DISCRETE EVENT SIMULATION OF THE INDIAN RAILWAY NETWORK

CS4900 : UGRC Report

April 21, 2023

Aditya Viraj Rao
Faculty Mentor : Prof. Narayanaswamy N S
Indian Institute of Technology Madras

Abstract
The report presents a simulation of the Indian railway network using Python programming language and the SimPy simulation framework. It runs the simulation purely automatic signaling protocols without any data about the signal color at each time interval.

Contents

1 Introduction

The Indian railway network is one of the world's largest and most complex railway systems, with over 7,000 stations, more than 115,000 km of track, and over 23 million passengers transported daily.
Such a vast network's efficient and reliable operation requires sophisticated planning, scheduling, management tools, and a deep understanding of the network's dynamics and behaviour. Simulation modelling is one approach that can help to gain insights into the behaviour of complex systems such as railway networks.
The simulation is based on real-world data and incorporates stochastic elements to capture the inherent variability of railway opera-

tions. The simulation provides a valuable tool for modelling and analysing the Indian railway network and exploring the impact of different operational and construction decisions on network performance.

2 Literature Review and Related Works

2.1 Railway mixed-traffic simulation software: COST

This publication by a group from IIT Bombay aims to create a discrete event simulator with inputs such as track geometry like descent. The train features like maximum acceleration, train time tables and the signalling data for each signal at each time interval.

Given a Traction-Effort vs Speed chart for a particular locomotive, in which the acceleration usually decreases with increasing speed, this particular simulator requires to have a constant value of the acceleration such that the maximum speed is achieved in the same distance as the variable acceleration curve: we call this the equivalent constant acceleration.

As can be seen, almost all railway technology options can be modelled, and the impact on throughput/capacity/congestion can be assessed on critical sections of railways. The scenarios of importance need to be identified and modelled through the data entry mechanisms and then compared systematically using this tool by IITB. Each such effort requires careful data validation and experimentation, and providing direct interface options for all such possibilities is unrealistic. The tool can impose the given traffic on a given infrastructure and tabulate performance statistics which the user can use to gauge the efficiency or performance of their protocol.

The detailed logic that allows for different signalling regimes to be modelled follows from this. Every train moves at a velocity to respect the occupancy information provided by movements of higher-priority trains already scheduled. They emphasize that this is a capacity planning tool, not an online control tool, so the simulated scenarios assume that all train movements are known in advance. The tool computes the section occupancies, throughput, and performance measures accordingly. The logic has a lot in common with control in real-time but needs significantly different interfaces and utilities to be able to use for that purpose. The complete tool is built using JAVA, and also the simulation is run like a discrete event simulation but without using SimJava. The research group has created a particular discrete event simulator for this software.
COST (Cost Optimization and Simulation Tool) A detailed comparison can be made on the timetable generated before and after crucial scenarios, like adding a loop line or adding/removing temporary speed restrictions (TSR).

2.2 Analysing various delays using macroscopic simulation

This paper aims to use simulation to investigate the relationship between primary delays and the resulting punctuality. Furthermore, the potential for macroscopic simulation to cover larger networks and/or several cases is also highlighted by utilising a fast, macroscopic railway traffic simulation tool for the study. As a case study, we use the region of Skåne in Southern Sweden, using the actual timetable and operational data for 2019 and a draft timetable for 2025.

A simulation is a standard tool in railway research for evaluating the punctuality and robustness of timetables and disturbance scenarios. A wide range of examples of applications can be found. But these microscopic simulators model using the exact track layout, signal and station positions. These are also very computationally expensive. Thus this paper uses **macroscopic simulation tool** called **PROTON** to study the various delays

Train delays can be classified as primary or secondary. Primary delays occur when a train door fails to close, while secondary delays occur when other trains are delayed. It is important to consider the proportions of primary and secondary delays to reduce the occurrence of primary delays.

2.3 Dwell time based on traffic

Big cities such as Paris have experienced a steady growth in demand for passenger transportation, with 70% of railway trips made inside the suburban area. To keep providing a good quality of service, operating companies seek to design timetables that perform well even in saturated conditions. Knowledge of the mechanisms determining train dwell times is important, as they dimension the network's capacity and are partly responsible for operations stability. Additionally, a structural source of instability lies in alighting and boarding passengers. Accurate estimations of train dwell times are also needed during the operational phase.

This paper proposes a novel approach for assessing the dependency of dwell time on passenger flows in public transport. It proposes a definition and method for computing the minimum dwell time for a given passenger flow, which can be used to estimate network capacity, design countdown systems, and classify data according to the main determinanImplementing station of dwell time mothe del into the simulation tool yields higher accuracy, classification of stations and integration of data about neglected phenomena could lead to more accurate predictions.

2.4 RailML

RailML (Railway Markup Language) is an XML-based data exchange format designed for the interoperability of railway applications and systems. It provides a standardized way to exchange data between railway systems, applications, and organizations, including infrastructure managers, railway operators, and rolling stock manufacturers.

RailML includes a variety of data models and schemas that cover different aspects of railway operations, including infrastructure, timetables, rolling stock, train operations, signalling, and power supply. These data models continuously evolve to incorporate new standards, technologies, and requirements from the railway industry.

This had a significant impact on how we approached our problem statement. This project has a couple of main entities for each other measurements for attributes.

We similarly had the idea of implementing the various entities in a railway network as classes in Python with attributes as the class variables. We created methods in the class to change the required class variable to the desired value in a logically correct way.

This tool has a detailed description of the elements in a train, especially engine specifications, which are needed for the acceleration profile. But since we aren't considering the acceleration profiles of the train, we decided to take inspiration from this to have our Objects and methods for the objects.

3 Simpy Features Used and Miscellaneous

This section contains the features used in simply and miscellaneous aspects of it

3.1 Simpy Features Used

- **SimPy environment**
  This SimPy feature allows us to parse a common object of type **Environment,** which all objects can mutually access. This helps run the events parallelly.

- **env.timeout**
  This feature helps us in proceeding with the simulation. In our case, each train object executes its run function, creating a time clock for each train. We use the timeout feature to increase the clock by **t seconds.**

- **env.now**
  This very elegant API call gives us the current time elapsed so far on each clock of the train

- **.run (until = ..)**
  Using this feature, we give input to SimPy about the last second until each individual clock should run. If trains are still running past this time, the log file of their further activities won't be maintained.

3.2 Miscellaneous

Parallelization:- We can find parallel components of our simulation and use **multiple threads, shared resources, and mutual exclusion** to run the code faster. But some level of parallelization is already done by SimPy. It is using **8 threads** irrespective of the simulation.

4 Data Scraping and Organising

We scrapped train names and timetables from the web. We used scrappy spider, a web scraping tool, to speed up the scraping process. Then the scraped data was stored in a database. For now, we initially considered the trains running in the South Central part of the railway system to simulate a smaller subset of the data.

We read the database and organized the information into a dictionary of $train\_name, time\_table$.

We also scraped information using Selenium automation and beautiful soup to get station code to station name information from here. Made a dictionary with the key as the station code and the name as the value.

From the timetables data, we traversed each timetable and constructed a Python dictionary to store the neighbors of a given class when given a Station code. From this Dictionary, we could identify the junctions stations as they will have more than 2 stations as neighbors.

5 Why SimPy?

SimPy, a python Library

5.1 NS3

We initially started with modeling the system of trains as a network topology.

With equivalents being:-

- packet - train
- router - station
- channel/medium - tracks
- network protocol - signaling protocol

This was very interesting as a large amount of research was done on network protocols that can be equivalently modified for the train systems. Also, there will be multiple layers in the network simulator which can be assumed as different stages of the train coming, servicing, and leaving the platform based on the availability of the tracks.
But NS3 had its challenges. They are:-

- We are unable to track a packet between 2 routers, but we need a segment-wise position at each instant to identify bottlenecks or congestions.
- The number of packets vs the number of routers is vast compared to the number of trains vs the number of stations.
- Due to this, the network protocols designed to handle packet drops and heavy packet inflow will be complicated to get equivalents for in the railway network.
- Also, implementing the railway network in NS3 required heavy network concepts, and implementing railway components as classes would have been difficult.

So after considering these issues of equivalency and complexity, we have preferred SimPy.

5.2 Ease of using

- Flexible modeling: SimPy provides a flexible modeling framework that allows you to create models that accurately represent the real-world behavior of a railway network. You can model complex interactions between trains, tracks, stations, and other elements of the network.
- Event-driven simulation: SimPy uses an event-driven simulation approach, which means that the simulation progresses in discrete time steps, with events triggering actions in the simulation. This approach allows you to simulate complex interactions between different elements of the railway network, and to model the impact of delays and other factors on the system.
- Concurrency support: SimPy provides support for concurrency, which is important for simulating a railway network where multiple trains may be operating on the same tracks simultaneously.
- SimPy is written in Python, a popular programming language with a large ecosystem of libraries and tools. This makes it easy to integrate SimPy with other Python-based tools and to customize and extend your simulation model as needed. Since the scraped information was all in a database, we used SQLite3, a Python library, to read from the DB and put the information into a dictionary of $train\_name, time\_table$.

Finally, because of the ease of using Python, we have decided to go with Python to run our Simulation.

6 Assumptions

This section contains the assumptions in this code.

- Trains come to cruising speed and stop in negligible time. Thus we aren't considering the acceleration profile of the train.
- Train is smaller than 1km and runs every day of the week (as we are missing the day of the week date from scraping).
- Trains require no acceleration or deceleration to overtake other trains.
- Each segment is of fixed 1km length and, for the initial scenario, has capacity = 1.
- Trains that start before our input start time are neglected for this experiment.
- There are some very close stations, i.e., closer than 1 km. In such cases, we neglect the travel time to the second station.

7 Inputs for the code

- The starting time of the simulation
- How long should the simulation run for
- The initial state space which is a list of $(train\_id, segment\_id)$ at the start time.
- What should be the station capacity
- We are assuming that **every station is a special object of the segment**
- If no cruising speed is given then the code runs using the average speed between pair stations as the cruising speed

8 Methodology

The Basic Components Used and Assumptions taken for the code

8.1 Basic Components

We have divided all the entities involved into classes with class variables and class methods. For simplicity, in the start, we took four main classes:-

- **Segment**
  contains variables like:-
  capacity
  name - The procedure used for naming this is if a segment is the 5th segment on track going from station S1 to S2. The name is **S1_S2_5.**
  The SimPy environment (env) is also passed to each segment when being initialized Has methods:-
  Request (train_id) - returns true if this particular segment has space to accommodate a new train. Else, false.
  Release (train_id) - to deallocate the capacity allotted to this train in the particular segment

- **Station**
  contains variables like:-
  capacity
  name - station code which can use the get_station to get station object
  segment* (coming up in assumptions)

  The SimPy environment (env) is also

passed to each segment when initialized.
Has methods:-
Run(t in seconds) - which prints the list of trains present in this station at every minute

- **Timetable**
contains variables like:-
list L ⇒ which is a list of tuples **(station,travel_time,service_time, cummulative_distance)**
here travel_time is the time it reaches that particular station, and cumulative_distance is the distance from the previous station. So the **strt_station** - which is the station object of the starting station name
The SimPy environment (env) is also passed to each object when being initialized Has methods:-
make_schedule () - Important method to create a structure of the network.
This method uses the prev list, which the timetabled class was initialized with, to make station objects of stations and put them in a dictionary. It also makes segment objects for tracks between each pair of stations and adds them to a dictionary which is further added to a dictionary with key = **S1_S2.**

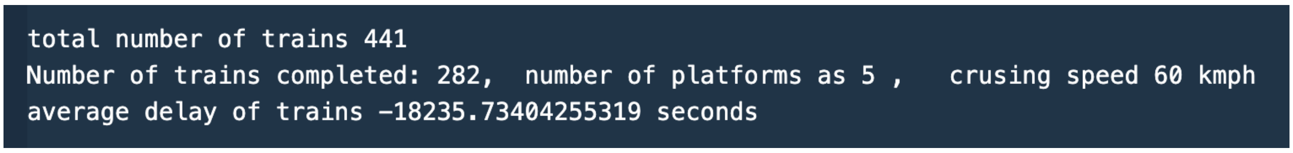
- **Train**
contains variables like:-
a timetable object
train_id or the name
seg_prsnt - which stores the current segment at which the train is present at
speed - which is a constant cruising speed* (elaborated in the assumptions)
delay - which maintains the delay of each train running and gets updated every time the train goes to a station
The SimPy environment (env) is also passed to each object when being initialized
Has method:-
run () - The main simulation happens here. This function takes the train's timetable and stars the train at strt_station from the timetable object. Checks if the next segment has sufficiently free. If yes, it enters the next segment and releases the previous segment. Else waits for 1 second and rechecks if the next segment is free. After entering each segment, the train prints the train_id, segment_id, and time of entry in the log file. It finally terminates at the last station.
At the last it prints the delay for arrival at the last station along with time of arrival.

9 Observations

9.1 Scheduled vs 60kmph

We are interested in 2 different scenarios. One where all the trains run as per schedules and the other where all trains cruise at 60kmph speed rather their speeds derived from the timetables.

```
total number of trains 441
Number of trains completed: 132 number of platforms as 5,  cruising speed as per timetable
average delay of trains 877.1515151515151 seconds
```

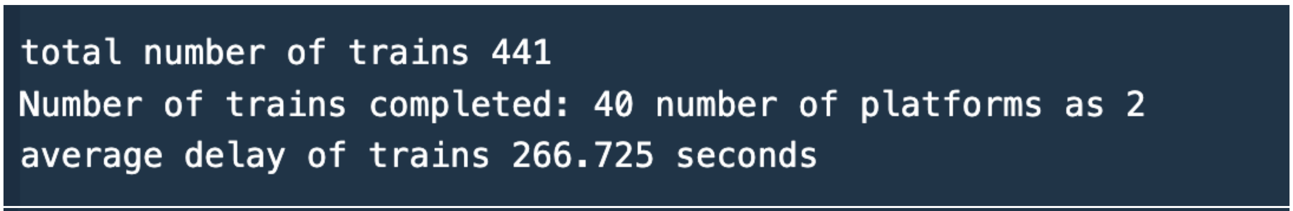


```
total number of trains 441
Number of trains completed: 282,  number of platforms as 5 ,   crusing speed 60 kmph
average delay of trains -18235.73404255319 seconds
```

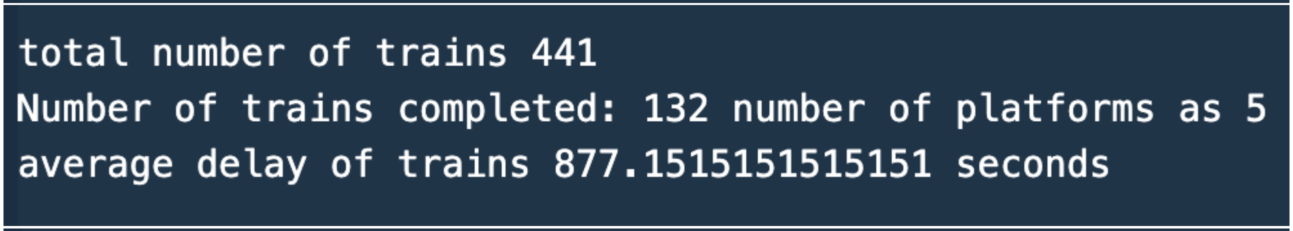
We can clearly observe here that running every train at 60kmph is better in terms of delays, but maybe constrained due to engine and track constraints.

9.2 Varying the station capacity

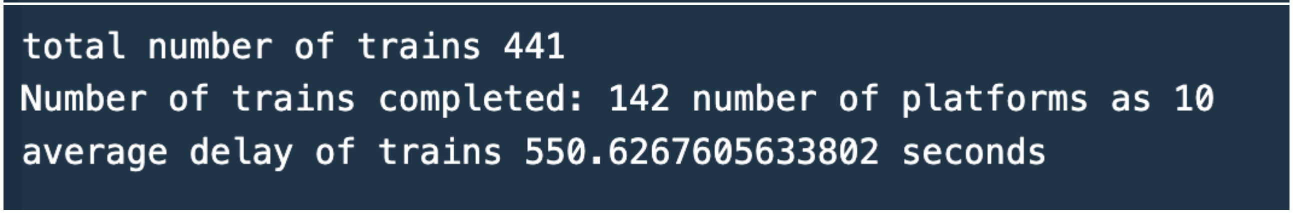
We can see the number of trains finishing before the day increases as the platforms in a station increase. This especially helps in stations like junctions and also at places where there 2 trains with extremely different speed contending for the platform. When there are 2 free platforms for the contending trains, the faster one comes, gets serviced and travels away at its fastest and not hindering any movement for the slower train.



```
total number of trains 441
Number of trains completed: 40 number of platforms as 2
average delay of trains 266.725 seconds
```



```
total number of trains 441
Number of trains completed: 132 number of platforms as 5
average delay of trains 877.1515151515151 seconds
```



```
total number of trains 441
Number of trains completed: 142 number of platforms as 10
average delay of trains 550.6267605633802 seconds
```

From this we can observe that the delays and number of trains finishing per day will be higher with higher platforms. But higher platforms come with higher constructional costs, so we should aim to strive a fine balance between delays and costs.

10 Conclusions

By constructing a network model and simulating its operation, it is possible to explore different scenarios, test different operational strategies, and identify areas for improvement. This report presents a simulation of the Indian railway network using the Python programming language and the SimPy simulation framework.

In 7 months, Railways lost 24 years due to train delays . Read more here

To address this significant concern, our project's main objective is to reduce delays in the Railway network towards which simulation of the new protocols, additions in the infrastructure or other decisions can be made on the framework developed by this research.

11 Scope of further research

Simulating an Indian railway network using SimPy can offer numerous research opportunities, particularly in transportation and logistics. The Indian railway network is one of the largest railway networks in the world, and studying its operation using simulation can help identify potential bottlenecks and inefficiencies, which can lead to improvements in the overall efficiency and safety of the system.
Here are some potential research topics that can be explored using a SimPy-based Indian railway network simulator:

- **Capacity optimization**: By simulating the Indian railway network, researchers can explore ways to optimize the system's capacity by identifying bottlenecks and exploring strategies to reduce congestion.
- **Train scheduling and routing**: Simulations can be used to evaluate different train schedules and routing strategies to minimize delays and improve overall efficiency.
- **Safety analysis**: Researchers can use the simulator to study the system's safety by analyzing potential accidents and identifying prevention measures.
- **Energy efficiency**: Researchers can use the simulator to explore ways to reduce the system's energy consumption by optimizing train speeds, reducing idle time, and exploring alternative energy sources.

12 Acknowledgements

I am grateful to my guide, Prof. N.S. Narayanaswamy, for all his time and advice and for always being accessible to assist me. A very special thanks to Prof Ayon Chakraborty for his elegant insights about using the OOPs style approach and various other advice. I would also like to express my gratitude to the Department of Computer Science and Engineering at IIT Madras for enabling undergrads like me to participate in research through the UGRC program. Very grateful to work with my research mates Hari Charan K and Yuvan S for their insights and help. Also would like to thank Aditya Sampath (from NITT) for helping us with the Scrapy Spider and other scraping tools. Thanks for your insightful comments and suggestions.

13 References

- COST paper by IITB
- RAILML
- Dwell time based on traffic
- Analysing various delays using macroscopic simulation
- SimPy documentation