Laboratory 14

Urban Mobility and Smart Cities: Simulation of Bus Scheduling

Module: Advanced Topics and Case Studies

Duration: 135 Minutes

Contents

Objective

In this lab, you will use a Python simulator to model one bus route (**R12**) over a full day. You will:

- run the simulator with a **Static Schedule** (fixed 20 min headway),
- run it with a **Smart Headway Policy** (10 min in peaks, 20 min off-peak),
- compare performance using key indicators (waiting times, passengers served),
- run a small **parameter sweep**,
- and perform a **stress test & equity check** at stop level.

The focus is on following the practical steps correctly and interpreting the outputs.

1 Simulation Scenario & Files

Route R12 Scenario

- **Route:** Single bus line R12 from CBD to suburbs.
- **Time horizon:** 24 hours (0–1440 minutes).
- **Stops:** 20 stops, indexed from 0 to 19.
- **Passengers:** Arrive randomly at each stop. Each passenger boards the first bus that arrives after their arrival time, if capacity is available.
- **Comparison:**
  - **Static schedule:** buses every 20 minutes all day.
  - **Smart schedule:** buses every 10 minutes during peak hours (08:00–18:00), every 20 minutes off-peak.

Notebook & Main Variables

You will work in the notebook:

- `Urban_Mobilities_and_Smart_Cities_Lab_Session.ipynb`

Important configuration variables at the top of the notebook:

```
simulation_time        = 1440  # 24 hours
initial_bus_interval   = 20    # min, baseline headway
bus_capacity           = 80
stop_positions         = np.arange(0, 20, 1)
passenger_arrival_rate = 5     # min between passenger arrivals

np.random.seed(42)            # do NOT change for the main activities
```

The simulator generates a table with (among others) the following columns:

- `stop` – stop index (0–19),
- `arrival_time` – when the passenger arrived at the stop,
- `bus_time` – departure time of the bus the passenger boarded,
- `wait_time` – waiting time in minutes,
- `type` – `"Static"` or `"Smart"`.

At the end of the notebook, you will see:

- `combined_df` – all passengers, both modes,
- `summary_df` – key performance indicators by mode,
- `hourly_avg_reset` – hourly average waiting times.

Reflection & Insight

Whenever you change parameters for an experiment, change only *one* at a time and keep the random seed at 42 so that results are comparable.

2 Setup & Initial Run

Task 1: Open the Notebook and Check Parameters

Goal: Ensure everyone starts from the same baseline configuration.
Steps:

1. Open the notebook `Urban_Mobilities_and_Smart_Cities_Lab_Session.ipynb` in Google Colab or Jupyter.
2. Find the first code cell with the configuration (look for `simulation_time`, `initial_bus_interval`, etc.).
3. Confirm the values are:
   - `simulation_time = 1440`
   - `initial_bus_interval = 20`
   - `bus_capacity = 80`
   - `stop_positions = np.arange(0, 20, 1)`
   - `passenger_arrival_rate = 5`
   - `np.random.seed(42)`
4. If any of these differ, change them now.

Checkpoint: You should be able to explain in one sentence what each of these parameters controls.

Task 2: Run All Cells Once

Goal: Make sure the full simulation (Static + Smart) runs without errors.
Steps:

1. In the notebook menu, choose **Runtime** → **Restart and run all** (or the equivalent in your environment).
2. Wait until all cells are executed. This may take a short time depending on your machine.
3. Scroll to the bottom of the notebook and confirm that you see:
   - printed summary statistics (a table labelled something like `summary_df`),
   - printed hourly averages,
   - several plots (boxplot, hourly line plot, heatmap),
   - CSV files created: `r12_results.csv`, `r12_summary.csv`, `r12_hourly_avg.csv`.

Reflection & Insight

If you see errors, do *not* hack random lines. First *restart* the kernel, check the parameter cell, then run all again.

3 Activity 1 – Static vs Smart Baseline

In this activity, you use the default parameters to understand the overall behaviour of both schedules and extract basic KPIs.

Task 3: Read and Record the Baseline KPIs

Goal: Extract the key performance indicators for both Static and Smart schedules.
Steps:

1. Locate the printed table `summary_df`. It should have one row for `Static` and one for `Smart`.
2. In your lab notes (or below), write down for each mode:
   - average waiting time,
   - maximum waiting time,
   - number of passengers served.

Template:

Mode	Avg Wait (min)	Max Wait (min)	Passengers Served
Static	*(copy from summary_df)*	*(copy)*	*(copy)*
Smart	*(copy from summary_df)*	*(copy)*	*(copy)*

Small calculation:

- Compute (Smart Avg Wait) minus (Static Avg Wait).
- Compute (Smart Passengers) minus (Static Passengers).

Reflection & Insight

If Smart has a lower average wait but only serves a few more passengers, it may simply be smoothing peaks. If it serves many more passengers and reduces waits, the policy is clearly beneficial.

Task 4: Inspect the Main Plots

Goal: Visually compare Static and Smart performance.
Steps:

1. Find the **boxplot** comparing `Static` vs `Smart` waiting times.
   - Note which mode has the lower median.
   - Note which mode has the longer upper whisker (worst waits).
2. Find the **hourly average waiting time** plot.
   - Identify the hours where waiting times are highest.
   - Compare the lines for Static and Smart during peak hours (morning and evening).
3. Take a screenshot or save these figures (you will need them for your homework report).

Questions to answer in one or two sentences:

- At what times of day does the Smart policy clearly reduce waiting time?
- Is there any time where Static performs similarly or better?

4 Activity 2 – Parameter Sweep (Experiment A)

Now you explore how sensitive the system is to *one* parameter. You will change only one value, re-run the notebook, and compare KPIs.

Task 5: Choose and Change One Parameter

Goal: Perform a controlled experiment with a single change.

Step 0 – Choose ONE parameter to change:

- **Headway:** Change `initial_bus_interval` (e.g. from 20 to 18 or 22).
- **Capacity:** Change `bus_capacity` (e.g. from 80 to 60 or 100).
- **Demand:** Change `passenger_arrival_rate` (e.g. from 5 to 3 minutes between arrivals).

Steps:

1. In the configuration cell, change only the chosen parameter value. Example (headway tuning):

```
initial_bus_interval = 18   # instead of 20
```

2. Leave all other parameters, especially `np.random.seed(42)`, unchanged.
3. Use **Restart and run all**. Wait until all cells finish.
4. Again, find the `summary_df` table and record the new KPI values for both Static and Smart.

Optional: multiple scenarios

If time allows, repeat the above for 2–3 different values of the same parameter (e.g. intervals 16, 18, 20, 22). Record the values in a table like:

Scenario	Param value	Mode	Avg Wait	Passengers
A1	20	Static	...	...
		Smart	...	...
A2	18	Static	...	...
		Smart	...	...

Reflection & Insight

Make sure you always know *which* run produced which numbers. Label your scenarios clearly (A1, A2, ...).

5 Activity 3 – Stress Test & Equity Check (Experiment B)

Here you push the system into a more extreme situation and check whether the Smart policy is still fair, especially for the worst-off stops.

Task 6: Define a Stress Scenario

Goal: Simulate a "tough" condition for the network.
Choose ONE stress scenario (example options):

- **Higher demand:**

```
passenger_arrival_rate = 3   # more passengers per hour
```

- **Lower capacity:**

```
bus_capacity = 50            # smaller bus
```

- **Shifted peak window:** In the code that constructs the smart headway times, change the condition. For example, from:

```
480 <= t <= 1080   # 08:00--18:00
```

to:

```
540 <= t <= 1110   # 09:00--18:30
```

Steps:

1. Apply exactly one stress change in the configuration or smart-schedule code.
2. Restart the kernel and run all cells again.
3. Ensure the notebook finishes successfully and produces new plots and summary tables.

Task 7: Stop-level Equity Analysis for SMART

Goal: Identify which stops suffer the longest waits under the Smart policy.
Steps:

1. Create a data frame with only Smart passengers. In a new cell, run:

```
smart_df = combined_df[combined_df["type"] == "Smart"]
```

2. Compute, for each stop:
   - mean waiting time,
   - 95th percentile waiting time,
   - number of passengers.

   Example code (adapt column names if needed):

```
def p95(x):
    return x.quantile(0.95)

stop_stats = smart_df.groupby("stop")["wait_time"].agg(
    mean_wait="mean",
    p95_wait=p95,
    n_passengers="size"
)

print(stop_stats)
```

3. From `stop_stats`, note:
   - the stop with the highest `mean_wait`,
   - the stop with the highest `p95_wait`.
4. Look at the **wait-time heatmap** (Stop vs Hour) produced by the notebook.
   - Identify which stops and hours are darkest (worst waits).

Write down (2–4 sentences):

- Which stops are the worst-off under Smart in the baseline case?
- How does this change in your stress scenario?
- Does Smart improve the situation for the worst stops, or mainly improve the average?

Reflection & Insight

A schedule can look "good" on average but still be unfair to a few stops. Your job here is to catch that.

6 Homework – Tune & Test Route R12

For homework, you will design *one* small improvement or variation and test it using the same simulator. This will become a short report.

Task 8: Choose a Track and Run Your Scenario

Goal: Run one clearly defined redesign of the system.

Pick ONE track:

- **Track A – Headway tuning:** Keep the Smart logic, but change `initial_bus_interval` (for example from 20 to 18).
- **Track B – Capacity sensitivity:** Keep the schedule the same, but change `bus_capacity` (for example 60 or 100).
- **Track C – Peak window shift:** Keep the intervals the same, but shift the Smart peak window (e.g. from 08:00–18:00 to 09:00–18:30).

Steps:

1. Restart the notebook kernel.
2. Apply your chosen change (only *one*).
3. Run all cells so that both Static and Smart are simulated for your new scenario.
4. Save:
    - the new `summary_df` (screenshot or values),
    - the hourly average wait plot (Static vs Smart),
    - the boxplot comparing Static vs Smart.
5. Compare your redesigned scenario to the original baseline (Task 3 & 4 results).

What your written report should at least include:

- A short description of your change (which parameter, old vs new value).
- A table comparing the main KPIs (Static vs Smart, baseline vs your scenario).
- One or two key plots (hourly averages, boxplot).
- A short discussion of the main changes and the trade-off you observe (better waits vs more buses / capacity / etc.).

Reflection & Insight

Be explicit: "I changed X from A to B. As a result, average waiting time changed by Y minutes and passengers served changed by Z. This is good/bad because . . . "