



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	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. 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	15
3.6. Conectividad entre estadios y áreas de alta densidad	17
4. Análisis con GTFS-Realtime	19
4.1. Velocidad real vs. velocidad programada	19
4.2. Desempeño de horarios (schedule times)	22
4.3. Regularidad de headways y <i>bus bunching</i>	25
4.4. Headway observado vs. headway programado	28

5. Discusión de resultados 30

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.

Nota: todos los datos utilizados para este análisis pueden encontrarse en https://drive.google.com/drive/folders/19Hic_CPM1u5DPWin90Dc3BLOUnqnwCB3?usp=sharing.

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).

Para un listado completo de prerrequisitos (incluyendo instalación de GDAL/rasterio/contextily), revisar la sección **Prerequisites** del [README.md](#) del proyecto.

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

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

Primero, generar las vistas materializadas necesarias para los análisis:

```
python static_analysis/queries/sql/run_sql.py
```

Luego, ejecutar el orquestador que refresca las vistas y genera las gráficas:

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

Este comando:

- Ejecuta los archivos SQL en `static_analysis/queries/sql/`, creando vistas materializadas para visualización en herramientas de mapeo (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.

2.4. Flujo GTFS-Realtime

2.4.1. Configuración y tablas realtime

Instalar dependencias adicionales y configurar la API de TransLink:

```
export TRANSLINK_GTFSR_API_KEY="your-translink-api-key"
```

Crear las tablas necesarias para datos realtime:

```
cat realtime_analysis/data_loading/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 desde el root del repositorio:

```
python -m realtime_analysis.data.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.data.build_realtime_trajectories
```

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. Consultas y análisis agregados con GTFS-Realtime

Primero, generar las vistas materializadas para los análisis:

```
python realtime_analysis/queries/sql/run_sql.py
```

Luego, ejecutar el orquestador que refresca las vistas y genera gráficos:

```
python -m realtime_analysis.queries.run_all_analyses
```

El comando anterior ejecuta los archivos SQL de `realtime_analysis/queries/sql/`, creando vistas `realtime_*` y vistas de visualización, y luego corre los scripts 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 para calcular el porcentaje de rutas por modo.

```
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;
```

Se presentan a continuación el mapa completo de la red y la distribución de viajes por ruta:



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

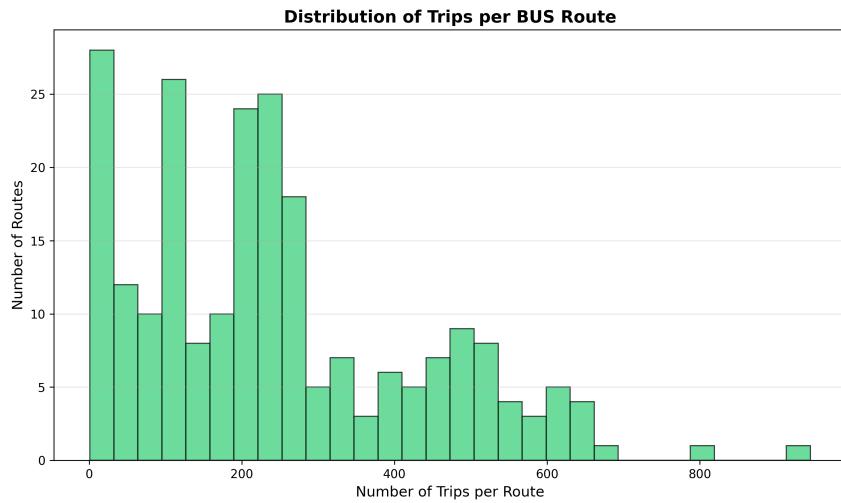


Figura 2: Distribución de viajes diarios por ruta de colectivo.

El mapa evidencia una red especialmente densa en el centro de Vancouver y en los corredores troncales hacia el este y el sur. Podemos ver el color amarillo abundante en el mapa, lo cual nos indica que las únicas rutas para analizar son las de buses. Por eso, nuestros análisis estarán restringidos a las rutas de buses. 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, agrupando los tramos por geometría y contando las rutas distintas que pasan por cada uno.

```
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;
```

Los resultados visuales derivados de esta vista son:

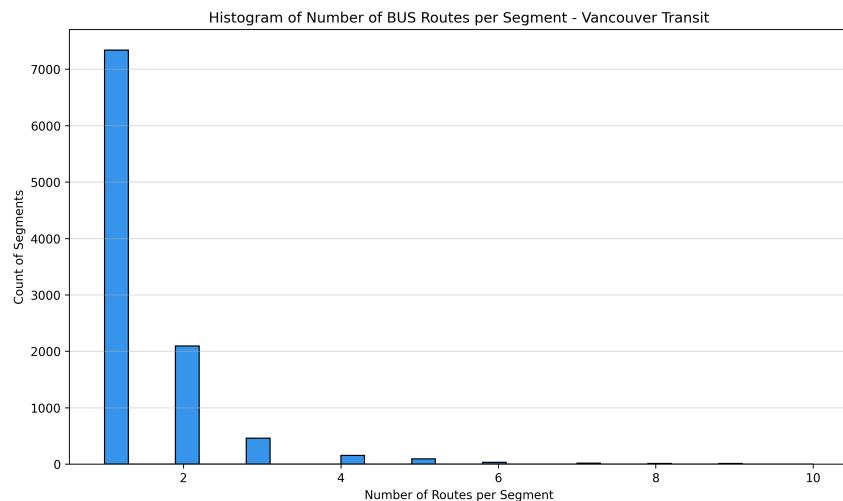


Figura 3: Histograma de cantidad de rutas por segmento.

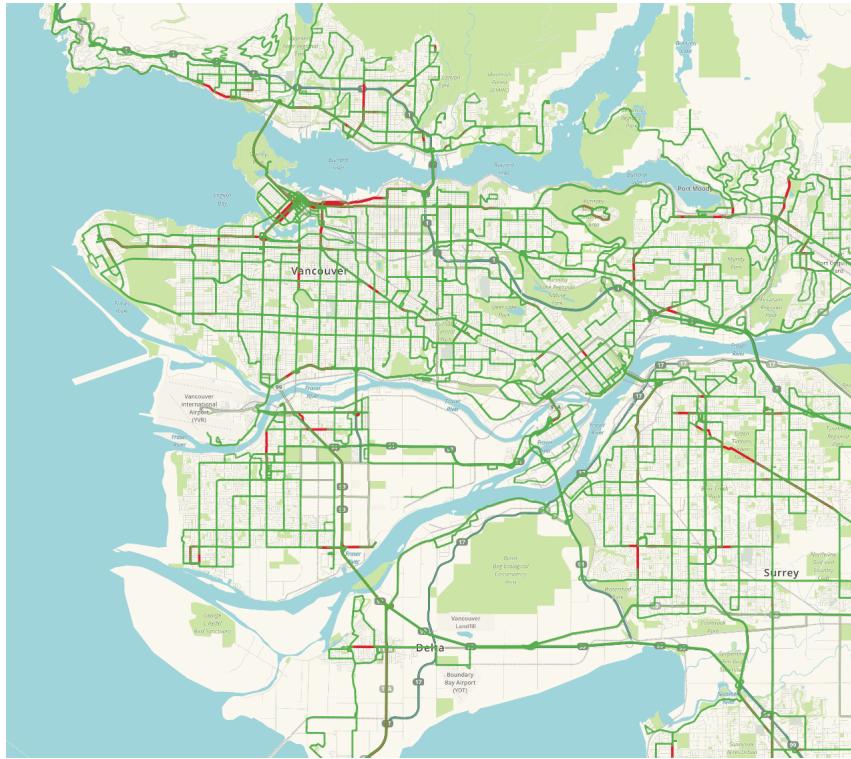


Figura 4: Mapa de densidad de rutas sobre la red de segmentos (verde = 1–2 rutas; rojo = 3 o más rutas por segmento).

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 la geometría de los segmentos con los horarios de llegada previstos. Esta información se materializa en una vista que calcula la velocidad en km/h para cada tramo entre paradas.

```

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;
```

A continuación se muestran los mapas de velocidad sobre la red y el histograma de distribución:

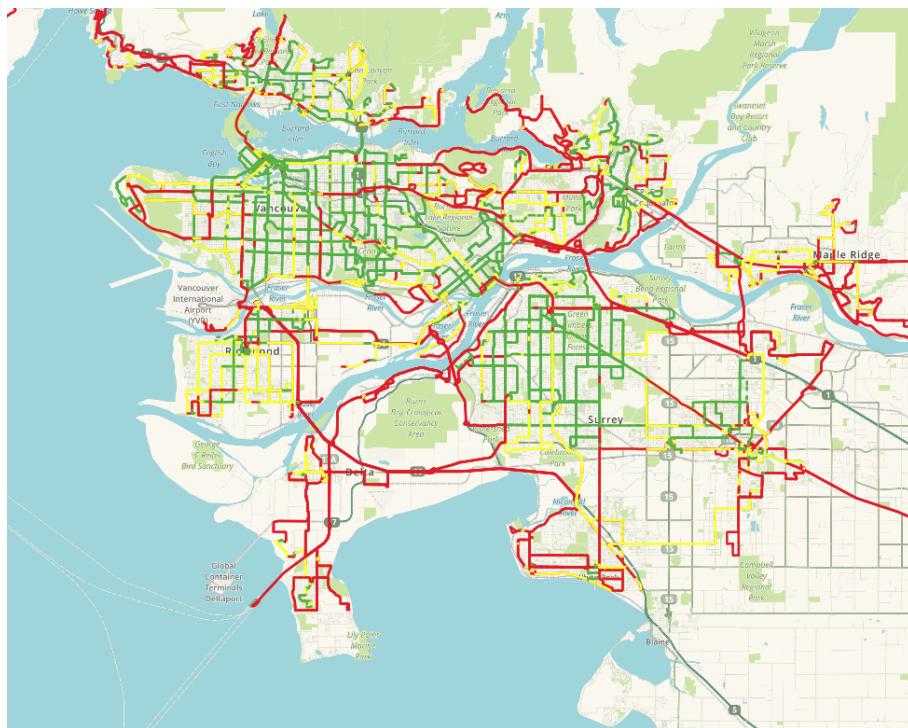


Figura 5: Mapa de velocidades programadas sobre la red de segmentos (3.5–19 km/h = verde punteado; 19–23 km/h = verde; 23–30 km/h = amarillo; >30 km/h = rojo).

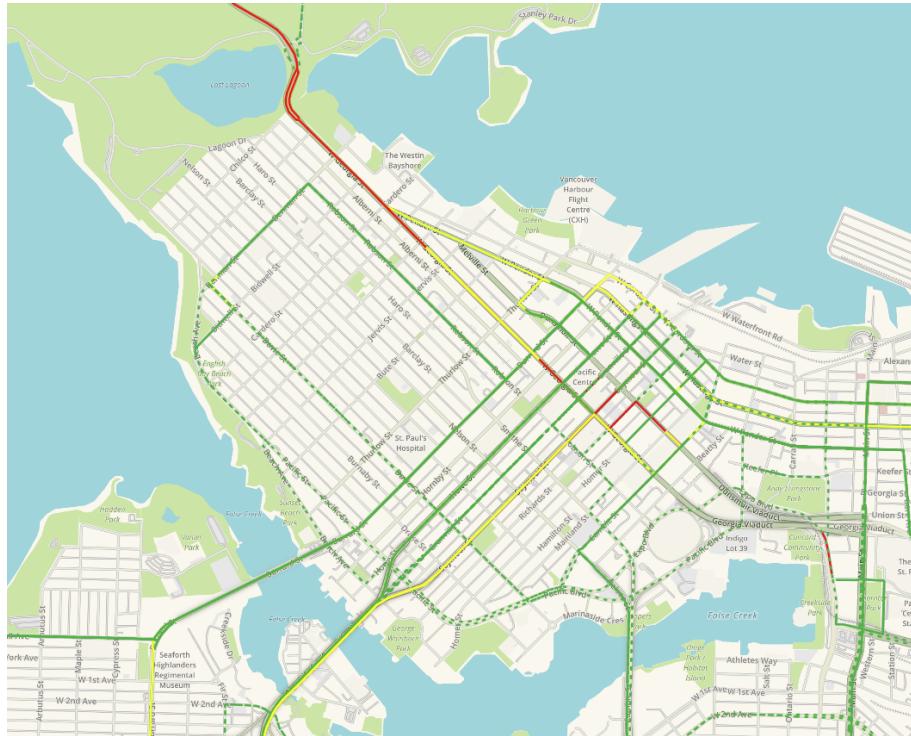


Figura 6: Detalle de zona más céntrica de la ciudad (más congestión).

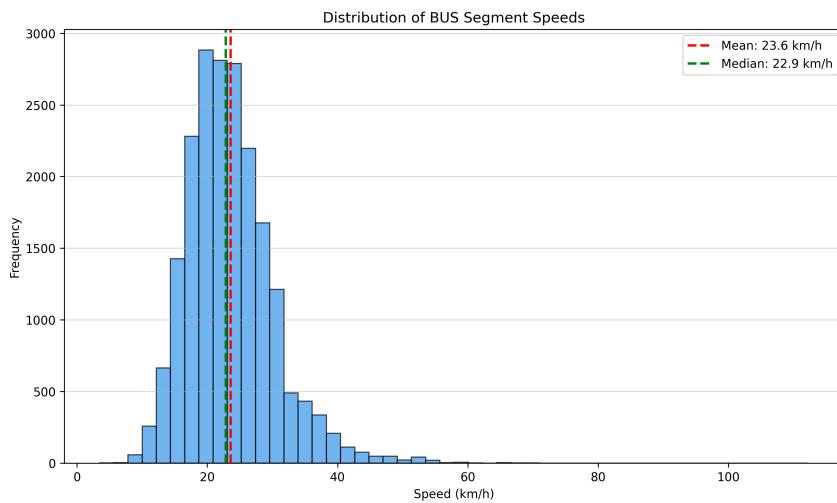


Figura 7: Histograma de velocidades programadas por segmento.

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 con datos de población. Se construye una vista que superpone estas áreas con la red de rutas, calculando métricas de cobertura como la longitud de rutas por kilómetro cuadrado.

```
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;
```

Los siguientes mapas y gráficos ilustran la relación entre población y transporte:

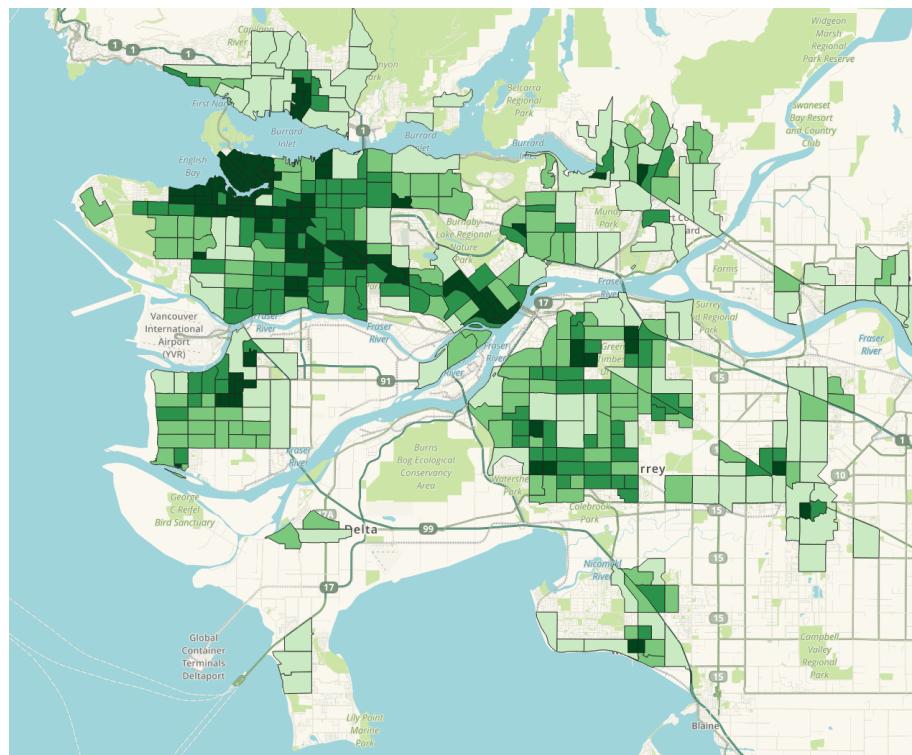


Figura 8: Mapa de densidad de población por área censal.

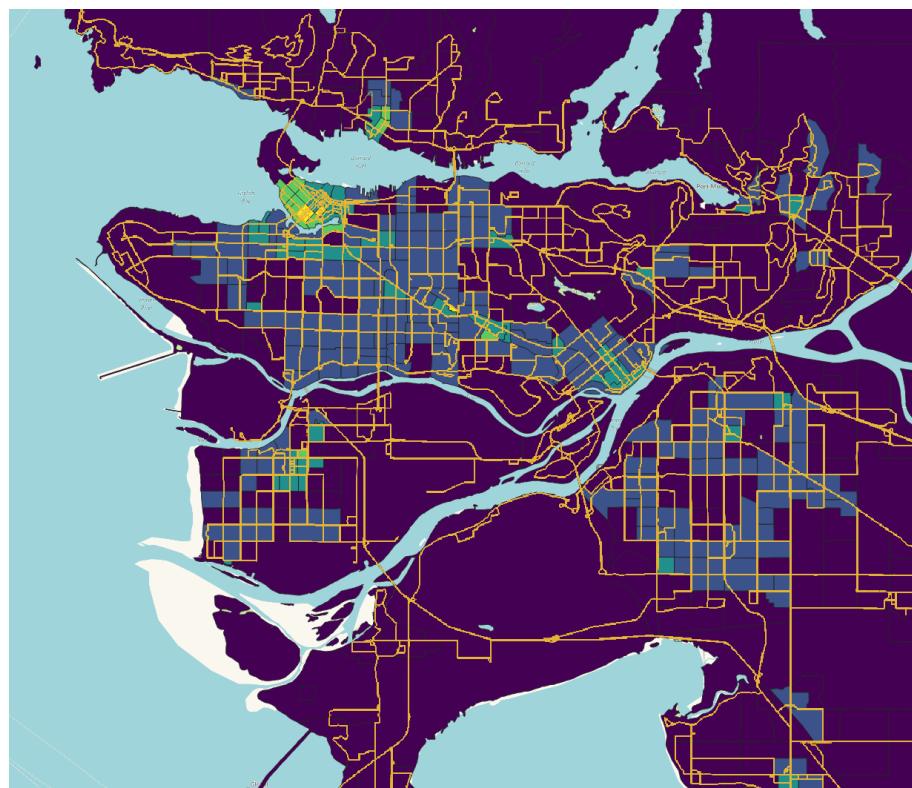


Figura 9: Superposición de densidad de población y red de rutas de colectivo.

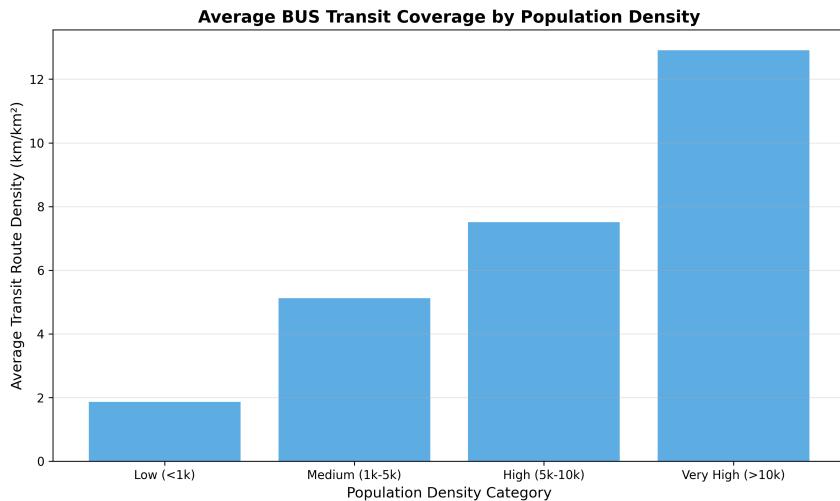


Figura 10: Cobertura de transporte por categoría de densidad poblacional.

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. 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.

3.5. Accesibilidad local a estadios

Los estadios se modelan como puntos de interés y se analiza la oferta de transporte en un radio de 600 m. La consulta identifica las paradas y rutas cercanas, generando una vista materializada para evaluar la accesibilidad local de cada recinto.

```

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;

```

El análisis de proximidad produce los siguientes mapas y gráficos de oferta:

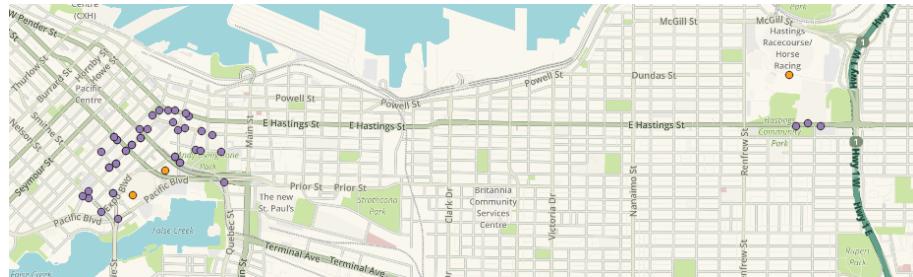


Figura 11: Mapa de estadios (naranja) y red de paradas (violeta) en un radio de 600 m.

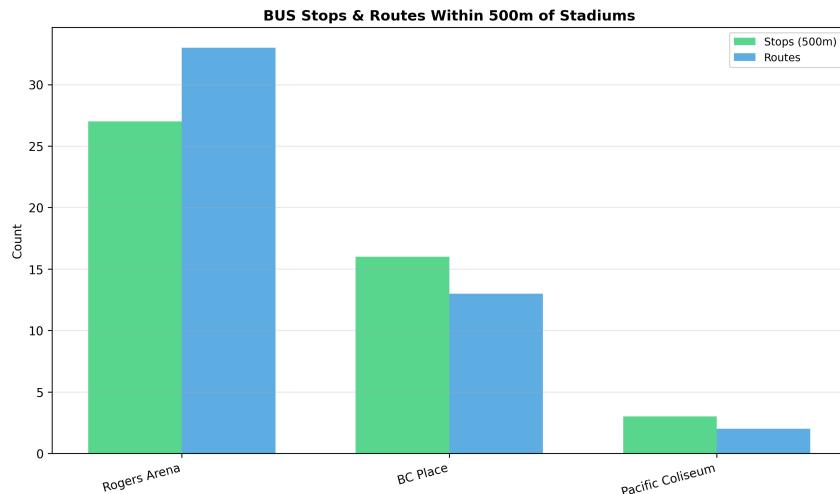


Figura 12: Paradas y rutas que sirven a cada estadio dentro de 600 m.

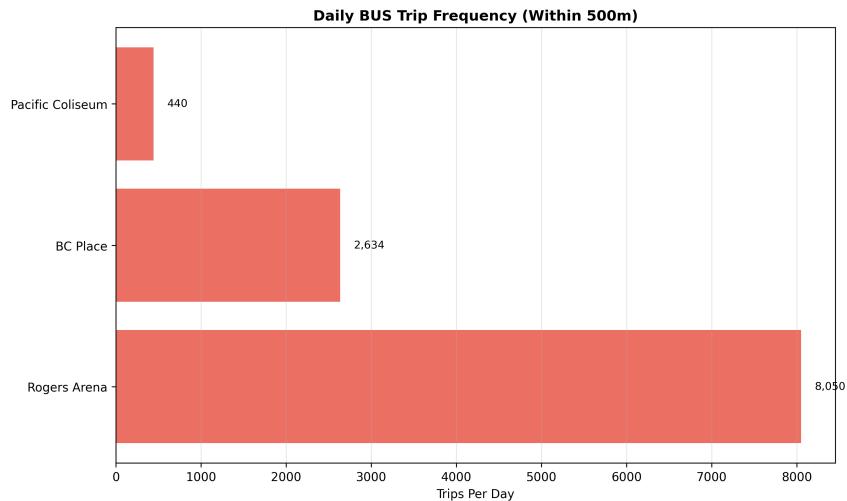


Figura 13: Viajes diarios que pasan cerca de cada estadio (radio de 600 m).

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 gráfico de barras confirma que los estadios céntricos reciben significativamente más viajes diarios que aquellos ubicados en zonas residenciales o periféricas.

Es interesante ver que aunque BC Place y Rogers Arena se encuentran muy cerca, su pequeña diferencia en distancia hace que BC Place reciba muchos menos viajes diarios.

3.6. Conectividad entre estadios y áreas de alta densidad

Finalmente se analiza la conectividad directa entre los estadios y las zonas de alta densidad poblacional. La consulta mide cuántas áreas densas alcanza cada estadio y genera geometrías completas de las rutas que realizan esta conexión.

```
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;


```

La conectividad con áreas densas se visualiza en los siguientes gráficos y mapas:

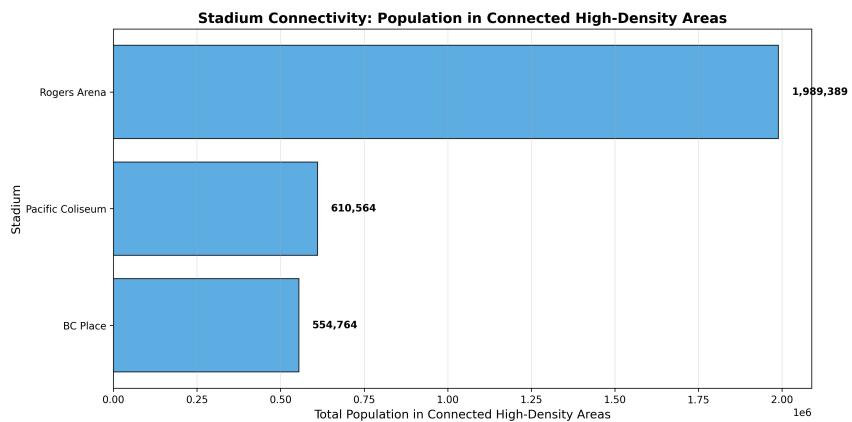


Figura 14: Población en áreas densas conectadas a cada estadio.

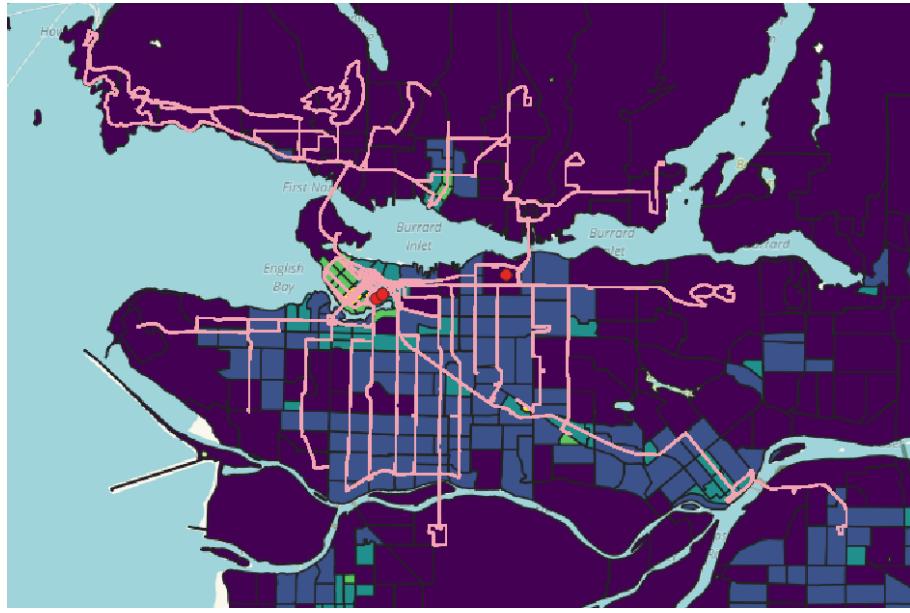


Figura 15: Rutas de colectivo que conectan estadios con áreas de alta densidad poblacional.

Los gráficos y mapas muestran, por un lado, qué estadios se asocian con mayor población en áreas de alta densidad y, por otro, los corredores principales que materializan esas conexiones sobre la red de transporte.

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`. Los resultados presentados en esta sección se basan en datos recolectados el 7 de diciembre de 2025 (fin de semana, domingo) entre las 7am y las 11am (hora local de Vancouver) y datos del 8 de diciembre de 2025 (dia de semana, lunes) entre las 8am y las 11am (hora local de Vancouver).

4.1. Velocidad real vs. velocidad programada

La comparación de velocidades se realiza mediante una vista materializada que calcula las velocidades programadas y observadas para cada segmento, permitiendo identificar desviaciones sistemáticas en la operación.

```

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;

```

La comparación espacial y estadística se resume en las siguientes figuras:

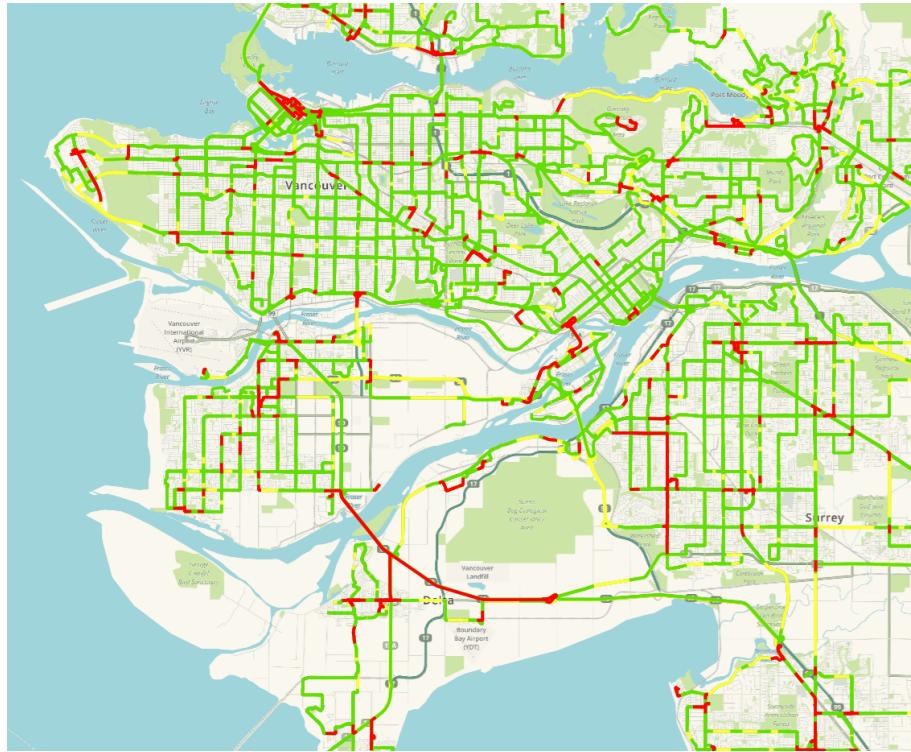


Figura 16: Mapa de diferencia de velocidad real vs. programada (rojo = más lento que lo programado, verde ≈ en línea con lo programado, amarillo = más rápido).

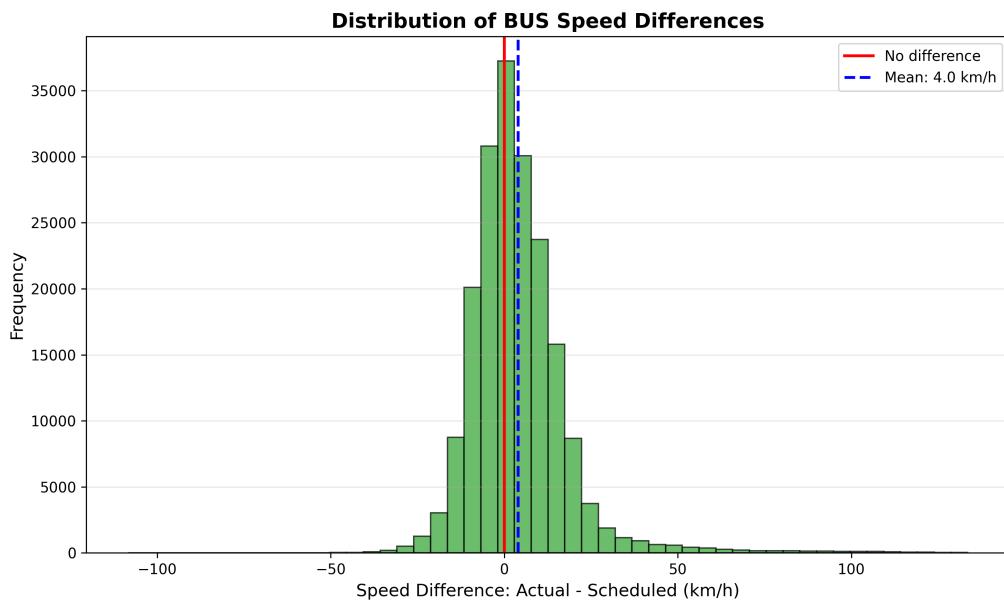


Figura 17: Distribución de la diferencia de velocidad (real – programada).

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). 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.

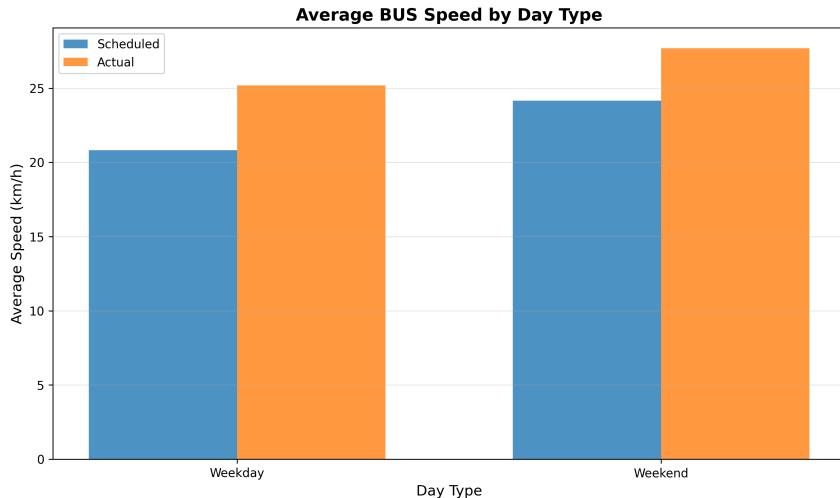


Figura 18: Comparación de velocidades programadas vs. observadas por tipo de día (día de semana vs. fin de semana).

Con los datos obtenidos, tenemos en cuenta tanto los fines de semana como los días de semana para poder tener una vista completa del funcionamiento del servicio. El gráfico muestra que las velocidades promedio son mayores durante los fines de semana, tanto programadas como observadas, lo cual es consistente con menor congestión vehicular en esos días.

4.2. Desempeño de horarios (schedule times)

La comparación de horarios programados vs. observados se realiza mediante una vista que registra para cada parada la hora prevista y la hora efectivamente observada, calculando la demora en minutos para cada evento de llegada.

```

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;

```

El desempeño de puntualidad se ilustra mediante un mapa y un histograma de demoras:

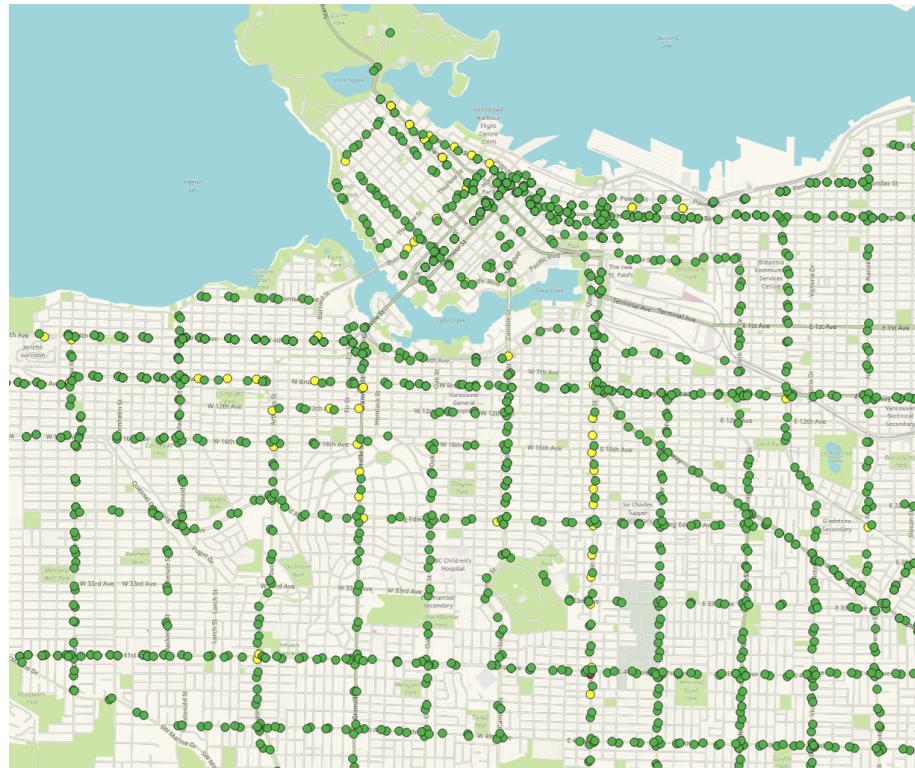


Figura 19: Mapa de desempeño horario en paradas (verde = dentro de ± 3 min, amarillo = $\pm 2\text{--}7$ min, rojo = más de ± 7 min).

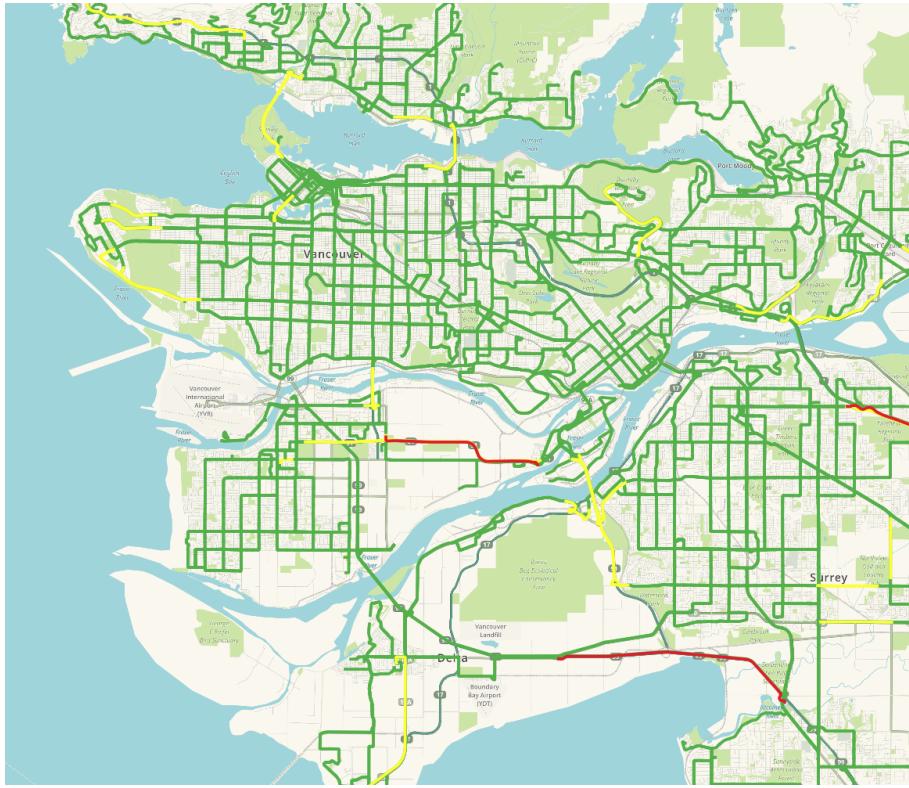


Figura 20: Mapa de demoras por segmento (entre paradas consecutivas) (verde = dentro de ± 3 min, amarillo = ± 2 – 7 min, rojo = más de ± 7 min).

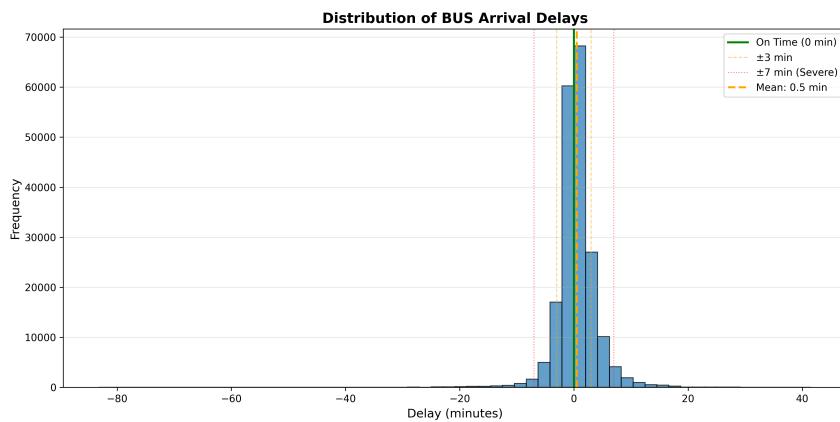


Figura 21: Distribución de demoras respecto al horario programado.

El primer mapa muestra las demoras acumuladas en cada parada (demora total desde el inicio del viaje), mientras que el segundo mapa muestra las demoras incrementales por segmento (demora acumulada durante el trayecto entre dos paradas consecutivas). Ambos mapas permiten ubicar espacialmente los puntos donde el sistema opera de forma mayormente puntual (en verde) frente a corredores donde predominan los retrasos o adelantos marcados. Los gráficos muestran una cola hacia valores positivos que indica un muy leve predominio de demoras por encima del horario programado, especialmente

en calles troncales. Sin embargo, podemos ver que los horarios se suelen cumplir con precisión, observando que la mayoría del mapa se encuentra en verde.

4.3. Regularidad de headways y *bus bunching*

El análisis de headways calcula los intervalos entre vehículos consecutivos en una misma parada y ruta. La vista materializada permite clasificar estos intervalos en categorías de regularidad y detectar fenómenos de *bus bunching*.

```
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;

```

La regularidad del servicio se visualiza a través de la distribución de headways y su clasificación por categorías:

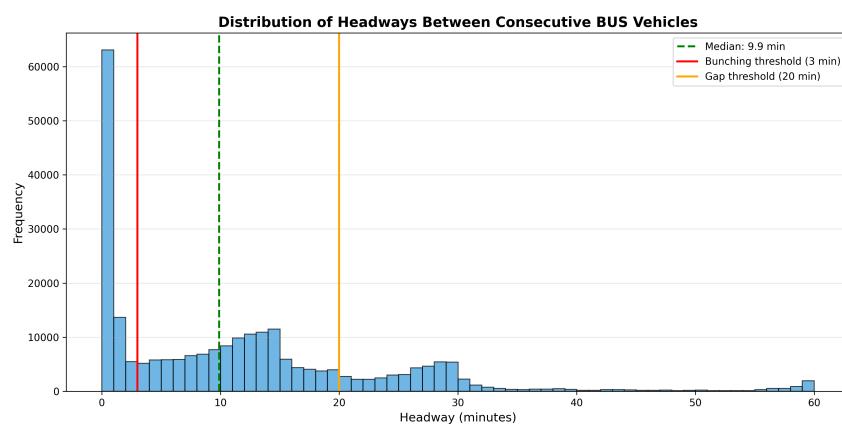


Figura 22: Distribución de headways entre vehículos.

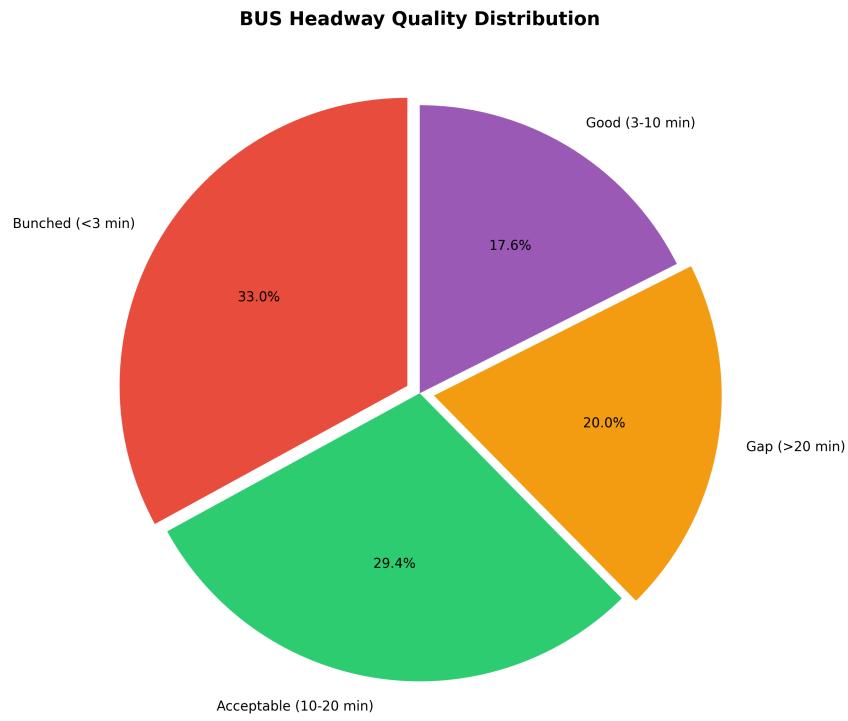


Figura 23: Proporción de headways por categoría de regularidad.

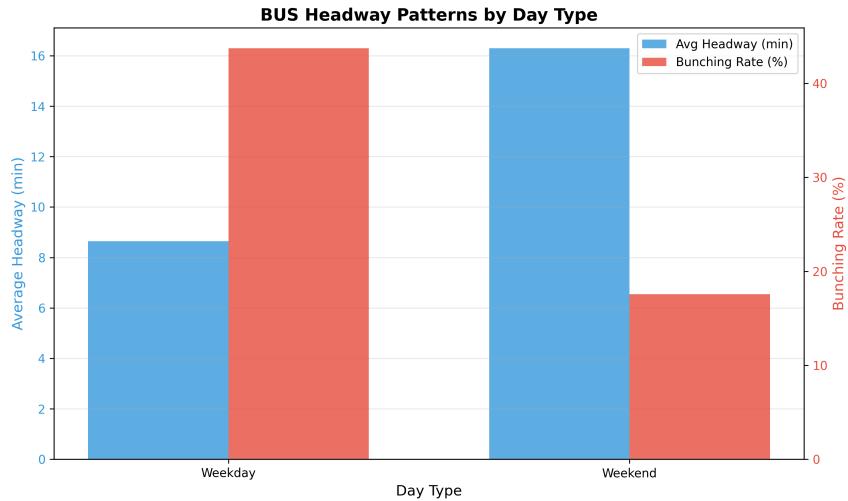


Figura 24: Comparación de headways y tasa de *bunching* por tipo de día (día de semana vs. fin de semana).

Los gráficos muestran la distribución de intervalos entre vehículos y la proporción de headways en cada categoría por ruta, lo que permite cuantificar la incidencia del *bus bunching* y de las brechas de servicio prolongadas. La comparación por tipo de día revela que los fines de semana presentan headways promedio más largos y menor incidencia de *bunching*, reflejando una menor frecuencia de servicio durante esos días. Debido a la hora

y días de los datos, creemos que el headway analizado es representativo del servicio en su totalidad.

4.4. Headway observado vs. headway programado

Se creó la vista `qgis_realtime_headway_vs_schedule` para comparar el headway observado en cada parada con el headway programado. Se filtran pares parada–ruta con al menos tres observaciones y se calculan: headway observado, headway programado, diferencia (observado – programado) y tasas de *bunching* y *gaps*. Con una muestra amplia de rutas y paradas, se observa una diferencia promedio cercana a +5 min entre el headway real y el programado.

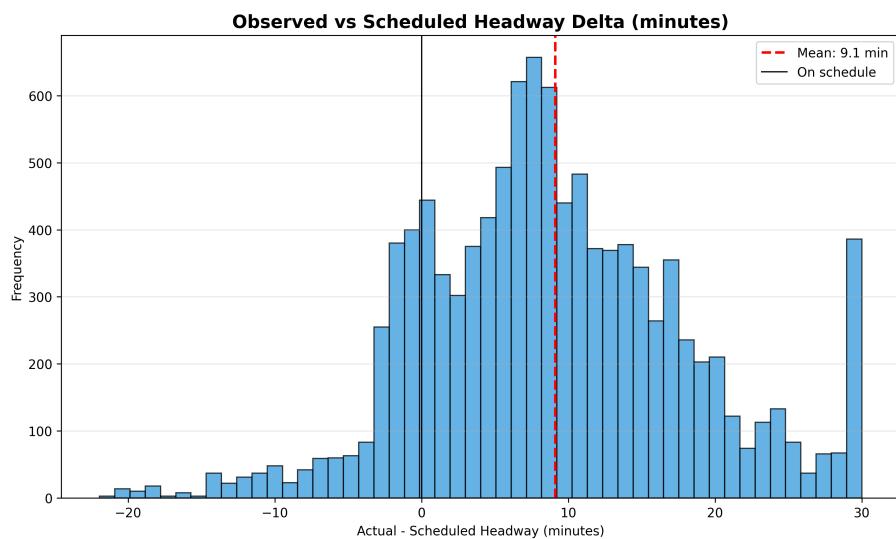


Figura 25: Distribución de la diferencia de headway (observado – programado).

La distribución muestra un corrimiento positivo: en la mayoría de paradas el intervalo real es varios minutos mayor que el planificado. Los peores casos aparecen en la ruta 210 (centro), con demoras de headway que rondan los 20 min sobre lo programado. En el extremo opuesto, algunas paradas de las rutas 531 y 602 presentan headways algo más cortos que los programados, reflejando sobreoferta puntual.

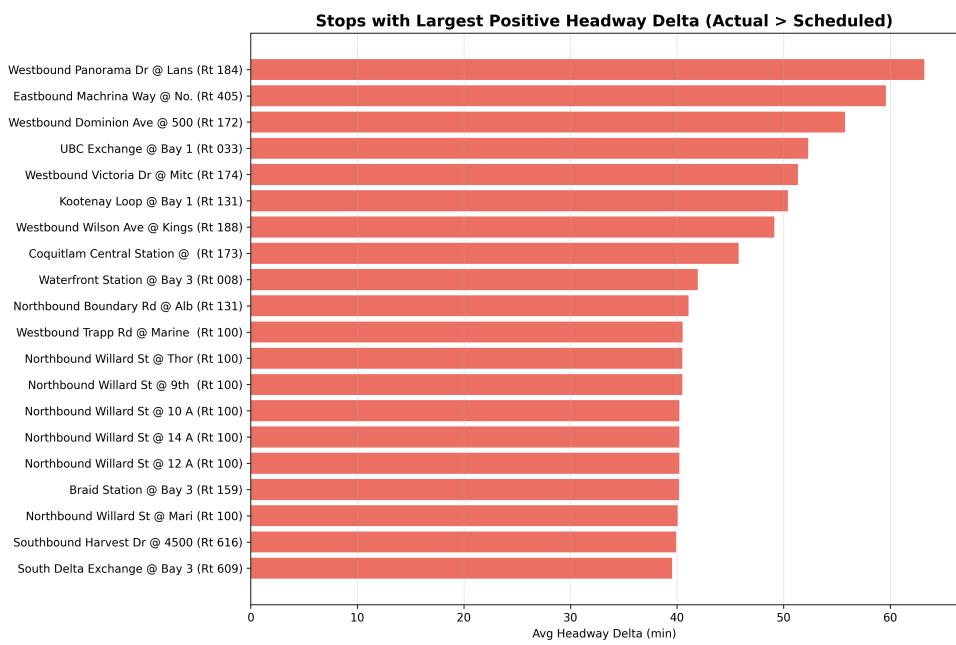


Figura 26: Paradas con mayor headway observado vs. programado.

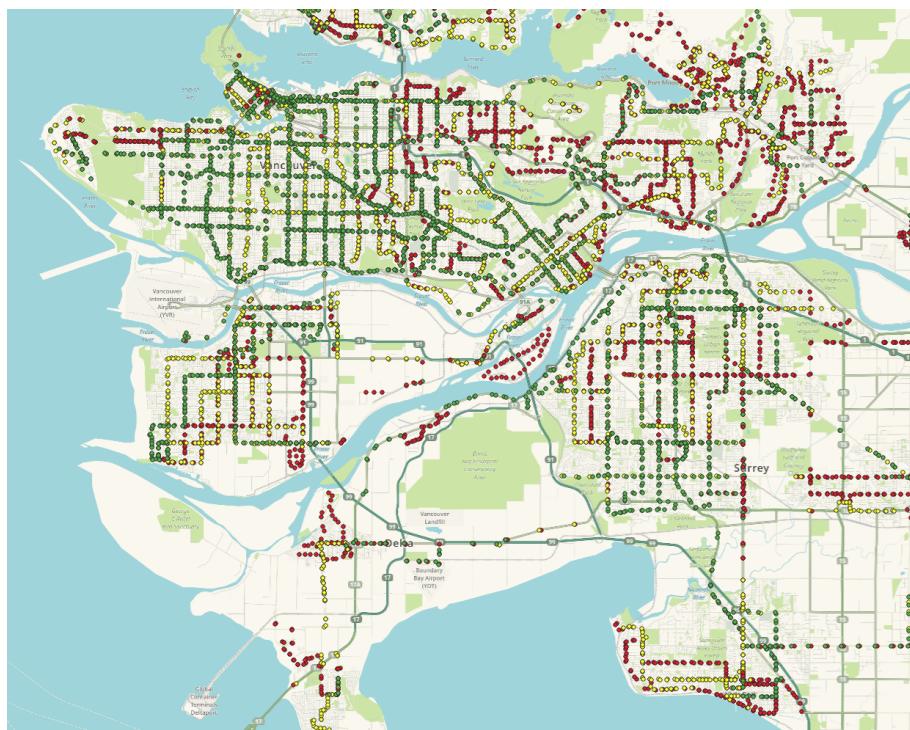


Figura 27: Mapa de diferencia de headway observado vs. programado por parada.

5. Discusión de resultados

En los análisis estáticos se observa una red con alta concentración de rutas y segmentos en el centro de Vancouver, donde también se registran mayores niveles de densidad poblacional y de oferta de transporte por segmento. Los mapas de velocidad programada permiten diferenciar corredores rápidos de tramos particularmente lentos, mientras que la superposición con la densidad poblacional muestra que algunas áreas densas en zonas periféricas presentan menor longitud de recorridos cercanos por unidad de superficie. Al analizar estos datos, podemos observar que las rutas de bus tienden a estar distribuidas y planeadas de una manera muy coherente, ofreciendo mayores rutas y frecuencia en zonas con mayor densidad poblacional.

En torno a los estadios, los mapas y gráficos de proximidad indican que los recintos céntricos disponen de mayor cantidad de paradas cercanas y de más viajes diarios, y que existen rutas específicas que conectan estos puntos de interés con las áreas de mayor densidad poblacional. Los estadios ubicados en zonas residenciales o periféricas se asocian en general con menos población conectada y con una red de segmentos de tránsito menos intensa. En este caso, podemos ver como el Pacific Coliseum, el cual se encuentra en una zona residencial, tiene menos paradas cercanas y menos viajes diarios que los estadios más céntricos.

Los resultados con datos en tiempo real muestran que, aunque muchos segmentos mantienen velocidades cercanas a las programadas, existen corredores donde la velocidad observada es sistemáticamente menor o mayor que la prevista. El horario y día elegido para el análisis es una buena muestra representativa del servicio funcionando no solamente en su momento mas congestionado (hora pico un dia de semana), sino tambien como funciona el servicio la mayor parte del tiempo. Los mapas y distribuciones de demoras por parada evidencian una cola hacia valores positivos, con retrasos concentrados en ciertos corredores y horarios. Sin embargo, además de que estas diferencias inevitablemente variarán dependiendo la hora del día, podemos observar que aún con muchos tramos no cumpliendo con la velocidad programada, el horario se suele cumplir con precisión.

Finalmente, el análisis de headways revela una mezcla de intervalos “bunched”, regulares y con grandes brechas, cuya distribución varía según la ruta. Estos datos presentan la información de que la mayoría de los buses tienden a circular de forma muy regular, con alrededor de 10 minutos de media, pero con más del 30 % de los buses circulando con menos de 3 minutos de intervalo, lo cual es un buen indicador de que el servicio es eficiente.

En conjunto, los resultados estáticos y en tiempo real ofrecen una imagen integrada de la estructura de la red, su relación con la densidad poblacional y el desempeño operativo observado durante el periodo analizado.