

## Decision-Theoretic Group Elevator Scheduling

**Daniel Nikovski and Matthew Brand**

Mitsubishi Electric Research Laboratories  
201 Broadway, Cambridge, MA 02139, USA  
nikovski@merl.com, brand@merl.com

### Abstract

We present an efficient algorithm for *exact* calculation and minimization of expected waiting times of all passengers using a bank of elevators. The dynamics of the system are represented by a discrete-state Markov chain embedded in the continuous phase-space diagram of a moving elevator car. The chain is evaluated efficiently using dynamic programming to compute measures of future system performance such as expected waiting time, properly averaged over all possible future scenarios. An elevator group scheduler based on this method significantly outperforms a conventional algorithm based on minimization of proxy criteria such as the time needed for all cars to complete their assigned deliveries. For a wide variety of buildings, ranging from 8 to 30 floors, and with 2 to 8 shafts, our algorithm reduces waiting times up to 70% in heavy traffic, and exhibits an average waiting-time speed-up of 20% in a test set of 20,000 building types and traffic patterns. While the algorithm has greater computational costs than most conventional algorithms, it is linear in the size of the building and number of shafts, and quadratic in the number of passengers, and is completely within the computational capabilities of currently existing elevator bank control systems.

Keywords: decision-theoretic planning and scheduling, applications of planning and scheduling, group elevator scheduling, dynamic programming, Markov chains

### Introduction

Group elevator scheduling is a hard problem that has been researched extensively due to its high practical significance (Bao *et al.* 1994; Koehler & Ottiger 2002). The problem is simply stated: New passengers arrive at a bank of elevators at random times and floors, making hall calls to signal for rides up or down. A ride destination is unknown until the passenger enters the car and makes a car call to request a stop. The scheduler must assign a car to serve each hall call in a way that optimizes overall system performance. The execution of the schedule is performed by alternating the direction of movement of each car and servicing all hall calls assigned to it in its current direction of motion.

The usual performance criterion to be optimized when scheduling passenger pick-ups is the average waiting time

(AWT) of all passengers in the system, i.e., the time period from the moment a passenger arrives until the moment this passenger boards some car, averaged over many arrivals. Alternative criteria are sometimes used as well, such as the average system time, defined as the time until a passenger arrives at the desired floor, or the average squared waiting time, which expresses a preference for a small variance in wait times.

Minimizing any of these criteria is an extremely complicated problem, for at least three reasons. First, the state space of the system is huge, since it is indexed by the position, direction, and velocity of all cars, the number of passengers inside each car, and the number waiting at each floor to board a car. A truly optimal scheduler would have to consider all this information when deciding how to serve a newly arrived passenger. Second, the dynamics of the system are accompanied by a large amount of uncertainty. While the motion of a car is completely determined by its current schedule, this schedule changes constantly, because it depends on the future arrival of passengers, which is a stochastic process. Passenger arrival events contain three types of uncertainty: the time of arrival, the floor of arrival, and the passenger's final destination. Third, if the scheduler is allowed to revoke previous assignments and continuously reassign calls to cars, an exponential number of calls-to-cars assignments have to be considered in a short time. (This mode of operation, typical of elevator systems in western countries, is known as a *reassignment policy*; the converse mode, when the scheduler never reconsiders an assignment after it has been made, is known as an *immediate policy*, and is typical in Japan.)

The insurmountable combinatorial complexity and stochastic nature of the problem have led practitioners in the field of elevator scheduling to consider alternative, more tractable optimization criteria that are hoped to correlate well with the AWT of passengers. The following section discusses several practical approaches used in commercial systems, as well as several academic studies in which idealizations of the problem have led to important insights, albeit with limited practical applicability. Our approach is rooted in the academic work, but the result is very practical: A linear-time algorithm that directly optimizes the AWT. Subsequent sections introduce the assumptions and general operation of our algorithm, and describe how the general

procedure, which has exponential complexity, is reduced to an efficient algorithm by embedding a discrete-state Markov chain in the continuous phase space of an elevator car, and evaluating the AWT represented by the chain by means of dynamic programming. Experimental results on a detailed commercial-grade simulator are presented as well.

### Supervisory Group Elevator Scheduling

Group elevator control is a specific planning and scheduling problem characterized by a very large state space, significant uncertainty, and numerous resource constraints such as finite car capacities, previous car calls, etc. As a result, most of its early proposed solutions have not been based on either classical AI planning or decision-theoretic planning, but rather on *ad hoc* approximations and heuristics.

The oldest elevator schedulers used the principle of *collective control* (Strakosch 1998), according to which cars always stop at the nearest call in their running direction. This strategy is far from optimal and usually results in *bunching* – the phenomenon where several cars arrive at the same floor at about the same time, with all cars but one wasting time. Hikihara and Ueshima (Hikihara & Ueshima 1997) analyzed the jamming effect occurring in down-peak traffic and concluded that it was due to emergent synchronization between multiple cars. Another approach is *zoning*, or *sectoring*, where the building is divided into zones, and each car is assigned a single zone. While this approach avoids bunching, it is also suboptimal when many passenger arrivals occur in the same zone (Barney & dos Santos 1985; Strakosch 1998).

Otis Elevator Company uses an optimization criterion consisting of a weighted sum of bonuses and penalties, called Relative System Response (RSR), computed for each car in turn (Bittar 1982). This criterion is largely heuristic and its relation to actual AWT is not clear. Otis also uses another criterion called Remaining Response Time (RRT), defined as the time necessary for a car to reach the floor of the new hall-call, given its current commitments for loading and unloading passengers already assigned to it (Powell & Williams 1992). This criterion is in fact well correlated with the AWT of the passengers who have signaled the current hall call, but misses completely the effect a potential assignment would have on hall calls previously assigned to that car. Furthermore, RRT includes only the time necessary for a car to pick up passengers assigned to it, but ignores the time required for these passengers to get off, since it is not known whether they would disembark before or after the new hall call is serviced.

Another group of scheduling methods uses fuzzy-logic rules which are supposed to prescribe the correct assignment of a car to a new hall call in a small number of prototypical situations (Ujihara & Tsuji 1988; Ujihara & Amano 1994). The ability of fuzzy inference to generalize over similar situations is used to ensure coverage of the whole state space of the system. The rules are either elicited from experts, or induced from a training set. It is a matter of speculation whether the rules are correct even in the prototypical situations, and whether the fuzzy inference mechanism generalizes those rules correctly to novel situations.

More rigorously motivated methods use approximations of the desired performance criterion that are computable in reasonable time. Kone Corporation employs a method called the Enhanced Spacing Principle, which computes an approximation of AWT based on estimating the probable number of stops and most likely reversal floor of a car servicing its current commitments (Siikonen 1997). The method proposed by Cho, Gagov, and Kwon uses the same idea, and extends the method to handle arbitrary probability distributions over destination floors (Cho, Gagov, & Kwon 1999). The correct computation of the probable number of stops and most likely car reversal floors is essential to these methods, and usually several well-known statistical formulae are employed (Barney & dos Santos 1985). These formulae, however, are only applicable under the very restrictive assumptions that an elevator car would reach its contract (maximal) speed within a distance no longer than half the space between two neighboring floors, i.e., it would be able to accelerate fully and stop completely during a trip between two consecutive floors. This assumption is grossly violated for modern elevators: even a typical elevator with contract speed of 180 m/min needs at least three floors to accelerate fully and stop completely, and the world's fastest elevators, installed by Mitsubishi Electric in the Yokohama Landmark Tower, have contract speeds of 750 m/min and require 39 floors in order to reach maximal speed and then come back to rest (Tanahashi & Araki 1994).

The problem of group elevator scheduling has also been approached from the point of view of classical planning, expressing the problem domain using the PDDL formalism (Koehler & Schuster 2000; Koehler 2001). Several obstacles to this approach have been identified, such as the lack of support for metric/resource constraints, no consideration for cost functions during planning, and lack of optimization capabilities. To these, we will also add the inability of classical planners to reason and plan under uncertainty — in fact, the proposed PDDL-based solution is only applicable to elevator banks with full destination control, i.e., all ride destinations are registered in advance on a destination panel (Koehler & Ottiger 2002). However, decision-theoretic planning, which is an extension of classical planning to stochastic and partially-observable domains, in conjunction with queueing theory models, seems like a very suitable formalism for this problem (Boutilier, Dean, & Hanks 1999).

Special traffic patterns, such as down-peak and up-peak traffic, can be handled very efficiently by special-purpose algorithms based on queueing theory. A provably optimal solution has been obtained for the case of pure up-peak traffic, when all passengers arrive in the lobby at a fixed rate and no other departure floors are allowed (Pepyne & Cassandras 1997). In order to make the problem tractable, however, the service time of elevators has been assumed to come from a fixed exponential distribution. This assumption, along with the requirement for pure up-peak traffic, severely limits the usefulness of the algorithm in practical schedulers.

For the case of down-peak traffic, similarly efficient algorithms are Finite Intervisit Minimization (FIM) and Empty the System Algorithm (ESA) (Bao *et al.* 1994). While they

have been demonstrated to outperform simpler algorithms by a margin of 34%, FIM and ESA are only applicable to down-peak traffic, because they assume that the destination of all passengers is the lobby. As soon as there is uncertainty in passengers' destinations, the assumptions of these algorithms are violated and they cannot be expected to perform well.

Nevertheless, the ESA algorithm contains a very important idea: Instead of minimizing AWT of all known and future arrivals, it limits the optimization only to the residual waiting time (RWT) of the passengers currently in the system. This includes all passengers currently in elevator cars and all passengers who have signaled their presence by making hall calls for elevators. The RWT of a single passenger is defined as the time between the current moment and the moment this passenger is picked up by a car (Bao *et al.* 1994). Minimizing the RWT of known passengers instead of the AWT of all known and future passengers is equivalent to the assumption that the current decision (assignment of a car to the current hall call) would not influence the waiting times of future passengers. While this assumption is clearly not true and should lead to suboptimal policies, its consequences can be expected to be relatively benign, since it can be expected that the stochasticity of the arrival process would eliminate the influence of the current decision in the long run.

In computational terms, this assumption eliminates two of the three sources of uncertainty identified above: the exact arrival times of passengers and the exact floor of their arrival. This is due to the fact that the scheduler needs only consider those passengers who have already arrived but have not been served yet—their exact arrival times and floors are known. The only uncertainty that remains is that of their destination floors. In this paper we show that one can compute exactly the expectation of the RWT of all known passengers with respect to an arbitrary probability distribution of their destination floors.

Before continuing, we will note the existence of algorithms that also consider the consequences of the current decision on future arrivals. Crites and Barto demonstrated an asynchronous algorithm for stochastic optimal control which uses neural networks and Q-learning (Crites & Barto 1996). Although their algorithm performed slightly better than FIM and ESA for one specific down-peak scenario (by 2.65%), it took 60,000 hours of simulated elevator operation to converge, which is not practical for real elevator systems. While trainable algorithms seem very promising for this problem area, the issue of correct generalization over the enormous state space and infinite horizon is very forbidding.

## Dynamic Programming for Exact Computation of Expected Average Waiting Times

### Initial Assumptions

Our key assumption, motivated above, is that future arrivals need not be factored into decisions about current passengers. There are in fact several ways in which this can be relaxed;

we begin with the strong assumption for simplicity of exposition. Under this assumption, the most informed decisions are made when the waiting times are estimated all the way out to the horizon where all known passengers have been delivered to their destinations. Hence this is an empty-the-system algorithm, but unlike the original ESA algorithm, our approach accommodates all traffic patterns and uncertainty about the state of the system. In short, we retain the ESA strategy but introduce new machinery for inference.

We will initially describe the algorithm under the assumption that the destination floors of passengers are equally likely, and later on explain how non-uniform destination probabilities can be handled (at a significant computational expense). We also assume that the full state of the system is known to the scheduler — most importantly, we assume that the number of people standing on each floor is known. While such information cannot be obtained only by inspecting the number of hall buttons pressed, approximations of various quality exist and will be discussed below. A scheduler operating under this assumption is known as an *omniscient* scheduler.

Another key assumption concerns the way passengers assigned to a car are being served. In general, if  $n$  passengers are assigned to a car, there are  $n!$  possible orders for them to be picked up. If all orders are allowed and will be considered by the scheduler, the corresponding planning problem has been shown to be NP-hard even for a single car (Seckinger & Koehler 1999). However, there exists a simple order of serving passengers assigned to a car that also conforms well to passengers' expectations and is rarely suboptimal: Keep moving in the current direction until all passengers who have requested rides in this direction are picked up and delivered; after that, move to the first hall call in the opposite direction, and repeat the same procedure for all opposite hall calls. Our planner assumes that all assignments would be served in this manner.

The discussion in this section assumes an immediate assignment policy, as is customary in Japan, i.e., new assignments are appended to the current schedule and previous assignments are never changed. Using the algorithm for a re-assignment policy would simply involve employing it as a subroutine on a set of proposed assignments, generated either exhaustively in a combinatorial manner, or after pruning the set of candidate assignments by means of heuristics or a systematic branch-and-bound algorithm, similarly to other scheduling algorithms (Bao *et al.* 1994). The last assumption we are making is that each car has infinite capacity — while not realistic, this assumption simplifies significantly the algorithm, and we will discuss possible ways to relax it.

### Optimization Criterion

Whenever a new hall call is generated at a particular floor in a particular direction, the algorithm minimizes the total residual waiting time of all currently waiting passengers, including the new arrival. All such passengers except the new one have already been assigned to a car; under the immediate assignment policy, their assignments will never be reconsidered. If the elevator group has a total of  $N_c$  cars, let  $W_i^-$ ,  $i \in [1, N_c]$  denote the expected waiting time of all

passengers currently assigned to car  $i$ , *excluding* the newly arrived passenger(s) signaling the current hall call, and similarly, let  $W_i^+$ ,  $i \in [1, N_c]$  denote the expected waiting time of all passengers currently assigned to car  $i$ , *including* the newly arrived passenger(s). We can then compute the expected waiting time  $W_i$  associated with assigning the new call to car  $i$  as

$$W_i = W_i^+ + \sum_{\substack{j=1 \\ i \neq j}}^{N_c} W_j^-, \quad i \in [1, N_c].$$

The car  $c$  chosen by the scheduler for assignment is the one, which minimizes the total expected RWT:  $c = \arg \min_i W_i$ . Note that since the number of waiting passengers is constant at the time of a particular decision step, such an assignment would also minimize the *average* expected RWT of current passengers, which is computed as the total RWT of all passengers divided by their number.

If  $W^- \doteq \sum_{i=1}^{N_c} W_i^-$ , the waiting times for each possible assignment can be expressed as  $W_i = W_i^+ - W_i^- + W^-$ . Since  $W^-$  is the same for each  $i$ , the assignment which minimizes  $\Delta W_i = W_i^+ - W_i^-$  is also the one which minimizes  $W_i$ . As a result, the optimal assignment can be found by computing  $W_i^+$  and  $W_i^-$  for each car, and choosing the car for which their respective difference is minimal.

Computing  $W_i^+$  and  $W_i^-$  for a particular car  $i$  is essentially the same problem. For  $W_i^-$  we compute the expected RWT given the state of the system and all currently scheduled elevator-to-passenger assignments. For  $W_i^+$  we temporarily add the new passenger to elevator  $i$ 's itinerary and recompute the expected RWT.

### Effect of Uncertainty in Passengers' Destination

By definition,  $W$  is the expected total RWT for all passengers currently assigned to be served by a car, subject to the constraints imposed by the car's current position, direction, and velocity, and the currently mandated stops at requested destination floors. The expectation of the waiting time is taken with respect to the uncertainty in the destinations of passengers who are yet to be picked up by the car. Since only their requested direction of travel is known, their destination can be any of the remaining floors in that direction. This is in contrast to the original ESA algorithm (Bao *et al.* 1994) which only works for pure down-peak traffic in which all passengers are delivered to the lobby. In classical ESA, the exact path traveled by the car is known, and the total waiting time of all passengers can be found by summing up the car travel times between floors where passengers are to be picked up, weighted by the number of passengers still waiting.

A straightforward extension of the ESA approach to the case when the destination of passengers is not known can be implemented by considering all possible destinations of each passenger (respectively, all possible paths the car can take), computing the total waiting time along each path, and weighting these times by the probability of the respective path. This is equivalent to generating a tree containing all

possible futures of the system (disregarding future passengers), and computing a weighted sum of the waiting times over all paths from the root to the leaves. If  $N_p$  passengers are assigned to a car in a building of  $N_f$  floors, each of the passengers has  $O(N_f)$  possible destinations, and the total complexity of such an implementation would be  $O(N_f^{N_p})$ —prohibitively high.

### Estimation of Expected RWT in Linear Time by Means of Dynamic Programming

It is possible, however, to reduce the complexity of computation to  $O(N_f N_p)$  by casting the problem into a dynamic programming framework. We will call the corresponding algorithm ESA-DP (ESA by Dynamic Programming).

Dynamic programming is commonly employed in stochastic scheduling algorithms where cost estimates on segments of a system's path can be reused in multiple paths (Bertsekas 2000). In order to solve a problem this way, one must typically discretize the state and identify branch points where system paths converge and then diverge again, so that the costs on a segment between two such points can be computed only once, and then reused for the computation of costs along all paths which include this segment.

### Trajectory Structure of an Elevator Car

Such branching points can readily be identified on the phase-space diagram of an elevator car shown in Fig. 1. Like any moving mechanical system, a car traveling in an elevator shaft has a phase-space diagram which describes the possible coordinates  $(x, \dot{x})$  for the car's position along the shaft  $x$  and its velocity  $\dot{x}$ . When the car is moving under constant acceleration without friction, its trajectory consists of segments which are parts of parabola, and more complicated equations of motion result in slightly different shapes of the traversed trajectories. However, even when the equations of motion of a car are nonlinear (e.g. include gear backlash) and/or include position derivatives higher than acceleration (e.g. jerk with a specified magnitude and duration), the motion of the car is very predictable and can be realized only on a small number of trajectories. Accordingly, these trajectories branch only on a small number of points, denoted by circles in Fig. 1, which always correspond to the last possible location at which a car should start decelerating if it is to stop at a particular floor in its direction of motion. A particular path of a car during its round-trip always consists of a finite number of such segments, whose endpoints are branching or resting points. Consequently, if the waiting time on each such segment can be computed, it can be reused for the computation along many paths that include that segment.

Reusing the costs on all individual segments can be achieved by embedding a discrete Markov chain into the original system of elevator movement, which in itself operates in continuous time and space. A Markov chain consists formally of a finite number of states  $S_i$ ,  $i \in [1, N_s]$ , an immediate cost  $C_{ij}$  of the transition between each pair of states  $S_i$  and  $S_j$ , a matrix  $P_{ij}$  of the probabilities of transition between states  $S_i$  and  $S_j$ , and a distribution  $\pi(S_i)$  which spec-

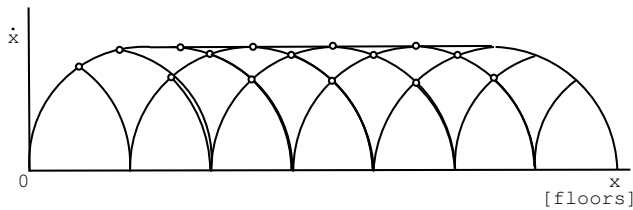


Figure 1: A schematic illustration of the phase space of a single car moving upwards in a shaft of a building with eight floors, not all of which have equal height. All branching points are denoted by circles.

ifies the probability that the system would start in state  $S_i$  (Bertsekas 2000).

In order for the chain to be Markovian, it should obey the Markov property: The probability  $P_{ij}$  of transitioning to state  $S_j$  should depend only on the starting state  $S_i$ , and not on the trajectory of the system before it entered  $S_i$ . If we define the states of the system to correspond only to the branching points in the phase-space diagram, the resulting chain would not be Markovian, because the probability of each branch depends on how many people are currently inside the car, and that number depends on how many of all waiting people have already been transported to their destinations in previous stops of the car.

Consequently, the number of people in the car must be included in the state of the Markov chain as well. However, the state needs only encode the number of currently waiting people who will board the car *after* the moment of assignment decision. This number does not include people who are already in the car at that time and have signaled their destinations by pressing car buttons. These “in-car” passengers influence the motion of the car too, by imposing constraints on its motion in the form of obligatory car stops, but these constraints are deterministic and have no impact on transition probabilities. These probabilities depend only on the uncertainty in the destinations of the passengers who are yet to board the car<sup>1</sup>.

Accordingly, a state  $S_i$  of the Markov chain is described by the four-tuple  $(f, d, v, n)$ , where  $f$  is the floor of the car,  $d$  is its current direction,  $v$  is its current velocity, and  $n$  is the number of *newly* boarded passengers, precisely, waiting passengers who enter the car in the course of evaluating the Markov chain. The variables  $d$  and  $n$  are discrete, and have predefined ranges:  $d$  can take only two values, “up” and “down”, while  $n$  ranges from 0 to the maximum number

<sup>1</sup>Technically, the disembarkation times of in-car passengers have some impact on the RWT of waiting in-hall passengers. Uncertainty over how many in-car passengers will disembark at each of the (known) requested stops could be marginalized out via dynamic programming with an expanded state descriptor. However, this represents a respectable increase in run-time for a negligible gain in accuracy; disembarkation times are very small relative to other time costs. We approximate the exact cost by assessing an  $N/n$ -second disembarkation cost for  $N$  in-car passengers and  $n$  requested stops. The disembarkation times of to-be-boarded passengers are modeled exactly.

of passengers assigned to a car, traveling in either direction. (This maximum number is reached, for example, when all passengers intend to get off the car at the last floor in the current direction of motion.)

The variables  $f$  and  $v$ , however, are essentially continuous, and in order to make the problem tractable, they have to be discretized. An inspection of the phase-space diagram suggests a straightforward discretization scheme for the velocity  $v$  — it can be seen that while accelerating from a particular floor, the car reaches branching points along its trajectory only at a small number of velocities (four in Fig. 1, including the quiescent state, when the velocity is zero). The reason for this is the limit on the maximum speed of any real elevator car. Depending on the inter-floor distance, maximum speed, and acceleration of the motors, this number of distinct velocities at branching points can be lower (for longer inter-floor distances, lower maximum speed, and greater acceleration), or higher (for shorter inter-floor distances, higher maximum speed, and lower acceleration). For a particular building and the elevator bank installed in it, this number is fixed and can be found easily, so henceforth we assume it is known and will denote it by  $N_v$ . Hence, the variable  $v$  would only take  $N_v$  discrete values, ranging from 0 (rest) to  $N_v - 1$  (maximum speed). Note that the same value of  $v$  can correspond to different physical velocities, depending on which floor the car stopped at last. Another interpretation of this variable is the number of branching points a car has encountered since its last stop.

There are several ways to discretize the floor variable  $f$ , the obvious one being to round the physical location of the car to the nearest floor. While such a discretization is possible, the resulting value for the floor is not conveniently related to the particular branching point represented by the Markov chain. A much more convenient discretization scheme is to choose for a value of  $f$  the floor at which the car *will stop* if it starts decelerating at that branching point. The advantage of such a discretization scheme becomes apparent, if we organize the states of the Markov chain in a regular structure, commonly called *trellis* in dynamic programming algorithms.

## Structure and Parameters of the Embedded Markov Chain

Fig. 2 shows a dynamic programming trellis for one particular Markov chain which corresponds to the situation when a car is moving down and is about to reach the branching point at which it will stop at floor 13, if it decelerates. It has already been scheduled to pick up a passenger at floor 7, and the scheduler is considering whether this car should also pick up a new hall call down, originating at floor 11. The embedded Markov chain has 84 states which are placed in a trellis matrix of 7 rows and 12 columns. States in a row represent branching points that share the property that the car will stop at the *same* floor, if it starts decelerating immediately. Note that this applies to branching points reached when the car is moving in a particular direction — when it is moving in the opposite direction, the branching points generally have different positions in the phase space diagram. The corresponding row of the trellis is labelled with the floor

at which the car can stop, as well as the direction of the movement of the car when it reaches the branching points. Since there is a separate row for each floor and direction, the trellis can have at most  $2N_f$  rows.

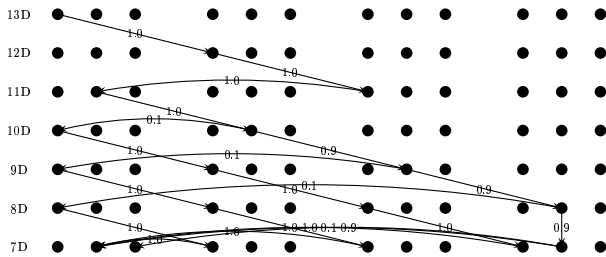


Figure 2: Simplified trellis structure for the embedded Markov chain of a single descending car. Rows signify floors; columns signify the number of recently boarded passengers; column groups signify elevator speeds. The empty descending car is about to reach the branching point for possible a stop at floor 13. It has been assigned hall calls at floors 7 and 11, each of which may increase the passenger load by one.

The states in each row of the trellis are organized into  $N_v$  groups (4 in figure 2), corresponding to the  $N_v$  possible velocity values at branching points (ordered so that the leftmost column correspond to zero velocity, and the rightmost column correspond to the maximum velocity of the car). Within a group, the states correspond to the number of people who are currently in the car and who were waiting in the halls at the beginning of the trellis (ranging from 0 to 2 in figure 2). If the maximum number of people that can be in the car at the same time is  $H$  (2 in figure 2), the width of the trellis is  $N_v(H + 1)$  states. This organization of states constitutes the trellis of the dynamic programming problem. It can be seen that not all of the states in the trellis can be visited by the car, because its motion is constrained by the current hall and car calls.

If we assume that the floor-value component  $f$  of the four-tuple used to describe a branching point is that of the floor where the car will stop, if it starts decelerating at this branching point, the first row of the trellis always contains the first branching point which the car will reach. Similarly, under this convention, the last row of the trellis always corresponds to the floor where the last passenger along the round-trip of the car will be picked up. This arrangement of rows conveniently spans the horizon which the dynamic programming algorithm has to consider, because the last moment which has to be considered is always the moment the last waiting passenger is picked up—after that, the residual waiting time of passengers assigned to the current car becomes zero.

The total cost  $C_{ij}$  incurred on a segment, measured as the waiting time of passengers who have not been picked up yet, can be expressed simply as the product of the number of these passengers and the duration of the segment.

The last remaining components of the embedded Markov chain are the transition probabilities  $P_{ij}$  of transitioning between each pair of states  $S_i$  and  $S_j$ . A large number of these transitions are deterministic and are always taken with probability one. Such are the transitions resulting from existing car and hall calls. For example, the initial trajectory of the car from floor 13 to floor 11 in Fig. 2 is deterministic — the empty car accelerates until it reaches the branching point for stopping at floor 11, where it stops to pick up the first hall-call passenger waiting there. After that, the car accelerates again until it reaches the branching floor for stopping at floor 10, from which it can take many different paths, depending on the unknown destination of that passenger.

At the branching point of floor 10, the passenger might be getting off at one of the next 10 floors, and hence the probability that this would be exactly floor 10 is 0.1. With probability 0.9, the passenger would not get off at floor 10, and the car will continue accelerating until the branching point for floor 9, with one passenger still on board, as reflected in the diagram of the Markov chain.

In the general case, when the car has  $k$  floors to go with  $n$  passengers on board, and we assume that a passenger would get off at any of the  $k$  floors with equal probability ( $1/k$ ), we can find the probability that  $x$  people would want to get off at the next floor by using the formula for the binomial probability function:

$$Pr(x, n, k) = \frac{n!}{(n-x)!x!} \frac{(k-1)^{n-x}}{k^n} \quad (1)$$

Therefore  $n-x$  people would remain on board the car with probability  $Pr(x, n, k)$ . A similar treatment will give  $Pr(x, n, k)$  when the destination probabilities are nonuniform but independent of where each passenger gets on; we show below how to exploit a matrix of destination probabilities that are conditioned on the floor of arrival. The number of remaining people  $n-x$  specifies which state within a group the Markov chain would enter with probability  $Pr(x, n, k)$ , but we still have to find which group (velocity setting) this state would be in. This velocity setting can be determined by inspecting the existing car and hall calls, as well as the number  $x$  of people getting off. If  $x > 0$  or there is a mandatory stop at the next floor due to a car or hall call, the velocity  $v$  at the next state would be zero; only when  $x = 0$  (nobody gets off the car at the next floor) and there are no car or hall calls for this floor, the car would accelerate (or maintain maximum speed, if it has already reached it).

The initial and terminal states of the Markov chain are always unique, and can easily be found by locating the first branching point the car would enter along its current trajectory, and the floor where the last waiting passenger would be picked up, respectively. The distance between the rows of the trellis containing the initial and terminal states effectively spans the planning horizon of the dynamic programming algorithm.

Once the initial state of the Markov chain has been found, the whole chain can be built by propagating the set of states which can be visited by the car from the initial state. The chosen arrangement of the states into a dynamic programming trellis provides a convenient order for doing this. By

inspecting the order of transitions in Fig. 2, it can be seen that if a transition is between different rows, the starting state is always in a row above the successor state, and if a transition is within the same row, the starting state is always to the right of the successor state. Consequently, the propagation of states proceeds row-wise from the upper-right corner of the trellis until the lower-left corner is reached. Each state might have multiple successor states, depending on the number of people in the car who might want to get off, and the probability of each of these transitions is determined by the binomial formula above.

It should be noted that trellis building and evaluation time can be significantly reduced by skipping highly improbable events. A simple way to do this is to simply neglect to add a transition for any disembarkation event whose probability (equation 1) lies below some threshold, for example, all passengers disembarking at the same mid-building floor. However, as the number of passengers in the system increases, nearly all disembarkation events have probability values close to zero. It is important to evaluate a sample of such cases representing the majority of the probability mass in equation 1. This is done very efficiently by adding transitions for events in order of descending probability, stopping when some large fraction of the probability mass is accounted for, say 99%.

### Evaluation of Travel Time

Once the trellis is built, it is used to evaluate the expected travel time of the car, starting from its current state. Opposite to the procedure for building the trellis, the algorithm for evaluating it starts from the bottom row of the trellis moving up, and processes the states in each row from left to right.

In essence, the algorithm iteratively computes the cost-to-go (expected remaining waiting time) of each state in the trellis that can be visited by the car, by means of a Bellman backup on all successor states, using the probabilities and costs for transitioning to these states. After all costs-to-go are computed, the overall expected residual waiting time from the moment the car enters the initial state is just the cost-to-go of that state.

This algorithm computes the expected remaining waiting time of a car's passengers only from the moment the car reaches the first state of the trellis. In general, however, when a hall call occurs, the car would be somewhere between two branching points. In order to find the total expected residual waiting time of a car's passengers from the moment a hall call occurs, the result returned by the algorithm has to be increased by the time it would take for the car to reach the first branching point, multiplied by the total number of passengers currently assigned to that car, in both directions.

### Experimental Verification of the Algorithm

The algorithm was benchmarked against a conventional method for supervisory group control which minimizes the round-trip time of the car along a single path, taking into consideration only stops due to existing car calls and stops where the car would pick up new passengers due to hall

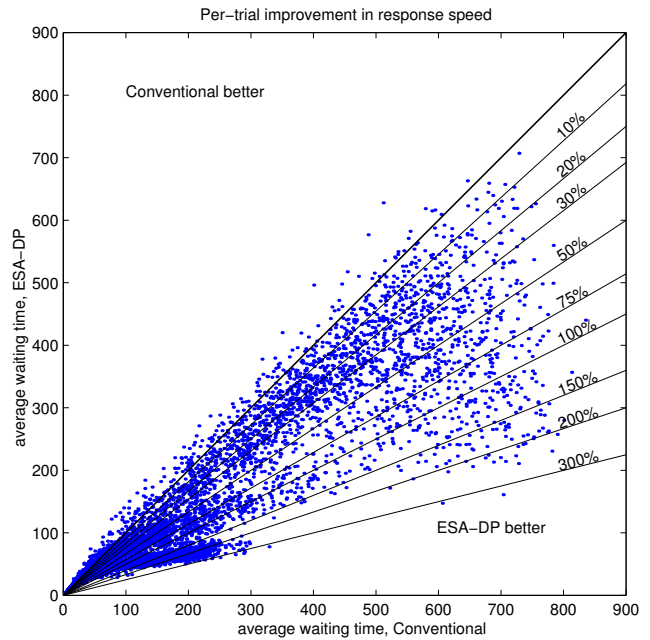


Figure 3: Waiting times of the ESA-DP scheduler plotted against waiting times of the conventional scheduler in identical scenarios, in seconds. Each dot represents an hour simulation in a different building type and arrival rate. Dots below the diagonal represent cases when ESA-DP achieves lower waiting time than the conventional scheduler, and vice versa for dots above the diagonal.

calls, but ignoring stops where those new passengers would be dropped off. We have been advised by industry experts that the benchmark scheduler method is considered to be highly competitive with the state of the art in currently deployed schedulers. The algorithm was tested on various buildings with height ranging from 8 to 30 floors, served by between 2 and 8 elevator shafts, whose cars were moving at a speed of 180 m/min. Each floor in these buildings was 4m tall, except for the lobby which was 5m tall.

The experiments explored the case of mixed up-peak and down-peak traffic which are some of the scenarios that pose extraordinary demands to the performance of the system. In addition, the up-peak case is accompanied by significant uncertainty in passenger destinations. Most (80%) of the traffic originated at the lobby and was directed approximately evenly to the upper floors, while the remaining 20% of the traffic was between floors other than the lobby. For the case of down-peak traffic, most of the traffic (80%) originated at the upper floors and directed to the lobby, 10% was directed from the upper floors, and the remaining 10% percent was inter-floor traffic not involving the lobby.

The performance of the two algorithms was tested under arrival rates ranging from 100 arrivals per hour up to the point where the elevator group was overwhelmed by the arrival stream and average waiting time exceeded two minutes. Such a point is reached at different rates for different

buildings and number of shafts in the elevator group. The scatter-plot in Fig. 3 plots the average waiting times of ESA-DP against those of the conventional scheduler in 20,000 hour-long trials in a detailed simulator. Each point represents the average wait time over 1 hour, with both schedulers fed identical arrival patterns. In general, the time savings resulting from the application of the ESA-DP algorithm, expressed as a percentage of the time of the conventional algorithm, also grow larger with increasing the arrival rate. At the rate, at which the elevator bank can be considered to be overwhelmed by the passenger flow, the savings are in the range of 30%-40%. This indicates that ESA-DP has a higher throughput and capacity, only saturating at much higher traffic rates.

The reduction in average waiting time and consequent improvement of system capacity can be quite dramatic, especially in tall buildings with relatively few shafts, where the system is heavily strained. The advantages of the ESA-DP algorithm fade away in buildings with (unrealistically) many shafts, partly because the system is never strained, and partly because of the conventional scheduler's zoning strategy. It should be noted that ESA-DP strongly dominates the conventional scheduler at all traffic rates in mixed and down-peak traffic. Successive papers will document elaborations of the ESA-DP approach that make it strongly dominant at all arrival rates in up-peak traffic too.

### Extensions of the Algorithm

Several of the underlying assumptions of the algorithm can be relaxed, which might result in better performance and easier installation on existing elevator banks. These are: completely observable system state, uniform probability distribution on passenger destinations, infinite capacity of the cars, and absence of future arrivals. Solutions for relaxing the first three assumptions are described below.

#### Partially-Observable System State

The ESA-DP algorithm assumes that the state of the system is completely observable, including the number of car and hall calls, and the number of people waiting per hall call. The exact number of hall and car calls is known because they are always registered by passengers by pressing car and hall buttons, but the exact number of people per hall call is not readily available.

There are two possibilities for dealing with this problem — one of them relies on technical devices, and the other one relies on statistical estimation techniques. The simplest possibility is to require each passengers to press individually a button in the desired direction, even when the button has already been pressed by a previous passenger. This would provide an accurate count of the number of passengers per hall call, but would most likely be abused by impatient passengers. The exact number of people waiting on a given floor can also be determined from sensing, e.g., a computer vision system observing the space in front of the elevator bank and counting the number of faces in the image. Such a solution is clearly within the current state of the art in computer vision (Kim & Moon 2001).

One may also estimate the expected number of arrivals at a floor since the hall button on that floor was first pressed. If the time elapsed since then is  $\Delta t$ , and the times between arrivals at this floor are i.i.d. exponentially distributed random variables with arrival rate  $\lambda$ , then the total number of new arrivals comes from a Poisson distribution whose mean is  $\lambda \cdot \Delta t$ . Hence, the expected number of passengers waiting at this floor is  $\lambda \cdot \Delta t + 1$ . Such estimates have been widely used by supervisory control algorithms, with minimal decrease in performance (Bao *et al.* 1994; Crites & Barto 1996). In order to apply this statistical estimation method, however, the arrival rates at each floor must be known. These might come either from online statistical estimates of the latest arrivals, or from known traffic profiles accumulated off-line from past data (Siikonen 1997).

#### Non-uniform Destination Probabilities Conditioned on Floor of Arrival

The assumption of a uniform probability distribution on passenger destinations will certainly be violated for most real buildings which usually have different number of occupants on each floor, so the traffic flow from the lobby to different floors cannot be assumed to be uniform. Furthermore, traffic between floors other than the lobby would usually be non-uniform as well, for example in the case when a single company is occupying adjacent floors in a building and there is a lot of traffic between these floors, but little or no traffic to and from other, unrelated floors.

As noted above, handling non-uniform destination probabilities which do not depend on the floor of arrival is straightforward and does not change the complexity of the algorithm — the only change necessary is in the probability used in the binomial formula. However, when destination probabilities are conditioned on the floor of arrival, the computational complexity changes drastically, due to the necessity to “remember” explicitly the state of each individual passenger (in or out of the car).

The favorable computational complexity of the original algorithm could be achieved because the scheduler could ignore the distinctions among the passengers within a car, and each state could be characterized simply by the number of people within the car. If the largest number of passengers that could be in the car at the same time was  $H$ , a total of  $H + 1$  states were sufficient to encode all distinctions necessary to quantify the future trajectory of the car, i.e. to make the chain fully Markovian. This was possible because individual passengers exited the car with the same probability, and the binomial formula could be used to aggregate multiple exits.

Conversely, the binomial formula cannot be used to compute transition probabilities when the respective probabilities of exiting the car vary among different passengers, because the disembarkation events for individual car passengers are not i.i.d. variables. One straightforward extension of the algorithm is to maintain individual Boolean state variables for each passenger, where each variable designates whether the passenger is inside the car or not. Let the overall state of the passengers inside the car be  $U = [u_1, u_2, \dots, u_H]$ , where each variable (fluent)  $u_i \doteq 1$  iff pas-

senger  $1 \leq i \leq H$  is in the car, and  $u_i \doteq 0$  otherwise. Furthermore, let  $p_{ij}$  be the probability that passenger  $i$  would get off at floor  $j$ , given that he/she is still in the car, and let  $q_{ij} \doteq 1 - p_{ij}$ . Then, the transition between state  $U$  and one of its successor states  $U' = [u'_1, u'_2, \dots, u'_H]$ ,  $u'_i \leq u_i$ ,  $1 \leq i \leq H$ , can be expressed as  $\prod_{i=1}^H (p_{ij}^{u_i - u'_i}) (q_{ij}^{1 - u_i + u'_i})$ .

While this extension is quite straightforward, both its computational time and storage space complexities are exponential in the number of passengers  $H$ . This is a frequent issue in many decision-theoretic planning problems, and many approaches for complexity reduction have been tried (Boutilier, Dearden, & Goldszmidt 1995). Significant computational leverage can be achieved by exploiting structure in the stochastic properties of the problem domain, and one primary manifestation of such structure is conditional independence between state fluents.

Such conditional independence is present in the elevator scheduling problem as well. The behavior of each individual passenger is independent of that of other passengers — decisions to get off or stay in the car are not influenced by other passengers. It can be expected that this structure can be exploited in the future to reduce the exponential complexity of the extended algorithm.

### Finite Car Capacity

When the number of waiting passengers assigned to a single car exceeds its physical limit, whether all of them can be transported within a single round trip of the car depends on their destinations. The basic version of the ESA-DP algorithm assumes that the car has infinite capacity, and all passengers would be transported in a single round trip — while this assumption simplifies computation, it would not correspond to reality in many cases<sup>2</sup>. Another way of handling the physical limits of the car is to divide the  $H + 1$  states within a single-velocity-group of the trellis into two subgroups: one subgroup of  $m + 1$  states, where  $m$  is the physical limit of the car, and another subgroup of  $H - m + 1$  states which correspond to the possible number of people left off as a result of a car overflow. The state propagation algorithm becomes more complicated, because now two components have to be propagated: one is the number of people inside the car, and the other one is the number of people left off.

A major difference from the basic case is that when computing the first component (number of people in the car), the number of people from existing car calls can no longer be ignored, since whether the car would pick up new hall-call passengers at a particular floor (and exactly how many) depends also on whether existing car-call passengers would get off at that floor to make room for new ones. The fact that all stops due to existing car calls are known in advance simplifies the computation, but still the exact number of car-call

<sup>2</sup>We are forced to make this unrealistic assumption in the current implementation because the elevator simulator automatically assigns to a car any new arrivals at floors where the car is already scheduled to stop, regardless of whether or not car has room to accommodate these passengers. The scheduler is not called and therefore does not get a chance to re-balance the load.

passengers getting off at a particular floor is not completely certain, when the number of such passengers exceeds the number of car buttons pressed. Even though the distribution on people getting off is completely defined, computing it is not trivial, because that distribution must also obey the constraints arising from existing car calls.

The propagation of the second component (the one encoding the possibility that between 0 and  $H - m$  passengers would be left off as a result of an overflow) can be implemented easily if one additional assumption is made: that once a passenger has been left off, he or she will remain unserved until the end of the round-trip. Under this assumption, the number of left-off passengers cannot decrease, and would either remain the same or increase as the state is propagated in time, depending on whether new overflows occur.

### Conclusions

This paper details an efficient scheduling algorithm based on dynamic programming for exact estimation and minimization of the expected waiting times of all known passengers in a group elevator system. Empirical comparison with a state-of-the-art scheduler in a very detailed discrete-event elevator bank simulator demonstrated that for a wide variety of buildings, ranging from 8 to 30 floors, and with 2 to 8 shafts, ESA-DP reduces waiting times by 30%-40% under very heavy traffic, and rarely under-performs the benchmark scheduler in light traffic.

The base ESA-DP algorithm was developed under the assumptions of no future passenger arrivals, unlimited car capacity, full state information, and a known marginal distribution over destination floors; even though the simulator does not respect most of these assumptions, the algorithm performs very well.

Most of these assumptions can be relaxed. We provided complete solutions for dealing with partial state information and handling non-uniform probability distributions on destination floors. The algorithm was also extended to deal with non-uniform probability distributions on destination floors conditioned on the arrival floor, although at a much higher computational cost. We also outlined how the algorithm could be extended to consider possible future passenger overflows due to finite car capacity. This may not be necessary because the base algorithm shows its greatest performance advantage at the heaviest traffic rates—precisely where we should expect to see it most punished for neglecting overflows.

The complexity of the base algorithm is linear<sup>3</sup> in both the number of cars and the number of existing hall calls, which allows it to be implemented on micro-controllers currently employed in existing elevators.

It is our hope that this paper will open the door to DP solutions to many other online scheduling problems. Forthcoming papers will describe extensions to ESA-DP that consider future arrivals, with significant further gains in performance.

<sup>3</sup>Run-time complexity is also linear in the total number of passengers, but in some cases the number of transitions between two slices of the trellis can be quadratic in the number of people in the car.

## Acknowledgments

We would like to thank Koichi Sasakawa and Masafumi Iwata from Sentan Soken, the Advanced Technology R&D Center of Mitsubishi Electric, for providing us with the detailed elevator simulator used in the experiments reported in this paper.

## References

- Bao, G.; Cassandras, C. G.; Djaferis, T. E.; Gandhi, A. D.; and Looze, D. P. 1994. Elevator dispatchers for down-peak traffic. Technical report, University of Massachusetts, Department of Electrical and Computer Engineering, Amherst, Massachusetts.
- Barney, G., and dos Santos, S. 1985. *Elevator Traffic Analysis, Design and Control*. England: IEE, Peter Peregrinus Ltd.
- Bertsekas, D. P. 2000. *Dynamic Programming and Optimal Control*. Belmont, Massachusetts: Athena Scientific. Volumes 1 and 2.
- Bittar, J. 1982. Relative system response elevator call assignments. *US Patent*. #4,363,381.
- Boutilier, C.; Dean, T.; and Hanks, S. 1999. Decision theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research* 11:1–94.
- Boutilier, C.; Dearden, R.; and Goldszmidt, M. 1995. Exploiting structure in policy construction. In *Extending Theories of Action: Formal Theory & Practical Applications: Papers from the 1995 AAAI Spring Symposium*, 33–38. AAAI Press, Menlo Park, California.
- Cho, Y.; Gagov, Z.; and Kwon, W. 1999. Elevator group control with accurate estimation of hall call waiting times. In *Proceedings of the 1999 International Conference on Robotics and Automation*, 447–452. Detroit, MI: IEEE.
- Crites, R. H., and Barto, A. G. 1996. Improving elevator performance using reinforcement learning. In Touretzky, D. S.; Mozer, M. C.; and Hasselmo, M. E., eds., *Advances in Neural Information Processing Systems*, volume 8, 1017–1023. The MIT Press.
- Hikihara, T., and Ueshima, S. 1997. Emergent synchronization in multi-elevator system and dispatching control. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences* E80-A(9):1548–1553.
- Kim, J.-H., and Moon, B.-R. 2001. Adaptive elevator group control with cameras. *IEEE Transactions on industrial electronics* 48(2):377–382.
- Koehler, J., and Ottiger, D. 2002. An AI-based approach to destination control in elevators. *AI Magazine* 23(3):59–79.
- Koehler, J., and Schuster, K. 2000. Elevator control as a planning problem. In Chien, S.; Kambhampati, S.; and Knoblock, C. A., eds., *Proceedings of the Fifth International Conference on AI Planning and Scheduling (AIPS)*, 1332–1339.
- Koehler, J. 2001. From theory to practice: AI planning for high performance elevator control. In Baader, F.; Brewka, G.; and Eiter, T., eds., *Lecture Notes in Computer Science*, volume 2174. Berlin: Springer Verlag. 459–462.
- Pepyne, D. L., and Cassandras, C. G. 1997. Optimal dispatching control for elevator systems during uppeak traffic. *IEEE transactions on control systems technology* 5(6):629–643.
- Powell, B. A., and Williams, J. N. 1992. Elevator dispatching based on remaining response time. *US Patent*. #5,146,053.
- Seckinger, B., and Koehler, J. 1999. Online-Synthese von Aufzugssteuerungen als Planungsproblem. In *13. Workshop Planen und Konfigurieren, Interner Bericht des Instituts für Informatik der Universität Würzburg*, 127–134.
- Siikonen, M.-L. 1997. Elevator group control with artificial intelligence. Technical Report A67, Helsinki University of Technology, Systems Analysis Laboratory, Helsinki, Finland.
- Strakosch, G. R. 1998. *Vertical transportation: elevators and escalators*. New York, NY: John Wiley & Sons, Inc.
- Tanahashi, T., and Araki, H. 1994. Drive-control equipment for 750 m/min elevators. *Mitsubishi Electric Advance* 67:8–9.
- Ujihara, H., and Amano, M. 1994. The latest elevator group-control system. *Mitsubishi Electric Advance* 67:10–12.
- Ujihara, H., and Tsuji, S. 1988. The revolutionary AI-2000 elevator group-control system and the new intelligent option series. *Mitsubishi Electric Advance* 45:5–8.