Facultad de Ciencias Económicas y Estadística

Universidad Nacional de Rosario

Anteproyecto de Tesina

Licenciatura en Estadística

"Optimización de rutas en la ciudad de Rosario mediante la teoría de grafos y el algoritmo de Dijkstra"

Director: Iván Millanes
Codirector: Diego Marfetán Molina
Alumna: Valentina Salvarezza

Índice

1 Introducción

En la actualidad, cuando una persona necesita desplazarse de un punto a otro, es común recurrir a aplicaciones como Google Maps (Alphabet Inc., 2023). Estas herramientas han transformado la manera en que se planifican los desplazamientos, proporcionando información precisa sobre las rutas más convenientes, ya sea caminando, en automóvil, en bicicleta o utilizando otros medios de transporte. Además de indicar el recorrido a seguir, muchas de estas aplicaciones estiman el tiempo de llegada y consideran factores como el tráfico o las condiciones meteorológicas para ofrecer mejores alternativas.

Sin embargo, surgen algunas preguntas fundamentales: ¿Cómo funcionan realmente estas aplicaciones? ¿Cómo son capaces de calcular la ruta más corta o más rápida en función de las preferencias del usuario? Responder estas preguntas requiere apoyarse en la Teoría de Grafos, una rama de las matemáticas y la informática que permite modelar relaciones y estructuras en diversas áreas, como redes de comunicación, análisis de redes sociales y optimización de sistemas logísticos. En el contexto de la planificación de rutas, el problema se modela mediante un grafo, donde los nodos representan intersecciones y las aristas, los caminos que los conectan. A través de este modelo es posible calcular las rutas más eficientes, evaluando cada trayecto en función de costos asociados, como la distancia o el tiempo de viaje, e identificando la opción óptima que minimice dichos costos.

Para resolver este problema, se emplea el algoritmo de Dijkstra, desarrollado por Edsger W. Dijkstra en 1956 y publicado en 1959 (Dijkstra, 1959). Este algoritmo, ampliamente utilizado en teoría de grafos, permite encontrar el camino más corto entre dos nodos cuando los pesos de las aristas son no negativos. Su eficiencia y robustez lo convierten en una herramienta clave en diversas aplicaciones, como sistemas de navegación, redes informáticas y planificación logística. Uno de los servidores que implementa este algoritmo es OSRM (*Open Source Routing Machine*) (Luxen & Vetter, 2011), un enrutador de código abierto diseñado para procesar datos provenientes de OpenStreetMap (OSM) (OpenStreetMap, s.f.). OSRM, al integrarse con OSM, calcula rutas optimizadas en función de distintos criterios, como el medio de transporte o restricciones en el camino. Su capacidad para manejar grandes volúmenes de datos y ofrecer respuestas rápidas lo convierte en una solución ampliamente utilizada en aplicaciones de movilidad, logística y optimización de rutas.

En este trabajo, se desarrolla una aplicación web utilizando la librería Shiny (Chang, 2023)(**incluir cita a shiny usando la info que sale en citation("shiny")**) (**corregido**)

del lenguaje R. Shiny es una herramienta interactiva que permite crear aplicaciones web dinámicas y reactivas directamente desde RStudio. Esta aplicación está diseñada para calcular recorridos óptimos entre diferentes puntos de la ciudad de Rosario, permitiendo al usuario seleccionar intersecciones como puntos de origen y destino. A partir de esta selección, se genera el recorrido más corto o el más rápido, según la preferencia del usuario. Comprender cómo se estructuran y procesan estos datos no solo permite analizar en profundidad el funcionamiento de estas herramientas, sino también explorar aplicaciones en movilidad urbana, planificación del transporte público y optimización logística. Para establecer la conexión entre R y el motor de enrutamiento OSRM se utiliza el paquete osrm (Giraud, 2022), que permite integrar directamente los cálculos de rutas dentro de la aplicación Shiny.

2 Objetivos

2.1 Objetivo General

El objetivo principal de esta tesina es desarrollar una Shiny App en RStudio para la optimización de rutas en la ciudad de Rosario, aplicando el algoritmo Dijkstra y utilizando el motor de enrutamiento OSRM para el cálculo de recorridos. La aplicación permitirá considerar criterios de distancia y tiempo de viaje, y estará diseñada específicamente para vehículos automóviles.

2.2 Objetivos Específicos

- Analizar los datos cartográficos representados por nodos (intersecciones) y aristas (calles), que constituyen un tipo de dato no convencional, con el fin de evaluar su utilidad en el cálculo y optimización de rutas.
- Desarrollar e implementar el algoritmo de optimización Dijkstra y su extensión Multi-Level Dijkstra (MLD) para obtener soluciones eficientes de recorridos en grafos ponderados.
- Desplegar servidores locales de OSRM que permitan generar recorridos ponderados por distancia y tiempo de viaje mediante la implementación del algoritmo de Dijkstra, obteniendo de esta forma dos tipos de rutas: la más corta y la más rápida.

3 Metodología

3.1 Teoría de Grafos

La teoría de grafos es una rama central de las matemáticas y la informática que se dedica al estudio de las relaciones entre objetos representados por nodos o vértices, y las aristas, que son las conexiones entre ellos (Diestel, 2017; Ahuja, Magnanti y Orlin, 1993; Bast et al., 2016). Un grafo G se define formalmente como $G = (V, E)$, donde V es un conjunto no vacío de vértices y E es el conjunto de aristas que conectan pares de vértices. Estos modelos permiten representar intersecciones como nodos y calles como aristas, lo que facilita el análisis y la optimización de recorridos.

Algunos de los grafos que se utilizan para modelar redes viales son los siguientes:

- **Grafo simple:** Las aristas no tienen dirección, por lo que el recorrido puede hacerse en ambos sentidos. Este tipo de grafo se utiliza cuando se desea representar conexiones bidireccionales, como calles donde se puede circular en ambos sentidos. Un caso de aplicación podría darse en el desplazamiento a pie, ya que al caminar es posible utilizar ambos sentidos de la calle.
- **Grafo dirigido:** En este caso, las aristas tienen una orientación específica, representada como un par ordenado de vértices $(v, w) \in E, \forall\, v,\ w \in V$. Esto implica que la conexión desde v hacia w no necesariamente permite el recorrido inverso, lo que lo hace adecuado para modelar calles con sentido único.
- **Grafo ponderado:** Cada arista $e \in E$ tiene asociado un peso definido por una función $p : E \to \mathbb{R}^+$. El valor $p(e)$ representa el costo de la arista, que puede reflejar distancia, tiempo de viaje u otra magnitud. La longitud de un camino se calcula como la suma de los pesos de las aristas que lo integran, lo que permite construir recorridos según diferentes criterios.

En síntesis, la elección del tipo de grafo determina cómo se representan las restricciones y características de la red vial. En el presente trabajo se emplean principalmente grafos dirigidos y ponderados, donde la ponderación se realiza por distancia o por tiempo, según el criterio de optimización considerado.

3.2 Algoritmo de Dijkstra

El algoritmo de Dijkstra, propuesto por Edsger W. Dijkstra en 1956, es un método ampliamente utilizado para determinar el camino de costo mínimo entre un vértice de origen o y un vértice destino z dentro de un grafo ponderado. En estos grafos, cada arista tiene un peso que puede representar distancia, tiempo u otro tipo de costo, y se asume que estos pesos son no negativos. Este algoritmo resulta fundamental en sistemas de navegación y optimización de rutas, ya que permite calcular caminos mínimos de manera rápida y precisa.

La idea central del algoritmo consiste en avanzar desde un nodo de origen hacia los nodos adyacentes, seleccionando las rutas de menor costo en cada paso, hasta llegar al destino. Para ello, se define la función de distancias $D : V \to \mathbb{R}^+$, donde a cada nodo $v \in V$ se le asigna un valor $D(v)$, que representa el costo mínimo conocido desde el nodo de origen o hasta el nodo v. No se utiliza la notación $D(o, v)$ porque el vértice de origen o se mantiene fijo a lo largo de toda la ejecución del algoritmo, por lo que basta con indicar $D(v)$. Inicialmente, se establece que $D(o) = 0$, mientras que $\forall\ v \neq o$ se fija $D(v) = \infty$, lo que indica que aún no se ha encontrado un camino de costo mínimo desde el origen. Este procedimiento ha sido ampliamente documentado en la literatura clásica de algoritmos (Cormen et al., 2009) y continúa siendo base para desarrollos más avanzados en planificación de rutas y sistemas de transporte (Bast et al., 2016).

Se define el conjunto T como el conjunto de todos los nodos no visitados, es decir, aquellos nodos cuya distancia mínima desde el origen aún no ha sido confirmada. Formalmente, $T = V - \{o\}$, donde V es el conjunto de todos los nodos del grafo y o es el nodo de origen. Es importante notar que $t \in T$, donde t es cualquier nodo del conjunto T.

Estos valores se actualizan a medida que se exploran nuevas rutas y pueden clasificarse en dos tipos:

- **Etiquetas temporales:** Corresponden a valores de $D(v)$ que aún pueden modificarse si se encuentra un recorrido de menor costo hacia el vértice v.
- **Etiquetas permanentes:** Son aquellos valores de $D(v)$ que ya se consideran definitivos, representan el costo mínimo desde el nodo de origen o hasta el vértice v. Una vez que un valor se vuelve permanente, no se actualiza nuevamente.

El procedimiento del algoritmo se desarrolla de la siguiente manera:

1. **Inicialización:** Se establece el costo mínimo desde el nodo de origen hacia sí mismo $D(o) = 0$, y para todos los demás nodos $D(t) = \infty$, ya que no se conoce el costo hacia ellos. **Donde t pertenece al conjunto T ver si podemos poner esto en la oración anterior, quizas usando notacion matemática** (**incluido, ver si te gusta como quedo**), el cual se define como el conjunto de todos los nodos no visitados, es decir, aquellos nodos cuya distancia mínima desde el origen aún no ha sido confirmada.

2. **Selección del nodo con menor costo:** Se selecciona el nodo u del conjunto T con el valor más bajo de D, es decir, el nodo cuyo costo desde el origen es el mínimo entre los nodos no visitados.

3. **Cambiar nodo temporal a permanente:** El nodo u se elimina del conjunto T porque su costo ha sido confirmado y su etiqueta se convierte en permanente.

4. **Actualización de los costos:** Para cada nodo $t \in T$ vecino de u, si el costo de llegar a t desde el origen pasando por u es menor que el valor actual de $D(t)$, se actualiza su valor utilizando la fórmula: $D(t) = min\{D(t), D(u) + p(u,t)\}$, donde $p(u,t)$ representa el peso de la arista que conecta u con t.

5. **Repetir hasta encontrar el camino mínimo:** Este proceso se repite hasta que todos los nodos hayan sido visitados.

El algoritmo garantiza que, al asignar una etiqueta permanente a un nodo, se ha determinado el costo mínimo desde el origen hacia dicho nodo. Esto se debe a que, durante su ejecución, se evalúan todas las rutas posibles para llegar a cada nodo, asegurando la solución óptima. Sin embargo, este enfoque resulta computacionalmente costoso cuando se aplica al cálculo de recorridos en ciudades grandes, como Rosario. Por este motivo, en contextos de mayor escala, se utiliza una extensión del algoritmo llamada Multi-Level Dijkstra.

3.2.1 Multi-Level Dijkstra (MLD)

El Multi-Level Dijkstra (MLD) es una extensión del algoritmo de Dijkstra diseñada para optimizar la eficiencia en el cálculo de rutas en redes viales grandes, como las que se encuentran en ciudades extensas. Su objetivo principal es reducir la complejidad computacional y el tiempo de procesamiento al calcular rutas sobre grandes bases de datos geográficos (Goldberg & Silverstein, 1997; Delling et al., 2011; Bast et al., 2016).

El enfoque del MLD se basa en una estructura jerárquica de niveles:

- **Nivel 0:** el grafo original se divide en pequeñas celdas que representan barrios o vecindarios, incluyendo todas las calles e intersecciones internas.
- **Nivel 1:** varias celdas de nivel 0 se agrupan en unidades más grandes, como distritos, reduciendo la complejidad al analizar recorridos entre distintas zonas de la ciudad.
- **Nivel 2 y superiores:** los distritos se combinan en macrozonas que abarcan áreas extensas, por ejemplo, Norte, Sur, Este y Oeste.

Durante la fase de preprocesamiento, el algoritmo identifica los nodos frontera, definidos como aquellos vértices que poseen al menos una arista que conecta con un vértice ubicado en otra celda, y calcula previamente los caminos más cortos entre ellos. Con esta información se construye un grafo reducido (*overlay graph*) que funciona como un sistema de atajos entre celdas y permite que el cálculo de rutas se limite únicamente a las áreas relevantes.

El procedimiento del Multi-Level Dijkstra (MLD) funciona de la siguiente manera:

- Cuando el origen y destino se encuentran dentro de la misma celda de nivel 0, la búsqueda se realiza directamente sobre el subgrafo local mediante el algoritmo de Dijkstra, sin necesidad de utilizar MLD.
- Cuando el origen y destino están en celdas distintas, se utiliza un grafo reducido de niveles superiores para realizar un "salto" entre regiones. Luego, se desciende a niveles más detallados hasta llegar al nivel más bajo y reconstruir el trayecto completo, calle por calle.

Este enfoque permite mantener la precisión en la ruta óptima, mientras se reduce significativamente la cantidad de nodos a explorar, lo que resulta crucial para aplicaciones de movilidad urbana que manejan grandes volúmenes de datos geográficos, como los provenientes de OSM.

En resumen, MLD combina la exhaustividad del algoritmo de Dijkstra con una estructura jerárquica eficiente, optimizando tanto el tiempo de cálculo como la gestión de rutas en redes urbanas extensas.

3.3 Implementación y configuración de OSRM para la ciudad de Rosario

Open Source Routing Machine (OSRM) es una herramienta de enrutamiento de código abierto que utiliza datos geográficos de OpenStreetMap (OSM) (Luxen & Vetter, 2011; OpenStreetMap, s.f.) para calcular rutas óptimas en redes viales. Implementa el algoritmo de Dijkstra y su extensión optimizada Multi-Level Dijkstra, lo que le permite generar rutas rápidas y precisas incluso con grandes volúmenes de datos. OSRM se utiliza principalmente en aplicaciones de navegación y en sistemas donde se requiere cálculo eficiente de rutas.

3.3.1 Obtención de los datos

Los datos geográficos se obtienen de OpenStreetMap mediante la plataforma BBbike (BBBike, s.f.), que permite descargar áreas específicas en formato `.osm.pbf`. Este formato es requerido por OSRM ya que está comprimido y estructurado de manera eficiente para ser procesado rápidamente durante la fase de preparación de datos.

Se descargan los datos correspondientes a toda la ciudad de Rosario, asegurando que solo se incluyan los elementos (intersecciones y calles) relevantes para el cálculo de rutas. De esta forma, se evita el procesamiento de información irrelevante y se optimiza el rendimiento del sistema.

3.3.2 Cálculo de rutas

Aunque OSRM ofrece un servidor público para el cálculo de rutas, este servicio tiene limitaciones importantes. Si se superan ciertos volúmenes de solicitudes (*requests*), puede saturarse o incluso rechazar peticiones. Debido a esta restricción, y para garantizar un funcionamiento estable durante las pruebas y el desarrollo, se optó por implementar servidores de OSRM locales. Se configuró un servidor para calcular las rutas más cortas, priorizando la distancia, y otro servidor para calcular las rutas más rápidas, priorizando el tiempo de viaje.

Es importante destacar que se decidieron configurar dos servidores locales y no uno, ya que un único servidor no permite calcular rutas con criterios diferentes, ni de manera simultánea ni en distintas consultas. Esto hizo necesario implementar dos servidores para poder aplicar ambos criterios de forma independiente.

esta parte no la pondria en el anteproyecto, solo en la tesina modificada

3.3.3 Despliegue en la nube

si nombras shinyapps hay que definirlo, mejor decir directamente publicar en la web, o publicar online, algo asi modificado

Una vez construidos los servidores de OSRM locales, es necesario desplegarlos en la nube. Esto se debe a que, al publicar la aplicación en la web, los servidores deben estar accesibles de forma remota. Si permanecen en la red local, la aplicación no podrá comunicarse con ellos una vez desplegada.

Por este motivo, ambos servidores se despliegan en la nube utilizando los servicios de DonWeb, una plataforma de hosting que permite contratar servidores virtuales (VPS) con acceso remoto. Esto garantiza que los servicios de enrutamiento estén disponibles desde cualquier ubicación.

Este despliegue es clave para hacer posible la construcción de la aplicación. Al contar con los servidores OSRM en la nube, se vuelve posible conectarlos a la aplicación y habilitar una funcionalidad central: generar recorridos según el criterio seleccionado.

4 Cronograma de Trabajo

Tarea	**Período Estimado**
Revisión bibliográfica	Marzo - Abril 2025
Desarrollo del marco teórico sobre teoría de grafos	Abril - Mayo 2025
Análisis e implementación del algoritmo de Dijkstra y su extensión Multi-Level Dijkstra	Mayo - Junio 2025
Configuración e implementación del servidor OSRM para la ciudad de Rosario	Junio - Julio 2025
Desarrollo de la Shiny App con recorridos reales	Julio - Septiembre 2025
Redacción del informe final	Septiembre - Noviembre 2025
Revisión general del trabajo	Noviembre - Diciembre 2026
Fecha tentativa de presentación y defensa	Febrero 2026

5 Bibliografía

Ahuja, R. K., Magnanti, T. L., & Orlin, J. B. (1993). *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall.

Alphabet Inc. (2005). *Google Maps*. https://www.google.com/maps.

Bast, H., Delling, D., Goldberg, A., Müller-Hannemann, M., Pajor, T., Sanders, P., & Werneck, R. F. (2016). Route planning in transportation networks. En *Algorithm Engineering* (pp. 19-80). Springer. https://doi.org/10.1007/978-3-662-48433-3_3

BBBike. (n.d., n.d.). *BBBike extract service*. https://download.bbbike.org/osm/

Chang, W. (2023). *Shiny: Web Applications in R*. https://shiny.posit.co/

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (3rd ed.). MIT Press.

Delling, D., Sanders, P., Schultes, D., & Wagner, D. (2011). Customizable route planning. En P. Pardalos & S. Rebennack (Eds.), *Experimental Algorithms* (Vol. 6630, pp. 376-387). Springer. https://doi.org/10.1007/978-3-642-20662-7_32

Diestel, R. (2017). *Graph Theory* (5.ª ed.). Springer. https://doi.org/10.1007/978-3-662-53622-3

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, *1*(1), 269-271. https://doi.org/10.1007/BF01386390

Giraud, T. (2022). osrm: Interface Between R and the OpenStreetMap-Based Routing Service OSRM. *Journal of Open Source Software*, *7*(78), 4574. https://doi.org/10.21105/joss.04574

Goldberg, A. V., & Silverstein, C. (1997). Implementations of Dijkstra's algorithm based on multi-level buckets. En P. M. Pardalos, D. W. Hearn, & W. W. Hager (Eds.), *Network Optimization* (Vol. 450, pp. 292-327). Springer. https://doi.org/10.1007/978-3-642-59179-2_15

Luxen, D., & Vetter, C. (2011). Real-time routing with OpenStreetMap data. *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, 513-516. https://doi.org/10.1145/2093973.2094062

OpenStreetMap. (n.d., n.d.). *OpenStreetMap: datos cartográficos abiertos*. https://www.openstreetmap.org

Wickham, H., & Grolemund, G. (2017). *R for Data Science*. O'Reilly Media.

Zilske, M., Neumann, A., & Nagel, K. (2015). OpenStreetMap for traffic simulation. *Transportation Research Procedia*, *10*, 844-853. https://doi.org/10.1016/j.trpro.2015.09.037