ECE 590 Final Project: Scheduling with E Elevators

Hansung Kang, Kevin Kuo

May 4, 2018

1 Introduction

Network World reports that the cumulative time that office workers spend waiting for elevators in a major city is around 9 years. We spend a lot of time on elevators and waiting for elevators. Even though elevators are common, we have not yet found an optimal solution for the elevator scheduling problem that minimizes everyone's time spent on the elevator.

This problem is still an open problem and is interesting because of the ubiquitousness of elevators. The elevator scheduling problem is reducible to a travelling salesman problem, which is NP-hard. In the real world, there exist many implementations of this problem, but there is not one industrial standard.

Amazon, Inc chooses to schedule an elevator using a kiosk system. In this implementation of the elevator, there exists one kiosk at each floor that prompts the users to choose which floor they'd like to go to. Then, the kiosk tells them which elevator to take, and the elevator takes the user to her destination floor.

Another implementation includes when the elevator always goes up, or always goes down, and collects people who need to go in that respective direction until all requests are completed.

These varying implementations exist across the world, and some try to optimize user experience while others guarantee that each user reaches the destination. We aim to develop a model of an elevator and check for its specifications.

2 Problem

There exist many algorithms for implementing an elevator, and finding the optimized algorithm is beyond the scope of this class, as it is an unsolved problem and is NP-hard. However, we plan to model and verify a One Elevator model of the elevator system, and then extend it to multiple elevators. In this system,

we have to take into account many variables. These variables include, but are not limited to, the following:

- The destinations floors chosen inside the elevator
- The user's intention on each floor (go up, or go down)
- The current direction the elevator is moving

We build a model with these variables in mind and check the model using the model checking techniques presented in class, using UPPAAL.

3 Approach

We first model one elevator, and then extend to E elevators by allocating the inputs to the elevator system among the elevators.

4 1-Elevator System

For the 1 elevator system, we model the elevator in what we call a "non-interrupted service" philosophy. We first present the algorithm. Then we present the model, and some results of verification of its properties.

4.1 Algorithm

The algorithm is simple. The basic idea is to not stop the elevator for requests that require the elevator to change direction before fulfilling all requests that are possible without changing direction. If the elevator is going up, then any requests for the floor beneath the elevator will be ignored, and any requests to go down will be ignored, except for the top floor. The same holds true for when the elevator is going down. The algorithm will not be discussed in detail in this report, and it can be found in the uppaal implementation included with this report.

4.2 Model

The model is simple. There are many states, some represented as variables and some represented using state buttons. I will just list the variables here, so we can look at the algorithm. The block diagram can be found in Figure 1.

Let there be n floors. Here, $n = 4$ for demo purposes.

int[0,1] door_closed;
int[0,1] stopped;
int[0,1] elev_dests[4]; // floor requests from inside the elevator
int[-1,1] build_dests[4]; // direction requests from outside the elevator

int[-1,1] direction; // -1 is going down, 1 is going up, 0 is not moving
int floor; // 0, ..., n-1
int [0,1] blocked;

The elevator and door models can be seen below in figures 1 and 2.

4.3 Verification of Properties

There were many properties we wanted to test and verify. The most important of these being that if there is a service request, then it is fulfilled. On Figure 3, the first and third property checks that. Simply said, if there exists any request, then it implies that it will have been fulfilled.

For fault tolerance, the 2nd and 4th properties check whether if there are no current requests, then is it possible for a request to appear randomly with no additional outside input. The 5th property checks whether if the door can be closed and blocked at the same time, ever, and we're glad this property is satisfied. The 6th property checks whether there is a deadlock.

5 Extension to E Elevators

5.1 Approach

For E elevators, we wish to minimize the worst-case total time it would take for a person to wait for an elevator and get to their destination. In the following sections, we model the environment mathematically and, through a worst-case time analysis, arrive at an optimal algorithm to minimize the worst-case time.

5.2 Construction

We start by defining the sets $X_1 = \mathbb{Z}_N$ and $X_2 = \{\uparrow, \downarrow\}$. Letting $X = (X_1 \times X_2) - \{(N, \uparrow), (1, \downarrow)\}$, we can represent both the elevator position and the position of a person requesting an elevator:

For instance,

- If an elevator is at $(1, \uparrow) \in X$, then it is currently on the 2^{nd} floor (1^{st} floor is at $0 \in X_1$) and moving upwards.
- If a person requesting an elevator is at $(4, \downarrow) \in X$, then she is on the 5^{th} floor and wants to move downwards.

To make X easier to deal with, we wish to arrange its elements in a circle like in Figure 4. By setting the convention that elevators can only move clockwise in this circle, we can more easily visualize their paths. For instance, if an elevator wanted to move from the 3^{rd} floor heading up to the 2^{nd} floor heading down, it could travel clockwise from $(2, \uparrow)$ to $(N, \downarrow)$ to $(1, \downarrow)$. (Realistically, if the elevator did not have to stop anywhere else, it could just move straight down

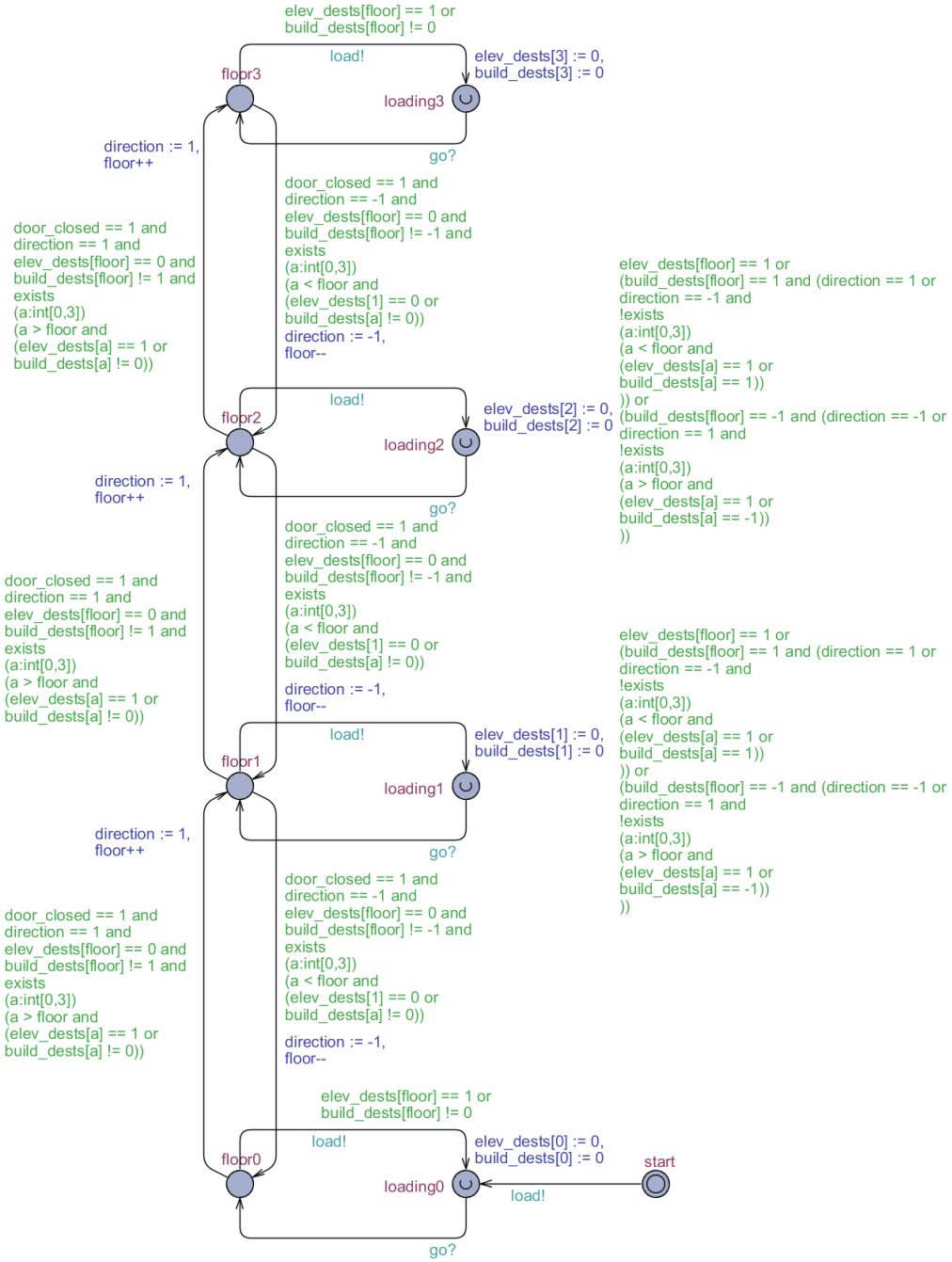

Figure 1: Elevator Model

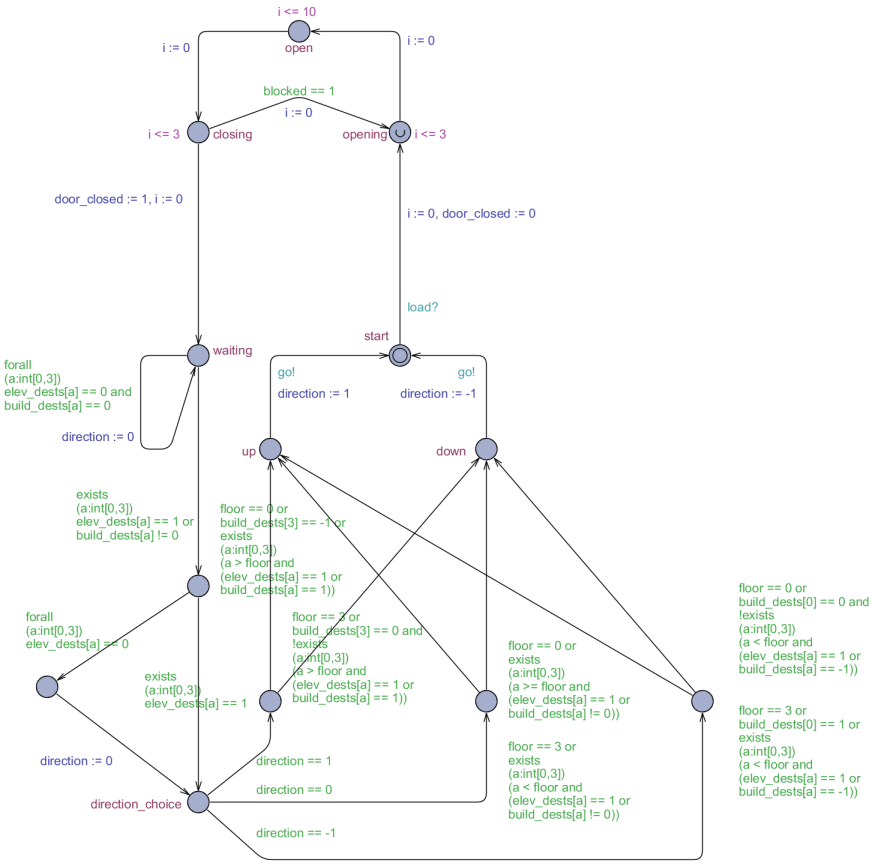

Figure 2: Door Model

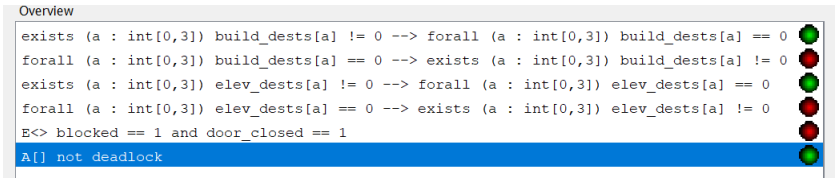

Figure 3: Verification

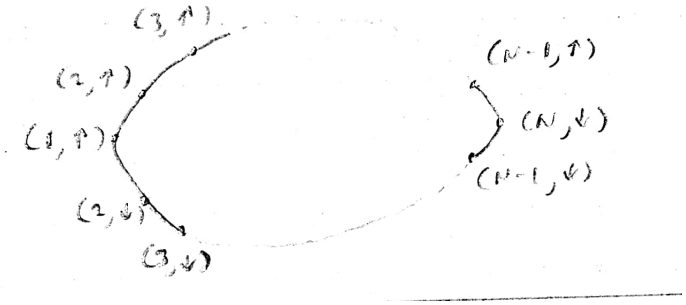

Figure 4: Representing X as a circle

– i.e. skip from $(2, \uparrow)$ to $(2, \downarrow)$, and then go to $(1, \downarrow)$. But, since the following analysis just considers the worst-case timing for the elevator (i.e. stops on every floor in between), we do not need to consider this shorter path.)

To create this circle, we map X onto $\mathbb{Z}_{2N-2}$ via $f : X \to \mathbb{Z}_{2N-2}$ defined as

$$f(x) = \begin{cases} \pi_1(x) & (\text{if } \pi_2(x) = \uparrow) \\ (2N-2) - \pi_1(x) & (\text{if } \pi_2(x) = \downarrow) \end{cases}$$

where $\pi_1(x)$ is the projection of x onto X_1, and $\pi_2(x)$ is the projection of x onto X_2.

Thus, using Figure 4 to picture $f(X)$, the top half the circle represents the values 0 to $N-1$ in $\mathbb{Z}_{2N-2}$, and the bottom half represents the values N to $2N-1$.

5.3 Worst-Case Time Analysis

Define the map $t : X \times X \to \mathbb{Z}_N$ to be $t(x_1, x_2) = f(x_2) - f(x_1) \ (mod\ N)$.

Proposition 1. *Suppose an elevator $e \in X$ had to travel from x_1 to x_2 ($x_1, x_2 \in X$), and the act of moving one floor and stopping took one time unit. Then $t(x_1, x_2)$ is the worst-case time it would take to make that travel.*

Proof. e must travel clockwise in Figure 4. Thus, letting T be the time e would take, $T \leq f(x_2) - f(x_1) \ (mod\ N) = t(x_1, x_2)$. This upper bound is achievable if e has to stop at every floor along the path from $f(x_1)$ to $f(x_2)$. □

If a person waits at some point $x \in X$ with destination $d \in X$, and elevator $e \in X$ had to come to pick them up, then, by Proposition 1, the worst-case time to travel from e to x is $t(e, x)$, and the worst-case time to travel from x to d is $t(x, d)$. Thus, the worst-case time T it would take for elevator $e \in X$ to pick up the person and take them to their destination would be

$$T = t(e, x) + t(x, d)$$

Let $\{e_1, e_2, ..., e_E\} \subset X$ be a set of elevators. Then let T_{e_i} be the worst-case time defined above for elevator e_i. Thus

$$T_{e_i} = t(e_i, x) + t(x, d) \tag{1}$$

Proposition 2. *For some* $i \in [1, E]$, $\left(T_{e_i} = \min_{j \in [1,E]} T_{e_j}\right) \Leftrightarrow \left(t(e_i, x) = \min_{j \in [1,E]} t(e_j, x)\right)$

Proof. If $T_{e_i} = \min_{j \in [1,E]} T_{e_j}$, then, by (1), $t(e_i, x) = T_{e_i} - t(x, d) = \left(\min_{j \in [1,E]} T_{e_j}\right) - t(x, d) = \min_{j \in [1,E]} \left(T_{e_j} - t(x, d)\right) = \min_{j \in [1,E]} t(e_j, x)$.

If $t(e_i, x) = \min_{j \in [1,E]} t(e_j, x)$, then $T_{e_i} = t(e_i, x) + t(x, d) = \left(\min_{j \in [1,E]} t(e_j, x)\right) + t(x, d) = \min_{j \in [1,E]} (t(e_j, x) + t(x, d)) = \min_{j \in [1,E]} T_{e_j}$ □

5.4 Algorithm

Going off the previous section, to minimize the worst-case time the person at x needs to wait to get to her destination, we pick the elevator e_i such that $T_{e_i} = \min_{j \in [1,E]} T_{e_j}$. By Proposition 2, this is equivalent to choosing the elevator e_i such that $t(e_i, x) = \min_{j \in [1,E]} t(e_j, x)$. Thus, by the definition of t, we choose the elevator e_i such that $f(x) - f(e_i) \ (mod\ N)$ is minimized. Note that this expression does not depend on the destination floor the person wants to go to. Thus, in a rough sense, just choosing the "closest" elevator to the person requesting one is the optimal approach to minimizing the worst-case time.

The following algorithm summarizes the approach we derived:

Let S be a system of E elevators $\{e_1, e_2, ..., e_E\} \subset X$. An input to S would be a person represented as an element in X. We are basically dividing the total set of inputs to S among the elevators.

Algorithm 1 Minimize worst-case travel time

```
1:  procedure MinimizeWorstCaseTime
2:      for input x ∈ X into S do
3:          minWeight ← ∞
4:          minElevator ← ∅
5:          for each elevator e ∈ {e_1, e_2, ..., e_E} ∈ X do
6:              criterion ← f(x) − f(e)
7:              if criterion < minWeight then
8:                  minWeight ← criterion
9:                  minElevator ← e
10:         add x to the set of minElevator's inputs
```

5.5 A Rough Average-Time Analysis

Proposition 3. *Suppose that the people requesting elevators are uniformly distributed throughout the building. Also assume that their destination floors are uniformly distributed throughout the building. Then, in the long term, the elevators must also be uniformly distributed throughout the building.*

Proof. Algorithm 1 does not take into account the actual position of any person in the building – just the relative position with respect to the elevators. Since the people requesting elevators and their destination floors are uniformly distributed throughout the building, re-numbering the building floors by adding an arbitrary integer to each floor number (modulo N) does not change the values calculated in Algorithm 1, and so the long-term distribution of elevators in the building is unchanged. If the elevators tended to be more focused on a specific set of floors, then the re-numbering process would cause the elevators to be focused on a different set of floors (since those floors would be re-numbered). This would contradict the fact that the elevators' long-term distribution should remain unchanged after the re-numbering, and so thus the elevators must be uniformly distributed throughout the building. □

Thus, using Algorithm 1, since the elevators are uniformly distributed in the long term, a rough estimate of the average time a person requesting an elevator would have to wait is $\frac{1}{2}\frac{N}{E}$ time units. For $E = 1$, this lines up with intuition – if a person's position is uniformly distributed, and the elevator's position is uniformly distributed, then the absolute difference between the person's and elevator's positions is uniformly distributed between 0 and N, and so the person would have to wait $N/2$ time units on average.

5.6 Pitfalls

In this algorithm, we do not consider the following factors:

- Elevator capacity
- What happens when an elevator fails
- Time it takes an elevator to collect people at a floor
- The actual time it takes for an elevator to travel between floors.

We can address the first two bullet points: Step 5 of Algorithm 1 could be modified to not consider elevators that are either full or dysfunctional.

There is another pitfall to this approach: we did not consider an elevator's existing stopping points. This could lead to a scenario where one elevator could be assigned a lot of people and others would not; the elevators could become unbalanced in the number of people they have to accommodate. In Algorithm 1, this can be accounted for by improving the criterion set in Step 6. For instance, the criterion could be set to $f(x) - f(e) + P_e(n)$, where $P_e(n)$ is some weighting

factor dependent on the number of people n currently in elevator e. If $P_e(n)$ increased monotonically with respect to n, then elevators that are more empty would become more likely to be chosen for the person at x.