My Role Report

Simulation & Optimization for AI-Powered Train Traffic Control

Author: Saksham Khandelwal
IIT Bombay – Smart India Hackathon 2025
Date: September 08, 2025

Problem Understanding and Design Goals

Railway section control is a constrained scheduling problem with strong safety requirements: one train per block section, minimum headways governed by blocking-time theory, platform capacities and lengths, mandatory dwell times, and train priorities. Real-time disturbances (primary delays, temporary speed restrictions, platform outages) propagate secondary delays.

1. Scope of My Work

I am responsible for Simulation and Optimization.

• Simulation Lead – Build and maintain a microscopic, event-driven SimPy engine with strict safety constraints and KPI computation.

• Optimization Developer – Implement a two-tier decision engine: fast Greedy rolling dispatcher and a short-horizon CP-SAT optimizer. This would be my second work that I can work upon along with someone assisting him.

• Integration Engineer – Expose APIs and push live data to the React frontend

2. Tools, Languages, Frameworks

Primary languages: Python (backend, simulation, optimization), JSON (data interchange).

Core libraries:

• Simulation: SimPy, NumPy, Pandas, MatPlotLib, NetworkX.

• Optimization: OR-Tools CP-SAT (intervals/no-overlap/precedence).

• Simulation Interface: React, FastAPI, Pydantic models, WebSockets.

3. Simulation Module

I will implement a discrete-event simulation using SimPy. Each train is a process that requests track resources (block sections) and station resources (platforms). The simulation enforces no-overlap of trains on the same block and respects minimum headways derived from blocking-time components: approach time, occupancy time, and clear time. Station dwell times and platform capacities are applied; platforms also respect length constraints so long trains cannot be assigned to short platforms. Junctions are handled as shared resources with controlled entry to avoid conflicting movements.

Event types include: planned departure/arrival, actual arrival/clear, block entry/exit, dwell start/end, conflict detection, and disruption injection. For each event, the simulator updates the train state (location, delay, queue) and logs to an event stream from which KPIs are computed. The simulator is deterministic under a fixed random seed to support reproducible experiments and demo runs, further we can connect it to real Indian Railways API that upon taking that data simulate the real-world situation.

Key responsibilities in the Simulation Module:

• Infrastructure loader: parse CSV/JSON for nodes, blocks, platforms, speeds, gradients, and junction topology.

• Train/timetable loader: parse and validate train attributes (type, priority, length, performance) and planned schedules.

• Simulating the optimization: call the Optimization Engine and apply its decision (hold/release/order).

• Scenario runner: load predefined scenarios (Normal, Rush, Incident), apply disruptions, and export results.

NOTE: Frontend Integration-

• React frontend for the simulation part in the final website

• API Calls –

- GET /state → current trains, positions, resource usage, KPIs snapshot.
- POST /simulate → run/restart simulation with selected scenario; returns run id.
- POST /optimize → trigger optimization at next tick; returns decisions and new schedule deltas. This calls the optimization engine
- Later on, these would be synced to the realtime Railways API

4. Optimization Module

The optimization component operates in a rolling horizon. At each decision epoch, it extracts the current state from the simulator and computes an action sequence that minimizes weighted total delay while preserving feasibility. The design balances response time and quality via a two-tier approach: an instantaneous Greedy dispatcher and a short-horizon constraint-programming solve, with an optional local metaheuristic in between.

Tier-1 Greedy: each ready-to-move train is scored using a weighted function combining (a) train priority, (b) current lateness, (c) predicted downstream conflict pressure, and (d) local platform pressure. The best feasible action is selected; feasibility is verified by a lightweight checker shared with the simulation. This tier responds in well under 100 ms and guarantees a decision even when the solver is time-boxed.

Tier-1.5 Metaheuristic (optional): a Variable Neighborhood Search (or Tabu Search) explores small changes around the Greedy plan—e.g., swapping precedence of a conflict pair or adjusting dwell by small amounts—to reduce downstream knock-on delays. This runs for a bounded iteration budget and returns the best improvement found.

Tier-2 CP-SAT: a constraint model over a 20–30 minute window. Each (train, resource) usage is an interval variable with start and end times; resources enforce 'no overlap' across intervals; precedence binaries resolve conflicts; minimum dwell and platform capacities are encoded as constraints. The objective minimizes weighted total delay and proxy terms for secondary delay. The solve is strictly time-limited (≈2–5 s) and only the first 5–10 minutes of the solution are committed, after which the window rolls forward.

I wouldn't work on the optimization alone as this would be the second task for me assisting someone that is dedicatedly working on this subsystem.

5. Coupling Simulation and Optimization

Simulation pauses at conflicts, sends snapshot to Optimizer. Optimizer returns decision, Simulator applies. Fairness ensured by adding wait penalties. Stability via commitment horizon. This loop repeats until scenario end.

6. Work Plan

Setup & Basics (Week 1)
Goal: Set up environment and model the railway network. Simulate train movements (without optimization).

- Use **SimPy** for event-based simulation:
  - Train requests a track section (resource).
  - If busy → wait; if free → proceed.
- Model travel time = track length / train speed.

Optimization (Week 2-3)
Goal: Introduce scheduling with Integer Linear Programming (ILP).

- Define ILP problem with **OR-Tools (CP-SAT Solver)**:
  - Variables: start times / train order on track sections.
  - Constraints:
    - Safety headway (minimum gap between trains).
    - Platform capacities.
    - Priority rules (express > passenger > freight).
  - Objective: minimize total delay / maximize throughput.
- Test with small case (e.g., 2 stations, 3 trains).

Integrate Optimization + Simulation (Week 4)
Goal: Use optimizer to guide simulation.

- Run optimizer → generate schedule (precedence + timings).
- Feed schedule into **SimPy** simulation.
- Introduce disruptions (e.g., train delayed by 5 minutes) and re-run optimizer in real time.

Phase 6: Integration with UI & APIs (Alongside as work goes)
Goal: Deliver hackathon-ready system.

- Expose optimizer as an API using **FastAPI along with** integrating API with **React** dashboard.