MITSUBISHI ELECTRIC RESEARCH LABORATORIES
http://www.merl.com

Exact Calculation of Expected Waiting Times for Group Elevator Control

Daniel Nikovski, Matthew Brand

TR2004-148 December 2004

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 controller based on this method significantly outperforms benchmark algorithms, and although slower than them, is completely within the computational capabilities of currently existing elevator bank controllers.

IEEE Transactions on Automatic Control

Final version submitted June 2004.

Exact Calculation of Expected Waiting Times for Group Elevator Control

Daniel Nikovski, Matthew Brand

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 controller based on this method significantly outperforms benchmark algorithms, and although slower than them, is completely within the computational capabilities of currently existing elevator bank controllers.**

Keywords— **optimal control, group elevator scheduling, dynamic programming, Markov chains**

Final submission to IEEE Transactions on Automatic Control regenerated July 14, 2005

I. Introduction

Group elevator scheduling is a hard optimal control problem which has been researched extensively due to its high practical significance [1]. 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 controller must assign a car to each hall call in a way that optimizes overall system performance.

The usual performance criterion to be optimized 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. Minimizing it is an extremely complicated problem, and one of the most important reasons for that is the uncertainty in destination floors — in order to compute the expected AWT of all passengers, the system must consider all possible ways of picking up these passengers by the respective cars assigned to them, which ways in turn depend on the unknown destination floors of passengers. Since the unknown destination of each passenger potentially influences the waiting time of all other passengers assigned to the same car, the scheduler must consider a potentially exponential number of possible paths of that car. Furthermore, future arrivals of new passengers will occur at unknown future times and floors, introducing additional waits to existing passengers that are very hard to quantify.

In order to make the problem tractable, existing algorithms simplify the problem significantly. The Empty-the-System Algorithm (ESA), closely related to the ETA algorithm, [1], [2], ignores all future arrivals, and selects assignments that would serve the existing passengers (thus emptying the system) in the shortest possible time. When, inevitably, a new passenger does arrive in the future, the current schedule can either be extended with the car that will be assigned to serve that passenger, or the whole schedule can be recomputed. The former procedure is the only option in Japan, where assignments are expected to be announced immediately after receiving new hall calls and cannot be revoked later, while the latter procedure is more typical of western countries, where pre-announcements are not expected, and the scheduler has more latitude to balance the schedule as long as possible.

Both scheduling modes, though, are reduced to the problem of estimating the expected AWT for a given schedule. As pointed out above, one of the most significant difficulties in computing AWT is the uncertainty in destination floors and the resulting combinatorial explosion of possible paths to be considered. The usual simplification employed to deal with this difficulty in ESA/ETA is to pretend that the destination floor of each passenger is known, which entails a single deterministic path for all cars serving their respective passengers. Computing the AWT in this case can be done in linear time. Different heuristics are possible: the ESA algorithm assumes that the destination floor of all passengers going down is the lobby, while ETA assumes that the destination floor of passengers lies halfway between their boarding floor and

the end of the building in their desired direction of movement. Clearly, these assumptions are not very reasonable, and have the effect that waiting times are computed on a single path that is not even very likely. In the next section, we demonstrate that such a simplification is not necessary, and describe an algorithm that can compute the proper expectation of AWT over all possible paths of an elevator car.

II. Dynamic Programming for Exact Computation of Expected Average Waiting Times

A. 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) signalling 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

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

The car c chosen by the controller 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^+ we temporarily add the new passenger to elevator i's itinerary and recompute the expected RWT.

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. 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 control algorithms where cost estimates on segments of a system's path can be reused in multiple paths [3]. 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.

B. Trajectory Structure of an Elevator Car

Such branching points can readily be identified on the phase-space diagram of an elevator car shown in Figure 1. Like any moving mechanical system, a car travelling 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 parabolae, 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 Figure 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.

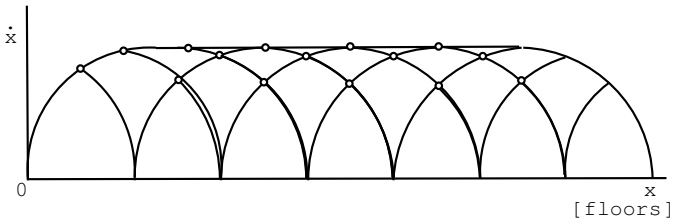

Fig. 1. A schematic illustration of the phase space of a single car moving upwards in a shaft of a building with eight floors, not all of which have equal height. All branching points are denoted by circles.

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 specifies the probability that the system would start in state S_i [3].

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 signalled 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.

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 of passengers assigned to a car, travelling 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 Figure 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 would 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 encountering since its last stop.

There are several ways to discretize the floor vari-

able 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.

C. Structure and Parameters of the Embedded Markov Chain

Figure 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 controller 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.

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). This organization of states constitutes the trellis of the dynamic programming problem. It can

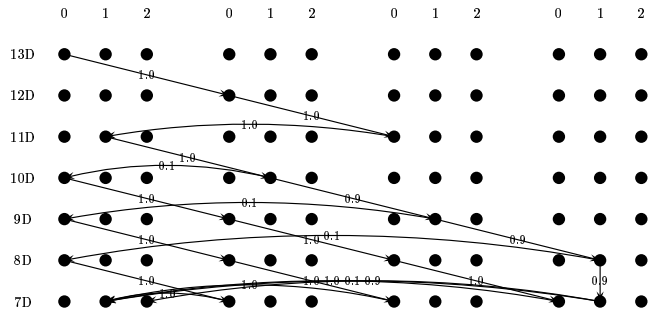

Fig. 2. Simplified trellis structure for the embedded Markov chain of a single descending car. Rows signify floors; columns signify the number of recently boarded passengers; column groups signify elevator speeds. The empty descending car is about to reach the branching point for possible a stop at floor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

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 Figure 2 is determinis-

tic — 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:

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

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. 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).

D. Building and Evaluating the Markov Chain

The Markov chain can be constructed easily by propagating forward in time the set of possible states, starting with the initial state. For each possible state in turn, the possible successor states are computed and marked as such, and the probabilities of transitions to these state are recorded. Evaluation of the Markov chain proceeds in the opposite order, starting from the terminal state(s), and updating the cost-to-go (RWT) of each possible state, until the initial state is reached. The implementational details of these two algorithms, as well as an experimental comparison with conventional control, are given in [4]. Note that this procedure computes the expected RWT 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 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.

REFERENCES

[1] Gang Bao, Christos G. Cassandras, Theodore E. Djaferis, Asif D. Gandhi, and Douglas P. Looze, "Elevator dispatchers for down-peak traffic," Technical report, University of Massachusetts, Department of Electrical and Computer Engineering, Amherst, Massachusetts, 1994.

[2] G.C. Barney, *Elevator Traffic Handbook*, Spon Press, London, 2003.

[3] Dimitri P. Bertsekas, *Dynamic Programming and Optimal Control*, Athena Scientific, Belmont, Massachusetts, 2000, Volumes 1 and 2.

[4] Daniel Nikovski and Matthew Brand, "Decision-theoretic group elevator scheduling," Technical Report TR2003-61, Mitsubishi Electric Research Laboratories, Cambridge, Massachusetts, http://www.merl.com/papers/TR2003-61, 2003.

Daniel Nikovski received the PhD degree in robotics from Carnegie Mellon University in 2002, and is a research scientist at Mitsubishi Electric Research Laboratories in Cambridge, MA.

Matthew Brand received the PhD degree in computer science from Northwestern University in 1994, and is a senior research scientist at Mitsubishi Electric Research Laboratories in Cambridge, MA.