



Análisis GTFS Vancouver

Trabajo Final

73.82 - Bases de Datos Espaciales y de Movilidad

Autores:

Tomas Camilo Gay Bare¹

Manuel E. Dithurbide²

Diciembre 2025

¹tgaybare@itba.edu.ar

²mdithurbide@itba.edu.ar

Índice

1. Introducción	3
2. Metodología de replicación	3
2.1. Instalación de dependencias	3
2.2. Requisitos de la base de datos	3
2.3. Flujo GTFS estático	4
2.3.1. Descarga y preprocesamiento	4
2.3.2. Importación de tablas GTFS	4
2.3.3. Creación de estructuras MobilityDB	4
2.3.4. Carga de densidad poblacional (opcional)	4
2.3.5. Consultas y visualizaciones estáticas	5
2.4. Flujo GTFS-Realtime	5
2.4.1. Configuración y tablas realtime	5
2.4.2. Ingesta de feeds GTFS-Realtime	5
2.4.3. Construcción de trayectorias reales	6
2.4.4. Análisis puntual de un recorrido	6
2.4.5. Consultas y análisis agregados con GTFS-Realtime	6
3. Análisis de GTFS estático	7
3.1. Composición y alcance de la red	7
3.2. Densidad de rutas por segmento	9
3.3. Velocidades programadas por segmento	10
3.4. Densidad poblacional y cobertura de transporte	13
3.5. Accesibilidad local a estadios	16
3.6. Conectividad entre estadios y áreas de alta densidad	18
4. Análisis con GTFS-Realtime	20
4.1. Velocidad real vs. velocidad programada	20
4.2. Desempeño de horarios (schedule times)	23
4.3. Segmentos de demora y patrones de congestión	25

4.4. Regularidad de headways y <i>bus bunching</i>	27
5. Discusión y trabajo futuro	29

1. Introducción

Este proyecto analiza el sistema de transporte de Vancouver utilizando datos GTFS estáticos y GTFS-Realtime de TransLink. Se implementa un pipeline reproducible sobre PostgreSQL con extensiones PostGIS y MobilityDB, utilizando Python para análisis y visualización. El pipeline permite caracterizar la red estática, comparar planificación vs. operación y generar visualizaciones para identificar problemas de servicio.

2. Metodología de replicación

En esta sección se documenta paso a paso el pipeline completo, desde la instalación hasta la generación de los resultados presentados en las secciones siguientes.

2.1. Instalación de dependencias

Situarse en la raíz del repositorio y ejecutar:

```
pip install -r requirements.txt
pip install -r static_analysis/requirements.txt
pip install -r realtime_analysis/requirements.txt
npm install -g gtfs-via-postgres
```

Configurar la conexión a la base de datos exportando las variables de entorno:

```
export PGHOST=localhost
export PGPORT=5432
export PGUSER=postgres
export PGPASSWORD=postgres
export PGDATABASE=gtfs
```

Para hacer estas variables persistentes, agregarlas al perfil del shell (e.g., `~/.bashrc`, `~/.zshrc`).

2.2. Requisitos de la base de datos

Antes de ejecutar cualquier comando, asegurarse de que:

- La base de datos `gtfs` existe y está corriendo.
- Las extensiones PostGIS y MobilityDB están instaladas y habilitadas.
- Las variables de entorno de conexión están exportadas (ver sección anterior).

Se puede verificar la conexión y las extensiones con:

```
psql -h $PGHOST -p $PGPORT -U $PGUSER -d $PGDATABASE \
-c "SELECT_PostGIS_version(), MobilityDB_version();"
```

2.3. Flujo GTFS estático

2.3.1. Descarga y preprocesamiento

Ejecutar el script de descarga y limpieza:

```
bash static_analysis/data/download_data.sh
```

Este paso genera los archivos prunados en static_analysis/data/gtfs_pruned/.

2.3.2. Importación de tablas GTFS

Con la base de datos en marcha, importar las tablas:

```
cd static_analysis/data/gtfs_pruned
gtfs-to-sql --require-dependencies -- *.txt \
| psql -h $PGHOST -p $PGPORT -U $PGUSER -d $PGDATABASE
```

2.3.3. Creación de estructuras MobilityDB

Desde la raíz del repositorio:

```
cat static_analysis/data_loading/mobilitydb_import.sql \
| psql -h $PGHOST -p $PGPORT -U $PGUSER -d $PGDATABASE
```

Este script genera tablas como scheduled_trips_mdb, route_segments, vistas agregadas y columnas geométricas.

2.3.4. Carga de densidad poblacional (opcional)

Para los análisis que combinan densidad poblacional y cobertura de transporte, se utiliza un GeoJSON de áreas censales de Vancouver. El siguiente script calcula la densidad (pop/a) e importa la capa population_areas en la base de datos:

```
python static_analysis/data/download_population_data.py \
--geo static_analysis/data/population/vancouver_geo.geojson
```

2.3.5. Consultas y visualizaciones estáticas

Las vistas materializadas y las gráficas de análisis se generan con un único comando que orquesta las consultas SQL y los scripts de visualización:

```
cd static_analysis/queries  
python run_all_analyses.py
```

Este comando:

- Ejecuta los archivos SQL en `static_analysis/queries/sql/`, creando vistas materializadas `qgis_*` para visualización en QGIS (densidad de rutas, segmentos de velocidad, población vs. transporte, etc.).
- Ejecuta los scripts de `static_analysis/queries/visualizations/`, generando gráficos en formato PNG (histogramas, distribuciones por ruta, comparaciones de velocidad, métricas de proximidad a estadios, etc.).

Los resultados se guardan en `static_analysis/queries/results/`, organizados por tipo de análisis. Las vistas `qgis_*` pueden cargarse directamente en QGIS para producir mapas más detallados.

2.4. Flujo GTFS-Realtime

2.4.1. Configuración y tablas realtime

Instalar dependencias adicionales y configurar la API de TransLink:

```
pip install -r realtime_analysis/requirements.txt  
export TRANSLINK_GTFSR_API_KEY="your-translink-api-key"
```

Crear las tablas necesarias para datos realtime:

```
cat realtime_analysis/realtime_schema.sql \  
| psql -h $PGHOST -p $PGPORT -U $PGUSER -d $PGDATABASE
```

2.4.2. Ingesta de feeds GTFS-Realtime

Ejecutar el proceso de ingestá durante un intervalo de tiempo:

```
python -m realtime_analysis.ingest_realtime \  
--duration-minutes 20 \  
--poll-interval 30
```

Este comando consulta periódicamente los endpoints de posiciones y actualizaciones de viaje, almacenando los mensajes en tablas `realtime_*`.

2.4.3. Construcción de trayectorias reales

A partir de los puntos GPS se construyen trayectorias *map-matched* sobre los recorridos estáticos:

```
python -m realtime_analysis.build_realtime_trajectories --hours  
2
```

Las trayectorias resultantes se almacenan en `realtime_trips_mdb`. Los puntos GPS crudos se deduplican y se ajustan (*snap*) a las formas programadas antes de insertarse en la tabla.

2.4.4. Análisis puntual de un recorrido

Para comparar detalladamente un viaje programado vs. su ejecución real:

```
python -m realtime_analysis.analyze_realtime \  
--route-short-name 99
```

Los resultados se escriben en `realtime_analysis/output/` como mapas HTML, gráficos de tiempos y archivos CSV con métricas por segmento.

2.4.5. Consultas y análisis agregados con GTFS-Realtime

Las vistas materializadas y los análisis agregados sobre datos realtime se generan de forma análoga al flujo estático, a través de un único script orquestador:

```
cd realtime_analysis/queries  
python run_all_analyses.py
```

Este comando ejecuta los archivos SQL de `realtime_analysis/queries/sql/`, creando vistas `realtime_*` y `qgis_realtime_*`, y luego corre los scripts de visualización para producir gráficos en PNG y resúmenes en CSV.

También es posible ejecutar cada análisis de forma individual:

```
cd realtime_analysis/queries/visualizations  
python speed_vs_schedule_analysis.py  
python schedule_times_analysis.py  
python delay_segments_analysis.py  
python headway_analysis.py
```

Los resultados se almacenan en `realtime_analysis/queries/results/`, organizados por tipo de análisis.

3. Análisis de GTFS estático

3.1. Composición y alcance de la red

La primera consulta caracteriza la composición de la red en términos de tipo de servicio (bus, subway, rail, ferry, etc.), utilizando las tablas GTFS routes y un CTE de mapeo de códigos:

```
WITH route_types(route_type, name) AS (
    SELECT '0', 'streetcar' UNION
    SELECT '1', 'subway' UNION
    SELECT '2', 'rail' UNION
    SELECT '3', 'bus' UNION
    SELECT '4', 'ferry' UNION
    SELECT '11', 'trolley'
),
route_groups AS (
    SELECT
        route_type,
        COUNT(*) AS qty,
        ROUND(COUNT(*) * 100.0 / SUM(COUNT(*)) OVER (), 2) AS perc
    FROM routes
    GROUP BY route_type
)
SELECT name, qty, perc
FROM route_groups g JOIN route_types t ON g.route_type = t.
    route_type
ORDER BY perc DESC;
```

Esta consulta devuelve, para cada modo, la cantidad de rutas y su porcentaje sobre el total. A partir de la vista materializada qgis_route_visualization, QGIS genera un mapa de la red completa (Figura 1), mientras que el script route_visualization.py construye gráficos de oferta por ruta que complementan la lectura visual de la red.

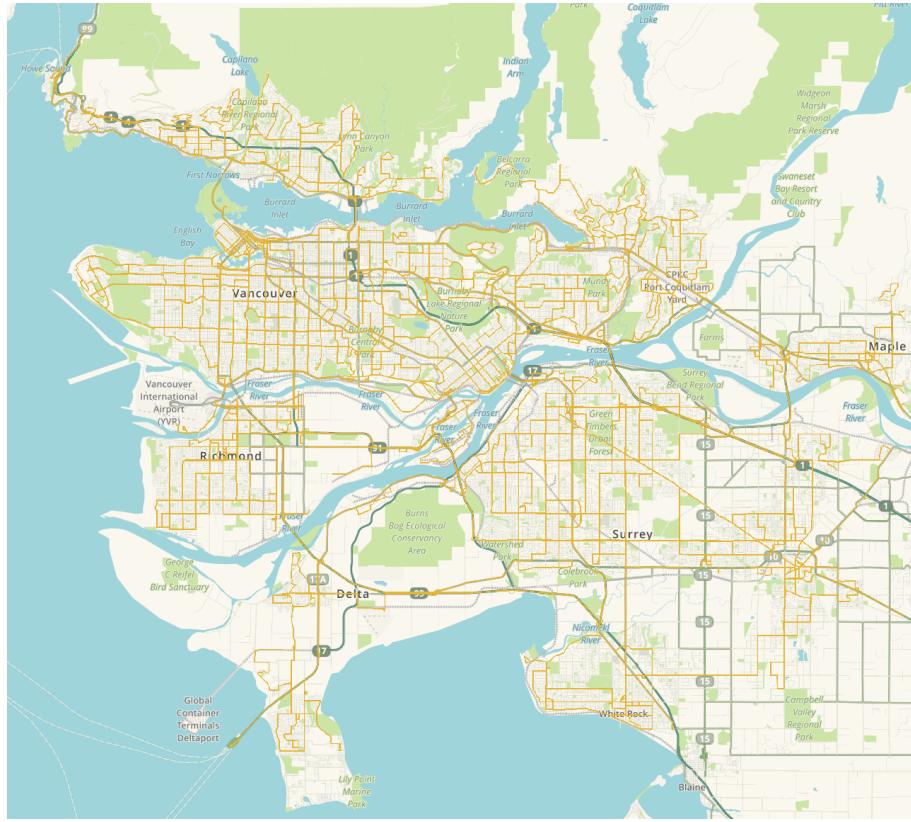


Figura 1: Mapa de la red de rutas de Vancouver a partir de GTFS estático.

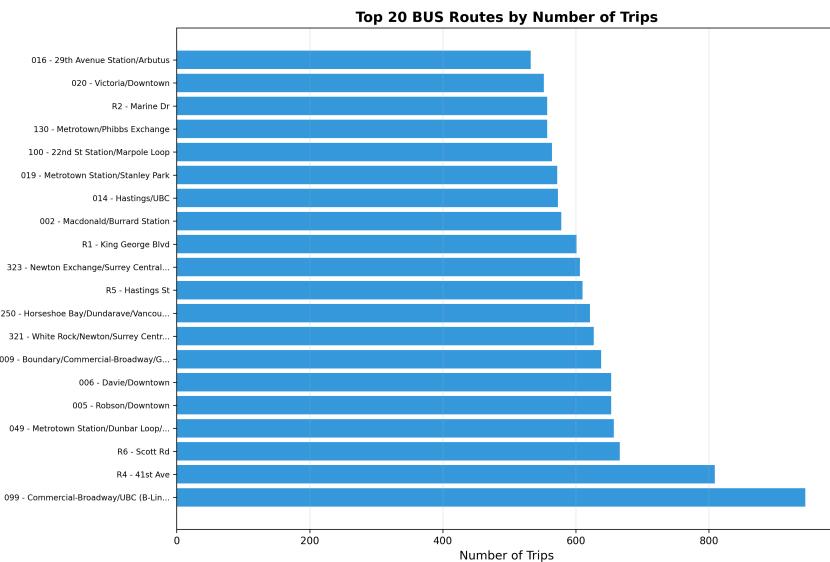


Figura 2: Rutas de colectivo ordenadas por cantidad de viajes diarios.

El mapa evidencia una red especialmente densa en el centro de Vancouver y en los corredores troncales hacia el este y el sur. El gráfico de barras muestra que un subconjunto reducido de rutas concentra gran parte de los viajes diarios, mientras que la mayoría opera

con menor frecuencia, típicamente en zonas periféricas.

3.2. Densidad de rutas por segmento

Para estudiar la superposición de recorridos sobre el espacio urbano se construye una vista materializada de densidad de rutas por segmento (`segment_route_density`) basada en `route_segments`. Un fragmento simplificado de la definición es:

```
DROP MATERIALIZED VIEW IF EXISTS segment_route_density;
CREATE MATERIALIZED VIEW segment_route_density AS
SELECT
    stop1_id || stop2_id AS segment_id,
    seg_geom,
    COUNT(DISTINCT route_id) AS num_routes
FROM route_segments
WHERE seg_geom IS NOT NULL
GROUP BY stop1_id, stop2_id, seg_geom;
```

En la implementación final esta lógica se encapsula en las consultas de `static_analysis/queries/sc` y se materializa junto con el resto de vistas al ejecutar `static_analysis/queries/run_all_analysis`. El script `route_density_analysis.py` resume esta información y genera un histograma de densidad (Figura 3), mientras que la vista se utiliza como capa en QGIS para construir el mapa de la Figura 4.

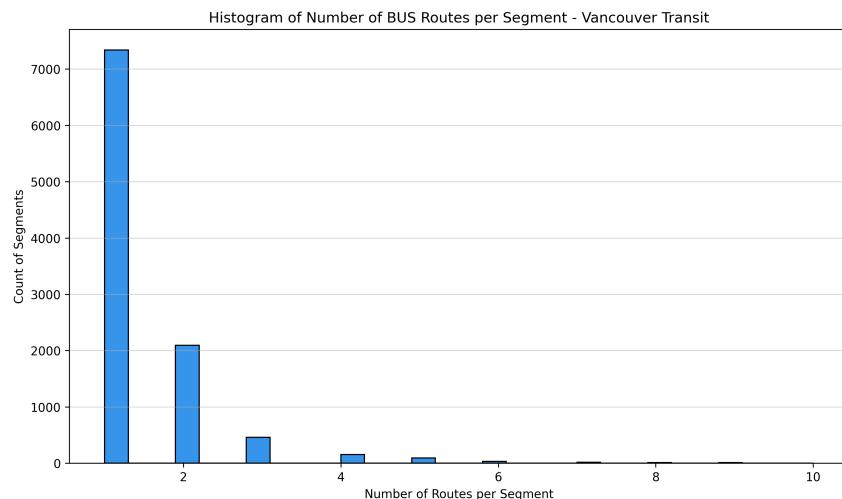


Figura 3: Histograma de cantidad de rutas por segmento.

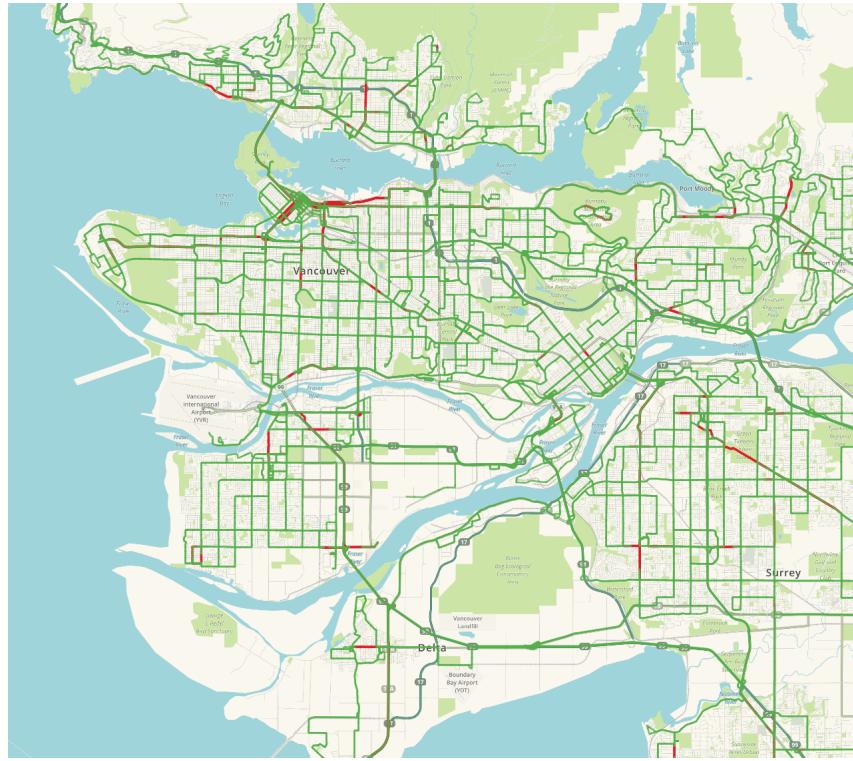


Figura 4: Mapa de densidad de rutas sobre la red de segmentos (verde = 1–2 rutas; rojo = 3 o más rutas por segmento).

El histograma evidencia que la mayoría de segmentos tiene pocas rutas superpuestas, mientras que un número reducido de corredores concentra un alto nivel de servicio. En el mapa estos corredores de alta densidad aparecen en rojo alrededor del centro de la ciudad y en ejes troncales, mientras que las zonas periféricas se observan principalmente en verde, con menor superposición de recorridos.

3.3. Velocidades programadas por segmento

La velocidad promedio por segmento y por ruta se obtiene combinando `route_segments` con los horarios de llegada previstos, y se materializa en una vista de segmentos con velocidad programada. Un esquema básico de cálculo de velocidades a partir de los horarios es:

```

DROP MATERIALIZED VIEW IF EXISTS schedule_speeds;
CREATE MATERIALIZED VIEW schedule_speeds AS
SELECT
    s.route_id || stop1_sequence || stop2_sequence AS id,
    AVG(seg_length / EXTRACT(EPOCH FROM
        (stop2_arrival_time - stop1_arrival_time)) * 3.6) AS
        speed_kmh,
    seg_geom
FROM route_segments s
    
```

```

WHERE stop2_arrival_time <> stop1_arrival_time
AND seg_length > 0
GROUP BY s.route_id, stop1_sequence, stop2_sequence, seg_geom;

```

Sobre esta base se construye la vista `qgis_speed_segments`, que agrega estadísticas por ruta y filtra los segmentos inválidos. A partir de esta vista:

- QGIS genera mapas de velocidad promedio sobre la red (Figuras 5 y 6),
- el script `speed_analysis.py` construye histogramas y gráficos agregados por ruta y modo (Figura 7).

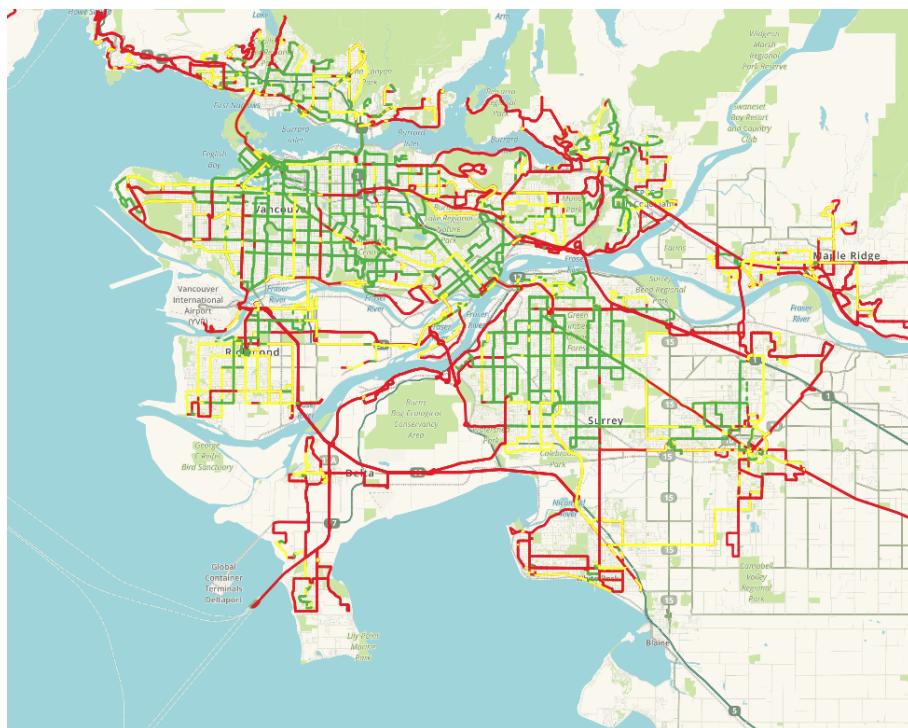


Figura 5: Mapa de velocidades programadas sobre la red de segmentos (3.5–19 km/h = verde punteado; 19–23 km/h = verde; 23–30 km/h = amarillo; >30 km/h = rojo).

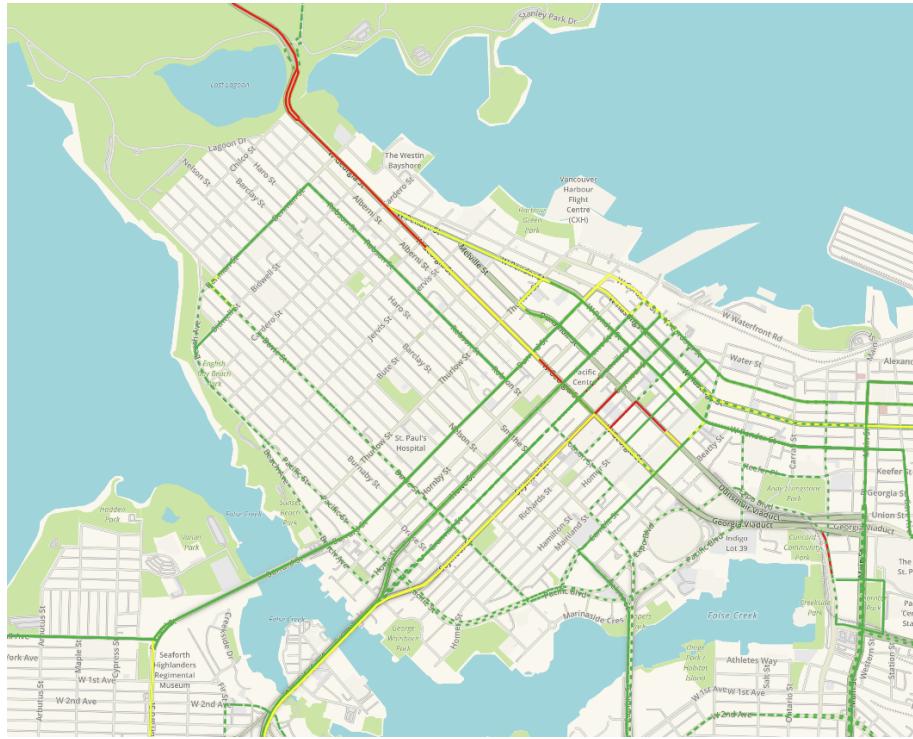


Figura 6: Detalle de zonas particularmente lentas en la red de segmentos.

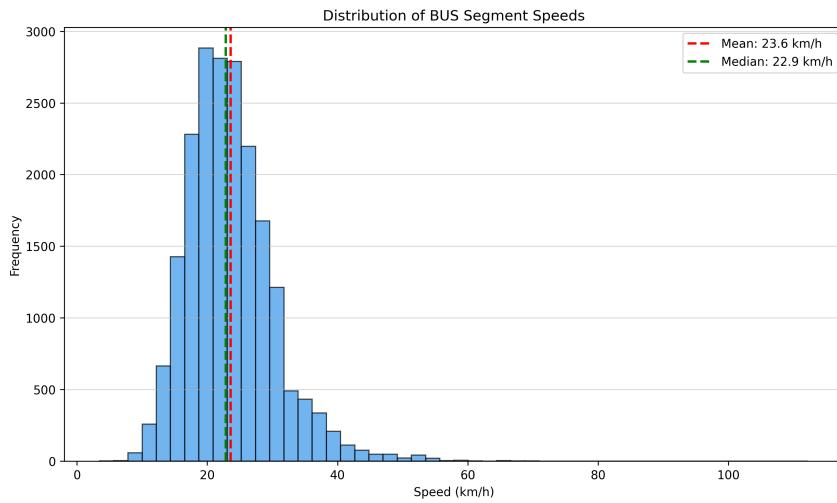


Figura 7: Histograma de velocidades programadas por segmento.

Los mapas permiten localizar rápidamente los corredores con mayores velocidades (en rojo y amarillo) frente a zonas más lentas (verde punteado), asociadas generalmente a áreas céntricas o con alta fricción. El histograma muestra una distribución concentrada en torno a velocidades medias, con colas hacia valores muy bajos que corresponden a tramos fuertemente congestionados.

3.4. Densidad poblacional y cobertura de transporte

Para relacionar la oferta de transporte con la demanda potencial se incorporan áreas censales de Vancouver con población y superficie. A partir de este insumo se construye una capa de densidad poblacional y una superposición con la red de transporte utilizando las vistas population_density y qgis_population_transit_overlay.

Un esquema simplificado de la superposición entre densidad y recorridos es:

```
CREATE MATERIALIZED VIEW qgis_population_transit_overlay AS
WITH population_areas AS (
    SELECT
        id,
        geom,
        population_density,
        CAST(pop AS double precision) AS population,
        CAST(a AS double precision) AS area_km2
    FROM population_density
    WHERE geom IS NOT NULL
        AND population_density IS NOT NULL
),
bus_route_segments AS (
    SELECT DISTINCT
        rs.stop1_id || rs.stop2_id AS segment_id,
        rs.seg_geom
    FROM route_segments rs
    JOIN routes r ON rs.route_id = r.route_id
    WHERE r.route_type = '3'
        AND rs.seg_geom IS NOT NULL
)
SELECT
    pa.id,
    pa.population_density,
    COUNT(DISTINCT brs.segment_id) AS num_segments,
    COALESCE(SUM(ST_Length(
        ST_Intersection(pa.geom, brs.seg_geom)::geography)) / 1000,
        0) AS route_length_km,
    pa.area_km2,
    route_length_km / NULLIF(pa.area_km2, 0) AS
        route_density_km_per_km2,
    pa.geom
FROM population_areas pa
LEFT JOIN bus_route_segments brs
    ON ST_Intersects(pa.geom, brs.seg_geom)
GROUP BY pa.id, pa.population_density, pa.geom, pa.area_km2;
```

Estas consultas se ejecutan automáticamente al correr static_analysis/queries/run_all_analysis que primero crea las vistas materializadas y luego genera los gráficos.

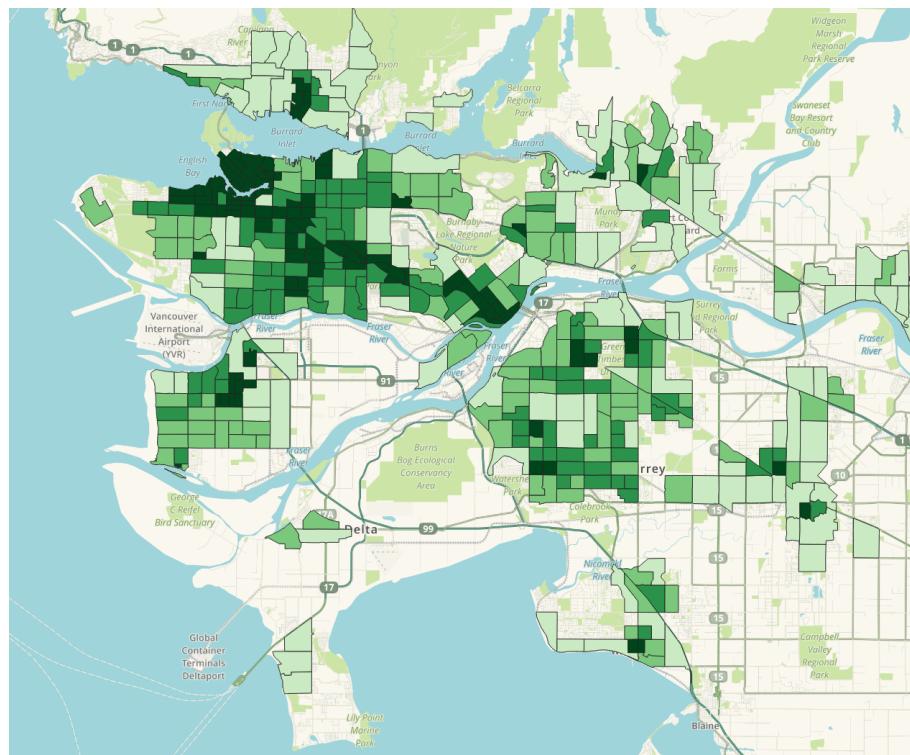


Figura 8: Mapa de densidad de población por área censal.

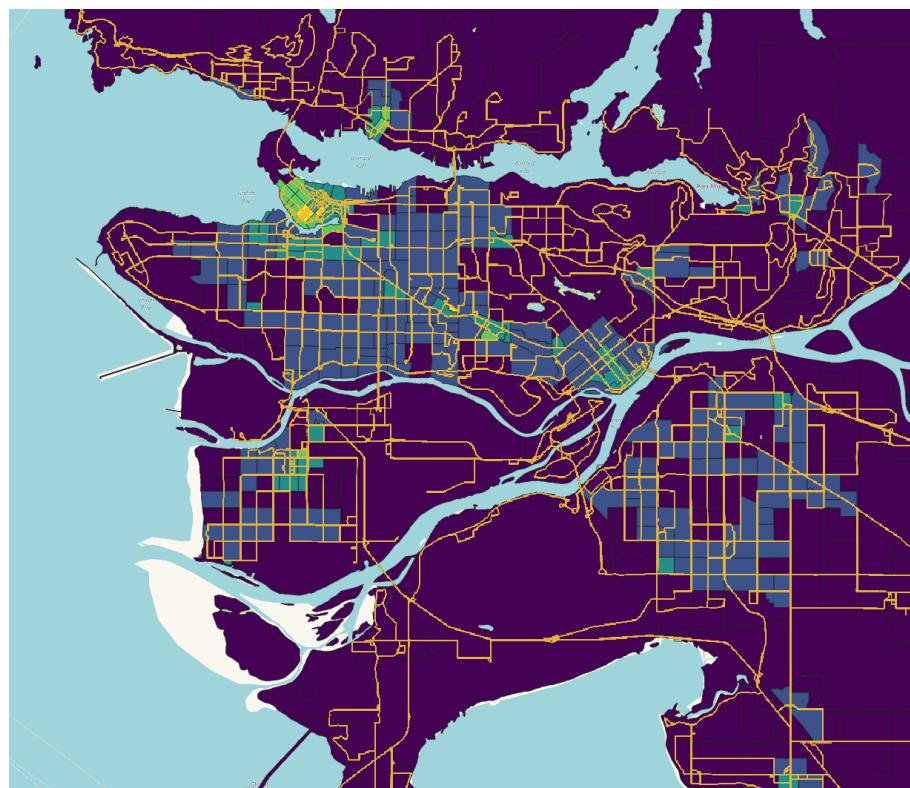


Figura 9: Superposición de densidad de población y red de rutas de colectivo.

El primer mapa destaca las zonas de mayor concentración poblacional, mientras que la superposición con las rutas permite identificar corredores bien servidos frente a “manchas” densas con menor cobertura.

El script `population_density_analysis.py` genera gráficos que resumen esta relación:

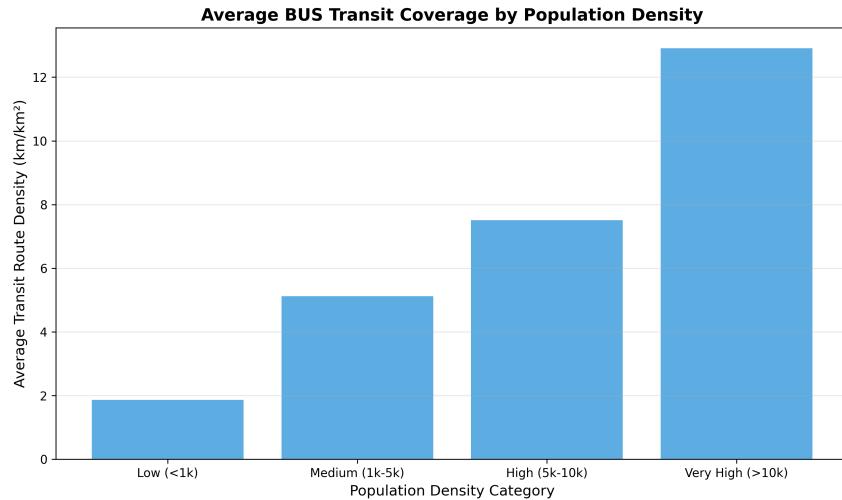


Figura 10: Cobertura de transporte por categoría de densidad poblacional.

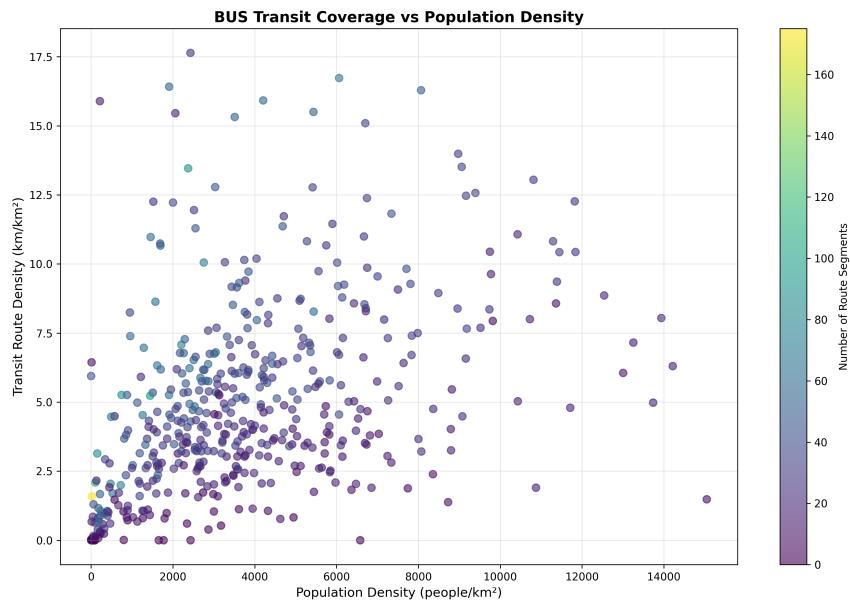


Figura 11: Relación entre densidad poblacional y longitud de rutas cercanas.

En conjunto, los mapas y gráficos muestran que las zonas céntricas concentran tanto mayor densidad poblacional como mejor cobertura de transporte, mientras que algunas áreas densas en la periferia aparecen con niveles de servicio más limitados, sugiriendo posibles oportunidades de mejora.

3.5. Accesibilidad local a estadios

Los estadios y arenas deportivas se modelan como puntos de interés dentro del área de estudio. La tabla `football_stadiums` se puebla con coordenadas de BC Place, Rogers Arena y Pacific Coliseum, y se analizan las paradas y rutas de colectivo dentro de un radio de 600 m (consistente en todo el trabajo). Las vistas `qgis_stadiums` y `qgis_stadium_proximity` alimentan los mapas y gráficos de esta sección.

La tabla de estadios y una vista básica de proximidad pueden definirse como:

```
CREATE TABLE IF NOT EXISTS football_stadiums (
    id serial PRIMARY KEY,
    name text NOT NULL,
    team text,
    latitude float,
    longitude float,
    geom geometry(Point, 4326)
);

CREATE MATERIALIZED VIEW qgis_stadium_proximity AS
SELECT
    s.name AS stadium_name,
    s.team,
    st.stop_id,
    st.stop_name,
    ST_DistanceSphere(s.geom, st.stop_loc::geometry) AS distance_m,
    st.stop_loc::geometry AS geom
FROM football_stadiums s
JOIN stops st
ON ST_DistanceSphere(s.geom, st.stop_loc::geometry) <= 600;
```

En el proyecto, `static_analysis/queries/sql/08_stadium_proximity.sql` contiene una versión extendida de esta consulta, y el script `stadium_proximity_analysis.py` utiliza la vista materializada correspondiente para generar los gráficos.

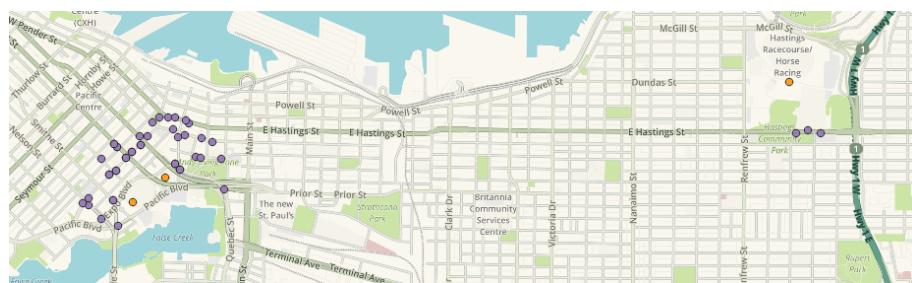


Figura 12: Mapa de estadios y red de paradas y rutas en un radio de 600 m.

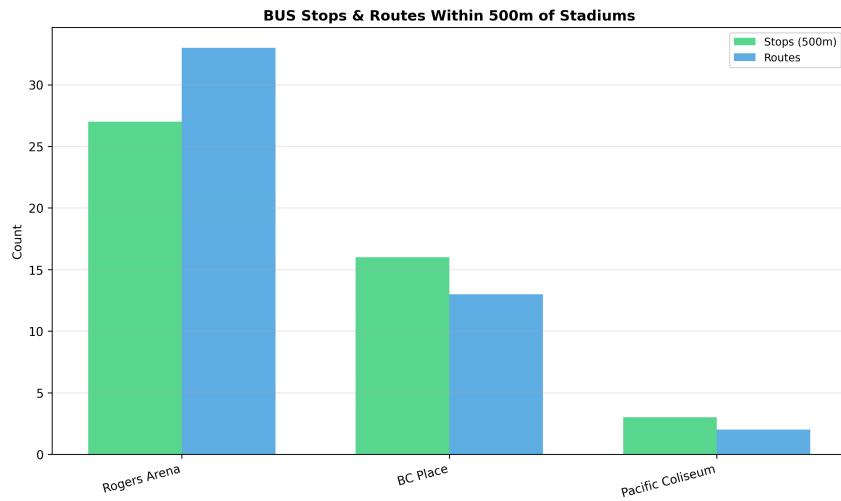


Figura 13: Paradas y rutas que sirven a cada estadio dentro de 600 m.

El mapa muestra que BC Place y Rogers Arena se encuentran en un nodo concentrado de paradas y rutas de colectivo, mientras que otros recintos presentan menor densidad de servicio en su entorno inmediato.

El script `stadium_proximity_analysis.py` complementa la lectura espacial con un gráfico de oferta diaria:

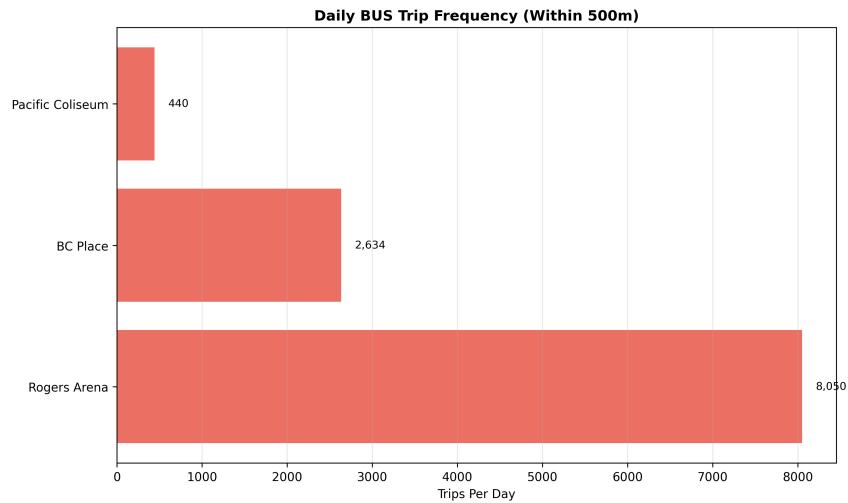


Figura 14: Viajes diarios que pasan cerca de cada estadio (radio de 600 m).

En este gráfico se observa que los estadios céntricos reciben significativamente más viajes diarios que aquellos ubicados en zonas residenciales o periféricas.

3.6. Conectividad entre estadios y áreas de alta densidad

Finalmente se analiza cómo los estadios se conectan con las zonas de mayor densidad poblacional. La consulta `12_stadium_population_overlay.sql` construye la vista `qgis_stadium_population_overlay`, que mide cuántas áreas densas alcanza cada estadio, cuánta población queda conectada y qué tan intensa es la red de segmentos que realiza esa conexión. La consulta `13_stadium_density_connecting_routes.sql` devuelve las rutas completas que vinculan estadios con áreas de alta densidad, usadas como capa en QGIS.

Un fragmento representativo de la agregación de conectividad por estadio es:

```
WITH stadium_connectivity_stats AS (
    SELECT
        s.id AS stadium_id,
        s.name AS stadium_name,
        s.team,
        s.geom AS stadium_geom,
        COUNT(DISTINCT stc.density_area_id) AS
            num_high_density_areas_connected,
        COALESCE(SUM(stc.population), 0) AS
            total_population_connected,
        COALESCE(SUM(rs.total_route_segments), 0) AS
            total_connecting_segments,
        COALESCE(SUM(rs.total_route_length_km), 0) AS
            total_route_length_km
    FROM football_stadiums s
    LEFT JOIN stadium_to_density_connectivity stc
        ON s.id = stc.stadium_id
    LEFT JOIN route_statistics rs
        ON stc.route_id = rs.route_id
    GROUP BY s.id, s.name, s.team, s.geom
)
CREATE MATERIALIZED VIEW qgis_stadium_population_overlay AS
SELECT
    stadium_name,
    team,
    num_high_density_areas_connected,
    total_population_connected,
    total_connecting_segments,
    total_route_length_km,
    stadium_geom AS geom
FROM stadium_connectivity_stats;
```

De forma análoga, `13_stadium_density_connecting_routes.sql` agrupa todos los segmentos de cada ruta conectora usando `ST_Collect` para obtener una geometría de recorrido completa que luego se utiliza en QGIS.

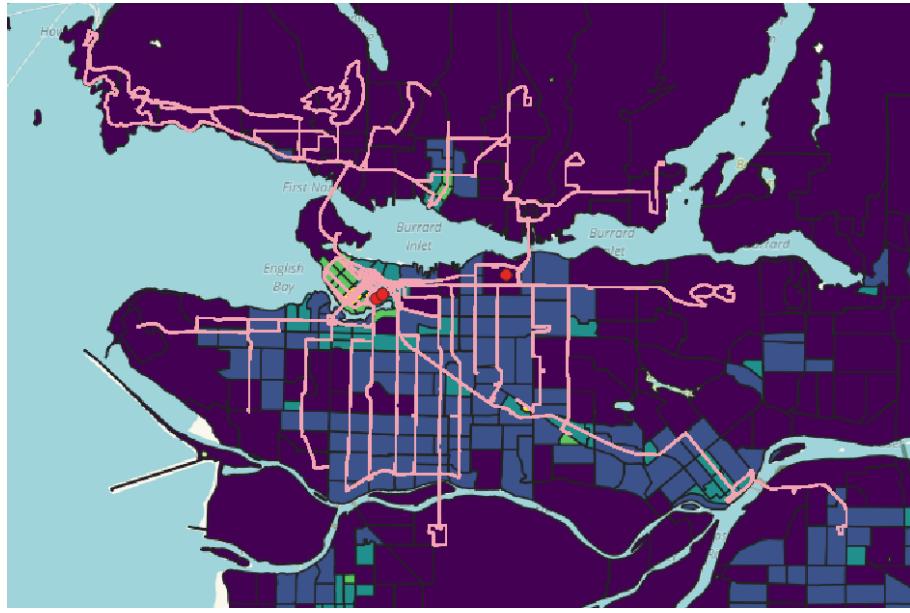


Figura 15: Rutas de colectivo que conectan estadios con áreas de alta densidad poblacional.

En el mapa se observan claramente los corredores que vinculan los estadios con las “manchas” de mayor densidad, permitiendo evaluar si la red ofrece conexiones directas o si la accesibilidad depende de transbordos complejos.

El script `stadium_population_analysis.py` genera gráficos que resumen estas métricas a nivel agregado:

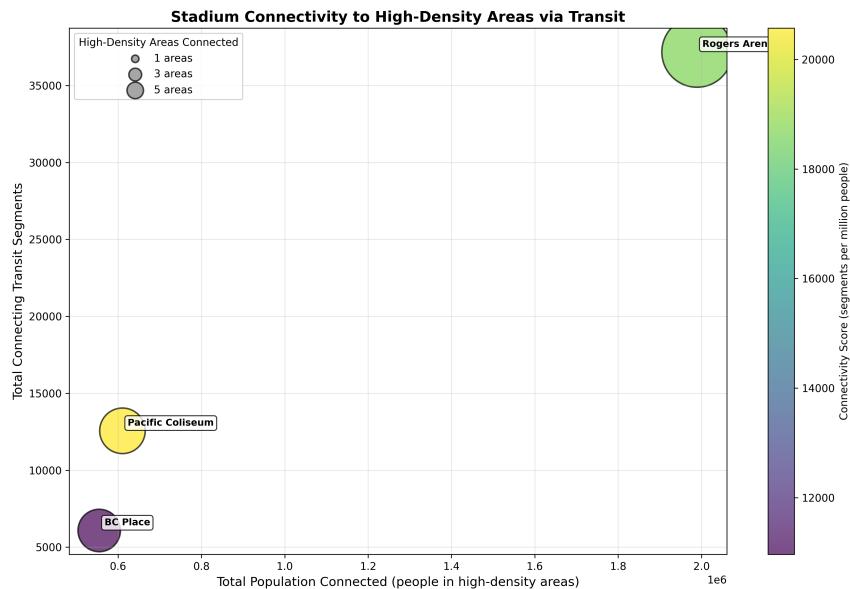


Figura 16: Relación entre población conectada y segmentos de tránsito conectores por estadio.

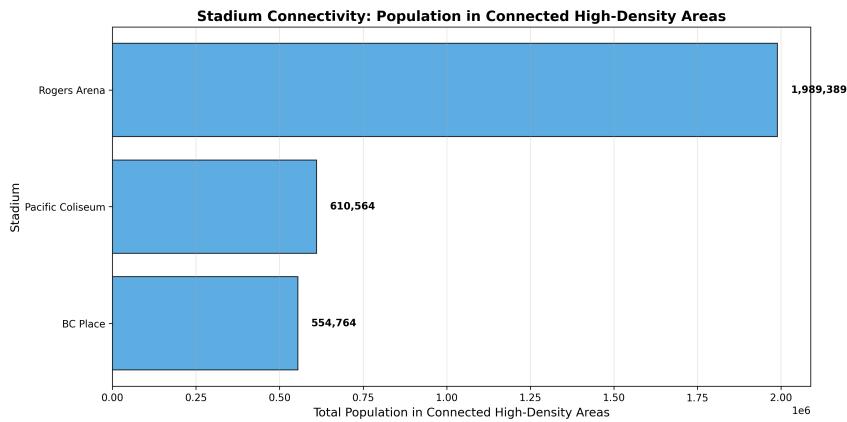


Figura 17: Población en áreas densas conectadas a cada estadio.

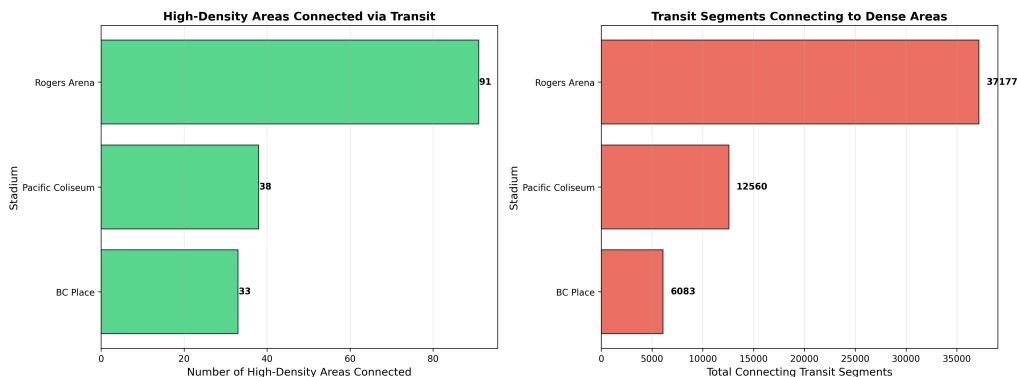


Figura 18: Comparación de áreas densas conectadas y segmentos de tránsito por estadio.

Los gráficos muestran, por un lado, qué estadios logran conectar a mayor población en áreas de alta densidad y, por otro, cuán intensa es la infraestructura de transporte que sostiene esas conexiones. En conjunto con el mapa de rutas conectoras, permiten identificar estadios con buena cobertura hacia las zonas más densas de la ciudad frente a otros cuya accesibilidad depende de unos pocos recorridos.

4. Análisis con GTFS-Realtime

Los análisis sobre datos en tiempo real se basan en las trayectorias *map-matched* almacenadas en `realtime_trips_mdb`, las cuales se comparan con sus equivalentes programados en `scheduled_trips_mdb`.

4.1. Velocidad real vs. velocidad programada

La comparación de velocidades se realiza mediante la vista materializada `realtime_speed_comparison` que calcula las velocidades programadas y observadas para cada segmento. Un esquema

simplificado de la consulta es:

```
DROP MATERIALIZED VIEW IF EXISTS realtime_speed_comparison;
CREATE MATERIALIZED VIEW realtime_speed_comparison AS
WITH with_next_stop AS (
    SELECT
        d.trip_instance_id,
        d.trip_id,
        d.route_id,
        d.service_date,
        d.stop_sequence,
        d.stop_id,
        d.actual_arrival,
        LEAD(d.stop_sequence) OVER w AS next_stop_sequence,
        LEAD(d.stop_id) OVER w AS next_stop_id,
        LEAD(d.actual_arrival) OVER w AS next_actual_arrival
    FROM rt_trip_updates_deduped d
    WINDOW w AS (PARTITION BY d.trip_instance_id ORDER BY d.
        stop_sequence)
)
SELECT
    w.trip_instance_id,
    w.trip_id,
    r.route_short_name,
    w.route_id,
    rs.seg_length AS segment_length_m,
    EXTRACT(EPOCH FROM (rs.stop2_arrival_time - rs.
        stop1_arrival_time)) AS scheduled_seconds,
    EXTRACT(EPOCH FROM (w.next_actual_arrival - w.actual_arrival))
        AS actual_seconds,
    (rs.seg_length / NULLIF(EXTRACT(EPOCH FROM (rs.
        stop2_arrival_time - rs.stop1_arrival_time)), 0) * 3.6) AS
        scheduled_speed_kmh,
    (rs.seg_length / NULLIF(EXTRACT(EPOCH FROM (w.
        next_actual_arrival - w.actual_arrival)), 0) * 3.6) AS
        actual_speed_kmh
FROM with_next_stop w
JOIN route_segments rs
    ON rs.trip_id = w.trip_id
    AND rs.stop1_sequence = w.stop_sequence
LEFT JOIN routes r ON r.route_id = w.route_id
WHERE w.next_actual_arrival IS NOT NULL
    AND rs.seg_length > 10
    AND EXTRACT(EPOCH FROM (rs.stop2_arrival_time - rs.
        stop1_arrival_time)) > 0
    AND EXTRACT(EPOCH FROM (w.next_actual_arrival - w.
        actual_arrival)) > 0;
```

A partir de esta vista se generan tanto mapas como gráficos:

- un mapa de diferencia de velocidad que resume espacialmente qué tramos son sistemáticamente más lentos o más rápidos que lo previsto,
- histogramas y gráficos por ruta y por hora que cuantifican estas diferencias.

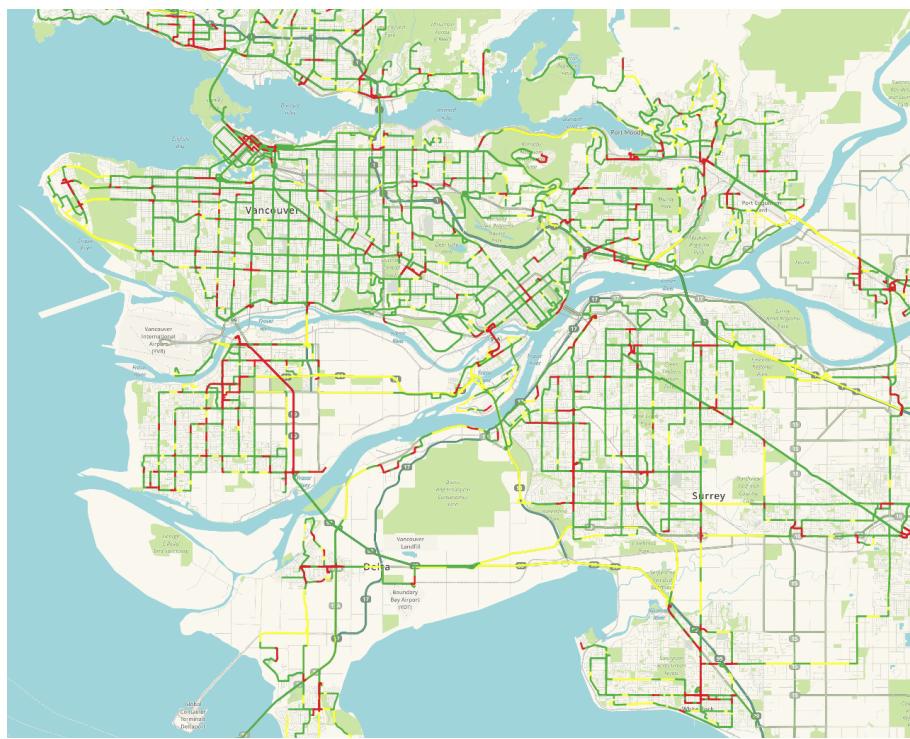


Figura 19: Mapa de diferencia de velocidad real vs. programada (rojo = más lento que lo programado, verde ≈ en línea con lo programado, amarillo = más rápido).

En este mapa se identifican rápidamente corredores donde los buses circulan de forma sistemáticamente más lenta que lo previsto (en rojo), así como tramos que tienden a adelantarse sobre el horario (en amarillo).

A nivel agregado, se utilizan gráficos para resumir la distribución de estas diferencias:

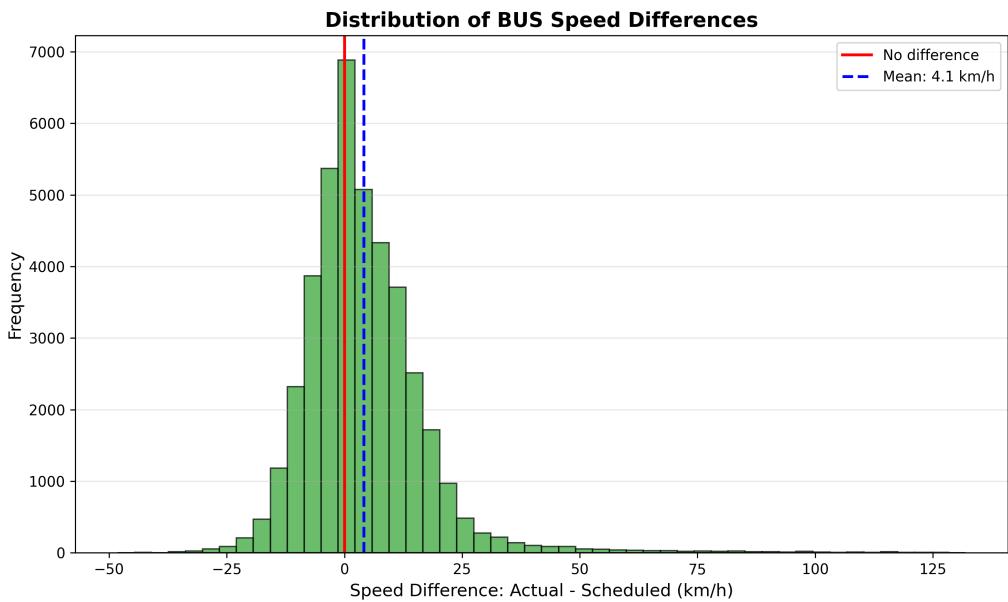


Figura 20: Distribución de la diferencia de velocidad (real – programada).

El histograma de diferencias muestra una distribución aproximadamente unimodal, con una masa importante de segmentos con velocidades similares a las previstas y colas que capturan tanto episodios de fuerte sobreejecución como segmentos marcadamente más lentos que el plan.

4.2. Desempeño de horarios (schedule times)

La comparación de horarios programados vs. observados se realiza mediante la vista materializada `realtime_schedule_times`, que registra para cada parada la hora prevista y la hora efectivamente observada. Un fragmento representativo es:

```
DROP MATERIALIZED VIEW IF EXISTS realtime_schedule_times;
CREATE MATERIALIZED VIEW realtime_schedule_times AS
SELECT
    d.trip_instance_id,
    d.trip_id,
    r.route_short_name,
    r.route_long_name,
    d.route_id,
    d.service_date,
    d.stop_sequence,
    d.stop_id,
    s.stop_name,
    ts.arrival_time AS scheduled_arrival_interval,
    d.actual_arrival,
    d.actual_departure,
    d.arrival_delay_seconds,
```

```

d.departure_delay_seconds,
d.arrival_delay_seconds / 60.0 AS delay_minutes,
EXTRACT(hour FROM d.actual_arrival) AS hour_of_day,
EXTRACT(dow FROM d.actual_arrival) AS day_of_week,
CASE
    WHEN EXTRACT(dow FROM d.actual_arrival) IN (0, 6) THEN '
        Weekend'
    ELSE 'Weekday'
END AS day_type
FROM rt_trip_updates_deduped d
JOIN routes r ON r.route_id = d.route_id
LEFT JOIN stops s ON s.stop_id = d.stop_id
LEFT JOIN transit_stops ts
    ON ts.trip_id = d.trip_id
    AND ts.stop_sequence = d.stop_sequence
WHERE d.arrival_delay_seconds IS NOT NULL;

```

A partir de esta vista se construye un mapa de puntualidad y varios gráficos agregados.

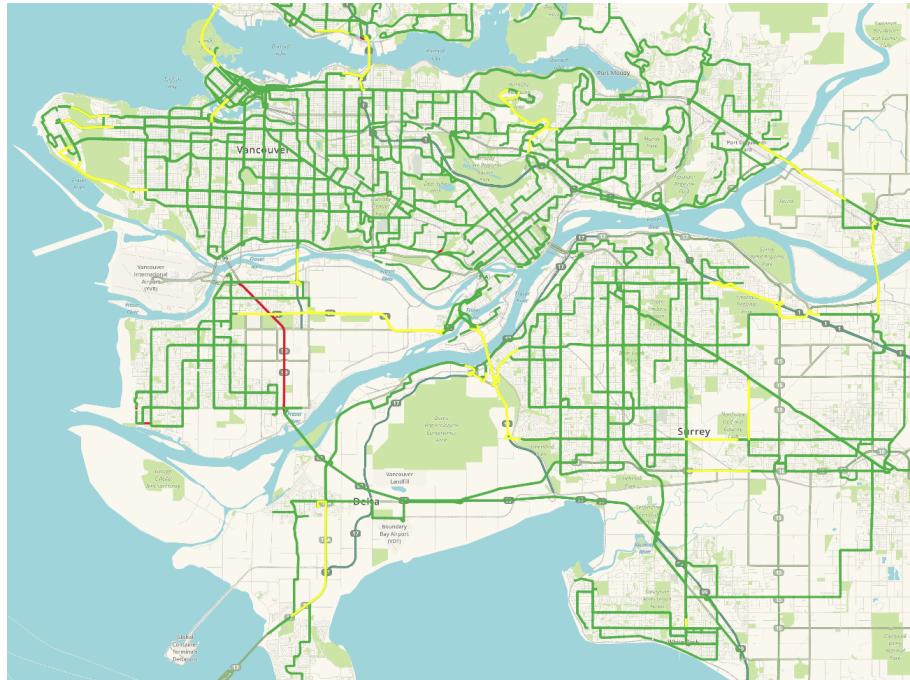


Figura 21: Mapa de desempeño horario (amarillo = adelantado ≥ 3 min, verde = dentro de ± 3 min, rojo = retrasado ≥ 3 min).

El mapa permite ubicar espacialmente los puntos donde el sistema opera de forma mayormente puntual (en verde) frente a corredores donde predominan los retrasos o adelantos marcados.

A nivel estadístico, se analizan las distribuciones de demora y su variación por hora y por ruta:

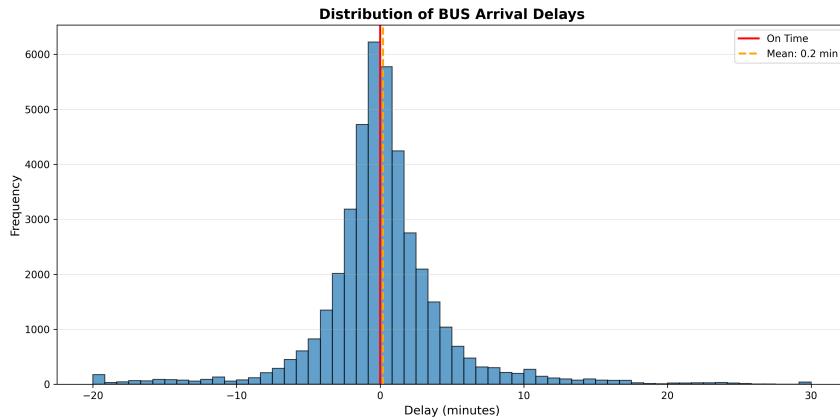


Figura 22: Distribución de demoras respecto al horario programado.

Estos gráficos muestran una cola hacia valores positivos que indica predominio de demoras por encima del horario programado, especialmente en horas pico y en determinadas rutas troncales.

4.3. Segmentos de demora y patrones de congestión

El análisis de demoras por segmento se basa en la vista materializada `realtime_delay_analysis`, que compara para cada tramo entre paradas el tiempo programado con el tiempo efectivamente observado. La estructura básica de la vista es:

```

DROP MATERIALIZED VIEW IF EXISTS realtime_delay_analysis;
CREATE MATERIALIZED VIEW realtime_delay_analysis AS
WITH with_next AS (
    SELECT
        d.trip_instance_id,
        d.trip_id,
        d.route_id,
        d.service_date,
        d.stop_sequence AS from_seq,
        d.stop_id AS from_stop_id,
        d.actual_arrival AS from_arrival,
        d.arrival_delay_seconds AS from_delay,
        LEAD(d.stop_sequence) OVER w AS to_seq,
        LEAD(d.stop_id) OVER w AS to_stop_id,
        LEAD(d.actual_arrival) OVER w AS to_arrival,
        LEAD(d.arrival_delay_seconds) OVER w AS to_delay
    FROM rt_trip_updates_deduped d
    WINDOW w AS (PARTITION BY d.trip_instance_id ORDER BY d.
        stop_sequence)
)
SELECT
    w.trip_instance_id,
    w.trip_id,

```

```

r.route_short_name,
w.route_id,
w.from_seq,
w.to_seq,
rs.seg_length AS segment_length_m,
rs.geom,
EXTRACT(EPOCH FROM (rs.stop2_arrival_time - rs.
stop1_arrival_time)) AS scheduled_seconds,
EXTRACT(EPOCH FROM (w.to_arrival - w.from_arrival)) AS
actual_seconds,
(w.to_delay - w.from_delay) AS segment_delay_change,
(EXTRACT(EPOCH FROM (w.to_arrival - w.from_arrival)) -
EXTRACT(EPOCH FROM (rs.stop2_arrival_time - rs.
stop1_arrival_time))) / 60.0 AS segment_delay_minutes,
EXTRACT(hour FROM w.from_arrival) AS hour_of_day,
EXTRACT(dow FROM w.from_arrival) AS day_of_week,
CASE
    WHEN EXTRACT(dow FROM w.from_arrival) IN (0, 6) THEN '
        Weekend'
    ELSE 'Weekday'
END AS day_type,
CASE
    WHEN EXTRACT(hour FROM w.from_arrival) BETWEEN 7 AND 9
        THEN 'Morning_Rush'
    WHEN EXTRACT(hour FROM w.from_arrival) BETWEEN 16 AND 18
        THEN 'Evening_Rush'
    WHEN EXTRACT(hour FROM w.from_arrival) BETWEEN 9 AND 16
        THEN 'Midday'
    WHEN EXTRACT(hour FROM w.from_arrival) BETWEEN 18 AND 22
        THEN 'Evening'
    ELSE 'Night'
END AS time_period
FROM with_next w
JOIN route_segments rs
    ON rs.trip_id = w.trip_id
    AND rs.stop1_sequence = w.from_seq
LEFT JOIN routes r ON r.route_id = w.route_id
WHERE w.to_arrival IS NOT NULL
    AND rs.seg_length > 10
    AND EXTRACT(EPOCH FROM (rs.stop2_arrival_time - rs.
stop1_arrival_time)) > 0
    AND EXTRACT(EPOCH FROM (w.to_arrival - w.from_arrival)) > 0;

```

A partir de esta vista se generan:

- mapas de los “peores” segmentos en términos de demora promedio,
- gráficos por hora, tipo de día y severidad de la demora.

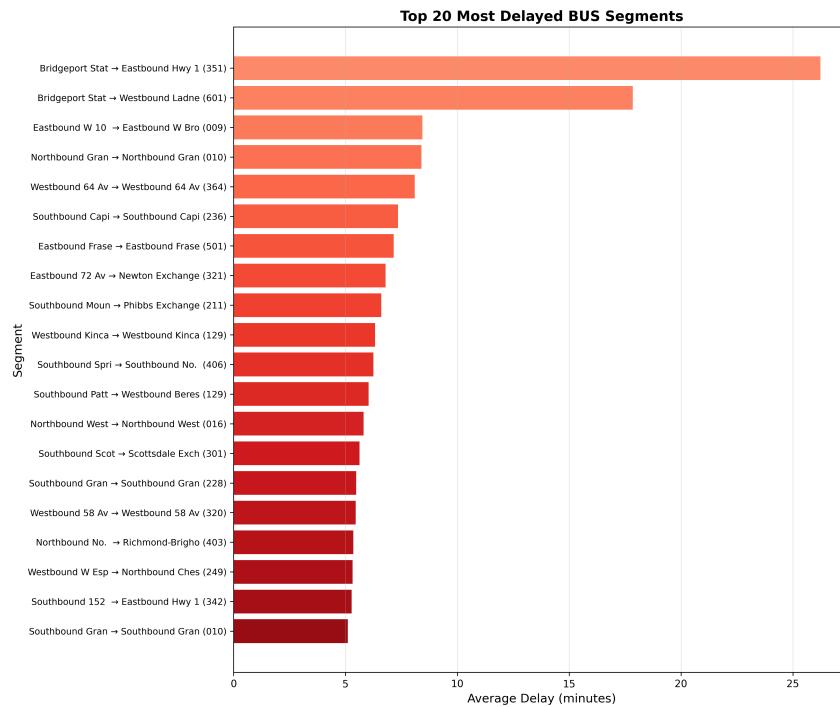


Figura 23: Segmentos con mayores demoras promedio.

La cartografía de segmentos problemáticos, combinada con gráficos de distribución por franja horaria, permite identificar cuellos de botella recurrentes donde la operación se degrada sistemáticamente respecto del plan.

4.4. Regularidad de headways y *bus bunching*

El análisis de headways se basa en la vista materializada `realtime_headway_stats`, que calcula los intervalos entre vehículos consecutivos en una misma parada y ruta. Un fragmento de la consulta es:

```
DROP MATERIALIZED VIEW IF EXISTS realtime_headway_stats;
CREATE MATERIALIZED VIEW realtime_headway_stats AS
WITH stop_arrivals AS (
    SELECT
        rtu.route_id,
        r.route_short_name,
        rtu.stop_id,
        s.stop_name,
        rtu.trip_instance_id,
        rtu.trip_id,
        rtu.arrival_time,
        EXTRACT(hour FROM rtu.arrival_time) AS hour_of_day,
        EXTRACT(dow FROM rtu.arrival_time) AS day_of_week,
        CASE
```

```

    WHEN EXTRACT(dow FROM rtu.arrival_time) IN (0, 6) THEN '
        Weekend'
    ELSE 'Weekday'
END AS day_type
FROM rt_trip_updates rtu
JOIN routes r ON r.route_id = rtu.route_id
LEFT JOIN stops s ON s.stop_id = rtu.stop_id
WHERE rtu.arrival_time IS NOT NULL
    AND rtu.stop_id IS NOT NULL
),
with_prev AS (
    SELECT
        *,
        LAG(arrival_time) OVER (
            PARTITION BY route_id, stop_id
            ORDER BY arrival_time
        ) AS prev_arrival,
        LAG(trip_instance_id) OVER (
            PARTITION BY route_id, stop_id
            ORDER BY arrival_time
        ) AS prev_trip_instance_id
    FROM stop_arrivals
)
SELECT
    route_id,
    route_short_name,
    stop_id,
    stop_name,
    trip_instance_id,
    prev_trip_instance_id,
    arrival_time,
    prev_arrival,
    EXTRACT(EPOCH FROM (arrival_time - prev_arrival)) / 60.0 AS
        headway_minutes,
    hour_of_day,
    day_of_week,
    day_type
FROM with_prev
WHERE prev_arrival IS NOT NULL
    AND trip_instance_id != prev_trip_instance_id
    AND EXTRACT(EPOCH FROM (arrival_time - prev_arrival)) > 0
    AND EXTRACT(EPOCH FROM (arrival_time - prev_arrival)) < 7200;

```

El script `headway_analysis.py` utiliza esta vista para clasificar los intervalos como:

- **Bunched** (< 3 minutos): vehículos demasiado próximos.
- **Good** (3–10 minutos): intervalo deseable en rutas frecuentes.

- **Acceptable** (10–20 minutos): servicio menos frecuente pero razonable.
- **Gap** (> 20 minutos): esperas prolongadas.

La Figura 24 muestra la distribución de headways obtenida a partir de los registros realtime.

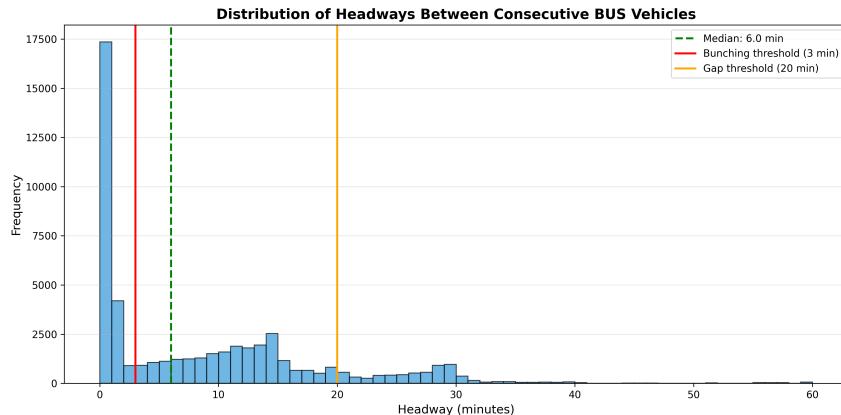


Figura 24: Distribución de headways entre vehículos.

Otros gráficos (no mostrados aquí) exploran la proporción de headways en cada categoría por ruta y por periodo del día, permitiendo cuantificar la incidencia del *bus bunching* y de las brechas de servicio prolongadas.

5. Discusión y trabajo futuro

El pipeline desarrollado permite integrar datos GTFS estáticos y GTFS-Realtime en una única base de datos espacial y reproducir de forma sistemática una serie de consultas y visualizaciones sobre la red de transporte de Vancouver. La separación clara entre scripts de ingesta, consultas SQL y scripts de visualización facilita la extensión del análisis a nuevas ciudades o a períodos de tiempo adicionales.

Como trabajo futuro se identifican varias líneas de avance: incorporar análisis de robustez ante fallas (por ejemplo, cierres de estaciones), extender las métricas de confiabilidad al nivel de pasajero (tiempos puerta a puerta) e integrar datos de demanda (conteos o validaciones) para estudiar la relación entre oferta, congestión y ocupación vehicular.