

**UNIVERSIDAD AUTÓNOMA  
METROPOLITANA**

---

---

UNIDAD CUAJIMALPA



**RESOLUCIÓN DEL PROBLEMA DE  
RUTAS VEHICULARES CON  
VENTANAS DE TIEMPO MEDIANTE  
UN ALGORITMO HÍBRIDO ENTRE  
COLONIA DE HORMIGAS Y  
RECOCIDO SIMULADO**

**Proyecto Terminal**

QUE PRESENTA:  
**ALEJANDRO MARTÍNEZ GUZMÁN**

**LICENCIATURA EN INGENIERÍA EN  
COMPUTACIÓN**

Departamento de Matemáticas Aplicadas y Sistemas

División de Ciencias Naturales e Ingeniería

Asesor:  
**EDWIN MONTES OROZCO**

**Junio 2025**



# Declaración

Yo, ALEJANDRO MARTÍNEZ GUZMÁN, declaro que este trabajo titulado «*Resolución del Problema de Rutas Vehiculares con Ventanas de Tiempo mediante un Algoritmo Híbrido entre Colonia de Hormigas y Recocido Simulado*» es de mi autoría. Asimismo, confirmo que:

- Este trabajo fue realizado en su totalidad para la obtención de grado en esta Universidad.
- Ninguna parte de esta tesis ha sido previamente sometida a un examen de grado o titulación en esta u otra institución.
- Todas las citas han sido debidamente referenciadas y atribuidas a sus autores.

Firma: \_\_\_\_\_

Fecha: \_\_\_\_\_



# Agradecimientos y Dedicatoria

Quisiera comenzar agradeciendo a Dios, por ser mi guía y acompañante en cada etapa de mi vida académica. Su presencia me ha dado fortaleza y sabiduría para superar los retos y alcanzar mis metas.

A la Universidad Autónoma Metropolitana, en especial a la Unidad Cuajimalpa y al Departamento de Matemáticas Aplicadas y Sistemas, por ser una pieza clave en mi formación académica, profesional y personal, sin dejar atrás a cada uno de mis profesores, de quienes siempre guardaré un recuerdo especial.

Al Dr. Edwin Montes Orozco, amigo, profesor y asesor de este proyecto, por su constante apoyo, orientación, paciencia y valiosos consejos durante todo el proceso de investigación y desarrollo. Su conocimiento, experiencia y confianza fueron determinantes para la culminación exitosa de este trabajo, así como su experticia compartida en proyectos de la industria.

A mis padres, Laura y Alejandro, por creer siempre en mí, apoyarme incondicionalmente y enseñarme a confiar en mis capacidades. Su amor, cariño, motivación y consejos constantes han sido el pilar que me sostiene y la inspiración para esforzarme y alcanzar mis metas.

A mi hermana Linda, por su apoyo, compañía y amor incondicional, siempre motivándome y acompañándome en cada etapa de mi vida.

A mis abuelos maternos, Margarita y Salvador, gracias por consentirme, brindarme cariño y enseñanzas que han marcado mi vida desde pequeño.

A mis abuelos paternos, Flor y Trinidad, gracias por su apoyo, ejemplo y cariño constante, que me han acompañado en cada paso de mi formación y crecimiento personal.

A Alpha

A Jaime López, por ser mi gran amigo, por estar siempre dispuesto a ayudar y hacerme sentir que puedo contar con él para lo que sea.

A Sara Hernandez,

A Rebeca López, por recordarme que enseñar también es aprender.

A Alfredo López,

A mi tío Javier Martínez, quien me introdujo al mundo de la computación cuando era niño y a quien considero una fuente de inspiración al elegir esta carrera.

A todos, gracias por ser parte de este gran sueño llamado Ingeniería en Computación.

# **Resumen**

Aquí va el resumen en español. Este apartado debe sintetizar brevemente el objetivo del trabajo, la metodología empleada y los resultados más relevantes.



# **Abstract**

Here goes the abstract in English. Briefly describe the goal of your project, methodology, and key results.



*"Tu talento es un regalo de Dios para ti. Lo que haces con él es tu regalo de regreso a Dios."*

— **Leo Buscaglia**



# Índice general

<b>Resumen</b>	<b>V</b>
<b>Abstract</b>	<b>VII</b>
<b>Índice de Figuras</b>	<b>VI</b>
<b>Índice de Tablas</b>	<b>VII</b>
<b>Índice de Fórmulas</b>	<b>IX</b>
<b>Lista de Abreviaciones</b>	<b>XI</b>
<b>1 Introducción</b>	<b>1</b>
1.1 Planteamiento del Problema . . . . .	1
1.2 Justificación . . . . .	2
1.3 Objetivos . . . . .	2
1.3.1 Objetivo General . . . . .	2
1.3.2 Objetivos Específicos . . . . .	2
1.4 Alcances y Limitaciones . . . . .	3
1.5 Estructura del Documento . . . . .	3
<b>2 Marco Teórico / Estado del Arte</b>	<b>5</b>
2.1 Antecedentes . . . . .	5
2.2 Conceptos Clave . . . . .	6
2.2.1 El Problema en Optimización . . . . .	6
2.2.1.1 Optimización Continua . . . . .	7
2.2.1.2 Optimización Discreta o Combinatoria . . . . .	8
2.2.2 Complejidad Computacional y Algorítmica . . . . .	8
2.2.3 El Problema P vs NP . . . . .	9
2.2.3.1 Clases P y NP . . . . .	10
2.2.3.2 NP-Completo y NP-Difícil . . . . .	11
2.2.4 Problemas de Optimización Combinatoria . . . . .	12
2.2.4.1 Problema del Agente Viajero (TSP) . . . . .	13
2.2.4.2 Problema del Problema Ruteo de Vehículos (VRP) . . . . .	15
2.2.4.3 Problema de Ruteo de Vehículos con Ventanas de Tiempo (VRP-TW) . . . . .	17

2.2.5 Métodos de Solución para el VRP-TW . . . . .	21
2.2.5.1 Métodos Exactos . . . . .	21
2.2.5.2 Heurísticas . . . . .	22
2.2.5.3 Metaheurísticas . . . . .	24
2.2.5.4 Algoritmos Híbridos . . . . .	30
2.3 Referencias Relevantes . . . . .	31
<b>3 Materiales y Métodos</b>	<b>33</b>
3.1 Materiales . . . . .	33
3.1.1 Datos e instancias del problema . . . . .	33
3.1.2 Software . . . . .	34
3.1.3 Hardware . . . . .	35
3.1.4 Herramientas de apoyo . . . . .	36
3.2 Métodos . . . . .	36
3.2.1 Descripción general del algoritmo híbrido . . . . .	36
3.2.2 Calibración de parámetros mediante Evolucion Diferencial (DE) . . . . .	37
3.2.3 Construcción de rutas iniciales mediante Optimización por Colonia de Hormigas (ACO) . . . . .	44
3.2.4 Refinamiento de rutas mediante Recocido Simulado (SA) . . . . .	53
<b>4 Resultados</b>	<b>67</b>
<b>5 Conclusiones</b>	<b>69</b>

# Índice de figuras

2.1	Tasa de crecimiento de varias funciones en análisis de algoritmos [40]. . . . .	9
2.2	Clasificación de los problemas computacionales en decidibles e indecidibles [37]. . . . .	10
2.3	Relación entre las clases de complejidad: P, NP, NP-completo y NP-difícil.	12
2.4	Ejemplificación de una solución al Problema del Agente Viajero (TSP). . . . .	13
2.5	Ejemplificación de una solución al Problema de Ruteo de Vehículos (VRP).	15
2.6	Ejemplificación de una solución al Problema de Ruteo de Vehículos con Ventanas de Tiempo (VRP-TW). . . . .	18
3.1	Pseudocódigo del algoritmo de Evolución Diferencial (DE) aplicado al problema VRP-TW. . . . .	44
3.2	Pseudocódigo de verificación de clientes factibles en ACO. . . . .	46
3.3	Proceso de construcción probabilística de una solución completa por una hormiga. El algoritmo maneja dinámicamente la asignación de vehículos y el avance temporal cuando no hay clientes factibles disponibles, garantizando el cumplimiento de todas las restricciones del VRP-TW. . . . .	48
3.4	Cálculo del fitness de una solución como la suma de las distancias totales recorridas por todos los vehículos. Cada ruta se evalúa sumando las distancias entre clientes consecutivos, incluyendo los trayectos desde y hacia el depósito. . . . .	49
3.5	Mecanismo de actualización de feromonas en tres fases: evaporación uniforme para favorecer exploración, depósito proporcional al fitness de las hormigas elite, y refuerzo adicional en la mejor solución global para intensificar la explotación de rutas prometedoras. . . . .	51
3.6	Algoritmo principal de ACO con integración de Simulated Annealing para la resolución del problema VRP-TW. Las hormigas construyen soluciones de forma probabilística, las mejores se optimizan con SA, y se actualiza la matriz de feromonas considerando únicamente las soluciones elite y reforzando la mejor solución global. . . . .	53
3.7	Pseudocódigo del operador Swap Intra-Ruta. . . . .	55
3.8	Pseudocódigo del operador 2-opt. . . . .	55
3.9	Pseudocódigo del operador Or-opt. . . . .	55
3.10	Pseudocódigo del operador Swap Inter-Ruta. . . . .	56
3.11	Pseudocódigo del operador Reinscripción Intra-Inter. . . . .	56
3.12	Pseudocódigo del operador 2.5-opt. . . . .	56
3.13	Pseudocódigo del operador Cross-exchange. . . . .	57

3.14 Pseudocódigo del operador Relocate-chain. . . . .	57
3.15 Algoritmo de verificación completa de restricciones para una ruta. Valida secuencialmente las ventanas de tiempo de cada cliente considerando tiempos de viaje, tiempos de servicio y posibles esperas. Además verifica que la capacidad total no exceda el límite del vehículo y que sea posible retornar al depósito dentro de su ventana de tiempo. . . . .	60
3.16 Proceso de limpieza de vehículos vacíos de la flota. Este procedimiento se ejecuta después de aplicar operadores inter-ruta que pueden transferir todos los clientes de un vehículo, periódicamente cada 20 iteraciones, y al finalizar el algoritmo SA para asegurar una solución compacta. . . . .	62
3.17 Evaluación de la función objetivo para una solución completa. Recorre todos los vehículos de la flota y suma las distancias entre clientes consecutivos en cada ruta, incluyendo los trayectos desde y hacia el depósito. . . . .	63
3.18 Algoritmo completo de Recocido Simulado para refinamiento de soluciones VRP-TW. El proceso incluye selección adaptativa de operadores según características de la instancia, criterio de aceptación de Metropolis con tolerancia numérica, limpieza condicional y periódica de vehículos vacíos, y gestión explícita de memoria. La temperatura se reduce geométricamente hasta alcanzar el umbral final, con protección contra valores extremadamente bajos. . . . .	65

# Índice de tablas

3.1	Clasificación de instancias según número de clientes . . . . .	37
3.2	Rangos de parámetros para instancias pequeñas . . . . .	39
3.3	Rangos de parámetros para instancias medianas . . . . .	39
3.4	Rangos de parámetros para instancias grandes . . . . .	40
3.5	Parámetros del Algoritmo Evolutivo Diferencial . . . . .	43



# Índice de fórmulas

2.1 Función objetivo TSP: minimizar el costo total del recorrido . . . . .	13
2.2 Restricción de salida única de cada nodo . . . . .	14
2.3 Restricción de entrada única a cada nodo . . . . .	14
2.4 Restricción de eliminación de subciclos . . . . .	14
2.5 Variables binarias que indican si la ruta va de $i$ a $j$ . . . . .	14
2.6 Función objetivo VRP: minimizar el costo total del recorrido . . . . .	16
2.7 Restricción: entrada única a cada cliente . . . . .	16
2.8 Restricción: salida única desde cada cliente . . . . .	16
2.9 Restricción: número de vehículos que regresan al depósito . . . . .	16
2.10 Restricción: número de vehículos que salen del depósito . . . . .	16
2.11 Restricción de conectividad: eliminación de subrutas . . . . .	16
2.12 Variables binarias: decisión de ruta entre nodos . . . . .	16
2.13 Función objetivo VRPTW: minimizar el costo total del recorrido . . . . .	19
2.14 Restricción: cada cliente es visitado exactamente una vez . . . . .	19
2.15 Restricción: cada vehículo inicia su ruta en el depósito . . . . .	19
2.16 Restricción: cada vehículo termina su ruta en el depósito . . . . .	19
2.17 Restricción: conservación de flujo para cada vehículo y cliente . . . . .	19
2.18 Restricción: respeto a la ventana de tiempo y orden de servicio . . . . .	19
2.19 Restricción: inicio del servicio dentro de la ventana de tiempo . . . . .	19
2.20 Restricción: capacidad máxima del vehículo . . . . .	19
2.21 Variables binarias y tiempos no negativos . . . . .	19
2.22 Vector mutante en Evolución Diferencial (DE/rand/1) . . . . .	26
2.23 Cruce binomial en Evolución Diferencial . . . . .	27
2.24 Operador de selección en Evolución Diferencial . . . . .	27
2.25 Regla de transición en ACO . . . . .	28
2.26 Distribución de Boltzmann . . . . .	29



# **Lista de Abreviaciones**

<b>ACO</b>	Ant Colony Optimization ( <i>Optimización por Colonias de Hormigas</i> )
<b>DE</b>	Differential Evolution ( <i>Evolución Diferencial</i> )
<b>EA</b>	Evolutionary Algorithm ( <i>Algoritmo Evolutivo</i> )
<b>GA</b>	Genetic Algorithm ( <i>Algoritmo Genético</i> )
<b>NP</b>	Nondeterministic Polynomial Time ( <i>Tiempo Polinomial No Determinista</i> )
<b>P</b>	Polynomial Time ( <i>Tiempo Polinomial</i> )
<b>SA</b>	Simulated Annealing ( <i>Recocido Simulado</i> )
<b>TB</b>	Tabu Search ( <i>Búsqueda Tabú</i> )
<b>TSP</b>	Travelling Salesman Problem ( <i>Problema del Viajante</i> )
<b>VRP</b>	Vehicle Routing Problem ( <i>Problema de Ruteo de Vehículos</i> )
<b>VRPTW</b>	Vehicle Routing Problem with Time Windows ( <i>Problema de Ruteo de Vehículos con Ventanas de Tiempo</i> )



# Capítulo 1

## Introducción

### 1.1. Planteamiento del Problema

Dentro de la investigación operativa y la logística, modelos clásicos como el Problema del Agente Viajero (TSP, *Traveling Salesman Problem*) y el Problema de Ruteo de Vehículos (VRP, *Vehicle Routing Problem*) constituyen la base para entender y abordar problemáticas logísticas reales. Sin embargo, en entornos modernos donde existen restricciones temporales estrictas para la atención de clientes, se requieren modelos más complejos, como el Problema de Ruteo de Vehículos con Ventanas de Tiempo (VRP-TW, *Vehicle Routing Problem with Time Windows*).

El VRP-TW consiste en planificar rutas para una flota de vehículos de manera que se minimice el costo total (distancia, tiempo o recursos), asegurando que cada cliente sea atendido dentro de un intervalo de tiempo predefinido y sin violar las capacidades de los vehículos. Este problema pertenece a la clase NP-difícil y presenta una alta complejidad combinatoria, lo que hace inviable su resolución mediante algoritmos exactos para instancias de gran tamaño.

En este contexto, las metaheurísticas se presentan como una alternativa eficaz para obtener soluciones de buena calidad en tiempos razonables. Entre ellas, la Optimización por Colonia de Hormigas (ACO, *Ant Colony Optimization*) ha demostrado ser útil en la construcción de rutas iniciales, mientras que el Recocido Simulado (SA, *Simulated Annealing*), apoyado en diversas heurísticas locales, permite mejorar las soluciones generadas. No obstante, ambos enfoques pueden beneficiarse de mecanismos adicionales de exploración y explotación del espacio de búsqueda; por ello, se propone incorporar el Algoritmo Evolutivo Diferencial (DE, *Differential Evolution*) como componente global de refinamiento.

El problema a resolver, por tanto, se centra en diseñar e implementar un algoritmo híbrido que combine de manera coordinada y efectiva ACO, SA y DE. En este esquema, el DE se emplea para calibrar los parámetros de los otros métodos, permitiendo abordar

el VRP-TW a partir de modelos más simples como el TSP y el VRP, considerando tanto la calidad de la solución como su robustez y eficiencia computacional.

## 1.2. Justificación

El VRP-TW es un problema fundamental en la logística moderna, con aplicaciones directas en la optimización de rutas de distribución y transporte, lo que impacta significativamente en la reducción de costos operativos y la mejora del servicio al cliente. Sin embargo, debido a su complejidad combinatoria y a su pertenencia a la clase NP-difícil, las técnicas exactas resultan impracticables para resolver instancias de tamaño realista.

Las soluciones basadas en metaheurísticas han demostrado ser una alternativa eficiente; sin embargo, la mayoría de los enfoques existentes se centran en una sola técnica, lo que limita la exploración y explotación del espacio de soluciones. La integración de métodos híbridos, como la combinación de ACO, SA y DE, puede potenciar las capacidades de búsqueda y refinamiento, mejorando la calidad y robustez de las soluciones obtenidas.

Este trabajo se justifica en la necesidad de desarrollar un método que aporte mayor eficiencia y calidad para resolver el VRP-TW, utilizando instancias clásicas ampliamente reconocidas en la literatura, como son las propuestas por Solomon, y así contribuir al avance de las técnicas metaheurísticas aplicadas en problemas logísticos complejos.

## 1.3. Objetivos

### 1.3.1. Objetivo General

Diseñar, implementar, probar y evaluar un algoritmo híbrido que combine la ACO, el SA con múltiples heurísticas locales y el DE, con el objetivo de resolver de manera eficiente el VRP-TW.

### 1.3.2. Objetivos Específicos

1. Estudiar y analizar los problemas TSP, VRP y VRP-TW para identificar sus similitudes estructurales y diferencias, estableciendo así una base conceptual que permita abordar de forma progresiva el problema principal VRP-TW.
2. Diseñar un algoritmo híbrido que combine el ACO para la construcción de soluciones iniciales y el SA para la mejora de dichas soluciones, integrando el DE como un componente de autoajuste de parámetros o como mecanismo para generar

configuraciones óptimas que guíen el comportamiento de las otras metaheurísticas.

3. Evaluar el rendimiento del algoritmo híbrido en términos de calidad de la solución y eficiencia computacional, utilizando instancias de prueba estándar aceptadas en la literatura, y comparando los resultados con enfoques clásicos y metaheurísticos individuales.

## 1.4. Alcances y Limitaciones

### Alcances

- El trabajo se centra en la resolución del problema VRP-TW.
- Se desarrolla un enfoque híbrido que integra tres técnicas metaheurísticas: ACO, SA y un DE, aplicadas de manera complementaria.
- El enfoque es probado con instancias clásicas propuestas por Solomon, ampliamente reconocidas en la literatura, lo que permite una validación objetiva y reproducible de los resultados obtenidos.
- Se analizan métricas relevantes como la calidad de la solución, el tiempo de ejecución y la estabilidad del algoritmo ante múltiples ejecuciones.

### Limitaciones

- El enfoque propuesto está diseñado para instancias estáticas de los problemas de ruteo; no se contempla el tratamiento de versiones dinámicas o en tiempo real.
- La calidad de los resultados depende en gran medida de la correcta calibración de los parámetros de ACO y SA; sin embargo, esta calibración no se realiza manualmente, sino que es llevada a cabo automáticamente por el DE.
- No se consideran restricciones adicionales como múltiples depósitos, flotas heterogéneas o prioridades diferenciadas entre clientes.
- Las pruebas están limitadas a conjuntos de datos artificiales y benchmarks académicos, sin validación en entornos productivos reales.
- Al tratarse de métodos metaheurísticos, no se garantiza la obtención del óptimo global, sino soluciones cercanas al óptimo dentro de un tiempo razonable.

## 1.5. Estructura del Documento



# Capítulo 2

## Marco Teórico / Estado del Arte

### 2.1. Antecedentes

El estudio de problemas de optimización en rutas se remonta al desarrollo del Problema del Agente Viajero (TSP), formulado a principios del siglo XX, cuyo objetivo es encontrar la ruta de costo mínimo que recorra un conjunto de ciudades exactamente una vez, regresando al punto de partida. A lo largo de las décadas, este problema se ha consolidado como uno de los pilares de la optimización combinatoria [43].

Más adelante, se propuso el Problema de Ruteo de Vehículos (VRP) como una generalización del TSP, considerando una flota de vehículos que deben atender a múltiples clientes desde un depósito central. Este problema fue introducido por primera vez por Dantzig y Ramser en 1959 para optimizar rutas de distribución de gasolina [13].

Durante la década de 1970, el interés por resolver variantes del VRP aumentó significativamente, y surgieron métodos heurísticos como el algoritmo de Clarke y Wright (1964), que se convirtió en un referente para aproximaciones iniciales [9].

Entre la década de 1970 y la de 1980, se desarrollaron versiones especializadas como el *Fleet Routing Problem*, sistemas *Dial-a-Bus* y el diseño de redes de transporte, así como enfoques probabilísticos y la incorporación de incertidumbre [21, 26, 32].

Dentro de estas variantes, el Problema de Ruteo de Vehículos con Ventanas de Tiempo (VRP-TW) surgió como una extensión crítica que incorpora restricciones temporales para reflejar mejor las condiciones reales de operación, tales como horarios de entrega y recogida. Este problema fue estudiado ampliamente a partir del trabajo de Solomon en 1987, quien introdujo instancias benchmark y algoritmos heurísticos clásicos [48].

Durante la década de 1990, se desarrollaron enfoques exactos basados en programación entera y técnicas como *branch-and-bound*, *branch-and-cut* y *column generation*, aplicados con éxito a instancias pequeñas [16, 11]. Sin embargo, la elevada complejidad

dad del VRP-TW motivó el uso creciente de heurísticas y metaheurísticas, tales como:

- **Algoritmos Genéticos (GA)**, inspirados en la evolución natural, propuestos por Holland en 1975, que utilizan operadores genéticos para evolucionar una población de soluciones [28].
- **Algoritmo de Colonias de Hormigas (ACO)**, propuesto inicialmente por Dorigo en 1992, basado en el comportamiento colectivo de las hormigas para resolver problemas combinatorios [18].
- **Recocido Simulado (SA)**, inspirado en el proceso físico de enfriamiento de metales, introducido por Kirkpatrick, Gelatt y Vecchi en 1983, que permite escapar de óptimos locales mediante la aceptación controlada de soluciones peores [31].
- **Búsqueda Tabú (TB)**, una técnica heurística que explora iterativamente soluciones vecinas para mejorar la calidad de la solución, evitando ciclos mediante la prohibición temporal de movimientos [1].

Estas metaheurísticas han demostrado ser eficaces para obtener soluciones cercanas al óptimo en tiempos razonables [21].

En las últimas dos décadas, la investigación se ha enfocado en algoritmos híbridos, técnicas de aprendizaje automático aplicadas al VRP, optimización robusta y modelos dinámicos, buscando mejorar la eficiencia y adaptabilidad de las soluciones en contextos logísticos cada vez más complejos y cambiantes [53].

El presente trabajo se inscribe en esta línea de investigación, proponiendo un modelo basado en un algoritmo híbrido con metaheurísticas para abordar el VRP-TW, considerando específicamente restricciones operativas como la capacidad limitada de los vehículos y los intervalos estrictos de tiempo de servicio asignados a cada cliente. Con el objetivo de mejorar la calidad y eficiencia de las rutas generadas, se implementaron tres enfoques metaheurísticos: el Algoritmo Evolutivo Diferencial (DE), el Algoritmo de Optimización por Colonias de Hormigas (ACO) y el Recocido Simulado (SA).

## 2.2. Conceptos Clave

### 2.2.1. El Problema en Optimización

Los problemas de optimización constituyen una herramienta fundamental en ciencias computacionales, ingeniería, logística y muchas otras disciplinas. En el contexto de este trabajo de investigación, resultan especialmente relevantes porque permiten modelar situaciones en las que se busca obtener el mejor resultado posible bajo ciertas restricciones, como minimizar costos, distancias o tiempos.

De acuerdo con [15], los problemas de optimización se dividen de manera natural en dos categorías: problemas con variables continuas y problemas con variables discretas. A estos últimos se les conoce como problemas de optimización combinatoria.

La función  $f : S \rightarrow \mathbb{R}$ , donde el conjunto  $S$  es un subconjunto de  $\mathbb{R}^n$ , se denomina función objetivo, función de costo beneficio, mientras que el conjunto  $S$  se identifica como el conjunto factible o el conjunto de soluciones posibles [15].

Como definición formal, se puede decir que el problema general de optimización consiste en encontrar un elemento  $x \in S$  que optimice la función objetivo. En el caso de un problema de minimización, se expresa de la siguiente manera:

$$\min_{x \in S} f(x).$$

Por su parte, en un problema de maximización, se utiliza la notación:

$$\max_{x \in S} f(x).$$

[15]

### 2.2.1.1. Optimización Continua

En la optimización continua para problemas de minimización, el objetivo es encontrar un punto  $x_{opt}$  dentro del conjunto factible  $S$  tal que el valor de la función objetivo en ese punto sea menor o igual que el valor en cualquier otro punto del conjunto, es decir,

$$f(x_{opt}) \leq f(x), \quad \forall x \in S.$$

Para problemas de maximización, se busca un punto  $x_{opt}$  en  $S$  donde la función objetivo tome un valor mayor o igual al de cualquier otro punto factible, esto es,

$$f(x_{opt}) \geq f(x), \quad \forall x \in S.$$

A este punto se le denomina solución óptima global, y el valor correspondiente  $f(x_{opt})$  se conoce como costo óptimo. Además, el conjunto de todas las soluciones óptimas se representa por  $S_{opt}$  [15].

### 2.2.1.2. Optimización Discreta o Combinatoria

Una instancia de un problema de optimización combinatoria puede representarse mediante una pareja  $(S, f)$ , donde  $S$  es un conjunto finito que contiene todas las soluciones posibles, y  $f$  es una función real, denominada función objetivo o función costo, definida como

$$f : S \rightarrow \mathbb{R}.$$

Para problemas de minimización, se busca una solución  $i_{opt} \in S$  tal que

$$f(i_{opt}) \leq f(i), \quad \forall i \in S,$$

mientras que para problemas de maximización, se requiere que

$$f(i_{opt}) \geq f(i), \quad \forall i \in S.$$

La solución  $i_{opt}$  se denomina solución óptima global, y el valor  $f(i_{opt})$  representa el costo óptimo. El conjunto de todas las soluciones óptimas se denota como  $S_{opt}$ . Un problema de optimización combinatoria se define entonces como el conjunto de todas sus instancias [15].

Comprender esta distinción es esencial para abordar el problema central de este trabajo VRP-TW, el cual se enmarca dentro de la optimización combinatoria. En este tipo de problemas, el número de soluciones posibles es finito, y el desafío consiste en identificar cuál de ellas representa la mejor opción según un criterio específico.

Esta base teórica permite introducir los problemas clásicos de enrutamiento, como el TSP, el VRP y sus variantes, como el VRP-TW, los cuales serán analizados más adelante por su relación directa con la planificación eficiente de rutas en contextos reales.

### 2.2.2. Complejidad Computacional y Algorítmica

El estudio de la complejidad en ciencias computacionales busca comprender qué tan difícil es resolver un problema, tanto desde su estructura teórica como desde los recursos requeridos para su solución. Esta dificultad se aborda desde dos enfoques: la complejidad computacional, que clasifica los problemas según el crecimiento de recursos necesarios en función del tamaño de la entrada, y la complejidad algorítmica, que evalúa el desempeño de algoritmos específicos al aplicarse sobre dichas entradas [24, 43].

De forma simplificada, la complejidad computacional se refiere al análisis general del problema sin importar el algoritmo, mientras que la algorítmica se enfoca en el

tiempo y espacio consumido por un algoritmo concreto [37].

Esta distinción es clave al abordar problemas de optimización combinatoria como el TSP, el VRP y su variante principal en este trabajo: el VRP-TW, que presentan una explosión combinatoria (crecimiento factorial) de soluciones conforme crece el número de clientes.

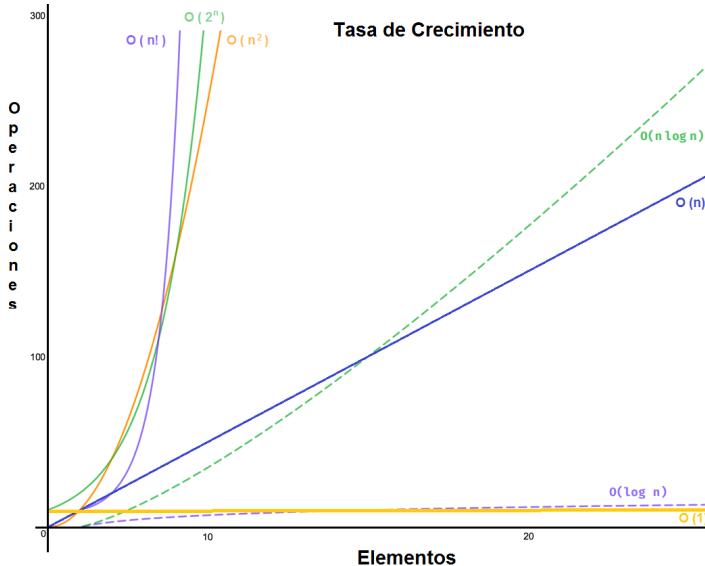


Figura 2.1: Tasa de crecimiento de varias funciones en análisis de algoritmos [40].

En la Figura 2.1 se observa cómo las funciones constantes  $\mathcal{O}(1)$  y logarítmicas  $\mathcal{O}(\log n)$  crecen lentamente, incluso para entradas de gran tamaño. Las funciones polinomiales, como  $\mathcal{O}(n)$  y  $\mathcal{O}(n \log n)$ , presentan un crecimiento más acelerado pero aún manejable. En contraste, las funciones exponenciales  $\mathcal{O}(2^n)$  y factoriales  $\mathcal{O}(n!)$  crecen de manera extremadamente rápida, lo que vuelve intratables los problemas asociados a medida que aumenta el tamaño de la entrada. Por esta razón, problemas como el TSP, VRP y VRP-TW se consideran de alta complejidad computacional, ya que su espacio de búsqueda crece factorialmente.

### 2.2.3. El Problema P vs NP

En ciencias de la computación, todos los problemas pueden clasificarse en dos grupos principales: problemas decidibles y problemas indecidibles. Un problema es *indecidible* cuando no existe ningún algoritmo que pueda resolverlo, incluso si se dispone de tiempo y recursos ilimitados. En consecuencia, no es posible determinar si dicho problema terminará su ejecución. Por contraste, un problema es *decidible* cuando existe (o podría existir) un algoritmo capaz de resolverlo [37].

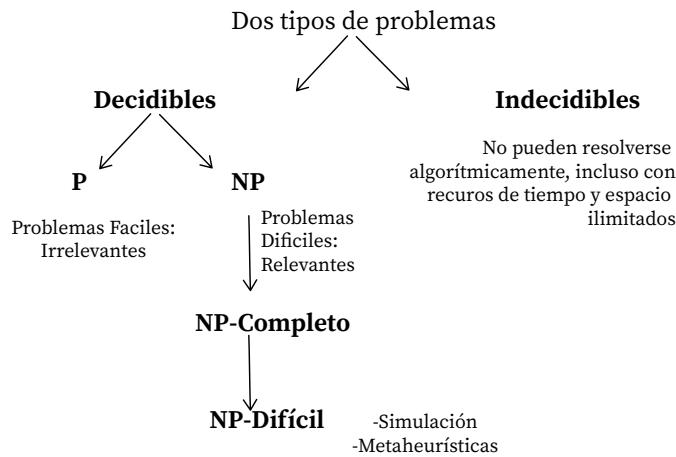


Figura 2.2: Clasificación de los problemas computacionales en decidibles e indecidibles [37].

En este trabajo, los problemas abordados —el TSP, el VRP y el VRP-TW— son considerados problemas decidibles. A continuación, se presenta una clasificación más detallada dentro de esta categoría, basada en la teoría de la complejidad computacional.

### 2.2.3.1. Clases P y NP

Dentro de los problemas decidibles, las clases P y NP juegan un papel fundamental [37].

#### Clase P

Un problema pertenece a la clase P si puede resolverse mediante un algoritmo cuyo tiempo de ejecución está acotado por una función polinomial respecto al tamaño de la entrada. Formalmente, existe un polinomio  $p(n)$  y constantes  $\alpha, n_0 > 0$  tales que:

$$D(n) \leq \alpha \cdot p(n) \quad \forall n \geq n_0,$$

lo que se expresa en notación asintótica como

$$D = O(n^k).$$

Aunque se considera que algoritmos con tiempo polinomial son “eficientes”, en la práctica el grado del polinomio importa, ya que un polinomio de grado muy alto puede

ser inviable. Un ejemplo clásico de problema en P es la multiplicación de números naturales [22].

### Clase NP

Un problema pertenece a la clase NP si, dado un candidato a solución certificado, es posible verificar su validez en tiempo polinomial. Es importante notar que esto no implica necesariamente que exista un algoritmo eficiente para encontrar la solución, sino que su validación es eficiente. Un ejemplo típico es el TSP: si se proporciona una ruta candidata, es fácil comprobar en tiempo polinomial la longitud total de la ruta para verificar si cumple cierta condición [22].

### La gran incógnita: $P = NP?$

Una de las preguntas abiertas más importantes en ciencias de la computación es si  $P = NP$  o  $P \neq NP$ , es decir, si todo problema cuya solución puede verificarse rápidamente también puede resolverse rápidamente. Esta cuestión tiene profundas implicaciones teóricas y prácticas [37].

#### 2.2.3.2. NP-Completo y NP-Difícil

Dentro de la clase NP existe una subclase de problemas llamados NP-completos, que son los problemas más difíciles en NP. Si se encontrara un algoritmo eficiente para cualquiera de ellos, entonces todos los problemas en NP podrían resolverse eficientemente [37].

Por otro lado, los problemas NP-difíciles pueden ser incluso más complejos que los NP-completos, ya que no requieren pertenecer a NP (es decir, su solución no tiene por qué ser verificable en tiempo polinomial), pero son al menos tan difíciles como los problemas NP-completos. Pueden incluir problemas de decisión, búsqueda u optimización [37].

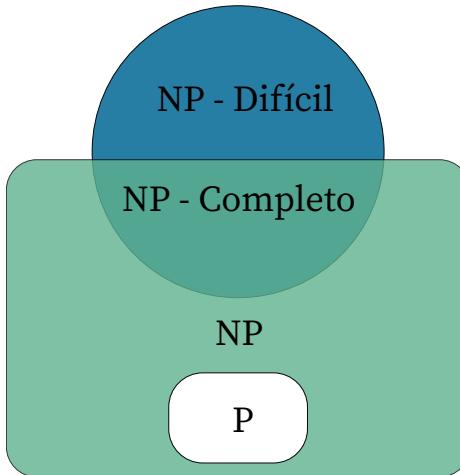


Figura 2.3: Relación entre las clases de complejidad: P, NP, NP-completo y NP-difícil.

El crecimiento exponencial del tiempo requerido para resolver problemas como el TSP, el VRP y el VRP-TW los ubica dentro de la clase NP. Estos problemas, aunque decidibles, presentan una complejidad tan elevada que no se conoce ningún algoritmo exacto que los resuelva eficientemente para instancias grandes. Por ello, más adelante se analizará su clasificación específica dentro de NP, así como las estrategias metaheurísticas empleadas para aproximar soluciones en tiempos razonables.

#### 2.2.4. Problemas de Optimización Combinatoria

De manera complementaria a la definición formal presentada en la sección 2.2.1.2, Papadimitriou y Steiglitz [44] definen la optimización combinatoria como el estudio de problemas de optimización en los que el conjunto de soluciones factibles es discreto, o puede discretizarse mediante un proceso de enumeración, y donde se busca una solución que optimice una función objetivo definida sobre dicho conjunto.

Los problemas de optimización combinatoria constituyen una clase amplia de modelos aplicables a diversas áreas, especialmente en la planificación y el enrutamiento de vehículos. Dentro de este campo destacan el TSP, que representa la forma más básica de optimización de rutas, y sus extensiones, que incorporan restricciones adicionales propias de escenarios reales, como la capacidad limitada de los vehículos o las ventanas de tiempo en las que deben realizarse las entregas.

En este contexto, el VRP y el VRP-TW se han consolidado como dos de los desafíos más estudiados, debido a su relevancia práctica y su elevada complejidad computacional. A continuación, se describen estos problemas en detalle.

### 2.2.4.1. Problema del Agente Viajero (TSP)

El Problema del Agente Viajero, conocido por sus siglas en inglés como TSP (*Travelling Salesman Problem*), es uno de los problemas más reconocidos y complejos dentro de las ciencias computacionales. Ha sido estudiado desde diversas ramas de la ingeniería y por múltiples motivos. Su aplicación principal consiste en determinar rutas desde diferentes enfoques, ya sea en procesos que requieren una secuencia específica o en operaciones logísticas relacionadas con el transporte, con el objetivo de encontrar la ruta óptima considerando criterios de minimización de distancia o de costo [35].

En la Figura 2.4 se presenta una representación gráfica de una posible solución al TSP. En ella, un vehículo parte de un nodo inicial(depósito), recorre una serie de ciudades o puntos de entrega siguiendo una secuencia determinada, y regresa al punto de partida, completando así un circuito cerrado que minimiza la distancia total recorrida.

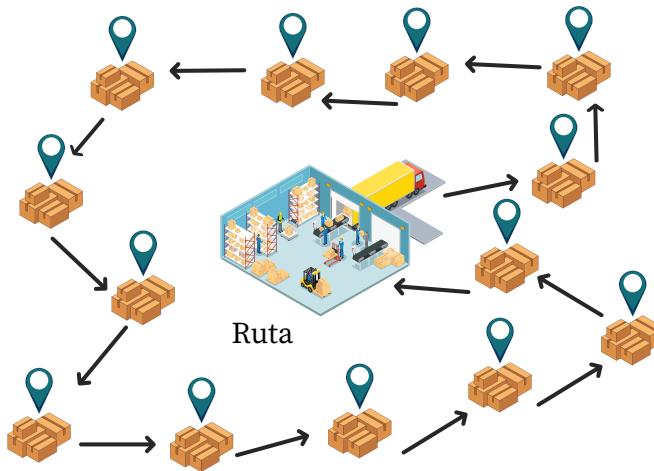


Figura 2.4: Ejemplificación de una solución al Problema del Agente Viajero (TSP).

Según [51], el TSP se define sobre un grafo  $G = [N, A, C]$ , donde  $N$  es el conjunto de nodos,  $A$  es el conjunto de arcos y  $C = [c_{ij}]$  es la matriz de costos (distancias) entre nodos  $i$  y  $j$ . El objetivo es encontrar un ciclo hamiltoniano de costo mínimo que recorra todos los nodos una sola vez y regrese al punto de partida.

#### Modelo Matemático del Problema del Agente Viajero (TSP)

$$\min \sum_{i=1}^N \sum_{j=1}^N c_{ij} x_{ij} \quad (2.1)$$

sujeto a:

$$\sum_{j=1}^N x_{ij} = 1, \quad \forall i = 1, \dots, N \quad (2.2)$$

$$\sum_{i=1}^N x_{ij} = 1, \quad \forall j = 1, \dots, N \quad (2.3)$$

$$u_i - u_j + Nx_{ij} \leq N - 1, \quad \forall i, j = 2, \dots, N, \quad i \neq j \quad (2.4)$$

$$x_{ij} \in \{0, 1\}, \quad \forall i, j = 1, \dots, N \quad (2.5)$$

donde:

- $x_{ij} = 1$  si la ruta va de  $i$  a  $j$ , y 0 en caso contrario;
- $u_i$  son variables auxiliares usadas para eliminar subciclos, con  $i = 2, \dots, N$  [51].

La función objetivo (2.1) representa la suma total de los costos (o distancias) de los arcos que conforman el recorrido. La restricción en (2.2) garantiza que, al salir de cada nodo, sólo se dirija a un único nodo siguiente. La restricción en (2.3) aseguran que a cada nodo únicamente se llegue una vez. La restricción (2.4) es fundamental, ya que previene la formación de subciclos que no incluyan a todos los nodos, asegurando así un recorrido único y completo. Finalmente, la restricción (2.5) indica que las variables  $x_{ij}$  son binarias, es decir, sólo pueden tomar los valores 0 o 1, lo que permite representar la inclusión o exclusión de una ruta entre nodos [51].

### Complejidad del TSP

Como explica [44], el TSP es NP-difícil, y su versión de decisión es NP-completa. Esto implica que la búsqueda de un algoritmo eficiente para resolverlo representa un desafío fundamental en la teoría de la computación, pues encontrar una solución polinomial para el TSP permitiría resolver eficientemente toda la clase *NP*. Esta dificultad justifica la utilización de algoritmos heurísticos y metaheurísticos, dado que las técnicas exactas sólo resultan prácticas para instancias

El TSP ha sido tradicionalmente considerado como el punto de partida para el estudio de problemas más complejos de enrutamiento, como el VRP y sus variantes, entre ellas el VRP-TW. Su análisis proporciona una base conceptual sólida para comprender

aspectos fundamentales como la construcción de rutas, la minimización de costos y el manejo de restricciones operativas. Estudiar el TSP permite introducir de forma gradual nuevas variables y condiciones, lo que lo convierte en un modelo introductorio ideal para el desarrollo de enfoques logísticos más realistas.

#### 2.2.4.2. Problema del Problema Ruteo de Vehículos (VRP)

El Problema del Ruteo de Vehículos, conocido por sus siglas en inglés como VRP (*Vehicle Routing Problem*), puede interpretarse como la convergencia de dos problemas clásicos de optimización combinatoria: el Problema del Agente Viajero (TSP) mencionado anteriormente en la sección 2.2.4.1, en el que se asume una capacidad infinita de los vehículos, y el Problema de Empaqueamiento en Compartimentos (BPP) [14]. En su formulación más básica, el VRP considera un depósito central desde donde parte una flota de vehículos para atender a un conjunto de clientes distribuidos geográficamente. Cada vehículo debe visitar un subconjunto de clientes exactamente una vez, respetando su capacidad de carga y satisfaciendo la demanda de los clientes asignados. El objetivo es minimizar el costo total asociado a las rutas, las cuales inician y finalizan en el depósito [42].

La Figura 2.5 muestra una representación gráfica de una solución al VRP. En este ejemplo, varios vehículos parten desde un depósito central para atender a distintos clientes distribuidos geográficamente. Cada ruta ha sido asignada de manera que se respeten las restricciones de capacidad de carga de los vehículos, y se minimice la distancia total recorrida, cumpliendo así con los objetivos del problema.

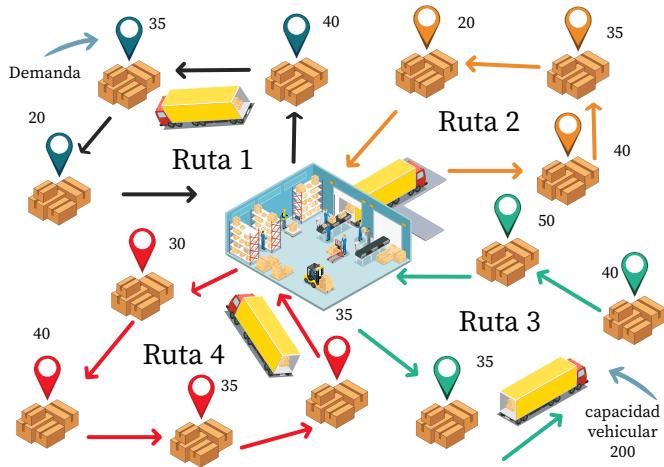


Figura 2.5: Ejemplificación de una solución al Problema de Ruteo de Vehículos (VRP).

Según [53], el VRP se define sobre un grafo  $G = (V, A)$ , donde  $V = \{0, 1, \dots, n\}$  representa el conjunto de nodos, con el nodo 0 correspondiente al depósito y los nodos

restantes a los clientes,  $A$  es el conjunto de arcos, y  $C = [c_{ij}]$  es la matriz de costos asociados a los arcos  $(i, j)$ . El objetivo es determinar un conjunto de rutas de costo mínimo para una flota de vehículos idénticos, de modo que: cada cliente sea visitado exactamente una vez por un solo vehículo, cada ruta comience y termine en el depósito, y la demanda total de los clientes en cada ruta no exceda la capacidad del vehículo.

### Modelo Matemático del Problema de Ruteo de Vehículos (VRP)

$$\min \sum_{i \in V} \sum_{j \in V} c_{ij} x_{ij} \quad (2.6)$$

sujeto a:

$$\sum_{i \in V} x_{ij} = 1, \quad \forall j \in V \setminus \{0\} \quad (2.7)$$

$$\sum_{j \in V} x_{ij} = 1, \quad \forall i \in V \setminus \{0\} \quad (2.8)$$

$$\sum_{i \in V} x_{i0} = K \quad (2.9)$$

$$\sum_{j \in V} x_{0j} = K \quad (2.10)$$

$$\sum_{i \in S} \sum_{j \in S} x_{ij} \geq r(S), \quad \forall S \subseteq V \setminus \{0\}, S \neq \emptyset \quad (2.11)$$

$$x_{ij} \in \{0, 1\}, \quad \forall i, j \in V \quad (2.12)$$

donde:

- $V$  es el conjunto de nodos (clientes más depósito), con el depósito representado por el nodo 0;

- $c_{ij}$  es el costo (o distancia) de viajar del nodo  $i$  al nodo  $j$ ;
- $x_{ij} = 1$  si se viaja directamente del nodo  $i$  al nodo  $j$ , 0 en otro caso;
- $K$  es el número de vehículos disponibles en el depósito;
- $S$  es un subconjunto de clientes, y  $r(S)$  representa el número mínimo de vehículos necesarios para satisfacer la demanda total de los clientes en  $S$  [53].

La función objetivo (2.6) minimiza el costo total de todas las rutas. Las restricciones (2.7) y (2.8) aseguran que cada cliente sea visitado exactamente una vez. Las restricciones (2.9) y (2.10) controlan el número de vehículos que entran y salen del depósito. La restricción (2.11) evita la formación de subrutas que no incluyan al depósito, garantizando conectividad. Finalmente, la restricción (2.12) indica que las variables de decisión son binarias [53].

### Complejidad del VRP

Como señalan Laporte y Nobert [34], el VRP es NP-difícil, incluso en su versión más simple con un solo vehículo (es decir, el TSP). Esto significa que no se conoce ningún algoritmo que lo resuelva de forma exacta en tiempo polinomial, y su complejidad crece exponencialmente con el número de clientes. Por ello, la mayoría de las instancias prácticas requieren el uso de técnicas heurísticas o metaheurísticas, ya que los métodos exactos son computacionalmente inviables para problemas de tamaño medio o grande.

El VRP representa la extensión natural del TSP hacia escenarios logísticos más realistas y complejos, incorporando restricciones operativas que reflejan las limitaciones del mundo real. Su estudio es fundamental para comprender cómo las restricciones de capacidad, múltiples vehículos y la gestión de flotas impactan en la optimización de rutas. El VRP sirve como base conceptual para abordar problemas logísticos avanzados como la distribución con ventanas de tiempo, el ruteo con múltiples depósitos y la optimización de última milla en el comercio electrónico. Analizar el VRP permite desarrollar una comprensión profunda de los compromisos inherentes entre la minimización de costos, la utilización eficiente de recursos y la satisfacción de restricciones operativas, estableciendo los fundamentos teóricos necesarios para abordar las complejidades de los sistemas de distribución modernos.

#### 2.2.4.3. Problema de Ruteo de Vehículos con Ventanas de Tiempo (VRP-TW)

El Problema de Ruteo de Vehículos con Ventanas de Tiempo, conocido por sus siglas en inglés como VRP-TW (*Vehicle Routing Problem with Time Windows*), es una extensión del clásico VRP mencionado anteriormente en la sección 2.2.4.2 que incorpora

restricciones temporales. En este problema, además de satisfacer las demandas específicas de cada cliente, es imprescindible que las visitas se realicen dentro de una franja horaria determinada para cada uno de ellos [53]. Específicamente, el servicio a un cliente debe comenzar en un instante mayor o igual al inicio de su ventana de tiempo, y el arribo al punto de servicio no puede exceder el límite superior de dicha ventana. Asimismo, en caso de que un vehículo llegue antes del inicio de la ventana, deberá esperar hasta el momento indicado para poder atender al cliente [42].

En este contexto, el objetivo fundamental del VRP-TW es determinar un conjunto de rutas para una flota de vehículos homogéneos que partan y regresen al depósito, visiten cada cliente exactamente una vez dentro de su respectiva ventana temporal, y sin exceder la capacidad de los vehículos. Todo esto debe lograrse minimizando el costo total del recorrido, usualmente expresado en términos de distancia o tiempo [48].

En la Figura 2.6 se muestra una solución típica al VRP-TW. En este caso, las rutas de múltiples vehículos parten desde un único depósito, deben cumplir con las capacidades máximas y además respetar las ventanas temporales de atención en cada cliente, optimizando el recorrido total.

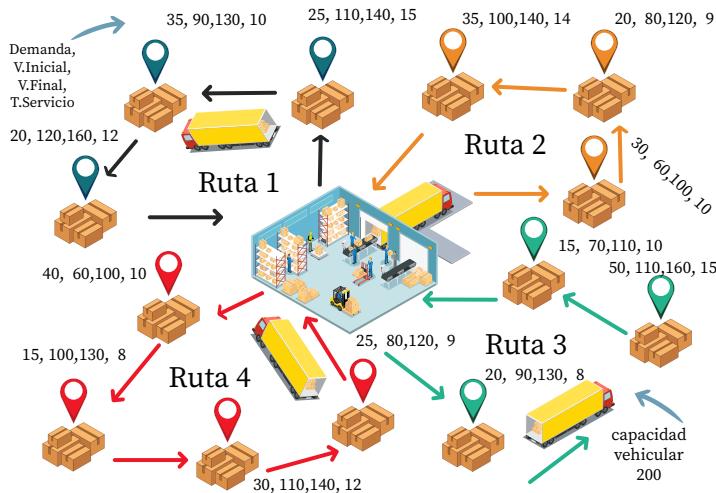


Figura 2.6: Ejemplificación de una solución al Problema de Ruteo de Vehículos con Ventanas de Tiempo (VRP-TW).

Según [53], el VRP-TW se define sobre un grafo dirigido  $G = (V, A)$ , donde  $V = \{0, 1, \dots, n, n + 1\}$  representa el conjunto de nodos, siendo 0 y  $n + 1$  el depósito (nodos origen y destino, respectivamente), y  $N = V \setminus \{0, n + 1\}$  el conjunto de clientes. Cada arco  $(i, j) \in A$  tiene asociado un costo  $c_{ij}$  y un tiempo de viaje  $t_{ij}$ . Además, cada cliente  $i$  tiene una demanda  $d_i$ , un tiempo de servicio  $s_i$ , y una ventana de tiempo  $[a_i, b_i]$  dentro de la cual debe iniciarse su servicio.

**Modelo Matemático del Problema de Ruteo de Vehículos con Ventanas de Tiempo(VRP-TW)**

$$\min \sum_{k \in K} \sum_{i \in V} \sum_{j \in V} c_{ij} x_{ijk} \quad (2.13)$$

sujeto a:

$$\sum_{k \in K} \sum_{j \in V} x_{ijk} = 1, \quad \forall i \in N \quad (2.14)$$

$$\sum_{j \in V} x_{0jk} = 1, \quad \forall k \in K \quad (2.15)$$

$$\sum_{i \in V} x_{i,n+1,k} = 1, \quad \forall k \in K \quad (2.16)$$

$$\sum_{j \in V} x_{ijk} - \sum_{j \in V} x_{jik} = 0, \quad \forall k \in K, \forall i \in N \quad (2.17)$$

$$w_{jk} \geq w_{ik} + s_i + t_{ij} - M(1 - x_{ijk}), \quad \forall k \in K, \forall i, j \in V \quad (2.18)$$

$$a_i \leq w_{ik} \leq b_i, \quad \forall k \in K, \forall i \in V \quad (2.19)$$

$$\sum_{i \in V} \sum_{j \in V} d_i x_{ijk} \leq C, \quad \forall k \in K \quad (2.20)$$

$$x_{ijk} \in \{0, 1\}, \quad w_{ik} \geq 0, \quad \forall k \in K, \forall i, j \in V \quad (2.21)$$

donde:

- $V = \{0, 1, \dots, n, n + 1\}$  es el conjunto de nodos, donde 0 y  $n + 1$  representan el depósito;
- $N = V \setminus \{0, n + 1\}$  es el conjunto de clientes;
- $K$  es el conjunto de vehículos disponibles;
- $A$  es el conjunto de arcos permitidos entre nodos;
- $c_{ij}$  es el costo o distancia de viajar del nodo  $i$  al nodo  $j$ ;
- $x_{ijk}$  es variable binaria que indica si el vehículo  $k$  viaja directamente del nodo  $i$  al nodo  $j$ ;
- $w_{ik}$  es el tiempo de inicio del servicio del vehículo  $k$  en el nodo  $i$ ;
- $s_i$  es el tiempo de servicio requerido en el nodo  $i$ ;
- $t_{ij}$  es el tiempo de viaje entre los nodos  $i$  y  $j$ ;
- $[a_i, b_i]$  es la ventana de tiempo para el nodo  $i$ ;
- $d_i$  es la demanda del cliente  $i$ ;
- $C$  es la capacidad máxima de cada vehículo;
- $M$  es un número grande usado para activar o desactivar restricciones condicionales.

La función objetivo (2.13) minimiza el costo total de todas las rutas. Las restricciones (2.14) garantizan que cada cliente sea visitado exactamente una vez. Las restricciones (2.15) y (2.16) aseguran que cada vehículo comience y termine su ruta en el depósito. La restricción (2.17) mantiene el balance de flujo para cada vehículo y cliente. La restricción (2.18) impone el orden y respeto a las ventanas temporales mediante los tiempos de servicio y de viaje, con ayuda del parámetro  $M$ . La restricción (2.19) asegura que el servicio inicie dentro de la ventana asignada. La restricción (2.20) limita la capacidad máxima que puede transportar cada vehículo. Finalmente, la restricción (2.21) define los dominios de las variables de decisión [53].

### Complejidad del VRP-TW

Sabemos que la complejidad del VRP es NP-difícil, como se señala en la Subsección 2.2.4.2 y en [34]. El VRP-TW es una generalización del VRP clásico, y también pertenece a la clase de problemas NP-difícil. De hecho, incluso si se ignoran las ventanas de tiempo, el problema sigue siendo difícil de resolver. Al incorporar restricciones temporales, el VRP-TW se vuelve aún más complejo, tanto desde el punto de vista computacional como algorítmico. Según Bräysy y Gendreau [8] y Toth y Vigo [53], las ventanas de tiempo reducen significativamente el conjunto de soluciones viables, aumentando así la dificultad del problema.

El VRP-TW representa una evolución del VRP, el cual a su vez deriva del TSP. Estos problemas están orientados a entornos donde las restricciones temporales y operativas son determinantes para la calidad del servicio.

Esta variante refleja de forma más precisa los desafíos logísticos actuales, en los que los clientes no solo deben ser visitados, sino también atendidos dentro de intervalos de tiempo específicos. Esto impone una coordinación precisa entre la planificación de rutas, la gestión de recursos y el cumplimiento de niveles de servicio.

Por ello, resulta fundamental comprender los fundamentos del TSP y del VRP, ya que constituyen la base teórica sobre la cual se desarrolla el VRP-TW, que es el problema principal abordado en este trabajo. A partir de este punto, se profundizará en las técnicas y metodologías empleadas para resolver dicha variante.

### 2.2.5. Métodos de Solución para el VRP-TW

Los problemas de enrutamiento como el TSP, el VRP y su extensión con ventanas de tiempo, el VRP-TW, mencionados en secciones anteriores, pertenecen a la clase de problemas NP-difíciles [44, 34, 53]. Esta clasificación se debe al crecimiento exponencial del tiempo requerido para resolverlos a medida que aumenta el número de nodos o clientes en la instancia.

Dada su complejidad computacional, estos problemas han sido abordados mediante diversas estrategias de solución que tienen como objetivo minimizar el costo total del recorrido, respetando las restricciones impuestas. Entre ellos se incluyen los métodos exactos, así como las técnicas heurísticas y metaheurísticas.

A continuación, se describen estos enfoques, junto con ejemplos representativos aplicados en los problemas mencionados.

#### 2.2.5.1. Métodos Exactos

Son aquellos que parten de una formulación matemática del problema, generalmente como modelos de programación lineal entera o similares, y alcanzan soluciones factibles mediante algoritmos que acotan el conjunto de soluciones posibles [36]. Estos métodos garantizan la obtención de una solución óptima; sin embargo, su principal limitación radica en el alto costo computacional que implican, por lo que su eficiencia se restringe a instancias de tamaño reducido, de hasta aproximadamente 50 clientes en problemas de enrutamiento[42].

Con base en la clasificación presentada por [42], los métodos exactos pueden agruparse en las siguientes categorías:

1. **Técnicas de relajación:** consisten en simplificar el modelo relajando ciertas restricciones complejas, lo que permite obtener cotas inferiores o superiores que

guían la búsqueda de soluciones.

2. **Búsqueda directa en árbol:** algoritmos como *Branch and Bound* y *Branch and Cut* exploran el espacio de soluciones mediante estructuras jerárquicas, descartando ramas no prometedoras.
3. **Programación dinámica:** resuelve el problema descomponiéndolo en subproblemas más pequeños que se abordan de forma recursiva y eficiente.
4. **Programación lineal entera:** utiliza modelos enteros mixtos como formulación clásica para problemas como el VRP-TW, y sirve frecuentemente de base para otros enfoques híbridos.

Dado que el presente trabajo se enfoca en técnicas heurísticas y metaheurísticas, no se abordará en detalle la implementación de estos métodos exactos. Para una revisión más amplia, se remite al lector a [42].

#### 2.2.5.2. Heurísticas

Heurística es un concepto cuyo origen se remonta a la Grecia clásica, derivado de la palabra griega *heuriskein*, que significa encontrar o descubrir. Según la historia, se asocia con *eureka*, la famosa exclamación atribuida a Arquímedes [2].

Para ejemplificar este concepto, se expone la siguiente definición:

Según Zanakis et al. (citado en [38]), las heurísticas son “*procedimientos simples, a menudo basados en el sentido común, que se supone que obtendrán una buena solución (no necesariamente óptima) a problemas difíciles de un modo sencillo y rápido*”.

El desarrollo de las técnicas heurísticas clásicas aplicadas al VRP-TW se llevó a cabo principalmente entre las décadas de 1960 y 1990. Estas pueden clasificarse en tres categorías generales: métodos constructivos, métodos de dos fases y heurísticas de mejora [42], las cuales se describen a continuación.

**Métodos constructivos:** Las heurísticas constructivas elaboran progresivamente una solución factible, considerando los costos asociados en cada paso del proceso. Sin embargo, este tipo de métodos no incorpora una fase de mejora posterior sobre la solución obtenida [53].

**Métodos de dos fases:** Las heurísticas de dos fases dividen el problema en dos componentes principales: la agrupación de vértices en rutas factibles y la construcción de dichas rutas, pudiendo existir retroalimentación entre ambas etapas. Estas técnicas se clasifican en dos enfoques: agrupación primero, ruta después y ruta primero, agrupación después. En el primer enfoque, los vértices se organizan inicialmente en grupos factibles y luego se construye una ruta vehicular para cada uno. En el segundo, se

genera primero un recorrido global que abarca todos los vértices y posteriormente se segmenta en rutas factibles [53].

En el presente trabajo se emplearon heurísticas basadas en los métodos de mejora, por lo que no se profundizará en los demás enfoques. Para una descripción más detallada, se recomienda consultar [42, 53, 41].

**Métodos de mejora:** Los métodos de mejora tienen como propósito optimizar una solución factible existente mediante una serie de intercambios o modificaciones de aristas y vértices, tanto dentro como entre rutas vehiculares, con el fin de disminuir el costo total del recorrido [53].

Entre los operadores clásicos utilizados en los métodos de mejora para problemas de enrutamiento destacan los siguientes:

1. **Operador de intercambio  $\lambda$ :** Consiste en eliminar  $\lambda$  aristas de la solución y reconectar los  $\lambda$  segmentos restantes. Una solución es óptima si no puede ser mejorada utilizando  $\lambda$  intercambios [41].
2. **Algoritmo de Lin-Kernighan:** Utiliza la idea de intercambiar un subconjunto de arcos por otro, donde dichos subconjuntos (y su cardinalidad) se modifican durante la ejecución del algoritmo. Se determinan dos conjuntos de aristas  $x_1, \dots, x_k$  e  $y_1, \dots, y_k$ , tales que al realizar un intercambio entre estos disminuya el costo de la solución. Las aristas  $x$  deben ser parte de la ruta, ambos conjuntos deben ser disjuntos y, además, eliminar las aristas  $x$  y agregar las aristas  $y$  formando una ruta cerrada [42].
3. **Operador Or-opt:** Consiste en eliminar una secuencia de  $k$  clientes consecutivos de la ruta y colocarlos en otra posición de la ruta, de modo que permanezcan consecutivos y en el mismo orden [42, 53, 41].
4. **GENI y GENIUS:** Surgen dentro de un método de solución para el TSP y tienen como principal característica que la inserción de un cliente en una ruta no necesariamente ocurre entre dos clientes adyacentes [42].
5. **Algoritmos de transferencias cíclicas:** Son movimientos multi-ruta que intentan eliminar clientes de una ruta y reubicarlos en otra de manera cíclica. También existe un movimiento que consiste en mover clientes de una ruta a otra en la misma posición [42].
6. **Operadores de Van Breedam:** El primero, denominado Cadena de Traslado (*String Relocation*), se define como una secuencia de  $m$  nodos que es transferida de una ruta a otra manteniendo el orden en la ruta original. El segundo, denominado Cadena de Intercambio (*String Exchange*), intercambia una secuencia de  $m$  clientes con una secuencia de  $n$  clientes entre dos rutas [42, 53, 41].

En relación con lo expuesto en la subsubsection 2.2.3.1, los problemas de decisión pertenecientes a la clase NP presentan una complejidad tal que no es posible garantizar la obtención de una solución óptima en tiempo polinómico razonable.

En este contexto, los métodos heurísticos constituyen una alternativa eficaz para abordar el problema de optimización combinatoria VRP-TW, donde los métodos exactos resultan inviables a gran escala.

Por lo tanto, las heurísticas permiten obtener soluciones satisfactorias en tiempos razonables, priorizando el equilibrio entre la calidad de la solución y la eficiencia computacional. Aunque no garantizan la optimalidad, su aplicación resulta esencial cuando las soluciones exactas son prohibitivas.

### 2.2.5.3. Metaheurísticas

El término metaheurística fue introducido por Fred Glover en 1986 [2]. Etimológicamente, deriva de la composición de dos palabras de origen griego: “meta” y “heurística”. El segundo término ha sido descrito en la subsección anterior 2.2.5.2, mientras que el prefijo “meta” puede traducirse como “más allá de” o “en un nivel superior” [38].

Con este término, Glover pretendía definir un *procedimiento maestro de alto nivel que guía y modifica otras heurísticas para explorar soluciones más allá de la simple optimalidad local* [38].

Para ilustrar este concepto, cabe mencionar la definición propuesta por J. P. Kelly et al., citado por [38]:

*Las metaheurísticas son métodos aproximados especialmente diseñados para abordar problemas complejos de optimización combinatoria, donde los heurísticos tradicionales resultan ineficaces. Estas técnicas ofrecen un marco flexible para desarrollar algoritmos híbridos que integran conceptos provenientes de la inteligencia artificial, la evolución biológica y procesos estadísticos.*

Para el problema clásico VRP-TW, se han implementado diversas metaheurísticas debido a su capacidad para ofrecer soluciones de buena calidad en tiempos razonables, especialmente cuando las técnicas exactas se vuelven ineficaces por la complejidad combinatoria del espacio de búsqueda.

Según [2], las metaheurísticas se clasifican de acuerdo con la fuente de inspiración que las fundamenta. A continuación, se describen brevemente tres categorías relevantes.

## Algoritmos Evolutivos

Los algoritmos evolutivos (EA, *Evolutionary Algorithms*) constituyen una familia de técnicas inspiradas en los principios de la evolución biológica propuestos por Charles Darwin, principalmente en la selección natural y la supervivencia del más apto. Su objetivo es buscar soluciones óptimas o cercanas al óptimo mediante un proceso iterativo que simula la evolución de una población de individuos [20].

El procedimiento general de un algoritmo evolutivo puede describirse de la siguiente manera [3]:

1. **Inicialización:** Se genera una población inicial de individuos de manera aleatoria o siguiendo algún criterio heurístico.
2. **Evaluación:** Cada individuo es evaluado mediante la función de aptitud.
3. **Selección:** Se eligen los individuos más aptos para reproducirse.
4. **Recombinación y mutación:** Se aplican operadores que combinan y modifican las soluciones seleccionadas para crear nuevas generaciones.
5. **Reemplazo:** Se forma una nueva población con los individuos más aptos de la generación actual y las nuevas soluciones.
6. **Criterio de parada:** El proceso se repite hasta alcanzar un número máximo de iteraciones o una solución satisfactoria.

Cada individuo representa una posible solución al problema y se evalúa mediante una función de aptitud o *fitness*, que mide qué tan buena es la solución respecto al objetivo. A partir de esta población inicial, se aplican operadores evolutivos que imitan la reproducción y la mutación biológica, generando nuevas soluciones.

## Algoritmos Genéticos

Los algoritmos genéticos (GA, *Genetic Algorithms*) son una de las técnicas más representativas dentro de los algoritmos evolutivos. Propuestos inicialmente por John Holland en la década de 1970 [28], estos algoritmos utilizan una representación basada en *cromosomas*, los cuales codifican posibles soluciones al problema. Los operadores principales que emplean son [25]:

- **Cruzamiento (crossover):** combina la información de dos individuos para generar descendencia con características de ambos.
- **Mutación:** introduce pequeñas variaciones aleatorias para mantener la diversidad genética y evitar la convergencia prematura.

- **Selección:** elige los individuos más aptos para reproducirse, simulando la presión evolutiva.

Estos algoritmos destacan por su capacidad para explorar de manera eficiente grandes espacios de búsqueda y adaptarse a distintas configuraciones del problema. Gracias a la combinación de operadores genéticos y mecanismos de selección, los GA permiten generar soluciones robustas y competitivas incluso en escenarios de alta complejidad combinatoria.

Aunque los algoritmos evolutivos (EA) y los algoritmos genéticos (GA) han sido descritos de manera general, a continuación se presenta una variante de especial relevancia para esta investigación. Dicha técnica será fundamental en el desarrollo del presente trabajo, ya que se implementará como parte del enfoque de optimización propuesto. En la siguiente sección se describe el funcionamiento del algoritmo de Evolución Diferencial (DE).

### Evolución Diferencial (DE)

Dentro de la familia de los algoritmos evolutivos, la Evolución Diferencial (*Differential Evolution*, DE) se ha consolidado como una de las técnicas más eficaces para la optimización de problemas continuos, debido a su simplicidad conceptual, bajo número de parámetros de control y alta capacidad de exploración del espacio de búsqueda [49, 47].

El DE comparte los fundamentos biológicos de otras metaheurísticas evolutivas, como la selección natural y la supervivencia del más apto, pero se diferencia por su mecanismo basado en combinaciones vectoriales de diferencias entre individuos de la población, lo que le permite mantener un equilibrio adecuado entre exploración y explotación.

Este algoritmo trabaja con una población de soluciones candidatas representadas como vectores de valores reales, que evolucionan a lo largo de varias generaciones mediante tres operadores principales: mutación, recombinación y selección. De manera general, su funcionamiento puede describirse de la siguiente forma:

- **Mutación:** genera un vector mutante  $v_i$  combinando individuos de la población. Una de las estrategias más utilizadas es **DE/rand/1**:

$$v_i = x_{r_1} + F \cdot (x_{r_2} - x_{r_3}) \quad (2.22)$$

donde  $r_1, r_2, r_3$  son índices aleatorios distintos entre sí y diferentes de  $i$ , y  $F$  es el factor de mutación ( $F \in [0, 2]$ , típicamente entre 0.5 y 1). Este operador introduce diversidad y permite explorar nuevas regiones del espacio de búsqueda.

- **Recombinación:** combina el vector mutante con el vector original  $x_i$  para formar un vector de prueba  $u_i$ , siguiendo el esquema de cruce binomial:

$$u_{i,j} = \begin{cases} v_{i,j} & \text{si } \text{rand}() \leq CR \\ x_{i,j} & \text{en otro caso} \end{cases} \quad (2.23)$$

donde  $CR \in [0, 1]$  es la tasa de cruce o probabilidad de recombinación. Este operador combina información de distintas soluciones, favoreciendo la explotación de configuraciones prometedoras.

- **Selección:** compara el vector de prueba  $u_i$  con el individuo original  $x_i$  mediante la función objetivo, conservando únicamente el de mejor aptitud para la siguiente generación:

$$x_i^{(t+1)} = \begin{cases} u_i & \text{si } f(u_i) < f(x_i^{(t)}) \\ x_i^{(t)} & \text{en otro caso} \end{cases} \quad (2.24)$$

Este mecanismo garantiza la mejora progresiva de la población, evitando el deterioro de soluciones de alta calidad.

## Metaheurísticas inspiradas en la biología

Una clase particular de metaheurísticas que ha ganado gran popularidad en los últimos años se basa en reglas inspiradas en conceptos biológicos, tales como la evolución natural, el comportamiento de las hormigas o los patrones de vuelo de las aves. A este conjunto de técnicas se le conoce como metaheurísticas bio-inspiradas [10]. Estas técnicas buscan reproducir comportamientos colectivos o procesos naturales que conducen a la emergencia de soluciones eficientes sin supervisión centralizada.

Un ejemplo representativo de este tipo de técnicas es la Optimización por Colonias de Hormigas, la cual es una metaheurística bio-inspirada y relativamente reciente [2].

## Optimización por Colonias de Hormigas

La Optimización por Colonias de Hormigas conocido por sus siglas en inglés como ACO (*Ant Colony Optimization*) es una metaheurística bio-inspirada que toma como referencia el comportamiento colectivo de las hormigas al buscar caminos eficientes entre su colonia y una fuente de alimento [19].

En la naturaleza, las hormigas depositan una sustancia química llamada feromona sobre el suelo al desplazarse. A mayor cantidad de feromona en un camino, mayor es la probabilidad de que otras hormigas lo sigan. Con el tiempo, los caminos más cortos tienden a concentrar más feromonas, ya que son recorridos con mayor frecuencia. Este mecanismo colectivo da lugar a la autoorganización y a la aparición de soluciones eficientes sin supervisión centralizada [15, 19].

De forma análoga, en ACO cada hormiga representa una solución candidata al problema de optimización. Las decisiones de construcción de soluciones están guiadas por dos factores: la cantidad de feromonas acumuladas en cada componente de la solución y una medida heurística que representa la deseabilidad local. La selección del siguiente componente se realiza mediante una regla de transición estocástica que pondera ambos factores [15].

El algoritmo opera iterativamente: un conjunto de hormigas construye soluciones, se evalúan los costos de las mismas, se actualiza la cantidad de feromonas en función de la calidad de las rutas encontradas y se repite el proceso durante un número determinado de iteraciones o hasta alcanzar una condición de parada. Con el tiempo, el sistema converge hacia soluciones de alta calidad [19].

### Modelo de selección probabilística en la Optimización por Colonias de Hormigas

En el algoritmo de Optimización por Colonias de Hormigas, la probabilidad de que una hormiga en el nodo  $i$  seleccione el nodo  $j$  como siguiente destino se calcula utilizando la siguiente expresión [19]:

$$p_{ij}^k(t) = \begin{cases} \frac{[\tau_{ij}(t)]^\alpha \cdot [\eta_{ij}]^\beta}{\sum_{l \in \mathcal{N}_i^k} [\tau_{il}(t)]^\alpha \cdot [\eta_{il}]^\beta}, & \text{si } j \in \mathcal{N}_i^k, \\ 0, & \text{en otro caso.} \end{cases} \quad (2.25)$$

donde:

- $p_{ij}^k(t)$ : probabilidad de que la hormiga  $k$  se desplace del nodo  $i$  al nodo  $j$  en el tiempo  $t$ ,
- $\tau_{ij}(t)$ : cantidad de feromonas en el borde  $(i, j)$  en el tiempo  $t$ ,
- $\eta_{ij} = \frac{1}{d_{ij}}$ : visibilidad heurística, usualmente el inverso de la distancia,
- $\alpha$ : parámetro que regula la influencia de la feromona,
- $\beta$ : parámetro que regula la influencia de la heurística,
- $\mathcal{N}_i^k$ : conjunto de nodos aún no visitados por la hormiga  $k$  al estar en  $i$ .

Esta regla de transición balancea el conocimiento aprendido (feromonas) y la heurística (visibilidad), dirigiendo la exploración de las hormigas a lo largo del espacio de soluciones. Fue propuesta originalmente por Dorigo, Maniezzo y Colomi en sus estudios sobre el TSP [19].

## Metaheurísticas inspiradas en la física

Este tipo de metaheurísticas toma como referencia principios y fenómenos físicos para guiar el proceso de búsqueda de soluciones óptimas. Un ejemplo representativo es el Recocido Simulado [2].

### Recocido Simulado

El Recocido Simulado (\*Simulated Annealing\*, SA) es una metaheurística inspirada en el proceso físico de recocido de los metales, donde se calienta un sólido a altas temperaturas y luego se enfria de manera controlada, permitiendo que sus partículas se reorganicen y alcancen una estructura de mínima energía [15].

De forma análoga, en el contexto de la optimización combinatoria, cada solución posible del problema se considera un “estado” del sistema, y su calidad se mide como si fuera una “energía”. El objetivo es encontrar el estado (solución) con la menor energía (mejor valor de la función objetivo) [15].

El algoritmo comienza con una solución inicial y, en cada iteración, genera una nueva solución vecina. Si la nueva solución es mejor, se acepta; si es peor, puede ser aceptada con una cierta probabilidad que depende de un parámetro llamado “temperatura”. A medida que el algoritmo avanza, la temperatura disminuye, reduciendo así la probabilidad de aceptar soluciones peores. Este mecanismo permite escapar de óptimos locales y explorar mejor el espacio de soluciones [15].

Este enfoque resulta particularmente útil en problemas de optimización combinatoria, donde el objetivo es minimizar una función de costo (por ejemplo, distancia total recorrida, tiempo o penalizaciones por restricciones de ventanas de tiempo). Al incorporar una aceptación probabilística de soluciones no óptimas, el algoritmo equilibra la exploración y la explotación, favoreciendo una búsqueda más efectiva del óptimo global.

### Modelo de distribución de Maxwell-Boltzmann aplicado al Recocido Simulado

La distribución de Maxwell-Boltzmann describe la probabilidad de que un sistema físico adopte un estado con una energía  $E_i$  a una temperatura  $T$ . Esta probabilidad se expresa como:

$$p_i = \frac{e^{-E_i/kT}}{Z} \quad (2.26)$$

donde se tiene que:

- $p_i$ : probabilidad de que el sistema se encuentre en el estado  $i$ ,
- $E_i$ : energía del estado  $i$ ,
- $k$ : constante de Boltzmann,
- $T$ : temperatura absoluta del sistema,
- $Z = \sum_j e^{-E_j/kT}$ : función de partición que normaliza las probabilidades.

Este modelo permite explicar cómo se distribuyen los estados energéticos en sistemas en equilibrio térmico, y es ampliamente utilizado como base probabilística en algoritmos como el SA para justificar la aceptación de soluciones peores durante las fases iniciales del proceso de enfriamiento [30].

Cada una de las metaheurísticas mencionadas —Evolución Diferencial (DE), Optimización por Colonias de Hormigas (ACO) y Recocido Simulado (SA)— puede aplicarse eficazmente a problemas de optimización complejos. Estas técnicas han demostrado ser herramientas valiosas en la resolución del VRP-TW, gracias a su capacidad para explorar amplios espacios de búsqueda y aproximarse a soluciones óptimas en tiempos computacionales razonables.

Si bien existen numerosas estrategias metaheurísticas documentadas en la literatura, en este trabajo se han seleccionado estas tres por su relevancia teórica, versatilidad y desempeño comprobado en problemas de optimización combinatoria. En el siguiente capítulo se describirá detalladamente la forma en que cada una de estas técnicas fue adaptada e implementada dentro del contexto de la presente investigación.

#### 2.2.5.4. Algoritmos Híbridos

Los algoritmos híbridos combinan características de diferentes metaheurísticas con el objetivo de aprovechar las ventajas de cada una y superar las limitaciones individuales. Este tipo de enfoques surge de la necesidad de mejorar la eficiencia, precisión y robustez en la resolución de problemas de optimización complejos [20, 5].

En un algoritmo híbrido, se pueden combinar técnicas evolutivas con búsquedas locales, métodos exactos o incluso otras metaheurísticas, generando un enfoque que equilibra exploración y explotación de manera más efectiva que las técnicas puras [25]. La idea central es utilizar las fortalezas de cada técnica para complementar las debilidades de las demás, obteniendo así soluciones de mayor calidad en menos tiempo.

Los algoritmos híbridos son especialmente valiosos para el problema VRP-TW, donde la complejidad del espacio de soluciones puede ser muy elevada [54]. Gracias a la combinación de estrategias, estos algoritmos pueden:

- Mejorar la convergencia hacia soluciones cercanas al óptimo.

- Reducir la probabilidad de estancamiento en óptimos locales.
- Manejar restricciones adicionales de manera más eficiente.

#### Ejemplos de algoritmos híbridos:

- **GA + Búsqueda Local:** combina algoritmos genéticos con operadores de mejora local para refinar las soluciones generadas por la población; es útil en el TSP, el VRP y su variante con ventanas de tiempo, el VRP-TW.
- **DE + Recocido Simulado (SA):** integra Evolución Diferencial con Recocido Simulado, equilibrando exploración global y ajuste fino de soluciones.
- **ACO + Algoritmo Genético:** emplea la información de feromonas generada por Colonias de Hormigas como semilla para un algoritmo genético, acelerando la convergencia en problemas VRP-TW.

Por estas razones, los algoritmos híbridos se han consolidado como herramientas poderosas para resolver problemas complejos en logística, planificación de rutas, diseño de redes y otras áreas donde las metaheurísticas puras podrían no ser suficientes [5, 20].

En este trabajo se adopta un enfoque híbrido que integra el Algoritmo Evolutivo Diferencial (DE), la Optimización por Colonias de Hormigas (ACO) y el Recocido Simulado (SA) para abordar el VRP-TW, como se mencionó anteriormente. Esta combinación busca aprovechar las fortalezas complementarias de cada técnica. De esta manera, el enfoque híbrido propuesto no solo mejora la calidad de las soluciones obtenidas, sino que también contribuye a reducir el tiempo de cómputo necesario para alcanzar resultados competitivos.

En el siguiente capítulo se explicará con detalle cada etapa de nuestro algoritmo híbrido.

## 2.3. Referencias Relevantes

El Problema del Agente Viajero (TSP) constituye uno de los problemas fundamentales en optimización combinatoria y ha sido objeto de estudio desde principios del siglo XX. [12] presentaron la primera formulación matemática rigurosa del TSP y desarrollaron el método de planos de corte, estableciendo las bases teóricas para su resolución. Posteriormente, [27] propusieron un algoritmo de programación dinámica que reduce la complejidad computacional, aunque el problema permanece como NP-difícil. La importancia del TSP radica en que constituye el fundamento teórico de problemas más complejos de ruteo, siendo ampliamente estudiado tanto desde la perspectiva teórica como aplicada.

El Problema de Ruteo de Vehículos (VRP), introducido por [13], representa una generalización natural del TSP al considerar múltiples vehículos con capacidad limitada. Este trabajo seminal estableció el VRP como un problema central en logística y distribución. [9] desarrollaron el algoritmo de ahorros, una de las heurísticas constructivas más influyentes que continúa siendo referencia en el campo. [33] realizó una revisión comprehensiva de los métodos exactos para el VRP, consolidando el conocimiento acumulado durante tres décadas de investigación. [52] compilaron el estado del arte en su obra “The Vehicle Routing Problem”, que se ha convertido en referencia obligada para investigadores en el área.

El Problema de Ruteo de Vehículos con Ventanas de Tiempo (VRP-TW), que constituye el enfoque principal de este trabajo, fue formalmente definido por [48], quien además generó el conjunto de instancias de prueba que se ha convertido en el estándar de la industria para evaluar algoritmos. Solomon clasificó las instancias en seis categorías (C1, C2, R1, R2, RC1, RC2) según la distribución geográfica de los clientes y la amplitud de las ventanas de tiempo, estableciendo un marco de referencia que permanece vigente. [17] desarrollaron métodos exactos basados en generación de columnas y programación dinámica, demostrando la viabilidad teórica de resolver instancias de tamaño moderado de manera óptima.

En cuanto a enfoques metaheurísticos para el VRP-TW, [6, 7] publicaron una revisión exhaustiva en dos partes que analiza más de 200 trabajos, clasificando las metaheurísticas en búsqueda local, métodos basados en población y enfoques híbridos. [50] introdujeron la Búsqueda Tabú adaptativa para el VRP-TW, estableciendo nuevos estándares de calidad en las soluciones. [23] aplicaron el sistema de Colonia de Hormigas con múltiples colonias (MACS-VRP-TW), demostrando la efectividad de algoritmos bio-inspirados en este dominio.

Los algoritmos genéticos han mostrado particular eficacia para el VRP-TW. [46] desarrollaron un algoritmo genético con operadores especializados que considera simultáneamente la minimización de vehículos y distancia total. [29] propusieron un algoritmo evolutivo de dos fases que logró resultados competitivos en instancias de gran escala. Más recientemente, [55] presentaron un algoritmo genético híbrido (HGSADC) que incorpora búsqueda local intensiva y mecanismos de control de diversidad, estableciendo nuevos mejores resultados conocidos en múltiples instancias de Solomon.

La hibridación de metaheurísticas representa una tendencia consolidada en la literatura reciente. [39] combinaron búsqueda local guiada y penalización adaptativa, mientras que [45] desarrollaron el enfoque de Large Neighborhood Search (LNS) adaptativo, que ha sido extensamente aplicado en variantes del VRP. [4] presentaron una revisión de métodos exactos de última generación, destacando enfoques branch-and-cut-and-price que han permitido resolver instancias previamente intratables.

# Capítulo 3

## Materiales y Métodos

### 3.1. Materiales

Para el desarrollo y evaluación del algoritmo híbrido propuesto para resolver el VRP-TW, se emplearon los siguientes materiales y recursos:

#### 3.1.1. Datos e instancias del problema

Para la evaluación del algoritmo implementado se utilizaron conjuntos de datos de referencia para el problema de VRP-TW:

- **Instancias Solomon para VRP-TW:** Se emplearon las instancias clásicas de Solomon, incluyendo los conjuntos:
  - C101-C109, C201-C208
  - R101-R112, R201-R211
  - RC101-RC108, RC201-RC208

Cada instancia contiene un número específico de clientes (25, 50 o 100), permitiendo evaluar el desempeño de los algoritmos en problemas de diferente tamaño y complejidad.

- **Información de vehículos:** Cada vehículo de la flota cuenta con una capacidad máxima, utilizada para asegurar que la asignación de clientes no exceda los límites permitidos. Por ejemplo:

Número de vehículos	Capacidad
25	200

- **Información de clientes:** Cada cliente está definido por coordenadas geográficas, demanda, ventana de tiempo y tiempo de servicio, siguiendo el formato:

CUST NO.	XCOORD	YCOORD	DEMAND	READY TIME	DUE DATE	SERVICE TIME
1	45	30	10	0	200	10
2	20	50	15	0	180	15
:	:	:	:	:	:	:

Esta información es utilizada por los algoritmos para determinar la asignación de clientes a vehículos y planificar las rutas cumpliendo con las restricciones de capacidad y ventanas de tiempo.

En conjunto, estas instancias permiten evaluar el desempeño, eficiencia y calidad de las soluciones generadas por los algoritmos para distintos tamaños y configuraciones del problema VRPTW.

### 3.1.2. Software

Para el desarrollo de esta investigación se utilizaron los siguientes lenguajes, entornos y bibliotecas:

- **Lenguajes de programación:**

- **C:** Fue el lenguaje principal utilizado para la implementación de los algoritmos, debido a su naturaleza de bajo y alto nivel, lo que permitió un control preciso de la memoria y estructuras de datos, así como una ejecución rápida, fundamental para la simulación de problemas de alta complejidad como el VRP-TW.
- **Python:** Se empleó principalmente para procesar los archivos JSON generados por la implementación en C, y para la creación de gráficos estáticos y animaciones dinámicas de las rutas de los vehículos, aprovechando bibliotecas especializadas en visualización.

- **Entorno de desarrollo:** Visual Studio Code fue utilizado como IDE principal por su versatilidad, compatibilidad con múltiples lenguajes y facilidad para depurar y organizar proyectos complejos.

- **Bibliotecas:**

- **C:**

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include <dirent.h>
#include <sys/stat.h>
#include <unistd.h>
#include <string.h>
#include <cjson/cJSON.h>
```

Estas bibliotecas permitieron manejar entrada/salida, estructuras de datos, funciones matemáticas, manipulación de directorios, tiempos de ejecución y procesamiento de archivos JSON.

- **Python:**

```
import json
import os
import sys
import matplotlib.pyplot as plt
import matplotlib.animation as animation
import numpy as np
import time
from matplotlib.patches import Circle
```

Estas bibliotecas se usaron para el análisis de datos, generación de gráficos estáticos y animados, manejo de arreglos y matrices, así como la visualización de rutas y simulaciones dinámicas.

### 3.1.3. Hardware

El desarrollo completo del código y la ejecución de los experimentos se realizaron utilizando una computadora principal con las siguientes características, que también se utilizó para la recolección de resultados:

- **Procesador:** Intel Core i7 de 10.<sup>a</sup> generación.
- **Memoria RAM:** 20 GB.
- **Sistema operativo:** Linux (Debian 12).

Adicionalmente, se emplearon otras computadoras para la recolección de datos:

### Computadora 1:

- **Procesador:** Intel Core i5 de 11.<sup>a</sup> generación.
- **Memoria RAM:** 12 GB.
- **Sistema operativo:** Windows 11.

### Computadora 2:

- **Procesador:** Intel Core i3 de 6.<sup>a</sup> generación.
- **Memoria RAM:** 8 GB.
- **Sistema operativo:** Linux (Ubuntu).

Estas configuraciones permitieron realizar pruebas de ejecución de los algoritmos y generar los datos necesarios para el análisis de resultados, considerando diferentes capacidades de hardware para evaluar la eficiencia y escalabilidad de los métodos implementados.

### 3.1.4. Herramientas de apoyo

Para complementar el desarrollo de esta investigación y facilitar la presentación de resultados, se emplearon las siguientes herramientas de apoyo:

- **Redacción y documentación:** Se utilizó L<sup>A</sup>T<sub>E</sub>X como principal herramienta para la escritura de la tesis y la documentación del proyecto. Esto permitió generar un documento con formato profesional, manejo automático de referencias bibliográficas, índices, tablas, figuras y ecuaciones, asegurando uniformidad y claridad en la presentación de los resultados.
- **Control de versiones y organización:** Se utilizó Git como sistema de control de versiones para mantener un registro completo de los cambios realizados en el código fuente, asegurando trazabilidad y facilitando la colaboración eficiente durante la recolección.

## 3.2. Métodos

### 3.2.1. Descripción general del algoritmo híbrido

Para resolver el problema de VRP-TW, se propone un algoritmo híbrido que combina tres técnicas metaheurísticas: Algoritmo Evolutivo Diferencial (DE), Optimización

por Colonia de Hormigas (ACO) y Recocido Simulado (SA). Este enfoque aprovecha la capacidad de exploración global del ACO, la explotación local del SA y la calibración automática de parámetros mediante DE, con el objetivo de generar soluciones de alta calidad de manera eficiente.

El flujo jerárquico del método híbrido es el siguiente:

1. **Algoritmo Evolutivo Diferencial (DE):** se encarga de calibrar los parámetros de ACO y SA para optimizar la búsqueda de soluciones.
2. **Optimización por Colonia de Hormigas (ACO):** genera rutas iniciales factibles, cumpliendo las restricciones de capacidad y ventanas de tiempo de los vehículos o clientes.
3. **Recocido Simulado (SA):** refina las rutas iniciales aplicando heurísticas de mejora local sin violar las restricciones mencionadas con anterioridad y aceptando soluciones subóptimas bajo un criterio probabilístico, lo que permite escapar de óptimos locales.

El objetivo principal del enfoque propuesto es generar soluciones de alta calidad para instancias de VRP-TW, cumpliendo con las restricciones de capacidad de los vehículos y las ventanas de tiempo de los clientes, y optimizando la eficiencia y robustez del método a través de la calibración automática de parámetros y el refinamiento de rutas.

### 3.2.2. Calibración de parámetros mediante Evolucion Diferencial (DE)

Como primera etapa del enfoque híbrido, se emplea el Algoritmo Evolutivo Diferencial (DE) con el objetivo de calibrar los parámetros de las metaheurísticas ACO y SA de manera automática. Esta calibración permite maximizar la calidad de las soluciones obtenidas por ACO y SA para el VRP-TW.

Como se hizo mencion en la sección anterior, cada instancia fue segmentada en 3 partes la cual se especifica en la siguiente tabla:

Tamaño del problema	Número de clientes
Pequeña	25
Mediana	50
Grande	100

Cuadro 3.1: Clasificación de instancias según número de clientes

Sin importar la instancia o sus medidas, se realizó el proceso de calibración que optimiza simultáneamente los siguientes parámetros:

**■ Para ACO:**

- $\alpha$ : influencia de la información de feromonas
- $\beta$ : influencia de la información heurística (distancia)
- $\gamma$ : peso de las restricciones de ventanas de tiempo
- $\rho$ : tasa de evaporación de feromonas
- Número de hormigas
- Porcentaje de hormigas
- Número de iteraciones

**■ Para SA:**

- Temperatura inicial ( $T_0$ )
- Temperatura final ( $T_f$ )
- Factor de enfriamiento ( $\alpha_{cool}$ )
- Número de iteraciones

Para ambas metaheurísticas, es necesario definir rangos mínimos y máximos de los parámetros, y ajustar estos rangos según el tamaño específico de la instancia (pequeña, mediana o grande). Esto asegura que el algoritmo pueda adaptarse dinámicamente a la complejidad del problema.

**Configuración de rangos de parámetros según tamaño de instancia**

La calibración efectiva del algoritmo híbrido requiere establecer rangos de búsqueda apropiados para los parámetros de ACO y SA, adaptados a la complejidad de cada instancia. A continuación se presentan los rangos definidos para tres categorías de tamaño: pequeña (25 clientes), mediana (50 clientes) y grande (100 clientes). Estos rangos se ajustan progresivamente para equilibrar la exploración del espacio de soluciones con la eficiencia computacional.

### Instancias Pequeñas (25 clientes)

Parámetros ACO			Parámetros SA		
Parámetro	Mín.	Máx.	Parámetro	Mín.	Máx.
$\alpha$	2.0	3.5	Temperatura inicial	1200.0	1800.0
$\beta$	2.5	4.0	Temperatura final	0.01	0.1
$\gamma$	1.0	2.5	Factor enfriamiento	0.95	0.98
$\rho$	0.3	0.5	Iteraciones SA	80	120
Número de hormigas	3	6			
Iteraciones ACO	60	80			

Cuadro 3.2: Rangos de parámetros para instancias pequeñas

Para instancias pequeñas se establecen rangos moderados que equilibran explotación y explotación sin incurrir en costos computacionales innecesarios. Los valores de  $\alpha$  (2.0–3.5) y  $\beta$  (2.5–4.0) balancean la influencia de feromonas e información heurística, mientras que  $\rho$  (0.3–0.5) mantiene diversidad mediante evaporación moderada. El parámetro  $\gamma$  (1.0–2.5) prioriza inicialmente el cumplimiento de ventanas de tiempo. Con 3–6 hormigas y 60–80 iteraciones ACO, seguidas de 80–120 iteraciones SA con temperatura inicial moderada (1200.0–1800.0), se logra refinamiento eficiente en espacios de búsqueda reducidos.

### Instancias Medianas (50 clientes)

Parámetros ACO			Parámetros SA		
Parámetro	Mín.	Máx.	Parámetro	Mín.	Máx.
$\alpha$	2.5	4.0	Temperatura inicial	1600.0	2200.0
$\beta$	3.5	5.0	Temperatura final	0.005	0.05
$\gamma$	1.5	3.0	Factor enfriamiento	0.97	0.99
$\rho$	0.25	0.45	Iteraciones SA	120	200
Número de hormigas	6	12			
Iteraciones ACO	100	150			

Cuadro 3.3: Rangos de parámetros para instancias medianas

En instancias medianas se incrementan los rangos para fortalecer la exploración en espacios de búsqueda más amplios. Los valores elevados de  $\alpha$  (2.5–4.0) y  $\beta$  (3.5–5.0) intensifican la influencia de feromonas e información heurística. La tasa de evaporación reducida  $\rho$  (0.25–0.45) preserva mejor las buenas soluciones en la memoria colectiva. Con 6–12 hormigas y 100–150 iteraciones ACO, seguidas de refinamiento SA más exhaustivo (120–200 iteraciones, temperatura inicial 1600.0–2200.0), se equilibra calidad de solución y tiempo de cómputo en problemas de complejidad intermedia.

### Instancias Grandes (100 clientes)

Parámetros ACO			Parámetros SA		
Parámetro	Mín.	Máx.	Parámetro	Mín.	Máx.
$\alpha$	3.0	5.0	Temperatura inicial	2500.0	4000.0
$\beta$	4.0	6.0	Temperatura final	0.001	0.01
$\gamma$	2.0	4.0	Factor enfriamiento	0.97	0.995
$\rho$	0.10	0.30	Iteraciones SA	200	300
Número de hormigas	10	20			
Iteraciones ACO	120	180			

Cuadro 3.4: Rangos de parámetros para instancias grandes

Para instancias grandes se emplean rangos amplios que permiten exploración robusta en espacios de alta dimensionalidad. Los valores máximos de  $\alpha$  (3.0–5.0) y  $\beta$  (4.0–6.0) intensifican la explotación de soluciones prometedoras, mientras que  $\rho$  reducida (0.10–0.30) preserva conocimiento acumulado por más tiempo. El parámetro  $\gamma$  (2.0–4.0) alcanza sus valores máximos, reforzando el cumplimiento de ventanas de tiempo en problemas complejos. Con 10–20 hormigas, 120–180 iteraciones ACO y refinamiento SA intensivo (200–300 iteraciones, temperatura inicial elevada 2500.0–4000.0), se justifica el mayor costo computacional mediante la obtención de soluciones de alta calidad.

La configuración progresiva de estos rangos según el tamaño de instancia es fundamental para el desempeño del algoritmo DE. Los rangos más amplios en instancias grandes permiten mayor exploración, evitando convergencia prematura en espacios complejos con múltiples óptimos locales. Por otro lado, los rangos más acotados en instancias pequeñas aceleran la convergencia sin sacrificar calidad, aprovechando la menor complejidad del problema. Esta estratificación facilita la calibración automática mediante DE y garantiza la replicabilidad de los experimentos, permitiendo que el algoritmo híbrido se adapte eficientemente a diferentes escalas del VRP-TW.

### Representación de individuos

En el contexto del DE, cada individuo de la población no representa una solución directa del problema VRP-TW, sino una configuración completa de los parámetros que controlan el comportamiento de las metaheurísticas ACO y SA. Es decir, un individuo codifica cómo se ejecutarán los algoritmos de optimización, incluyendo aspectos como la importancia relativa de la información heurística, la cantidad de agentes exploradores, el número de iteraciones y las estrategias de enfriamiento de SA.

Esta representación se realiza mediante un vector de parámetros reales de dimensión  $D = 12$ :

$$\mathbf{x}_i = [\alpha, \beta, \gamma, \rho, nh, \%h, niter\_ACO, T_0, T_f, \alpha_{cool}, fc, niter\_SA]$$

donde cada componente controla un aspecto específico de las metaheurísticas:

- $\alpha, \beta, \gamma, \rho$ : parámetros de ACO que determinan el balance entre exploración y explotación, la importancia de feromonas, heurística de distancia y factibilidad temporal.
- $nh$ : número de hormigas (agentes exploradores) en ACO.
- $\%h$ : porcentaje de hormigas respecto al total de número de hormigas.
- $niter\_ACO$ : número de iteraciones del ACO.
- $T_0, T_f$ : temperaturas inicial y final de SA.
- $\alpha_{cool}$ : factor de enfriamiento de SA.
- $fc$ : factor de control de SA que modula la aceptación de soluciones subóptimas.
- $niter\_SA$ : número de iteraciones de SA.

La existencia de individuos en el DE permite que el algoritmo explore de manera automática el espacio de parámetros, buscando combinaciones que generen mejores soluciones al VRP-TW. En otras palabras, cada individuo es una “receta” de cómo se ejecutará ACO+SA, y la población evoluciona para descubrir la configuración más efectiva para cada instancia.

Cada componente del vector está acotada dentro de rangos específicos  $[min_j, max_j]$ , que dependen del tamaño de la instancia (pequeña, mediana o grande), asegurando que los valores generados sean factibles y coherentes con el problema a resolver, y evitando configuraciones que puedan producir resultados inestables o poco eficientes.

### Función de Aptitud

La función de aptitud evalúa la calidad de cada individuo, representado por un vector de parámetros, mediante la ejecución del algoritmo híbrido ACO+SA. Su propósito es cuantificar el desempeño de cada configuración de parámetros para determinar qué tan efectiva resulta en la resolución del VRP-TW.

El proceso de evaluación se desarrolla conforme a los siguientes pasos:

1. Recibir un individuo  $x_i$  que representa una configuración específica de parámetros del modelo.

2. Inicializar las estructuras del problema (matrices de distancias, visibilidad, feromonas y ventanas de tiempo) de acuerdo con los valores contenidos en  $x_i$ .
3. Ejecutar el algoritmo ACO utilizando los parámetros  $\alpha, \beta, \rho, \gamma$ , número de hormigas y número de iteraciones asociados al individuo  $x_i$ , generando una solución inicial de rutas.
4. Aplicar el algoritmo SA con sus respectivos parámetros ( $T_0, T_f, \alpha_{cool}$ , factor de control y número de iteraciones) para mejorar las rutas obtenidas por ACO.
5. Calcular el *fitness* del individuo a partir del costo total de las rutas finales, recopilando la información necesaria para su evaluación.

### Definición de la Función de Aptitud

La función de aptitud se define a partir de la función objetivo del VRP-TW como:

$$f(\mathbf{x}_i) = \text{Costo}_{\text{total}} \quad (3.1)$$

donde:

- $\text{Costo}_{\text{total}}$  representa la distancia total recorrida por todos los vehículos en la solución generada.

**Objetivo:** Minimizar  $f(\mathbf{x}_i)$ , es decir, encontrar la configuración de parámetros que produzca el menor costo total de rutas, cumpliendo las restricciones impuestas por el VRP-TW.

Cada evaluación de la función de aptitud implica ejecutar de forma completa el proceso híbrido ACO → SA, lo cual constituye el componente de mayor costo computacional dentro del algoritmo evolutivo diferencial (DE). Por este motivo, los valores del tamaño de población y del número de generaciones deben seleccionarse cuidadosamente para lograr un equilibrio entre precisión y tiempo de ejecución.

### Parámetros del Algoritmo Evolutivo Diferencial

El DE requiere configurar cuatro parámetros principales que controlan su comportamiento y rendimiento:

Parámetro	Valor	Descripción
Tamaño de población ( $NP$ )	20–30	Número de configuraciones de parámetros evaluadas en cada generación. Valores mayores aumentan la diversidad pero incrementan el costo computacional.
Número de generaciones ( $G$ )	50–150	Número de iteraciones del algoritmo evolutivo. Se ajusta según el tamaño de la instancia: 50–80 para instancias pequeñas, 80–120 para medianas, y 100–150 para grandes.
Factor de mutación ( $F$ )	0.5	Controla la magnitud de la perturbación en el operador de mutación. Un valor de 0.5 proporciona un balance adecuado entre exploración y explotación.
Tasa de cruce ( $CR$ )	0.5	Probabilidad de que cada componente del vector provenga del vector mutante durante la recombinación. Un valor de 0.5 garantiza igual influencia de ambos padres.

Cuadro 3.5: Parámetros del Algoritmo Evolutivo Diferencial

### Criterio de parada

El algoritmo DE finaliza su ejecución cuando se alcanza el número máximo de generaciones. Esto ocurre al completarse el número de generaciones  $G$  especificado.

El criterio principal utilizado en la implementación es el número máximo de generaciones, garantizando que el algoritmo realice una exploración suficientemente exhaustiva del espacio de parámetros antes de retornar la mejor configuración encontrada.

Al finalizar, el algoritmo encuentra el individuo con el menor fitness de toda la ejecución, el cual representa la mejor configuración de parámetros para ACO y SA, además de la mejor ruta encontrada durante el proceso evolutivo.

### Pseudocódigo del Algoritmo Evolutivo Diferencial

A continuación se presenta el pseudocódigo del algoritmo DE implementado para la calibración automática de parámetros:

---

**Algorithm 3.1:** Evolución Diferencial para Calibración de Parámetros en VRP-TW
 

---

**Input:**  $NP$  (población),  $G$  (generaciones), instancia VRP-TW  
**Output:** Mejor individuo con parámetros óptimos

- 1 **Inicialización:**
- 2 Leer instancia; Inicializar matrices (distancias, visibilidad, feromonas, ventanas).
- 3 Definir rangos según tamaño de instancia.
- 4 **para**  $i = 1$  **hasta**  $i = NP$  **hacer**
  - 5 Generar objetivo $[i]$  aleatorio; Evaluar con ACO+SA.
- 6 **fin para**
- 7 MejorIndividuo  $\leftarrow$  individuo con menor fitness.
- 8 **Evolución:**
- 9 **para**  $g = 1$  **hasta**  $g = G$  **hacer**
  - 10 **para**  $i = 1$  **hasta**  $i = NP$  **hacer** (*Mutación*)
    - 11 Seleccionar  $r_1, r_2, r_3$  distintos; Calcular ruidoso $[i]$  con  $DE/rand/1$ .
    - 12 Limitar ruidoso $[i]$  a rangos válidos.
  - 13 **fin para**
  - 14 **para**  $i = 1$  **hasta**  $i = NP$  **hacer** (*Recombinación*)
    - 15 Generar aleatorio  $r$ ; prueba $[i] \leftarrow$  objetivo $[i]$  si  $r \leq 0,5$ , sino ruidoso $[i]$ .
  - 16 **fin para**
  - 17 **para**  $i = 1$  **hasta**  $i = NP$  **hacer** (*Evaluación*)
    - 18 Evaluar prueba $[i]$  con ACO+SA.
    - 19 si fitness(prueba $[i]$ ) < fitness(MejorIndividuo) entonces actualizar.
  - 20 **fin para**
  - 21 **para**  $i = 1$  **hasta**  $i = NP$  **hacer** (*Selección*)
    - 22 si fitness(prueba $[i]$ ) < fitness(objetivo $[i]$ ) entonces objetivo $[i] \leftarrow$  prueba $[i]$ .
  - 23 **fin para**
  - 24 **fin para**
  - 25 Retornar MejorIndividuo.

---

Figura 3.1: Pseudocódigo del algoritmo de Evolución Diferencial (DE) aplicado al problema VRP-TW.

### 3.2.3. Construcción de rutas iniciales mediante Optimización por Colonia de Hormigas (ACO)

Una vez calibrados los parámetros mediante DE, la segunda etapa del enfoque híbrido consiste en construir soluciones factibles para el VRP-TW utilizando Optimización por Colonia de Hormigas (ACO). Este algoritmo bio-inspirado simula el comportamiento de las hormigas reales al buscar alimento, utilizando rastros de feromonas para encontrar rutas eficientes que minimicen la distancia total recorrida mientras respetan las restricciones de capacidad de los vehículos y las ventanas de tiempo de los clientes.

### Fundamentos del algoritmo ACO

El algoritmo ACO se basa en tres componentes principales que guían la construcción probabilística de soluciones:

1. **Información de feromonas ( $\tau_{ij}$ ):** Representa el conocimiento acumulado sobre qué conexiones entre clientes han formado parte de buenas soluciones en iteraciones previas. Mayor nivel de feromona indica rutas prometedoras.
2. **Información heurística ( $\eta_{ij}$ ):** Proporciona conocimiento a priori sobre la calidad de las conexiones, típicamente basado en la distancia euclídea entre clientes. Se define como  $\eta_{ij} = 1/d_{ij}$ , favoreciendo clientes cercanos.
3. **Información de ventanas de tiempo ( $\omega_{ij}$ ):** Componente adicional específico para VRP-TW que considera la factibilidad temporal de visitar el cliente  $j$  después del cliente  $i$ . Se calcula como  $\omega_{ij} = 1/l_j$ , donde  $l_j$  es el límite superior de la ventana de tiempo del cliente  $j$ .

### Construcción probabilística de rutas

Cada hormiga artificial construye una solución completa siguiendo una estrategia constructiva voraz aleatorizada. El proceso inicia desde el depósito y en cada paso, la hormiga  $k$  selecciona el siguiente cliente  $j$  a visitar desde su posición actual  $i$  con una probabilidad dada por:

$$p_{ij}^k = \frac{[\tau_{ij}]^\alpha \cdot [\eta_{ij}]^\beta \cdot [\omega_{ij}]^\gamma}{\sum_{l \in N_i^k} [\tau_{il}]^\alpha \cdot [\eta_{il}]^\beta \cdot [\omega_{il}]^\gamma} \quad (3.2)$$

donde:

- $N_i^k$  es el conjunto de clientes factibles que la hormiga  $k$  puede visitar desde  $i$
- $\alpha$  controla la influencia de la información de feromonas
- $\beta$  controla la influencia de la información heurística (distancia)
- $\gamma$  controla la influencia de las ventanas de tiempo

Un cliente  $j$  se considera factible si cumple simultáneamente:

- No ha sido visitado previamente en la ruta actual
- La capacidad del vehículo no se excede:  $q_{\text{actual}} + d_j \leq Q$

- La ventana de tiempo puede satisfacerse:  $t_{\text{llegada}} \leq l_j$
- El vehículo puede regresar al depósito después de servir al cliente  $j$  dentro de su límite temporal

En el siguiente psudocódigo detalla el proceso de verificación de factibilidad para determinar qué clientes pueden ser visitados desde la posición actual del vehículo.

---

**Algorithm 3.2:** Cálculo de Clientes Factibles

---

**Input:** Cliente origen, vehículo  $v$ , lista tabú  $T$ , configuración VRP

**Output:** Conjunto de clientes factibles

```

1 Candidatos  $\leftarrow \emptyset$ .
2 para  $i = 1$  hasta  $n - 1$  hacer (excluir depósito)
3   si  $i \notin T$  entonces
4      $t_{\text{viaje}} \leftarrow d_{\text{origen},i} / v.\text{velocidad}$ .
5      $t_{\text{llegada}} \leftarrow v.t_{\text{actual}} + t_{\text{viaje}}$ .
6      $t_{\text{retorno}} \leftarrow d_{i,0} / v.\text{velocidad}$ .
7      $t_{\text{total}} \leftarrow t_{\text{llegada}} + s_i + t_{\text{retorno}}$ .
8     si  $t_{\text{llegada}} \geq e_i$  y  $t_{\text{llegada}} \leq l_i$  entonces
9       si  $v.q + d_i \leq v.Q$  entonces
10        si  $t_{\text{total}} \leq l_0$  entonces
11          Candidatos  $\leftarrow$  Candidatos  $\cup \{i\}$ .
12        fin si
13      fin si
14    fin si
15  fin si
16 fin para
17 Retornar Candidatos.

```

---

Figura 3.2: Pseudocódigo de verificación de clientes factibles en ACO.

### Gestión de restricciones temporales

El manejo de las ventanas de tiempo es crítico en el VRP-TW. Para cada cliente  $j$  visitado, se calculan:

$$t_{\text{llegada}}^j = t_{\text{salida}}^i + \frac{d_{ij}}{v} \quad (3.3)$$

**Ecuación 3.3:** Calcula el tiempo de llegada del vehículo al cliente  $j$ , sumando el tiempo de salida del cliente anterior  $i$  y el tiempo de viaje desde  $i$  hasta  $j$ .

$$t_{\text{inicio\_servicio}}^j = \max(t_{\text{llegada}}^j, e_j) \quad (3.4)$$

**Ecuación 3.4:** Determina el instante en que inicia el servicio en el cliente  $j$ . Si el vehículo llega antes de  $e_j$ , debe esperar hasta el inicio de la ventana; si llega dentro de la ventana, comienza el servicio inmediatamente.

$$t_{\text{salida}}^j = t_{\text{inicio\_servicio}}^j + s_j \quad (3.5)$$

**Ecuación 3.5:** Calcula el tiempo de salida del vehículo del cliente  $j$ , sumando al inicio del servicio el tiempo de servicio requerido  $s_j$ .

### Mecanismo de gestión de flota

Cuando una hormiga no encuentra clientes factibles desde su posición actual, se implementa una estrategia de avance temporal: el tiempo del vehículo se incrementa gradualmente (en intervalos de un minuto) hasta que algún cliente se vuelva accesible o se alcance el límite temporal del vehículo. Si después de este proceso no hay clientes disponibles y el número de vehículos utilizados no excede el máximo permitido, se asigna un nuevo vehículo a la flota. El vehículo actual cierra su ruta regresando al depósito, y el nuevo vehículo inicia desde el depósito con carga y tiempo reiniciados. Si se alcanza el límite máximo de vehículos sin poder atender a todos los clientes, la hormiga se reinicia completamente para construir una nueva solución.

### Proceso de construcción de soluciones

El algoritmo 3.3 presenta el proceso completo de construcción de una solución por parte de una hormiga, integrando la selección probabilística de clientes, la gestión de restricciones y la asignación dinámica de vehículos.

---

**Algorithm 3.3:** Construcción de Solución por una Hormiga
 

---

**Input:** Matrices  $\tau, \eta, \omega$ , parámetros  $\alpha, \beta, \gamma$ , configuración VRP  
**Output:** Solución completa (conjunto de rutas)

- 1 Inicializar lista tabú  $T \leftarrow \{0\}$  (depósito visitado).
- 2 Crear vehículo inicial en el depósito:  $v_1$  con carga  $q = 0$ , tiempo  $t = 0$ .
- 3 posición  $\leftarrow 0$  (depósito), Rutas  $\leftarrow \emptyset$ .
- 4 **mientras**  $|T| <$  número total de clientes **hacer**
- 5   Candidatos  $\leftarrow \emptyset$ .
- 6   **mientras** Candidatos  $= \emptyset$  y  $t <$  límite temporal del depósito **hacer**
- 7     Candidatos  $\leftarrow$  CalcularClientesFactibles(posición,  $v$ ,  $T$ ).
- 8     **si** Candidatos  $= \emptyset$  **entonces**
- 9        $t \leftarrow t + 1$  (*avance temporal*).
- 10     **fin si**
- 11   **fin mientras**
- 12   **si** Candidatos  $= \emptyset$  **entonces**
- 13     Cerrar ruta actual: Agregar retorno al depósito.
- 14     **si** Vehículos usados  $<$  Vehículos máximos **entonces**
- 15       Crear nuevo vehículo en el depósito.
- 16        $q \leftarrow 0, t \leftarrow 0$ , posición  $\leftarrow 0$ .
- 17     **si no**
- 18       Reiniciar hormiga completamente.
- 19     **fin si**
- 20   **si no**
- 21     **para cada** cliente  $j \in$  Candidatos **hacer**
- 22        $p_j \leftarrow \frac{[\tau_{\text{posición},j}]^\alpha \cdot [\eta_{\text{posición},j}]^\beta \cdot [\omega_{\text{posición},j}]^\gamma}{\sum_{l \in \text{Candidatos}} [\tau_{\text{posición},l}]^\alpha \cdot [\eta_{\text{posición},l}]^\beta \cdot [\omega_{\text{posición},l}]^\gamma}.$
- 23     **fin para**
- 24     Seleccionar cliente  $j^*$  según distribución de probabilidad  $\{p_j\}$ .
- 25     Aregar  $j^*$  a ruta actual,  $T \leftarrow T \cup \{j^*\}$ .
- 26     Actualizar:  $q \leftarrow q + d_{j^*}$ ,  $t \leftarrow t + t_{\text{viaje}} + s_{j^*}$ .
- 27     posición  $\leftarrow j^*$ .
- 28   **fin si**
- 29 **fin mientras**
- 30 Cerrar última ruta: Agregar retorno al depósito.
- 31 Retornar Rutas.

---

Figura 3.3: Proceso de construcción probabilística de una solución completa por una hormiga. El algoritmo maneja dinámicamente la asignación de vehículos y el avance temporal cuando no hay clientes factibles disponibles, garantizando el cumplimiento de todas las restricciones del VRP-TW.

### Evaluación de la función objetivo

Una vez construida la solución, se evalúa su calidad calculando la distancia total recorrida por todos los vehículos. El algoritmo 3.4 describe este proceso de evaluación.

---

#### **Algorithm 3.4:** Evaluación de Fitness de una Solución

---

**Input:** Solución (conjunto de rutas), matriz de distancias  $D$   
**Output:** Fitness total de la solución

- 1 FitnessTotal  $\leftarrow 0$ .
- 2 **para cada** vehículo  $v$  en la Solución **hacer**
- 3     FitnessVehículo  $\leftarrow 0$ .
- 4     Ruta  $\leftarrow$  obtener ruta del vehículo  $v$ .
- 5     **para**  $k = 0$  hasta  $|Ruta| - 2$  **hacer**
- 6          $i \leftarrow Ruta[k]$ ,  $j \leftarrow Ruta[k + 1]$ .
- 7         FitnessVehículo  $\leftarrow$  FitnessVehículo  $+ D_{ij}$ .
- 8     **fin para**
- 9      $v.fitness \leftarrow$  FitnessVehículo.
- 10    FitnessTotal  $\leftarrow$  FitnessTotal  $+$  FitnessVehículo.
- 11 **fin para**
- 12 Retornar FitnessTotal.

---

Figura 3.4: Cálculo del fitness de una solución como la suma de las distancias totales recorridas por todos los vehículos. Cada ruta se evalúa sumando las distancias entre clientes consecutivos, incluyendo los trayectos desde y hacia el depósito.

### Actualización de feromonas

Al finalizar cada iteración, todas las hormigas han construido soluciones completas. Las feromonas se actualizan mediante dos mecanismos complementarios que balancean exploración y explotación.

**Evaporación de feromonas:** Reduce uniformemente los niveles de feromona para evitar convergencia prematura y permitir exploración de nuevas rutas:

$$\tau_{ij} \leftarrow (1 - \rho) \cdot \tau_{ij}, \quad \forall(i, j) \quad (3.6)$$

donde  $\rho \in [0, 1]$  es la tasa de evaporación.

**Depósito de feromonas:** Las mejores hormigas (determinadas por un porcentaje  $p_{elite}$  de la población ordenada por fitness) depositan feromona en las aristas que utilizaron:

$$\tau_{ij} \leftarrow \tau_{ij} + \sum_{k \in \text{Elite}} \Delta\tau_{ij}^k \quad (3.7)$$

donde:

$$\Delta\tau_{ij}^k = \begin{cases} \frac{1}{L_k} & \text{si la hormiga } k \text{ usó la arista } (i, j) \\ 0 & \text{en otro caso} \end{cases} \quad (3.8)$$

siendo  $L_k$  el costo total (distancia recorrida) de la solución de la hormiga  $k$ .

**Refuerzo elitista:** Adicionalmente, se aplica un refuerzo extra a la mejor solución global encontrada hasta el momento, multiplicando su contribución de feromonas por un factor de 2:

$$\tau_{ij} \leftarrow \tau_{ij} + 2 \cdot \frac{1}{L_{best}}, \quad \forall (i, j) \in \text{MejorRuta} \quad (3.9)$$

donde  $L_{best}$  es el costo de la mejor solución encontrada.

El algoritmo 3.5 detalla el proceso completo de actualización de la matriz de feromonas.

---

**Algorithm 3.5:** Actualización de Matriz de Feromonas

---

**Input:** Matriz  $\tau$ , conjunto Elite de hormigas, MejorSoluciónGlobal, tasa  $\rho$

**Output:** Matriz  $\tau$  actualizada

- 1 **Fase 1: Evaporación**
- 2 **para**  $i = 0$  hasta  $n - 1$  **hacer**
- 3     **para**  $j = 0$  hasta  $n - 1$  **hacer**
- 4         **si**  $i \neq j$  **entonces**
- 5              $\tau_{ij} \leftarrow (1 - \rho) \cdot \tau_{ij}$ .
- 6         **fin si**
- 7     **fin para**
- 8 **fin para**
- 9 **Fase 2: Depósito por hormigas elite**
- 10 **para cada** hormiga  $k \in \text{Elite}$  **hacer**
- 11      $\Delta\tau \leftarrow 1/\text{Costo}_k$ .
- 12     **para cada** vehículo  $v$  en Solución $_k$  **hacer**
- 13         **para cada** arista  $(i, j)$  en ruta de  $v$  **hacer**
- 14             **si**  $i \neq j$  **entonces**
- 15                  $\tau_{ij} \leftarrow \tau_{ij} + \Delta\tau$ .
- 16                  $\tau_{ji} \leftarrow \tau_{ji} + \Delta\tau$  (*grafo no dirigido*).
- 17             **si no**
- 18                  $\tau_{ii} \leftarrow 0$  (*eliminar autoenlaces*).
- 19             **fin si**
- 20         **fin para**
- 21     **fin para**
- 22 **fin para**
- 23 **Fase 3: Refuerzo elitista**
- 24  $\Delta\tau_{best} \leftarrow 2/\text{CostoMejorGlobal}$ .
- 25 **para cada** vehículo  $v$  en MejorSoluciónGlobal **hacer**
- 26     **para cada** arista  $(i, j)$  en ruta de  $v$  **hacer**
- 27          $\tau_{ij} \leftarrow \tau_{ij} + \Delta\tau_{best}$ .
- 28          $\tau_{ji} \leftarrow \tau_{ji} + \Delta\tau_{best}$ .
- 29     **fin para**
- 30 **fin para**
- 31 Retornar  $\tau$ .

---

Figura 3.5: Mecanismo de actualización de feromonas en tres fases: evaporación uniforme para favorecer exploración, depósito proporcional al fitness de las hormigas elite, y refuerzo adicional en la mejor solución global para intensificar la explotación de rutas prometedoras.

### Criterio de factibilidad

Una solución se considera factible si:

- Todos los clientes son visitados exactamente una vez
- Ninguna ruta excede la capacidad del vehículo
- Todas las ventanas de tiempo se respetan estrictamente
- Cada ruta inicia y termina en el depósito
- El número de vehículos utilizados no excede el máximo disponible

El algoritmo garantiza la factibilidad mediante la construcción de listas de candidatos que verifican todas estas restricciones antes de la selección probabilística.

### **Algoritmo principal ACO integrado**

El algoritmo 3.6 presenta la estructura completa del algoritmo ACO, integrando todas las fases descritas previamente: construcción de soluciones, evaluación, ordenamiento, aplicación de Simulated Annealing a las mejores hormigas, y actualización de feromonas.

---

**Algorithm 3.6:** Optimización por Colonia de Hormigas para VRP-TW
 

---

**Input:** Parámetros  $\alpha, \beta, \gamma, \rho$ , número de hormigas  $m$ , iteraciones  $n_{iter}$ , porcentaje elite  $p_{elite}$

**Output:** Mejor solución de rutas encontrada

- 1 **Inicialización:**
- 2 Inicializar matriz de feromonas  $\tau_{ij} = 1/l_j$  para todo  $i, j$ .
- 3 Calcular matriz de visibilidad  $\eta_{ij} = 1/d_{ij}$ .
- 4 Calcular matriz de información temporal  $\omega_{ij} = 1/l_j$ .
- 5 MejorSoluciónGlobal  $\leftarrow$  NULL, MejorCostoGlobal  $\leftarrow \infty$ .
- 6 **Iteración principal:**
- 7 **para**  $iter = 1$  hasta  $n_{iter}$  **hacer**
- 8   **para**  $k = 1$  hasta  $m$  **hacer**
- 9     Solución $_k \leftarrow$  ConstruirSolución( $\alpha, \beta, \gamma$ ).
- 10    Costo $_k \leftarrow$  EvaluarFitness(Solución $_k$ ).
- 11   **fin para**
- 12   OrdenarHormigas por Costo ascendente.
- 13   Elite  $\leftarrow$  primeras  $\lceil m \times p_{elite} \rceil$  hormigas.
- 14   **para cada** hormiga  $k \in$  Elite **hacer**
- 15     Solución $_k \leftarrow$  AplicarSimulatedAnnealing(Solución $_k$ ).
- 16     Costo $_k \leftarrow$  EvaluarFitness(Solución $_k$ ).
- 17     **si** Costo $_k <$  MejorCostoGlobal **entonces**
- 18       MejorSoluciónGlobal  $\leftarrow$  Solución $_k$ .
- 19       MejorCostoGlobal  $\leftarrow$  Costo $_k$ .
- 20     **fin si**
- 21   **fin para**
- 22   ActualizarFeromonas(Elite, MejorSoluciónGlobal,  $\rho$ ).
- 23   **si**  $iter < n_{iter}$  **entonces**
- 24     ReiniciarTodasHormigas().
- 25   **fin si**
- 26 **fin para**
- 27 Retornar MejorSoluciónGlobal.

---

Figura 3.6: Algoritmo principal de ACO con integración de Simulated Annealing para la resolución del problema VRP-TW. Las hormigas construyen soluciones de forma probabilística, las mejores se optimizan con SA, y se actualiza la matriz de feromonas considerando únicamente las soluciones elite y reforzando la mejor solución global.

### 3.2.4. Refinamiento de rutas mediante Recocido Simulado (SA)

La tercera y última etapa del enfoque híbrido consiste en refinar las soluciones generadas por ACO mediante **Recocido Simulado (SA)**, una metaheurística de búsqueda local que permite escapar de óptimos locales a través de la aceptación probabilística de soluciones que empeoran temporalmente la función objetivo. SA recibe como entrada

las mejores rutas construidas por ACO y aplica operadores de vecindad para explorar sistemáticamente el espacio de soluciones cercanas, buscando mejoras incrementales que reduzcan el costo total de las rutas mientras se mantiene la factibilidad.

### Fundamentos del algoritmo SA

El algoritmo SA está inspirado en el proceso metalúrgico de recocido, donde un metal se calienta a alta temperatura y luego se enfriá gradualmente para alcanzar un estado de mínima energía. En el contexto de optimización, la *temperatura* controla la probabilidad de aceptar soluciones deteriorantes, permitiendo escapar de óptimos locales en etapas tempranas (alta temperatura) y convergiendo hacia soluciones de alta calidad en etapas finales (baja temperatura).

Los componentes principales del algoritmo son:

- **Temperatura inicial ( $T_0$ ):** Valor alto que permite gran exploración inicial, aceptando movimientos deteriorantes con alta probabilidad.
- **Temperatura final ( $T_f$ ):** Valor muy bajo que marca el fin del algoritmo, cuando prácticamente solo se aceptan mejoras. Se establece un umbral mínimo de  $10^{-10}$  para evitar problemas numéricos.
- **Factor de enfriamiento ( $\alpha_{\text{cool}}$ ):** Parámetro  $\in (0, 1)$  que controla la velocidad de descenso de temperatura según  $T_{t+1} = \alpha_{\text{cool}} \cdot T_t$ .
- **Iteraciones por nivel de temperatura ( $n_{\text{iter}}$ ):** Número de iteraciones a realizar en cada nivel de temperatura antes de enfriar, permitiendo una exploración adecuada del vecindario.

### Operadores de vecindad adaptativos

Para explorar el espacio de soluciones, SA implementa un conjunto de ocho operadores de vecindad especializados que modifican las rutas de manera controlada. La selección de operadores se realiza de forma **adaptativa** según dos criterios fundamentales: el tamaño de la instancia (número de clientes) y el número de vehículos en la solución actual. Esta estrategia permite ajustar dinámicamente la exploración del espacio de búsqueda según las características específicas del problema.

**Clasificación de operadores:** Los operadores se dividen en dos categorías principales:

**Operadores intra-ruta:** Modifican la secuencia de clientes dentro de una misma ruta:

- **Swap intra-ruta:** Intercambia las posiciones de dos clientes dentro de la misma ruta.

---

**Algorithm 3.7:** Operador Swap Intra-Ruta
 

---

**Input:** Solución  $s$ , configuración VRP

**Output:** Solución vecina  $s'$  o rechazo del movimiento

- 1 Seleccionar vehículo aleatorio con  $\geq 2$  clientes.
  - 2 Seleccionar dos posiciones distintas  $i, j$  en la ruta.
  - 3 Intercambiar clientes en posiciones  $i$  y  $j$ .
  - 4 **si** verificar\_restricciones( $s'$ ) **entonces** retornar  $s'$  **si no** revertir.
- 

Figura 3.7: Pseudocódigo del operador Swap Intra-Ruta.

- **2-opt:** Invierte un segmento de la ruta, eliminando cruces geométricos.

---

**Algorithm 3.8:** Operador 2-opt
 

---

**Input:** Solución  $s$ , configuración VRP

**Output:** Solución vecina  $s'$  o rechazo del movimiento

- 1 Seleccionar vehículo con  $\geq 3$  clientes.
  - 2 Seleccionar dos posiciones  $k < m$  que definan un segmento.
  - 3 Invertir el orden de clientes en el segmento  $[k, m]$ .
  - 4 **si** verificar\_restricciones( $s'$ ) **entonces** retornar  $s'$  **si no** revertir.
- 

Figura 3.8: Pseudocódigo del operador 2-opt.

- **Or-opt:** Reubica un segmento de 1–3 clientes consecutivos dentro de la misma ruta.

---

**Algorithm 3.9:** Operador Or-opt
 

---

**Input:** Solución  $s$ , configuración VRP

**Output:** Solución vecina  $s'$  o rechazo del movimiento

- 1 Seleccionar vehículo con  $\geq 3$  clientes.
  - 2 Seleccionar segmento de 1–3 clientes consecutivos.
  - 3 Eliminar segmento; Seleccionar nueva posición no solapada.
  - 4 Insertar segmento en nueva posición.
  - 5 **si** verificar\_restricciones( $s'$ ) **entonces** retornar  $s'$  **si no** restaurar.
- 

Figura 3.9: Pseudocódigo del operador Or-opt.

**Operadores inter-ruta:** Transfieren clientes o segmentos entre diferentes rutas:

- **Swap inter-ruta:** Intercambia un cliente de una ruta con un cliente de otra ruta.

---

**Algorithm 3.10:** Operador Swap Inter-Ruta

---

**Input:** Solución  $s$ , configuración VRP

**Output:** Solución vecina  $s'$  o rechazo del movimiento

- 1 Seleccionar dos vehículos distintos con  $\geq 1$  cliente cada uno.
  - 2 Seleccionar posición  $i$  en vehículo 1 y posición  $j$  en vehículo 2.
  - 3 Intercambiar clientes entre ambas posiciones.
  - 4 **si** verificar\_restricciones(vehículo 1) y verificar\_restricciones(vehículo 2)  
**entonces**
  - 5     retornar  $s'$  **si no** revertir.
- 

Figura 3.10: Pseudocódigo del operador Swap Inter-Ruta.

- **Reinserción intra-inter:** Extrae un cliente y lo reinsera en otra posición, ya sea en la misma ruta o en una ruta diferente.

---

**Algorithm 3.11:** Operador Reinserción Intra-Inter

---

**Input:** Solución  $s$ , configuración VRP

**Output:** Solución vecina  $s'$  o rechazo del movimiento

- 1 Seleccionar vehículo origen con  $\geq 1$  cliente.
  - 2 Seleccionar cliente  $c$  y eliminarlo de su posición.
  - 3 Seleccionar vehículo destino y posición de inserción.
  - 4 Insertar cliente  $c$  en nueva posición.
  - 5 **si** verificar\_restricciones(ambos vehículos) **entonces** retornar  $s'$  **si no** revertir.
- 

Figura 3.11: Pseudocódigo del operador Reinserción Intra-Inter.

- **2.5-opt:** Mueve un segmento de 1–2 clientes desde un vehículo origen hacia un vehículo destino.

---

**Algorithm 3.12:** Operador 2.5-opt

---

**Input:** Solución  $s$ , configuración VRP

**Output:** Solución vecina  $s'$  o rechazo del movimiento

- 1 Seleccionar vehículo origen con  $\geq 2$  clientes.
  - 2 Seleccionar segmento de tamaño 1–2 clientes.
  - 3 Eliminar segmento del origen; Seleccionar vehículo destino.
  - 4 Insertar segmento en posición aleatoria del destino.
  - 5 **si** verificar\_restricciones(ambos) **entonces** retornar  $s'$  **si no** restaurar.
- 

Figura 3.12: Pseudocódigo del operador 2.5-opt.

- **Cross-exchange:** Intercambia segmentos de 1–2 clientes entre dos vehículos diferentes.

---

**Algorithm 3.13:** Operador Cross-exchange

---

**Input:** Solución  $s$ , configuración VRP**Output:** Solución vecina  $s'$  o rechazo del movimiento

- 1 Seleccionar dos vehículos distintos con  $\geq 1$  cliente cada uno.
  - 2 Seleccionar segmentos de tamaño 1–2 en cada vehículo.
  - 3 Eliminar ambos segmentos; Intercambiar posiciones.
  - 4 Insertar segmentos en vehículos opuestos.
  - 5 **si** verificar\_restricciones(ambos) **entonces** retornar  $s'$  **si no** restaurar.
- 

Figura 3.13: Pseudocódigo del operador Cross-exchange.

- **Relocate-chain:** Reubica una cadena de 2–4 clientes consecutivos desde un vehículo origen hacia un vehículo destino.
- 

**Algorithm 3.14:** Operador Relocate-chain

---

**Input:** Solución  $s$ , configuración VRP**Output:** Solución vecina  $s'$  o rechazo del movimiento

- 1 Seleccionar vehículo origen con  $\geq 2$  clientes.
  - 2 Seleccionar cadena de 2–4 clientes consecutivos (máx. 50 % de la ruta).
  - 3 Eliminar cadena del origen; Seleccionar vehículo destino.
  - 4 Insertar cadena completa en destino.
  - 5 **si** verificar\_restricciones(ambos) **entonces** retornar  $s'$  **si no** restaurar.
- 

Figura 3.14: Pseudocódigo del operador Relocate-chain.

**Estrategia adaptativa de selección:** La probabilidad de aplicar cada operador se ajusta dinámicamente según tres configuraciones basadas en el tamaño de la instancia:

**Instancias pequeñas ( $n \leq 26$  clientes):**

- *Múltiples vehículos:* Se priorizan operadores inter-ruta intensivos (cross-exchange 25 %, relocate-chain 20 %, swap inter 20 %) para facilitar consolidación de rutas y redistribución de cargas.
- *Un solo vehículo:* Se enfatiza la optimización intra-ruta con relocate-chain (30 %), 2.5-opt (25 %) y reinserción (20 %) para mejorar la secuencia de visitas.

**Instancias medianas ( $26 < n \leq 51$  clientes):**

- *Múltiples vehículos:* Balance entre operadores inter-ruta (cross-exchange 30 %, relocate-chain 25 %, swap inter 20 %) e intra-ruta para mantener diversificación.
- *Un solo vehículo:* Mayor peso en relocate-chain (35 %), 2.5-opt (30 %) y reinserción (20 %) para optimización intensiva de la ruta única.

### Instancias grandes ( $n > 51$ clientes):

- *Múltiples vehículos*: Dominio de relocate-chain (35 %) y cross-exchange (30 %) para manejar la complejidad combinatoria, complementado con swap inter (20 %).
- *Un solo vehículo*: Fuerte énfasis en relocate-chain (40 %) y 2.5-opt (35 %) con operadores adicionales en menor proporción.

### Criterio de aceptación de Metropolis

En cada iteración, SA genera una solución vecina  $s'$  a partir de la solución actual  $s$  aplicando probabilísticamente uno de los ocho operadores de vecindad según la estrategia adaptativa. La decisión de aceptar o rechazar  $s'$  se basa en el **criterio de Metropolis**:

$$\text{Aceptar } s' = \begin{cases} \text{Sí} & \text{si } f(s') < f(s) - \epsilon \text{ (siempre aceptar mejoras)} \\ \text{Sí} & \text{si } \Delta f > \epsilon, T > 10^{-10} \text{ y } r < e^{-\Delta f/T} \text{ (aceptar empeoramientos)} \\ \text{No} & \text{en otro caso} \end{cases} \quad (3.10)$$

donde:

- $\Delta f = f(s') - f(s)$  es la diferencia de costo
- $T$  es la temperatura actual
- $\epsilon = 10^{-6}$  es una tolerancia numérica para comparaciones de punto flotante
- $r$  es un número aleatorio uniforme en  $[0, 1]$

La probabilidad de aceptar un empeoramiento es mayor cuando:

- La temperatura  $T$  es alta (exploración inicial)
- La diferencia de costo  $\Delta f$  es pequeña (empeoramiento leve)

### Esquema de enfriamiento

La temperatura se reduce gradualmente siguiendo un esquema de **enfriamiento geométrico**:

$$T_{t+1} = \alpha_{\text{cool}} \cdot T_t \quad (3.11)$$

con  $\alpha_{\text{cool}} \in [0,95, 0,995]$  según el tamaño de la instancia. En cada nivel de temperatura  $T_t$ , se realizan  $n_{\text{iter}}$  iteraciones antes de enfriar, permitiendo una exploración adecuada del vecindario antes de intensificar la búsqueda.

El algoritmo finaliza cuando la temperatura alcanza  $T_f$  o cuando  $T < 10^{-10}$  (proyección numérica).

### Manejo de factibilidad y limpieza de vehículos

Dado que los operadores de vecindad pueden generar soluciones infactibles que violan restricciones de capacidad o ventanas de tiempo, se implementan mecanismos de validación rigurosos descritos en el algoritmo 3.15.

---

**Algorithm 3.15:** Verificación de Restricciones y Factibilidad

---

**Input:** Vehículo  $v$ , configuración VRP, matriz de distancias  $D$

**Output:** true si factible, false en caso contrario

- 1 **Inicialización:**
- 2  $t \leftarrow 0$  (tiempo acumulado),  $q \leftarrow 0$  (capacidad acumulada).
- 3  $\text{clienteAnterior} \leftarrow \text{NULL}$ .
- 4 **Recorrido de la ruta:**
- 5 **para cada** nodo en  $v.\text{ruta}$  **hacer**
- 6     cliente  $\leftarrow$  nodo.cliente.
- 7     **si** clienteAnterior  $\neq \text{NULL}$  **entonces**
- 8         **si** clienteAnterior  $\neq 0$  **entonces**
- 9              $t \leftarrow t + s_{\text{clienteAnterior}}$  (*tiempo de servicio*).
- 10         **fin si**
- 11          $t \leftarrow t + D_{\text{clienteAnterior}, \text{cliente}} / v.\text{velocidad}$  (*tiempo de viaje*).
- 12         **si**  $t < e_{\text{cliente}}$  **entonces**
- 13              $t \leftarrow e_{\text{cliente}}$  (*esperar hasta ventana inicial*).
- 14         **fin si**
- 15     **si no** (*primer cliente desde depósito*)
- 16         **si** cliente  $\neq 0$  **entonces**
- 17              $t \leftarrow D_{0, \text{cliente}} / v.\text{velocidad}$ .
- 18             **si**  $t < e_{\text{cliente}}$  **entonces**
- 19                  $t \leftarrow e_{\text{cliente}}$ .
- 20             **fin si**
- 21         **fin si**
- 22     **fin si**
- 23     **si** cliente  $\neq 0$  **entonces**
- 24          $q \leftarrow q + d_{\text{cliente}}$  (*acumular demanda*).
- 25         **si**  $t > l_{\text{cliente}}$  **entonces**
- 26             retornar false (*violación ventana de tiempo*).
- 27         **fin si**
- 28     **fin si**
- 29     clienteAnterior  $\leftarrow$  cliente.
- 30 **fin para**
- 31 **Validación de capacidad:**
- 32 **si**  $q > v.Q$  **entonces**
- 33     retornar false (*violación de capacidad*).
- 34 **fin si**
- 35 **Validación de retorno al depósito:**
- 36 **si** clienteAnterior  $\neq 0$  **entonces**
- 37      $t \leftarrow t + s_{\text{clienteAnterior}}$ .
- 38      $t \leftarrow t + D_{\text{clienteAnterior}, 0} / v.\text{velocidad}$ .
- 39     **si**  $t > l_0$  **entonces**
- 40         retornar false (*no puede regresar a tiempo*).
- 41     **fin si**
- 42 **fin si**
- 43 retornar true.

---

Figura 3.15: Algoritmo de verificación completa de restricciones para una ruta. Valida secuencialmente las ventanas de tiempo de cada cliente considerando tiempos de viaje, tiempos de servicio y tiempos de espera. Siempre que se viola alguna restricción, el algoritmo retorna false.

El mecanismo de validación opera en tres niveles:

1. **Validación de capacidad:** Después de aplicar un operador, se verifica que ninguna ruta exceda la capacidad máxima del vehículo  $Q$  mediante la acumulación de demandas:

$$\sum_{i \in \text{ruta}} d_i \leq Q \quad (3.12)$$

2. **Validación temporal:** Se recalculan los tiempos de llegada considerando:

$$t_{\text{llegada}}^j = t_{\text{salida}}^i + \frac{d_{ij}}{v} + s_i \quad (3.13)$$

donde  $t_{\text{inicio}}^j = \max(t_{\text{llegada}}^j, e_j)$  y se verifica que  $t_{\text{inicio}}^j \leq l_j$  para todo cliente  $j$ .

3. **Validación de retorno:** Se asegura que el vehículo pueda regresar al depósito dentro de su ventana de tiempo:

$$t_{\text{último}} + s_{\text{último}} + \frac{d_{\text{último},0}}{v} \leq l_0 \quad (3.14)$$

**Limpieza de vehículos vacíos:** Los operadores inter-ruta pueden dejar vehículos sin clientes. El algoritmo 3.16 describe el proceso de eliminación de vehículos vacíos.

---

**Algorithm 3.16:** Eliminación de Vehículos Vacíos
 

---

**Input:** Flota de vehículos  
**Output:** Número de vehículos activos

- 1 `vehiculos_activos`  $\leftarrow 0$ .
- 2 `vehiculo_actual`  $\leftarrow$  flota.cabeza.
- 3 **mientras** `vehiculo_actual`  $\neq$  NULL **hacer**
- 4     `siguiente`  $\leftarrow$  `vehiculo_actual.siguiente`.
- 5     **si** `vehiculo_actual.clientes_contados` = 0 **entonces**
- 6         *// Vehículo vacío, eliminar de la flota*
- 7         **si** `vehiculo_actual.anterior`  $\neq$  NULL **entonces**
- 8             `vehiculo_actual.anterior.siguiente`  $\leftarrow$  `siguiente`.
- 9         **si no**
- 10             `flota.cabeza`  $\leftarrow$  `siguiente`.
- 11         **fin si**
- 12         **si** `siguiente`  $\neq$  NULL **entonces**
- 13             `siguiente.anterior`  $\leftarrow$  `vehiculo_actual.anterior`.
- 14         **si no**
- 15             `flota.cola`  $\leftarrow$  `vehiculo_actual.anterior`.
- 16         **fin si**
- 17         Liberar memoria de `vehiculo_actual`.
- 18         **si no**
- 19             `vehiculos_activos`  $\leftarrow$  `vehiculos_activos` + 1.
- 20         **fin si**
- 21         `vehiculo_actual`  $\leftarrow$  `siguiente`.
- 22 **fin mientras**
- 23 **retornar** `vehiculos_activos`.

---

Figura 3.16: Proceso de limpieza de vehículos vacíos de la flota. Este procedimiento se ejecuta después de aplicar operadores inter-ruta que pueden transferir todos los clientes de un vehículo, periódicamente cada 20 iteraciones, y al finalizar el algoritmo SA para asegurar una solución compacta.

La limpieza de vehículos se realiza:

- Despues de operadores swap\_inter, reinserción, cross-exchange y relocate-chain
- Periódicamente cada 20 iteraciones
- Al finalizar el algoritmo

Cuando se eliminan vehículos, se re-evalúa la función objetivo completa mediante el algoritmo 3.17.

---

**Algorithm 3.17:** Evaluación de Fitness para SA

---

**Input:** Solución (hormiga), matriz de distancias  $D$

**Output:** Fitness global actualizado

- 1  $\text{hormiga.fitness\_global} \leftarrow 0.$
- 2  $\text{vehiculo\_actual} \leftarrow \text{hormiga.flota.cabeza}.$
- 3 **mientras**  $\text{vehiculo\_actual} \neq \text{NULL}$  **hacer**
- 4      $\text{fitness\_vehiculo} \leftarrow 0.$
- 5      $\text{nodo\_actual} \leftarrow \text{vehiculo\_actual.ruta.cabeza}.$
- 6     **mientras**  $\text{nodo\_actual.siguiente} \neq \text{NULL}$  **hacer**
- 7          $i \leftarrow \text{nodo\_actual.cliente}.$
- 8          $j \leftarrow \text{nodo\_actual.siguiente.cliente}.$
- 9          $\text{fitness\_vehiculo} \leftarrow \text{fitness\_vehiculo} + D_{ij}.$
- 10         $\text{nodo\_actual} \leftarrow \text{nodo\_actual.siguiente}.$
- 11     **fin mientras**
- 12      $\text{vehiculo\_actual.fitness\_vehiculo} \leftarrow \text{fitness\_vehiculo}.$
- 13      $\text{hormiga.fitness\_global} \leftarrow \text{hormiga.fitness\_global} + \text{fitness\_vehiculo}.$
- 14      $\text{vehiculo\_actual} \leftarrow \text{vehiculo\_actual.siguiente}.$
- 15 **fin mientras**
- 16 **retornar**  $\text{hormiga.fitness\_global}.$

---

Figura 3.17: Evaluación de la función objetivo para una solución completa. Recorre todos los vehículos de la flota y suma las distancias entre clientes consecutivos en cada ruta, incluyendo los trayectos desde y hacia el depósito.

### Estructura de datos y gestión de memoria

La implementación utiliza una estructura jerárquica eficiente:

- **Hormiga:** Representa una solución completa con atributos de fitness global, número de vehículos necesarios y una lista enlazada de vehículos (flota).
- **Vehículo:** Contiene una ruta (lista enlazada de clientes), capacidad acumulada, tiempos de salida/llegada, velocidad y fitness individual.
- **Ruta:** Lista doblemente enlazada de nodos que representan clientes, permitiendo inserciones y eliminaciones eficientes en cualquier posición.
- **Gestión de memoria:** En cada iteración se realiza:
  - Copia profunda de la solución actual para generar solución vecina
  - Liberación explícita de memoria al aceptar/rechazar movimientos
  - Protección contra fugas de memoria mediante funciones especializadas

### Algoritmo principal de Recocido Simulado

El algoritmo 3.18 presenta la estructura completa del proceso de refinamiento mediante SA.

**Algorithm 3.18:** Recocido Simulado para Refinamiento de Rutas VRP-TW

---

**Input:** Solución inicial  $s_0$  (de ACO),  $T_0$ ,  $T_f$ ,  $\alpha_{\text{cool}}$ ,  $n_{\text{iter}}$

**Output:** Solución refinada  $s^*$

**1 Inicialización:**

- 2  $s_{\text{vecina}} \leftarrow \text{copiar}(s_0)$ .
- 3  $s_{\text{actual}} \leftarrow \text{copiar}(s_0)$ .
- 4  $s^* \leftarrow \text{copiar}(s_0)$ .
- 5  $T \leftarrow T_0$ .

**6 Ciclo de enfriamiento:**

- 7 **mientras**  $T > T_f$  y  $T > 10^{-10}$  **hacer**
- 8     **para**  $iter = 1$  **hasta**  $n_{\text{iter}}$  **hacer**
- 9         Liberar( $s_{\text{vecina}}.\text{flota}$ ).
- 10          $s_{\text{vecina}} \leftarrow \text{copiar}(s_{\text{actual}})$ .
- 11         *// Selección adaptativa de operador*
- 12          $r \leftarrow \text{aleatorio}[0, 1]$ .
- 13         operador  $\leftarrow \text{SeleccionarOperador}(r, \text{instancia.tamaño}, s_{\text{vecina}}.\text{vehículos})$ .
- 14         aceptado  $\leftarrow \text{AplicarOperador}(\text{operador}, s_{\text{vecina}})$ .
- 15         **si** no aceptado **entonces** continuar.
- 16         *// Limpieza de vehículos vacíos (condicional)*
- 17         **si** operador  $\in \{\text{swap\_inter, reinserción, cross-exchange, relocate-chain}\}$
- 18             **entonces**
- 19                  $v_{\text{antes}} \leftarrow s_{\text{vecina}}.\text{vehiculos\_necesarios}$ .
- 20                  $s_{\text{vecina}}.\text{vehiculos\_necesarios} \leftarrow \text{EliminarVehiculosVacíos}(s_{\text{vecina}}.\text{flota})$ .
- 21                 **si**  $v_{\text{antes}} \neq s_{\text{vecina}}.\text{vehiculos\_necesarios}$  **entonces**
- 22                     EvaluarFitness( $s_{\text{vecina}}$ ).
- 23                 **fin si**
- 24             **fin si**
- 25             *// Limpieza periódica*
- 26             **si**  $iter \bmod 20 = 0$  **entonces**
- 27                 EliminarVehiculosVacíos( $s_{\text{vecina}}.\text{flota}$ ).
- 28                 **fin si**
- 29             *// Evaluación de fitness*
- 30             EvaluarFitness( $s_{\text{vecina}}$ ).
- 31              $\Delta f \leftarrow s_{\text{vecina}}.\text{fitness} - s_{\text{actual}}.\text{fitness}$ .
- 32             *// Criterio de aceptación de Metropolis*
- 33             aceptar  $\leftarrow \text{false}$ .
- 34             **si**  $\Delta f < -10^{-6}$  **entonces**
- 35                 aceptar  $\leftarrow \text{true (mejora)}$ .
- 36             **si no, si**  $\Delta f > 10^{-6}$  y  $T > 10^{-10}$  **entonces**
- 37                  $p \leftarrow e^{-\Delta f/T}$ .
- 38                 **si**  $\text{aleatorio}[0, 1] < p$  **entonces**
- 39                     aceptar  $\leftarrow \text{true (aceptar empeoramiento)}$ .
- 40                 **fin si**
- 41             **fin si**
- 42             *// Actualización de soluciones*
- 43             **si** aceptar **entonces**
- 44                 Liberar( $s_{\text{actual}}.\text{flota}$ ).
- 45                  $s_{\text{actual}} \leftarrow \text{copiar}(s_{\text{vecina}})$ .
- 46             **si**  $s_{\text{vecina}}.\text{fitness} \leq s^*.\text{fitness} - 10^{-6}$  **entonces**



# **Capítulo 4**

## **Resultados**



# **Capítulo 5**

## **Conclusiones**



# Bibliografía

- [1] Emile Aarts and Jan K Lenstra. *Local Search in Combinatorial Optimization*. John Wiley & Sons, 1988.
- [2] Orlando Antonio Suárez. Una aproximación a la heurística y metaheurísticas. *INGE@UAN – Tendencias en la Ingeniería*, 1(2), 2014.
- [3] Thomas Bäck, David B. Fogel, and Zbigniew Michalewicz, editors. *Handbook of Evolutionary Computation*. Oxford University Press and Institute of Physics Publishing, New York, NY, 1997.
- [4] Roberto Baldacci, Aristide Mingozzi, and Roberto Roberti. Recent exact algorithms for solving the vehicle routing problem under capacity and time window constraints. *European Journal of Operational Research*, 218(1):1–6, 2012.
- [5] Christian Blum and Andrea Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, 35(3):268–308, 2003.
- [6] Olli Bräysy and Michel Gendreau. Vehicle routing problem with time windows, part i: Route construction and local search algorithms. *Transportation Science*, 39(1):104–118, 2005.
- [7] Olli Bräysy and Michel Gendreau. Vehicle routing problem with time windows, part ii: Metaheuristics. *Transportation Science*, 39(1):119–139, 2005.
- [8] Olli Bräysy and Michel Gendreau. Vehicle routing problem with time windows, part i: Route construction and local search algorithms. *Transportation Science*, 39(1):104–118, 2005.
- [9] G Clarke and J W Wright. Scheduling of vehicles from a central depot to a number of delivery points. *Operations research*, 12(4):568–581, 1964.
- [10] Carlos A. Coello Coello. Metaheurísticas bio-inspiradas para optimización: La última frontera. <https://www.math.cinvestav.mx/sites/default/files/platica-escuela-2022-Coello.pdf>, 2022. Presentación en la Escuela de Verano de Matemáticas, CINVESTAV-IPN.

- [11] Jean-François Cordeau and Gilbert Laporte. A unified tabu search heuristic for vehicle routing problems with time windows. *Journal of the operational research society*, 53(5):528–536, 2002.
- [12] George B Dantzig, D Ray Fulkerson, and Selmer M Johnson. Solution of a large-scale traveling-salesman problem. *Journal of the Operations Research Society of America*, 2(4):393–410, 1954.
- [13] George B Dantzig and John H Ramser. The truck dispatching problem. *Management science*, 6(1):80–91, 1959.
- [14] Julio Mario Daza, Jairo R. Montoya, and Francesco Narducci. Resolución del problema de enrutamiento de vehículos con limitaciones de capacidad utilizando un procedimiento metaheurístico de dos fases. *Revista EIA*, 0(12):23–38, diciembre 2009.
- [15] Sergio Gerardo de los Cobos Silva, John Goddard Close, Miguel Ángel Gutiérrez Andrade, and Alma Edith Martínez Licona. *Búsqueda y exploración estocástica*. Libros CBI, Universidad Autónoma Metropolitana Iztapalapa, México, 2010.
- [16] Martin Desrochers, Jacques Desrosiers, and Marius M Solomon. A new optimization algorithm for the vehicle routing problem with time windows. *Operations research*, 40(2):342–354, 1992.
- [17] Jacques Desrosiers, Yvan Dumas, Marius M Solomon, and François Soumis. Time constrained routing and scheduling. In *Handbooks in Operations Research and Management Science*, volume 8, pages 35–139. Elsevier, 1995.
- [18] Marco Dorigo. *Optimization, Learning and Natural Algorithms*. Phd thesis, Politecnico di Milano, Milán, Italia, 1992.
- [19] Marco Dorigo and Luca Maria Gambardella. Ant colonies for the travelling salesman problem. *BioSystems*, 43(2):73–81, 1997.
- [20] A. E. Eiben and J. E. Smith. *Introduction to Evolutionary Computing*. Springer, Berlin, Heidelberg, 2003.
- [21] Burak Eksioglu, Arif Volkan Vural, and Arnold Reisman. The vehicle routing problem: A taxonomic review. *Computers & Industrial Engineering*, 57(4):1472–1483, 2009.
- [22] Xavier Alejandro Flores Cabezas. Problemas del milenio: P vs np. *Revista de Divulgación Amarun*, 1:1–15, 2014.
- [23] Luca Maria Gambardella, Éric Taillard, and Giovanni Agazzi. Macs-vrptw: A multiple ant colony system for vehicle routing problems with time windows. In *New Ideas in Optimization*, pages 63–76. McGraw-Hill, 1999.
- [24] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

- [25] David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading, Massachusetts, 1989.
- [26] Bruce L. Golden and Arjang A. Assad. Vehicle routing: Methods and studies. In *Studies in Management Science and Systems*. North-Holland, 1988.
- [27] Michael Held and Richard M Karp. A dynamic programming approach to sequencing problems. *Journal of the Society for Industrial and Applied Mathematics*, 10(1):196–210, 1962.
- [28] John H Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.
- [29] Jörg Homberger and Hermann Gehring. Two evolutionary metaheuristics for the vehicle routing problem with time windows. *INFOR: Information Systems and Operational Research*, 37(3):297–318, 1999.
- [30] Scott Kirkpatrick, C. Daniel Gelatt, and Mario P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.
- [31] Scott Kirkpatrick, C. Daniel Gelatt Jr., and Mario P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.
- [32] Gilbert Laporte. The vehicle routing problem: An overview of exact and approximate algorithms. *European Journal of Operational Research*, 59(3):345–358, 1992.
- [33] Gilbert Laporte. The vehicle routing problem: An overview of exact and approximate algorithms. *European Journal of Operational Research*, 59(3):345–358, 1992.
- [34] Gilbert Laporte and Yves Nobert. Exact algorithms for the vehicle routing problem. *Annals of Discrete Mathematics*, 31:147–184, 1987.
- [35] Erasmo López, Óscar Salas, and Álex Murillo. El problema del agente viajero: Un algoritmo determinístico usando búsqueda tabú. *Revista de Matemática: Teoría y Aplicaciones*, 21(1):127–144, 2014.
- [36] Armin Lüer, Magdalena Benavente, Jaime Bustos, and Bárbara Venegas. El problema de rutas de vehículos: Extensiones y métodos de resolución, estado del arte. In *Workshop Internacional EIG2009*, Temuco, Chile, 2009. Universidad de La Frontera, Departamento de Ingeniería de Sistemas.
- [37] Carlos Maldonado. Un problema fundamental en la investigación: Los problemas p vs. np. *Revista Logos, Ciencia & Tecnología*, 4(2):10–20, 2013.
- [38] Abraham Duarte Muñoz, Juan José Pantrigo Fernández, and Micael Gallego Carrillo. *Metaheurísticas*. Dykinson, Madrid, España, 2007.
- [39] Yuichi Nagata and Olli Bräysy. A powerful route minimization heuristic for the vehicle routing problem with time windows. *Operations Research Letters*, 37(5):333–338, 2009.

- [40] Numerentur. Gráfico de complejidad computacional. <https://numerentur.org/wp-content/uploads/2019/08/grafico-complejidad-computacionala3.png>, 2019. Imagen descargada el 15 de julio de 2025.
- [41] Alfredo Olivera. Heurísticas para problemas de ruteo de vehículos. Technical report, Instituto de Computación, Facultad de Ingeniería, Universidad de la República, Montevideo, Uruguay, agosto 2004. Monografía/Informe Técnico.
- [42] Edwin Montes Orozco. Metaheurísticas para el problema de ruteo de vehículos con ventanas de tiempo (vrp-tw), 2017.
- [43] Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.
- [44] Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Dover Publications, New York, 1998.
- [45] David Pisinger and Stefan Ropke. A general heuristic for vehicle routing problems. *Computers & Operations Research*, 34(8):2403–2435, 2007.
- [46] Jean-Yves Potvin and Samy Bengio. The vehicle routing problem with time windows part ii: Genetic search. *INFORMS Journal on Computing*, 8(2):165–172, 1996.
- [47] Kenneth Price, Rainer Storn, and Jouni A. Lampinen. *Differential Evolution: A Practical Approach to Global Optimization*. Springer, Berlin, Heidelberg, 2005.
- [48] Marius M. Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, 35(2):254–265, 1987.
- [49] Rainer Storn and Kenneth Price. Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4):341–359, 1997.
- [50] Éric Taillard, Philippe Badeau, Michel Gendreau, François Guertin, and Jean-Yves Potvin. A tabu search heuristic for the vehicle routing problem with soft time windows. *Transportation Science*, 31(2):170–186, 1997.
- [51] J. Torres-Jiménez, A. Aguilar-Meléndez, and M. G. Cedillo-Campos. El problema del agente viajero con restricciones de ventanas de tiempo: revisión y clasificación de métodos de solución. *Revista Iberoamericana de Automática e Informática Industrial*, 15(4):440–452, 2018.
- [52] Paolo Toth and Daniele Vigo. *The Vehicle Routing Problem*. Society for Industrial and Applied Mathematics, 2002.
- [53] Paolo Toth and Daniele Vigo. *Vehicle Routing: Problems, Methods, and Applications*. SIAM (Society for Industrial and Applied Mathematics), 2nd edition, 2014.

- [54] Jhonny Vargas Paredes and Víctor Penit Granado. Estudio y aplicación de metaheurísticas y comparación con métodos exhaustivos, 2016. Disponible en: <https://docta.ucm.es/rest/api/core/bitstreams/69240770-849a-4014-a672-6fad30912f88/content>.
- [55] Thibaut Vidal, Teodor Gabriel Crainic, Michel Gendreau, and Christian Prins. A hybrid genetic algorithm with adaptive diversity management for a large class of vehicle routing problems with time-windows. *Computers & Operations Research*, 40(1):475–489, 2012.