Universidad Autónoma
Metropolitana

UNIDAD CUAJIMALPA

Análisis de la Calidad de Servicio en Redes de Telecomunicaciones Utilizando Algoritmos de Enjambre de Partículas

Proyecto Terminal

QUE PRESENTA:
JOSÉ ALBERTO POSADAS GUDIÑO

Licenciatura en Ingeniería en Computación

Departamento de Matemáticas Aplicadas y Sistemas

División de Ciencias Naturales e Ingeniería

Asesor y Responsable de la tesis:
DR. EDWIN MONTES OROZCO
DRA. KAREN SAMARA MIRANDA CAMPOS

Mayo de 2025

Índice general

Índice de figuras

Capítulo 1

Introducción

En la actualidad, se volvió necesario estar conectados a Internet para realizar actividades cotidianas como enviar mensajes, hacer videollamadas, ver películas en línea o trabajar desde casa. Las redes de telecomunicaciones permiten la transmisión eficiente y confiable de datos, siendo esenciales para las actividades cotidianas de la sociedad moderna. Sin embargo, a medida que más personas y dispositivos se conectan a Internet al mismo tiempo, mantener una buena calidad de conexión se vuelve un gran desafío.

Para que una red de telecomunicaciones funcione correctamente, es importante que cumpla con ciertos aspectos como ofrecer una buena velocidad, evitar retrasos en la entrega de la información, no perder datos en el camino y brindar una cobertura adecuada. A estos aspectos se les conoce como **Calidad de Servicio**. Si alguno de estos aspectos falla, el usuario lo nota enseguida: hay interrupciones en las llamadas, demoras al cargar contenido o problemas al bajar archivos.

Con el fin de mejorar este servicio, han surgido diferentes tecnologías como **Wi-Fi 6**, una versión más avanzada de las redes inalámbricas que usamos todos los días. Esta tecnología permite que más dispositivos se conecten al mismo tiempo sin saturar la red, haciendo que está funcione de una mejor manera. Pero incluso con estas mejoras, es necesario encontrar formas de organizar y manejar la red para que funcione lo mejor posible.

Este proyecto propone un enfoque basado en la observación de comportamientos colectivos en la naturaleza. En ella, se observa cómo ciertos grupos

de animales, como peces o aves, se desplazan en conjunto siguiendo patrones coordinados. A partir de estas observaciones, se han desarrollado algoritmos que imitan dichos comportamientos. Uno de ellos es el algoritmo de **Optimización por Enjambre de Partículas**(PSO, por sus siglas en inglés). Este tipo de algoritmos permite encontrar soluciones eficientes a problemas complejos, como la distribución óptima de recursos en una red para mejorar su funcionamiento.

La idea principal de este trabajo es aplicar este algoritmo para mejorar el rendimiento de una red Wi-Fi 6 real, en este caso, para una fábrica de ropa, la cual proporcionó distintos datos con los que se pudo realizar un análisis. En este estudio, se tomaron en cuenta los principales aspectos que afectan la **Calidad de Servicio**, tales como la velocidad de transmisión, el retardo, la pérdida de paquetes y la cobertura. A partir de estos datos, se implementó el algoritmo con el objetivo de encontrar configuraciones de red más eficientes.

La implementación del código permitió simular diferentes escenarios y evaluar cómo el algoritmo puede ajustar parámetros para optimizar el desempeño. De esta forma, se busca demostrar que, es posible mejorar la calidad de la conexión.

1.1. Objetivos

En esta sección se presentan los objetivos generales y particulares del proyecto, los cuales establecen la dirección del estudio y los resultados esperados. El objetivo general define el propósito central del trabajo, mientras que los objetivos particulares detallan los pasos necesarios para alcanzar dicha meta.

1.1.1. Objetivo General

Desarrollar y validar un enfoque basado en algoritmos de enjambre de partículas para optimizar la calidad de servicio (QoS) en redes de telecomunicaciones, mejorando métricas clave como latencia, jitter, pérdida de paquetes y cobertura.

1.1.2. Objetivos Particulares

- Diseñar e implementar un algoritmo de inteligencia de enjambre que optimice la calidad de servicio en redes Wi-Fi 6.
- Identificar y definir métricas clave de QoS para redes Wi-Fi 6, y formular funciones objetivo que permitan su optimización eficiente.
- Realizar simulaciones y pruebas experimentales para evaluar el desempeño del algoritmo en distintas condiciones de red.
- Comparar el rendimiento del enfoque propuesto con metodologías convencionales para validar su eficacia.

1.2. Justificación

En el contexto actual, la creciente demanda de redes inalámbricas eficientes exige la implementación de estrategias avanzadas para optimizar la calidad del servicio (QoS). La estabilidad y el rendimiento de estas redes impactan directamente en aplicaciones críticas como videoconferencias, transmisión de datos en tiempo real y entornos industriales.

Los algoritmos de inteligencia de enjambre han demostrado ser una alternativa prometedora en la optimización de sistemas complejos debido a su capacidad de adaptación y exploración eficiente del espacio de soluciones. La aplicación de estos métodos en redes Wi-Fi 6 busca optimizar dinámicamente la asignación de recursos, minimizando métricas críticas como latencia, pérdida de paquetes y congestión, incluso en entornos de alta densidad de dispositivos.

Este proyecto busca contribuir con un enfoque innovador basado en la optimización por enjambre de partículas para mejorar el desempeño de las redes inalámbricas.

Capítulo 2

Conocimientos Preliminares

En este capítulo se presentan conceptos importantes para comprender el tema abordado en el proyecto. A lo largo de este, se profundizará en definiciones y ejemplos que muestran la relación de estos principios con el objetivo general del proyecto, permitiendo así una comprensión más detallada.

2.1. Optimización

La optimización, también es conocida como programación matemática, está se puede encontrar en el estudio de problemas complejos de decisión [37]. Es un proceso que implica asignar valores a un conjunto de variables interrelacionadas, enfocándose en un modelo diseñado para cuantificar la calidad de las decisiones del problema [30]. En otras palabras, la optimización nos ayuda a lograr integrar el sistema con sus procesos, abarcando desde el planteamiento del problema hasta el análisis que permite obtener la solución [22].

En general, la optimización se refiere a seleccionar la mejor opción dentro de un conjunto de alternativas posibles, basándose en un criterio de decisión específico. Utilizando modelos de optimización, tanto deterministas como estocásticos, se busca obtener un uso óptimo de los recursos[35].

2.1.1. Modelo

El modelo es una abstracción de un problema real de las características de un sistema, que se formula matemáticamente para obtener resultados óptimos. Debido a que las funciones matemáticas permiten utilizar procesos para analizar y resolver problemas de este estilo. Existen dos tipos principales de

modelado: el determinista y el estocástico, que se distinguen según los niveles de complejidad y riesgo involucrados[35].

Aunque existen dos tipos de modelado, este trabajo se enfoca en los problemas con un modelo estocástico. Por ello, nos enfocaremos en el concepto y las características de este modelo.

Modelo Estocástico

Los modelos estocásticos abordan la incertidumbre teniendo en cuenta la variabilidad para encontrar soluciones óptimas. Estos modelos son útiles en contextos donde los datos y las condiciones pueden cambiar de manera impredecible. Para su medición, es necesario contar con una función que permita calcular las probabilidades asociadas[35].

2.1.2. Tipos de Problemas

Dentro de la optimización, se encuentran principalmente dos clases de problemas: P y NP, los cuales se categorizan según su complejidad computacional. La complejidad de un problema se refiere al número de operaciones necesarias para resolverlo en función del tamaño de la entrada [28].

Clase P

La clase P (polynomial time) agrupa los problemas que pueden resolverse en un tiempo polinomial con un algoritmo determinista. Esto significa que, si el tamaño de la entrada crece, el tiempo de ejecución aumenta de manera razonable y controlada. Estos problemas pueden resolverse de manera eficiente en una máquina de Turing determinista, lo que los hace tratables computacionalmente incluso para entradas de gran tamaño [28] [11].

Clase NP

La clase NP (nondeterministic polynomial time) incluye problemas para los cuales no se conoce un algoritmo eficiente que los resuelva en tiempo polinomial. Sin embargo, si se proporciona una solución candidata, esta puede verificarse en tiempo polinomial usando una máquina de Turing determinista [28].

Dentro de esta clasificación se encuentran los problemas NP-completos, que son los más difíciles dentro de la clase NP. Un problema se considera NP-completo si cumple dos condiciones:

1. Se puede verificar en tiempo polinomial.

2. Se puede reducir a cualquier otro problema NP en tiempo polinomial.

Esto implica que, si se encuentra un algoritmo eficiente para un problema NP-completo, automáticamente se podrían resolver todos los problemas en NP de manera eficiente [28] [11].

Clase NP-Difícil

La clase NP-Difícil (NP-Hard) contiene problemas que, en términos de dificultad, son al menos tan difíciles como los problemas NP-completos. Sin embargo, no necesariamente pertenecen a NP, ya que algunos problemas NP-Difíciles pueden no ser verificables en tiempo polinomial. En otras palabras, un problema NP-Difícil es aquel al que cualquier problema NP se puede reducir en tiempo polinomial, pero no todos los problemas NP-Difíciles son NP-completos.

Un ejemplo clásico de un problema NP-Difícil es el problema del viajante (TSP, Traveling Salesman Problem) en su versión de optimización, donde se busca el recorrido de menor costo que pase por un conjunto de ciudades. Aunque su versión de decisión (*¿existe un camino con costo menor o igual a un valor dado?*) es NP-completa, la versión de optimización no necesariamente pertenece a NP, ya que encontrar una solución óptima no implica que pueda verificarse en tiempo polinomial [28].

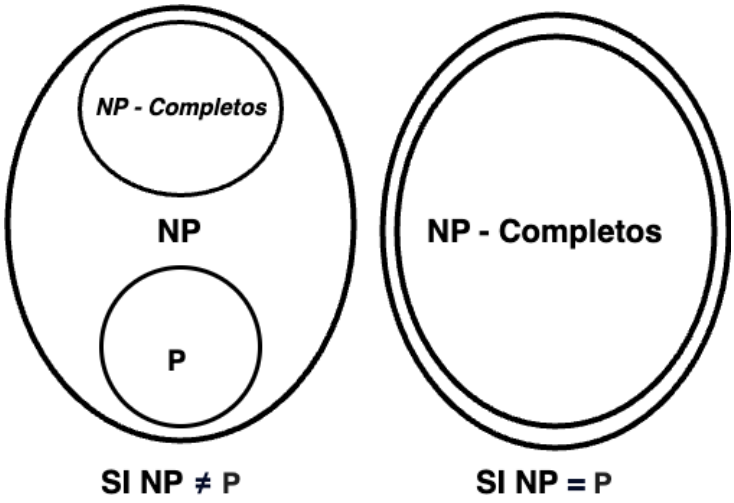

Figura 2.1: Posibles relaciones entre P, NP, NP-Completos y NP-Difíciles[38].

La relación entre estas clases se representa en la Figura 2.1, donde:

- P está completamente contenido dentro de NP.
- NP-Completos es un subconjunto de NP, donde todos los problemas son igualmente difíciles dentro de la clase NP.
- NP-Difíciles es una clase más amplia, que incluye tanto problemas NP-completos como problemas que pueden ser aún más difíciles y no necesariamente verificables en tiempo polinomial.

2.2. Estrategias de Resolución para Problemas de Optimización (NP-Difíciles)

La búsqueda y la exploración estocástica son técnicas importantes en la optimización y la inteligencia artificial, utilizadas para encontrar las mejores soluciones en problemas complejos donde los modelos estocásticos están presentes y cuyas soluciones no son factibles en tiempos polinómicos. Un ejemplo de estos es el problema del agente viajero. Donde la variabilidad y la incertidumbre juegan un papel crucial[14].

Los problemas de optimización combinatoria se caracterizan por tener variables de tipo discreto, es decir, se busca la solución dentro de un conjunto finito o infinito numerable de objetos. El objetivo es evaluar la calidad de cada objeto dentro del conjunto. Estos problemas suelen resolver el problema computacional conocido como explosión combinatoria, donde el número de posibles soluciones crece debido a las diferentes combinaciones de elementos[14].

En los problemas de optimización, comúnmente se considera que la solución se reduce a buscar el máximo o mínimo dentro de un conjunto de elementos. Sin embargo, en los problemas de combinatoria, esta tarea se vuelve compleja debido a que la cantidad de soluciones posibles aumenta exponencialmente con el tamaño del problema. Esto significa que, aunque sea posible encontrar una solución, esta no resulta viable para problemas de gran tamaño, debido al tiempo excesivo que se requiere para obtener la mejor solución[14].

Por lo tanto, el enfoque en la resolución de problemas de optimización combinatoria se centra en desarrollar algoritmos que puedan resolver estos problemas en tiempo polinomial[14].

Dentro de estas técnicas, se encuentran varios métodos estocásticos, como los algoritmos genéticos, el recocido simulado y la búsqueda tabú[14] . Sin embargo, este trabajo se centrará en el análisis de redes utilizando algoritmos de inteligencia de enjambre.

Con el avance de la inteligencia artificial, se ha observado cómo distintas áreas de la IA simulan comportamientos naturales. Un claro ejemplo se encuentra en los grupos gregarios, como peces, abejas, lobos, y hormigas. A partir de estas observaciones, se desarrolló la inteligencia de enjambre, basada en el comportamiento colectivo de estas estructuras sociales, aplicándose en la resolución de diversos problemas [38].

Uno de los primeros trabajos en este campo fue realizado por Craig Reynolds, quien simuló el comportamiento de las parvadas de aves basándose en tres reglas simples. Estas reglas, aunque se aplicaban localmente por cada individuo, producían resultados globales [14]. Este enfoque sentó las bases para el desarrollo de la inteligencia de enjambre, demostrando cómo interacciones simples pueden generar patrones complejos y organizados en sistemas naturales y artificiales.

A partir de las bases que sentó Craig Reynolds se generaron algoritmos como

el de "optimización por enjambre de partículas", el cual tenía como objetivo simular como las aves de una parvada buscaban alimento (figura 2.2). La ubicación de cada ave puede representar una posible solución, mientras que la comida que encuentra cada ave simboliza la evaluación de una posible solución[38].

Dentro de este contexto, regularmente se utiliza una función que mide la diferencia entre la solución encontrada y el resultado ideal que buscamos. A esta diferencia se le conoce como error. La función que evalúa el error determina la calidad de las soluciones. Dependiendo de si buscamos maximizar un valor o minimizar el error, el proceso se convierte en una tarea de optimización[38]. De este modo, el algoritmo ajusta la posición de cada "ave" en el espacio de soluciones, buscando continuamente la mejor solución posible, hasta que se cumplan ciertas condiciones.

Figura 2.2: Vuelo sincronizado de los estorninos europeos[38].

Optimizacion por Enjambre de Particulas (Particle Swarm Optimization, PSO)

La optimización por enjambre de partículas, conocida en inglés como Particle Swarm Optimization (PSO), es una técnica heurística en la que una colección

de partículas o agentes que se mueven en un espacio de N dimensiones. Este algoritmo se basa en una población donde cada individuo (partícula) trabaja conjuntamente para resolver un problema, utilizando los principios de evaluación, comparación e imitación. Una de las ventajas de este algoritmo es que su implementación es sencilla y corta [14] [26].

Además de la simplicidad en su implementación, una de sus características principales, es la manera en que las partículas interactúan y cooperan en su búsqueda de soluciones. Este proceso se basa en la comunicación entre las partículas, donde cada una representa un posible estado en el espacio de soluciones[26].

En el PSO, cada partícula se desplaza dentro del espacio de soluciones en busca de una posición óptima. Los agentes se comunican entre sí, y aquellos con una mejor posición, determinada por una función objetivo, atraen a los demás hacia ellos [15]. Este proceso permite que el conjunto de partículas explore de manera eficiente el espacio de soluciones, aumentando las probabilidades de encontrar el óptimo global.

En 1982, Reynolds propuso que cada partícula sigue tres principios fundamentales [17]:

- **Separación**: Cada agente se trata de mover lejos de sus vecinos si está muy cerca.
- **Alineación**: Cada agente se dirige hacia la dirección promedio de sus vecinos.
- **Cohesión**: Cada agente trata de ir hacia la posición promedio de sus vecinos.

Figura 2.3: Principios de una partícula en PSO[17].

En 1995, Kennedy y Eberhart agregaron un enfoque basado en la idea de un roost o punto de atracción. En este modelo, cada agente individual es atraído hacia la ubicación del roost, lo que representa una meta o solución ideal. Además, cada agente tiene la capacidad de recordar la posición más cercana al roost que ha alcanzado durante su trayectoria. Los agentes también interactúan entre sí compartiendo información sobre sus posiciones más cercanas al roost, originalmente intercambiando esta información con todos los demás agentes del sistema[17].

PSO Global (gbest)

En la versión PSO global, cada partícula del enjambre se comunica con todas las demás. Esto significa que todas las partículas conocen la mejor posición global alcanzada por cualquier partícula en el enjambre. La velocidad de cada partícula se actualiza de acuerdo a la mejor posición que ha encontrado personalmente (componente cognitivo) y la mejor posición global del enjambre (componente social). La ecuación para la velocidad es la siguiente:

$$v_{ij}(t+1) = v_{ij}(t) + c_1 r_1(t)[y_{ij}(t) - x_{ij}(t)] + c_2 r_2(t)[\hat{y}_{ij}(t) - x_{ij}(t)] \quad (2.1)$$

Donde:

- $v_{ij}(t)$ es la velocidad de la partícula i en la dimensión j en el tiempo t.
- $x_{ij}(t)$ es la posición actual de la partícula.
- $y_i(t)$ es la mejor posición encontrada por la partícula i.
- c_1 y c_2 son coeficientes que controlan la influencia cognitiva y social respectivamente.
- r_1 y r_2 son números aleatorios en el rango $[0, 1]$.

La posición se actualiza sumando la nueva velocidad a la posición actual:

$$x_{ij}(t+1) = x_{ij}(t) + v_{ij}(t+1) \quad (2.2)$$

Pseudocódigo Básico de gbest

El enfoque *global best* (**gbest**) en Particle Swarm Optimization (PSO) utiliza la mejor solución encontrada por todo el enjambre como referencia para actualizar la velocidad de cada partícula. Este enfoque favorece la convergencia rápida, aunque puede conducir a *óptimos locales*. En la literatura, este esquema fue descrito por primera vez por Kennedy y Eberhart (1995) y ha sido ampliamente estudiado por su simplicidad y efectividad en problemas bien definidos.

Algoritmo 1 *PSO Global (gbest)*

Crear e inicializar un enjambre n_x-dimensional
repetir Condición de paro
    para cada partícula $i = 1$ hasta n_s **hacer**
        si $f(x_i) < f(y_i)$ **entonces**
            $y_i = x_i$ ▷ Mejor posición individual
        fin si
        si $f(y_i) < f(\hat{y})$ **entonces**
            $\hat{y} = y_i$ ▷ Mejor posición global del enjambre
        fin si
    fin para
    para cada partícula $i = 1$ hasta n_s **hacer**
        Actualizar la velocidad usando la ecuación: (2.1)
        Actualizar la posición usando la ecuación: (2.2)
    fin para
hasta que Se cumpla la condición de paro
Fin *PSO Global (gbest)*

Este enfoque es adecuado para problemas donde la información global es confiable y puede llevar a una solución óptima sin exploración excesiva.

PSO Local (lbest)

A diferencia del esquema *gbest*, el modelo *local best* (**lbest**) limita la influencia social a un vecindario de partículas. Esto aumenta la exploración del espacio de búsqueda y permite mantener la diversidad dentro del enjambre. En este

caso, la velocidad de cada partícula se actualiza en función de su mejor experiencia personal y la mejor experiencia dentro de su vecindario.

$$v_{ij}(t+1) = v_{ij}(t) + c_1 r_1(t)[y_{ij}(t) - x_{ij}(t)] + c_2 r_2(t)[\hat{y}_{ij}(t) - x_{ij}(t)] \quad (2.3)$$

Donde $\hat{y}_{ij}(t)$ representa la mejor posición encontrada en el vecindario de la partícula i. Este modelo es más robusto en problemas multimodales o cuando se desea mantener exploración prolongada del espacio de soluciones.

Pseudocódigo Básico de lbest

Algoritmo 2 *PSO Local (lbest)*

Crear e inicializar un enjambre n_x-dimensional
repetir Condición de paro
    para cada partícula $i = 1$ hasta n_s **hacer**
        si $f(x_i) < f(y_i)$ **entonces**
            $y_i = x_i$ ▷ Mejor posición individual
        fin si
        si $f(y_i) < f(\hat{y}_i)$ **entonces**
            $\hat{y}_i = y_i$ ▷ Mejor posición del vecindario
        fin si
    fin para
    para cada partícula $i = 1$ hasta n_s **hacer**
        Actualizar la velocidad usando la ecuación: (2.3)
        Actualizar la posición usando la ecuación: (2.2)
    fin para
hasta que Se cumpla la condición de paro
Fin *PSO Local (lbest)*

Este esquema está especialmente indicado para problemas complejos y con múltiples óptimos, donde se requiere evitar la convergencia prematura y promover la exploración continua del espacio de soluciones. Referencias clave sobre este modelo incluyen a Shi y Eberhart (1998), quienes propusieron mejoras en la dinámica de actualización del enjambre.

Diferencias entre gbest y lbest

En PSO global (gbest), las partículas convergen más rápidamente debido a la fuerte interconexión entre todas las partículas. Sin embargo, esta rápida convergencia puede llevar a una menor diversidad en las soluciones y un mayor riesgo de quedar atrapadas en mínimos locales. Mientras que en PSO local (lbest), las partículas tienen menos interconexión, lo que permite una mayor diversidad en las soluciones. Esto reduce la probabilidad de quedar atrapadas en mínimos locales, pero la convergencia puede ser más lenta.

2.3. Optimización Multiobjetivo

En muchos problemas de optimización, no basta con mejorar un único criterio; es necesario considerar simultáneamente múltiples objetivos en conflicto, como minimizar costos mientras se maximiza la eficiencia o reducir el consumo energético sin afectar el rendimiento. Estos problemas, conocidos como problemas de optimización multiobjetivo, plantean desafíos adicionales debido a que es imposible reducir todos los criterios a una única métrica común [36].

Dado que los métodos tradicionales de optimización no siempre son aplicables a este tipo de problemas, han surgido enfoques alternativos que permiten encontrar soluciones eficientes. Entre estos enfoques destacan las metaheurísticas, que han demostrado ser herramientas eficaces para la búsqueda de soluciones en espacios complejos. Para abordar el conflicto entre los objetivos, estas técnicas han incorporado mecanismos de preservación de diversidad y el uso de indicadores especializados, permitiendo obtener un conjunto de soluciones que no pueden mejorarse en un criterio sin empeorar otro, formando lo que se conoce como frente de pareto [13].

La implementación de estos métodos se ha visto facilitada por el desarrollo de frameworks y herramientas especializadas, que no solo ofrecen una colección de algoritmos avanzados, sino también plantillas de código y funciones de análisis para evaluar los resultados obtenidos. Gracias a su diseño modular, estas herramientas permiten la integración de restricciones específicas del problema y la configuración personalizada de los algoritmos, lo que ha impulsado su uso en diversas áreas [36]

2.4. Redes Complejas

Las redes complejas son estructuras fundamentales en varias áreas de conocimiento, desde sistemas naturales y artificiales, redes sociales hasta circuitos eléctricos. Estas estructuras representan componentes de un sistema y las relaciones entre ellos, y contienen propiedades topológicas, tales como la distribución de conexiones, el coeficiente de agrupamiento y la existencia de caminos cortos entre nodos. Además, se encuentran características y estadísticas, como patrones emergentes y comportamientos colectivos, los cuales no se pueden predecir en modelos de redes normales [7].

Para entender de una mejor manera estas redes, se necesita empezar con la teoría de gráficas, la cual es un área de las matemáticas que estudia a los diferentes entes como nodos y a sus relaciones o enlaces conocidos como aristas.

2.4.1. Introducción a la Teoría de Gráficas

La teoría de gráficas, la cual es una base de las redes complejas, se creó a partir de un problema muy conocido, se trata de los puentes de Königsberg. Este problema se origina en el siglo XVIII, con el interés de Leonhard Euler por el enigmático problema de los puentes de Königsberg, el cual consistía en demostrar si se podía recorrer 7 puentes divididos en 4 zonas, la restricción más importante de este problema, es que se debía de encontrar una ruta que permitiera recorrer los 7 puentes sin cruzar dos veces el mismo [20][16].

Euler demostró que no es posible, debido a que se dio cuenta que no es un ciclo euleriano, sino un camino euleriano, ya que el punto de partida es distinto al de terminación[16]. Sin embargo, la parte importante de este problema fue la introducción de conceptos como el de grafo, donde los puentes y los caminos se representaban como aristas y vértices. Este enfoque permitió representar cualquier estructura compleja como un grafo.

Figura 2.4: Puentes de Königsberg [10].

2.4.2. ¿Qué es un Grafo?

Un grafo es una estructura utilizada para representar relaciones entre distintos elementos. Está compuesto por dos componentes principales: los vértices o nodos, que representan los elementos o entidades del grafo, y las aristas o enlaces, que simbolizan las conexiones entre los vértices. En el contexto de una red social, los usuarios serían los vértices, mientras que las aristas representarían las relaciones de amistad entre los usuarios[32].

Cuando hablamos de grafos, podemos clasificarlos principalmente en tres tipos: Grafos no dirigidos (también llamados no orientados), grafos dirigidos (u orientados) y grafos ponderados. [32][27].

Grafos no Dirigidos (no orientado)

En un grafo no dirigido, los lados no tienen una dirección. Esto significa que si hay un lado entre dos vértices A y B, se puede ir de A a B y de B a A sin importar el orden. Los lados de este tipo de grafo se representan como pares de vértices, como por ejemplo { A, B }[32].

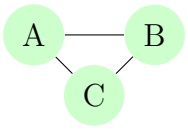

Figura 2.5: Grafo no dirigido

Grafos Dirigidos (u orientados)

En un grafo orientado, los lados tienen una dirección. Es decir, si hay un lado que va de A a B, solo se puede ir en esa dirección. En este caso los lados se representan como pares ordenados, como (A, B), donde A es el punto de inicio (extremo izquierdo) y B es el punto final (extremo derecho)[32].

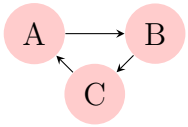

Figura 2.6: Grafo dirigido

Grafo Ponderado

En este tipo de grafos cada conexión entre los nodos tienen un valor numérico, conocido como peso o coste. Estos pesos suelen representar costos, distancias, o cualquier otra magnitud relacionada con cierto problema que se este analizando. Estos son útiles para resolver problemas de optimización como el camino más corto, donde los valores de los pesos influyen en la solución[27].

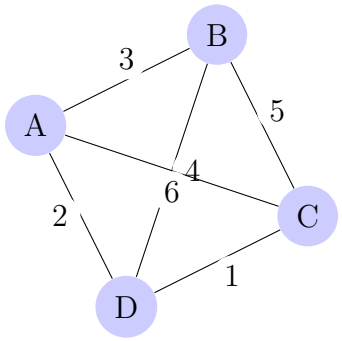

Figura 2.7: Grafo ponderado

2.4.3. Características de las Redes Complejas

A diferencia de una estructura de grafos simple, que pueden entenderse con reglas sencillas, las redes complejas se caracterizan por la emergencia, la autoorganización y la autoadaptación. Por lo que, no solo se estudian por su

estructura topológica, sino también por las dinámicas que estas estructuras permiten[34].

- **Emergencia:** Las redes complejas tienen propiedades emergentes, es decir que, tienen comportamientos que son impredecibles.
- **Autoadaptación:** Las redes complejas se pueden adaptar dinámicamente ante distintos cambios en su entorno, para que aunque se restructure las conexiones se mantengan sus funcionalidades.
- **Autoorganización:** Este tipo de redes se organizan por sí mismas, sin necesidad de tener un ente líder del sistema pueden funcionar.

2.4.4. Modelos de Redes

Para comprender las diversas estructuras y comportamientos de la redes complejas, los investigadores han desarrollado distintos modelos que permiten identificar a una red de acuerdo a sus características, según lo que se extrae de su estructura completa de la red compleja. Estos modelos son útiles para simular su comportamiento y analizar cómo responde ante distintos cambios que presentan.

Redes Aleatorias

Este modelo, tiene como característica principal que sus conexiones entre los nodos se generan de forma aleatoria. Esto quiere decir que, no tiene una estructura predefinida, sino que se generan a partir de reglas de probabilidad [34].

Se tienen dos modelos para las redes aleatorias[34]:

- G_n, m**:** En este se tienen un número fijo de nodos y de conexiones, dentro de este se genera una cantidad de aristas entre pares de nodos seleccionados aleatoriamente.
- G_n, p**:** En este se tiene un número establecido de nodos, aunque el número de conexiones se genera entre pares con cierta probabilidad de estar conectados por una arista, cabe mencionar que cada conexión se determina de manera independiente para cada par de nodos.

Ambos modelos están presentes en la investigación , principalmente cuando se tiene un "componente gigante", el cual es una gran área de la red que se encuentra conecta entre sí. Con esto se investiga los casos en los que el número de conexiones o la probabilidad de conexión, alcanza un umbral de interés para el investigador.

Redes Regulares

Esta red se caracteriza por presentar estructuras en donde cada nodo tiene el mismo número de conexiones, es decir que tienen un alto coeficiente de agrupamiento y una longitud promedio alta [34].

Redes de Mundo Pequeño

En estas redes, los nodos están conectados a través de un número relativamente pequeño de pasos, este modelo fue introducido por Watts y Strogatz en 1998 para modelar redes sociales [39]. En este sus principales características radican en que es posible llegar de un nodo a otro a través de una camino de conexiones cortas, aunque los nodos no están conectados directamente. Al igual que tienen un alto nivel de agrupamiento, ya que se podría plantear el caso, en el que si dos nodos están conectados a un mismo tercer nodo, es probable que también estén conectados entre sí. Esto nos hace pensar en el caso de una red social, donde es común que amigos de un amigo también sean amigos entre sí [34].

Redes de Libre escala

Estas redes se caracterizan por seguir una distribución de grados a partir de la ley de potencia. Esto quiere decir que, una gran cantidad de nodos tiene pocas conexiones, mientras que existen nodos que tienen un número elevado de conexiones, a esto se les conoce como "hubs". Una característica importante de estas redes es que no importa a qué nivel de escala se analicen, ya que, si se examina un grupo pequeño de nodos, la distribución y la estructura mantendrán el seguimiento de la ley de potencia, de ahí es donde proviene su nombre [2] [7] [18].

En la figura 2.8 se muestran las cuatro modelos de redes presentadas anteriormente.

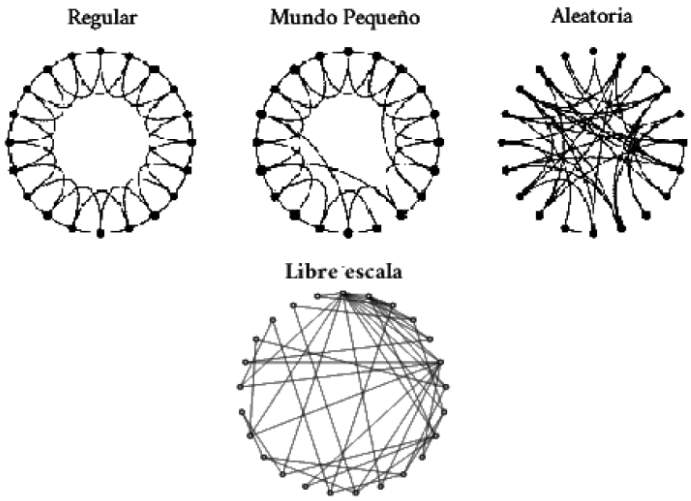

Figura 2.8: Modelos de las redes complejas[39].

2.4.5. Métricas Estructurales

En el análisis de redes complejas, las métricas estructurales nos permiten describir y cuantificar las características de la red. Estas en su mayoría se derivan de la teoría de gráficas, que proporciona una serie de bases para entender la relaciones entre los nodos y los enlaces de un sistema [34].

- **Grado:** Es la propiedad más básica de un vértice, este se define generalmente por el número de conexiones que tiene un nodo. Es decir que, son todas las aristas que se tienen para un vértice.
- **Grado medio:** Se obtiene a partir de todos los grados de nodos, ya que es el promedio de estos.
- **Distribución de grado:** La distribución de grado se obtiene por la función de distribución $P(k)$, que es la probabilidad de que un nodo seleccionado al azar tenga exactamente k enlaces o conexiones.
- **Distancia:** Es definida como el número de enlaces del camino más corto que conectan a dos nodos.

- **Diámetro de la red:** Regularmente se define con una letra D, la cual es la máxima distancia entre todos los caminos más cortos entre un par de nodos.

2.5. Calidad en el Servicio en Redes WiFi

Esta seccion explora los principios de la Calidad de Servicio en redes Wi-Fi, abordando los fundamentos de Wi-Fi 6, la importancia del QoS y las métricas clave para evaluar el rendimiento de la red.

2.5.1. Protocolo WIFI 6

Wi-Fi 6 es también conocido como 802.11ax, es un estándar inalámbrico diseñado para proporcionar mayores velocidades, menor latencia y mayor eficiencia en la red en comparación con versiones anteriores como Wi-Fi 5[25]. Gracias a tecnologías como el Acceso Múltiple por División de Frecuencia Ortogonal Multiusuario (MU OFDMA) y las Múltiples Entradas y Múltiples Salidas Multiusuario (MU MIMO), Wi-Fi 6 permite a varios dispositivos compartir el mismo canal simultáneamente, lo que reduce la latencia y mejora el rendimiento en redes. Además, incluye nuevos mecanismos para la reutilización espacial (SR) que permiten a más dispositivos operar en el mismo espacio físico sin interferencias significativas. Por otro lado, la eficiencia energética se ve incrementada a través de características como el Target Wake Time (TWT), que optimiza el uso de la batería en los dispositivos conectados.[33].

2.5.2. ¿Qué es QoS?

En las distintas aplicaciones que se conectan a la red, se pueden generar diferentes tipos de tráfico, y congestión en la red, por lo que, es necesario garantizar su adecuado funcionamiento tomando diferentes medidas. Por ejemplo, se genera bastante latencia cuando se maneja el tráfico de voz, ya que si se retrasan los paquetes se suele escuchar voces entrecortadas. Por otro lado, si en esa misma red se utiliza FTP, el cual se utiliza para descargas de

archivos, es menos propenso a estos retrasos, por lo que se prioriza el tráfico de voz, para que ambos servicios funcionen de manera efectiva y con buena calidad [31].

El concepto de Calidad de Servicio (QoS) se refiere a la implementación de técnicas y procedimientos que permiten darle preferencia a ciertos tipos de tráfico contra otros, para mejorar la calidad de todos los servicios que se lleven a cabo en esa red[31], su significado más general incluye cualquier aspecto que influya en la satisfacción del usuario al utilizar un servicio. Según la Unión Internacional de Telecomunicaciones (UIT-T), QoS se define como "el efecto global de las prestaciones de un servicio que determinan el grado de satisfacción de un usuario". Además, el Grupo de Trabajo de Ingeniería de Internet (IETF) complementa esta definición al describir QoS como el "conjunto de requisitos del servicio que debe cumplir la red en el transporte de un flujo" [12].

2.5.3. Métricas para Medir la QoS

Las métricas de Calidad de Servicio (QoS) son parámetros cuantificables que permiten evaluar el rendimiento de la red y la calidad del servicio ofrecido. El QoS se enfoca en optimizar cuatro aspectos fundamentales de la red: ancho de banda, latencia, jitter y pérdida de paquetes[31].

1. **Latencia:** Es el tiempo que tarda un paquete en llegar a su destino desde que se envía. Factores como el tiempo de procesamiento en routers, la longitud del cable y la conversión de señales afectan la latencia.

2. **Jitter:** Es la variación en la latencia de los paquetes. En aplicaciones como la telefonía IP, los paquetes deben llegar en intervalos constantes. Si estos intervalos varían, se producen cortes en la voz.

3. **Perdida de paquetes:** Ocurre cuando los paquetes no llegan a su destino o caducan. Puede ser causada por colas de espera demasiado largas o sobrecarga de la red.

4. **Ancho de banda:** Es la cantidad de bits que pueden transmitirse por un enlace en un segundo. En el contexto de QoS, el Commited Information Rate (CIR) es el ancho de banda mínimo garantizado por

el proveedor. Si la red tiene poca carga, es probable que permita exceder ese límite, pero si es mayor reducirá el ancho de banda disponible.

5. **Cobertura:** Es el área geográfica en la que una red puede proporcionar conectividad efectiva. En redes cableadas, depende de la infraestructura física y la distancia entre dispositivos, mientras que en redes inalámbricas es influenciada por la potencia de transmisión, la frecuencia utilizada y la presencia de obstáculos que puedan interferir con la señal.

Capítulo 3

Estado del arte

El Estado del Arte es una revisión detallada de los avances científicos y tecnológicos en un área de estudio específica. Su propósito es identificar los enfoques, metodologías y soluciones que han sido explorados previamente, permitiendo contextualizar la investigación actual. En este capítulo, se presenta un análisis detallado de los estudios más relevantes sobre la optimización de la Calidad de Servicio (QoS) en redes de telecomunicaciones utilizando Algoritmos de Enjambre de Partículas (PSO).

Para estructurar esta revisión, se han identificado y analizado investigaciones desde tres perspectivas principales: el desarrollo del algoritmo PSO, su aplicación en la optimización de QoS en redes de telecomunicaciones y la optimización en las redes de telecomunicaciones mediante heurísticas. A través de esta revisión, se busca comprender cómo PSO ha sido utilizado en este campo y qué avances ha permitido, estableciendo así las bases para esta investigación.

3.1. Metodología de la Investigación

Para llevar a cabo esta revisión del estado del arte, se realizó una búsqueda sistemática de literatura en bases de datos académicas reconocidas y repositorios especializados. Se seleccionaron artículos publicados en revistas, conferencias científicas y documentos técnicos que abordan la optimización

de la Calidad de Servicio (QoS) en redes de telecomunicaciones mediante Algoritmos de Enjambre de Partículas (PSO).

El proceso de selección de los estudios relevantes se estructuró en las siguiente manera:

1. **Definición de criterios de búsqueda:** Se establecieron términos clave relacionados con la temática de estudio, tales como Particle Swarm Optimization (PSO), Quality of Service (QoS) in telecommunications, swarm intelligence, network optimization, entre otros.

2. **Consulta en fuentes especializadas:** Se accedió a bases de datos científicas de amplio reconocimiento, tales como IEEE Xplore, SpringerLink, ScienceDirect y Google Scholar, entre otras.

3. **Selección de documentos:** Se analizaron los resúmenes y conclusiones de los artículos encontrados para identificar aquellos con un enfoque directo en la optimización de QoS mediante PSO.

4. **Análisis y categorización de la información:** Los estudios seleccionados fueron organizados en tres categorías principales: Desarrollo del algoritmo PSO, optimización en redes de telecomunicaciones con heurísticas y aplicaciones de PSO en la optimización de QoS en redes.

3.2. Desarrollo del Algoritmo PSO

El algoritmo de Optimización por Enjambre de Partículas (PSO) es un paradigma de Inteligencia de Enjambre inspirado en el comportamiento colectivo de enjambres sociales en la naturaleza [1]. Desarrollado a mediados de la década de 1990 [1], PSO es conocido por su simplicidad, flexibilidad y alta eficiencia en la resolución de problemas de optimización global [40]. A diferencia de los enfoques deterministas, PSO introduce aleatoriedad en el proceso de búsqueda, lo que le permite evitar quedar atrapado en óptimos locales en situaciones complejas [40].

Desde su origen, PSO ha experimentado numerosas modificaciones y extensiones. Los investigadores han desarrollado nuevas aplicaciones, derivado nue-

vas versiones y publicado estudios teóricos sobre la influencia de diversos parámetros y aspectos del algoritmo [1].

3.2.1. Variantes y Mejoras del PSO

A lo largo de los años, se han desarrollado diversas variantes e híbridos de PSO para mejorar su rendimiento en diferentes tipos de problemas. Estos incluyen modificaciones en la **inicialización del enjambre**, **operadores de mutación** y el **peso de inercia**[24], así como combinaciones con otros algoritmos como **Algoritmos Genéticos** (GA), **Optimización por Colonia de Hormigas** (ACO) y otros, buscando explotar las ventajas de cada técnica y mitigar sus limitaciones [1] [41].

La hibridación es un modelo genérico de dos o más algoritmos que explota sus ventajas mientras disminuye sus impedimentos, mejorando la explotación y exploración del espacio de búsqueda [1][8]. En el contexto de la optimización de redes, las variantes de PSO que abordan problemas de optimización discreta y combinatoria [6] y aquellas que incorporan mecanismos para mejorar la convergencia y evitar el estancamiento [5] son particularmente relevantes. También se han explorado topologías de población alternativas (como anillo, aleatoria, von Neumann) para influir en la diversidad y la velocidad de convergencia del algoritmo [41].

3.3. Optimización en Redes de Telecomunicaciones Mediante Heurísticas

En el ámbito de las telecomunicaciones, la optimización juega un papel fundamental para asegurar la eficiencia, la robustez y la sostenibilidad de las redes. Los problemas de diseño y gestión en este sector a menudo son computacionalmente complejos, lo que ha impulsado el desarrollo y la aplicación de algoritmos heurísticos para encontrar soluciones de alta calidad en tiempos razonables.

3.3.1. Robustez y Conectividad de Redes

Un área importante de optimización se centra en la **robustez de las redes**, donde el objetivo es garantizar la continuidad de la comunicación frente a fallos en los enlaces. El problema del aumento de 2-conectividad por aristas (2-edge-connectivity augmentation) busca añadir un conjunto de aristas de mínimo coste a una red existente para hacerla **robusta contra fallos de un único enlace**[4]. En este contexto, se han investigado diversas heurísticas, incluyendo métodos constructivos y metaheurísticas, que exploran diferentes estructuras de vecindad para la búsqueda local [4].

Algunas métricas clave aquí son el **coste de las aristas** añadidas y la **garantía de la conectividad** tras la adición. Se han comparado experimentalmente diferentes algoritmos, incluyendo métodos exactos y heurísticas, para resolver instancias generales y a gran escala, explotando enfoques como la **cobertura de conjuntos y vecindades de gran escala** [4]. Una aplicación práctica de este problema es la extensión de redes de telecomunicaciones existentes para aumentar su resistencia contra fallos de un solo enlace [4].

3.3.2. Distribución de Contenido en Streaming

Otro campo relevante es la optimización de las redes de **distribución de contenido en streaming (CDNs)** [9]. El diseño eficiente de estas redes implica decisiones cruciales sobre el número y la ubicación de los servidores réplica para equilibrar el **ancho de banda** de la red y los **costes de los servidores** [9]. La complejidad radica en la interrelación con los protocolos de entrega de media en streaming, las decisiones de redirección de clientes y el problema de enrutamiento de contenido [9].

Las heurísticas desarrolladas para este problema abordan la **ubicación de servidores**, **la selección de servidores** y el **enrutamiento**, considerando protocolos como la **difusión periódica**(periodic broadcast) y la **fusión jerárquica**(hierarchical merging) [9].Las métricas importantes incluyen el **ancho de banda consumido**, la **latencia de inicio**(start-up delay) y el **coste total de la red**, que considera tanto los costes de los servidores como los costes de la red [9]. Se busca optimizar estas métricas mediante la aplicación de heurísticas que ofrecen aproximaciones rápidas y precisas [9]. El

objetivo principal de las CDNs es proporcionar el mejor rendimiento disponible a sus clientes, desplegando la menor cantidad posible de réplicas [9].

3.3.3. Optimización Energética en WLANs

La optimización del **consumo de energía en redes de acceso inalámbrico** (WLANs) es también un área crítica [29]. El creciente consumo energético del sector de las Tecnologías de la Información y la Comunicación (TIC) ha impulsado la necesidad de enfoques de gestión de red más eficientes energéticamente [29]. Se han desarrollado algoritmos heurísticos basados en una combinación de **enfoques voraces** (greedy) y **métodos de búsqueda local** para adaptar el consumo de energía de la red a la demanda diaria, asegurando la **cobertura** y la **capacidad** requerida por los usuarios activos [29].

Las métricas clave en este contexto son el **consumo de energía** (en vatios o kWh/mes) y el **coeficiente de ahorro energético**, comparado con el consumo de redes sin mecanismos de optimización [29]. La evaluación de estas heurísticas se realiza comparando sus resultados con los obtenidos por modelos de programación lineal entera (ILP), analizando el compromiso entre la calidad de la solución y el tiempo de computación [29]. Estos algoritmos buscan explotar los recursos mínimos de la red en cada momento, garantizando a los usuarios activos un nivel de calidad de servicio satisfactorio [29].

3.4. Optimización de Redes: QoS

La optimización de redes mejora el rendimiento y la eficiencia en la transmisión de datos. Un aspecto clave es la Calidad de Servicio (QoS). Esta sección explora investigaciones sobre la optimización de la QoS desde diferentes perspectivas.

3.4.1. Optimización de la Calidad de Experiencia (QoE) en Redes Celulares LTE

Uno de los enfoques principales se centra en la optimización de la calidad de experiencia (QoE) en redes celulares LTE mediante el ajuste de parámetros de traspaso entre celdas [3]. Este trabajo de tesis doctoral investiga cómo modificar los márgenes de traspaso (HOM) y los temporizadores (TTT) para mejorar la QoE de los usuarios. Se proponen algoritmos para el reparto de tráfico que ajustan los márgenes de traspaso de forma heurística para equilibrar la QoE entre celdas vecinas y para maximizar la QoE global del sistema con criterios de optimalidad. Además, se desarrollan algoritmos de optimización del traspaso por movilidad que consideran explícitamente la QoE de los usuarios de borde de celda.

Para lograr esto, se emplean técnicas de inteligencia artificial. Por un lado, se utilizan controladores de lógica difusa para implementar reglas de ajuste de los márgenes de traspaso, aprovechando la experiencia de los operadores de red [3]. Estos controladores toman decisiones sobre si aumentar o no el margen de traspaso para cada adyacencia particular [3]. Por otro lado, para la optimización del traspaso por movilidad, se emplean técnicas de aprendizaje por refuerzo, implementadas con redes neuronales, buscando la configuración óptima de los parámetros de traspaso no solo para la robustez del enlace radio, sino también para maximizar la QoE de los usuarios antes y después del traspaso [3].

La motivación principal radica en la complejidad de la gestión de redes celulares debido a la diversidad de servicios y la necesidad de optimizar la percepción del servicio por parte del usuario final, considerando que diferentes servicios pueden tener comportamientos completamente distintos con relación a la QoE bajo idénticas condiciones de red [3].

3.4.2. Optimización de Redes de Comunicación por Cable

Otro enfoque se presenta en el contexto de la optimización de redes de comunicación por cable, donde se utiliza la optimización multiobjetivo para el diseño de la red [19]. El objetivo principal es determinar la topología radial

óptima minimizando el costo de inversión y optimizando el nivel de señal en los nodos de conexión de los usuarios [19]. El modelo matemático planteado incluye restricciones sobre el número máximo de divisores y acopladores permitidos por nodo, así como la necesidad de que la configuración sea radial, sin lazos ni nodos desconectados [19].

Para resolver este problema multiobjetivo, se emplea un algoritmo genético especializado basado en NSGA-II (Non-Dominated Sorting Genetic Algorithm II), buscando obtener un conjunto de soluciones Pareto-óptimas que ofrezcan al diseñador información sobre las configuraciones de mínimo costo que cumplen con los requerimientos de señal [19]. La métrica de .ªspacio cubierto"se utiliza para evaluar la distribución de las soluciones obtenidas en el frente de Pareto [19].

3.4.3. Confiabilidad en Redes de Telecomunicaciones

También, se aborda la confiabilidad en redes de telecomunicaciones a través de la simulación de Monte Carlo [23]. Este trabajo se enfoca en evaluar la capacidad de una red para mantener su operatividad ante posibles fallos en los enlaces [23]. Se presenta una herramienta computacional que permite a los ingenieros analizar la robustez de una red considerando aspectos como la capacidad de los enlaces, las tasas de falla y reparación, y el enrutamiento del tráfico [23]. Si bien no se centra directamente en la optimización de la QoS en tiempo real, la confiabilidad es un factor importante que influye en la calidad del servicio ofrecido a largo plazo.

Se exploran algoritmos de enrutamiento como el de Ford-Fulkerson para maximizar el flujo de datos en la red, buscando la mínima pérdida de capacidad del sistema en caso de contingencias [23]. El simulador desarrollado permite modelar la red con un número limitado de enlaces y evaluar la demanda y el tráfico, ofreciendo una visión de la robustez del sistema ante fallas [23].

Capítulo 4

Materiales y Métodos

En el analisis de redes modernas, es necesario entender las métricas de Calidad de Servicio (QoS). Estas métricas permiten medir aspectos importantes del rendimiento de la red, asegurando una buena calidad de experiencia a los usuarios.

Asimismo, es necesario considerar las tecnologías que se utilizaran en la red a analizar. En este proyecto, se tomará como referencia el protocolo Wi-Fi 6. Además, se utilizarán Access Point compatibles con esta tecnología, cuya importancia radica en su influencia sobre la cobertura, la calidad de la señal, y otros factores.

Este capítulo se dedicará a explorar en profundidad estas métricas y tecnologías, proporcionando una mejor comprensión del tema.

4.1. Materiales

El correcto diseño e implementación de una red Wi-Fi 6 requiere dispositivos con características técnicas avanzadas que permitan optimizar el rendimiento y garantizar una calidad de servicio (QoS) óptima. En este estudio, se han utilizado Access Points (APs) UAP-AC-LR WiFi 6 de UniFi Ubiquiti, los cuales ofrecen un equilibrio entre cobertura, estabilidad y gestión eficiente del tráfico de red en entornos de alta densidad.

La selección de estos APs se realizó considerando su compatibilidad con el estándar Wi-Fi 6, su capacidad para manejar múltiples usuarios simultáneos y su eficiencia en la administración del ancho de banda. A continuación, se presentan las características clave de estos dispositivos y su impacto en la calidad de la red.

4.1.1. Características de los Access Points

Los valores máximos sugeridos en la tabla para un desempeño óptimo en redes Wi-Fi 6 están influenciados por diversas características técnicas de los Access Points, como la potencia de transmisión, la ganancia de antena, la capacidad de MIMO (Multiple-Input Multiple-Output) y el ancho de banda soportado.

1. **Latencia y Jitter:** La latencia y el jitter dependen de la capacidad del Access Point para gestionar múltiples conexiones y optimizar la transmisión de datos. Tecnologías como MU-MIMO y OFDMA permiten dividir los recursos de la red de manera eficiente, reduciendo la espera en la transmisión de paquetes y minimizando variaciones en el tiempo de entrega de los mismos. Además, el mayor ancho de banda disponible en Wi-Fi 6 facilita el manejo de múltiples flujos simultáneos, contribuyendo a una comunicación más estable y con menor retardo.

2. **Paquetes Perdidos:** La pérdida de paquetes se ve afectada por la potencia de transmisión y la eficiencia en la gestión del tráfico. Un Access Point UAP-AC-LR WiFi 6 de UniFi Ubiquiti con una mayor potencia de TX puede mejorar la estabilidad de la conexión al asegurar que las señales alcancen los dispositivos de manera más confiable. Además, la interferencia con otras redes y la saturación del espectro también pueden influir en la pérdida de paquetes, por lo que estos dispositivos incluyen mecanismos avanzados de gestión de tráfico.

3. **Cobertura (dBm):** La cobertura está determinada por la potencia de transmisión y la ganancia de antena. Los UAP-AC-LR WiFi 6 cuentan con antenas de alta ganancia (3 dBi en 2.4 GHz y 6 dBi en 5 GHz), lo que permite extender el alcance de la señal y mejorar la penetración en entornos con obstáculos. Su capacidad de operar en doble banda

simultánea (2.4 GHz y 5 GHz) permite una mayor flexibilidad y estabilidad de la conexión en distintas condiciones.

En conjunto, estas características afectan el rendimiento de la red Wi-Fi 6, asegurando que los dispositivos conectados puedan operar dentro de los límites recomendados para una experiencia de usuario óptima.

Especificaciones Técnicas de los Access Points

Para este estudio, se utilizaron Access Points UAP-AC-LR WiFi 6 de UniFi Ubiquiti, los cuales cuentan con las siguientes especificaciones clave:

- Compatibilidad con Wi-Fi 6 (802.11ax), con soporte para tecnologías avanzadas como MU-MIMO, OFDMA y BSS Coloring.
- Capacidad de transmisión en doble banda simultánea (2.4 GHz y 5 GHz), con un ancho de banda total de hasta 1775 Mbps.
- Manejo optimizado del tráfico, mejorando la eficiencia en entornos de alta densidad mediante gestión de QoS avanzada.
- Tecnología Beamforming, que mejora la señal en función de la ubicación de los dispositivos conectados.
- Soporte para hasta 300 dispositivos simultáneamente, asegurando estabilidad en redes con múltiples conexiones.
- Mecanismos de ahorro de energía como Target Wake Time (TWT), que optimiza el consumo de batería en dispositivos móviles.
- Seguridad mejorada con soporte para WPA3, reduciendo vulnerabilidades frente a ataques de fuerza bruta.
- Gestión centralizada a través del software UniFi Controller, que permite monitoreo y configuración avanzada de la red.

Este estudio se llevó a cabo en las instalaciones de una bodega industrial ubicada en Naucalpan, Estado de México, donde se evaluó el desempeño de

la red Wi-Fi 6 y se implementaron estrategias de optimización de calidad de servicio (QoS).

La infraestructura de la red en esta empresa está conformada por múltiples Access Points UAP-AC-LR WiFi 6 de UniFi Ubiquiti, distribuidos estratégicamente para garantizar una cobertura eficiente y una conectividad estable en diferentes áreas de trabajo.

A continuación, se presenta el esquema de distribución de la red dentro de la bodega en Naucalpan, detallando la ubicación de los Access Points, sus parámetros de configuración y la manera en que se gestionó el tráfico de red para optimizar la latencia, el jitter y la pérdida de paquetes en función de las necesidades operativas de la empresa.

4.1.2. Distribución de Red del Caso de Prueba

Los datos utilizados en este estudio fueron proporcionados por una fábrica de ropa ubicada en Naucalpan, Estado de México y corresponden a mediciones realizadas en sus instalaciones. Dichos datos reflejan condiciones operativas reales y permiten evaluar el desempeño de la red Wi-Fi 6 implementada en ese entorno específico.

A continuación, se presenta la distribución de la red inalámbrica en las instalaciones.

Planta Baja

La siguiente imagen muestra el mapa de la planta baja, donde se encuentran las áreas de facturación, recepción y algunas oficinas. Dado que el espacio no es muy amplio, se han instalado **dos antenas WiFi U6 Lite** para cubrir la demanda de conectividad.

Las ubicaciones de estas antenas están marcadas en el mapa con **rombos de color amarillo**.

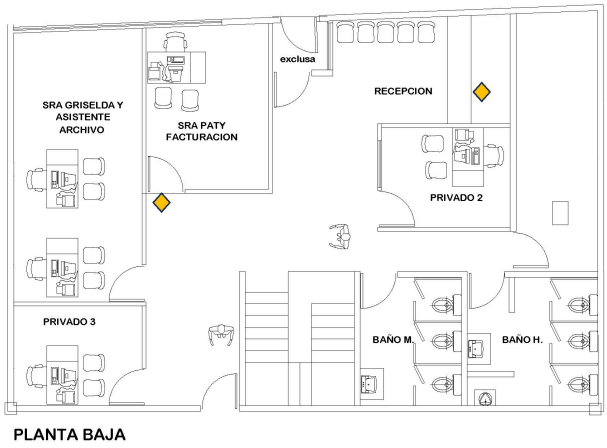

Figura 4.1: Mapa de la planta baja

Segunda Planta – Oficinas

En esta sección, que abarca diversas oficinas y salas de juntas, también se han instalado **dos antenas WiFi U6 Lite**. Dado que el área es relativamente pequeña en comparación con la bodega, la cobertura de estas antenas es suficiente para garantizar una conexión estable.

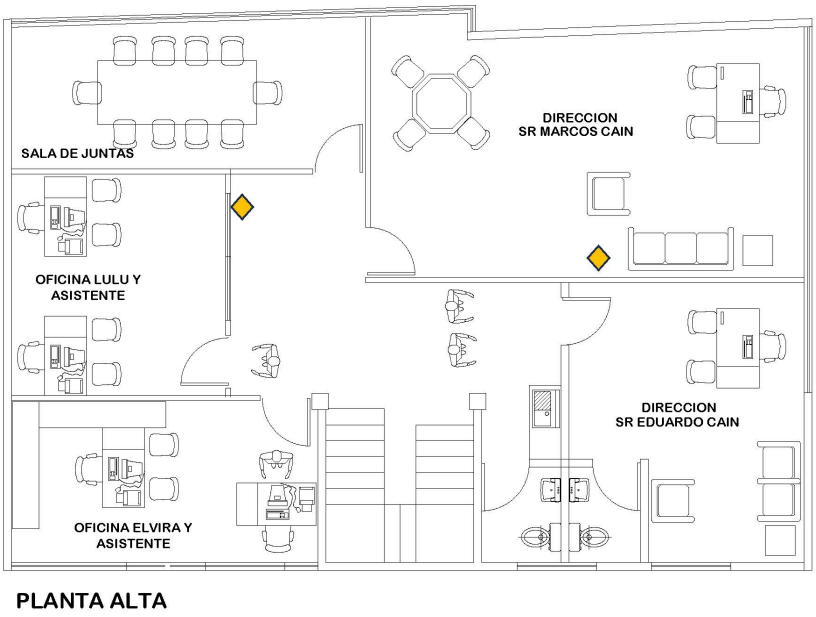

Figura 4.2: Mapa de la segunda planta

En total, hay **cuatro antenas U6 Lite** distribuidas en las oficinas. Las **otras cuatro antenas**, de tipo **U6 LR (Long Range)**, han sido ubicadas en la bodega debido a sus mayores capacidades de cobertura.

Bodega – Parte 1

La bodega, por su tamaño, requiere antenas con mayor alcance. En esta primera sección, se han instalado **dos antenas WiFi U6 LR**, ya que estas tienen una mayor potencia de transmisión de señal. No obstante, al abarcar un área más extensa, puede haber solapamiento de señal, lo que podría generar interferencias entre los canales.

Uno de los puntos de acceso (AP) está ubicado en la zona de **plotter**, mientras que el otro se encuentra en el área identificada como **"bodega salida"**.

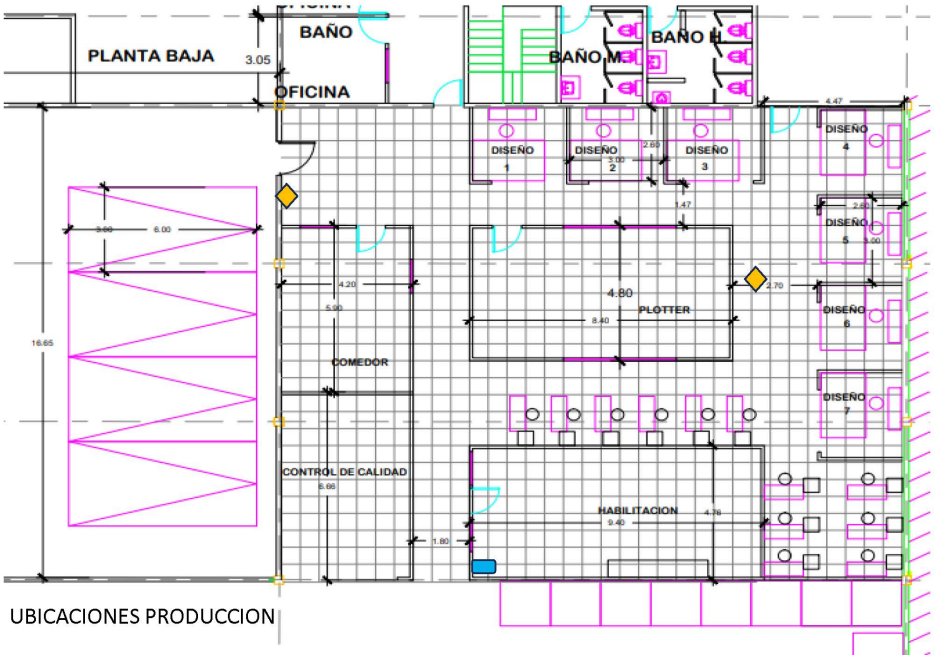

Figura 4.3: Primera parte de la bodega

Bodega – Parte 2

En la segunda sección de la bodega, se han instalado **dos antenas adicionales U6 LR**, necesarias para garantizar una cobertura completa del área.

Uno de los AP está identificado como **"bodega central"**, mientras que el otro se encuentra en **"bodega esquina"**. Con estas dos antenas, se completa la distribución de **ocho antenas WiFi** en la bodega.

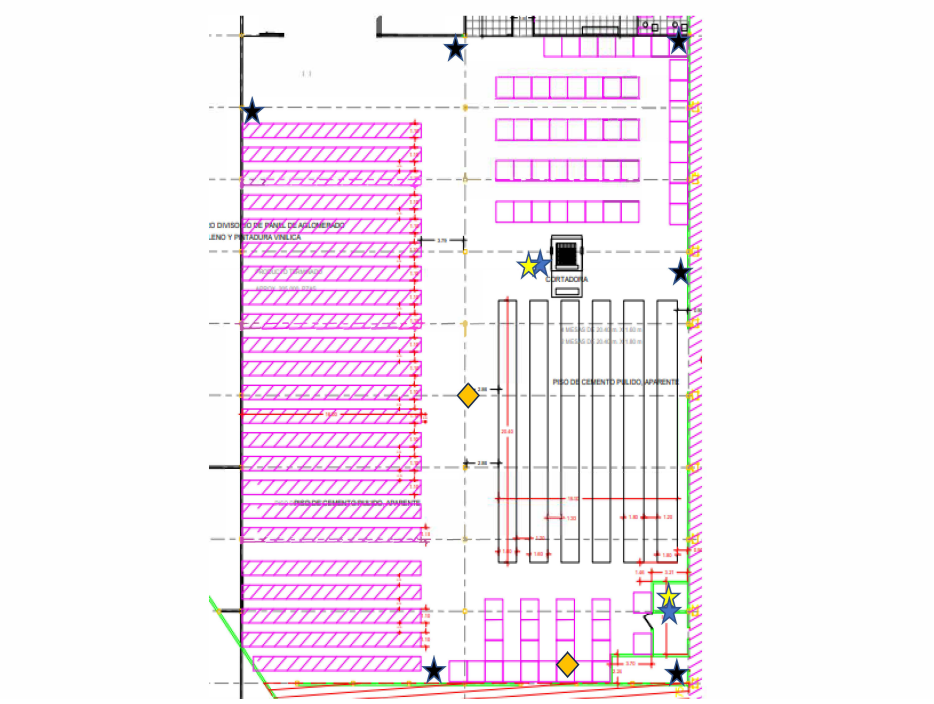

Figura 4.4: Segunda parte de la bodega

4.2. Métodos

En esta sección se describe el proceso para poder resolver el problema de optimización, basado en el análisis y la evaluación de la calidad en el servicio (QoS) para redes WiFi. Por lo tanto, a continuación se resumen los principales procedimientos a tratar y en las subsecciones siguientes, se presentan los pasos necesarios para poder lograrlo.

1. **Definición de métricas QoS:** Selección de las métricas más relevantes para la calidad de servicio, como latencia, jitter, pérdida de paquetes, y ancho de banda.

2. **Desarrollo de funciones objetivo:** Diseño de funciones matemáticas que modelen la optimización de las métricas seleccionadas, considerando diferentes condiciones de tráfico.

3. **Implementación del algoritmo PSO:** Desarrollo o adaptación del algoritmo de enjambre de partículas para optimizar los parámetros de QoS en las simulaciones.

4. **Simulación en redes:** Realización de simulaciones en redes de telecomunicaciones en diferentes condiciones (carga baja, media y alta) para evaluar el desempeño del algoritmo.

5. **Validación en redes reales:** Comparación de los resultados obtenidos en simulaciones con pruebas experimentales en entornos de red reales.

4.2.1. Modelo de Optimización

Para evaluar y optimizar la calidad de servicio (QoS) en redes Wi-Fi 6, se ha definido un modelo basado en múltiples objetivos que busca mejorar métricas clave como latencia, jitter, pérdida de paquetes y cobertura. A continuación, se presentan los datos de la instancia utilizada para la optimización, que incluyen mediciones de diferentes puntos de acceso (APs) dentro de la red.

Aunque el ancho de banda es un parámetro relevante en redes Wi-Fi 6, este estudio se enfoca exclusivamente en latencia, jitter, pérdida de paquetes y cobertura, ya que estos parámetros están directamente relacionados con la percepción del usuario final respecto a la calidad del servicio. Además, en el contexto específico evaluado, el ancho de banda disponible supera ampliamente la demanda de los usuarios y dispositivos, por lo que mejorar otros indicadores como latencia y estabilidad genera un mayor impacto en la calidad percibida.

Entonces, los valores obtenidos reflejan el comportamiento de la red en distintas ubicaciones, permitiendo identificar las áreas con deficiencias en el desempeño. Estos valores servirán como base para la optimización, asegurando que el modelo pueda ajustarse a las condiciones reales de la infraestructura de red.

A continuación, se presenta el modelo de optimización, que define las funciones objetivo y restricciones para garantizar la QoS.

Funciones Objetivo

1. Función de Latencia (L):

$$L = \sum_{i=1}^{N} L_i$$

Descripción: Minimizar la latencia es crucial para mejorar la rapidez y eficiencia de la red.

2. Función de Jitter (J):

$$J = \sum_{i=1}^{N} J_i$$

Descripción: Minimizar el jitter es esencial para garantizar la estabilidad de la conexión, especialmente en aplicaciones en tiempo real.

3. Función de Pérdida de Paquetes (P):

$$P = \sum_{i=1}^{N} (P_{Tx,i} + P_{Rx,i})$$

Descripción: Minimizar la pérdida de paquetes asegura una transmisión de datos confiable.

4. Función de Cobertura (C):

$$C = \sum_{i=1}^{N} C_i$$

Descripción: Maximizar la cobertura garantiza que todas las áreas críticas estén adecuadamente cubiertas.

Restricciones

1. Latencia Máxima Permitida:

$$L_i \leq L_{\max}, \quad \forall i$$

Descripción: Cada AP debe mantener su latencia dentro de un límite para garantizar tiempos de respuesta aceptables.

2. Jitter Máximo Permitido:

$$J_i \leq J_{\max}, \quad \forall i$$

Descripción: El jitter debe ser bajo para evitar fluctuaciones en la calidad de la conexión, especialmente en transmisiones en tiempo real.

3. Pérdida de Paquetes Máxima Permitida:

$$P_{Tx,i} \leq P_{\text{Tx,max}}, \quad P_{Rx,i} \leq P_{\text{Rx,max}}, \quad \forall i$$

Descripción: Limitar la pérdida de paquetes garantiza que la comunicación entre dispositivos sea confiable.

4. Cobertura Mínima Requerida:

$$\sum_{i=1}^{N} C_i \geq C_{\min}$$

Descripción: La red debe cubrir todas las áreas críticas con una señal de calidad mínima aceptable.

4.2.2. Modelo de Optimización con Múltiples Objetivos

El diseño eficiente de una red WiFi debe satisfacer las necesidades de cobertura, confiabilidad y eficiencia energética, asegurando además una Calidad de Servicio (QoS) adecuada. Este proyecto busca optimizar la ubicación y configuración de los puntos de acceso (APs) para maximizar el rendimiento general de la red mientras optimizando parámetros clave como latencia, jitter, pérdida de paquetes y cobertura.

En este modelo, buscamos maximizar un objetivo combinado que represente la calidad de servicio global (QoS). Las métricas originalmente de minimización se transforman en métricas de maximización utilizando funciones inversas.

Las funciones objetivo individuales se transforman de la siguiente manera:

$$L' = \frac{1}{L} \quad \text{(Latencia Inversa)}$$
$$J' = \frac{1}{J} \quad \text{(Jitter Inverso)}$$
$$P' = \frac{1}{P} \quad \text{(Pérdida de Paquetes Inversa)}$$
$$C = C \quad \text{(Cobertura, ya es una métrica de maximización)}$$

La función objetivo combinada que se desea maximizar se define como:

$$\text{Maximizar } F = w_1 L' + w_2 J' + w_3 P' + w_4 C \tag{4.1}$$

Sujeto a:

$$L \leq L_{\max}, \text{(Latencia máxima permitida)} \tag{4.2}$$

$$J \leq J_{\max}, \text{(Jitter máximo permitido)} \tag{4.3}$$

$$P \leq P_{\max}, \text{(Pérdida de paquetes máxima permitida)} \tag{4.4}$$

$$C \geq C_{\min}, \text{(Cobertura mínima requerida)} \tag{4.5}$$

Donde:

- L', J', P', C son las métricas transformadas.
- w_i son los pesos que determinan la importancia relativa de cada métrica.

Con base en el modelo descrito en las ecuaciones 4.1 a 4.5, a continuación se describen los datos de la instancia (datos reales obtenidos en la fábrica de ropa):

Parámetro	**Valor**
PB Site	
Latencia (ms)	144.38
Jitter (ms)	137.85
Paquetes Perdidos (Pmax %)	6.77
Cobertura (dBm)	-67
Marcos	
Latencia (ms)	121.24
Jitter (ms)	52.3
Paquetes Perdidos (Pmax %)	10.86
Cobertura (dBm)	-53
PA Sala Juntas	
Latencia (ms)	135.26
Jitter (ms)	60.97
Paquetes Perdidos (Pmax %)	4.4
Cobertura (dBm)	-56
PB Oficinas	
Latencia (ms)	190.4
Jitter (ms)	83.5
Paquetes Perdidos (Pmax %)	2.94
Cobertura (dBm)	-56
Bodega Central	
Latencia (ms)	118.36
Jitter (ms)	69.64
Paquetes Perdidos (Pmax %)	0.85
Cobertura (dBm)	-73
Bodega Plotter	
Latencia (ms)	143.79
Jitter (ms)	79.68
Paquetes Perdidos (Pmax %)	1.96
Cobertura (dBm)	-63
Bodega Salida	
Latencia (ms)	182.05
Jitter (ms)	84.2
Paquetes Perdidos (Pmax %)	1.11
Cobertura (dBm)	-73
Bodega Esquina	
Latencia (ms)	207.76
Jitter (ms)	119.55
Paquetes Perdidos (Pmax %)	0.61
Cobertura (dBm)	-71

Tabla 4.1: Parámetros de QoS y Consumo por AP.

A partir de estos datos, se construyen las funciones que representan cada

métrica de interés para la optimización.

4.2.3. Descripción del Archivo de Instancia

Para facilitar el procesamiento de los datos obtenidos en la medición de calidad de servicio (QoS) en cada uno de los puntos de acceso, se creó un archivo denominado `instancia.txt`. Este archivo es utilizado directamente por el algoritmo de optimización para realizar sus cálculos y generar soluciones.

El archivo está estructurado como un archivo de texto plano que contiene exactamente 8 filas, una para cada punto de acceso analizado en el estudio de caso, y 4 columnas, correspondientes a cada una de las métricas QoS en el siguiente orden:

1. Latencia (ms)
2. Jitter (ms)
3. Porcentaje de pérdida de paquetes (%)
4. Cobertura (dBm)

A continuación se muestra un ejemplo del contenido del archivo `instancia.txt`: métodos

```
144.38 137.85 6.77 -67
121.24 52.3 10.86 -53
135.26 60.97 4.4 -56
190.4 83.5 2.94 -56
118.36 69.64 0.85 -73
143.79 79.68 1.96 -63
182.05 84.2 1.11 -73
207.76 119.55 0.61 -71
```

Cálculo de Valores Agregados

Para el análisis y modelado, se calculan los valores totales de cada métrica en la red:

1. Función de Latencia Total:

$L = 144{,}38+121{,}24+135{,}26+190{,}4+118{,}36+143{,}79+182{,}05+207{,}76 = 1243{,}24\,\text{ms}$

2. Función de Jitter Total:

$J = 137{,}85 + 52{,}3 + 60{,}97 + 83{,}5 + 69{,}64 + 79{,}68 + 84{,}2 + 119{,}55 = 687{,}69\,\text{ms}$

3. Función de Pérdida de Paquetes Total:

$$P = 6{,}77 + 10{,}86 + 4{,}4 + 2{,}94 + 0{,}85 + 1{,}96 + 1{,}11 + 0{,}61 = \ 29{,}5\,\%$$

4. Función de Cobertura Total:

$C = -67+(-53)+(-56)+(-56)+(-73)+(-63)+(-73)+(-71) = -512\,\text{dBm}$

Estos valores proporcionan una referencia inicial para la evaluación del rendimiento de la red y sirven como base para la optimización multiobjetivo.

Con base en los datos presentados en esta sección, podemos establecer la base para la optimización de la calidad de servicio en la red Wi-Fi 6 mediante el enfoque de Optimización por Enjambre de Partículas Multiobjetivo (MOPSO).

4.2.4. Consideraciones para la Implementación

- **Normalización de Métricas:** Las métricas transformadas (L', J', P') deben ser normalizadas para evitar que una métrica domine a las demás debido a diferencias en su escala.
- **Elección de Pesos (w_i):** Los pesos deben asignarse cuidadosamente según la prioridad de cada métrica en el contexto de la red. Por ejemplo, si la latencia es crítica, w_1 debe ser mayor que los otros pesos.

Valores maximos sugeridos

Con base en la información presentada en la Subsección 4.1.1, los valores máximos y mínimos sugeridos para garantizar un desempeño óptimo en redes Wi-Fi 6 son los siguientes:

Parámetro	Valor Maximo
Latencia (ms)	50
Jitter (ms)	10
Paquetes Perdidos (Pmax %)	1
Cobertura (dBm)	-67

Tabla 4.2: Valores maximos sugeridos.

4.2.5. Método de Resolución

Con base en la información del algoritmo de Optimización por Enjambre de Partículas (Particle Swarm Optimizacion, PSO) descrito en la Subsección 2.2, se adaptó para poder resolver el problema multiobjetivo presentado en las ecuaciones (4.1) a (4.5).

Para resolver el problema de optimización de la calidad de servicio (QoS) en redes Wi-Fi 6, se implementó un algoritmo de Optimización por Enjambre de Partículas Multiobjetivo (MOPSO). Este enfoque permite encontrar soluciones óptimas considerando múltiples métricas simultáneamente, como la latencia, el jitter, la pérdida de paquetes y la cobertura.

El algoritmo se diseñó para ajustar dinámicamente la configuración de la red de acuerdo con las mediciones de desempeño, logrando una solución adaptativa y eficiente. Se aplicaron técnicas de búsqueda local para mejorar la convergencia y garantizar soluciones de calidad dentro de un margen de variabilidad definido.

PSO Multiobjetivo

El PSO Multiobjetivo (MOPSO) desarrollado en este trabajo adapta la metodología clásica de optimización por enjambre de partículas a un contexto donde se optimizan múltiples criterios simultáneamente. A diferencia del PSO convencional, que busca un solo óptimo global, en MOPSO se busca encontrar un conjunto de soluciones no dominadas que representen el mejor compromiso entre las métricas consideradas.

En este modelo, cada partícula en el enjambre representa una configuración posible de los parámetros de QoS de la red. Su estado se define por:

- **Posición:** Valores actuales de latencia, jitter, pérdida de paquetes y cobertura.
- **Velocidad:** Vector de cambio que define el desplazamiento en el espacio de soluciones.
- **Mejor posición personal (pbest):** La mejor solución encontrada por la partícula individualmente.
- **Mejor posición de vecindario (lbest):** La mejor solución dentro de un subconjunto de partículas cercanas.

El enjambre se inicializa de manera aleatoria dentro de un rango definido basado en las mediciones de la red:

- Se generan valores aleatorios para cada métrica de QoS dentro de un margen de variabilidad permitido.
- Se establecen límites inferiores y superiores para cada variable.
- Se asignan velocidades iniciales pequeñas para evitar oscilaciones abruptas.

Después, cada partícula es evaluada utilizando la siguiente función de calidad global:

$$F = w_1 L' + w_2 J' + w_3 P' + w_4 C \tag{4.6}$$

donde:

- $L' = \frac{1}{L}$ (Latencia inversa, para que sea una métrica de maximización).
- $J' = \frac{1}{J}$ (Jitter inverso).
- $P' = \frac{1}{P}$ (Pérdida de paquetes inversa).
- C (Cobertura, que ya es una métrica de maximización).
- w_1, w_2, w_3, w_4 son pesos que asignan importancia a cada métrica.

Por otro lado, las partículas se mueven en el espacio de soluciones según la siguiente ecuación de actualización de velocidad:

$$v_{ij}(t+1) = v_{ij}(t) + c_1 r_1(pbest_{ij} - x_{ij}(t)) + c_2 r_2(lbest_{ij} - x_{ij}(t)) \qquad (4.7)$$

donde:

- $x_{ij}(t)$ es la posición actual de la partícula i en la dimensión j.
- $v_{ij}(t)$ es la velocidad actual.
- $pbest_{ij}$ es la mejor posición personal.
- $lbest_{ij}$ es la mejor posición del vecindario.
- c_1, c_2 son coeficientes de peso para la influencia personal y social.
- r_1, r_2 son valores aleatorios en el rango $[0, 1]$ que introducen variabilidad.

Pseudocódigo Principal de MOPSO

Con el objetivo de describir el funcionamiento del algoritmo principal, se presenta el Pseudocódigo 0, el cual muestra el flujo general del algoritmo MOPSO:

Algoritmo 3 *MOPSO*
1: **leer** $enjambre, iteraciones, c1, c2$
2: Abrir el archivo para guardar los resultados
3: **si** no se puede abrir el archivo **entonces**
4: Mostrar mensaje de error y salir
5: **fin si**
6: Calcular velocidad máxima ($vmax$) basada en los límites del problema
7: **para** cada iteración desde 0 hasta *iteraciones* **hacer**
8: **para** cada partícula del *enjambre* **hacer** Evaluar la función objetivo utilizando `evaluaFO`
9: **fin para**
10: Actualizar el archivo del Frente de Pareto utilizando `actualizarArchivoPareto`
11: Actualizar velocidad y posición de cada partícula utilizando `actualizarVelocidadYPosicion`
12: **fin para**
13: **para** cada solución en el archivo Pareto **hacer**
14: Escribir sus objetivos en el archivo de resultados
15: **fin para**
16: Cerrar el archivo de resultados
Fin *MOPSO*

El pseudocódigo principal delega el cálculo de la función objetivo y las operaciones sobre el archivo Pareto a funciones auxiliares descritas a continuación.

Función de Evaluación

Algoritmo 4 *evaluaFO*
1: **leer** *individuo*
2: Calcular los objetivos del *individuo*
3: *calidadServicio* ← Suma ponderada de los objetivos usando los pesos
Fin *evaluaFO*

Esta función se encarga de evaluar cada partícula en base a su configuración, calculando los cuatro objetivos propuestos y combinándolos con la fórmula

definida en la ecuación (4.1).

Actualización del Archivo Pareto

Algoritmo 5 *actualizarArchivoPareto*

```
1: leer enjambre
2: para cada partícula p en el enjambre hacer
3:     esDominada ← falso
4:     para cada solución q en el archivo Pareto hacer
5:         si domina(q, p) entonces
6:             esDominada ← verdadero
7:             break
8:         fin si
9:     fin para
10:    si no esDominada entonces
11:        Agregar p al archivo Pareto
12:    fin si
13: fin para
Fin actualizarArchivoPareto
```

Esta rutina permite mantener actualizado el conjunto de soluciones no dominadas (frente de Pareto), filtrando aquellas que son dominadas por otras.

Verificación de Dominancia

Algoritmo 6 *domina*

```
1: leer a, b
2: Comparar a y b en todos sus objetivos
3: si a es mejor o igual en todo y mejor en al menos uno entonces
4:     regresar 1                                  ▷ a domina a b
5: si no
6:     regresar 0
7: fin si
Fin domina
```

Actualización de Velocidad y Posición

Algoritmo 7 *actualizarVelocidadYPosicion*
1: **leer** $enjambre, c1, c2, vmax$
2: **para** cada partícula **hacer**
3: Obtener guía del archivo Pareto utilizando `seleccionarGuia`
4: Calcular nueva velocidad y posición
5: **fin para**
6: Limitar velocidad utilizando `limitarVelocidad`
7: Limitar posición `limitarPosicion`
Fin *actualizarVelocidadYPosicion*

Esta función orquesta el movimiento de cada partícula hacia una guía seleccionada del archivo Pareto, con control de límites de velocidad y posición.

Selección de Guía

Algoritmo 8 *seleccionarGuia*
1: Elegir una solución al azar del archivo Pareto
2: Devolver esa solución como guía
Fin *seleccionarGuia*

Límites de Movimiento

Algoritmo 9 *limitarVelocidad*
1: **leer** $enjambre, vmax$
2: **para** cada partícula **hacer**
3: Limitar su velocidad entre $-vmax$ y $vmax$
4: **fin para**
Fin *limitarVelocidad*

Algoritmo 10 *limitarPosicion*
1: **leer** $enjambre$
2: **para** cada partícula **hacer**
3: Limitar su posición dentro de los valores permitidos
4: **fin para**
Fin *limitarPosicion*

Cada uno de estos componentes auxiliares referenciados desde el Pseudocódigo 0 es fundamental para la ejecución completa y estructurada del algoritmo MOPSO, permitiendo un seguimiento modular y organizado del código fuente.

Calibración de Parámetros con Evolución Diferencial

Para mejorar el desempeño del algoritmo de Optimización por Enjambre de Partículas Multiobjetivo (MOPSO), se realizó una calibración de parámetros utilizando Evolución Diferencial (DE). Este método se utilizó para ajustar automáticamente los coeficientes del PSO (c_1, c_2), el tamaño del enjambre y el margen de diferencia en la inicialización de partículas, asegurando así una mejor exploración y explotación del espacio de búsqueda.

DE es un algoritmo de optimización basado en la evolución biológica, diseñado para manejar espacios de búsqueda continuos. A diferencia de otros algoritmos evolutivos, DE utiliza la mutación diferencial entre individuos de la población para generar nuevas soluciones.

El proceso de DE sigue tres pasos principales en cada iteración:

1. Mutación: Se genera un vector mutado combinando diferencias de tres soluciones aleatorias.
2. Cruzamiento: Se crea un vector de prueba intercambiando componentes entre el vector mutado y el vector original.
3. Selección: Si el vector de prueba tiene una mejor función objetivo, reemplaza al individuo original en la siguiente generación.

Entonces, para la calibración del MOPSO, se definió un espacio de búsqueda para los parámetros, optimizando su configuración para mejorar la convergencia y evitar mínimos locales. Los parámetros ajustados fueron:

- Coeficientes de aprendizaje del MOPSO: c_1 (componente cognitiva) y c_2 (componente social).
- Tamaño del enjambre: Número total de partículas en la simulación.

- Margen de variabilidad: Intervalo dentro del cual se inicializan las partículas en función de los valores de QoS.

El proceso se llevó a cabo con los siguientes pasos:

1. Se generó una población inicial de parámetros (c_1, c_2, tamaño del enjambre, margen de diferencia) de forma aleatoria dentro de rangos definidos empíricamente.

2. Se evaluó cada conjunto de parámetros ejecutando el Mopso con los valores generados y midiendo su desempeño en términos de calidad de la solución y estabilidad.

3. Se aplicó mutación y recombinación diferencial en cada iteración para explorar mejores configuraciones.

4. Se seleccionaron los parámetros que minimizaron la función de error definida como:

$$E = w_1 \cdot L + w_2 \cdot J + w_3 \cdot P - w_4 \cdot C \tag{4.8}$$

   donde L es la latencia, J el jitter, P la pérdida de paquetes y C la cobertura. Los pesos w_i fueron asignados según la importancia relativa de cada métrica.

Después de 100 iteraciones del algoritmo de Evolución Diferencial, se obtuvo la mejor configuración de PSO:

- $c_1 = 0{,}178481$ (Peso de la componente cognitiva).
- $c_2 = 0{,}112131$ (Peso de la componente social).
- Tamaño óptimo del enjambre: 10 partículas.
- Número de iteraciones : 10.

Capítulo 5

Resultados

En este capítulo se presentan los principales hallazgos obtenidos tras el desarrollo e implementación del proyecto. Se analiza cómo varían la Calidad de Servicio al ajustar los parámetros de la red mediante el enfoque propuesto. Asimismo, se comparan los valores antes y después de la optimización, con el objetivo de evaluar la efectividad del método aplicado. Al igual que, se incluyen visualizaciones que permiten observar de manera más clara las mejoras alcanzadas en la calidad del servicio.

5.1. Ejecución Inicial del Algoritmo MOPSO

Antes del proceso de calibración, el algoritmo MOPSO fue ejecutado con los siguientes parámetros:

```
./mopso 30 100 0.8 0.5
```

donde:

- `30`: Tamaño del enjambre (número de partículas)
- `100`: Número de iteraciones
- `0.8`: Coeficiente cognitivo c_1
- `0.5`: Coeficiente social c_2

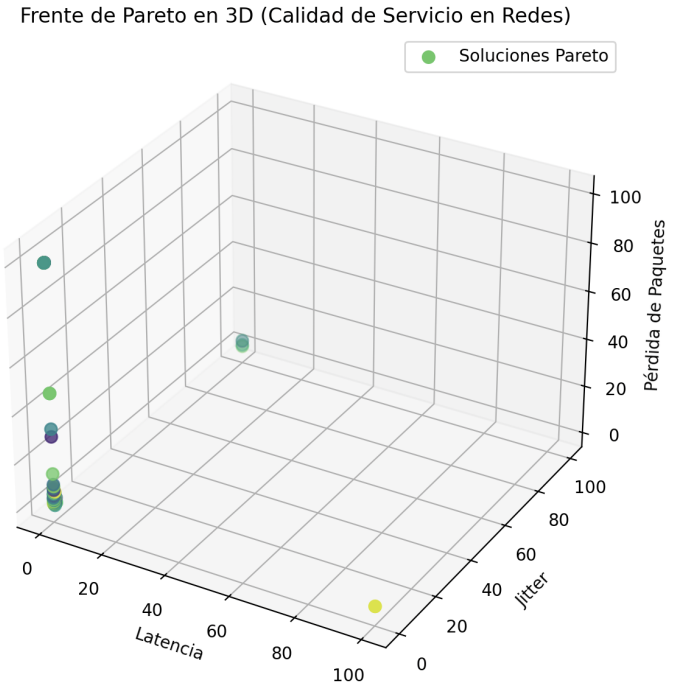

Figura 5.1: Pruebas iniciales del algoritmo MOPSO en red Wi-Fi 6.

Segun la figura 5.1 se muestran los resultados de las pruebas iniciales realizadas con el algoritmo MOPSO. En esta imagen se observa la distribución de soluciones generadas por el algoritmo durante las primeras iteraciones del proceso de optimización. Las soluciones representan combinaciones de las métricas de calidad de servicio.

La figura permite mostrar cómo las partículas comienzan a explorar el espacio de soluciones en busca del frente de Pareto. A pesar de tratarse de pruebas iniciales, ya se puede notar la tendencia del algoritmo a agrupar soluciones en ciertas regiones del espacio, lo que indica un comportamiento adecuado en cuanto a la exploración del problema. Estos resultados sirven como base para la configuración inicial del algoritmo y ajustar los parámetros en pruebas posteriores.

5.2. Optimización de Parámetros Mediante el Algoritmo de Evolución Diferencial

Para mejorar el rendimiento del modelo, se empleó el Algoritmo de Evolución Diferencial (DE), el cual optimizó los parámetros de manera progresiva a lo largo de las iteraciones. A continuación, se presentan varias gráficas que ilustran el comportamiento del algoritmo durante el proceso. Estas gráficas permiten analizar cómo evolucionó el valor de fitness, observar el comportamiento según el tamaño del enjambre, analizar la variación de los valores de fitness a lo largo de las iteraciones, y estudiar los cambios en los parámetros C1 y C2.

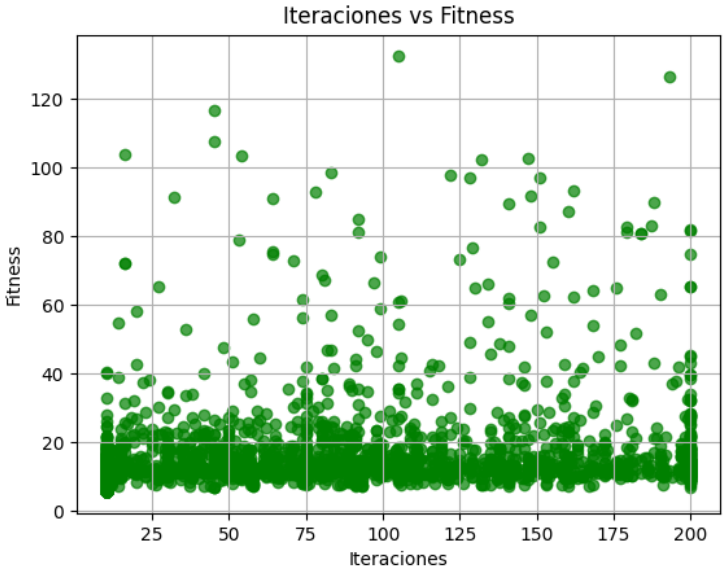

Figura 5.2: Iteraciones VS fitness.

La gráfica 5.2 muestra la evolución del valor de fitness a lo largo de las iteraciones. Se observa que en las primeras iteraciones hay una mejora significativa en el valor de fitness, lo que indica que el algoritmo encuentra rápidamente soluciones prometedoras. A medida que incrementan las iteraciones, el ritmo de mejora disminuye, sugiriendo que el algoritmo comienza a converger hacia

una solución óptima.

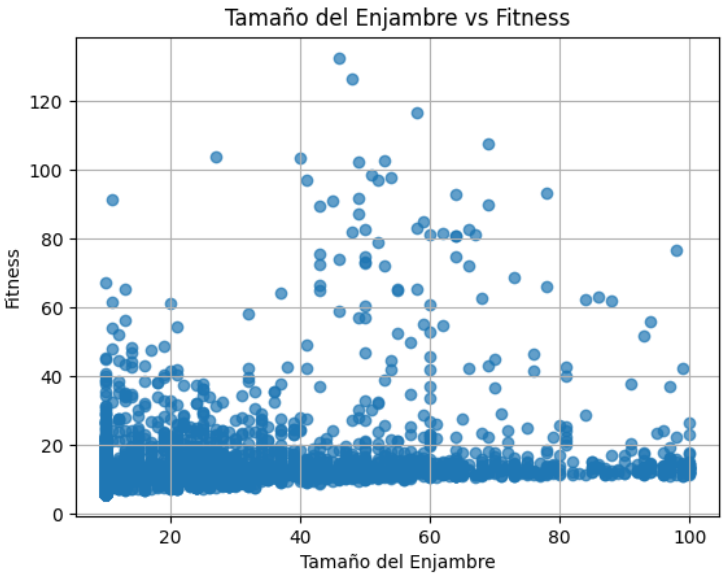

Figura 5.3: Enjambre VS fitness.

La gráfica 5.2 muestra la relación entre el tamaño del enjambre y el valor de fitness. Se observa una alta concentración de soluciones con buen rendimiento cuando el tamaño del enjambre es reducido, lo que sugiere que un enjambre pequeño permite al algoritmo converger de manera eficiente hacia soluciones óptimas, al mismo tiempo que mantiene un bajo costo computacional.

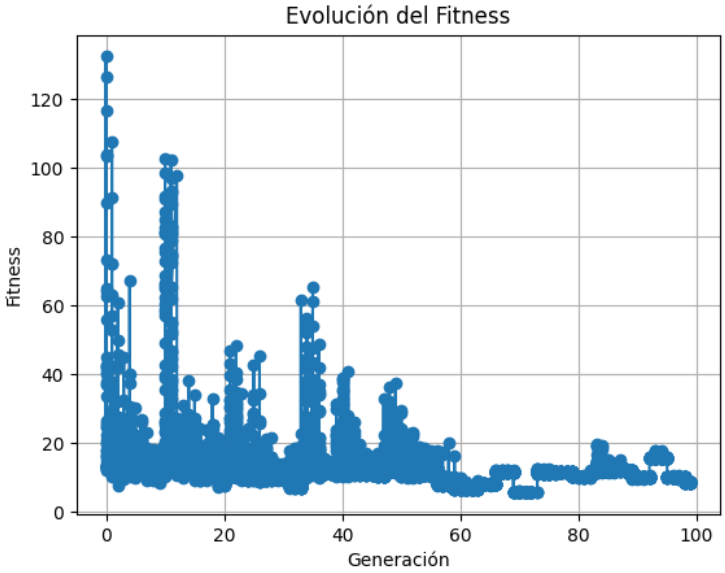

Figura 5.4: Evolución del Fitness a lo Largo de las Generaciones.

La gráfica 5.4 ilustra la mejora del valor del fitness a lo largo de las generaciones. Se puede notar que, tras un número determinado de generaciones, el valor tiende a estabilizarse, lo que confirma la convergencia del algoritmo. Esta estabilización sugiere que las soluciones encontradas en esas generaciones representan configuraciones casi óptimas bajo los criterios definidos.

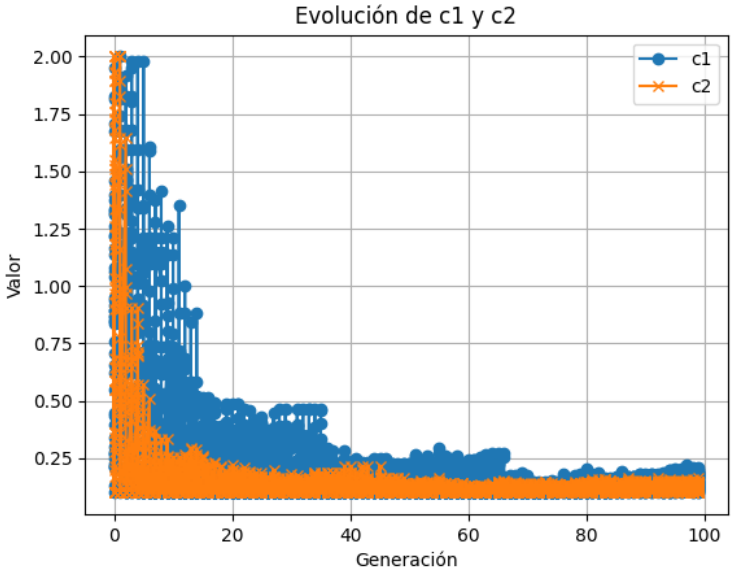

Figura 5.5: Evolución de los parámetros de C1 y C2.

En la gráfica 5.5 se observa cómo varían dinámicamente los parámetros de C1 (componente cognitiva) y C2 (componente social) durante el proceso, primero se puede observar que C1 se mantiene mayor, lo cual favorece la exploración individual de cada partícula. Sin embargo, a lo largo de las generaciones, los valores decrecen, llegando a un punto en donde se permite un equilibrio entre exploración y explotación, lo que resulta esencial para evitar mínimos locales y favorecer la convergencia global.

5.3. Análisis de Resultados del Algoritmo MOPSO

El algoritmo MOPSO fue ejecutado 30 veces para evaluar su capacidad de optimización multiobjetivo en la mejora de la Calidad de Servicio (QoS) en redes Wi-Fi 6. La QoS fue evaluada mediante una función compuesta de los siguientes cuatro objetivos:

1. **Latencia (ms):** Inversa de la latencia, favoreciendo valores bajos.

2. **Jitter (ms):** Inversa del jitter, favoreciendo estabilidad en la transmisión.

3. **Pérdida de Paquetes (%):** Inversa de la tasa de pérdida, premiando redes más fiables.

4. **Potencia de Señal (dBm):** Transformada a un rango positivo para aportar al QoS.

Cada uno de estos objetivos fue ponderado según su importancia relativa para la red:

$$\text{QoS} = 2{,}0\times\text{Latencia}^{-1}+1{,}0\times\text{Jitter}^{-1}+0{,}2\times\text{Pérdida}^{-1}+0{,}5\times(\text{Potencia de Señal}+100)$$

5.3.1. Resultados Numéricos

Tabla 5.1: Resultados de Calidad de Servicio obtenidos en cada ejecución del MOPSO

Índice	Calidad de Servicio (QoS)
0	67.91
1	46.17
2	97.52
3	49.34
4	144.19
5	66.04
6	43.66
7	34.31
8	45.64
9	63.48
10	63.49
11	70.59
12	85.88
13	47.97
14	54.06
15	40.23
16	42.23
17	46.90
18	58.80
19	106.99
20	72.01
21	48.66
22	46.79
23	98.84
24	38.45
25	72.57
26	50.96
27	76.18
28	75.40
29	78.03

La tabla anterior resume los valores obtenidos para la función de calidad de servicio en cada ejecución. Se puede observar una alta variabilidad, con un máximo de 144.19 y un mínimo de 34.31.

5.3.2. Estadísticas Descriptivas

- **Valor Mínimo:** 34.31
- **Valor Máximo:** 144.19
- **Promedio:** 64.44
- **Mediana:** 61.14
- **Desviación Estándar:** 24.21

5.3.3. Interpretación de Resultados

Las soluciones con calidad de servicio superior a 100 suelen tener:

- Latencia menor a 10 ms
- Jitter estable (< 3 ms)
- Pérdida de paquetes cercana a 0
- Potencia de señal cercana a -67 dBm (valor transformado alto)

Por otro lado, las soluciones con QoS baja (< 50) se vinculan a:

- Alta latencia (> 20 ms)
- Jitter variable (> 6 ms)
- Pérdidas significativas ($> 3\,\%$)
- Potencia de señal pobre (< -85 dBm)

5.3.4. Visualización

Figura 5.6: Calidad de Servicio obtenida por cada ejecución del MOPSO.

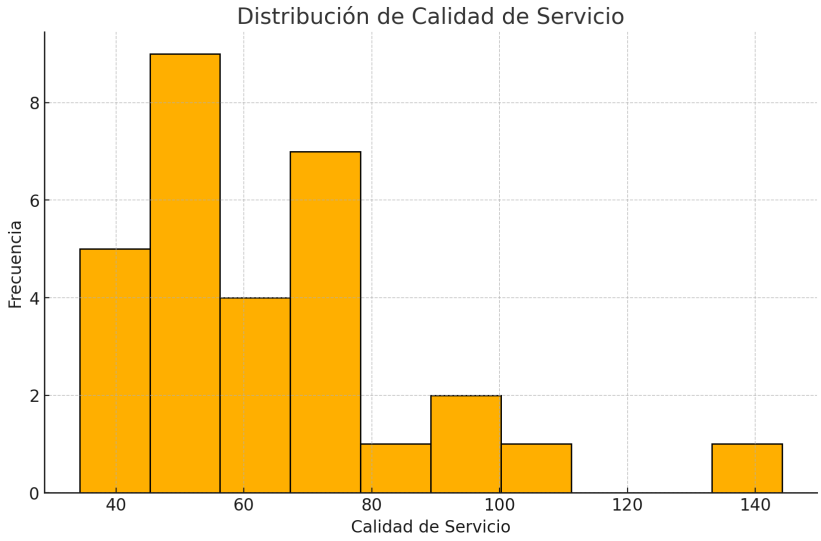

Figura 5.7: Distribución de la Calidad de Servicio en las ejecuciones.

5.3.5. Configuraciones y Acciones Físicas Asociadas

El algoritmo MOPSO optimiza una función que combina variables medibles en simulaciones, pero con un impacto físico directo en la infraestructura de red. En la siguiente tabla se muestran ejemplos representativos de configuraciones generadas por el algoritmo y su correspondiente nivel de QoS:

Tabla 5.2: Configuraciones de variables que generan distintos niveles de QoS

Latencia (ms)	Jitter (ms)	Pérdida (%)	Potencia (dBm)	QoS
15.98	3.38	0.88	-82.67	9.31
19.30	7.97	0.64	-74.65	13.22
17.06	5.11	3.32	-86.97	6.89
15.90	6.12	0.78	-73.52	13.78
13.47	1.17	1.06	-83.34	9.52
17.92	6.56	1.91	-85.79	7.48
13.75	6.51	4.12	-76.51	12.09
22.84	6.55	0.58	-89.54	5.82
24.27	9.49	4.21	-70.93	14.77
12.67	7.14	0.57	-89.89	5.70

La relación entre estas variables y las decisiones de configuración física en la red se describe en la siguiente tabla:

Tabla 5.3: Relación entre Variables Optimizadas y Acciones Físicas en la Red

Variable Optimizada	**Significado Técnico**	**Acciones en el Entorno Físico**
Latencia	Tiempo que tarda un paquete en llegar a su destino desde el AP.	Cambiar ubicación o altura de los APs para mejorar visibilidad. Optimizar canalización y distribución de tráfico. Aplicar tecnologías como OFDMA y MU-MIMO.
Jitter	Variación temporal en el envío de paquetes.	Evitar interferencias electromagnéticas (motores, muros metálicos). Redistribuir APs para reducir congestión. Aplicar políticas QoS por tipo de tráfico.
Pérdida de paquetes	Porcentaje de paquetes que no llegan a su destino.	Ajustar la potencia de los APs para evitar solapamientos. Instalar más APs en zonas críticas. Minimizar interferencias en el espectro.
Potencia de señal (dBm)	Intensidad de la señal desde el punto de vista del dispositivo receptor.	Reubicar APs a zonas elevadas. Aumentar su número en zonas con baja cobertura. Ajustar orientación o banda (2.4/5GHz).

5.3.6. Discusión Aplicada

- **Latencia:** se reduce mediante mejor ubicación de APs y menor interferencia.
- **Jitter:** se estabiliza evitando congestionamiento y mejorando la cober-

tura.

- **Pérdida:** mejora con menor solapamiento y ajuste de potencia.
- **Potencia de Señal:** se incrementa ajustando altura, orientación y densidad de APs.

Capítulo 6

Conclusiones

El desarrollo y aplicación del algoritmo MOPSO para la optimización de la Calidad de Servicio (QoS) en redes Wi-Fi 6 ha demostrado ser una estrategia eficaz y viable para contextos reales. A partir de la formulación multiobjetivo basada en latencia, jitter, pérdida de paquetes y potencia de señal, se logró capturar de manera equilibrada los factores críticos que afectan el rendimiento de la red. Esta estrategia permitió abordar el problema de forma integral, permitiendo un análisis que no se limita a un solo parámetro, sino que evalúa la calidad de la red desde varias dimensiones simultáneamente.

Los resultados obtenidos muestran una diversidad de soluciones que reflejan escenarios reales, desde configuraciones "subóptimas"hasta configuraciones "altamente eficientes". Esto indica que el algoritmo es capaz de explorar adecuadamente el espacio de soluciones, evitando caer en ́optimos locales y manteniendo la diversidad necesaria para adaptarse a condiciones variables dentro del entorno industrial analizado. Esta capacidad de adaptación es fundamental en redes empresariales modernas, donde las condiciones del entorno pueden cambiar rápidamente y de forma impredecible.

Asimismo, la relación establecida entre los valores computacionales y las acciones físicas posibles permite a los encargados de redes tomar decisiones informadas. Por ejemplo, ajustar la ubicación o potencia de los APs en base a los resultados simulados representa una forma directa de traducir el modelo en intervenciones de mejora. Esta correspondencia entre variables numéricas y acciones técnicas concretas refuerza la utilidad práctica del modelo, con-

virtiéndolo en una herramienta aplicable para ingeniería de redes.

En conjunto, este proyecto demuestra que el uso de metaheurísticas como MOPSO no solo es relevante para la teoría de optimización, sino que tiene un gran potencial para impactar directamente en la planeación, diagnóstico y mejora de infraestructuras de redes modernas. El modelo puede integrarse en sistemas de monitoreo o en simuladores de red para analizar distintas configuraciones antes de su implementación física, ahorrando costos y aumentando la eficiencia del despliegue. Además, su flexibilidad lo convierte en una base ideal para incorporar futuras extensiones que contemplen movilidad de usuarios, cambios en la topología o condiciones externas adversas.

6.1. Trabajos futuros

Como trabajo futuro a partir de esta investigación, se propone extender el presente estudio hacia escenarios en los que se utilicen enfoques híbridos que combinen MOPSO con otros algoritmos de inteligencia de enjambre, como algoritmos genéticos, colonia de hormigas o recocido simulado, con el objetivo de aprovechar las fortalezas de cada técnica y mejorar el rendimiento global del sistema. Estas combinaciones pueden permitir un balance más efectivo entre exploración y explotación en el espacio de soluciones, lo que resultaría en una mayor robustez y adaptabilidad ante distintos escenarios de optimización.

Asimismo, se recomienda realizar una comparación exhaustiva del rendimiento de MOPSO frente a otros algoritmos evolutivos multiobjetivo. Este análisis permitiría evaluar la calidad, diversidad y convergencia de los frentes de Pareto generados por cada enfoque, así como su eficiencia computacional en diferentes contextos de red.

Capítulo 7

Apéndices

En este capítulo se presentan las funciones utilizadas en el desarrollo del código. Estas funciones son necesarias para la implementación del algoritmo y proporcionan la base para el procesamiento de los datos, la optimización y la obtención de resultados. Se incluyen explicaciones detalladas de su funcionamiento, parámetros y retornos, con el objetivo de facilitar su comprensión.

7.1. MOPSO

Este programa en C implementa el algoritmo de optimización **MOPSO** (Multi-Objective Particle Swarm Optimization) para resolver un problema de múltiples objetivos, evaluando soluciones basadas en una función objetivo relacionada con las métricas de QoS establecidas en el modelo 4.2.1 .

El código está diseñado para leer una instancia de entrada desde un archivo, generar una población de partículas, evaluar su desempeño, y construir un archivo de soluciones.

Definición de Variables para Evaluación

Estos valores se usan para ponderar los objetivos al calcular la calidad del servicio y establecer los límites de búsqueda para cada variable.

```
1 float pesos[] = {2.0, 1.0, 0.2, 0.5};
2 float limites_max[] = {50.0, 10.0, 1.0, -67.0};
```

```
float limites_min[] = {0.0, 0.0, 0.0, -100.0};
```

Definición de la Estructura Particula

Esta estructura representa una partícula en el algoritmo de optimización de enjambre. Cada partícula tiene varios atributos:

- **Posición:** Contiene la posición de la partícula en el espacio de soluciones.
- **Velocidad:** Contiene la velocidad de la partícula en cada dimensión.
- **Mejor Posición:** Contiene la mejor posición conocida de la partícula hasta el momento.
- **Objetivos:** Es un arreglo que almacena los valores de los objetivos de la partícula.
- **Calidad de Servicio:** Un valor que representa la solución encontrada por la partícula.

```
typedef struct{
    float *posicion;
    float *velocidad;
    float *mejorPosicion;
    float objetivos[NUM_VARIABLES];
    float calidadServicio;
}Particula;
```

Definición de la Estructura Solución Pareto

Esta estructura representa una solución dentro del conjunto de Pareto. Esta estructura contiene:

- **Posición**: Un arreglo que contiene las posiciones de la solución en el espacio de búsqueda.

- **Objetivos**: Un arreglo que almacena los valores de los objetivos para esta solución.
- **Calidad de Servicio**: Un valor que representa la calidad de la solución según los objetivos.

```
typedef struct{
    float posicion[NUM_VARIABLES];
    float objetivos[NUM_VARIABLES];
    float calidadServicio;
}SolucionPareto;
```

Función Genera Aleatorio

Esta función genera un número aleatorio flotante entre los valores min y max. Se utiliza para inicializar las posiciones de las partículas dentro del rango permitido.

```
void generaaleatorio(float *elemento, float inferior, float
    superior){
    float val = (float)rand() / RAND_MAX;
    *elemento = inferior + ((superior - inferior) * val);
}
```

Función evaluaFO

Esta función calcula los valores de la función objetivo para una solución dada y, posteriormente, obtiene la calidad de servicio como una suma ponderada de dichos objetivos.

```
void evaluaFO(float *individuo, float *objetivos, float *
    calidadServicio){
    objetivos[0] = fmin(1.0 / fmax(individuo[0], 0.01),
        1000.0);
    objetivos[1] = fmin(1.0 / fmax(individuo[1], 0.01),
        1000.0);
    objetivos[2] = fmin(1.0 / fmax(individuo[2], 0.01),
        1000.0);
    objetivos[3] = (individuo[3] + 100.0);
```

```
6
7       *calidadServicio = pesos[0] * objetivos[0] +
8                          pesos[1] * objetivos[1] +
9                          pesos[2] * objetivos[2] +
10                         pesos[3] * objetivos[3];
11  }
```

Función Domina

Esta función determina si una solución **a** domina a otra solución **b**. Se considera que **a** domina a **b** cuando no es peor en ninguno de los objetivos y es mejor en al menos uno de ellos.

```
1  int domina(float *a, float *b){
2      int mejor = 0;
3      for(int i = 0; i < NUM_VARIABLES; i++){
4          if(a[i] > b[i]) mejor = 1;
5          else if(a[i] < b[i]) return 0;
6      }
7      return mejor;
8  }
```

Función Actualiza Archivo Pareto

Esta función recorre todas las partículas del enjambre y agrega al archivo de Pareto aquellas que no están dominadas por ninguna solución del archivo. Además, reemplaza las soluciones del archivo que son dominadas por las nuevas.

```
1  void actualizarArchivoPareto(Particula *enjambre){
2      for(int i = 0; i < TAM_ENJAMBRE; i++){
3          int esDominado = 0;
4          for(int j = 0; j < tamArchivoPareto; j++){
5              if(domina(archivoPareto[j].objetivos, enjambre[i
                   ].objetivos)){
6                  esDominado = 1;
7                  break;
8              }
9          }
10         if(!esDominado && tamArchivoPareto < ARCHIVO_PARETO){
```

```
11              for(int k = 0; k < NUM_VARIABLES; k++){
12                  archivoPareto[tamArchivoPareto].posicion[k] =
                        enjambre[i].posicion[k];
13                  archivoPareto[tamArchivoPareto].objetivos[k]
                        = enjambre[i].objetivos[k];
14              }
15              archivoPareto[tamArchivoPareto].calidadServicio =
                    enjambre[i].calidadServicio;
16              tamArchivoPareto++;
17          }
18      }
19  }
```

Función Seleccionar Guia

Esta función selecciona aleatoriamente una solución del archivo de Pareto. Esta solución se usa como guía para que las partículas actualicen su trayectoria.

```
1  void seleccionarGuia(float *lbest){
2      if(tamArchivoPareto == 0) return;
3      int idx = rand() % tamArchivoPareto;
4      for(int j = 0; j < NUM_VARIABLES; j++){
5          lbest[j] = archivoPareto[idx].posicion[j];
6      }
7  }
```

Función Limitar Posición

Esta función verifica que cada variable de la posición esté dentro de sus límites mínimos y máximos. Si alguna variable se sale del rango, se ajusta al límite correspondiente.

```
1  void limitarPosicion(Particula *enjambre){
2      for(int i = 0; i < TAM_ENJAMBRE; i++){
3          for(int j = 0; j < NUM_VARIABLES; j++){
4              if(enjambre[i].posicion[j] < limites_min[j])
                   enjambre[i].posicion[j] = limites_min[j];
5              if(enjambre[i].posicion[j] > limites_max[j])
                   enjambre[i].posicion[j] = limites_max[j];
```

```
        }
    }
}
```

Función Limitar velocidad

Esta función restringe la velocidad de cada particula dentro de un rango. Esto evita que las partículas se desplacen demasiado rápido en el espacio de búsqueda.

```
void limitarVelocidad(Particula *enjambre, float vmax){
    for(int i = 0; i < TAM_ENJAMBRE; i++){
        for(int j = 0; j < NUM_VARIABLES; j++){
            if(enjambre[i].velocidad[j] > vmax) enjambre[i].
               velocidad[j] = vmax;
            if(enjambre[i].velocidad[j] < -vmax) enjambre[i].
               velocidad[j] = -vmax;
        }
    }
}
```

Función Actualizar Posición y Velocidad

Esta función actualiza la velocidad y la posición de una partícula. Aplica los factores de aprendizaje c1 y c2. Luego, limita la velocidad y la posición resultantes.

```
void actualizarVelocidadYPosicion(Particula *enjambre, float
   c1, float c2, float vmax){
    for(int i = 0; i < TAM_ENJAMBRE; i++){
        float lbest[NUM_VARIABLES];
        seleccionarGuia(lbest);

        for(int j = 0; j < NUM_VARIABLES; j++){
            float r1 = (float)rand() / RAND_MAX;
            float r2 = (float)rand() / RAND_MAX;
            enjambre[i].velocidad[j] += c1 * r1 * (enjambre[i
               ].mejorPosicion[j] - enjambre[i].posicion[j])
               +
```

```
10                                          c2 * r2 * (lbest[j] -
                                                enjambre[i].
                                               posicion[j]);
11              enjambre[i].posicion[j] += enjambre[i].velocidad[
                    j];
12          }
13      }
14
15      limitarVelocidad(enjambre, vmax);
16      limitarPosicion(enjambre);
17  }
```

Función Cargar Instancia

Esta función lee un archivo de texto que contiene una matriz de valores de tamaño 8x4. Cada fila y columna se almacenan en una matriz, con el fin de guardar los valores correspondientes a cada Access Point con sus respectivas métricas. Como se menciona en la sección Descripción del Archivo de Instancia.

```
1  void cargarInstancia(const char *archivo_instancia){
2      FILE *file = fopen(archivo_instancia, "r");
3      if(!file){
4          printf("No se pudo abrir el archivo %s\n",
               archivo_instancia);
5          exit(1);
6      }
7      for(int i = 0; i < 8; i++){
8          for(int j = 0; j < NUM_VARIABLES; j++){
9              fscanf(file, "%f", &instancia[i][j]);
10         }
11     }
12     fclose(file);
13 }
```

Función MOPSO

Esta es la función principal del algoritmo **MOPSO**. En cada iteración, evalúa los objetivos y la calidad de servicio de las partículas. Actualiza el archivo

de soluciones. Luego, actualiza la velocidad y posición de cada partícula. Al finalizar, escribe las soluciones en un archivo.

```
void MOPSO(Particula *enjambre, int iteraciones, float c1,
    float c2){
    FILE *archivo = fopen("pareto_results.txt", "w");
    if(!archivo){
        printf("Error al abrir el archivo de resultados.\n");
        return;
    }
    float vmax = 0.1 * (limites_max[0] - limites_min[0]);  //
        Velocidad maxima basada en el rango

    for(int a = 0; a < iteraciones; a++){
        //printf("\nIteracion %d", a + 1);
        for(int i = 0; i < TAM_ENJAMBRE; i++){
            evaluaFO(enjambre[i].posicion, enjambre[i].
                objetivos, &enjambre[i].calidadServicio);
        }
        actualizarArchivoPareto(enjambre);
        actualizarVelocidadYPosicion(enjambre, c1, c2,vmax);
    }

    for(int i = 0; i < tamArchivoPareto; i++){
        fprintf(archivo, "%f %f %f %f\n", archivoPareto[i].
            objetivos[0], archivoPareto[i].objetivos[1],
            archivoPareto[i].objetivos[2], archivoPareto[i].
            objetivos[3]);
    }
    fclose(archivo);
}
```

7.2. Calibración de Métricas

El propósito general de este código es optimizar los parámetros de un algoritmo de optimización multi-objetivo basado en el **MOPSO** (Multi-Objective Particle Swarm Optimization). Para lograrlo, se utiliza un enfoque de **Evolución Diferencial** (DE) para ajustar los parámetros del algoritmo, como el tamaño del enjambre, el número de iteraciones y los coeficientes de aceleración (c1 y c2).

El código implementa un proceso de optimización en el que se generan individuos con valores aleatorios para estos parámetros dentro de un rango específico. Luego, se evalúan sus soluciones mediante el algoritmo **MOPSO** y se utiliza la Evolución Diferencial para ajustar los parámetros, buscando una mejor solución en función de una métrica de rendimiento **Fitness**.

Definición de Parámetros Globales

Se definen constantes que determinan el tamaño de la población, el número de generaciones y los valores límite para los parámetros de MOPSO, estos valores controlan el comportamiento del algoritmo de Evolución Diferencial.

```
#define POP_SIZE 10    //tamanio de la poblacion en DE
#define GEN_MAX 50     //numero maximo de generaciones en DE
#define F 0.5          //factor de mutacion en DE
#define CR 0.5         //factor de recombinacion en DE
```

También se definen los rangos de búsqueda de los parámetros del algoritmo MOPSO. Aquí se establecen los valores mínimo y máximo del tamaño del enjambre (tamEnjambre), el número de iteraciones (iteraciones), y los coeficientes de inercia c1 y c2.

```
//rango de parametros del Algoritmo MOPSO
#define MIN_ENJAMBRE 10
#define MAX_ENJAMBRE 100
#define MIN_ITER 10
#define MAX_ITER 200
#define MIN_C 0.1
#define MAX_C 2.0
```

Definición de la Estructura Individuo

La estructura Individuo representa la configuración del algoritmo MOPSO con cuatro atributos:

- **tamEnjambre:** Número de partículas en el enjambre.
- **iteraciones:** Número de iteraciones del algoritmo MOPSO.

- **c1, c2:** Coeficientes de aceleración en MOPSO.

```
typedef struct{
    int tamEnjambre;
    int iteraciones;
    double c1;
    double c2;
}Individuo;
```

Función Random Range

La función random range genera un número aleatorio dentro de un rango específico, tomando un valor mínimo y un valor máximo definidos.

```
//generamos un numero aleatorio en un rango dado
double random_range(double min, double max){
    return min + ((double)rand() / RAND_MAX) * (max - min);
}
```

Función Evaluar

Esta función evalúa un Individuo ejecutando el algoritmo MOPSO con sus parámetros específicos.

1. Construye un comando en C para el codigo del MOPSO con los valores del Individuo.
2. Ejecuta el comando en la terminal y almacena los resultados en un archivo.
3. Abre el archivo y extrae el valor del fitness, que representa la calidad de la configuración evaluada.

```
double evaluar(Individuo ind){
    char comando[500];
    snprintf(comando, sizeof(comando), "./mopso %d %d %lf %lf
        > resultados.txt",
```

```
4               ind.tamEnjambre, ind.iteraciones, ind.c1, ind.c2
                    );
5
6       system(comando);
7
8       FILE *fp = fopen("resultados.txt", "r");
9       if (!fp) return -1e9; //si falla, devolver un valor chico
            (mala solucion)
10      double fitness;
11      fscanf(fp, "%lf", &fitness);
12      fclose(fp);
13      return fitness;
14  }
```

Función Evolucion Diferencial

Esta función ejecuta la Evolución Diferencial para encontrar la mejor combinación de parámetros que optimicen la función de evaluación.

Inicialización de la Población

Se crea una población inicial de individuos con valores aleatorios dentro de los rangos definidos.

```
1   void evolucion_diferencial(){
2       FILE *log = fopen("historial_configuraciones.csv", "w");
3      if(log)
4          fprintf(log, "Generacion,TamEnjambre,Iteraciones,c1,c2
               ,Fitness\n");
5       Individuo poblacion[POP_SIZE];
6       Individuo mejor;
7       double mejor_fitness = 1e9;
8
9       srand(time(NULL));
10      for(int i = 0; i < POP_SIZE; i++){
11          poblacion[i].tamEnjambre = (int)random_range(
                MIN_ENJAMBRE, MAX_ENJAMBRE);
12          poblacion[i].iteraciones = (int)random_range(MIN_ITER
                , MAX_ITER);
13          poblacion[i].c1 = random_range(MIN_C, MAX_C);
14          poblacion[i].c2 = random_range(MIN_C, MAX_C);
```

```
15      }
```

Proceso Evolutivo

Se seleccionan tres individuos a, b y c de la población de manera aleatoria.

```
1       for(int gen = 0; gen < GEN_MAX; gen++){
2           for(int i = 0; i < POP_SIZE; i++){
3               int a, b, c;
4               do{ a = rand() % POP_SIZE; }while (a == i);
5               do{ b = rand() % POP_SIZE; }while (b == i || b ==
                    a);
6               do{ c = rand() % POP_SIZE; }while (c == i || c ==
                    a || c == b);
```

Creación de Individuo

Se genera un nuevo individuo combinando parámetros de a, b y c con probabilidad CR y factor de mutación F.

```
1               Individuo trial = poblacion[i];
2               if ((rand() / (double)RAND_MAX) < CR) trial.
                    tamEnjambre = (int)(poblacion[a].tamEnjambre +
                     F * (poblacion[b].tamEnjambre - poblacion[c].
                    tamEnjambre));
3               if ((rand() / (double)RAND_MAX) < CR) trial.
                    iteraciones = (int)(poblacion[a].iteraciones +
                     F * (poblacion[b].iteraciones - poblacion[c].
                    iteraciones));
4               if ((rand() / (double)RAND_MAX) < CR) trial.c1 =
                    poblacion[a].c1 + F * (poblacion[b].c1 -
                    poblacion[c].c1);
5               if ((rand() / (double)RAND_MAX) < CR) trial.c2 =
                    poblacion[a].c2 + F * (poblacion[b].c2 -
                    poblacion[c].c2);
```

Corrección de Valores Fuera de Rango

Se asegura que los valores no excedan los límites establecidos.

```
            if (trial.tamEnjambre < MIN_ENJAMBRE) trial.
                tamEnjambre = MIN_ENJAMBRE;
            if (trial.tamEnjambre > MAX_ENJAMBRE) trial.
                tamEnjambre = MAX_ENJAMBRE;
            if (trial.iteraciones < MIN_ITER) trial.
                iteraciones = MIN_ITER;
            if (trial.iteraciones > MAX_ITER) trial.
                iteraciones = MAX_ITER;
            if (trial.c1 < MIN_C) trial.c1 = MIN_C;
            if (trial.c1 > MAX_C) trial.c1 = MAX_C;
            if (trial.c2 < MIN_C) trial.c2 = MIN_C;
            if (trial.c2 > MAX_C) trial.c2 = MAX_C;
```

Evaluación y Selección

Si la nueva solución es mejor que la solución actual, la reemplaza.

```
            double fitness_trial = evaluar(trial);
            double fitness_actual = evaluar(poblacion[i]);
            if(log)
        fprintf(log, "%d,%d,%d,%lf,%lf,%lf\n", gen, trial.
            tamEnjambre, trial.iteraciones, trial.c1, trial.c2
            , fitness_trial);
            if(fitness_trial < fitness_actual){
                poblacion[i] = trial;
                if(fitness_trial < mejor_fitness){
                    mejor_fitness = fitness_trial;
                    mejor = trial;
                }
            }
        }
    }
```

Impresión de Resultados

Se muestran los mejores parámetros encontrados como resultado de la ejecución del algoritmo de evolución diferencial.

```
    // Mostramos la mejor configuracion encontrada
    printf("Mejores parametros para el Algoritmo MOPSO:\n");
```

```
    printf("Tamanio del enjambre: %d\n", mejor.tamEnjambre);
    printf("Numero de iteraciones: %d\n", mejor.iteraciones);
    printf("Parametro c1: %lf\n", mejor.c1);
    printf("Parametro c2: %lf\n", mejor.c2);
}
```

7.3. Visualización Frente de Pareto (MOPSO)

El código carga los resultados de un archivo de texto, que contiene soluciones obtenidas mediante el MOPSO. Luego, visualiza las soluciones en un gráfico 3D para analizar la relación entre métricas de Calidad de Servicio (QoS) en Redes.

Definición del Archivo y Carga de Datos

Se almacena en una variable el nombre del archivo que contiene las soluciones del frente de Pareto. Luego, el archivo se lee y su contenido se guarda en una variable, estructurada como una matriz donde cada fila representa una solución y cada columna corresponde a una métrica diferente.

```
archivo_pareto = "pareto_results.txt"
# Cargamos los datos del archivo
pareto_data = np.loadtxt(archivo_pareto)
```

Separación en Variables Individuales

Se separan las columnas en variables individuales, es decir, se extraen las métricas de calidad de servicio (QoS) de cada solución: latencia, jitter, pérdida de paquetes y cobertura.

```
latencia = pareto_data[:, 0]
jitter = pareto_data[:, 1]
perdida_paquetes = pareto_data[:, 2]
cobertura = pareto_data[:, 3]
```

Creación de la Visualización

Se crea una figura para visualizar los datos. Después, se representa un conjunto de puntos en el espacio tridimensional, donde cada punto está definido por tres variables **latencia, jitter y pérdida de paquetes**. El color de cada punto se asigna en función de la variable **cobertura**.

```
1
2 fig = plt.figure(figsize=(10, 7))
3 ax = fig.add_subplot(111, projection='3d')
4
5 ax.scatter(latencia, jitter, perdida_paquetes, c=
    cobertura, cmap='viridis', s=50, label="Soluciones
    Pareto")
6 ax.set_xlabel("Latencia")
7 ax.set_ylabel("Jitter")
8 ax.set_zlabel("Perdida de Paquetes")
9 ax.set_title("Frente de Pareto en 3D (Calidad de
    Servicio en Redes)")
```

7.4. Visualización de Resultados del Algoritmo MOPSO

En esta sección se presentan diferentes gráficas que permiten analizar el comportamiento del algoritmo MOPSO a lo largo de las generaciones. Se incluyen visualizaciones de la evolución del valor de fitness, los cambios en los parámetros de aprendizaje c1 y c2, así como la relación entre el tamaño del enjambre, el número de iteraciones y el rendimiento obtenido. Estos gráficos son útiles para evaluar la efectividad de las configuraciones utilizadas.

Cargar datos

Se carga el archivo .csv en un DataFrame de pandas para su manipulación y análisis.

```
1 df = pd.read_csv("historial_configuraciones.csv")
```

Gráfica 1: Evolución del Fitness

Muestra cómo evoluciona el valor de fitness a lo largo de las generaciones. Esta gráfica permite observar si el algoritmo converge o mejora con el tiempo.

```
plt.figure()
plt.plot(df["Generacion"], df["Fitness"], marker='o')
plt.title("Evolucion del Fitness")
plt.xlabel("Generacion")
plt.ylabel("Fitness")
plt.grid(True)
plt.savefig("evolucion_fitness.png")
plt.show()
```

Gráfica 2: Evolución de los parámetros c1 y c2

Se representa gráficamente cómo varían los parámetros de aceleración c1 y c2 durante las generaciones.

```
plt.figure()
plt.plot(df["Generacion"], df["c1"], label="c1", marker=
    'o')
plt.plot(df["Generacion"], df["c2"], label="c2", marker=
    'x')
plt.title("Evolucion de c1 y c2")
plt.xlabel("Generacion")
plt.ylabel("Valor")
plt.legend()
plt.grid(True)
plt.savefig("evolucion_c1_c2.png")
plt.show()
```

Gráfica 3: Tamaño del enjambre vs Fitness

Se genera un diagrama de dispersión que relaciona el tamaño del enjambre con el valor de fitness, lo que ayuda a identificar si enjambres más grandes tienden a obtener mejores resultados.

```
plt.figure()
plt.scatter(df["TamEnjambre"], df["Fitness"], alpha=0.7)
plt.title("Tamanio del Enjambre vs Fitness")
plt.xlabel("Tamanio del Enjambre")
plt.ylabel("Fitness")
plt.grid(True)
plt.savefig("enjambre_vs_fitness.png")
plt.show()
```

Gráfica 4: Iteraciones vs Fitness

Se representa la relación entre la cantidad de iteraciones ejecutadas y el valor de fitness alcanzado, lo que permite evaluar la eficiencia del algoritmo en función del tiempo.

```
plt.figure()
plt.scatter(df["Iteraciones"], df["Fitness"], alpha=0.7,
    color="green")
plt.title("Iteraciones vs Fitness")
plt.xlabel("Iteraciones")
plt.ylabel("Fitness")
plt.grid(True)
plt.savefig("iteraciones_vs_fitness.png")
plt.show()
```

7.5. Consulta del Código Fuente

Para obtener un análisis más detallado de la implementación, se puede consultar el código fuente completo disponible en el repositorio de GitHub en el siguiente enlace [21]. En él, se incluyen todos los archivos correspondientes a la implementación de los algoritmos, funciones auxiliares y estructuras utilizadas a lo largo del proyecto.

Bibliografía

[1] H. E. Abdelkader, A. G. Gad, A. A. Abohany, and S. E. Sorour. An efficient data mining technique for assessing satisfaction level of online learning for higher education students during the covid-19. *IEEE Access*, 2022.

[2] R. Albert, H. Jeong, and A.-L. Barabási. Diameter of the world-wide web. *Nature*, 401:130–131, 1999.

[3] María Luisa Marí Altozano. *Optimización de la calidad de experiencia en redes celulares mediante el ajuste del traspaso entre celdas*. PhD thesis, Universidad de Málaga, 2021.

[4] J. Bang-Jensen, M. Chiarandini, and P. Morling. A computational investigation on heuristic algorithms for 2-edge-connectivity augmentation. *Technical Report*, 2008.

[5] A. Banks, J. Vincent, and C. Anyakoha. A review of particle swarm optimization part i: Background and development. *Natural Computing*, 6(2):133–155, 2007.

[6] A. Banks, J. Vincent, and C. Anyakoha. A review of particle swarm optimization part ii: Hybridisation, combinatorial problems and applications. *Natural Computing*, 7(1):1–32, 2008.

[7] A.-L. Barabási and R. Albert. Emergence of scaling in random networks. *Science*, 286:509–512, 1999.

[8] A. Bhattacharya, R. T. Goswami, and K. Mukherjee. A feature selection technique based on rough set and improvised pso algorithm (psors-fs) for permission based detection of android malwares. *International Journal*

of Machine Learning and Cybernetics, 10:1893–1907, 2019. doi: 10.1007/s13042-018-0899-8.

[9] S. Bhulai, R. van der Mei, and M. Wu. Heuristics for the design and optimization of streaming content distribution networks. *Technical Report*, 2004.

[10] Ingeniería Básica. El problema de los puentes de königsberg, 2019. URL `https://ingenieriabasica.es/el-problema-de-los-puentes-de-konigsberg/`. Accedido: 23 de septiembre de 2024.

[11] Stephen Cook. The p versus np problem. *Clay Mathematics Institute*, 2(6):3, 2000.

[12] Marely del Rosario Cruz Felipe, Reinier Martínez Gómez, and Yosuan Crespo García. Análisis de la qos en redes inalámbricas. *Revista Cubana de Ciencias Informáticas*, 7(1):86–96, 2013.

[13] Bibiana Andrea Cuartas Torres. *Metodología para la optimización de múltiples objetivos basada en ag y uso de preferencias*. PhD thesis, 2009.

[14] Sergio Gerardo De los Cobos. *Búsqueda y exploración estocástica*. Universidad Autónoma Metropolitana, 2010.

[15] Sergio Gerardo de los Cobos Silva, Miguel Ángel Gutiérrez Andrade, Eric Alfredo Rincón García, Pedro Lara Velázquez, and Manuel Aguilar Cornejo. Colonia de abejas artificiales y optimización por enjambre de partículas para la estimación de parámetros de regresión no lineal. *Revista de matematica: teoria y aplicaciones*, 21(1):107–126, 2014.

[16] Universidad Autónoma de México. Combinatoria: Los puentes de königsberg, 2024. URL `https://prometeo.matem.unam.mx/recursos/Licenciatura/IPM_UAM_CUAJIMALPA//scorm_player/2062/content/index.html`. Último acceso: 22 de septiembre de 2024.

[17] Russell C Eberhart, Yuhui Shi, and James Kennedy. *Swarm intelligence*. Elsevier, 2001.

[18] M. Faloutsos, P. Faloutsos, and C. Faloutsos. On power-law relationships of the internet topology. *Computer Communication Review*, 29:251–262, 1999.

[19] Luis Fernando Galindres G., Antonio H. Escobar Z., and Ramón A. Gallego Rendón. Optimización de redes de comunicación por cable, un enfoque multiobjetivo. *Revista Ingeniería y Desarrollo*, 28, Julio-Diciembre 2010. ISSN 0122-3461.

[20] Diego Antonio Gonzalez Moreno. *Introducción a la Teoría de las Gráficas*. México: UAM, Unidad Cuajimalpa, División de Ciencias Naturales e Ingeniería, 2017.

[21] José Alberto Posadas Gudiño. Análisis de la calidad de servicio en redes de telecomunicaciones utilizando algoritmos de enjambre de partículas. `https://github.com/Jose1503-posadas/Optimizacion-Red-Telecomunicaciones-MOPSO.git`, 2025.

[22] Arthur D. Hall. *Ingeniería de sistemas*. C.E.C.S.A., 1983.

[23] IEL2-I-2004-29. Confiabilidad en sistemas de telecomunicaciones. Proyecto de Grado, 2004.

[24] M. Imran, N. Hashim, N. Khalid, E. Naveed, and H. Tariq. A survey of particle swarm optimization algorithms. *Information Sciences*, 1(1): 1–34, 2013.

[25] Intel. What is wi-fi 6e?, 2023. URL `https://www.intel.com/content/www/us/en/products/docs/wireless/wi-fi-6e.html`. Accessed: 2024-09-26.

[26] Tim M. Jones. *Ai Application Programming (Charles River Media Programming)*. Charles River Media, Inc., USA, 2005. ISBN 1584504218.

[27] Cristina Jordan Lluch. Grafos ponderados. problema de los caminos más cortos, 2024.

[28] Bernhard H Korte, Jens Vygen, B Korte, and J Vygen. *Combinatorial optimization*, volume 1. Springer, 2011.

[29] J. Lorincz, A. Capone, and D. Begušić. Heuristic algorithms for optimization of energy consumption in wireless access networks. *KSII Transactions on Internet and Information Systems*, 5(4):626–648, 2011.

[30] David G. Luenberger. *Linear and Nonlinear Programming*. Addison-Wesley, 1984.

[31] Guillermo Marqués. *QoS en routers y switches Cisco*. Lulu. com, 2016.

[32] Oscar Meza and Maruja Ortega. Grafos y algoritmos, 2006.

[33] Erfan Mozaffariahrar, Fabrice Theoleyre, and Michael Menth. A survey of wi-fi 6: Technologies, advances, and challenges. *Future Internet*, 14 (10):293, 2022.

[34] M. E. J. Newman. *Networks: An Introduction*. Oxford University Press, Oxford, UK, 2010. ISBN 978-0-19-920665-0.

[35] Rodrigo Perez Pena. *Introducción a los modelos de optimización*. Sello Editorial UniPiloto, 2019.

[36] Aurora Ramírez, José Raúl Romero, Carlos García-Martínez, and Sebastián Ventura. Jclec-mo: a java suite for solving many-objective optimization engineering problems. *Engineering Applications of Artificial Intelligence*, 81:14–28, 2019.

[37] Juan José Salazar-González. *Programación matemática*. Ediciones Díaz de Santos, 2001.

[38] Marco Antonio Márquez Vera. Inteligencia de enjambre: de los sistemas naturales a los artificiales. *Revista Digital Universitaria*, 24(1), 2023.

[39] D. J. Watts and S. H. Strogatz. Collective dynamics of small-world networks. *Nature*, 393(6684):440–442, 1998.

[40] H. Yang, Y. Zhou, Q. Luo, and J. Zhou. A novel swarm intelligence optimization approach: Sparrow search algorithm. *Systems Science & Control Engineering: An Open Access Journal*, 8(1):22–34, 2020.

[41] Y. Zhang, S.-H. Chen, and Z.-H. Ling. A comprehensive survey on particle swarm optimization algorithm and its applications. *Mathematical Problems in Engineering*, 2015:1–38, 2015.