U.PORTO FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

U.PORTO FC FACULDADE DE CIÊNCIAS UNIVERSIDADE DO PORTO

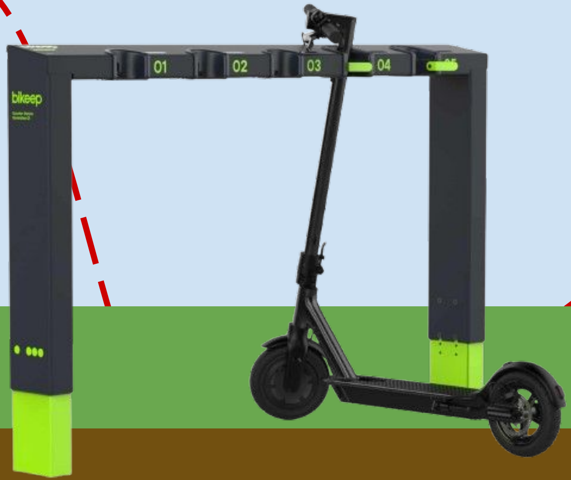
Master's in Artificial Intelligence (M.IA)
Modelling and Simulation (M.IA029)

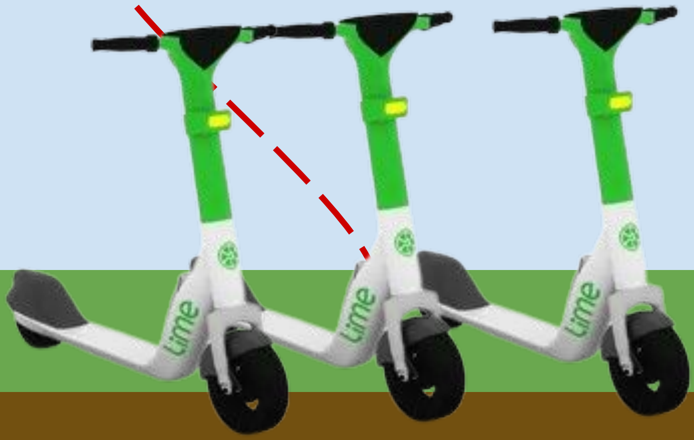
Management of Shared Mobility

Problem Formalization | Methods & Tools

Group IA1_20

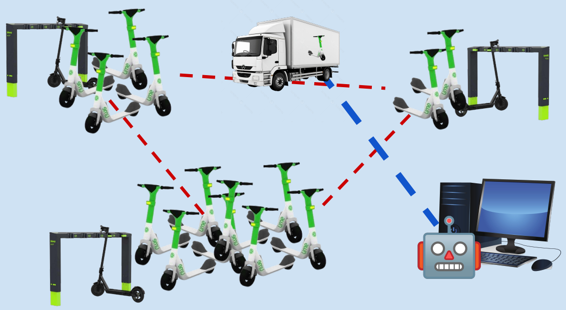
Eduardo Passos
Guilherme Silva
Valentina Cadime





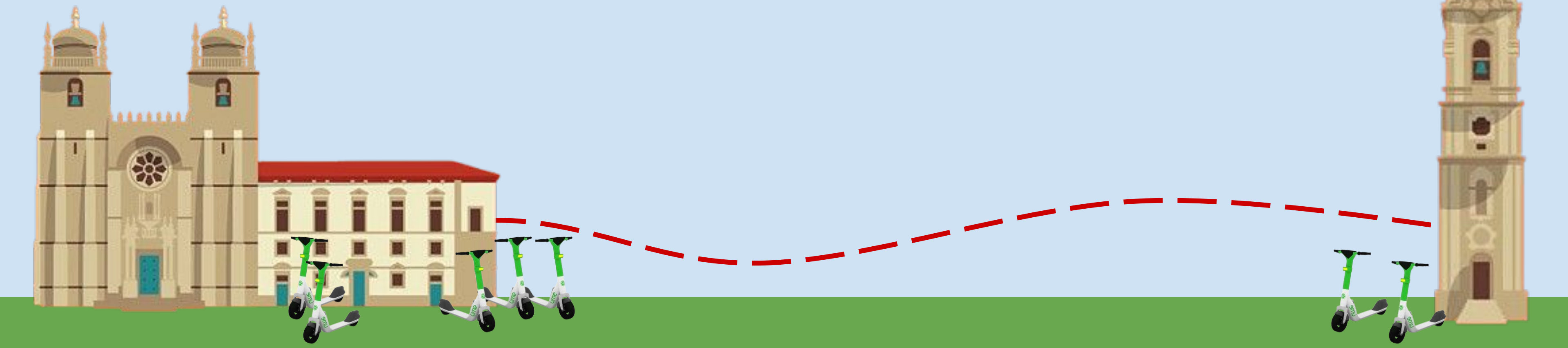
Problem Addressed

- Project focus: daily management of shared electric scooter fleets (Topic 7)
- Motivation: shared electrical mobility is becoming common in cities. Management is important.
- Aim: minimize operating costs while maximizing service, in quantity and quality
- Shared scooters/bikes suffer from spatial and temporal imbalance
- Demand varies by time of day, location, weather and events
- Stations may become empty or saturated, reducing availability
- Vehicles have battery constraints and charging delays
- Operators must balance service quality vs operational cost



Project Objectives

- Build a realistic simulation of an urban micromobility system
- Model demand, travel, battery, charging and relocation
- Incorporate stochastic exogenous factors (weather, events)
- Evaluate baseline, heuristic, and reinforcement policies learning under multiple scenarios
- Analyze trade-offs between availability, queues and cost



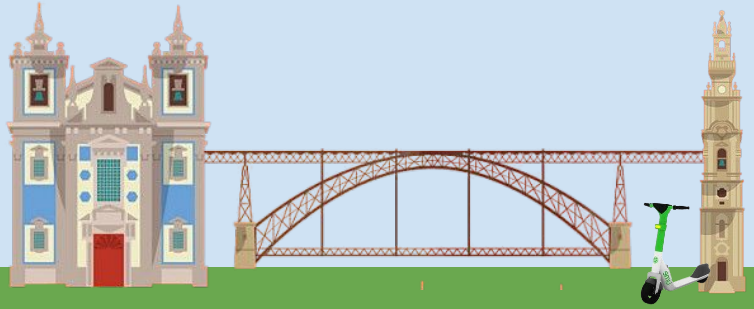
Methodology

Space Definition:

- **10 well-spaced station locations** from Porto's bike network (using OpenStreetMap API)
- **Pairwise distances** and **travel times** computed using shortest bike paths
- Defined station parameters: **capacity**, **demand**, **energy consumption**

Modelling:

- **Discrete-time simulation:** 5-min ticks (steps)
- **Configuration and State**
- **Agent-Environment interface:** RL-style API
  - reset(), step()
  - observations
  - actions ($act \in \mathbb{R}^4$)
- **Scenarios and Events:**
  - **Weather:** Discrete-time Markov chain
  - **Demand Peaks:** Morning and evening rush hour
  - **Events:** Random temporary boosts in demand (concerts, sports games, etc.)



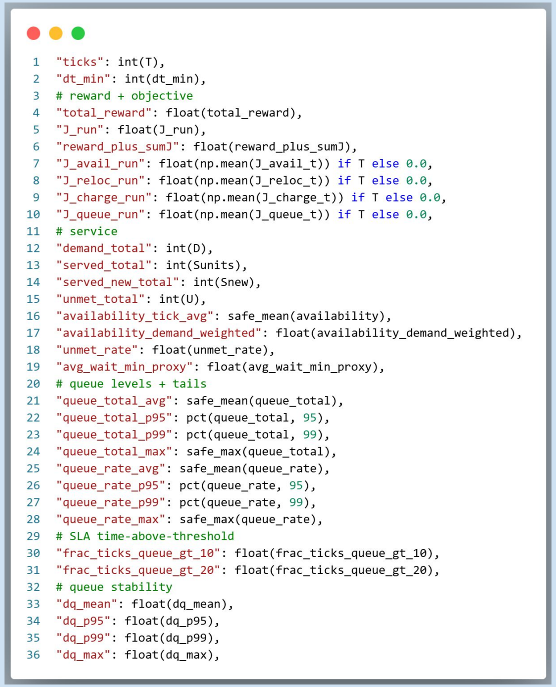
Methodology

Planners: Stored in a registry for easy swapping

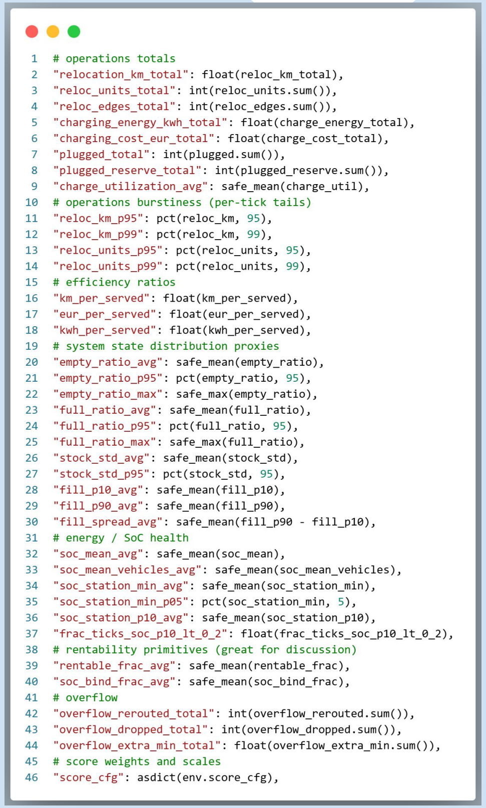
- **Relocation**
  - **Greedy:** From overfilled stations to near-empty ones
  - **Budgeted:** Greedy but with a maximum limit per tick
- **Charging**
  - **Greedy:** Prioritize charging at low-battery stations
  - **Slack:** Charge at stations with low expected demand

Extra: **No-op planner** for comparison

Metrics and KPIs:



```python
"ticks": int(T),
"dt_min": int(dt_min),
# reward + objective
"total_reward": float(total_reward),
"J_run": float(J_run),
"reward_plus_sumJ": float(reward_plus_sumJ),
"J_avail_run": float(np.mean(J_avail_t)) if T else 0.0,
"J_reloc_run": float(np.mean(J_reloc_t)) if T else 0.0,
"J_charge_run": float(np.mean(J_charge_t)) if T else 0.0,
"J_queue_run": float(np.mean(J_queue_t)) if T else 0.0,
# service
"demand_total": int(D),
"served_total": int(Sunits),
"served_new_total": int(Snew),
"unmet_total": int(U),
"availability_tick_avg": safe_mean(availability),
"availability_demand_weighted": float(availability_demand_weighted),
"unmet_rate": float(unmet_rate),
"avg_wait_min_proxy": float(avg_wait_min_proxy),
# queue levels + tails
"queue_total_avg": safe_mean(queue_total),
"queue_total_p95": pct(queue_total, 95),
"queue_total_p99": pct(queue_total, 99),
"queue_total_max": safe_max(queue_total),
"queue_rate_avg": safe_mean(queue_rate),
"queue_rate_p95": pct(queue_rate, 95),
"queue_rate_p99": pct(queue_rate, 99),
"queue_rate_max": safe_max(queue_rate),
# SLA time-above-threshold
"frac_ticks_queue_gt_10": float(frac_ticks_queue_gt_10),
"frac_ticks_queue_gt_20": float(frac_ticks_queue_gt_20),
# queue stability
"dq_mean": float(dq_mean),
"dq_p95": float(dq_p95),
"dq_p99": float(dq_p99),
"dq_max": float(dq_max),
```



```python
# operations totals
"relocation_km_total": float(reloc_km_total),
"reloc_units_total": int(reloc_units.sum()),
"reloc_edges_total": int(reloc_edges.sum()),
"charging_energy_kwh_total": float(charge_energy_total),
"charging_cost_eur_total": float(charge_cost_total),
"plugged_total": int(plugged.sum()),
"plugged_reserve_total": int(plugged_reserve.sum()),
"charge_utilization_avg": safe_mean(charge_util),
# operations burstiness (per-tick tails)
"reloc_km_p95": pct(reloc_km, 95),
"reloc_km_p99": pct(reloc_km, 99),
"reloc_units_p95": pct(reloc_units, 95),
"reloc_units_p99": pct(reloc_units, 99),
# efficiency ratios
"km_per_served": float(km_per_served),
"eur_per_served": float(eur_per_served),
"kwh_per_served": float(kwh_per_served),
# system state distribution proxies
"empty_ratio_avg": safe_mean(empty_ratio),
"empty_ratio_p95": pct(empty_ratio, 95),
"empty_ratio_max": safe_max(empty_ratio),
"full_ratio_avg": safe_mean(full_ratio),
"full_ratio_p95": pct(full_ratio, 95),
"full_ratio_max": safe_max(full_ratio),
"stock_std_avg": safe_mean(stock_std),
"stock_std_p95": pct(stock_std, 95),
"fill_p10_avg": safe_mean(fill_p10),
"fill_p90_avg": safe_mean(fill_p90),
"fill_spread_avg": safe_mean(fill_p90 - fill_p10),
# energy / SoC health
"soc_mean_avg": safe_mean(soc_mean),
"soc_mean_vehicles_avg": safe_mean(soc_mean_vehicles),
"soc_station_min_avg": safe_mean(soc_station_min),
"soc_station_min_p05": pct(soc_station_min, 5),
"soc_station_p10_avg": safe_mean(soc_station_p10),
"frac_ticks_soc_p10_lt_0_2": float(frac_ticks_soc_p10_lt_0_2),
# rentability primitives (great for discussion)
"rentable_frac_avg": safe_mean(rentable_frac),
"soc_bind_frac_avg": safe_mean(soc_bind_frac),
# overflow
"overflow_rerouted_total": int(overflow_rerouted.sum()),
"overflow_dropped_total": int(overflow_dropped.sum()),
"overflow_extra_min_total": float(overflow_extra_min.sum()),
# score weights and scales
"score_cfg": asdict(env.score_cfg),
```

Simulation Demonstration

Experimental Work: Algorithms

Heuristic Agent: Manually crafted rules

- Throttles actions based on previous performance **(reloc. and charging EWMA)**
- Adapts to time-of-day, waiting queues, station occupation

Episodic Multi-Armed Bandit: UCB1

- Search of different combinations (arms) of **relocation and charging budgets**
- **One combination** per simulation episode
- After many episodes, the arm with the **best average reward** is learnt (for that specific constant action)

Contextual Bandit: LinUCB

- Re-selects budget combinations **every "block"** (e.g. hour), instead of every episode
- **Uses the state of the system** and a **linear model** that predicts the expected reward to pick an arm

Soft-Actor Critic (SAC): RL

- Learns a policy (**Deep Neural Network**) that maps **continuous** observations to actions
- Finds highest cumulative reward
- Improves modelling of **uncertainty in demand** (events, weather) through **entropy maximization**

Experimental Work - Setup

Chosen KPIs:

- **Availability:** measures service quality
- **Queue Size:** captures user waiting and congestion
- **Relocation Distance:** reflects operational effort
- **Charging cost:** accounts for energy expenditure

Normalized per-tick objective:

$$J_t = w_A \cdot \frac{U_t}{A_0} + w_R \cdot \frac{R_t}{R_0} + w_C \cdot \frac{C_t}{C_0} + w_Q \cdot \frac{Q_t}{Q_0}$$

Calibration procedure:

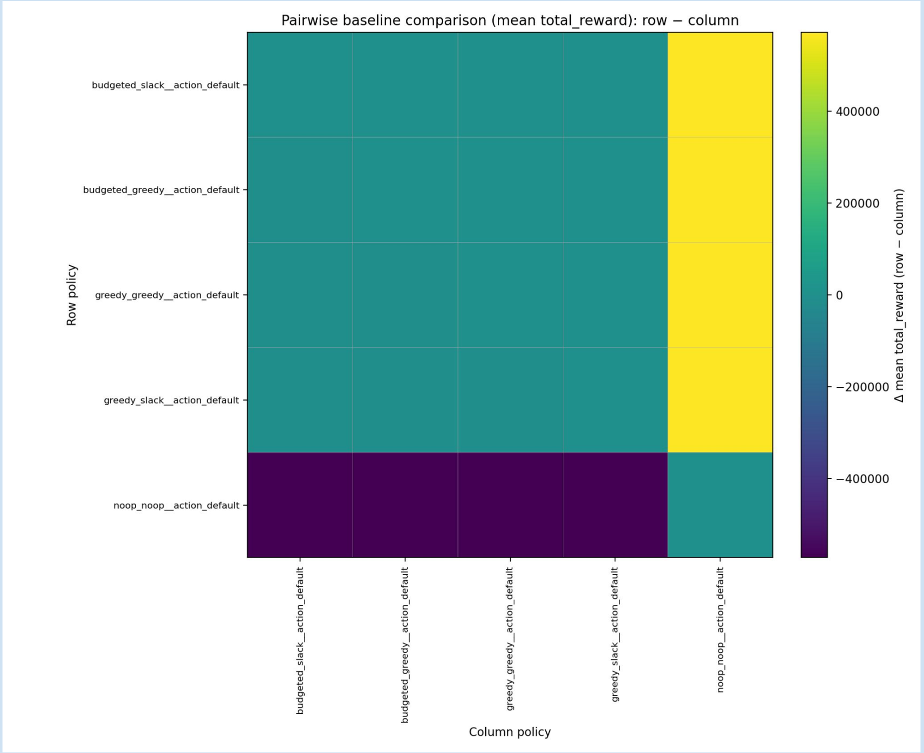
- Each raw signal has different units and scales
- A calibration script was used to
  - Run reference policies under nominal conditions
  - Estimate typical magnitudes of each signal
  - Set normalization constants so that all terms contribute comparably

Weight design rationale

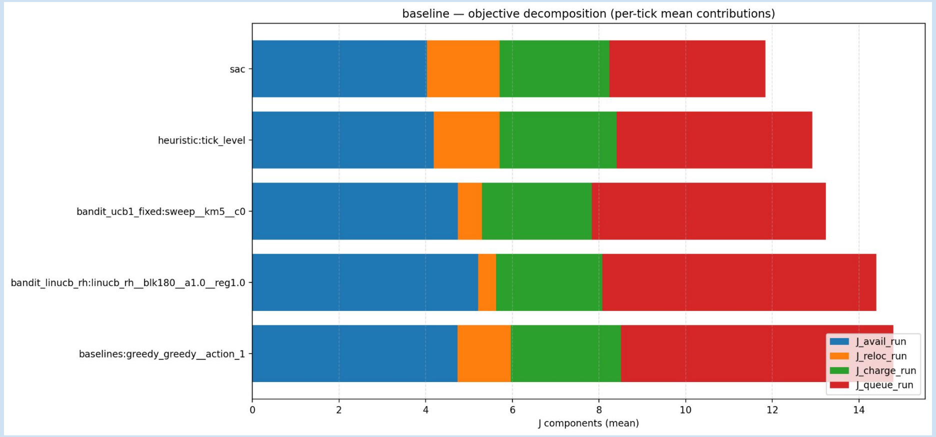
- wA = 5
- wQ = 4
- wR = 1
- wC = 1

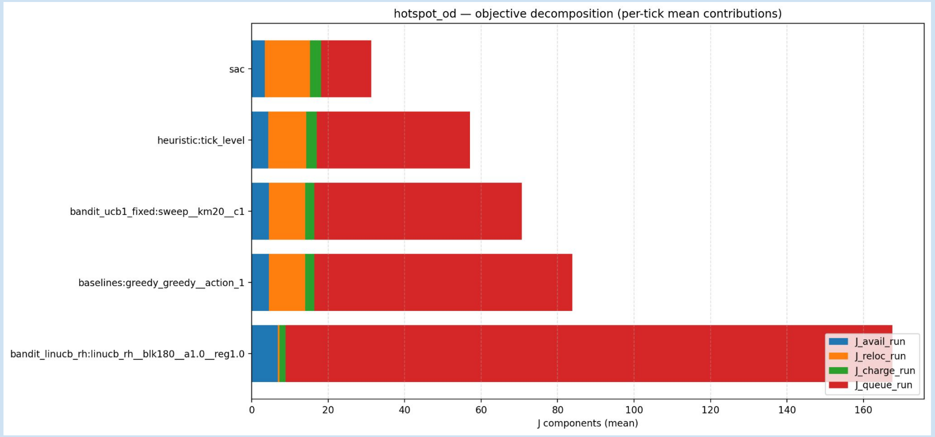
Baselines

- **Baseline set:** no-op, greedy, budgeted, slack
- **Evaluation:** same seeds, same action
- **Metric:** mean total reward (pairwise)
- **Observation:** all active baselines ≫ no-op
- **Observation:** Greedy, budgeted, and slack are near-equivalent in reward



Analysis of Results





Critical Analysis and Directions for Future Work

Critical Analysis

- Performance differences are dominated by queue dynamics, not operational costs
- Queue penalties dominate J_run in all scenarios
- Heuristics fail under demand heterogeneity
- Bandits reduce cost but sacrifice service quality

Future Work

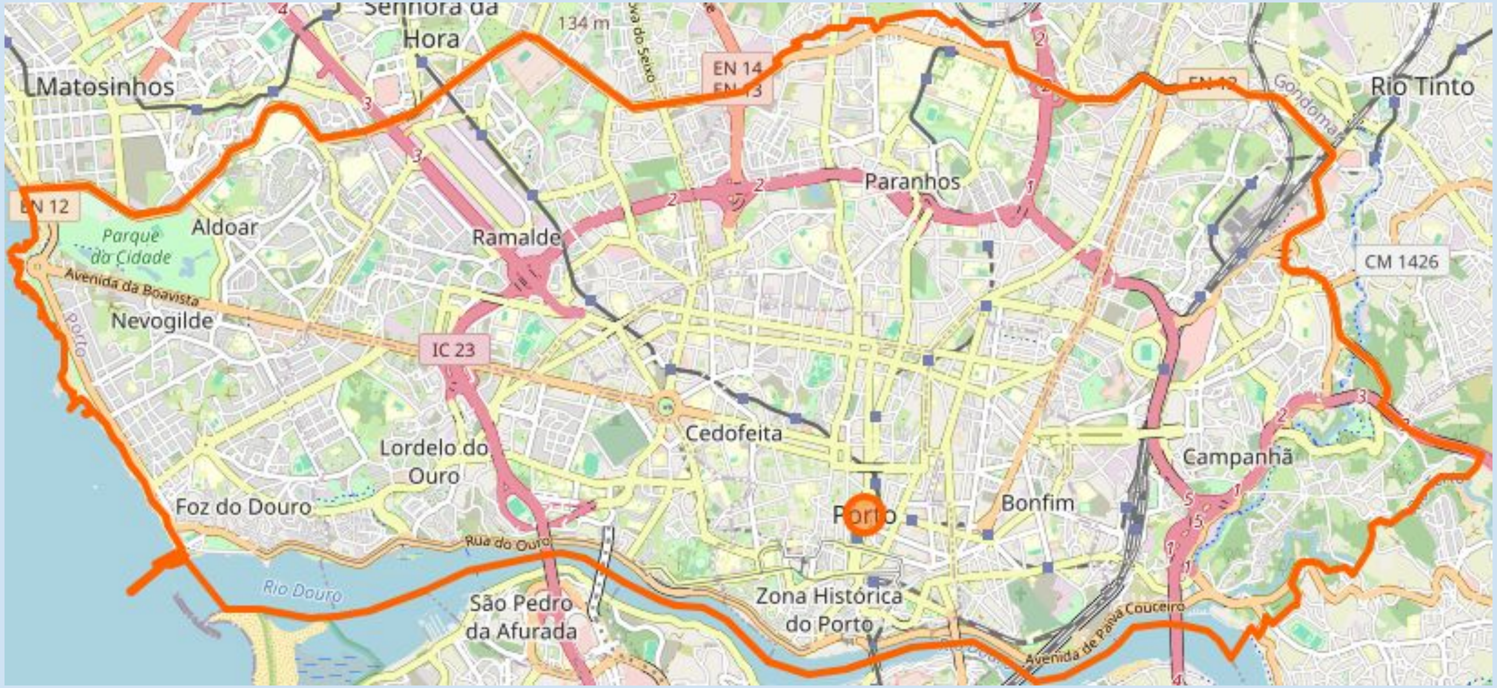
- Queue-aware learning objectives
- Multi-timescale decision making:
  - Fast loop for queue mitigation
  - Slow loop for fleet balancing
- Different frequencies so that agents can react to sudden spikes in demand, while not affecting the management in the long run (can be affected by noisy rewards from per tick action).
- Evaluate agents under different maps:
  - Maps with location nodes farther away from each other
  - Maps with more points overall

Thank you!

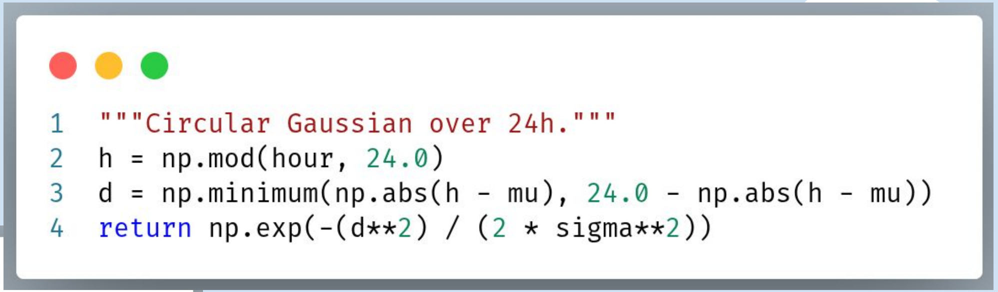
Any questions?

Backup slides

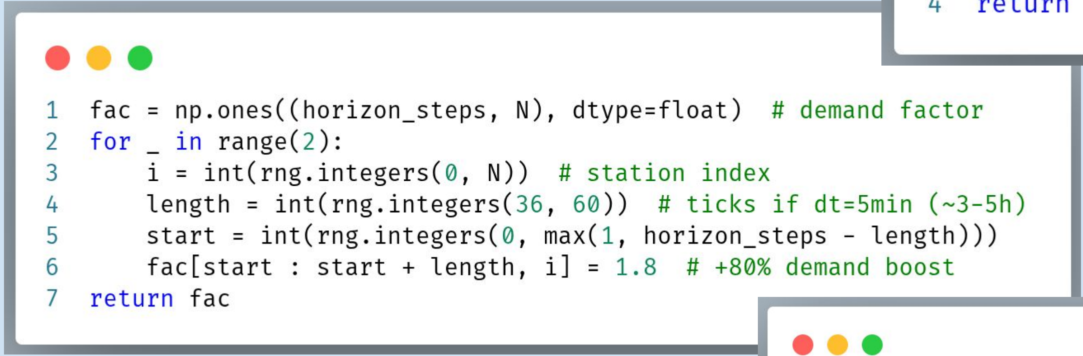
Physical Space Generation



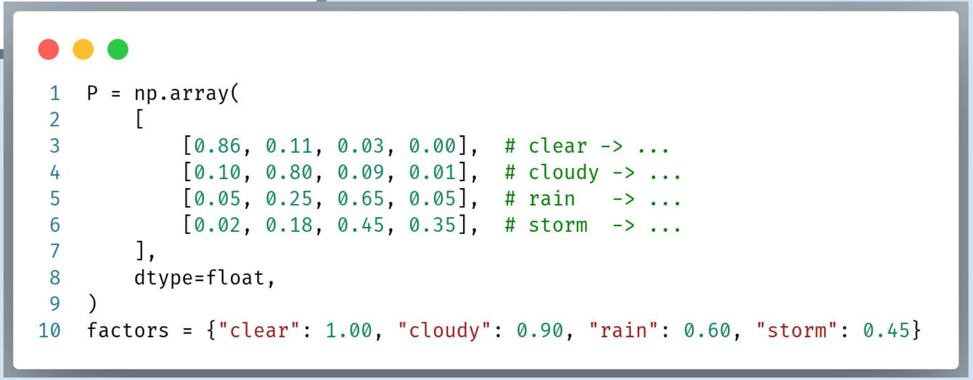
Modelling: Demand



```python
"""Circular Gaussian over 24h."""
h = np.mod(hour, 24.0)
d = np.minimum(np.abs(h - mu), 24.0 - np.abs(h - mu))
return np.exp(-(d**2) / (2 * sigma**2))
```

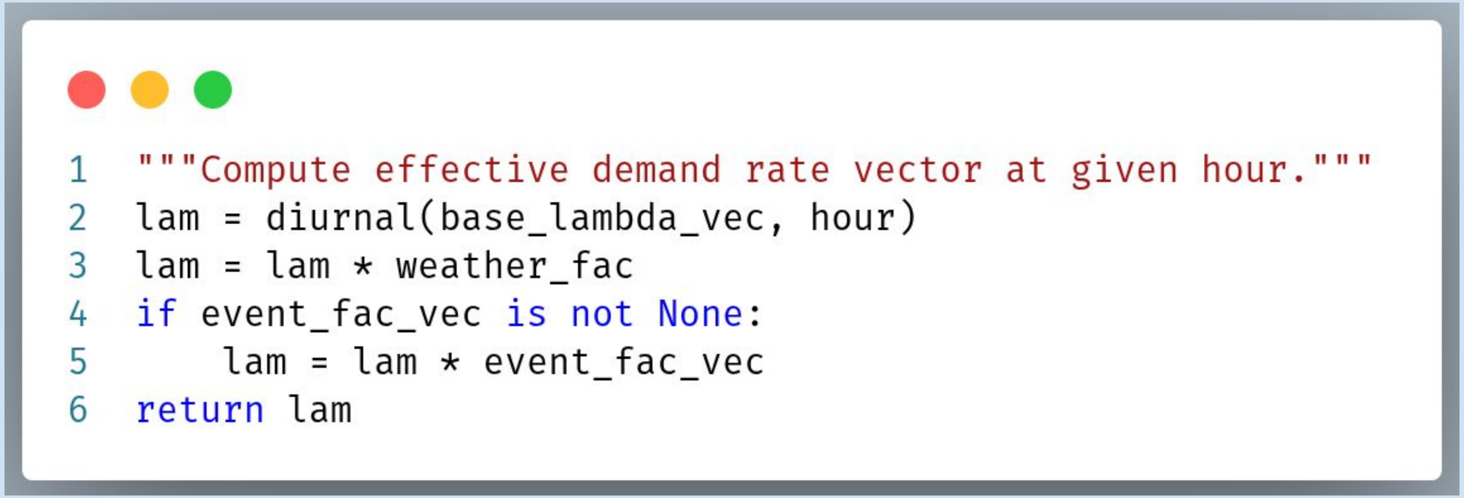


```python
fac = np.ones((horizon_steps, N), dtype=float)  # demand factor
for _ in range(2):
    i = int(rng.integers(0, N))  # station index
    length = int(rng.integers(36, 60))  # ticks if dt=5min (~3-5h)
    start = int(rng.integers(0, max(1, horizon_steps - length)))
    fac[start : start + length, i] = 1.8  # +80% demand boost
return fac
```



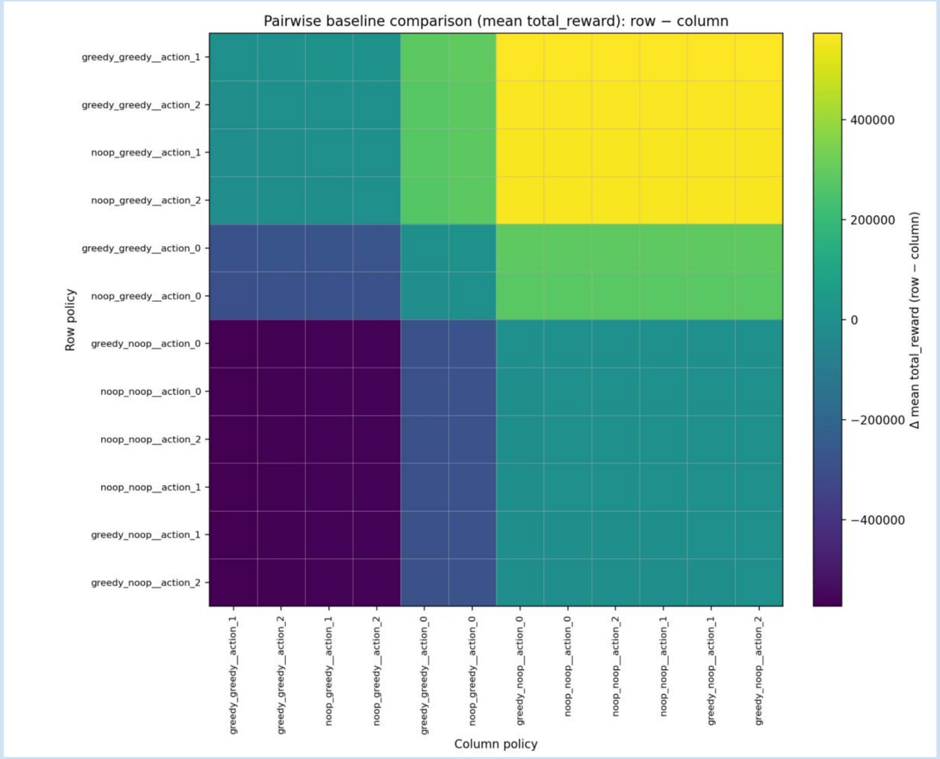
```python
P = np.array(
    [
        [0.86, 0.11, 0.03, 0.00],  # clear -> ...
        [0.10, 0.80, 0.09, 0.01],  # cloudy -> ...
        [0.05, 0.25, 0.65, 0.05],  # rain   -> ...
        [0.02, 0.18, 0.45, 0.35],  # storm  -> ...
    ],
    dtype=float,
)
factors = {"clear": 1.00, "cloudy": 0.90, "rain": 0.60, "storm": 0.45}
```

Modelling: Full Demand

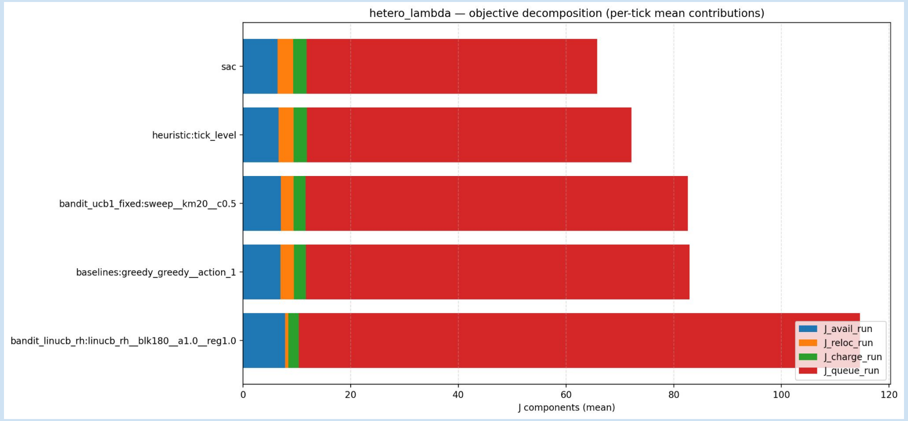


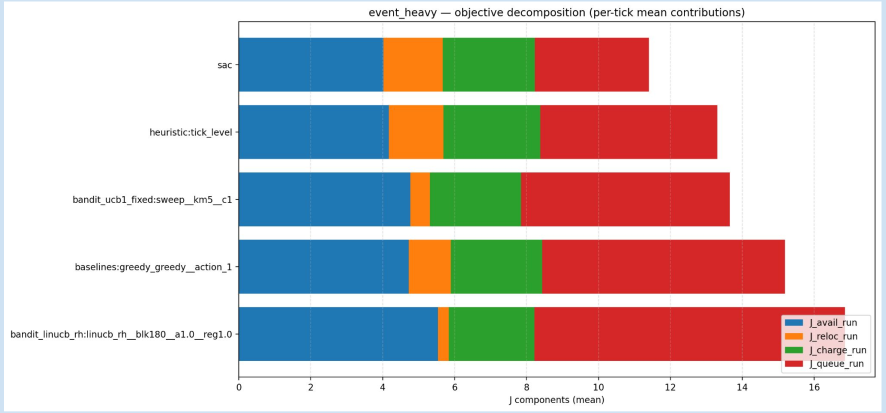
```python
"""Compute effective demand rate vector at given hour."""
lam = diurnal(base_lambda_vec, hour)
lam = lam * weather_fac
if event_fac_vec is not None:
    lam = lam * event_fac_vec
return lam
```

Planner Algorithms Comparison



More Scenarios


hetero_lambda — objective decomposition (per-tick mean contributions)
sac
heuristic:tick_level
bandit_ucb1_fixed:sweep__km20__c0.5
baselines:greedy_greedy__action_1
bandit_linucb_rh:linucb_rh__blk180__a1.0__reg1.0
J_avail_run
J_reloc_run
J_charge_run
J_queue_run
J components (mean)


event_heavy — objective decomposition (per-tick mean contributions)
sac
heuristic:tick_level
bandit_ucb1_fixed:sweep__km5__c1
baselines:greedy_greedy__action_1
bandit_linucb_rh:linucb_rh__blk180__a1.0__reg1.0
J_avail_run
J_reloc_run
J_charge_run
J_queue_run
J components (mean)