UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

AYATORI: CREACIÓN DE MÓDULO BASE PARA PROGRAMAR ALGORITMOS DE PLANIFICACIÓN DE RUTAS EN PYTHON USANDO GTFS

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL EN COMPUTACIÓN

FELIPE IGNACIO LEAL CERRO

PROFESOR GUÍA:
EDUARDO GRAELLS GARRIDO

MIEMBROS DE LA COMISIÓN:
NELSON BALOIAN TATARYAN
HERNÁN SARMIENTO ALBORNOZ

SANTIAGO DE CHILE
2023

Resumen

El presente informe detalla la creación de un módulo en Python para programar algoritmos de planificación de rutas, utilizando la información del transporte público disponible en formato GTFS (General Transit Feed Specification), en el contexto del desarrollo de una Memoria para optar al título de Ingeniero Civil en Computación. La motivación principal es crear una herramienta base que permita desarrollar algoritmos que aporten en el estudio de planificación urbana y de transporte. Para evaluar la utilidad y correcta implementación de esta solución, se implementó una versión 'lite' de Connection Scan Algorithm, un algoritmo que utiliza la información en GTFS para calcular la mejor ruta para ir desde un punto A hasta un punto B. De esta implementación, se concluye que la herramienta creada funciona como se esperaba, cumpliendo exitosamente su objetivo.

Dijiste que tenías un sueño, y ahora... ¡se cumplirá! ¡Los sueños y los ideales tienen poder para cambiar el mundo!

-N.

Agradecimientos

A mi mamá, por enseñarme a estudiar, preocuparte por mi futuro, y darme una razón para salir adelante a pesar de todo. A mi papá, por todo tu amor, apoyo y respeto, por enseñarme a priorizar mi vida, por todos tus años de servicio como padre viudo, por nunca dejar de cuidarme, y ser mi ejemplo a seguir. Al amor de mi vida, por ser mi apoyo y compañía principal, por ayudarme a extender las fronteras de mis sueños y esperanzas, por dejarme estar en tu vida, darme alegría y luz en mis peores momentos, reirte de mis chistes, y todo tu amor incondicional.

Gracias Pauli, por amar a mi padre y darle la oportunidad de estar de nuevo felizmente casado, por tu amor y preocupación, y extender lo que entiendo por 'familia'. Gracias Ita, Renata y Felipe, por nuestra mutua adopción familiar y convertirse en mi abuela y hermanos. A mis tatas, Daniel y Pechita, por sentar las bases familiares que inspiraron mis valores y moral, por consentirme y preocuparse de mí aunque la distancia nos separe. A mi tía Helen, por su amor, apoyo, y preocupación constantes por mi bienestar. A mi tío Leo, por todas las risas y anécdotas que me ayudaron a apreciar la sobremesa en familia. A mis primos: Amalia, Cristobal, Benja, Naty y Bastian, por su amistad, amor, apoyo, y alegrías varias. Gracias especiales a Nico, mi hermano del alma. Gracias a todo el resto de mi familia extendida.

Gracias, tío Alvaro y tía Katy, por aceptarme como su yerno, por su preocupación y cariño, por darme una segunda familia, y por otorgarme la posibilidad de seguir trabajando en mi sueño cuando más lo necesité. Gracias, Florencia y Julieta, por enseñarme a ser un hermano mayor, todo su aceptación y amor. A toda la familia Luna, por aceptarme como uno más entre los suyos, y por todo el cariño, consejos, y buenas vibras.

Gracias a mi curso, 12°B, por acompañarme en la primera parte de mi vida y por los amigos que encontré entre sus filas. Gracias a Anime no Seishin Doukoukai, por darme la oportunidad de adquirir responsabilidades incluso con mis hobbies, y por todos los grandes amigos que me permitió hallar. Gracias a Ivancito, Kurisu, Gus y Chelo, por ser mi apoyo en los llantos y mi compañía en las celebraciones. Gracias Gabi, Julio, Gabo, Sofi, Naise, por su amistad. Gracias a Basti, Lucho y Seba, por ser mis primeros amigos en el mundo exterior y mantenerse a mi lado hasta el día de hoy. Gracias a todos aquellos compañeros de carrera con los que he podido compartir y colaborar al ir educándome, destacando especialmente a mi amigo Matías Vergara, y a todos los miembros de Team Michil.

Gracias a todas mis profesoras y profesores, por darme las herramientas para llegar a donde estoy hoy.

Tabla de Contenido

Índice de Ilustraciones

Estructura del Documento

Este informe presenta las distintas etapas del desarrollo de un módulo llamado 'ayatori', que contiene la base para programar algoritmos de planificación de rutas, correspondiendo a la Memoria para optar al título de Ingeniero Civil en Computación. El documento está dividido en 7 capítulos distintos, listados a continuación con su respectiva temática:

- **Capítulo 1: Introducción**. Entrega la base contextual y los objetivos del proyecto.
- **Capítulo 2: Estado del Arte**. Presenta los antecedentes del proyecto que componen su Marco Teórico, junto a los conceptos y herramientas a utilizar, para comprender la base teórica del mismo.
- **Capítulo 3: Diseño**. Explica el diseño de la solución propuesta, incluyendo el stack tecnológico a usar, el funcionamiento lógico de la solución, y el criterio de evaluación a considerar, explicitando el caso de estudio a realizarse.
- **Capítulo 4: Implementación**. Expone el desarrollo de las distintas fases de la implementación del módulo.
- **Capítulo 5: Resultados**. Muestra los resultados finales obtenidos al terminar la implementación, a través de ejemplos de uso del módulo, la ejecución del caso de prueba establecido, y la posterior evaluación.
- **Capítulo 6: Discusión**. Desarrolla las discusiones posteriores a la evaluación de resultados, considerando las implicancias y limitaciones de la solución, además de posibles líneas de trabajo futuro.
- **Capítulo 7: Conclusión**. Sintetiza el trabajo realizado y concluye el desarrollo del Trabajo de Título.

Posteriormente, se presenta la bibliografía utilizada y referenciada a lo largo del informe. Además, un anexo que muestra partes de implementaciones existentes del módulo.

Capítulo 1

Introducción

A día de hoy, es normal que las grandes ciudades experimenten cambios constantemente que las hagan crecer. Este fenómeno, común a nivel mundial, está presente también en Chile. Estudiando la situación local, existen múltiples causas asociadas, algunas de estas siendo más globales (como el cambio climático), y otras más específicas, como el importante aumento de la migración tanto interna como externa al país durante los últimos años, y la construcción de nueva infraestructura urbana. Si bien han existido ciertas condiciones que afecten negativamente el florecimiento de las ciudades, como la pandemia del COVID-19, la tendencia general de crecimiento se mantiene. Así es como, en un mundo donde las grandes urbes tienden a crecer exponencialmente, la planificación y buena gestión de las ciudades se ha visto afectada por el auge de estos fenómenos.

Es necesario, entonces, hallar maneras novedosas para comprender y caracterizar, correctamente, la vida de los habitantes de las grandes ciudades, tal como la capital de nuestro país, Santiago. En este mismo contexto, una arista muy importante a considerar es la movilidad vial, o el cómo las personas son capaces de movilizarse a través de las calles y avenidas de una ciudad, la cual es un factor determinante de la calidad de vida de sus habitantes. Las grandes ciudades suelen ser el hogar de una gran cantidad de personas, las cuales necesitan transportarse cada día para realizar sus jornadas de trabajo, de estudio, entre otras.

Existen múltiples registros de información que pueden ser utilizados para estudiar la movilidad urbana de ciudades como Santiago. Sin embargo, esto no implica que dicho estudio se pueda realizar sin inconvenientes notables. Por ejemplo, la *Encuesta Origen Destino* es una herramienta utilizada por los gobiernos para estudiar patrones de viajes de los habitantes de las ciudades, y el gobierno de Chile ha realizado esta encuesta en múltiples ciudades del país durante los últimos años. Esto, evidentemente, incluye también a Santiago, pero la última vez que se realizó fue en el año 2012, hace más de una década atrás [14]. Debido a esto, la información inferida gracias a la encuesta probablemente no represente, de forma correcta, la realidad actual del transporte en la capital, lo cual es una problemática común a esta clase de instrumentos de estudio. Se necesita, luego, una herramienta que permita hacer este trabajo más continuamente, y que represente al común de los habitantes de la ciudad.

Con respecto a los medios de movilización, la gente puede tener a su disposición múltiples tipos, tanto públicos como privados. Por ejemplo, se pueden mover a pie, en bicicleta, en auto, o utilizando el transporte público. Siguiendo la idea anterior, para poder caracterizar correctamente la movilidad urbana, sería útil verlo desde la perspectiva de un medio de transporte que esté disponible para toda la población, así que estudiar el uso del transporte público en Santiago resulta ser una buena opción para este fin. Dentro de la ciudad, esto incluye al Metro de Santiago y los buses de Red (antiguamente Transantiago), los cuales son usados por las personas en múltiples combinaciones, generando una cantidad enorme de rutas diferentes. Para almacenar y hacer pública la información del sistema de horarios de esta clase de medios de transporte, existe el formato GTFS (General Transit Feed Specification) [3] que utilizan las agencias de transporte en el mundo para estandarizar la información de, entre otras cosas, los diferentes servicios existentes, sus rutas y paradas respectivas.

Actualmente, existen múltiples algoritmos que se han diseñado con el fin de responder a consultas de movilidad. Por ejemplo, Connection Scan Algorithm [8] (CSA) es un algoritmo creado para responder de manera eficiente a consultas en los sistemas de información de horarios del transporte público, recibiendo como entrada una posición de origen y una posición de destino, y generando una secuencia de vehículos que el viajero debe tomar para recorrer una ruta entre ambos puntos. CSA, al igual que algoritmos que cumplen un objetivo similar, se alimentan de la información del transporte público disponible, enlazando esta información con los datos cartográficos de la ciudad a estudiar. Por este motivo, ya sea que se desee implementar un algoritmo existente o desarrollar uno nuevo, es crucial contar con una buena base de información y herramientas de programación que permitan la creación exitosa de nuevos mecanismos de estudio.

Este trabajo de título tiene por objetivo principal realizar un módulo en Python llamado Ayatori, que además de contar con la información del transporte en Santiago, contenga todas las definiciones y declaraciones necesarias para poder desarrollar algoritmos de generación de rutas. La idea es que el producto generado permita desarrollar estudios de movilidad vial de forma más actualizada y directa, a través de las herramientas que se puedan desarrollar usándolo como base. La visión a futuro es que puedan realizarse casos de estudio que visibilicen el impacto de la ampliación del transporte público disponible sobre los patrones de movilidad de las personas, y contribuir al desarrollo de nuevas tecnologías para programar soluciones de movilidad vial.

1.1. Objetivos

Objetivo General

El objetivo general de este trabajo de título es crear un módulo de trabajo en Python, con el fin de generar una base de programación para desarrollar algorimos de movilidad, focalizando su uso en Santiago de Chile. Para ello, se utilizarán los datos cartográficos de la ciudad provenientes de OpenStreetMap, un proyecto colaborativo de creación de mapas comunitarios [2], y la información del transporte público provista por la Red Metropolitana de Movilidad.

Objetivos Específicos

1. Obtener la información cartográfica de Santiago, además de la información del transporte público (en formato GTFS), y almacenarla en estructuras de datos pertinentes.
2. Enlazar la información de ambas fuentes de datos para ubicar las rutas de transporte en el mapa de Santiago.
3. Programar las definiciones para poder operar sobre estos datos, identificando lo necesario dentro de las estructuras de datos definidas y extrayendo la información que se necesite entregar al usuario.
4. Realizar un caso de prueba, utilizando el módulo para crear una implementación básica de un algoritmo de generación de rutas, y así ejemplificar la utilidad del trabajo realizado.

Capítulo 2

Estado del Arte

2.1. OpenStreetMap

OpenStreetMap (OSM) es un proyecto colaborativo cuyo propósito es crear mapas editables y de uso libre [10]. Los mapas, generados mediante la recopilación de información geográfica a través de dispositivos GPS móviles, incluyen detalles sobre las vías públicas (como pasajes, calles y carreteras), paradas de autobuses y diversos puntos de interés. Al ser un proyecto *Open-Source*, el desarrollo de los mapas locales es gestionado por organizaciones voluntarias de contribuyentes; en nuestro país, existe la Fundación OpenStreetMap Chile [2] cumpliendo ese papel. Se presenta la figura 2.1 a modo de ejemplo, donde se observa el mapa del Gran Santiago visualizado en su página web, en el que se pueden notar, entre otras vías, las carreteras más destacadas de la ciudad, como la Circunvalación Américo Vespucio y la Autopista Central.

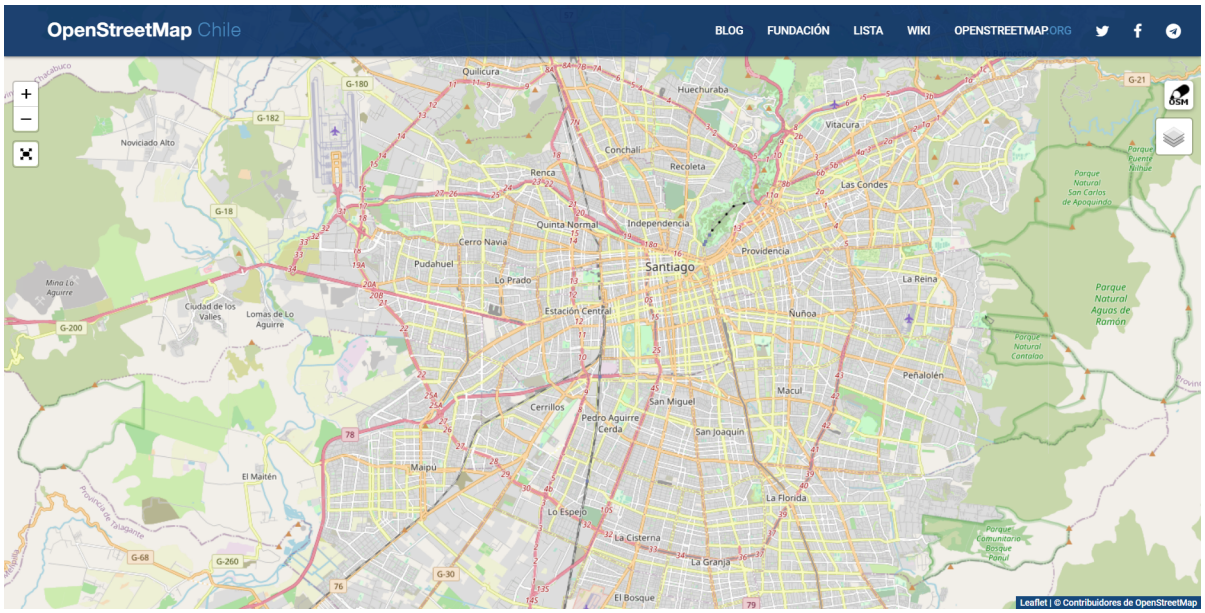

Figura 2.1: Mapa de Santiago en OpenStreetMap. Fuente: openstreetmap.cl .

2.1.1. OSM integrado en algoritmos

Para poder programar correctamente algoritmos de planificación de rutas, se requiere de la información cartográfica (o sea, mapas) de la ciudad en cuestión para poder ubicar los puntos que las rutas deben conectar. Para este fin, se pueden alimentar de los datos provenientes de OpenStreetMap (OSM), los cuales son utilizados para ubicar las coordenadas de los puntos de origen y destino en un mapa, y de esta forma obtener propiedades como la distancia entre los puntos, además de identificar detalles como las calles aledañas a las ubicaciones buscadas. Aquí también se obtienen las coordenadas de las paradas de los servicios de transporte público, tales como los paraderos de bus y las estaciones del Metro.

Anteriormente en la figura 2.1, se mostró una visualización de estos datos proveniente de la web de OpenStreetMap Chile. Además de analizar la información mediante esta página, el portal global de OpenStreetMap cuenta con un buscador que permite hallar direcciones de todo el mundo y visualizar un mapa de la zona [10]. Sin embargo, aparte de utilizar la información mediante visualizaciones web, los datos de OSM también se pueden descargar en distintos formatos para su uso. Esto permite una mayor versatilidad a la hora de crear herramientas que hagan uso de esta información, al poder obtener los datos en el formato más conveniente para trabajar con ellos.

Para integrar estos datos dentro de un módulo de Python, se puede importar alguna de las librerías existentes que permiten operar con estos datos. Por ejemplo, la librería **pyrosm** [22] funciona como un parser de la información de OSM en Python. **pyrosm** permite descargar la información más actualizada de la ciudad, almacenándola en un grafo dirigido donde cada nodo representan una intersección entre vías o algún lugar de interés, y las aristas entre los nodos representan las vías en sí; el que el grafo sea dirigido responde al sentido de las vías (es diferente una calle que es *doble vía* a una que va en un solo sentido). Trabajar con grafos permite otorgar propiedades a los componentes, como las coordenadas a los nodos (su latitud y longitud) o el largo a las aristas (representando la distancia entre ambos nodos que conecta). Además, de esta forma, la información de OSM se almacena en un formato conveniente para su fácil operación.

2.2. GTFS

Las Especificaciones Generales del Suministro de datos para el Transporte público, o en inglés, General Transit Feed Specification (GTFS), son un tipo de especificaciones ampliamente utilizado para definir y trabajar sobre datos de transporte público en las grandes ciudades. Este instrumento consiste en una serie de archivos de texto, recopilados en un archivo ZIP, de manera tal que cada archivo modela un aspecto específico de la información del transporte público, como paradas, rutas, viajes y horarios.

En la figura 2.2, se muestra el cómo se ven los archivos en GTFS, mostrando, como ejemplo, la información de los paraderos disponibles.

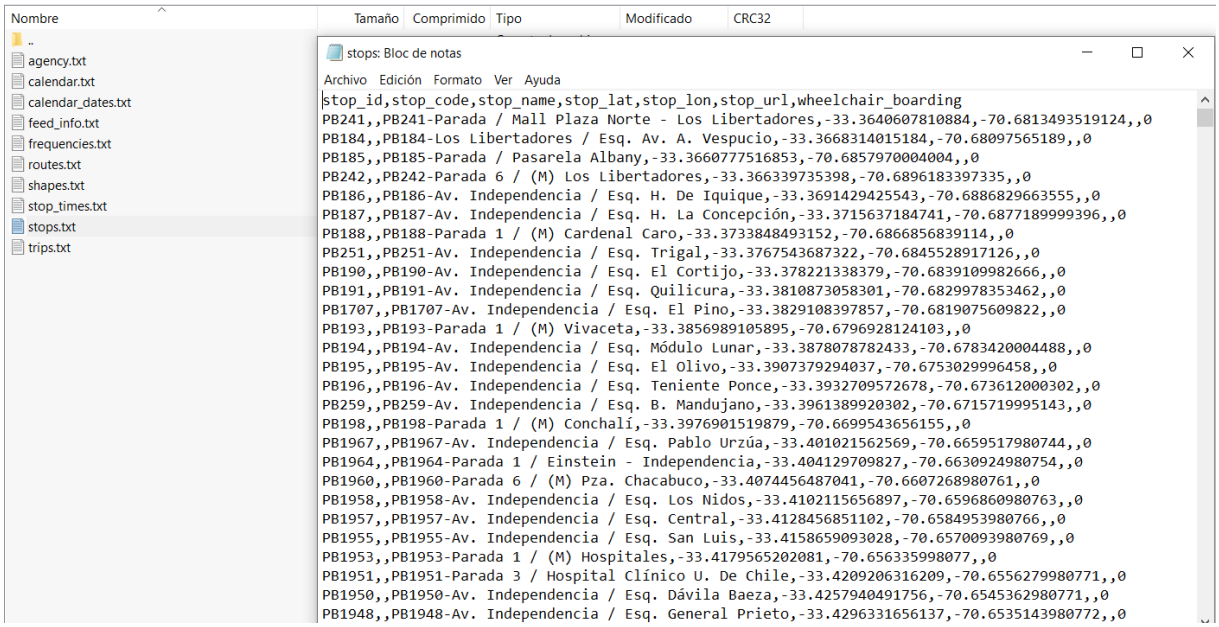

Figura 2.2: Visualización de archivos del feed GTFS para Santiago.

A nivel global, las organizaciones encargadas de la gestión administrativa del transporte público suelen utilizar este formato para compartir la información. En Santiago, el Directorio de Transporte Público Metropolitano (DTPM) es la entidad encargada de esta tarea. Este organismo, dependiente del Ministerio de Transportes y Telecomunicaciones, y cuya misión es mejorar la calidad del sistema de transporte público en la ciudad, tiene disponible públicamente esta información, y la actualiza periódicamente [6]. Al momento de la entrega de este informe, la última versión fue configurada para implementarse desde el 16 de septiembre de 2023.

La información en GTFS está contenida en diferentes archivos de texto, con sus valores separados por comas (similar a un CSV). Cada archivo concentra un área específica de los datos, las cuales se describen a continuación:

- **Agency**: entrega la información de las diferentes agencias de transporte que alimentan el GTFS. En este caso, se encuentra la Red Metropolitana de Movilidad (que engloba a todos los buses Red, antiguamente Transantiago), el Metro de Santiago, y EFE Trenes de Chile.
- **Calendar Dates**: especifica fechas especiales que alteran el funcionamiento habitual de los recorridos que varían por día. Para la última versión, este archivo contiene todas las fechas de feriados que caen entre lunes y sábado.
- **Calendar**: especifica los diferentes recorridos que varían por día, con su tiempo de validez. Acá se especifican los recorridos de Red para tres formatos diferentes: el cronograma para los días laborales (lunes a viernes), el cronograma para los sábados, y el cronograma para los domingos.
- **Feed Info**: información de la entidad que publica el GTFS.
- **Frequencies**: listado que, para todos los viajes de los recorridos disponibles, incluye

sus tiempos de inicio y de término, y el *headway* o tiempo de espera estimado entre vehículos.

- **Routes**: contiene el identificador de cada ruta existente, su agencia, ubicación de origen y destino.
- **Shapes**: lista las diferentes 'formas' de los viajes de cada recorrido. Esto incluye el identificador de cada viaje (el recorrido y si acaso es de ida o retorno), y las latitudes y longitudes para cada secuencia posible.
- **Stop Times**: incluye las horas estimadas de llegada para que cada recorrido incluido en el GTFS llegue a cada parada incluida en su trayecto.
- **Stops**: contiene los identificadores, nombres, latitud y longitud de cada parada de transporte.
- **Trips**: contiene todos los viajes diferentes que realiza cada recorrido, señalando el nombre del recorrido, sus días de funcionamiento, si es de ida o retorno, y su dirección de destino.

Siguiendo este formato, los operadores de transporte pueden almacenar y publicar la información pertinente a sus sistemas, para que esta sea utilizada por las personas o entidades que lo estimen conveniente. Por ejemplo, los desarrolladores de aplicaciones que permitan a sus usuarios revisar el estado actual de los servicios de transporte público, con el fin de planificar sus viajes. Un ejemplo de flujo de información en el que estos datos pueden ser utilizados se detalla en el diagrama mostrado en la figura 2.3.

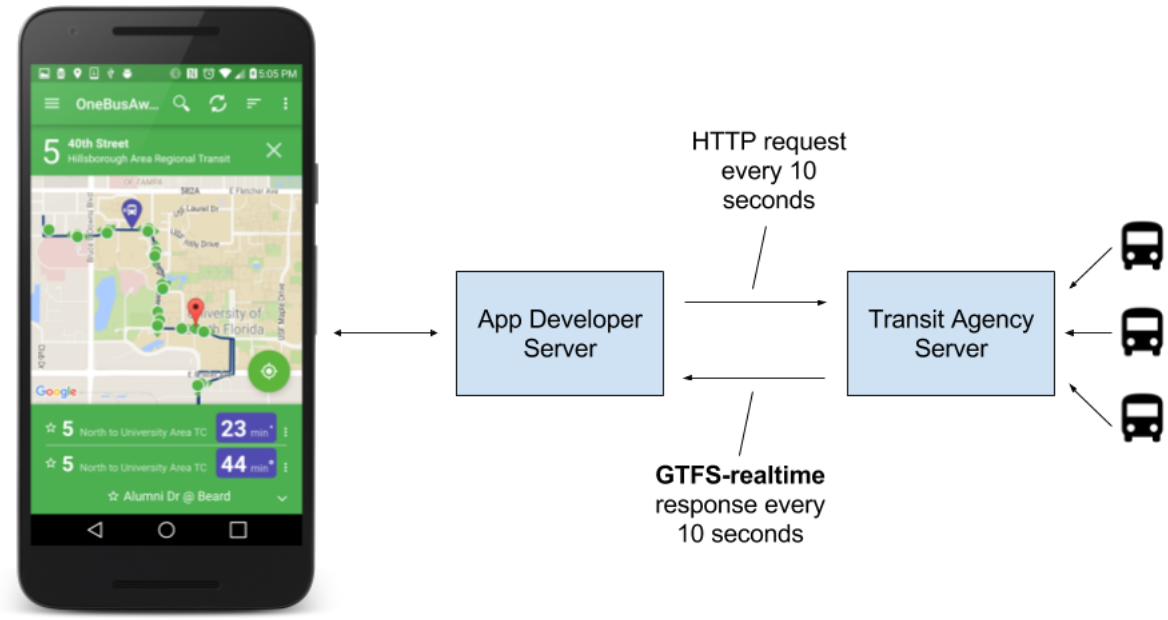

Figura 2.3: Diagrama de uso de datos en tiempo real en formato GTFS para una aplicación. Fuente: watrifeed.ml .

En este ejemplo, se muestra cómo una aplicación móvil se conecta al servidor que almacena sus datos, el cual hace consultas periódicas al servidor de la agencia de tránsito. Este, al contener la información de los recorridos (en este caso, de buses), responde con información en tiempo real en formato GTFS, que finalmente el servidor de la aplicación interpreta para mostrarle al usuario la ruta en un mapa. Si bien este ejemplo muestra una aplicación con información en vivo, también pueden realizarse aplicaciones con la información programada de los recorridos.

2.2.1. GTFS integrado en algoritmos

Los algoritmos de planificación de rutas necesitan tener a su disposición la información del transporte público, para ser capaces de calcular las rutas solicitadas. Esto implica que, para los distintos recorridos disponibles, se debe obtener los datos de sus rutas, paradas, horarios, y cualquier otra información que se estime necesaria para poder obtener la mejor ruta a seguir. Para este fin, es útil alimentar al algoritmo con la información del transporte público en formato GTFS, dado que así los datos están organizados de tal manera que son fácilmente accesibles, facilitando la programación y el cálculo de las rutas.

Similar al caso de OSM, se puede importar alguna librería existente que permita operar con los datos. Por ejemplo, la librería **pygtfs** [4] permite modelar archivos GTFS en Python. Esta librería almacena la información del transporte público en una base de datos relacional, tal que pueda ser usada en proyectos programados en este lenguaje de forma directa.

2.3. Datos: estructura y manejo de la información

Implementar algoritmos de planificación de rutas requiere trabajar con un gran volumen de datos. Sin ir más lejos, considerando el cómo se definen los nodos y aristas de OSM en **pyrosm** (como se mencionó en la sección 2.1.1), se deduce que, para una ciudad como Santiago, existe un volumen importante de información que debe almacenarse para poder operar con ella. Es por esta razón que es crucial saber elegir una buena herramienta para la creación de las estructuras de datos correspondientes. En esta misma línea, el tipo de estructura de datos a utilizar viene dado, precisamente, por la forma en la que se almacena la información de OSM: grafos. Dicho esto, y dado que existen múltiples librerías que manejan este tipo de estructura en Python, se debe elegir una que se adecúe mejor a las necesidades de este proyecto.

Una alternativa muy utilizada en conjunto a **pyrosm** es **networkx**, un paquete de Python para la creación, manipulación, y estudio de la estructura, dinámica, y funciones de redes complejas [7]. Esta librería está disponible para sistemas operativos Windows mediante **pip**, el sistema de gestión de paquetes de Python. La razón por la cual es ampliamente utilizada es por su simplicidad en el manejo y operación de la información. Sin embargo, su principal problema recae en su rendimiento, pues al estar programada completamente en Python, su desempeño es lento en comparación a otras opciones. Si, además, se toma en cuenta el gran volumen de datos que se requiere almacenar, se infiere que el uso de **networkx** terminará

generando un importante *bottleneck* o cuello de botella en el desempeño del algoritmo.

Por los motivos antes mencionados, se decide utilizar una librería diferente para este fin. La opción seleccionada es **graph-tool**, un módulo de Python creado para la manipulación y análisis de grafos [5]. A diferencia de otras herramientas, **graph-tool** posee la ventaja de tener una base algorítmica implementada en C++, un lenguaje de programación basado en compilación, por lo que su desempeño es mucho más eficiente. Esto permite trabajar con grandes volúmenes de información de mejor manera, por lo que demuestra ser una excelente librería para utilizar en este proyecto. Cabe destacar, eso sí, que **graph-tool** solo se encuentra disponible para sistemas operativos GNU/Linux y MacOS. Como consecuencia, la programación del algoritmo se realiza en Ubuntu, una distribución de GNU/Linux, mediante WSL2 (Windows Subsystem for Linux 2) [17].

2.4. Connection Scan Algorithm: un algoritmo de planificación de rutas

Dentro de los algoritmos diseñados para el fin de planificar rutas de transporte, se encuentra Connection Scan Algorithm (CSA), un algoritmo desarrollado para responder, de manera eficiente, consultas en sistemas de información de horarios [8]. Este algoritmo es capaz de optimizar los tiempos de viaje entre dos puntos determinados de origen y destino, siendo alimentado por distintas fuentes de información de transporte. Como salida, entrega una secuencia de vehículos (como trenes o buses) que un viajero debería tomar para llegar al destino desde el origen establecido. La base teórica tras el algoritmo hace que este analice las opciones disponibles y optimice el número de transbordos, tal que sea Pareto-eficiente, es decir, llegando al punto en el cual no es posible disminuir el tiempo de viaje en un medio de transporte sin tener que aumentar el de otro. En la figura 2.4, se grafica el funcionamiento antes descrito:

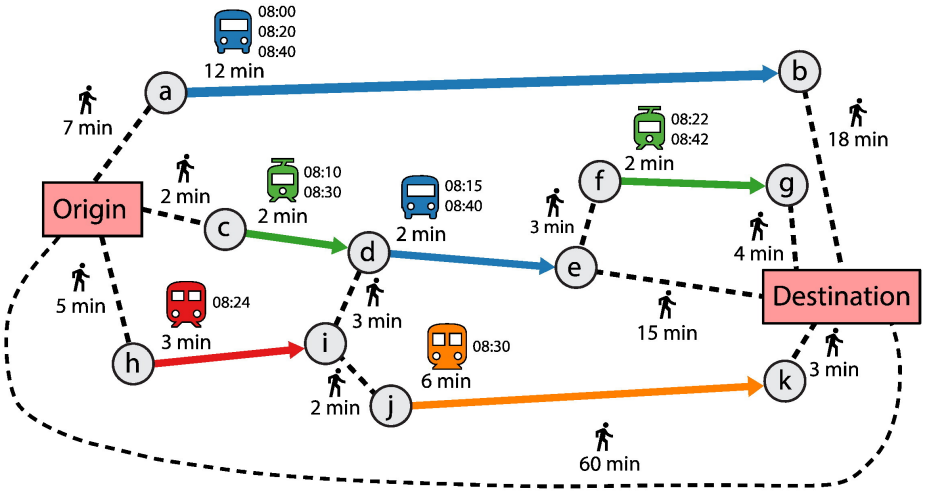

Figura 2.4: Diagrama explicativo del funcionamiento de Connection Scan Algorithm. Fuente: "Travel times and transfers in public transport: Comprehensive accessibility analysis based on Pareto-optimal journeys"(R. Kujala et al., 2017), vía sciencedirect.com .

Por ejemplo, si el punto de origen fuera la Facultad de Ciencias Físicas y Matemáticas de la Universidad de Chile (Beauchef 850, Santiago), y el destino fuera la Facultad de Derecho de la Universidad de Chile (Pio Nono 1, Providencia), se debieran evaluar los medios de transportes que pueden ser utilizados para ir desde las coordenadas del punto de origen hasta las del punto de destino, y los transbordos necesarios. Posibles rutas podrían abarcar:

1. Una ruta con uso exclusivo del Metro de Santiago (subiendo en estación Parque O'Higgins de Línea 2, combinando en Los Héroes a Línea 1 y bajando en Baquedano).
2. Una ruta con uso exclusivo de buses de Red (tomar el recorrido 121 y luego el recorrido 502).
3. Una ruta que realice transbordos entre ambos medios de transporte (subir al metro en estación Parque O'Higgins y bajar en Puente Cal y Canto, para luego tomar el recorrido 502).

Los recorridos del ejemplo se muestran en la figura 2.5, generada utilizando el portal de Google Maps, el servidor web de visualización de mapas de Google [11], por su simplicidad de uso. Las rutas aparecen enumeradas según la lista previa.

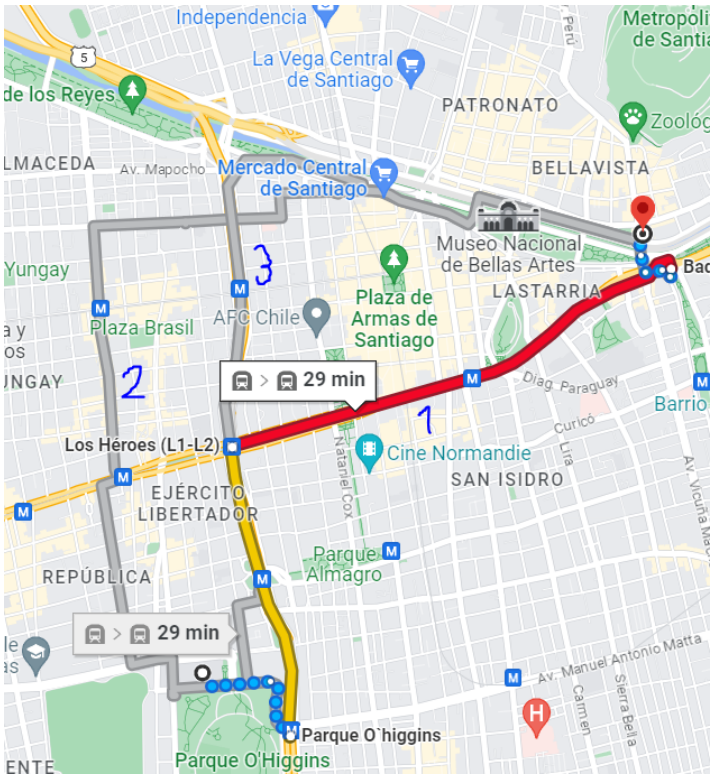

Figura 2.5: Posibles caminos para llegar desde FCFM hasta Derecho en transporte público. Fuente: Google Maps.

Ejemplificando el caso anterior para resolverlo mediante CSA, el algoritmo recibe como entrada las coordenadas del punto de origen y el punto de destino. Luego, revisando la información del transporte público, calcula las rutas posibles (como las descritas anteriormente). CSA entonces buscaría el punto óptimo de Pareto con respecto a los transbordos, y entregaría

la ruta recomendada para llegar al destino deseado. En este caso, al trabajar con información estática, se toman ciertos supuestos, como una velocidad de caminata fija entre transbordos y la continuidad operativa del servicio en todo momento.

Connection Scan Algorithm precisa, en primera instancia, ser capaz de obtener y almacenar la información del transporte público disponible, para así ser capaz de calcular la ruta óptima. Sin embargo, dado que el algoritmo trabaja con las coordenadas de los puntos de origen y destino, es bueno contar también con los datos cartográficos del sector o ciudad en cuestión donde se desea realizar el viaje, con el fin de obtener mejores visualizaciones de los resultados. Trabajando con ambos flujos de información, es posible crear una sólida implementación del algoritmo.

2.4.1. Utilidad como caso de prueba

Desde que Connection Scan Algorithm fue publicado, en marzo de 2017, ha sido implementado en varios formatos y lenguajes de programación. En el sitio web de Papers with Code, un portal que recopila códigos desarrollados sobre la idea central de diferentes papers, se muestran varias de estas implementaciones, enlazadas con su respectiva fuente de origen.

Destaca, entre estos, el repositorio de *ULTRA: UnLimited TRAnsfers for Multimodal Route Planning* [21], un framework desarrollado por el *Karlsruher Institut für Technologie* (KIT) en C++, para realizar planificaciones de viajes que incluyen diferentes medios de transporte. Este framework considera CSA, junto con otros algoritmos, para entregar posibles rutas entre dos puntos de una ciudad. Otra implementación disponible existe en el repositorio creado por Linus Norton, que creó una implementación del algoritmo en TypeScript [18].

Para demostrar la utilidad del módulo Ayatori para programar esta clase de algoritmos, se decide crear una implementación **básica** de CSA en Python como caso de prueba, estudiando rutas en Santiago. Además del hecho de que, por su arquitectura, el algoritmo requiera la información cartográfica de la ciudad y de su transporte público (ambas incluidas en Ayatori), la motivación principal para elegir este algoritmo es que, analizando el terreno actual, las implementaciones existentes están programadas en lenguajes diferentes a Python, por lo que existe una arista no explorada. La elección del lenguaje de programación motiva las decisiones posteriores de herramientas y librerías, que se mencionan en las secciones 2.1.1, 2.2.1, y 2.3, explicitadas y resumidas en la sección 3.1 del siguiente capítulo.

Capítulo 3

Diseño

3.1. Stack tecnológico

Basado en lo obtenido del capítulo anterior, se genera un formato para diseñar la solución propuesta. Para poder obtener toda la información necesaria para programar un algoritmo de planificación de rutas, se debe procesar correctamente tanto los datos cartográficos de Santiago, como la información del transporte público. Así, para desarrollar el proyecto, se define lo siguiente:

- El producto objetivo consiste en un módulo para el lenguaje de programación Python.
- La información cartográfica que se incluye en el módulo se obtiene desde OpenStreetMap, procesada mediante la librería **pyrosm** [22].
- La información del transporte público que se incluye en el modulo está almacenada en el formato GTFS, procesada mediante la librería **pygtfs** [4]. Para operar el módulo, los datos de transporte son obtenidos previamente, descargando la última versión desde el sitio web del Directorio de Transporte Público Metropolitano [3].
- Para almacenar y trabajar con la información obtenida, se utiliza la librería **graph-tool** [5] para trabajar con grafos.

Otras librerías que son utilizadas para cumplir de mejor forma los objetivos especificados en la sección 1.1 responden a la necesidad de procesar la información del módulo de forma tal que facilite su funcionamiento para el usuario final, a la hora de definir la entrada y la salida de los algoritmos de generación de rutas. En primer lugar, la librería **Nominatim** [13] permite que el usuario pueda ingresar como entrada una dirección en palabras, en vez de coordenadas numéricas, lo que facilita el uso de algoritmos y resta la necesidad de obtener las coordenadas de los puntos deseados por otro medio; **Nominatim** permite realizar la geocodificación de estas direcciones, buscando sus coordenadas en los datos de OpenStreetMap. En segundo lugar, la librería **folium** [9] permite visualizar datos cartográficos en un mapa de fácil uso. **folium** está basado en la librería **Leaflet.js** de JavaScript, por lo que aprovecha todas sus características para generar un mapa interactivo. Estas dos librerías son utilizadas en la implementación del caso de prueba para ejemplificar el tipo de uso del módulo Ayatori.

3.2. Funcionamiento lógico

Con el fin de facilitar el uso del módulo, las funcionalidades se almacenan en dos clases diferentes. La primera se encarga del almacenamiento y procesamiento de todos los datos provenientes de OpenStreetMap, mediante **pyrosm**. La segunda clase tiene por objetivo almacenar y procesar la información del transporte público, proveniente de **pygtfs**. De esta manera, ciudades como Santiago estarán representadas como una red de capas, donde una capa estará conformada por la información de la infraestructura urbana (calles, edificios, puntos de interés, etc.), y la otra estará conformada por la red de transporte público existente en ella (servicios, paradas, tiempos de espera, etc.)

3.2.1. OSMData: la clase de OSM

Al instanciar la clase **OSMData**, se descargan los datos más recientes de OpenStreetMap para Santiago, y se almacenan en un grafo de **graph-tool**. En este grafo, los nodos representan puntos de interés de la ciudad (pudiendo ser edificios o intersecciones), y las aristas representan a las vías, ya sean calles, pasajes o carreteras. Las aristas del grafo son dirigidas, cuya dirección representa el sentido de la vía (diferenciando las vías de un solo sentido de las llamadas *doble vía*).

Cada elemento del grafo generado posee propiedades para almacenar información relevante. En el caso de los nodos, sus propiedades dentro del grafo son:

- **Node ID**: el identificador del nodo dentro de los datos de OpenStreetMap.
- **Graph ID**: el identificador interno del nodo dentro del mismo grafo.
- **Lon**: la longitud de la ubicación asociada al nodo.
- **Lat**: la latitud de la ubicación asociada al nodo.

Por otro lado, las propiedades que poseen las aristas del grafo son:

- **u**: corresponde al vértice desde donde inicia la arista.
- **v**: corresponde al vértice hacia donde se dirige la arista.
- **Length**: corresponde al *tramo* cubierto por la arista, el cual debe ser mayor o igual a 2 para considerarse válida.
- **Weight**: el peso de la arista, que representa el *metraje* cubierto por la misma, es decir, la distancia física entre sus vértices.

La clase **OSMData** posee funcionalidades para visualizar los elementos del grafo de OSM, así como también funciones para operar con estos.

3.2.2. GTFSData: la clase de GTFS

Instanciando la clase **GTFSData**, se procesan los datos previamente descargados del transporte público (ubicados en un archivo llamado *gtfs.zip* en el mismo directorio, a menos que se indique lo contrario). La información de cada tabla es leída y procesada para cada servicio de transporte disponible, para así, posteriormente, crear un grafo de **graph-tool** independiente para cada servicio y almacenar sus datos. En este grafo, los nodos representan las paradas del servicio, y las aristas enlazan cada parada con la siguiente del recorrido que siga en la misma orientación.

Para cada grafo, sus elementos poseen propiedades para almacenar información, al igual que en el caso de OSM mencionado en la sección 3.2.1. En el caso de los nodos, se tiene:

- **Node ID**: el identificador de la parada.

Por otro lado, las aristas poseen las siguientes propiedades:

- **u**: corresponde al vértice desde donde inicia la arista. En este caso, el identificador de la parada de origen.
- **v**: corresponde al vértice hacia donde se dirige la arista. En este caso, el identificador de la parada de destino.
- **Weight**: el peso de la arista. Inicialmente, acá le damos peso 1 a todas las aristas.

Además de esto, se hace necesario crear un diccionario que almacene todos los datos que enlazan a una parada con una ruta en cuestión. Esto es debido a que existe información importante que cobra sentido únicamente al solapar los datos de una parada con los de una ruta. En específico, estos son:

- Orientación: refiere al sentido de la ruta cuando se detiene en una parada en específico. Cada ruta tiene un recorrido de ida y uno de vuelta, y por lo general, solo se detiene en un determinado paradero en uno de los sentidos.
- Número de Secuencia: al realizar una ruta en una orientación dada, el número de secuencia es el valor ordinal de una parada para esa ruta. En palabras simples, representa el orden en el que la ruta pasa por las paradas (la primera parada, la segunda, la tercera, etc.)
- Tiempos de llegada: representa la hora aproximada en la que una ruta llega a una parada.

La orientación y el número de secuencia deben utilizarse para filtrar las rutas que sirven para viajar entre dos puntos del mapa, mientras que los tiempos de llegada son cruciales para elegir la mejor ruta y entregar el resultado. Sin embargo, ninguno de estos datos son inherentes a una parada o a una ruta, pues, por ejemplo, no se puede decir que una ruta *posee* una orientación, sino que pasa por una parada al ir en cierta orientación. Por estos motivos, se opta por usar un diccionario anidado, aparte de los grafos por ruta, para almacenar esta clase de información. Este diccionario se denomina **route_stops**.

Al igual que para el caso anterior, la clase **GTFSData** posee funcionalidades para visualizar los elementos del grafo de GTFS, así como también funciones para operar con estos.

3.2.3. Funcionalidades entre clases

Además de las funcionalidades creadas como métodos dentro de las dos clases previamente mencionadas para operar con la información, es necesario crear funciones adicionales que crucen los datos provistos por OSM y los que están en formato GTFS. Esto permite obtener información útil para generar rutas de transporte, tal como los nodos del mapa de OSM a los que corresponden las paradas de una ruta en específico del transporte público, o hallar la lista de paradas que se encuentran cerca de un punto específico del mapa. El generar estas funcionalidades fuera de las clases provistas permite no caer en malas prácticas de diseño como tener que instanciar una clase dentro de otra. La especificación de estas funcionalidades, además de los métodos de cada clase, se ahondan con mayor profundidad en el capítulo 4 del informe (Implementación).

3.3. Criterio de Evaluación

Tal como fue discutido anteriormente, la motivación principal al desarrollar el módulo Ayatori es crear una base de programación para desarrollar algoritmos de generación de rutas, específicamente enfocadas en el uso del transporte público de la ciudad. Para efectos de este Trabajo de Título, se usa a Santiago como ejemplo para mostrar las capacidades del módulo, pero dada la naturaleza de los datos utilizados, si se quisiera estudiar la movilidad de otra ciudad, basta con modificar la procedencia de los datos (específicamente el lugar buscado en OSM y el archivo del transporte público en formato GTFS).

En cualquier caso, considerando que el usuario final del proyecto es cualquier programador que desee desarrollar algoritmos de generación de rutas para estudiar la movilidad vial, se debe definir un criterio de evaluación acorde para valorar la utilidad de la solución creada. En este caso, el criterio es:

- El usuario final deberá ser capaz de programar un algoritmo de generación de rutas de transporte público, utilizando únicamente la información provista por el módulo Ayatori, y obtener resultados útiles para realizar un estudio de movilidad.

Posterior a la implementación, se realiza un caso de prueba para analizar la utilidad de Ayatori. La finalidad es probar la efectividad de la solución desarrollada, ejemplificando la utilidad del módulo y evaluando el cumplimiento del criterio definido anteriormente. El caso de prueba definido consiste en programar una versión *lite* de Connection Scan Algorithm [8], que cuente con una visualización gráfica que mapee una ruta en Santiago de Chile, para ir desde un punto a otro de la ciudad utilizando el transporte público disponible (Metro de Santiago o buses Red). Además, la implementación debe hacer uso de la información provista por el módulo para entregarle información adicional al usuario, tal como los tiempos de espera estimados para los siguientes recorridos de la ruta buscada.

Cabe destacar que la definición de CSA considera transbordos entre distintos recorridos del transporte público. Esta funcionalidad no está implementada en este caso de prueba, por escapar del objetivo general del proyecto (definido en la sección 1.1. Por este motivo, se habla de una versión *lite* de CSA, que cumple con calcular la ruta más conveniente considerando distancias y tiempos de espera estimados, siendo suficiente para demostrar la utilidad del módulo. El desarrollo de este Caso de Prueba está documentado en la sección **??** del informe.

Capítulo 4

Implementación

En el presente capítulo, se detalla la implementación realizada del módulo Ayatori y todo el trabajo que corresponde a su desarrollo. El código fuente de la implementación ha sido almacenado en un repositorio de GitHub creado para este fin [16].

4.1. Obtención de datos

4.1.1. Procesamiento de OpenStreetMap

La información almacenada en OpenStreetMap puede ser descargada en formato PBF (Protocolbuffer Binary Format), para luego ser filtrada y procesada según lo necesitado. Geofabrik, un portal comunitario para proyectos relacionados con OpenStreetMap [23], tiene disponible para descarga la información de los distintos países del mundo, incluído Chile [24]. Con esto, es posible obtener la información geoespacial de Santiago y trabajar con ella, para lo cual es necesario procesarla correctamente. En un principio, se pretendía realizar este proceso manualmente, pero se descubrió una mejor alternativa, que permite automatizarlo.

pyrosm [22], la librería utilizada para procesar la información, permite leer datos de OpenStreetMap en formato PBF e interpretarla en estructuras de GeoPandas [15], librería de Python de código abierto para trabajar con datos geoespaciales. Además de esto, **pyrosm** también permite directamente descargar la información de una ciudad y actualizarla en caso de existir una versión anterior en el directorio, permitiendo automatizar este proceso. De esta forma, una vez descargada la información de Santiago, se pueden crear gráficos según se necesite para su representación.

Para realizar este procedimiento, dentro de la clase **OSMData** se programa el método **download_osm_file**, que usando el método **get_data** de **pyrosm**, descarga la información de la ciudad especificada. Como salida, entrega el puntero al archivo que contiene los datos cartográficos de dicho lugar. La definición de este método se muestra en el código 4.1:

```
def download_osm_file(self, OSM_PATH):
    fp = pyrosm.get_data(
        "Santiago", # Nombre de la ciudad
        update=True,
        directory=OSM_PATH)
    return fp
```

Código 4.1: Definición del método **get_osm_data()**.

Por otro lado, se define el método **create_osm_graph**, que utilizando el método anterior, crea un grafo con la información obtenida. Aquí se definen y evalúan las propiedades para cada elemento del grafo, tal y como fue mencionado en la sección 3.2.1. Finalmente, se retorna el grafo creado. De esta manera, la clase **OSMData** llama a este método para instancear el grafo como definición interna de la clase. Un fragmento de este método se aprecia en el código 4.2:

```
def create_osm_graph(self, OSM_PATH):
    fp = self.download_osm_file(OSM_PATH)
    osm = pyrosm.OSM(fp)
    nodes, edges = osm.get_network(nodes=True)
    graph = Graph()

    # Propiedades
    lon_prop = graph.new_vertex_property("float")
    lat_prop = graph.new_vertex_property("float")
    node_id_prop = graph.new_vertex_property("long")
    graph_id_prop = graph.new_vertex_property("long")
    u_prop = graph.new_edge_property("long")
    v_prop = graph.new_edge_property("long")
    length_prop = graph.new_edge_property("double")
    weight_prop = graph.new_edge_property("double")
    (...)

    return graph
```

Código 4.2: Fragmento del método **create_osm_graph()** que crea el grafo y las propiedades de sus elementos.

Capítulo 5

Resultados

Capítulo 6

Discusión

6.1. Implicancias

6.2. Limitaciones

6.3. Trabajo Futuro

Capítulo 7

Conclusión

Bibliografía

[1] Bicineta Chile. Mapa de Ciclovías de la Región Metropolitana. Disponible en `https://www.bicineta.cl/ciclovias`. Revisado el 2023/03/07.

[2] Fundación OpenStreetMap Chile. Mapa de OpenStreetMap Chile. Información disponible en `https://www.openstreetmap.cl`. Revisado el 2023/03/07.

[3] GTFS Community. General Transit Feed Specification. Disponible en `https://gtfs.org`. Revisado el 2023/06/28.

[4] Yaron de Leeuw. pygtfs. Repositorio disponible en `https://github.com/jarondl/pygtfs`. Revisado el 2023/06/29.

[5] Tiago de Paula Peixoto. graph-tool: Efficient network analysis with python. Documentación disponible en `https://graph-tool.skewed.de`. Revisado el 2023/07/17.

[6] Directorio de Transporte Público Metropolitano. GTFS Vigente. Disponible en `https://www.dtpm.cl/index.php/gtfs-vigente`. Revisado el 2023/07/22. Última versión: 2023/07/08.

[7] NetworkX developers. Networkx - network analysis in python. Documentación disponible en `https://networkx.org`. Revisado el 2023/07/22. Última versión: 2023/04/04.

[8] Julian Dibbelt, Thomas Pajor, Ben Strasser, and Dorothea Wagner. Connection Scan Algorithm. *ACM Journal of Experimental Algorithmics*, 23(1.7):1–56, 2018.

[9] Filipe Fernandes. Folium. Repositorio disponible en `https://github.com/python-visualization/folium`. Revisado el 2023/07/17.

[10] OpenStreetMap (Global). Mapa de OpenStreetMap. Información disponible en `https://www.openstreetmap.org`. Revisado el 2023/03/07.

[11] Google. Google Maps. Disponible en `https://www.google.com/maps/`.

[12] Eduardo Graells-Garrido. Aves: Análisis y Visualización, Educación y Soporte. Repositorio disponible en `https://github.com/zorzalerrante/aves`. Revisado el 2023/03/07.

[13] Sarah Hoffmann. Nominatim. Disponible en `https://nominatim.org`. Revisado el 2023/07/17.

[14] Universidad Alberto Hurtado. Actualización y recolección de información del sistema de transporte urbano, IX Etapa: Encuesta Origen Destino Santiago 2012. Encuesta origen destino de viajes 2012. Disponible en `http://www.sectra.gob.cl/biblioteca/detalle1.asp?mfn=3253` (2012). Revisado el 2023/03/07. Última versión lanzada el 2014.

[15] Kelsey Jordahl. Geopandas. Documentación disponible en https://geopandas.org. Revisado el 2023/03/07. Última versión: 2022/12/10.

[16] Felipe Leal. CC6909-Ayatori (Repositorio del Trabajo de Título). Repositorio disponible en https://github.com/Lysorek/CC6909-Ayatori. Revisado el 2023/03/07.

[17] Microsoft. Windows subsystem for linux. Documentación disponible en https://learn.microsoft.com/en-us/windows/wsl/. Revisado el 2023/07/17.

[18] Linus Norton. Connection Scan Algorithm (implementación en TypeScript). Repositorio disponible en https://github.com/planarnetwork/connection-scan-algorithm. Revisado el 2023/06/29.

[19] Data Reportal. Digital 2021 Report for Chile. Disponible en https://datareportal.com/reports/digital-2021-chile (2021/02/11). Revisado el 2023/03/07.

[20] Audrey Roy and Cookiecutter community. Cookiecutter: Better project templates. Documentación disponible en https://cookiecutter.readthedocs.io/en/stable/. Revisado el 2023/03/07.

[21] Jonas Sauer. ULTRA: UnLimited TRAnsfers for Multimodal Route Planning. Repositorio disponible en https://github.com/kit-algo/ULTRA. Revisado el 2023/03/07.

[22] Henrikki Tenkanen. Pyrosm: Read OpenStreetMap data from Protobuf files into GeoDataFrame with Python, faster. Repositorio disponible en https://github.com/HTenkanen/pyrosm. Revisado el 2023/03/07.

[23] Jochen Topf and Frederik Ramm. Geofabrik. Disponible en https://www.geofabrik.de. Revisado el 2023/03/07.

[24] Jochen Topf and Frederik Ramm. Geofabrik download server - chile. Disponible en https://download.geofabrik.de/south-america/chile.html. Revisado el 2023/03/07. Última versión: 2023/03/06.

[25] Papers with Code. Connection Scan Algorithm implementations. Disponible en https://cs.paperswithcode.com/paper/connection-scan-algorithm. Revisado el 2023/03/07.