ECM1414 – LIFT ELEVATOR CONTROL SYSTEM SPECIFICATION

Sam Harries, Logan Westwood, Jacob Evans,
Xuanjin Qu, Yanbin Huang

Group IDs
Logan Westwood - 740039554
Jacob Evans - 740048598
Xuanjin Qu - 740020891
Sam Harries - 730039610
Yanbin Huang - 730061231

Introduction

Background

In modern multi-story buildings, efficient lift systems play a crucial role in ensuring a seamless user experience while optimising energy consumption. They operate on control systems which manage requests and schedule trips in the most efficient and fast way. Mis designed systems lead to large wait times, increased energy consumption and higher operating costs. In a world of buildings hundreds of meters tall, designing smart and adaptable control systems becomes a necessity to avoid the aforementioned risks.

Purpose of the Project

The main objective of this project is to develop an intelligent and adaptable lift control system that manages passenger requests whilst optimizing life movement – in an efficient and effective manner. The system is designed to accommodate a configurable number of floors and passenger requests, ensuring low wait times and good efficiency in it's scheduling process – leading to lower energy consumption and costs to the operator. Using the SCAN and LOOK algorithms as well as data structures (e.g. queues and priority queues), we are aiming to enhance the overall responsiveness and performance of lift operations.

Overview

This report is structured as follows:

- **Objectives:** Discusses the key goals of the project, including request handling, algorithm implementation and performance simulation.
- **System Overview:** Provides a description of the system and it's key features.
- **Specification:** Details the data structures used, algorithm specifications and implementation details
- **Results:** Presents a description of simulation scenarios, performance metrics and algorithm comparisons.
- **Discussion:** Interprets the results, highlights system limitations and suggests future improvements.

Objectives

The goal of this project is to develop a lift control system that efficiently manages requests from passengers, implements optimized scheduling algorithms, and conducts performance simulations to evaluate its effectiveness. The system must dynamically handle multiple passenger requests while minimising wait times and unnecessary lift movement.

A key objective is to ensure the system can handle incoming requests from passengers waiting to go up or down. The implementation must support concurrent requests, ensuring that all calls are processed in an optimised manner. To achieve this, the lift control system will utilise scheduling algorithms to determine the most efficient sequence of operations, minimising travel time and improving response efficiency.

Prioritisation is also a critical component of the system. Requests must be handled based on direction of travel, proximity to the lift, and wait times to ensure fairness and prevent delays.

To check the system's effectiveness, a simulation of real-world scenarios will be implemented using real-world constraints such as capacity limits, and the time taken to move between floors. Statistics will be collected and charts generated to visualise the data.

System Overview

High-Level Description

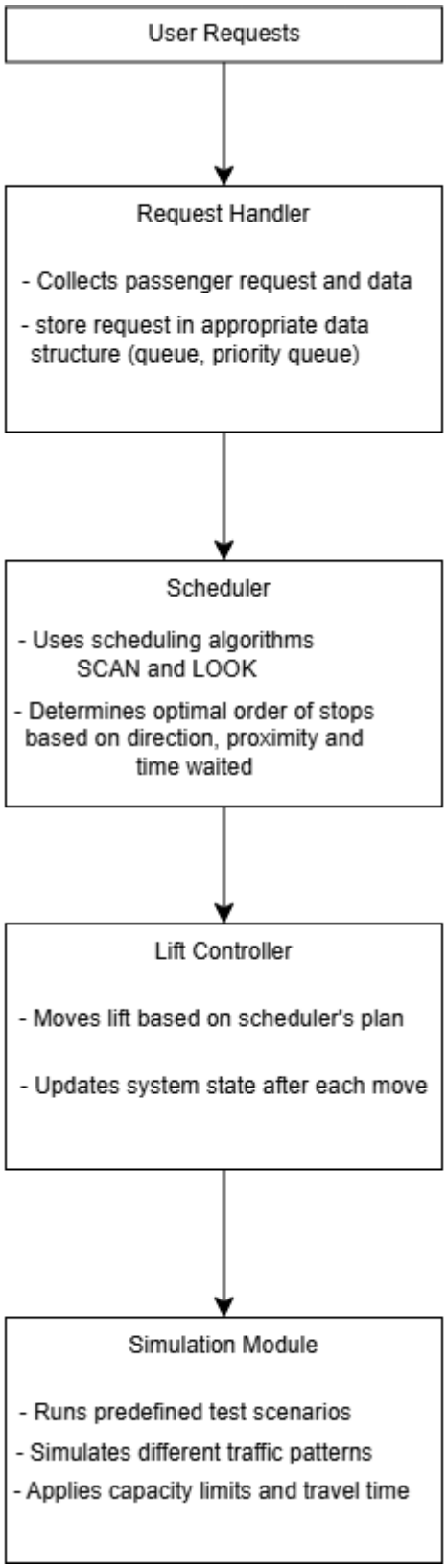
The system is designed to efficiently manage passenger requests in a multi-story building while optimising movement to minimise wait times and energy consumption. The system dynamically processes lift requests, schedules stops using algorithms, and simulates real world constraints such as capacity and travel time. Using data structures such as queues and priority queues alongside scheduling algorithms SCAN and LOOK ensure the system creates smooth lift movements despite varying patterns of demand. The key components of the system are:

1) Request Handler – Collects and processes lift requests from passengers, determining their desired destination and current floor.
2) Scheduler – Uses algorithms SCAN and LOOK to decide the most efficient sequence for handling requests
3) Lift Controller – Executes the movement of the lift based on scheduler decision, ensuring stops align with prioritised requests.
4) Simulation Module – Runs test scenarios with different building configurations and passenger demand levels, generating performance statistics.

Key Features:

1) Dynamic Request Handling – Processes multiple lift requests in real time, allowing for continuous operation and adaptive scheduling.
2) Efficient Scheduling Algorithms – Implements the SCAN and LOOK algorithms, reducing unneeded movement and optimising response times.
3) Scalability – Supports an adjustable number of floors and passengers, enabling the system to work for both small and large buildings
4) Simulation Performance Testing – Evaluates system performance under different traffic patterns and demand conditions, providing data on efficiency which can be used to iterate on and improve the system.
5) Prioritisation Mechanism – Determines which requests to serve first based on proximity, direction and time already waited by passengers.

Diagram


User Requests
Request Handler
- Collects passenger request and data
- store request in appropriate data structure (queue, priority queue)
Scheduler
- Uses scheduling algorithms SCAN and LOOK
- Determines optimal order of stops based on direction, proximity and time waited
Lift Controller
- Moves lift based on scheduler's plan
- Updates system state after each move
Simulation Module
- Runs predefined test scenarios
- Simulates different traffic patterns
- Applies capacity limits and travel time

Specification

Data Structures Used

The system relies on several key data structures to efficiently handle elevator requests and optimise scheduling. The system uses a queue (First In, First Out (FIFO) structure) to store floor requests in FIFO to ensure that requests are processed in the order they are sent/received. Additionally, a priority queue manages elevator requests based on priority, considering factors such as direction of travel and waiting time. The heap based-based structure allows for efficient retrieval of the highest-priority request, ensuring optimal scheduling.

A more specialised elevator request queue, implemented in ElevatorRequestQueue.java, extends the priority queue to manage real-time elevator calls. This structure prioritises requests based on the elevator's direction, reducing unnecessary stops and improving efficiency. Furthermore, Building.java and FloorState.java maintain the state of the entire system – tracking floors, the lift's position and pending requests. These components ensure that state changes are efficiently handled, allowing smooth execution of scheduling algorithms.

Algorithm Specification

Three key scheduling algorithms, SCAN, LOOK and MYLIFT, were used in the system. The SCAN algorithm, moves the lift in one direction until it reaches the last requested floor before reversing. This sweeping motion ensures fairness by servicing all requests systematically, making it particularly useful for large buildings with frequent requests.

The LOOK algorithm is a variation of SCAN that only moves as far as the last active request before reversing direction. This optimisation reduces unnecessary movement and improves response time, especially in environments with sparse requests. By limiting movement to active requests, LOOK provides a more efficient approach to elevator scheduling compared to SCAN.

MYLIFT enhances SCAN and LOOK by introducing dynamic decision-making based on wait times, floor skips and adaptive direction changes. MYLIFT tracks how long requests have been waiting, prioritising longer waiting floors over simple nearest floor logic. If a floor has been waiting too long, the elevator prioritises it even if it means reversing direction.

Instead of following SCAN or LOOK's sweeping motion, MYLIFT considers the distribution of requests. If most requests are below while moving up, it switches direction to where the most requests are. If the elevator passes too many floors without stopping, MYLIFT reassesses its direction to avoid inefficient long trips.

The time complexity of the algorithms also differ. SCAN has a worst-case complexity of O(N) per cycle, as it always sweeps the entire range of floors and N is the number of

floors. LOOK has a worst-case complexity of O(R), where R is the number of requests – as it only travels as far as the last requested floor and it's movements are proportional to the number of requests rather than total floors, making it faster when requests are clustered. MYLIFT has a worst case complexity of O(R + logN) per cycle. It iterates through R requests to determine priority (O(R)), uses hash maps to track wit times and visit history (O(1) lookups) and if implemented with a priority queue to select the most urgent request, it adds O(logN) operations for inserting/updating the queue. MYLIFT is in theory the most optimal in dynamic environments.

Implementation Details

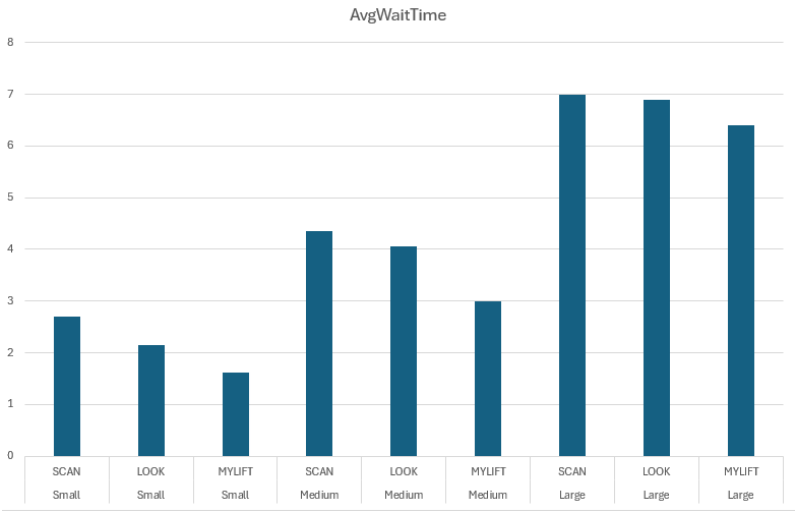
The system is initialised through ElevatorApp.java, where the program begins execution by reading the building configuration from input.txt. An instance of Building is created to manage floor and lift states. Once the system is running, user requests are added to ElevatorRequestQueue.java, which sorts and prioritises them using the priority queue.

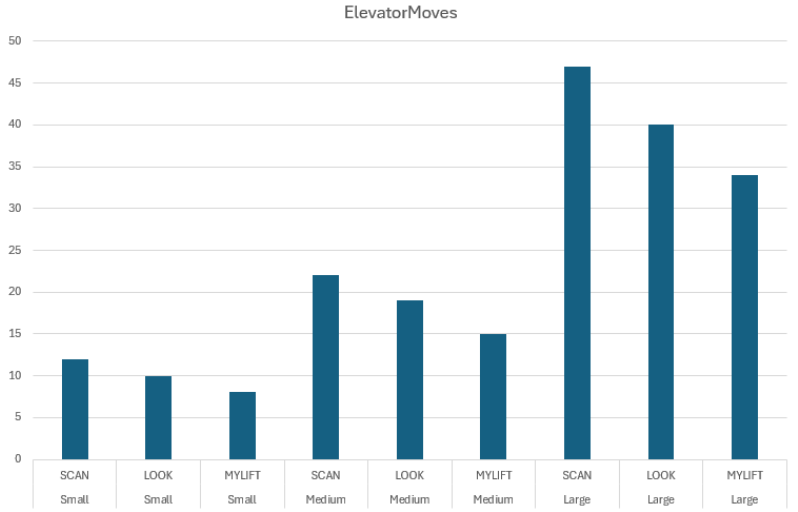
When a lift/elevator request is processed, the selected algorithm (SCAN, LOOK, MYLIFT) determines the next step using the NextStep() function in Algorithm.java. The system then updates lift movement and logs performance metrics. To evaluate efficiency, the implementation includes logging features that track key performance indicators such as wait times and overall system efficiency. These metrics are used for comparative analysis of the SCAN, LOOK and MYLIFT algorithms, providing insights into their effectiveness under different conditions.

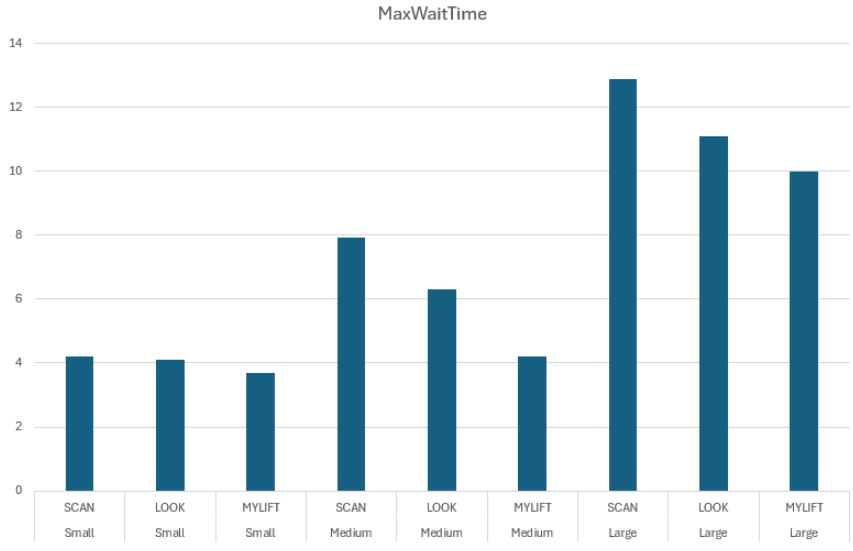
This ensures the lift system operates efficiently, adapting to varying building configurations and user demands while optimising passenger wait times and energy consumption.

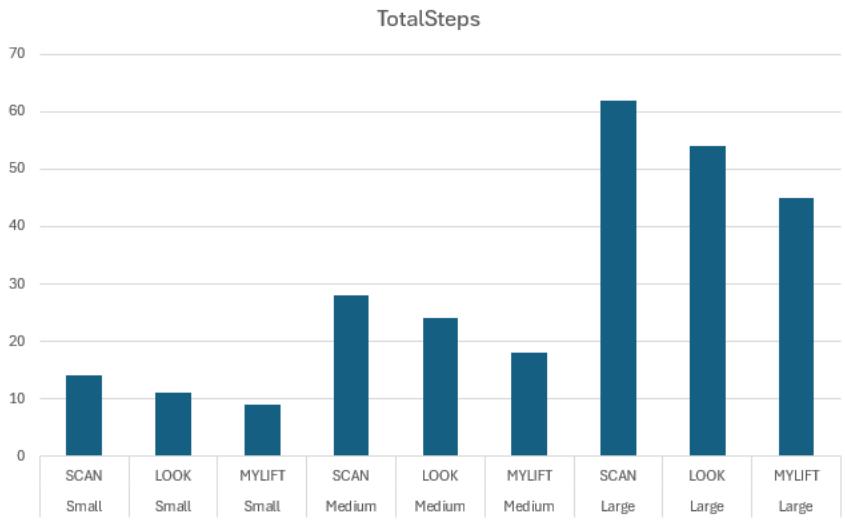
Results

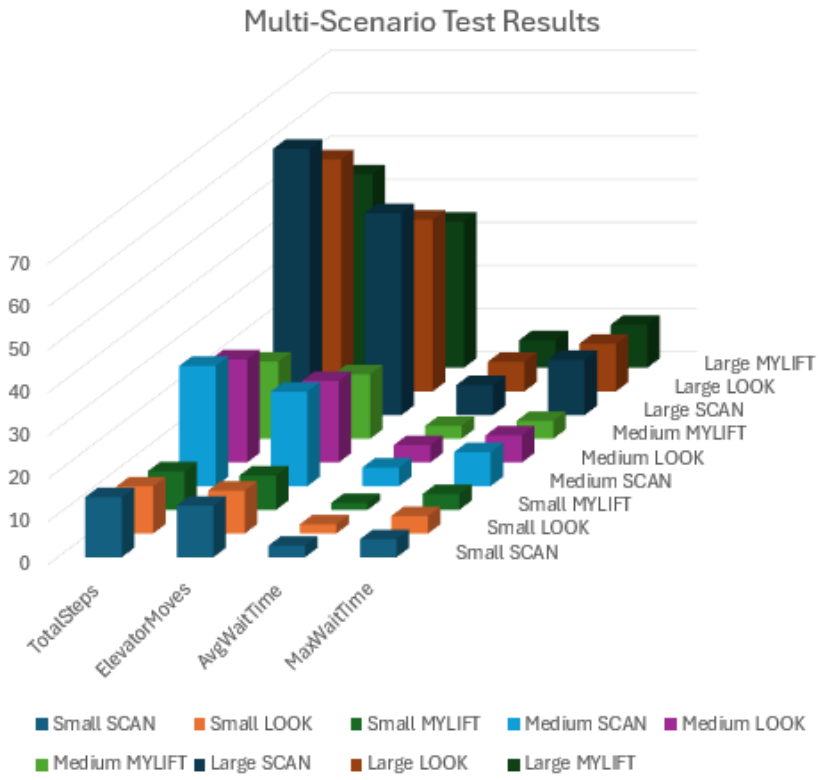
To analyse the algorithms' performance against each other, different test scenarios were developed in order to see how each algorithm operates under different circumstances. The metrics measured were: ElevatorMoves (number of), AvgWaitTimes, MaxWaitTimes and Total Steps. The algorithms were tested in small, medium and large buildings. See the charts below:


AvgWaitTime
8
7
6
5
4
3
2
1
0
SCAN Small
LOOK Small
MYLIFT Small
SCAN Medium
LOOK Medium
MYLIFT Medium
SCAN Large
LOOK Large
MYLIFT Large


ElevatorMoves
50
45
40
35
30
25
20
15
10
5
0
SCAN Small
LOOK Small
MYLIFT Small
SCAN Medium
LOOK Medium
MYLIFT Medium
SCAN Large
LOOK Large
MYLIFT Large


MaxWaitTime
14
12
10
8
6
4
2
0
SCAN Small
LOOK Small
MYLIFT Small
SCAN Medium
LOOK Medium
MYLIFT Medium
SCAN Large
LOOK Large
MYLIFT Large





Discussion

It is very clear immediately that SCAN is the least efficient, MYLIFT is the most efficient and LOOK is in between. This pattern holds for the different test scenarios small, medium and large buildings. A pitfall of this analysis is the fact that while different building sizes were tested, this is as far as it goes in terms of detail (due to time constraints). In the future, analysis can extend to buildings with clustered requests, sparse requests etc to get a complete picture of the different algorithms' performance relative to each other in very different scenarios.