Memoria Descriptiva
Routing App
Aplicación para gestión de rutas TSP

Gabriela Dayana Coaquira Suyo
Rafael Diego Nina Calizaya
Dennis Javier Quispe Saavedra
Christian Henry Venero Guevara
Alexander Villafuerte Quispe

Universidad San Agustín, Arequipa
Tecnología de objetos

03 de diciembre de 2025

Índice

1. Introducción....4
1.1. Objetivos....4
1.1.1. Objetivo general....4
1.1.2. Objetivos específicos....4
2. Marco Teórico....4
2.1. Pathfinding Algorithms....4
2.1.1. Dijkstra Algorithm:....5
2.1.2. A* Algorithm:....5
2.1.3. ALT Algorithm:....5
2.2. TSM Metaheuristics....5
2.2.1. Iterated Greedy (IG)....6
2.2.2. IG + Simulated Annealing (IGSA)....6
2.2.3. Iterated Local Search (ILS)....6
3. Análisis y Diseño....6
3.1. Requisitos funcionales....6
3.2. Requisitos no funcionales....6
3.3. Arquitectura del Sistema....7
3.3.1. Capa de Entidades Core:....7
3.3.2. Capa de Algoritmos:....7
3.3.3. Capa de Servicios:....7
3.3.4. Capa de Interfaz (UI):....7
3.4. Patrones de Diseño....8
3.4.1. Factory Method:....8
3.4.2. Strategy Pattern:....8
3.4.3. Template Method:....8
3.4.4. Model-View-Controller (MVC):....8
4. Implementación....8
4.1. Stack tecnológico....8
4.2. Optimizaciones de rendimiento....9
4.2.1. Serialización binaria....9
4.2.2. Inicialización perezosa....9
4.3. Diseño de la interfaz de usuario....10
4.3.1. Arquitectura general....10
4.3.2. MainWindow, controlador central....10
4.3.3. Control panel, configuración y controles:....10
5. Pruebas y validación....11
5.1. Tests unitarios básicos (GoogleTest):....11
6. Resultados....11
6.1. Test Environment y Dataset....11
6.2. Rendimiento ruta corta....11

6.3. Rendimiento TSP..... 12
7. Conclusiones..... 13
8. Trabajos futuros..... 13
9. Referencias..... 13
A. Apéndice A: Código fuente relevante..... 14

1. Introducción

En la era digital moderna, los sistemas de enrutamiento geográfico son críticos para la eficiencia logística. En ciudades intermedias como Arequipa, el crecimiento desordenado del parque automotor ha incrementado los tiempos de traslado,
afectando la productividad de las micro y pequeñas empresas MYPES. Estas organizaciones requieren herramientas que no solo encuentren la ruta más corta entre dos puntos, sino que optimicen secuencias complejas de entrega.
Este proyecto presenta una solución de software que integra algoritmos clásicos y heurísticas avanzadas sobre la red vial real de la ciudad. A diferencia de soluciones comerciales, nuestra propuesta permite la personalización de perfiles de
vehículo y funciona de manera offline, evaluando el desempeño de técnicas de optimización computacional frente a las limitaciones de infraestructura tradicionales.

1.1. Objetivos

1.1.1. Objetivo general

Desarrollar un sistema de enrutamiento geográfico offline que optimice rutas y secuencias de entrega en la ciudad de Arequipa para las micro y pequeñas empresas (MYPES).

1.1.2. Objetivos específicos

- Implementar algoritmos clásicos de búsqueda de rutas adaptados a la red vial real de Arequipa.
- Integrar heurísticas y técnicas avanzadas de optimización para resolver secuencias complejas de entrega en escenarios tipo TSP.
- Diseñar un módulo de personalización de perfiles de vehículo, que permita ajustar parámetros como velocidad, restricciones viales, capacidad de carga y preferencias de ruta.
- Construir una base de datos vial que funcione completamente offline, representando con precisión las calles y distancias del entorno urbano de Arequipa.
- Proporcionar una interfaz de usuario intuitiva, orientada a pequeñas empresas.

2. Marco Teórico

2.1. Pathfinding Algorithms

Los algoritmos de búsqueda de rutas (pathfinding algorithms) permiten encontrar el camino óptimo entre dos puntos dentro de un grafo ponderado. Estas técnicas son fundamentales en sistemas de navegación, logística urbana y aplicaciones que representan redes de transporte mediante grafos. La eficiencia de un algoritmo de

búsqueda determina la rapidez con la que un sistema puede calcular rutas, especialmente cuando trabaja con grafos de gran tamaño como los utilizados en esta aplicación. Algunos de los destacados son:

2.1.1. Dijkstra Algorithm:

El algoritmo de Dijkstra, propuesto en 1959 por Edsger Dijkstra, es uno de los métodos más influyentes para resolver el problema del camino más corto con pesos no negativos. Su principio consiste en expandir iterativamente el nodo de menor costo acumulado, actualizando las distancias mínimas hacia sus vecinos. Es óptimo y exacto, pero su rendimiento disminuye cuando el grafo es muy grande, ya que puede explorar un número considerable de nodos antes de alcanzar la meta.

2.1.2. A* Algorithm:

Hart, Nilsson y Raphael introdujeron en 1968 el algoritmo A*, una extensión heurística de Dijkstra que incorpora una función de estimación del costo restante (heurística). A* permite guiar la búsqueda hacia el objetivo, reduciendo la cantidad de nodos explorados y acelerando notablemente el cálculo de rutas. Siempre que la heurística sea admisible, A* garantiza una solución óptima.

2.1.3. ALT Algorithm:

Las técnicas de aceleración basadas en puntos de referencia, como ALT (A* + Landmarks + Triangle Inequality), fueron presentadas por Goldberg y Harrelson en 2005. ALT utiliza un conjunto de landmarks preseleccionados y la desigualdad triangular para producir heurísticas más fuertes que las geométricas tradicionales. Esto reduce la exploración innecesaria y mejora significativamente el rendimiento de A* en grafos grandes. Técnicas complementarias como Contraction Hierarchies también mejoran el rendimiento mediante preprocesamiento y reducción de nodos, aunque no forman parte directa de esta implementación.

2.2. TSM Metaheuristics

Las metaheurísticas aplicadas al Problema del Viajante (TSP) permiten obtener soluciones de buena calidad ante un problema NP-duro cuya complejidad crece factorialmente. En lugar de buscar la solución exacta, estas técnicas exploran el espacio de soluciones mediante procesos iterativos de perturbación, reconstrucción y búsqueda local.

2.2.1. Iterated Greedy (IG)

El método Iterated Greedy (IG) fue introducido por Ruiz y Stützle en 2007. Consiste en eliminar parcialmente una solución, reconstruirla mediante reglas deterministas y aplicar luego una búsqueda local. Su simplicidad y buen rendimiento lo hacen adecuado para instancias medianas del TSP, donde el tiempo de respuesta debe mantenerse bajo.

2.2.2. IG + Simulated Annealing (IGSA)

La variante IGSA combina IG con el criterio probabilístico del Simulated Annealing, técnica propuesta por Kirkpatrick et al. en 1983. Esto permite aceptar soluciones peores con cierta probabilidad, favoreciendo la exploración global y reduciendo el riesgo de quedar atrapado en óptimos locales. El README menciona esta variante como una de las utilizadas para mejorar la robustez del módulo TSP.

2.2.3. Iterated Local Search (ILS)

Iterated Local Search (ILS), formalizado por Lourenço, Martin y Stützle en 2003, consiste en aplicar una búsqueda local intensiva y luego perturbar la solución para escapar de los óptimos locales. Este proceso iterativo permite un equilibrio entre intensificación y diversificación. Debido a su simplicidad conceptual y rendimiento consistente, es una de las metaheurísticas estándar en la literatura del TSP.

3. Análisis y Diseño

3.1. Requisitos funcionales

1. El sistema debe cargar grafos urbanos de hasta 50 000 nodos y 60 000 aristas manteniendo un uso eficiente de memoria.
2. El sistema debe calcular rutas óptimas en menos de 100 ms para grafos urbanos típicos de 50 000 nodos.
3. El sistema debe resolver secuencias de 10 a 20 puntos de entrega (TSP) en menos de 5 segundos, dependiendo de la densidad del grafo.
4. El sistema debe proporcionar una UI fluida con 60 FPS, incluyendo funcionalidades de zoom y desplazamiento del mapa.

3.2. Requisitos no funcionales

1. El sistema no debe exceder 500 MB de memoria RAM al procesar grafos de hasta 50 000 nodos.
2. El sistema debe poder compilarse y ejecutarse sin modificaciones en Windows, macOS y Linux.
3. La arquitectura del software debe ser modular, con menos del 15 % de duplicación de código, favoreciendo la extensibilidad y la legibilidad.

4. La lógica crítica debe contar con una cobertura de pruebas de al menos 80 % mediante tests automatizados.

3.3. Arquitectura del Sistema

El diseño desacopla la lógica de negocio de la interfaz mediante cuatro capas principales:

3.3.1. Capa de Entidades Core:

Define los objetos inmutables como Node, Edge (con pesos y metadatos de OSM) y Graph. Se utiliza std::unique_ptr para la gestión semántica de la memoria del grafo.

3.3.2. Capa de Algoritmos:

Implementa la lógica matemática (Dijkstra, IG, IGSA, etc) independientemente de la UI.

3.3.3. Capa de Servicios:

Gestiona la ejecución asíncrona de tareas pesadas (carga de mapas, cálculo de matrices) utilizando hilos de ejecución (thread pool).

3.3.4. Capa de Interfaz (UI):

Desarrollada en Qt6, se encarga de la visualización e interacción con el usuario.

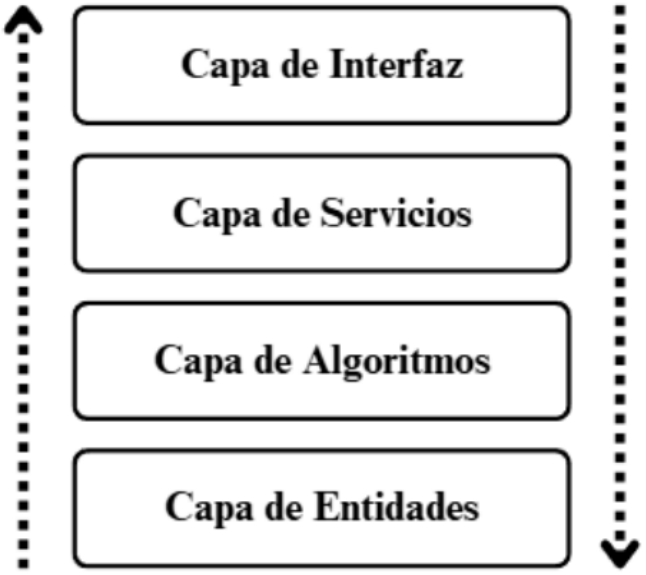

Figura 1: Diagrama de la arquitectura del sistema

3.4. Patrones de Diseño

Para mejorar el rendimiento del sistema y permitir pruebas más exigentes con mapas detallados, se aplicaron técnicas de ingeniería de software.

3.4.1. Factory Method:

Se implementó a través de las clases AlgorithmFactory, TspAlgorithmFactory y VehicleProfileFactory. Este patrón permite la instanciación dinámica de algoritmos (Dijkstra, A*, ALT), metaheuristicas (IG, IGSA, ILSB, IGN) y perfiles de vehículo (Automóvil, Peatón) basándose en la selección del usuario en tiempo de ejecución. Esto desacopla la lógica de la interfaz, permitiendo agregar nuevos perfiles o heurísticas en el futuro sin modificar la capa de servicios.

3.4.2. Strategy Pattern:

Se definió la interfaz IPathfindingAlgorithm como una estrategia base, permitiendo que el PathfindingService intercambie el comportamiento de búsqueda en tiempo real. Gracias a esto, el sistema puede alternar entre una búsqueda exhaustiva (Dijkstra) y una heurística (IG o IGSA para TSP) de manera transparente.

3.4.3. Template Method:

Dado que algoritmos como Dijkstra y A* comparten pasos estructurales (inicialización de estructuras, ciclo de exploración y reconstrucción del camino), se definió un esqueleto común en la clase abstracta. Las subclases solo sobrescriben los "hooks" específico.

3.4.4. Model-View-Controller (MVC):

Implementado implícitamente mediante la arquitectura de Qt6. La capa de Vista (MapWidget, MainWindow) está totalmente separada de la capa del Modelo (Graph, Node) y de los Controladores (Services). La comunicación entre estas capas se realiza mediante el sistema de Señales y Slots de Qt, garantizando que el procesamiento pesado de los algoritmos no congele la interfaz de usuario.

4. Implementación

4.1. Stack tecnológico

Se eligió C++17 por las siguientes razones: permite una gestión de memoria manual y precisa, crítico para grafos de más de 50k nodos, ofrece semántica de movimiento para evitar copias innecesarias de datos y posee una biblioteca estándar robusta para concurrencia.

Qt6 proporciona un framework robusto para la interfaz gráfica, facilitando la creación de componentes interactivos como el visor de mapas y gestionando la comunicación entre hilos mediante su sistema de Señales y Slots.

Componente	Tecnología	Versión	Propósito
Language	C++	17	Eficiencia y POO
UI Framework	Qt	6.10	Herramientas y compatibilidad con C++
Build System	CMake	3.16	Compatibilidad multiplataforma con C++ y Qt
Data	OSM	1.0	Enorme cantidad de datos de rutas

Tabla 1: Stack Tecnológico

4.2. Optimizaciones de rendimiento

Para mejorar el rendimiento del sistema y permitir pruebas más exigentes con mapas detallados, se aplicaron técnicas de ingeniería de software.

4.2.1. Serialización binaria

Reduce el tiempo de carga drásticamente. El parseo de XML es costoso computacionalmente; por ello, tras la primera lectura, el grafo se vuelca a un archivo binario plano con mapeo directo a memoria.

Formato	Tamaño (Mb)	Tiempo de carga (s)	Speedup
OSM	222	48	1x
Binary	44	3	16

Tabla 2: Comparación de tiempo de carga

4.2.2. Inicialización perezosa

Dijkstra tradicionalmente requiere inicializar distancias a infinito para todos los nodos (O(V)). Nuestra implementación solo inicializa estructuras para los nodos explorados durante la consulta actual utilizando un visited token o hash maps. Esto reduce la complejidad espacial y temporal, especialmente en consultas de corta distancia dentro de un grafo grande.

4.3. Diseño de la interfaz de usuario

4.3.1. Arquitectura general

La interfaz gráfica de usuario (GUI) del sistema de ruteo está desarrollada en Qt 6 y sigue una arquitectura modular con separación clara de responsabilidades. La ventana principal (MainWindow) coordina tres componentes principales.

La ventana utiliza QSplitter para dividir el espacio:

- Splitter horizontal: Separa el ControlPanel (izquierda) y el MapWidget (derecha)
- Splitter vertical: Separa la vista principal (arriba) del ResultsPanel (abajo)

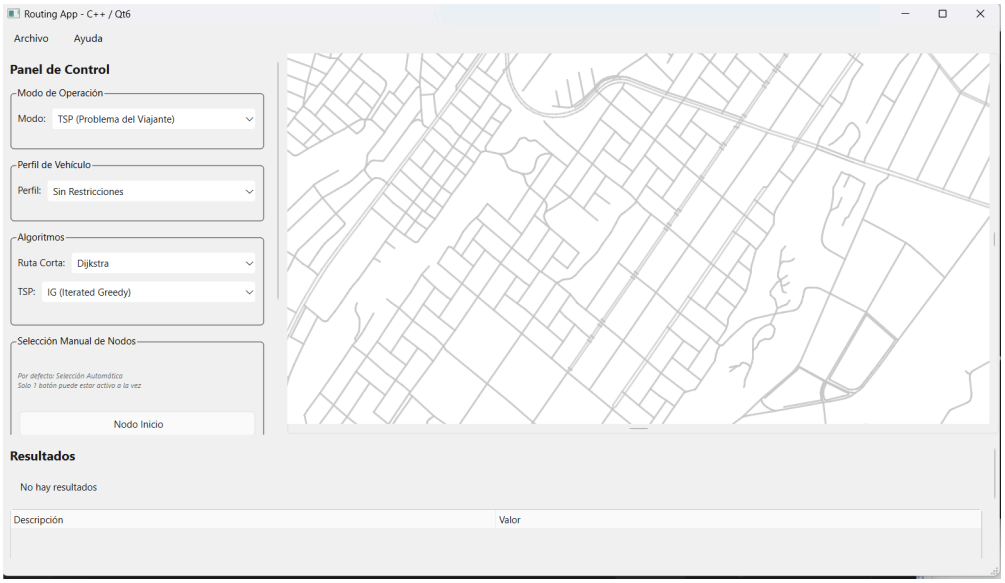

Figura 2: Ventana principal

4.3.2. MainWindow, controlador central

MainWindow coordina la comunicación entre componentes UI y servicios de negocio, actuando como mediador central. Configuración del layout:

- Panel de control con ancho fijo no redimensionable.
- Mapa ocupa todo el espacio horizontal restante.
- Splitters permiten ajuste manual de proporciones verticales/horizontales.

4.3.3. Control panel, configuración y controles:

Panel de configuración ubicado a la izquierda con scroll vertical automático. Componentes de configuración

5. Pruebas y validación

5.1. Tests unitarios básicos (GoogleTest):

- **GraphTest:** Operaciones básicas de grafo (add, get, adjacency)
- **DijkstraTest:** Pathfinding simple en grafo pequeño
- **TspMatrixTest:** Construcción de matriz básica.
- **TestRunner:** Test principal que evalúe la ejecución de los algoritmos implementados.

6. Resultados

6.1. Test Environment y Dataset

Para la evaluación de resultados de los algoritmos de Pathfinding y TSP, se utilizó un entorno de pruebas controlado, se utilizó un grafo real del mapa de la ciudad de Arequipa, Perú. La tabla a continuación muestra los valores aproximados que se usaron para realizar estas pruebas.

Métrica	**Valor**
Total de Nodos	30 258
Total de Aristas	71 320
N° de repeticiones	100 para Pathfinding, 5 para TSP

Tabla 3: Estadísticas Del Dataset

6.2. Rendimiento ruta corta

Para esta sección, se evaluó la eficiencia de los algoritmos de Pathfinding, es decir, Dijkstra y A*, para encontrar la ruta más corta entre 100 pares de nodos creados aleatoriamente. Los resultados promedio obtenidos se muestran en la siguiente tabla.

Algoritmo	**Tiempo promedio (s)**	**Distancia promedio (km)**
Dijkstra	0.0053	5.8
A*	0.0039	5.8
ALT	0.0007	5.8

Tabla 4: Rendimiento ruta corta

Estos resultados demuestran que, si bien el algoritmo A* es más rápido que Dijkstra, ambos son superados por el algoritmo ALT, con un tiempo promedio de consulta de solo 0.0007 s. Este rendimiento se debe a un preprocesamiento intensivo, donde se seleccionan hitos en el grafo. Aunque la distancia promedio sea la misma para los tres algoritmos, demostrando que los tres son óptimos, se debe tener en cuenta su velocidad al momento de elegir la heurística adecuada.

6.3. Rendimiento TSP

Para esta sección, se evaluó la eficiencia de los algoritmos meta-heurísticos: IG, IGSA, IGN y ILSBA, en el problema del viajante, con un conjunto de 15 nodos y probando los diferentes algoritmos de Pathfinding. Para esta pruebas se realizaron 5 repeticiones para cada algoritmo, los resultados promedios se muestran en las siguientes tablas:

Algoritmo TSP	**Dijkstra (km)**	**A* (km)**	**ALT (km)**
IG	11.2	11.2	11.2
IGSA	10.9	10.9	10.9
ILS_B	10.8	10.8	10.8

Tabla 5: Rendimiento ruta corta - Distancia

Algoritmo TSP	**Dijkstra (km)**	**A* (km)**	**ALT (km)**
IG	1.255	0.885	0.555
IGSA	1.280	0.910	0.580
ILS_B	1.315	0.945	0.615

Tabla 6: Rendimiento ruta corta - Tiempo

Los resultados de la Tabla V muestran que la calidad de la distancia final depende únicamente del algoritmo de optimización TSP usado, ya que los algoritmos de Pathfinding son igualmente óptimos. En este análisis, ILS_B es el algoritmo mejor optimizado, lo que prueba que su estrategia de exploración rigurosa y su capacidad de escapar de óptimos locales, son cruciales para obtener la mejor ruta, independientemente de qué tan rápido se haya calculado la matriz inicial.
En cuanto a la Tabla VI, sus resultados demuestran que el factor dominante en el tiempo de solución del TSP es la fase de pre-cálculo de la matriz de distancias. La solución es significativamente más rápida al usar el algoritmo ALT en comparación

con Dijkstra. Esto se debe a la eficiencia de ALT al calcular las N x N rutas necesarias para la matriz.

Tras estas pruebas, se puede concluir que la mejor combinación para el usuario es utilizar el algoritmo ALT para el pre-cálculo e ILS_B para la optimización, obteniendo así la solución de mayor calidad en el menor tiempo total posible.

7. Conclusiones

Este trabajo presentó un sistema completo de enrutamiento geográfico adaptado a la realidad de Arequipa. Junto con ello llegamos a las siguientes conclusiones:

1. La arquitectura en capas y el uso de C++ permitieron gestionar eficientemente grafos urbanos densos, superando las limitaciones de rendimiento típicas de implementaciones básicas.
2. La implementación de la serialización binaria fue fundamental para la experiencia de usuario, reduciendo los tiempos de espera inicial en un 94%.
3. Las metaheurísticas implementadas (IG, IGSA, etc) demostraron ser efectivas para resolver el TSP en tiempos operativos viables para una MYPE, ofreciendo un balance óptimo entre calidad de ruta y tiempo de cálculo.
4. La mejor combinación para el usuario final es utilizar el algoritmo ALT para el pre-cálculo e ILS_B para la optimización.

8. Trabajos futuros

Las siguientes mejoras se proponen como trabajo futuro:

1. Implementación de otros algoritmos de ruta corta como CH o CCH.
2. Incorporación de metaheurísticas adicionales.
3. Mejorar el rendimiento de la interfaz del programa.
4. Implementar una funcionalidad de tráfico en tiempo real para la generación de rutas óptimas.
5. Mejorar visualmente el diseño del mapa del programa.
6. Agregar capacidades de caché o memorización.

9. Referencias

[1] TomTom, “TomTom Traffic Index 2024: Average travel times in cities around the world,” TomTom International BV. [Online]. Available: https://www.tomtom.com/traffic-index/.

[2] P. E. Hart, N. J. Nilsson, and B. Raphael, “A formal basis for the heuristic determination of minimum cost paths,” IEEE Transactions on Systems Science and Cybernetics, vol. SSC-4, no. 2, pp. 100–107, 1968.

[3] A. V. Goldberg and C. Harrelson, “Computing the shortest path: A* meets graph theory,” in Proceedings of the Sixteenth Annual ACM–SIAM Symposium on Discrete Algorithms (SODA), Vancouver, Canada, 2005, pp. 156–165.

[4] R. Geisberger, P. Sanders, D. Schultes, and D. Delling, “Contraction hierarchies: Faster and simpler hierarchical routing in road networks,” in Proceedings of the 7th International Workshop on Experimental Algorithms (WEA), 2008, pp. 319–333.

[5] G. Gendreau, A. Hertz, and G. Laporte, “Metaheuristics for the traveling salesman problem,” Management Science, vol. 41, no. 8, pp. 1276–1290, 1995.

[6] F. Glover, “Tabu search—Part I,” ORSA Journal on Computing, vol. 1, no. 3, pp. 190–206, 1989.

[7] D. E. Goldberg, Genetic Algorithms in Search, Optimization, and Machine Learning. Reading, MA: Addison-Wesley, 1989.

[8] E. W. Dijkstra, “A note on two problems in connexion with graphs,” Numerische Mathematik, vol. 1, pp. 269–271, 1959.

A. Apéndice A: Código fuente relevante

Disponible en: https://github.com/rescobedoq/oep