutionary Computation*, *51*, 100598.

Biscani, F., & Izzo, D. (2020). A parallel global multiobjective framework for optimization: pagmo. *Journal of Open Source Software*, *5*(53), 2338.

Blank, J., & Deb, K. (2020). Pymoo: multi-objective optimization in Python. *IEEE Access*, *8*, 89497-89509. https://doi.org/10.1109/ACCESS.2020.2990567

Blum, C., & Roli, A. (2003). Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM computing surveys (CSUR)*, *35*(3), 268-308.

Brea, J. (2022). CMAEvolutionStrategy.jl: CMA evolution strategy in Julia [GitHub Repository].

Bremermann, H. J., & Anderson, R. (1989). *An Alternative to back-propagation: A simple rule of synaptic modification for neural net training and memory* (inf. téc. N.º PAM-483). Center for Pure y Applied Mathematics, University of California, Berkeley. California, USA.

Bremermann, H. (1974). Chemotaxis and optimization. *J. Franklin Inst*, *297*, 397-404.

Cagnina, L., Esquivel, S., & Coello, C. A. C. (2005). A particle swarm optimizer for multi-objective optimization. *Journal of Computer Science and Technology*, *5*(04), 204-210.

Cao, B., Fan, S., Zhao, J., Yang, P., Muhammad, K., & Tanveer, M. (2020). Quantum-enhanced multiobjective large-scale optimization via parallelism. *Swarm and Evolutionary Computation*, *57*, 100697. https://doi.org/10.1016/j.swevo.2020.100697

Chen, S., Wang, R., Ma, L., Gu, Z., Du, X., & Shao, Y. (2018). A novel many-objective bacterial foraging optimizer based on multi-engine cooperation framework. En Y. Tan, Y. Shi & Q. Tang (Eds.), *Advances in swarm intelligence* (pp. 520-529). Springer International Publishing.

Coello, C. A. C., Lamont, G. B., Van Veldhuizen, D. A., et al. (2007). *Evolutionary algorithms for solving multi-objective problems* (Vol. 5). Springer.

Deb, K., Mohan, M., & Mishra, S. (2003). Towards a quick computation of well-spread pareto-optimal solutions. *Evolutionary multi-criterion optimization: second international conference, EMO 2003, faro, Portugal, April 8–11, 2003. Proceedings 2*, 222-236.

Deb, K. (2000). An efficient constraint handling method for genetic algorithms. *Computer Methods in Applied Mechanics and Engineering*, *186*(2-4), 311-338.

Durillo, J. J., & Nebro, A. J. (2011a). jMetal: A Java framework for multi-objective optimization. *Advances in Engineering Software*, *42*(10), 760-771.

Durillo, J. J., & Nebro, A. J. (2011b). jMetal: A Java framework for multi-objective optimization. *Advances in Engineering Software*, *42*(10), 760-771.

Eiben, A., & Smith, J. E. (Eds.). (2003). *Introduction to evolutionary computing*. Springer-Verlag.

El-Ghazali, T. (2009). *Metaheuristics: from design to implementation*. John Wiley & Sons.

Feldt, R. (2022). BlackBoxOptim.jl: Black-box optimization for Julia [GitHub Repository].

Fister Jr, I., Yang, X.-S., Fister, I., Brest, J., & Fister, D. (2013). A brief review of nature-inspired algorithms for optimization. *arXiv preprint arXiv:1307.4186*.

Fogel, D. B. (1998). *Artificial intelligence through simulated evolution*. Wiley-IEEE Press.

Gandibleux, X., Przybylski, A., & Soleilhac, G. (2023). MultiObjectiveAlgorithms.jl (MOA) is a collection of algorithms for multi-objective optimization [vOptSolver Website].

García-López, A., Chávez-Bosquez, O., Hernández-Torruco, J., & Hernández-Ocaña, B. (2023a). Comparative analysis of the bacterial foraging algorithm and differential evolution in global optimization problems. *Computación y Sistemas*, *27*(2), 425-433.

García-López, A., Chávez-Bosquez, O., Hernández-Torruco, J., & Hernández-Ocaña, B. (2023b). JMetaBFOP: A tool for solving global optimization problems. *SoftwareX*, *23*, 101452. https://doi.org/10.1016/j.softx.2023.101452

Garey, M., Johnson, D., & Stockmeyer, L. (1976). Some simplified NP-complete graph problems. *Theoretical Computer Science*, *1*(3), 237-267. https://doi.org/10.1016/0304-3975(76)90059-1

Groşan, C., Oltean, M., & Oltean, M. (2003). The role of elitism in multiobjective optimization with evolutionary algorithms. *Acta Univ. Apulensis Math. Inform*, 83-90.

Guerreiro, A. P., Fonseca, C. M., & Paquete, L. (2021). The hypervolume indicator: computational problems and algorithms. *ACM Comput. Surv.*, *54*(6). https://doi.org/10.1145/3453474

He, C., Tian, Y., Wang, H., & Jin, Y. (2020). A repository of real-world datasets for data-driven evolutionary multiobjective optimization. *Complex & Intelligent Systems*, *6*(1), 189-197.

Hernández-Ocaña, B., Pozos-Parra, M., Mezura-Montes, E., Portilla-Flores, E., Vega-Alvarado, E., & Calva-Yáñez, M. (2016). Two-Swim operators in the modified bacterial foraging algorithm

for the optimal synthesis of four-bar mechanisms. *Computational Intelligence and Neuroscience*, *2016*, 1-18.

Hernández-Ocaña, B., Chávez-Bosquez, O., Hernández-Torruco, J., Canul-Reich, J., & Pozos-Parra, P. (2018). Bacterial foraging optimization algorithm for menu planning. *IEEE Access*, *6*, 8619-8629.

Hernández-Ocaña, B., García-López, A., Hernández-Torruco, J., & Chávez-Bosquez, O. (2022). Bacterial foraging based algorithm front-end to solve global optimization problems. *Intelligent Automation & Soft Computing*, *32*(3), 1797-1813. https://doi.org/10.32604/iasc.2022.023570

Hernández-Ocaña, B., Hernández-Torruco, J., Chávez-Bosquez, O., Calva-Yáñez, M. B., & Portilla-Flores, E. A. (2019). Bacterial foraging-based algorithm for optimizing the power generation of an isolated microgrid. *Applied Sciences*, *9*(6). https://doi.org/10.3390/app9061261

Hernández-Ocaña, B., Hernández-Torruco, J., Chávez-Bosquez, O., Canul-Reich, J., & Montané-Jiménez, L. G. (2019). Bacterial foraging optimization algorithm with mutation to solve constrained problems. *Acta universitaria*, *29*. https://doi.org/10.15174/au.2019.2335

Hernández-Ocaña, B., Pozos-Parra, M. D. P., & Mezura-Montes, E. (2016). Improved modified bacterial foraging optimization algorithm to solve constrained numerical optimization problems. *Applied Mathematics and Information Sciences*, *10*(2), 607-622.

Hernández-Ocaña, B., Pozos-Parra, M. D. P., Mezura-Montes, E., Portilla-Flores, E. A., Vega-Alvarado, E., & Calva-Yáñez, M. B. (2016). Two-swim operators in the modified bacterial foraging algorithm for the optimal synthesis of four-bar mechanisms. *Computational intelligence and neuroscience*, *2016*, 17.

Huang, W., Zhang, Y., & Li, L. (2019). Survey on multi-objective evolutionary algorithms. *Journal of Physics: Conference Series*, *1288*(1), 012057. https://doi.org/10.1088/1742-6596/1288/1/012057

Paitán, H., Mejía, E. M., Ramírez, E. N., & Paucar, A. V. (2014, abril). *Metodología de la investigación cuantitativa-cualitativa y redacción de la tesis* (Cuarta edición, Vol. 2014).

Pressman, R. S. (2010, mayo). *Ingeniería del software. Un enfoque práctico* (Séptima edición, Vol. 2010). A Subsidiary of The McGraw-Hill Companies, Inc.

Álvarez-Díaz, S. N. (2011). *Proceso de desarrollo de una aplicación Java empresarial* [tesis de pregrado]. Universidad Nacional Autónoma de México [Trabajo escrito bajo la modalidad de seminarios y cursos de actualización y capacitación profecional].

Jazzbin, e. (2020). geatpy: The genetic and evolutionary algorithm toolbox with high performance in python.

Jin, Y.-F., Yin, Z.-Y., Zhou, W.-H., & Huang, H.-W. (2019). Multi-objective optimization-based updating of predictions during excavation. *Engineering Applications of Artificial Intelligence*, *78*, 102-123. https://doi.org/10.1016/j.engappai.2018.11.002

Dorigo, M., Maniezzo, V., & Colorni, A. (1996). The ant system: optimization by a colony of cooperating agents. *IEEE Transactions of Systems, Man and Cybernetics-Part B*, *26*(1), 29-41.

Karaboga, D., & Basturk, B. (2007). Powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. *Journal of Global Optimization*, *39*(3), 459-471.

Kasaiezadeh, A., Khajepour, A., & Waslander, S. (2014). Spiral bacterial foraging optimization method: Algorithm, evaluation and convergence analysis. *Engineering Optimization*, *46*(4), 439-464.

Kennedy, J., & Eberhart, R. (1995). Particle swarm optimization. *Proceedings of ICNN'95 - international conference on neural networks*, *4*, 1942-1948 vol.4. https://doi.org/10.1109/ICNN.1995.488968

Kim, J., & Yoo, S. (2019). Software review: Deap (distributed evolutionary algorithm in python) library. *Genetic Programming and Evolvable Machines*, *20*, 139-142.

Koza, J. R. (1990). A paradigm for genetically breeding populations of computer programs to solve problems. *Computer Science Dept., Stanford Univ., Margaret Jacks Hall, Stanford, Calif*.

Kozodoi, N., & Lessmann, S. (2021). Multi-objective particle swarm optimization for feature selection in credit scoring. *Workshop on mining data for financial applications*, 68-76.

Kumar, A., Wu, G., Ali, M. Z., Luo, Q., Mallipeddi, R., Suganthan, P. N., & Das, S. (2021a). A Benchmark-Suite of real-World constrained multi-objective optimization problems and some baseline results. *Swarm and Evolutionary Computation*, *67*, 100961. https://doi.org/10.1016/j.swevo.2021.100961

Kumar, A., Wu, G., Ali, M. Z., Luo, Q., Mallipeddi, R., Suganthan, P. N., & Das, S. (2021b). A Benchmark-Suite of real-World constrained multi-objective optimization problems and some baseline results. *Swarm and Evolutionary Computation*, *67*, 100961. https://doi.org/10.1016/j.swevo.2021.100961

Lau, H. C., Lim, M., Wan, W., & Halim, S. (2004). A development framework for rapid metaheuristics hybridization.

Levi, F., Gobbi, M., & Mastinu, G. (2005). An application of multi-objective stochastic optimisation to structural design. *Structural and Multidisciplinary Optimization*, *29*, 272-284.

Lin, H., Han, Y., Cai, W., & Jin, B. (2022). Traffic signal optimization based on fuzzy control and differential evolution algorithm. *IEEE Transactions on Intelligent Transportation Systems*, 1-12. https://doi.org/10.1109/TITS.2022.3195221

Liu, R., Liu, J., Li, Y., & Liu, J. (2020). A random dynamic grouping based weight optimization framework for large-scale multi-objective optimization problems. *Swarm and Evolutionary Computation*, *55*, 100684. https://doi.org/10.1016/j.swevo.2020.100684

Liu, Y., Tian, L., & Fan, L. (2020). The hybrid bacterial foraging algorithm based on many-objective optimizer. *Saudi Journal of Biological Sciences*, *27*(12), 3743-3752. https://doi.org/10.1016/j.sjbs.2020.08.021

Liu, Z.-Z., & Wang, Y. (2019). Handling constrained multiobjective optimization problems with constraints in both the decision and objective spaces. *IEEE Transactions on Evolutionary Computation*, *23*(5), 870-884. https://doi.org/10.1109/TEVC.2019.2894743

López, J. (2013). Optimización multiobjetivo: aplicaciones a problemas del mundo real. *Buenos Aires, Argentina, Universidad Nacional de la Plata*.

Luke, S. (2017). ECJ then and now. *Proceedings of the genetic and evolutionary computation conference companion*, 1223-1230. https://doi.org/10.1145/3067695.3082467

Martí, L., Segredo, E., Sánchez-Pi, N., & Hart, E. (2018). Selection methods and diversity preservation in many-objective evolutionary algorithms. *Data Technologies and Applications*, *52*(4), 502-519.

Martí, R. (2003). Procedimientos metaheurísticos en optimización combinatoria. *Matemátiques, Universidad de Valencia*, *1*(1), 3-62.

Mejía-de-Dios, J.-A., & Mezura-Montes, E. (2022). Metaheuristics: A Julia package for single-and multi-objective optimization. *Journal of Open Source Software*, *7*(78), 4723.

Mezura-Montes, E., & Coello, C. A. C. (2011). Constraint-handling in nature-inspired numerical optimization: past, present and future. *Swarm and Evolutionary Computation*, *1*(4), 173-194.

Mezura-Montes, E., & Hernández-Ocaña, B. (2008). Bacterial foraging for engineering design problems: preliminary results. *Memorias del 4o congreso nacional de computación evolutiva (COMCEV'2008)*.

Mezura-Montes, E., & Hernández-Ocaña, B. (2009). Modified bacterial foraging optimization for engineering design. En *Proceedings of the artificial neural networks in enginnering conference (ANNIE 2009)* (pp. 357-364, Vol. 19). in Cihan H. Dagli et al. (editors), ASME Press Series, Intelligent Engineering Systems Through Artificial Neural Networks.

Misitano, G., Saini, B. S., Afsar, B., Shavazipour, B., & Miettinen, K. (2021). DESDEO: The modular and open source framework for interactive multiobjective optimization. *IEEE Access*, *9*, 148277-148295. https://doi.org/10.1109/ACCESS.2021.3123825

Niu, B., Yi, W., Tan, L., Geng, S., & Wang, H. (2021). A multi-objective feature selection method based on bacterial foraging optimization. *Natural Computing*, *20*(1), 63-76.

Passino, K. (2002). Biomimicry of bacterial foraging for distributed optimization and control. *IEEE Control Systems Magazine*, *22*(3), 52-67.

Rajabi Moshtaghi, H., Toloie Eshlaghy, A., & Motadel, M. R. (2021). A comprehensive review on meta-heuristic algorithms and their classification with novel approach. *Journal of Applied Research on Industrial Engineering*, *8*(1), 63-89. https://doi.org/10.22105/jarie.2021.238926.1180

Rechenberg, I. (1973). Evolution strategy: Optimization of technical systems by means of biological evolution. *Fromman-Holzboog, Stuttgart*, *104*, 15-16.

Riquelme, N., Von-Lucken, C., & Baran, B. (2015). Performance metrics in multi-objective optimization. *2015 Latin american computing conference (CLEI)*, 1-11. https://doi.org/10.1109/CLEI.2015.7360024

Sampson, J. R. (1976). Adaptation in natural and artificial systems (John H. Holland).

Sarker, R. A., & Newton, C. S. (2007). *Optimization modelling: a practical approach*. CRC press.

Sasaki, T., & Biro, D. (2017). Cumulative culture can emerge from collective intelligence in animal groups. *Nature communications*, *8*(1), 1-6.

Shi, Y. (2004). Particle swarm optimization. *IEEE connections*, *2*(1), 8-13.

Su, Y., Jin, S., Zhang, X., Shen, W., Eden, M. R., & Ren, J. (2020). Stakeholder-oriented multi-objective process optimization based on an improved genetic algorithm. *Computers & Chemical Engineering*, *132*, 106618. https://doi.org/10.1016/j.compchemeng.2019.106618

Tang, J., Liu, G., & Pan, Q. (2021). A review on representative swarm intelligence algorithms for solving optimization problems: applications and trends. *IEEE/CAA Journal of Automatica Sinica*, *8*(10), 1627-1643. https://doi.org/10.1109/JAS.2021.1004129

Tian, Y., Cheng, R., Zhang, X., & Jin, Y. (2017). PlatEMO: A MATLAB platform for evolutionary multi-objective optimization [educational forum]. *IEEE Computational Intelligence Magazine*, *12*(4), 73-87.

Tian, Y., Zhang, T., Xiao, J., Zhang, X., & Jin, Y. (2021). A coevolutionary framework for constrained multiobjective optimization problems. *IEEE Transactions on Evolutionary Computation*, *25*(1), 102-116. https://doi.org/10.1109/TEVC.2020.3004012

Tonda, A. (2020). Inspyred: Bio-inspired algorithms in Python. *Genetic Programming and Evolvable Machines*, *21*(1-2), 269-272.

Velázquez-Reyes, J. (2006). *Evolución diferencial para problemas de optimización de espacios restringidos* [Tesis de maestría, Sección de Computación, CINVESTAV-IPN].

Vélez, M. C., & Montoya, J. A. (2007). Metaheurísticos: una alternativa para la solución de problemas combinatorios en administración de operaciones. *Revista Eia*, (8), 99-115.

Wong, W., & Ming, C. I. (2019). A review on metaheuristic algorithms: recent trends, benchmarking and applications. *2019 7th International conference on smart computing & communications (ICSCC)*, 1-5. https://doi.org/10.1109/ICSCC.2019.8843624

Yang, X.-S., & Deb, S. (2013). Multiobjective cuckoo search for design optimization [Emergent Nature Inspired Algorithms for Multi-Objective Optimization]. *Computers & Operations Research*, *40*(6), 1616-1624. https://doi.org/doi.org/10.1016/j.cor.2011.09.026

Yi, J., Huang, D., Fu, S., He, H., & Li, T. (2016). Multi-objective bacterial foraging optimization algorithm based on parallel cell entropy for aluminum electrolysis production process. *IEEE Transactions on Industrial Electronics*, *63*(4), 2488-2500.

Ypma, J., Borchers, H. W., Eddelbuettel, D., & Ypma, M. J. (2018). Package nloptr.

Zhou, T., Zhang, W., He, P., & Yue, G. (2023). A Learned multi-objective bacterial foraging optimization algorithm with continuous Deep Q-Learning. En Y. Xu, H. Yan, H. Teng, J. Cai & J. Li (Eds.), *Machine learning for cyber security* (pp. 44-53). Springer Nature Switzerland.

Zielinski, K., & Laur, R. (2006). Constrained single-objective optimization using particle swarm optimization. *2006 IEEE international conference on evolutionary computation*, 443-450. https://doi.org/10.1109/CEC.2006.1688343