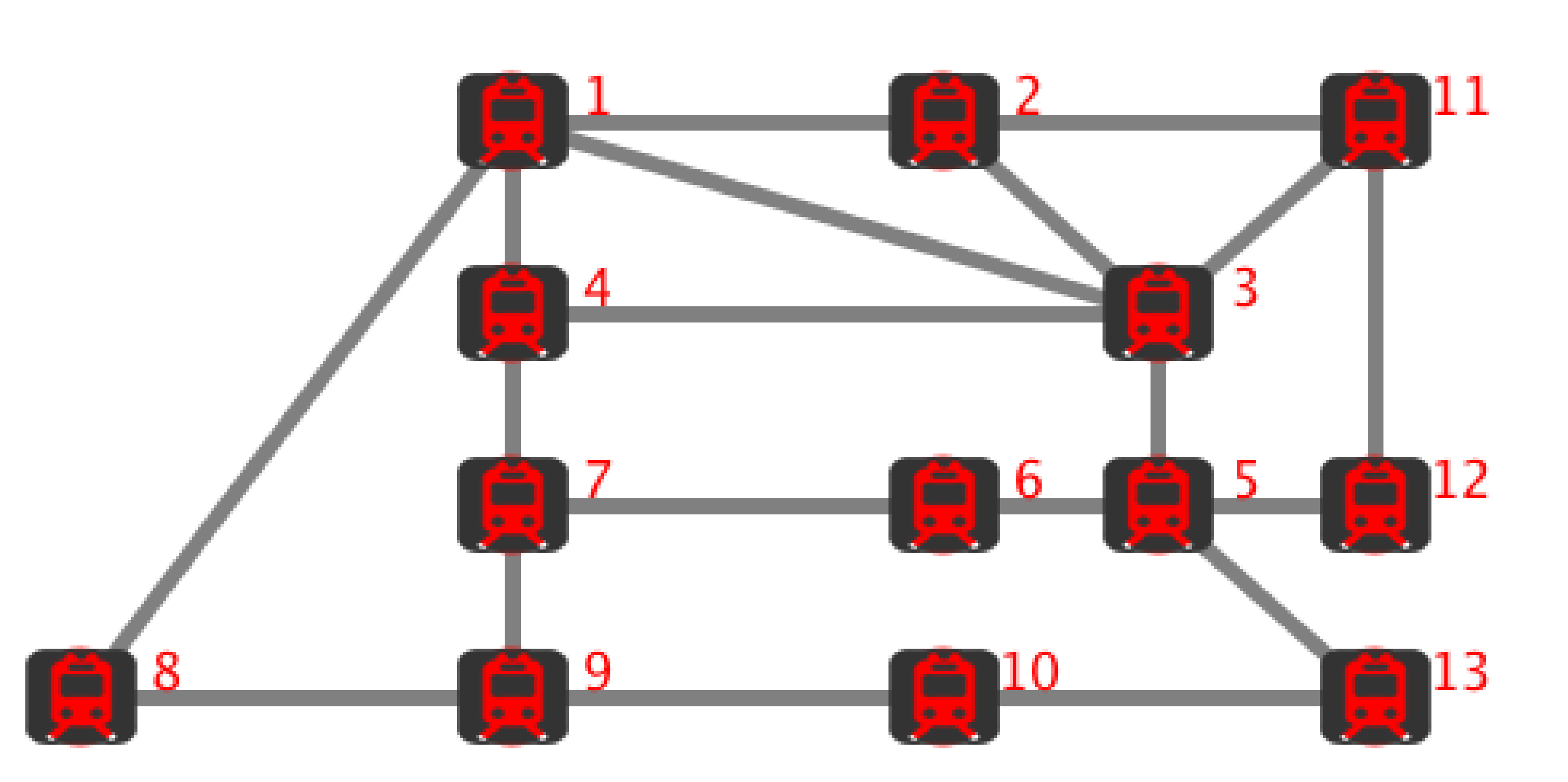
Train Dispatch

Group Members: Mike Lindenmayer, Ibrahim Oyekan, Manan Davda

Our Grid



Classes Involved

- Controller.java
- Edge.java
- Grid.java
- Main.java
- PriorityQueue.java
- Station.java
- TestCase.java
- Train.java
- Simulator.java

Grid.java

- Adjacency list that holds the collection of edges
- Function for adding edges to the grid
- Two functions for setting the lock state of an edge
- Uses Sequential search to find the edges in the Grid

Edge.java

- Data structure that has 4 values
    - Start
    - End
    - LockedState
    - Cost
- Allows the grid to hold the lock state of any track

Train.java

- Data structure that holds all the data for a specific train
    - ❖Path
    - ❖Distance Travelled
    - ❖Speed (mph)
    - ❖Status
    - ❖Last station
    - ❖Wait time
- Move train Function
    - Once the train has travelled the length of the track, it will return the edge it just completed for it to be available.

Controller.java

- It computes the Dijkstra’s algorithm for the train.
- Dijkstras algorithm finds the shortest path barring any locked tracks
- If it cannot find any path, it will return an empty path.
- It uses a priority queue of stations for computing the algorithm

PriorityQueue.java

- Data structure that holds three functions
  - Queue (Adds the station to the queue)
  - Dequeue (removes the station from the queue)
  - Sort (sorts the queue)
- Uses the stations data structure for holding cost and station numbers.

Main.java

- Handles the primary functionality of the simulator
- It uses a real time implementation in seconds
- Each second the trains perform one of three tasks
    - Assigns a new path to the train (Controller.java)
    - Move the train (train.java)
    - Remove the train from queue
- Holds priority queue of Trains based on their status
- Communicates with the simulator class to draw

TestCase.java

- Generates the text file for train sequence
- It creates our sequence of train to be used by our program
- Uses command line arguments to configure the number of trains and the time frame they spawn.

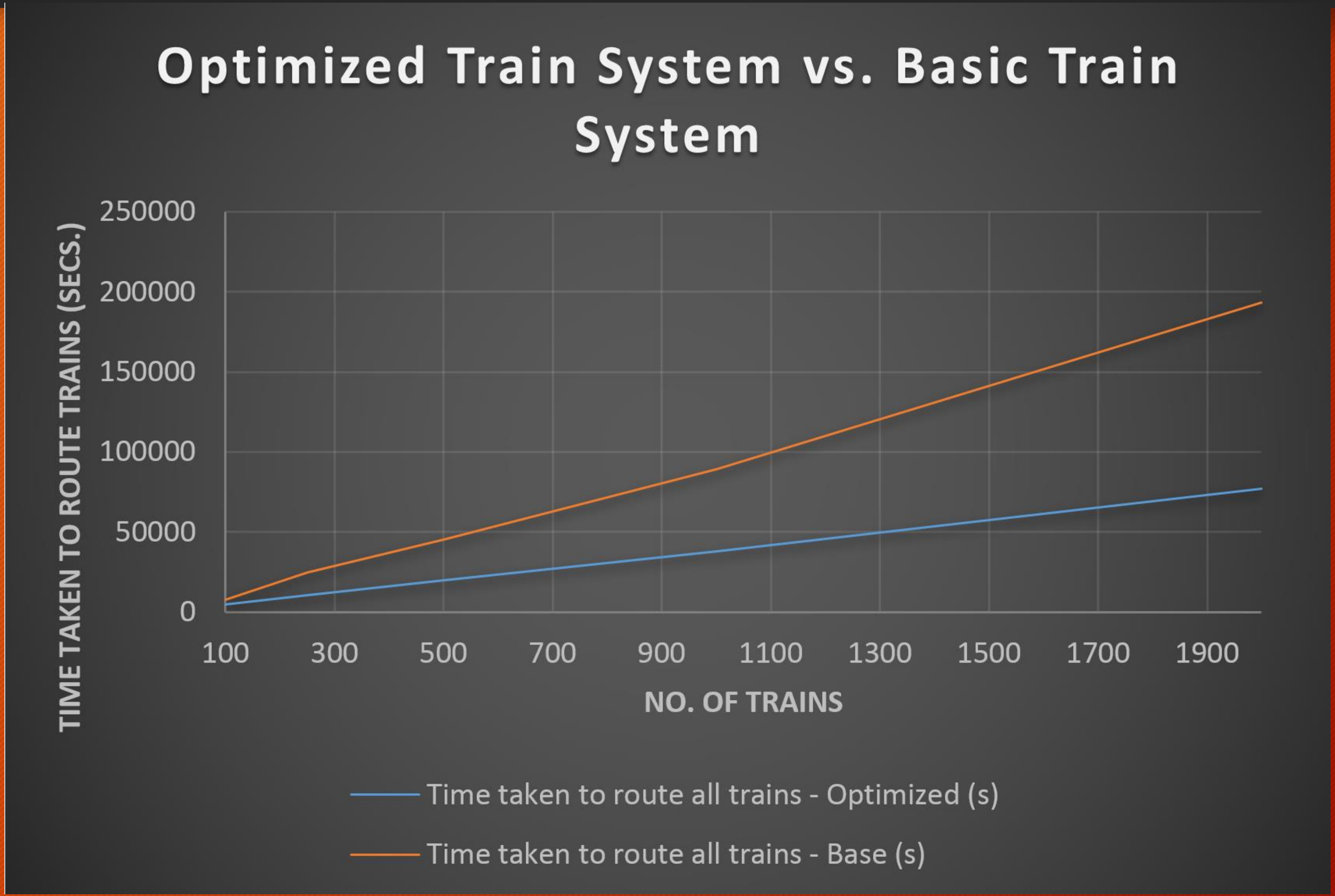
Simulator.java

- It visualizes the train using StdDtaw
- It shows the Locked (as Red) and Unlocked (as Green) tracks
- It uses 2 text files as input
  - Stations.txt(contains the grid of the stations)
  - Stations_path.txt (contains the path of the train tracks)

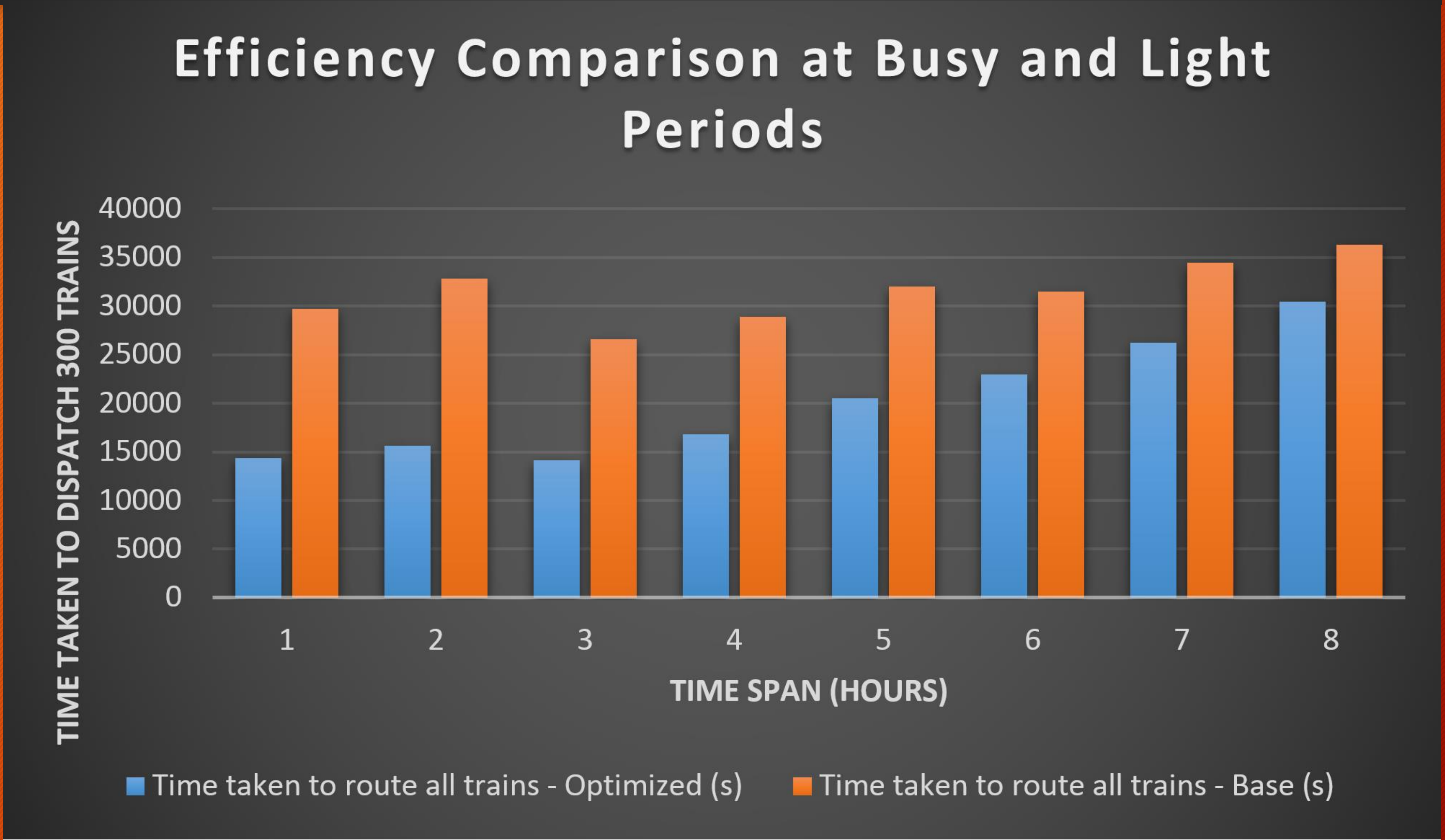
Base Case vs Optimized Case

- Base Case finds the shortest path and locks the whole path until it reaches the destination
- Optimized case has Three improvements
    - It frees the track after the train has crossed the train track instead of when it reaches the destination
    - Each time it arrives at the station it computes a new path
    - It only locks the path its currently travelling on.

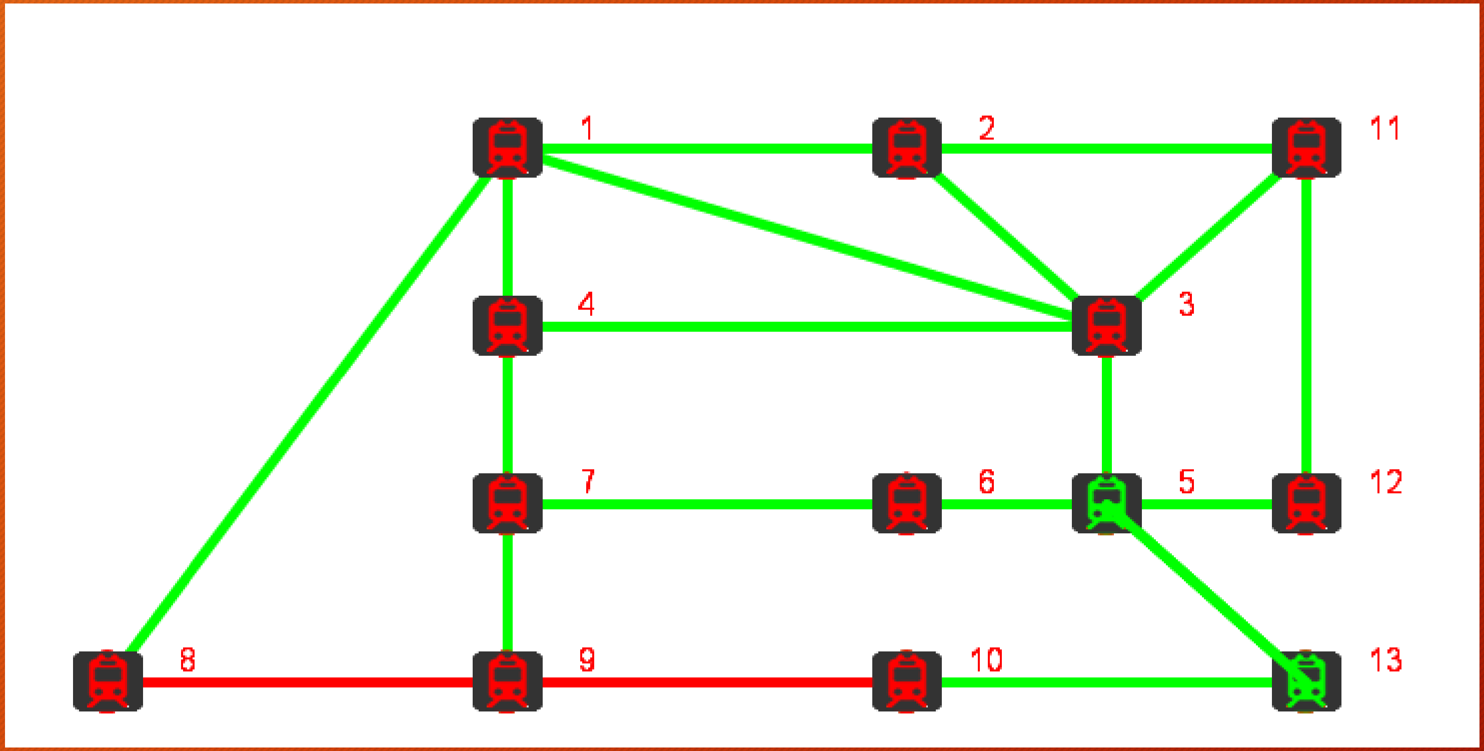
Results


Optimized Train System vs. Basic Train System
TIME TAKEN TO ROUTE TRAINS (SECS.)
250000
200000
150000
100000
50000
0
100
300
500
700
900
1100
1300
1500
1700
1900
NO. OF TRAINS
Time taken to route all trains - Optimized (s)
Time taken to route all trains - Base (s)

Results


Efficiency Comparison at Busy and Light Periods
TIME TAKEN TO DISPATCH 300 TRAINS
40000
35000
30000
25000
20000
15000
10000
5000
0
1
2
3
4
5
6
7
8
TIME SPAN (HOURS)
🟦 Time taken to route all trains - Optimized (s)
🟧 Time taken to route all trains - Base (s)

Simulator Presentation



Questions ?

Thank You