

MITSUBISHI ELECTRIC RESEARCH LABORATORIES

<http://www.merl.com>

## Optimal Parking in Group Elevator Control

Matthew Brand

Daniel Nikovski

TR2004-44 May 2004

### Abstract

We consider the problem of optimally parking empty cars in an elevator group so as to anticipate and intercept the arrival of new passengers and minimize their waiting times. Two solutions are proposed, for the down-peak and up-peak traffic patterns. We demonstrate that matching the distribution of free cars to the arrival distribution of passengers is sufficient to produce savings of up to 80 % in down-peak traffic. Since this approach is not useful for the much harder case of up-peak traffic, we propose a solution based on the representation of the elevator system as a Markov decision process (MDP) model with relatively few aggregated states, and determination of the optimal parking policy by means of dynamic programming on the MDP model.

*This paper has been presented at the International Conference on Robotics and Automation ICRA'04, April 28 – May 1, 2004, New Orleans, USA.*

This work may not be copied or reproduced in whole or in part for any commercial purpose. Permission to copy in whole or in part without payment of fee is granted for nonprofit educational and research purposes provided that all such whole or partial copies include the following: a notice that such copying is by permission of Mitsubishi Electric Research Laboratories, Inc.; an acknowledgment of the authors and individual contributions to the work; and all applicable portions of the copyright notice. Copying, reproduction, or republishing for any other purpose shall require a license with payment of fee to Mitsubishi Electric Research Laboratories, Inc. All rights reserved.

Copyright © Mitsubishi Electric Research Laboratories, Inc., 2004  
201 Broadway, Cambridge, Massachusetts 02139

Submitted May 2004.

# Optimal Parking in Group Elevator Control

Matthew Brand, Daniel Nikovski  
Mitsubishi Electric Research Laboratories,  
201 Broadway, Cambridge, MA 02139, USA.  
E-mail: {nikovski,brand}@merl.com

**Abstract**— We consider the problem of optimally parking empty cars in an elevator group so as to anticipate and intercept the arrival of new passengers and minimize their waiting times. Two solutions are proposed, for the down-peak and up-peak traffic patterns. We demonstrate that matching the distribution of free cars to the arrival distribution of passengers is sufficient to produce savings of up to 80% in down-peak traffic. Since this approach is not useful for the much harder case of up-peak traffic, we propose a solution based on the representation of the elevator system as a Markov decision process (MDP) model with relatively few aggregated states, and determination of the optimal parking policy by means of dynamic programming on the MDP model.

## I. INTRODUCTION

The passenger traffic pattern in modern buildings with multiple elevator shafts varies considerably throughout a typical business day: early in the morning, most of the passengers travel from the lobby to the upper floors, while at the end of the day, most of them leave the floors and travel primarily to the lobby in order to exit the building. These two traffic patterns, known as up-peak and down-peak regimes, respectively, have very irregular distribution over origin and destination of travel, which has the effect of leaving cars after completion of service at floors far from where they would be needed next. At the same time, these patterns have a specific probabilistic structure which can be exploited by a car scheduling algorithm.

Parking of free cars has been identified as an important part of elevator group supervisory control algorithms [1]. In this paper, we consider the problem of optimally dispatching empty cars to desired parking locations in a manner that minimizes the usual optimization criterion in group scheduling algorithms: the waiting time of future hall calls [2].

A parking algorithm could actively move empty cars so as to anticipate and intercept future hall calls—long before these calls actually occur. Reasoning about events far into the future and deciding on a suitable course of action can be handled by a planning sub-system that decides upon new parking locations for all free cars, as soon as their number changes, and a corresponding plan-execution sub-system that brings the free cars to these parking locations.

The idea of actively moving empty cars with the explicit purpose of parking them favorably with respect to future hall calls is present in several studies on optimal group elevator scheduling. One possibility is to use the statistical properties of the traffic pattern in order to dispatch cars to the floors where they would be most needed. In the case of pure up-peak traffic, it is clear that a car that has delivered all of its passengers should be dispatched immediately back to the lobby

for the next batch of passengers. This insight has been used in the provably optimal solution for pure up-peak traffic proposed by Pepyne and Cassandras [3]. Although pure up-peak traffic rarely exists in practice and there is almost always additional inter-floor traffic, active parking of cars is an important element of most industrial elevator control systems [1].

Our strategy is to re-park all available cars (i.e., the ones that are not currently serving passengers) as soon as their number changes, due to one of the following two events: either a car becomes free (empty) and available for service, or a car is assigned to service a new hall call and is no longer empty. We assume that the optimal parking locations for a fixed number of free cars when the arrival rates are fixed remains the same, i.e., we ignore the effects the remaining (busy) cars might have on the optimal parking locations of the free ones. Under this assumption, planning is reduced to the computation of a policy that maps sets of free cars of various cardinality to the corresponding parking locations. In other words, the planning system computes an universal plan [4] covering all possible numbers of free cars, and uses it as long as the traffic flow in the building remains relatively constant; as soon as the stochastic properties of the traffic change, a new universal plan (policy) is computed.

## II. PARKING POLICIES AND THEIR EXECUTION

We are considering a building of  $F$  floors equipped with  $N_c$  identical elevator cars. At any particular moment,  $C$  of these cars are free, i.e., have no hall or car calls assigned to them ( $0 \leq C \leq N_c$ ). When a new hall call is signaled, a scheduling algorithm assigns it to one of the  $N_c$  cars; according to the rules accepted in Japan, this assignment is never revoked later. As a result, the number  $C$  of empty cars will either decrease (if the new hall call is assigned to a free car) or remain the same (if the new hall call is assigned to a busy car).  $C$  increases when a car completes servicing all hall and car calls assigned to it, and becomes free. A parking decision is made and executed in this case too, in an identical manner.

We assume that the parking locations always coincide with one of the landings, i.e., a car is never parked between two adjacent landings (a formal proof that such parking positions are optimal in down-peak traffic is presented in [5]). Under this assumption, a parking policy is a mapping between the number of empty cars  $C$  and a vector  $\mathbf{x}$  of  $C$  parking locations  $x_i$ ,  $i = 1..C$ , such that  $1 \leq x_i \leq F$ . Thus, the number of possible policies (mappings) is  $F^C$ . Since some of these policies are identical up to a symmetry, we choose a canonical representation for a

policy such that  $x_i \geq x_j$  when  $i > j$ . Even after accounting for such symmetries, it is clear that the number of possible policies is very large.

When a parking decision has to be made, the free cars can be either already parked at a floor, or moving between floors in the process of executing a previous parking decision. We denote by  $y_i$ ,  $i = 1..C$  the floors where each car  $i$  is at that moment. If car  $i$  is not moving,  $y_i$  is simply the floor where the car is located; when car  $i$  is moving,  $y_i$  is the first floor where it can stop and possibly reverse its direction. (We assume that a car cannot reverse its direction between landings, even though such a possibility is likely to increase the responsiveness and efficiency of the parking algorithm, if it were allowed.)

Once the locations of the cars  $\mathbf{y} = [y_1, y_2, \dots, y_C]$  are known and the desired parking positions  $\mathbf{x}$  have been determined, a parking plan has to be devised and executed by a parking subsystem. The objective of this plan is to move the cars from their current positions  $\mathbf{y}$  to the desired parking floors  $\mathbf{x}$  as quickly as possible. Thus, the parking subsystem has to decide which of the cars should go to each of the parking locations. Since there are  $O(C!)$  possible matches between the  $C$  parking positions and the  $C$  cars, finding the optimal plan (matching) is not a trivial problem.

However, there exists a simple heuristic which allows the parking decision to be executed efficiently and within a short time: Preserve the vertical ordering of the cars. This heuristic can be implemented by first sorting the locations  $y_i$ ,  $i = 1..C$  of all cars in ascending order, while simultaneously sorting the ordinal numbers  $\mathbf{k} = [1, 2, \dots, C]$  of the cars in accordance with the sorting of  $y_i$ . Before sorting, the array of ordinal numbers is initialized so that  $k_i = i$ ,  $i = 1..C$ . For example, if initially  $\mathbf{y} = [5, 3, 8, 1]$  and  $\mathbf{k} = [1, 2, 3, 4]$ , after sorting we obtain  $\mathbf{y} = [1, 3, 5, 8]$  and  $\mathbf{k} = [4, 2, 1, 3]$ . Since the policy  $\mathbf{x}$  is already in canonical form, we can simply dispatch car  $k_i$  to location  $x_i$  for each  $i = 1..C$ . Continuing the example from above, if the policy is  $\mathbf{x} = [2, 4, 6, 8]$ , the controller would dispatch car 4 to the second floor, car 2 to the fourth floor, car 1 to the sixth floor, and car 3 to the eighth floor. This parking decision is very efficient: cars 1, 2, and 4 would have to move only by one floor, and car 3 wouldn't have to move at all, because it is already at the eighth floor (or about to stop there).

We now return to the problem of finding the optimal parking locations  $\mathbf{x}$  given a particular traffic pattern, building height, number of shafts, and speed and acceleration of the elevator cars. (It might be expected that the optimal parking location is also dependent on the current position and velocity of all cars, but we ignore this dependency in order to make the problem tractable.)

Our general strategy in the two cases described below (down-peak traffic and up-peak traffic) is to first analyze how the passenger flow influences the final positions of the cars when they become free, then identify inefficiencies resulting from uneven distribution of free cars, and finally decide how the cars should be re-parked so that the responsiveness of the system to new hall calls could be improved. The case of down-peak traffic

could be solved by optimization over an immediate horizon (only the next arrival), while the much harder case of up-peak traffic requires reasoning and planning over a sequence of events (arrivals).

### III. PARKING IN DOWN-PEAK TRAFFIC

Under down-peak traffic, most of the passengers depart from upper floors and are delivered to the lobby. As a result, when a car becomes free, it is usually located at the lobby. If such cars are left where they delivered the last passenger (the lobby), they would be far away from the locations where new calls are likely to originate (the upper floors). In order to amend this mismatch between where the cars are and where they would be needed the most, empty cars can be moved from the lobby to the upper floors as soon as they deliver the last passenger.

In order to make the problem tractable, for the case of down-peak traffic we choose to optimize the waiting time of only the first future hall call. (This approach, however, is not appropriate for up-peak traffic and will be extended in the next section.) Furthermore, we make the assumption that the first hall call would be served by one of the free cars, rather than one of the cars that already have passengers assigned to them, and we also assume that the new call would occur only after the desired parking location of the cars has been attained. We can define the expected waiting time of the first future call as a function of the state of free cars  $\mathbf{x}$  only:  $Q(\mathbf{x}) = \sum_{f=1}^F p_f \min_i T(x_i, f)$ , where  $p_f$  is the probability that the next passenger would arrive at floor  $f$ , and  $T(f_1, f_2)$  is the time it would take for a car parked at floor  $f_1$  to serve a call originating at floor  $f_2$ .

The number of possible placements of  $C$  cars at  $F$  landings is  $F^C$ , which is exponential in  $F$  and exhaustive computation of  $Q(\mathbf{x})$  for all possible  $\mathbf{x}$  is very expensive. However, intuition suggests that the optimal parking position would be the one that parks the cars as evenly as possible with respect to the arrival distribution of passengers. An even distribution of cars with respect to that probability would be one that positions the  $C$  available cars so that their respective probabilities of serving the next call would be equal ( $1/C$ ).

One approximate way to achieve this is to split the building into  $C$  zones, each served by one of the  $C$  cars. Given an array of cumulative arrival probabilities  $P_j$ ,  $j = 1..F$ , such that  $P_j = \sum_{i=1}^j p_i$ , this parking policy can be computed by Algorithm 1. This solution would be optimal with respect to the minimization criterion only when the average time to serve a call is the same for each zone. In practice, however, this time would be higher for larger zones, so a correction is necessary, in direction of shrinking relatively larger zones so that they cover arrivals with probability lower than  $1/C$ . A simple greedy algorithm could be employed to improve the solution, if necessary, although we have found that improvement was never necessary in practical down-peak regimes [5].

In order to analyze the active parking of cars so that their parking locations match the distribution of the incoming traffic, we performed experiments in down-peak traffic, where 80% of the traffic originated at the upper floors with destination the lobby, 10% originated at the lobby with destination the upper

---

**Algorithm 1**  $x[1..C]=\text{STATIONARYPOLICY}(C,F,P[1..F])$ 

---

```
1:  $d := 2C$ 
2:  $j := 1$ 
3: for  $i := 1$  to  $C$  do
4:    $k := 2i - 1$ 
5:   while  $P[j]d < k$  do
6:      $j := j + 1$ 
7:   end while
8:    $x[i] := j$ 
9: end for
```

---

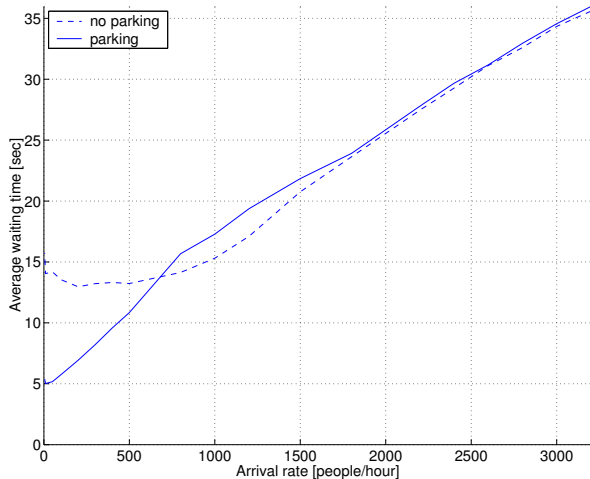


Fig. 1. Effect of parking in down-peak traffic in a building of 20 floors and 6 elevator shafts. For low and medium rates, savings in waiting time can reach 60% for this building; however, as the arrival intensity increases, parking becomes counter-productive and has to be turned off.

floors, and the remaining 10% was traffic among the upper floors only. The arrival rates at the upper floors were uniform, i.e.  $p_f = 0.9/(F - 1)$ . Under this condition, the optimal parking policy for  $C$  free cars is the even splitting of the building into  $C$  zones, with cars parked at the center of each zone. The parking positions were pre-computed for each possible number of free cars  $0 \leq C \leq N_c$ , and parking policies executed as described above.

Active parking was compared to the case when no parking was performed and free cars were left at the floor where the last passenger served by them was delivered. In both cases, we used a scheduling algorithm based on dynamic programming, described in a separate report [6]. The algorithm was tested on three buildings with heights of 10, 20, or 30 floors, served by either 3, 6, or 8 elevator shafts, whose cars were moving at a speed of 3m/s. Each floor in these buildings was 4m tall, except for the lobby, which was 5m tall.

The performance of the two algorithms was tested under arrival rates ranging from 5 arrivals per hour up to the point where the elevator group was overwhelmed by the arrival stream and average waiting time exceeded two minutes. Fifty runs were performed for each arrival rate to ensure statistical significance of the results.

A typical graph for the waiting time of the parking algorithm

is shown in Fig. 1. It shows that actively parking the free cars so that they are equally distributed along the height of the building is very beneficial at low arrival rates, sometimes resulting in savings in waiting time of more than 60%, but as the arrival rate increases, this strategy becomes counter-productive. One likely explanation is that there are very few free cars at such rates, and keeping them in motion causes arriving passengers at all floors to wait, while if a car is simply left parked where it last delivered a passenger, it could at least pick up a new passenger at that floor without delay. This “point of diminishing returns” can be estimated from the number of free cars, building height, and arrival rates by comparing the expected times to pick up the next (unknown) passenger arrival with and without re-positioning. However, it is both easier and more precise to estimate it empirically from direct simulation.

#### IV. PARKING IN UP-PEAK TRAFFIC

The parking solution based on matching the arrival distribution of passengers to the parking location of the cars, while successful for down-peak traffic, is not sufficient for up-peak traffic. The reason for this is the very uneven distribution of arrival rates — the majority of passengers arrive at the lobby, and most of the waiting time is generated by such passengers. Hence, it is of primary importance to reduce the waiting time at the lobby under this type of traffic. However, parking free cars with respect to *only* such passengers is not very efficient either — sending each and every car to the lobby immediately after it becomes free leaves the rest of the building uncovered, and the waiting times of passengers arriving at the upper floors start to dominate the overall average waiting time.

It is clear that if  $C$  free cars are available for service, some proportion of them should be sent to the lobby, while the remaining ones should be parked at the upper floors, again distributed evenly with respect to the arrival rates there. The question then becomes how to determine this proportion.

In order to find the correct proportion between cars parked at the lobby and cars parked at the upper floors, we formulate the decision problem as a Markov decision process (MDP). An MDP consists of a finite number of states  $S_i$ ,  $i = 1, N_s$ , a set of actions  $A_k$ ,  $k = 1, N_a$ , an immediate cost  $w_{ijk}$  of the transition between each pair of states  $S_i$  and  $S_j$  under action  $A_k$ , a matrix  $P_{ijk}$  of the probabilities of transition between states  $S_i$  and  $S_j$  under action  $A_k$ , and a distribution  $\pi(S_i)$  which specifies the probability that the system would start in state  $S_i$  [7].

The simple optimization criterion used for down-peak traffic — the immediate cost (waiting time)  $Q(\mathbf{x})$  of only the next arrival — is not adequate for the case of up-peak traffic. If only  $Q(\mathbf{x})$  were minimized, the computed optimal number of cars at the lobby would always be only one, since one car is sufficient to answer a potential single call at the lobby, and the remaining cars would be better used at the upper floors in order to minimize the waiting times of arrivals there. However, this parking policy is not efficient for up-peak traffic, where the first arrival at the lobby would use the single car parked there, and the lobby would remain uncovered for future calls there before any of the remaining cars could reach it.

A much more appropriate optimization criterion for this traffic pattern is the average waiting time over a longer time horizon (preferably infinitely long). In this case, it is more convenient to express the optimization criterion as the average waiting time over a sequence of  $N$  future arrivals:  $\bar{W}_N = 1/N < \sum_{i=1}^N Q(s_i) >$ , where  $s_i$  is the state of the elevator bank when call  $i$  occurs,  $Q(s_i)$  is the expected waiting time of passenger  $i$ , and the expectation  $\langle \rangle$  is taken with respect to the distribution of the next  $N$  arrivals. The true long-term average waiting time of passengers, which is the exact criterion we would like to optimize, is the limit of  $\bar{W}_N$  as  $N$  becomes infinitely large (i.e., the horizon becomes infinitely long):  $\bar{W}_\infty = \lim_{N \rightarrow \infty} \bar{W}_N$ .

Directly minimizing this optimization criterion is very hard, because the space of possible states of the system  $s$  is very large, and taking expectations with respect to all possible future arrivals is computationally very expensive. In order to formulate the optimization of this criterion in terms of long-term cost on an MDP with relatively few states, our strategy is to consider only a small number of all possible states of the system, and simplify the probabilistic structure of the evolution of these states as a result of choosing different parking policies.

The key to reducing the number of states in the MDP is the insight that a particular parking policy introduces a set of “attractor” states that the system converges to in the absence of passenger arrivals and cars completing service. These states are exactly the parking positions specified by the parking policy, and it is them that we choose to use as states of an aggregated MDP. However, since the system does not jump between such states instantly, but rather moves smoothly between them, we define the system to be in a particular state represented by a parking position not only when the system has assumed that position, but also when it is in the process of moving towards it.

In order to further reduce the number of states, we will assume that a parking position for the case of up-peak traffic is specified by the pair of numbers  $(L, U)$ , where  $L$  is the number of cars parked at the lobby, while  $U$  is the number of cars parked at the upper floors. We further make the assumption that the  $U$  cars are spaced evenly along the height of the building. Thus, once the pair  $(L, U)$  is given and the height of the building is known, the corresponding detailed parking position  $\mathbf{x}$  can be generated by parking  $L$  cars at the lobby and distributing the remaining  $U$  cars along the height of the building. As a consequence, we can define the immediate cost  $Q(L, U)$  of a state (position)  $(L, U)$  as the corresponding immediate cost (waiting time) of the complete position  $\mathbf{x}$ :  $Q(L, U) \doteq Q(\mathbf{x})$ .

Under the proposed notation for parking states, the decision that has to be made when  $C$  free cars are available is how many of them should be sent to the lobby ( $L$ ), and how many should be sent to the upper floors ( $U = C - L$ ). For example, if there are three free cars available, the possible decisions are  $(0, 3)$ ,  $(1, 2)$ ,  $(2, 1)$ , and  $(3, 0)$ . One very compact representation of such a policy is the  $N_c$ -dimensional vector of values  $L_C$ ,  $C = 1..N_c$ , whose  $C$ -th element specifies how many cars should be parked at the lobby when  $C$  of all cars are free.

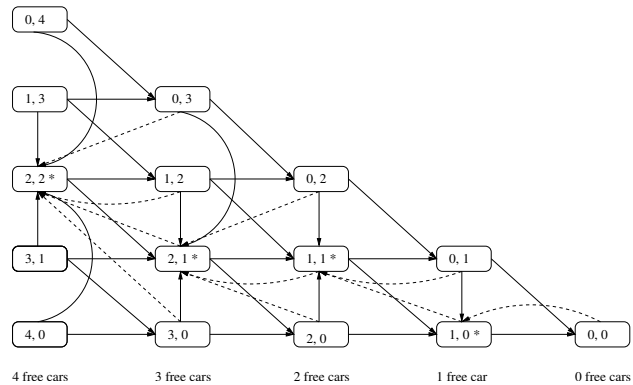


Fig. 2. Trellis structure for the embedded Markov decision process of a building with four shafts. The states in the same column represent different choices for allocating the same number of free cars into ones parked at the lobby, and others parked at the upper floors. Solid lines correspond to transitions resulting from passenger arrivals, while dashed lines correspond to cars returning from service and joining the set of free cars. One starred\* attractor state is identified as the optimal positioning for each quantity of free cars.

In a building with  $N_c$  shafts, the number of possible policies is  $N_c!$ , which makes it impractical to experimentally compare them and select the best overall parking policy. Such an experimental selection is further complicated by the stochastic nature of the arrival process. In order to meaningfully compare the statistical performance of two or more policies, they would have to be executed over many possible scenarios (sequences of passenger arrivals), which is an added factor to the computational burden of an already exponential computation. Instead, we employ dynamic programming on an MDP model describing the probabilistic structure of the state evolution of the system. As noted, the states in this model are aggregated “attractor” states corresponding to pairs (positions)  $(L_C, U_C)$  such that  $L_C + U_C = C$ ,  $C = 1..N_c$ . There are  $(N_c + 2)(N_c + 1)/2$  such states for a building with  $N_c$  shafts.

We organize the states in a regular structure known as trellis in dynamic programming problems, and specify the probabilities of transitioning between such states as a function of a particular parking policy. Fig. 2 shows the organization of 15 states for a building with 4 shafts in a trellis, along with the transition structure for one particular policy,  $[1, 1, 2, 2]$ . Each state in the trellis is labeled by two numbers, the first of which is  $L$ , and the second  $U$ . The two numbers for states in the same column of the trellis add up to the same number of free cars  $C$ , and thus such states correspond to the possible parking decisions when  $C$  cars are available. States in the same row have the same number of cars parked at the upper floors of the building, regardless of the number of free cars available. The state  $(0, 0)$  is present in the trellis as well, even though there is no decision to be made in this case, since there are no free cars to park.

The chosen parking policy determines the transitions that the MDP model would follow under the influence of the passenger traffic and the operation of the call scheduling algorithm (which works independently of the parking policy and can be arbitrary).

Solid lines depict transitions due to arrival of new passengers — such arrivals reduce the number of free cars, and the transitions are from left to right. The dashed lines depict transitions corresponding to cars returning from service — such events increase the number of free cars, and the transitions are from right to left. Finally, there are transitions between states within the same column — they exist because only one state within a column is stable, and when the cars end up in any of the other states in that column, the parking subsystem would start moving the cars towards the parking location. We would call such transient states *sliding states*.

In theory, if all probabilities of the model were given (i.e., the transition probabilities for all policies, not only for the one shown in Fig. 2), it should be possible to use policy iteration or value iteration [7] in order to determine efficiently the policy that minimizes directly the criterion stated above — the average waiting times of all passengers over an infinitely long horizon. In practice, finding the probabilities of cars returning from service (shown in dashed lines in Fig. 2) proved to be very hard. However, there is still a way to use only the left-to-right transitions (shown in solid lines in Fig. 2) for determining a suitable policy, if we amend slightly the criterion to be minimized. Instead of minimizing the *average* waiting time over an *infinitely long* horizon, we can choose to minimize the *cumulative* waiting time for only the next  $C$  hall calls for all states  $(L, U)$  such that  $L + U = C$ . While this results in minimizing different criteria for the states in different columns of the trellis, this is not a problem, because the selection of a parking state is performed only among states within the same column, whose optimization criterion is the same. We define the optimization criterion for state  $s_0$  located in column  $C$  simply as  $W_C(s_0) = \langle \sum_{i=1}^C Q(s_i) \rangle$ , where the expectation  $\langle \rangle$  is with respect to all possible trajectories of the system for the next  $C$  arrivals, and  $s_i$  is the state of the system when the  $i$ -th call occurs. The advantage of using this minimization criterion is that a recursive definition exists between  $W_C(s)$  and  $W_{C-1}(s')$ , where  $W_{C-1}(s')$  is the cumulative waiting time of the states  $s'$  in the next column in the trellis (the one to the right). In order to see this dependency, consider what would happen if the system is in state  $s = (L, U)$ , such that  $L + U = C$ , and a new hall call occurs. Since we are trying to determine whether  $s$  should be chosen as the parking state when  $C$  cars are available,  $s$  is a stable state under this assumption and the cars are at rest, awaiting the next hall at their parking positions.

The next hall call occurs at one of the floors, according to the arrival distribution of the building. This call incurs immediate cost of  $Q(L, U)$ , as defined above, and moves the system to a state in the next column to the right, where one less free cars are available. Depending on exactly where the call occurs, two things can happen: either a parked lobby car is dispatched to serve it, with probability  $P_l$ , or a car parked at the upper floors is used, with probability  $P_u = 1 - P_l$ . These probabilities are easy to compute, if the arrival distribution of passengers is specified in advance. These two events give rise to two transitions out of  $s$  to the right column: in Fig. 2, the transition with probability

$P_l$  leads to the state in the same row as  $s$ , and the transition with probability  $P_u$  leads to the state one row below that of  $s$ . Using these two probabilities, we can decompose  $W_C(s)$  as

$$W_C(L, U) = Q(L, U) + P_l W'_{C-1}(L-1, U) + P_u W'_{C-1}(L, U-1),$$

where  $W'_{C-1}(l, u)$  is the additional cost of the next  $C-1$  calls if the first of them occurs when  $l$  cars are parked at the lobby and  $u$  cars are parked at the upper floors. Note that, in general,  $W'_{C-1}(l, u) \neq W_{C-1}(l, u)$ , because  $W_{C-1}(l, u)$  is the expected cumulative cost starting from an ideally parked position for the  $C-1$  cars, while  $W'_{C-1}(l, u)$  is the expected cumulative cost of the  $C-1$  cars right after a car went into service, and the remaining  $C-1$  cars are *not* parked yet.

After both transitions, the further cost  $W'_{C-1}$  incurred by the system over the next  $C-1$  calls depends on whether the transition was to the optimal state in the next column to the right, or to a sliding state that would begin immediately transitioning to the optimal one. The difference between these two cases arises from the fact that if the transition was to the optimal state, the cars would not move before the next call, since they are already parked optimally, and the cost of answering the next call would *not* depend on exactly when it occurs. On the contrary, if the transition is to a sliding state, the cost of answering the next call depends strongly on exactly when it occurs — this cost is higher, if it occurs early and the cars are not parked optimally yet, and lower, if they have already assumed their optimal parking position.

We now return to the relationship between the additional cost  $W'_{C-1}(l, u)$  of serving  $C-1$  calls if the system was left with  $l+u$  cars, which are not optimally parked yet, and the estimates  $W_{C-1}(L, U)$  of the states in column  $C$ , each computed under the assumption that  $(L, U)$  is the optimal parking state. This relationship is straightforward if  $(l, u)$  is indeed the truly optimal parking state — in this case,  $W'_{C-1}(l, u) = W_{C-1}(l, u)$ . In all other cases,  $W_{C-1}(L^*, U^*) \leq W'_{C-1}(l, u) \leq W_{C-1}(l, u)$ , where  $(L^*, U^*)$  is the optimal state for  $C-1$  cars. This is so, because  $W_{C-1}(L^*, U^*) \leq W_{C-1}(l, u)$  for all states  $(l, u) \neq (L^*, U^*)$ , by virtue of  $(L^*, U^*)$  being the optimal parking state. In effect, the system initially starts in a non-optimal state  $(l, u)$  with cumulative cost-to-go  $W_{C-1}(l, u)$  higher than that of the optimal state  $(L^*, U^*)$ , and proceeds toward that optimal state.

The expected cumulative cost-to-go of the non-optimal state can be computed by quantifying this process of movement towards the optimal state, after imposing some assumptions. While sliding from the non-optimal state  $(l, u)$  towards the optimal state  $(L^*, U^*)$ , the system would go through a series of intermediate states. We can define the cost-to-go for each of them as  $w(t) \doteq W_{C-1}[s(t)]$ , with  $s(t)$  sliding from  $(l, u)$  to  $(L^*, U^*)$  such that  $s(0) = (l, u)$ . Let the time necessary to complete the parking move be  $T$ , i.e.,  $s(T) = (L^*, U^*)$ . In this case,  $w(t) = W_{C-1}(L^*, U^*)$  for  $t \geq T$ , and it is also true that  $w(0) = W_{C-1}(l, u)$ .

We assume that the arrivals of passengers to the elevator group are exponentially distributed with parameter  $\lambda$ , i.e. the probability density on the time  $t$  until the next arrival is

$P(t|\lambda) = \lambda e^{-\lambda t}$ ,  $t \geq 0$ . The expected cost-to-go  $W'_{C-1}(l, u)$  of the system sliding from  $(l, u)$  towards the optimal state  $(L^*, U^*)$  with respect to the distribution of the next arrival then is

$$W'_{C-1}(l, u) = \int_0^\infty P(t|\lambda)w(t)dt = \int_0^\infty \lambda e^{-\lambda t}w(t)dt.$$

In order to compute this integral, we assume that  $w(t)$  decreases linearly from  $w_0 = w(0) = W_{C-1}(l, u)$  to  $w_T = w(T) = W_{C-1}(L^*, U^*)$  over the interval  $(0, T)$ :  $w(t) = w_T + (w_0 - w_T)(T - t)/T$ ,  $0 < t < T$ . Under the chosen approximation of  $w(t)$  for the interval  $0 < t < T$ , the expected cost-to-go  $W'_{C-1}(l, u)$  with respect to the time of the next arrival can be computed by splitting the integral above over two intervals:

$$W'_{C-1}(l, u) = w_0 - \frac{(w_0 - w_T)(1 - e^{-\lambda T})}{\lambda T}.$$

The quantities  $w_0$  and  $w_T$  already incorporate in them the expectation over the *location* of the next arrival and the locations *and* times of the next  $C - 2$  arrivals, which turns the expression

$$W_C(L, U) = Q(L, U) + P_l W'_{C-1}(L - 1, U) + P_u W'_{C-1}(L, U - 1),$$

along with the approximations for computing  $W'_{C-1}(L - 1, U)$  and  $W'_{C-1}(L, U - 1)$  just derived above into a convenient recursive formula for the estimation of the cost-to-go of all states in the trellis. If reverse probabilities are ignored, as discussed, the state  $(0, 0)$  is terminal for the trellis and its cost-to-go could be backed up by means of the recursive formula, which is essentially a Bellman back-up of the long-term cost of the states [7]. The cost-to-go of state  $(0, 0)$  can be arbitrary, and for the sake of easier computation is set to zero.

As the process of backing-up proceeds from state  $(0, 0)$  towards columns with more and more free cars (from right to left), the optimal parking location for each number of free cars can be determined by comparing the costs-to-go of all states in the same column of the trellis. The optimal state then is  $(L_C^*, U_C^*) = \operatorname{argmin}_{(l, u) | l+u=C} [W_C(l, u)]$ .

It is essential that the optimal policy be determined as soon as the cost-to-go of all states in column  $C$  is backed up and before any back-ups in column  $C + 1$  are performed, because the back-ups for the states in column  $C + 1$  would need the optimal state for column  $C$  in order to determine which of the states in that column is stable and which ones are sliding. The whole process of backing up of the cost-to-go of parking states and parking policy determination is described in the algorithm DYNAMICPOLICY below. As discussed, this algorithm returns a vector of numbers  $L_C$ ,  $C = 1..N_c$ , each of which prescribes how many cars should be parked at the lobby if  $C$  free cars are available. This algorithm also makes use of the immediate cost  $Q(l, u)$  as overloaded above to take two scalar arguments  $l$  and  $u$  rather than a single vector argument  $\mathbf{x}$ . The branching probabilities  $P_l$  and  $P_u$  are assumed to have been pre-computed and stored in each state. The arguments of the algorithm are the number of cars  $N_c$  and the overall arrival rate at the building  $\lambda$ .

The algorithm DYNAMICPOLICY uses the externally computed function  $Time(C, u_1, u_2)$  which returns the time for the

---

**Algorithm 2**  $L[1..N_c] = \text{DYNAMICPOLICY}(N_c, \lambda)$ 


---

```

1:  $s[0][0].W := 0$ 
2:  $s[1][0].W := Q(1, 0)$ 
3:  $s[1][1].W := Q(1, 1)$ 
4:  $U[1] = 0$ 
5:  $L[1] = 1$ 
6: for  $C := 2$  to  $N_c$  do
7:    $w_T := s[C - 1][U[C - 1]].W$ 
8:    $mincost := \infty$ 
9:   for  $u := 0$  to  $C$  do
10:    if  $u = C$  then
11:       $W_u := 0$ 
12:    else
13:      if  $u = U[C - 1]$  then
14:         $W_u := w_T$ 
15:      else
16:         $w_0 := s[C - 1][u].W$ 
17:         $T := Time(C - 1, u, U[C - 1])$ 
18:         $W_u := w_0 - (w_0 - w_T)(1 - \exp(-\lambda T))/(\lambda T)$ 
19:      end if
20:    end if
21:    if  $u = 0$  then
22:       $W_l := 0$ 
23:    else
24:      if  $u - 1 = U[C - 1]$  then
25:         $W_l := w_T$ 
26:      else
27:         $w_0 := s[C - 1][u - 1].W$ 
28:         $T := Time(C - 1, u - 1, U[C - 1])$ 
29:         $W_l := w_0 - (w_0 - w_T)(1 - \exp(-\lambda T))/(\lambda T)$ 
30:      end if
31:    end if
32:     $s[C][u].W := Q(C - u, u) + s[C][u].P_l W_l + s[C][u].P_u W_u$ 
33:    if  $s[C][u].W < mincost$  then
34:       $mincost := s[C][u].W$ 
35:       $U[C] := u$ 
36:    end if
37:  end for
38:   $L[C] := C - U[C]$ 
39: end for
40: return  $L[1..N_c]$ 

```

---

cars to move from the configuration corresponding to the state in row  $u_1$ , column  $C$  of the trellis to the configuration corresponding to the state in row  $u_2$ , column  $C$  of the trellis. The algorithm starts computation from the second column of the trellis from the right, assuming that when only one car is available, it is always optimal to leave it parked at the lobby. This is true if at least half of the passengers arrive at the lobby.

The overall complexity of the computation is very low —  $O(C^2)$ , since the cost-to-go has to be computed only once per state, involving a constant number of operations per state, and there are  $(C + 1)(C + 2)/2$  states in the Markov chain. The policy has to be recomputed every time the estimate of arrival

rate changes significantly, which typically occurs at relatively long intervals, e.g. 5 or 15 minutes in modern elevator control systems.

We tested experimentally the performance of policies discovered by the dynamic programming algorithm described above on the same buildings used for down-peak traffic, with the statistical properties of the passenger flow reversed, so as to correspond to up-peak traffic. This time, 80% of the traffic originated at the lobby with destination the upper floors, 10% originated at the upper floors with destination the lobby, and the remaining 10% was traffic among the upper floors only. The arrival rates at the upper floors were uniform.

The comparison was performed with respect to another parking algorithm whose approach is to try to guarantee that a pre-specified number of cars are parked at the lobby at all times, so that they can intercept arrivals there. As soon as the number of cars parked at the lobby falls below that number, one of the remaining cars is selected to go to the lobby, either immediately, if it is currently free, or after completing the calls assigned to it, if it is busy. The deficiencies of this approach is that it is not clear how to determine the optimal number of cars to be parked at the lobby, and furthermore, the remaining free cars are not distributed optimally, but rather left parked where they delivered their last passenger. The parking policy found by the dynamic programming algorithm was able to improve waiting times in a manner much similar to that exhibited in down-peak traffic — savings at low and medium rates reached 82% (Fig. 3), and as the arrival rate increases, the parking solution should be turned off.



Fig. 3. Waiting times, in seconds, of two schedulers with and without parking, in both up-peak and down-peak traffic. Each dot represents an hour simulation in a different building type and arrival rate. Differences in waiting time in the range from 0 to 25 seconds are due to the parking controller, while differences above 25 seconds are due to the schedulers. (The parking controllers do not operate in this range, because there are no free cars to park for such building types, traffic patterns, and arrival rates.)

## V. CONCLUSIONS AND FUTURE RESEARCH

The problem of optimally parking elevator cars in two extremal types of passenger traffic was considered, and several solutions were proposed. For the case of down-peak traffic, distributing cars along the height of the building so as to minimize the waiting time of only the next passenger resulted in immediate savings in average waiting time for low and medium rates. While in most cases it is not possible to analytically find the optimal placement of cars, a relatively fast numerical optimization routine can be employed to find it.

Minimizing the waiting time of only the first passenger is not sufficient for the case of up-peak traffic, where the main question is how many cars should be kept at the lobby, given the height of the building and the overall arrival rate. The proposed solution to the problem of optimal parking for a group of elevators in up-peak traffic was based on the representation of the system as a Markov decision process with very few states corresponding to candidate parking locations, and a dynamic programming algorithm for minimizing the expected waiting time of several future passengers on a longer, but still limited horizon. This solution was able to capture the dependency between the arrival rate and the number of cars to be parked at the lobby, yielding very good performance for low and medium rates.

The main direction for improving the solution is to incorporate in the model the probabilities of events corresponding to cars returning from service. These probabilities are currently completely ignored, which requires a change in the criterion to be optimized by the dynamic programming algorithm. As a result, the optimal parking position for states with few cars is determined on the basis of the expected waiting time over a relatively short period. Furthermore, the ignored probabilities can be significant, and a reasonable approximation to the return-from-service probabilities is very likely to improve the parking policy.

## REFERENCES

- [1] G.C. Barney, *Elevator Traffic Handbook*, Spon Press, London, 2003.
- [2] 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.
- [3] David L. Pepyne and Christos G. Cassandras, "Optimal dispatching control for elevator systems during uppeak traffic," *IEEE transactions on control systems technology*, vol. 5, no. 6, pp. 629–643, 1997.
- [4] M. J. Schoppers, "Universal plans for reactive robots in unpredictable environments," in *Proceedings of the Tenth International Joint Conference on Artificial Intelligence (IJCAI-87)*, John McDermott, Ed., Milan, Italy, Aug. 1987, pp. 1039–1046, Morgan Kaufmann publishers Inc.: San Mateo, CA, USA.
- [5] Matthew Brand and Daniel Nikovski, "Optimal parking of elevator cars in supervisory group control," in *preparation for IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 2003.
- [6] Daniel Nikovski and Matthew Brand, "Decision-theoretic group elevator scheduling," in *13th International Conference on Automated Planning and Scheduling*, Trento, Italy, June 2003, pp. 133–142, AAAI.
- [7] Dimitri P. Bertsekas, *Dynamic Programming and Optimal Control*, Athena Scientific, Belmont, Massachusetts, 2000, Volumes 1 and 2.