Real-Time Combinatorial Optimization for Elevator Group Dispatching

Paul E. Utgoff, *Member, IEEE*, and Margaret E. Connell

***Abstract*—A new algorithm Minimize Vexation, version 10 (MV10) for managing the elevators of a group is presented. Cars are dispatched in a way that maximizes efficiency for all the individuals who use an elevator in the group of elevators. Much information about the individuals in the group is inferred or estimated, greatly aiding the decision making process. A variety of questions regarding novel components of the algorithm are posed, and the answers are provided by way of deep statistical analysis. Insights are offered into the nature of various objective functions and their effects on system performance. The service time of the MV10 algorithm compares favorably to that of known approaches to the problem, demonstrated again through statistical analysis. For comparison purposes and perspective, the elevator dispatching problem is recast in a separate system in which all individual travel plans are known ahead of time, giving an indication of the best attainable efficiency.**

***Index Terms*—Dispatching, elevators, intelligent control, optimization methods, real time systems.**

I. Introduction

A LARGE class of transportation problems consists of moving human would-be travelers from where they are to where they wish to be. One of these, the problem of dispatching building elevators, is our focus. One barely notices an elevator when it provides prompt efficient service. Yet, when confronted with a long wait, or a crowded car making many stops, it is common to imagine how the system ought to work, and to wonder how the designers could have failed so miserably. A building owner must provide elevator service that meets the needs of the occupants, or else the occupants will relocate. The quality of the elevator service is a major component of the overall utility and attraction of a building. It is essential that the elevator system that is designed for the building and its uses provide service that is good enough to satisfy the users and to avoid complaint.

Balancing the desire to provide good service is the need for the owner to use shafts efficiently. Each elevator requires space for the shaft in which it will travel. Floor space that would otherwise be usable (rentable) is lost to each shaft. One does not want to allocate any more shafts than are absolutely necessary. The challenge of saving shaft space and controlling elevators is addressed in a recently published paper in this journal [1] which considers multiple cars in a single shaft.

Manuscript received February 7, 2010; revised December 4, 2010; accepted March 11, 2011. This paper was recommended by Associate Editor F. Gomide.

P. E. Utgoff, deceased October 11, 2008.

M. E. Connell, retired, now resides at 5 Springfield St., Wilbraham, MA 01095 USA (e-mail: connell@cs.umass.edu).

Digital Object Identifier 10.1109/TSMCA.2011.2157134

We shall focus here on how a computer program, operating in real time, can decide automatically which car should go where and when. By improving efficiency, one reduces equipment requirements, sometimes including reduction of shaft space requirements. In this paper, we put aside issues of building design and address how best to direct the elevators to the floors, given a particular set of specified elevators and traffic conditions. We start with a discussion of the elevator dispatching problem followed by a survey of existing approaches to its solution.

II. Elevator Group Dispatching Problem

Elevators whose travel is coordinated by a decision-making process form an *elevator group*. A human passenger or would-be passenger, called a *user*, can identify a group by noticing a set of elevators whose doors are near to each other. People usually expect that all the elevators of the group will serve largely the same floors, and that the travel plans of these elevators are somehow coordinated. For historical reasons, it is common to refer to directing the travel of the cars of a group as *dispatching*. Coordinating the travel of the cars of a group is known as *elevator group dispatching*.

Definitions and Assumptions: Suppose that we are given a set L of landings. We are also given a set of elevators E, also called cars, each of which serves some subset of the landings. Typically, every car serves every landing, but this need not be so.

At each landing except the lowest and highest, there are two *hall call buttons*. A user standing in the hall may press a hall call button to call the car to that landing. The choice of button, either upward or downward, informs the system of the required direction of travel for the user waiting in the hall. We define U as the set of all up hall calls and D as the set of all down hall calls. Inside each car e_i is a set C_i of *car call* buttons, one for each landing that car e_i serves. The term, car call, refers to a car call button that has been pushed.

The dispatching problem is commonly simplified by adopting several customary tenets [2].

1) An elevator must never be allowed to bypass a car call for that elevator. The car must stop at a landing for which there is a car call if that landing is ahead of the car in the current direction of travel. The car calls that would cause the car to reverse direction must be ignored, until the run in the current direction has been completed.
2) A car with passengers aboard must drop off all its passengers, including those that it may pick up for the same direction while dropping off others, before changing its direction of travel.

3) An elevator may bypass a hall call in the same direction of travel, when the car is already full of riders or if that hall call is currently assigned to some other car.
4) Elevators may not stop between landings.
5) When a car begins its deceleration to stop at a landing, the car becomes thereafter committed to stopping at the landing.

Each car call in the direction ahead of the car will be honored in distance (floor) order until the run in that direction has been completed. Only when there are no car calls or assigned hall calls ahead may a car reverse its direction of travel. From these rules, it becomes evident that for a given car, its behavior is determined by its current direction of travel, its current assigned hall calls, and its car calls. This reduces the dispatching problem to that of assigning the hall calls in U and D to the cars in E.

Computational Complexity: The problem of how to assign hall calls to the cars is the essence of the group dispatching problem. There is the remaining uncertainty of how many users will board a car with open doors and where those users will want to disembark. For now we consider that the main element of control administered by the dispatcher is assignment of hall calls.

Suppose that there are h total hall calls in the system, and that there are $|E|$ total cars in service, then there are $|E|^h$ possible ways of assigning the hall calls to the cars. We shall refer to each possible set of assignments of hall calls to elevators as an *assignment set*, and to the set of all currently possible assignment sets as the *assignment space*. If one could afford to evaluate the cost of each assignment set in the assignment space, then one would do well to choose the assignment set of lowest cost. A simple cost metric, for example, would be to minimize the sum of each traveler's time in the elevator system.

Notice that the quantity $|E|^h$ is exponential in the number of hall calls. For example, with as few as 10 hall calls for 6 elevators, the assignment space would contain $6^{10} = 60\,466\,176$ distinct assignment sets. Given present computing capabilities, it is not feasible to consider this many possibilities for the real-time problem of dispatching cars to landings to serve users. We need an algorithm that can find a good, but possibly globally suboptimal, assignment set in an efficient manner.

The dispatching problem is further complicated by a variety of additional factors. Some hall calls have been waiting much longer than others. Each car is at a particular location, perhaps in transit, with its own set of car calls to serve. The specific demands that users will place on the system are not known ahead of time, and vary considerably. When a car opens its doors to let on the one or more users associated with a hall call or to let users off, anyone can get on and the set of additional car call buttons that will be pushed is not known. This is critical, because the car may have had few stops ahead, and come to have many stops ahead as users board and press car call buttons.

Because of these complexities and uncertainties, there are many ways of solving the dispatching problem as well as potentials for improvement.

III. Survey of Existing Approaches

We have noted that the number of possible assignment sets is exponential in the number of hall calls. It is not feasible to identify an optimal solution when there is heavy traffic. One must settle for non-optimal solutions, so the question is how well one can do given the real-time constraints of the problem. To handle the real-time aspect, advanced systems take a snapshot of the entire system, and then find an improved assignment set in a short enough period (milliseconds) that it is still valid. One can devise many different kinds of suboptimal algorithms, leaving head-to-head empirical comparisons as the principal method for estimating which algorithm may be expected to deliver better performance than another.

One must consider what it means to deliver customers efficiently. People do not like waiting. The elevator industry places emphasis on hall call waiting time because it believes that a person is typically most anxious when waiting in the hall. Conventional wisdom [3] says that people generally do not notice a hall call wait of 30 seconds or less, and become increasingly vexed as hall call waiting time grows beyond 30 seconds.

All group dispatching systems can be seen as searching the space of possible sets of assignments of hall calls to specific elevators. The search methods may differ in fundamental ways, but each ultimately selects a set of hall call assignments that will be imposed on the actual dispatching system. The rest of this section describes some of the computer-based dispatching algorithms that have been devised during the recent decades.

Assign Each New Hall Call to Best Car: One of the earliest approaches to group elevator dispatching was to assign each new hall call to the car that is nearest (in distance or time) and that is in its direction of motion, keeping this first assignment fixed throughout [4]. These approaches suffer when new hall calls keep appearing in front of (closer than) those that have already been assigned. In the worst case, all the hall calls will be assigned to one car. More generally, the impact of a new hall call assignment on those hall calls that have already been assigned is ignored entirely. The resulting search of assignment space is myopic, producing poor efficiency.

A somewhat improved approach is to assign each new hall call to the car that can handle it best overall. Since the assignment of a given hall call to an elevator will impact all car calls and all of the hall calls that come after it, the new hall call is assigned to a car in a manner that distributes the load. If the overall objective is computed as the sum of the individual car objectives, then one can identify a car for which the overall objective is minimized. This approach is quite intuitive, and it is the basis for old and new algorithms alike [4].

For some approaches, such as the Estimated Time of Arrival (ETA) algorithm with continuous reallocation [4], the hall calls are repeatedly put aside and then reassigned to the cars, following the order in which the hall calls were received. This affords a measure of responsiveness to changing conditions, but it performs essentially the same search of assignment-set space every time. Maintaining the single ordering over the hall calls limits severely how much of the assignment-set space will be considered. In a variant proposed by Rong *et al.* [5] heuristic rules select a few hall calls for serial reallocation, varying the

search order somewhat. This version of ETA, when assigning a call to a car, also considers the impact on all users already assigned to the car.

Sectoring: The static sectoring approach addresses the need to keep cars distributed throughout the building. The set of landings is divided into a fixed number of zones (sectors), each containing a group of contiguous floors. In general, if an elevator is in a particular zone at any given time, it is assigned to the zone and answers calls within it. The elevator no longer is assigned to the sector when it leaves it. There can be all sorts of modifications and rules to improve the system in special cases. Barney [4] reports that this approach works well for interfloor and uppeak traffic but not for down peak or heavy traffic.

In dynamic sectoring the zone boundaries are flexible and are determined by the elevators. The zone of an elevator is bounded by its position and where the nearest elevator ahead of it is positioned. Again this approach works well in uppeak and interfloor traffic but poorly in down peak traffic.

Channeling: A critical issue is how to reduce the number of stops that each car must make to service all of its calls. Each stop requires car deceleration, braking, door opening, unloading, loading, door closing, and acceleration. Any stop that can be eliminated reduces time for those who will be continuing onward. For example, if there were 4 cars and 48 people waiting in the lobby with 4 possible destination landings, it is preferable to have all people going to the same landing board the same car.

Channeling, an approach taken by Otis Elevator [6], determines and displays in the lobby which floors will be served by which car. This serves to reduce duplication of car calls among the cars of the group and is effective with uppeak traffic. The 48 passenger example above would be handled efficiently, assuming that all passengers board the indicated correct car.

Destination Control: Destination control has been deployed principally by Schindler [7]. Using keypads, each passenger enters his/her destination while still in the hallway. There are no car call buttons in the cars. The system then indicates which car will stop at each destination. Destination control is particularly good at grouping in heavy, uppeak traffic [4], [9] although passenger wait times may increase. Koehler [8] describes an auction system that uses destination control yet it still does greedy assignment. She reports an improvement of 10% over Schindler's Miconic 10. The disadvantage with destination control dispatching is that once a passenger is assigned to a car, the car is committed and cannot react to changing conditions. Peters [9] reports that destination control does not necessarily benefit down peak and lunchtime service because of its lack of flexibility. Thyssen Krupp Elevator Inc. [10] is using a hybrid, more flexible version of destination control, Estimated Time to Destination (ETD), responding to destination input at some floors while estimating the destination at others.

Knowledge-Based Methods: Expert systems were built to dispatch elevators in the 1980s and 1990s [8]. Once a traffic pattern in a building was identified by passenger counts or by human experts, fixed rules determined the elevator assignments. However, it was found that clear patterns were difficult to determine.

Siikonen [17] describes Kone's Traffic Master System (TMS9000) control system, which includes their Enhanced Spacing Principle (ESP) algorithm for allocating the hall calls to the elevators. The system accumulates statistics for traffic demand over the day of the week and the time of day and classifies the current traffic load into one of 26 known demand patterns. Individual users are modeled while the system attempts to minimize the user wait times collectively.

Siikonen [18] provides a broader view of her elevator-related research. She discusses dispatching as a processes of optimizing the hall call assignments by minimizing a cost function. Nevertheless, the hall calls are assigned in a greedy fashion from longest wait to shortest. The objective function can weight multiple factors, beyond just wait time, and a set of such weights can be selected automatically by fuzzy rules that recognize various traffic patterns.

Optimization and Adaptive Methods: One could apply a genetic algorithm [11] to the problem of finding a good assignment set by representing the assignment set as a string of car number assignments, with each position corresponding to a particular hall call. The fitness function could be the sum of the waiting times for all the users. Cortes *et al.* [12] has taken this approach. By repeatedly applying genetic operators such as crossover and mutation, one can simulate evolution. The result is a somewhat optimized set of hall call assignments. In the particular study cited, the comparison of the system using a genetic algorithm was to a straw man system in which each hall call is simply assigned to the physically nearest car, which is known to produce weak dispatching. The approach is of interest because it offers a different method of searching assignment-set space.

Crites and Barto [13] describe a down peak dispatching algorithm that was produced through reinforcement learning [14]. Each elevator's controller was modeled as an agent whose sole decision while descending was whether to stop at the next landing or to bypass it. The policy for whether to stop was learned by simulating down peak activity. The simulation runs very much faster than real time, so it is practical to obtain a policy in this manner. Dispatching performance compared favorably to several published algorithms.

Pepyne and Cassandras [15] offer a solution for purely uppeak traffic. They cast the problem as one of satisfying a lobby queue of clients served by n cars. Each car is filled, one at a time, to a certain proportion of full capacity, and then the car is sent on its way. There will be no other hall calls, nor will there be users boarding at floors other than the lobby. Pepyne and Cassandras show that their solution gives globally optimal performance in terms of average passenger waiting time, and further that the only decision to make is how full to fill a car before sending it along. The car should be filled to a certain fraction of its capacity, depending on the mean arrival rate of users at the lobby. They show further [16] how to estimate the mean arrival rate in real time.

Nikovski and Brand [19] present a method for assigning a new hall call to a car optimally, subject to certain assumptions. First, there is just a single new hall call to assign to a car, with all other earlier hall call assignments remaining fixed. Second, the probability distribution for destination floors ahead of every call is uniform. Under these conditions, their algorithm, which is based on dynamic programming, can determine the globally best car to which to assign a new hall call. Nikovski and Brand

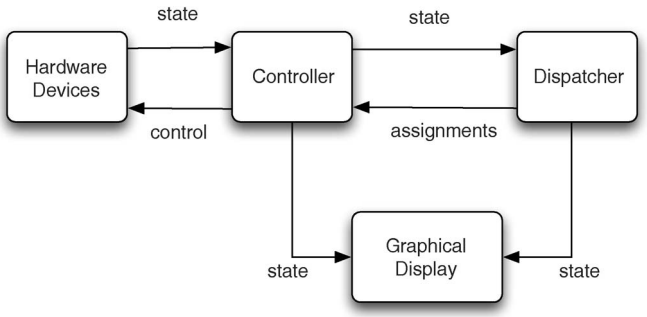

Fig. 1. Top level of MV10 system.

[20] went further addressing uncertainty with respect to future passengers. Using a probabilistic model which included the effects of future passengers arriving only at the lobby, their experiments showed improved waiting time in uppeak traffic.

IV. MV10 System

We first describe our own basic Minimize Vexation, version 10 (MV10) system in a comprehensive manner and then present the key elements of our dispatcher algorithm. The system was run on a 1.60 GHz speed processor. Nothing else was running simultaneously.

A. Top Level Description

The MV10 system consists of several components, as depicted in Fig. 1. The component of principal interest is called the *dispatcher*. It receives input information, in real time, that describes the state of the elevator group. It sends output information, in real time, telling the controller which hall calls are assigned to which cars or the landing at which an empty idle car should park. The dispatcher is implemented entirely as a software system, without access to any of the elevator group's hardware or devices, except for its communication lines. One advantage of this arrangement is that the dispatcher can be used either with a real elevator group or instead a simulated elevator group.

The second component of interest is called the *controller*. It is connected to all of the hardware devices of the elevator group. These devices include all the hall call buttons, car call buttons, lanterns, doors and cars. The controller is the only process that controls the hardware. In addition, the controller is connected to the dispatcher. The controller sends output information, in real time, that describes the state of the system and receives input information, in real time, from the dispatcher. The controller can instead be a simulator in which all of the hardware of an actual elevator group is simulated. In this case the actions of the users in the system are also simulated. The dispatcher does not know whether the controller is real or simulated. It can serve any group, simulated or real, whose controller communicates with it correctly. This plug compatible design is convenient. It should be emphasized that the dispatcher and the controller do not share memory and that all common information passes through communication lines.

The third component is a graphical user interface (GUI). This is optional because neither the controller nor the dispatcher require it to be able to function. When the GUI is present, it receives information from the controller or the dispatcher (or both) about the state of the system, including taking a particular car on or off line, and including injecting a hall or car call into the system, as though a user had pushed a corresponding button.

The dispatcher and or controller streams various items of information to output files from which reporting programs can produce statistics and summaries of interest.

TABLE I
MV10 Dispatcher Algorithm

```
tick_time ← 0
keepgoing ← 1
while(keepgoing)
   {
   real_time ← get actual time from the processor
   if (real_time >= tick_time)
      {
      tick_time ← real_time +0.01
      update_elevators; update dispatcher view of cars, users, demands
      if dispatcher's plan for an elevator has changed
         send the new plan to the controller
      update_inputs; get input from controller
            assign any new hall call to best car
      if one second has elapsed
         update_probabilities
         probabilities of hall call at landings and of user exit landings
      if one second has elapsed
         update_estimated_number_of_users_behind_hall_calls
      if one second has elapsed
         update_parking; update dispatcher view of placing idle cars
         send destination of idle car to controller
      }
   else
      {
      update_assignments; optimize assignments trying new set of assignments
         apply MV10 optimizaton algorithm. See Table II
      if new total cost is lower than current total cost
         make new plan for cars, based on new assignments
      }
   }
```

B. MV10 Dispatcher Algorithm

The MV10 system refers amorphously to all of the system that we developed, including the simulated controller, whereas the MV10 dispatcher algorithm refers only to the dispatcher algorithm. See Table I. Ours is a dynamic algorithm. It anticipates user needs and continually updates its model of cars, users and demands while persistently optimizing its cost function, all in real time.

The dispatcher algorithm manages a variety of tasks, all part of the algorithm, and is in some sense an operating system. The algorithm has a main loop that executes repeatedly. An elapsed time clock with 0.01 s resolution can be sampled at any time. For the purpose of the dispatcher, the clock is considered to have ticked when the last 0.01 s has transpired since a reference clock sample was stored. The main loop checks to see whether the clock has ticked a centisecond. If the clock has not ticked a centisecond, then the assignments are continually updated until it has. When updating, a different assignment set of hall calls to cars is constructed, evaluated and either accepted or rejected. If the centisecond clock has ticked, the algorithm updates its model of the cars, doors and users and sends any changed plans to the controller. A car's plans can change as a result of new assignments. It then checks all input

channels for input and processes all that is waiting. It may or may not then run an assortment of tasks. The decision to run a task depends on whether the prescribed amount of time has elapsed. This time is 1.0 s or more. It is a simple matter to store the clock and sleep the interval of time until a task is to be run. When the clock tick exceeds its time, the process is repeated. By giving different initial tick moments, the tasks do not run on the same loop iteration. By managing when to do what and how frequently, the dispatcher algorithm keeps current and has time to search for improved assignments of hall calls to cars. The dispatcher algorithm does as much optimizing as the computing resources and real-time constraints afford. It differs from other dispatching algorithms in this regard. Other dispatching algorithms do perform continuous reallocation of calls, but do so at fixed times or in response to fixed events. For instance, ETA [4] updates assignments every second and ETD [10] does so when new passenger information becomes available.

The MV10 group dispatching algorithm represents every user that it can infer to be in the system. Briefly, a newly pressed hall call button indicates the presence of a new user in the system. We infer the presence of additional users by recent arrival rates (explained later) at that floor in the same intended direction of travel. For those users that are estimated to be in a hallway, the system estimates the destination for each one, based on recent experience. We call the hall call user or each inferred user in a hallway an *ipick*. We call each new or inferred user in a car is an *idrop*. The destination floor for an idrop is determined either by the press of a car call button, or by computing the expected floor from a dynamic table of probabilities, described below.

The MV10 dispatching algorithm is responsive to hallway demand whenever and wherever it occurs. The algorithm performs traffic modeling, real-time discrete optimization, user modeling, dynamic probability table updating, and idle car placement. It also includes a mechanism for projecting specific users that would enter the system in the near future. We now look more closely at each of these elements of the algorithm.

1) Traffic Modeling: At the onset the dispatcher receives the building parameters and kinematic information from the controller. From these it can model the continuous travel of the cars. Ongoing input from the controller informs the dispatcher of key events such as a car stopping at a floor, a door opening or closing, a button push, a car's weight etc. Thus the dispatcher continuously keeps current and correct its model of the behavior of the actual elevator system. In the intervals between notifications of these events the dispatcher models ongoing traffic. Because the dispatcher is modeling the position and motion of the cars and of the cars' doors, its view of the actual system is not crude. This improves the accuracy of the computation.

2) Real-Time Discrete Optimization, and the MV10 Optimization Algorithm: The decision of which assignment set is best is a discrete optimization problem because each assignment set is a discrete point in the assignment space to be searched. An optimization algorithm must search for a best possible assignment set in a timely manner. We refer to an assignment set as a solution to the present dispatching problem. Furthermore, the optimizer must search while the system is in use and is therefore changing dynamically in real time. If the elevator system diverges too far from the optimizer's snapshot of the system and proposed assignment set, then the solution may no longer apply. The optimizer must be an anytime algorithm, which means that it must always maintain a best assignment set so far. It must search steadily, rapidly, and relentlessly for an improved assignment set, in such a way that any proposed solution will apply to the current system.

There is a class of optimization algorithms known as local search algorithms [21]. These find the best solution to a difficult problem when there may not be enough computational power or time to find an optimal solution. Search is accomplished when a candidate solution is randomly or iteratively changed and then evaluated according to some criterion. In the case of our elevator assignments, search is accomplished by randomly permuting the currently best solution. One assignment set can be altered to become a different assignment set by changing one or more of the hall call assignments of the assignment set. If the permutation is an improvement, then it becomes the new currently best solution, and otherwise it is discarded. Some algorithms, such as simulated annealing, allow accepting a permutation that is worse than the current best. Ours does not.

We take a snapshot of the current state of the elevator system as modeled by the dispatcher, search quickly within a small fraction of a second for an improved assignment set, and adopt the improvement if found. The new assignment set should apply because the system will not have become very different by a fraction of a second later.

A simple approach, which is used in the classic ETA algorithm, would be to perform a simple greedy assignment of the hall calls to the cars. One could assign the oldest hall call to the car that can serve it soonest and do this repeatedly for the list of hall calls sorted by age. Rong [5] improves on this with heuristics. Siikonen [18] starts searching from longest wait which is again a greedy approach. As previously mentioned, a significant disadvantage of this approach is that it is highly likely to make the same set of assignments each time it is called, as long as the cars are spending the expected amount of time at landings, and as long as the set of hall calls and car calls remains constant. It would be far better for an optimization algorithm to search more of the assignment space.

The MV10 optimization algorithm operates by considering one alternative assignment set each time it is called. See Table II. Unlike the greedy algorithms, it has a random element, as one often finds in discrete optimization algorithms that employ local search. The procedure selects up to three of the current hall call assignments at random. If there are fewer than three, all are selected. Each of these calls is "pitched" and considered for reassignment. In the same order that these three were selected at random, each is assigned to the car that can handle it best according to the objective function, given the assignments made so far. Note that by following the order in which the hall calls were selected at random, the algorithm considers permutations of assignment order for the selected calls. If the resulting assignment set is better overall, then it becomes the current best assignment set. Otherwise, the original assignments are retained. As discussed below, a side effect of evaluating an assignment set is that it produces a plan

TABLE II
MV10 OPTIMIZATION ALGORITHM AND OBJECTIVE FUNCTION

current_total_cost = cost of current assignment set.
randomly select (pitch) 3 hall calls, $hallcall_j (j = 1, 2, 3)$
calculate cost with current assignment set, omitting the users behind the 3 hall calls
for each pitched $hallcall_j$ taken in order of random selection
  assign $hallcall_j$ to car with lowest cost.
if (*new_total_cost* < *current_total_cost*)
  make a plan based on new assignments

The Objective function is defined below:
cost of assigning a group of $nusersCar_j$ to car_j:

$$CostCar_j = \sum_{k=1}^{nusersCar_j} vexation(s_{j,k})$$

where $nusersCar_j$ is number of users either in the car or assigned to the car.
where

$$vexation(s_{j,k}) = s_{j,k} \cdot 1.03^{s_{j,k}}$$

and where the wasted time for each user is

$$s_{j,k} = user_{j,k}(totalTimeinSystem) - user_{j,k}(idealtime)$$

$$totalcost = \sum_{j=1}^{ncars} CostCar_j$$

where *ncars* is the number of cars

of movement for each car. When an improved assignment set is found, each car can be told of its own hall call assignments. Although a set of assigned hall calls and car calls for a car and its current direction of travel determine its behavior, it can be expedient for the dispatcher simply to tell each car what to do because the dispatcher has already determined each car's itinerary. This is a detail of implementation.

The above discussion has assumed that all hall calls already have assignments to cars. This is true whenever the optimization algorithm is called. However, when a new hall call button is pushed, the system creates an ipick, and assigns the new hall call greedily, i.e., to the car that can handle it best according to the objective function. The top level dispatching algorithm performs a variety of tasks according to a schedule. The task of considering all pending input, for example notification of a new hall call, precedes that of updating the assignments. Thus, whenever one considers an alternative assignment set, every hall call already has an existing assignment to a car in the current assignment set.

3) Objective Function: Unlike many other algorithms that optimize waiting time [4], [5], [17], the MV10 objective function considers the total time that each inferred user is in the system. We have seen that destination control systems also optimize the time it takes for every user to reach his/her destination. The objective function maps a state of the elevator group to a single scalar value that summarizes the net quality of that state. Of course the elevator group is always in some state at any moment in time. Much of that state is due to the physical realities of the cars, the doors, the car call buttons, the hall call buttons and their timers, and the modeled users in the system and their destination goals. The only free variables are the hall call assignments. For the elevator group, there is a current assignment set. However, the system could decide upon a different assignment set, and it should do so if a better set can be found. One needs a fast method for assessing the quality of a given hypothesized state.[1]

[1]Note that Schindler [22] also employs simulated search in which they simulate just part of the travel to optimize an objective function. They first estimate the time needed to search followed by, if time allows, heuristic search with pruning.

An objective function calculates a set of measures, and then combines them arithmetically to determine the overall quality of the given state. We describe our measures here, but hasten to emphasize that it is usually a simple matter to change the measures. Our objective function measures the *vexation* of each user that is assumed to be in the system. See Table II. Vexation is a superlinear measure of the amount of unproductive time that a user is expected to spend in the system. If s indicates the wasted seconds (unproductive time), then $vexation(s) = s \cdot 1.03^s$. We would expect any monotonically increasing function to work well if it is relatively flat from 0 to 30 s, and relatively steep upwards for values beyond 30 s.

Unproductive time is that which exceeds the minimum possible time. In the ideal case, a user would enter the hall, find a car waiting in the right direction with its doors open, board the car, and travel immediately to his/her destination in a single jump. Any time that the user spends in the system that exceeds this ideal is wasted.

The dispatcher algorithm has a view of the current state of each elevator and its inferred users (including the first user behind a hall call button). At any given time it also has estimated the amount of time each user has already been in the system. The objective function begins simulation from the current state and at the current time and quickly computes the added time of delivering all inferred users to their final, estimated destinations and then computes the cost. In doing so it is adding future time to compute cost while simulating future travel. The objective function is computed by quickly simulating the pickup and delivery of all the ipicks and idrops for each car independently. Here, the moves are not actually executed, meaning that the current state of the elevators does not change. This simulation does not create new users. All ipicks board when possible, all boarding ipicks become idrops, and all idrops depart when possible, all according to the rules for pickup and delivery discussed at the outset. The number of inferred users and their destinations are estimated by the dispatcher algorithm and are explained in more detail in the following sections. Because we simulate handling the users in the objective function, it is straightforward to compute wasted time for each. It is also simple to compute other measures, if needed, for example costs associated with moving the cars.

Some of our implementation details are worth mentioning. First we model each car as a deterministic Markov decision process. There are a finite number of states. At each time step and with each state and each action we specify a new state which depends only on the current state. Second, the Markov decision process for a car can be simpler in the cost simulation, after a current car movement or door movement has been handled. One can charge for door opening and closing without simulating doors because the costs are fixed and known. Similarly, once the car is not in simulated motion, its travel from floor to floor can be calculated by floor to floor jump times stored in a lookup table.

Finally, the objective function returns not only the cost of a particular set of hall call assignments, but also the first few steps (such as where it should go, and for what reason) for each car, called its *plan*. It is efficient to store a car's plan so that the

group dispatcher can send specific instructions to the controller about what each car is to do in the near term. This is important so that lanterns, door times, and travel direction will be known to a car's controller in an actual system.

Typically for the heavily loaded system of 6 cars and 18 floors which we use in our experiments there are around 10 000 calls to the objective function per second on a 1.60 GHz speed processor.

4) Modeling Individual Users: An important part of our dispatcher algorithm design is that it attempts to model the specific users in the elevator group. A *user* is a person who appears in a hallway planning to ride an elevator, boards an elevator, and exits at his/her destination landing.

The representation makes it possible to be sensitive to every user's efficiency. This contrasts with the representations of some algorithms that just answer the known hall calls and known car calls as indicated by the pushed call buttons which is a coarse representation and ignores the users that are behind each button. The presence of the first person to push the button (register the call) is known to the system. A user that does not need to push an already pushed button is also behind the button, but the presence of such a user is not known unless other sensors are present. Although there may be an indication of the actual number of users in the car (viz. weight), the destination of each is uncertain. To the extent that a system can accurately infer the presence of all the users behind a button (hall or car) and these users' destinations, the system can improve its planning.

Other systems also model users. Siikonen [17] models the number of users behind a hall call with statistical forecasts, as well as the number of users in a car by the difference in the car's arriving and leaving weight. Most ETA algorithms (Barney [4]) take the number of people behind a hall call as one, and its destination floor as one of the equally probable possible destination floors. Of course, when each user indicates his/her presence and destination as in the destination control systems, there is no need for inference. In the hybrid algorithm ETD [10] the number of users in a car is estimated by weight change and passenger destination is inferred from statistical data, although the actual manner of how this is done is unspecified in the literature.

The value of the MV10 objective function is dependent on the estimate of the number of users behind a hall call, the estimate of the number of users in each car and each user's estimated destination. All of these estimates are used in computing the value of the objective function and are presented in detail below.

a) Estimating how many users in the hallway are behind each hall call button: A problem that arises regularly is how to estimate the specific demand on the group that is implied by a pushed hall call button. We do not know whether a lone user or a fast accumulating mob is waiting for service. There has been work on various kinds of hallway sensors, but we have instead taken the approach of attempting to estimate how many users are waiting at a landing with the intention of traveling in the same direction. The dispatcher's estimation of the number of users in the hallway is a feature directly involved in the assignment evaluation because all the users behind a hall call are inferred users and the time of their travel to destination contributes to the value of the cost function.

The challenge is to estimate how many users are waiting at a pushed hall call button. For this, we shall refer to a *period* as the set of events that start with the push of a hall call button and end with the next push of that hall call button in the indicated direction. Suppose a period for a given floor/direction starts at time t_{b1} and ends at time t_{b2}. The number of users that accumulate at a landing within a period can be estimated by the number of new users that get into all cars going in a particular direction at that landing during a period. Because both the minimum weight of a car and the closing weight are sent to the dispatcher by the controller, one can estimate the lowest number of riders and the closing number of riders and thus the number of riders who entered any particular car during the period. If k users boarded during this period, then one can say that a user appeared every $(t_{b2} - t_{b1})/k$ seconds on average during this period. If the button were pushed at a later time, we might estimate that users would continue to arrive at this average rate. We call this particular estimate of the average arrival rate the *drip rate*. For every hall call button, its drip rate in a particular direction is computed in this manner.

The drip rate which is based on recent history is put to good use. Once a hall call button is pushed, the dispatcher starts to drip new inferred users into the system according to the drip rate of that floor/direction. As new users are artificially dripped into the system, the importance of each active hall call is affected. This is particularly useful because our system is oriented to users actually in the system or believed to be in the system, not call buttons.

One would like to treat the set of users at a landing/direction combination as a group, because they will be served as a group by a car that is dispatched to service them. We provide for an ipick to represent more than one user and sometimes refer to an ipick as an *igroup*. To mention an igroup is to acknowledge a set of users assumed to be waiting at the hall call. The size of an igroup is limited to a carload of users. For drip rates that are slow, the car will likely answer the call before an artificial user is dripped into the system.

Consider a scenario such as a morning flood of demand at the ground floor. Initially, the system responds to a hall call as it would any other. For a while, a single car accommodates all the hallway users. As the time between calls begins to shorten and more users enter the cars, the system tracks along with a faster drip rate. If the rate at which cars arrive to service users falls behind, an igroup will become saturated, and a new igroup will be started. Now the system must send two cars, because it knows that one car cannot service all of the users.

An increase in hallway demand may rise rapidly. Suppose more users are waiting than can be accommodated by one car. In such a case, one or more users will be left behind. As the car leaves, it cancels the button, and a frustrated user will press it almost immediately afterward. The system handles this *repush* by immediately dripping 1/3 of a carload of users. As multiple cars begin to serve a call, more users are accommodated.

b) Estimating the destination for a user in the hallway: To be able to estimate the cost of delivering the current, inferred users in the system, it is helpful to be able to estimate the

landing at which each user will exit the car. When the objective function does a fast simulation to calculate the cost of dropping off all inferred users, the estimated destination of each user is important.

Whenever a hall call button is pushed, one knows that a user has entered the system with the desire to travel in the direction indicated. The dispatcher algorithm estimates the probability of traveling to any of the floors ahead of the landing, and from these probabilities determines a most likely destination landing. We assume that all the waiting hallway users are headed to the most likely landing for a single user. The computation is done as follows.

To be able to compute an expected dropoff landing that is relevant to current traffic, our MV10 dispatcher algorithm maintains a set of conditional probability distributions, one for each landing and possible direction of travel. For example, in a 10 story group, an upward bound user entering at landing 2 could travel to any of landings from 3 to 10, giving eight possible destinations. A downward bound user entering at landing 3 could travel to any of landings from 2 to 1, giving two possible destinations.

For each landing/direction, the destination (conditional) distribution is initialized to a uniform distribution. For example, a distribution with three possible destinations would set the probability of traveling to any one of the destinations to 1/3 each. Thereafter, each time an entering user pushes a car call button the probability of the entry and exit floor is bumped up, and then all the probabilities of that conditional distribution are normalized to sum to 1.0.

Though we have a set of independent destination distributions, it is convenient to store all of them in a single square 2-D matrix $\boldsymbol{D}$. For instance, if a user enters the system at landing i going up, then the "upward from landing i" distribution is stored in row i from column $i+1$ to column $|L|$ (the number of landings). The conditional distribution then indicates the probability of each destination landing.

The destination distribution $\boldsymbol{D}[i, j>i]$ or $\boldsymbol{D}[i, j<i]$ is adjusted by making the true destination landing j (the destination of a user who registers a car call upon entering the car) more probable and the other destinations of the conditional distribution less probable. If l is the number of landings ahead of the car, we define a constant $\gamma \leftarrow (2.0/l)$. To make destination j more probable, set $\boldsymbol{D}[i, j] \leftarrow \boldsymbol{D}[i, j]+\gamma$ and immediately normalize by dividing all the $\boldsymbol{D}[i, j>i]$ or the $\boldsymbol{D}[i, j<i]$ by $(1+\gamma)$.

As time goes by, with no system use, the conditional distributions in $\boldsymbol{D}$ should decay toward uniform. This is accomplished by adding a small constant to every element of a distribution and then normalizing.

For any user behind a hall call button one can identify the most probable destination floor. When there are a number of equally probable destination landings, we select the median landing from among them.

c) Estimating the number of users in a car and the destination of a user in a car: For each user that is believed to be in a hallway, an ipick exists. It holds the estimated time when that user entered the system, the pickup floor pf, and the estimated dropoff floor df. For each user that is believed to be in a particular elevator, an *idrop* exists.

A hall call button tells us that there is at least one user, and whether the destination landing is above or below the landing of the hall call button. There could be more than one user in the hallway intending the same direction of travel. Information is gained whenever a new car call button is pushed. It is most likely that a user who just boarded the car saw the need for the car call and pushed the button. This verifies that for the ipick corresponding to the user there should be a corresponding idrop. However, often a user boards a car and no new car call button is pushed. In this case it is not known where the user is heading.

Our focus here is on how to model the idrops (riders) in a given car. We assume that when a new car call button is pushed, it is a request from a user who boarded at the landing where the car last stopped and opened its doors. When the doors of a car have just closed, it is an appropriate time to immediately update the set of idrops for that car, to provide as accurate a model as possible.

When the doors open, all passengers who are getting off do so and any users awaiting entry board the car. The dispatcher algorithm gains useful information from the controller's monitored weight of the car. When a car arrives at a landing, we know its weight, from which we can estimate the number of riders. After all users who wanted to leave the car have done so and after all the users in the hallway who intended to board the car have done so, we have the final weight, giving an estimate of the final number of riders at door closing. Both the arrival and departure weight are sent to the dispatcher by the controller. The MV10 dispatcher algorithm models a rider getting off for each idrop whose df (dropoff floor) is that of the current landing. These could be less than or more than the actual number of riders getting off, as measured by the weight.

When the doors of the car close, the number of idrops (inferred users in the elevator) may disagree with the number of passengers on the car as estimated by the weight. This means that the set of idrops should be adjusted. If the number of idrops is too high, then a procedure *thin_idrops* is used. Similarly, if the number of idrops is too low, then a procedure *thicken_idrops* is used. We first describe the thin_idrops procedure and then the thicken_idrops procedure.

Thin_idrops: We know from the car's weight sensors (information passed to the dispatcher from the controller) that there are fewer riders on the car than have been modeled by the dispatcher. An inferred rider (idrop) may have an estimated, uncertain df (dropoff floor). The only idrop whose dropoff floor is presumed to be certain is the one that was created in response to the push of a car call button. Typically there will be more than one rider getting off (more than one idrop with the same df) at some floors. The algorithm should do its thinning from among these idrops with uncertain destinations. One first notes the candidate idrops and then thins from among them. The procedure makes use of the conditional probability distributions of entry to destination landings and deletes the newest duplicate idrops. Thinning continues until the number of idrops agrees with the actual (viz. weight) number of riders.

Thicken_idrops: We can know from the car's weight that there are more riders on the car than have been modeled. When the doors close at a landing, it may be necessary to generate new

idrops, to model the actual number of riders. These generated idrops will all be going to one of the existing car call landings. The procedure either reinstates saved idrops (explained below) or creates new ones.

When a car stops at a landing for which a car call button is pushed, it usually discharges at least one passenger, the idrop indicated by the car call button. The dispatcher may model multiple riders (idrops) getting off at a floor. These represent inferred users whose estimated destination is the current landing and who upon entry did not push a car call button because the desired destination button was already pushed. Since it is not certain that these additional idrops actually got off at the current landing, the dispatcher algorithm models them getting off but saves them and may choose to reinstate some of them with recomputed exit landings.

When the car doors close one needs to decide how many idrops to add traveling to each of the car call landings. First, if there are saved idrops at the floor some may be restored. Those idrops that are reinstated are those with the least probability of getting off at the current floor. When the number of users assumed to be on the car equals the number inferred from the weight, no more saved idrops are added. If after restoring idrops there is still a need to add more, new idrops are added, making use of the (entry-destination) conditional probability distributions.

5) Idle Car Placement: A car is *idle* when it has no calls to serve. Where idle cars are parked affects performance, although the positioning is not directly a part of the assignment evaluation. Only those cars which are currently idle at the time the dispatcher updates parking are considered for parking. It is usually advantageous to place any idle cars at landings so that any new hall calls can be served promptly. We know that Kone (Siikonen [17]) makes use of statistical traffic forecasts to park cars. We, on the other hand, make use of the history of recent traffic. One could spread the idle cars equally far apart so that every landing could be reached equally soon. One can do better still by being sensitive to where new hall calls are likely to occur. To this end, we maintain a probability distribution, probability of hall call at landing ($\boldsymbol{p}\mathbf{ohcal}$), as a vector of estimated probabilities, one for each landing. For the purpose of placing idle cars, we do not need to distinguish between the likelihood of an up or down hall call at the same landing. The estimated probability indicates how likely either an up or down hall call at a landing is, given the recent history of users entering the system.

When the system commences operation, the vector $\boldsymbol{p}\mathbf{ohcal}$ is initialized by: $\forall_{j\in[1,\ldots,|L|]}\mathbf{pohcal}_j \leftarrow (1.0/|L|)$, where $|L|$ is the number of landings. Three variables are initialized $\alpha \leftarrow (2.0/|L|)$, $d \leftarrow 1.0$, and $\beta \leftarrow (0.01/|L|)$. Thereafter, a computation maintains $\boldsymbol{p}\mathbf{ohcal}$. We increase any $\mathbf{pohcal}_j$ when a hall call user has appeared at landing l_j in need of service in either direction. This is when either of the hall call buttons is pushed. For each known user entering the system at landing j, we set $\mathbf{pohcal}_j \leftarrow \mathbf{pohcal}_j + \alpha$ and set $d \leftarrow d + \alpha$. When it is needed as a distribution, $\boldsymbol{p}\mathbf{ohcal}$ is first normalized by dividing each $\mathbf{pohcal}_j$ by d to produce the probability distribution.

Suppose that a long period of time were to transpire without any use of the system. We do not want the probability distribution $\boldsymbol{p}\mathbf{ohcal}$ to maintain values from long ago. Rather, as time goes by, the distribution should decay back to uniform. This can be accomplished by adding a small constant amount to every $\mathbf{pohcal}_j$ before normalizing. Normalization and decay are accomplished first by $d \leftarrow d + \beta \cdot |L|$, followed by: $\forall_{j\in[1,\ldots,|L|]}\mathbf{pohcal}_j \leftarrow (\mathbf{pohcal}_j + \beta)/d$, and finally $d \leftarrow 1.0$. Whenever $\boldsymbol{p}\mathbf{ohcal}$ is needed as a probability distribution for computational purposes, it should first be normalized. The dispatcher algorithm reconsiders the positions of the idle cars once per second.

Given a probability distribution $\boldsymbol{p}\mathbf{ohcal}$, we want to position the idle cars. First, we determine the number of idle elevators. Consider the case of exactly one idle car. It is a simple matter to compute the expected landing f by: $f = \sum_{j=1}^{|L|} \mathbf{pohcal}_j \cdot j$. One could park the idle car as near to the expected landing f as possible. Recall that cars must be positioned at landings. Suppose that the group contains an express zone, specifically a very large distance between two particular landings. One would prefer to compute the expected elevation and pick the landing closer to the expected elevation. This is what we do. If there were two idle cars, one would then split $\boldsymbol{p}\mathbf{ohcal}$ into two half distributions, one for the high landings and one for the low. The split should be placed so that the sum of the $\mathbf{pohcal}_j$ for the higher landings is 0.5 and that of the lower landings is 0.5. We view the higher landings as a conditional distribution for the car that will be positioned for them. The problem is identical to the case of one idle car. Similarly, we view the lower landings as a second conditional distribution for the other car. This then is a form of dynamic zoning. In general, for k idle cars, one divides $\boldsymbol{p}\mathbf{ohcal}$ into k conditional distributions, and places each of the k cars accordingly. Cars are then assigned from lowest to highest position.

6) Projecting Hall Calls in the Near Future: The dispatcher algorithm has a mechanism for anticipating the likelihood, position and direction of future hall calls. For example, if the up button on floor 2 has recently been pushed repeatedly, it is likely to be pushed soon again. The dispatcher takes this into account and assigns a car to this floor, using the assignment algorithm. The dispatcher creates a future ipick which is treated identically to an ipick by the assignment process and cost function. A difference between an ipick and a future ipick is that the first is created in response to a hall button being pushed and the second is created when the dispatcher anticipates that a hall call button will be pushed in the near future. A future ipick, like a drip, is a guess, but whereas a drip is created when a hall button has already been pushed, a future ipick is created when the button is not yet pushed. A car will not open its doors for a future ipick, it will merely park anticipating a button push.

The creation of future hall calls (up or down) is controlled by the future arrival rate, computed for each floor and direction. This rate is continually modified. It is an estimate of how often users are expected to arrive at a given floor wanting to be serviced in a particular direction. We have discussed in Section IV-B4 how the dispatcher drips a new ipick every $(t_{b2} - t_{b1})/k$ seconds when the hall call button has already been pushed. If a hall call button has not been pushed, the dispatcher might create a future hall call which will enter the system in $0.5 \cdot ((t_{b2} - t_{b1})/k + (currentTime - t_{b2})/k)$

seconds. Here, k is the number of new users that got into all cars at a given landing and in a given direction during the period from t_{b1} to t_{b2}. The estimate of when the future hall call will enter the system is set to the current time plus this appropriate future average arrival interval.

A future hall call is created at a landing in a particular direction only if there is no hall call or future hall call currently registered there, if the corresponding future arrival interval is between 0 and 20 s, and if there are fewer than 20 ipicks in the system. The expected exit floor of a future ipick is determined in the same way as it is for an ipick. One can see that as time passes without a new hall call at the floor, the value of the future arrival interval will increase.

There are two circumstances when a future up (down) hall call at a floor is deleted. The first is when a hall call in the same direction is entered at the floor. The second is when the future hall call has existed for more than 30 s in which case it was probably not a good projection for a new hall call.

V. Analysis of Algorithm Effects

A. Common Experimental Design

When attempting to design an algorithm to perform elevator group dispatching, one must make numerous design choices. Questions naturally arise about which design choices lead to the most effective group dispatching. Suppose that there are d design choices. We call any particular setting of these design variables an algorithm and compare design choices by comparing the corresponding algorithms that result from those choices. We compare the algorithms by observing performance in different traffic patterns.

Generally, there is a set of possible user traffic distributions over a sufficiently long interval of time. It is common to factor a traffic distribution into two components. The first is a probability distribution over the possible (pf, df) pairs, where pf is a pickup floor and df is a dropoff floor. One can envision this distribution as a matrix of probabilities, indexed by pf and by df. The second factor is an arrival rate for users entering the overall system (at some pf headed for some df). We can model passenger arrival by a Poisson distribution that corresponds to a specified mean arrival rate. For any passengers that enter the system at a discrete moment in time, one can select a (pf, df) for each passenger, according to its bivariate probability distribution.

There are, of course, infinitely many point to point probability distributions, and infinitely many mean arrival rates. One can characterize these in qualitative terms. For example, if most users enter at the lobby and exit at a higher floor, one can characterize the distribution as "up." If users arrive at a high rate, then one may characterize the traffic as "heavy." There are three distributions of general interest, which we shall characterize as "up," "down," and "meeting." There are many possible arrival rates, but we shall focus on two. One is "heavy," and the other is "light," which generally indicates that much of the system's capacity is not used. A combination of distribution and arrival rate can be characterized by combining their word descriptors, such as "up heavy," or "down light."

For brevity, we call each (*distribution*, *arrival rate*) pair a *traffic scenario*. When there are s scenario choices of interest, we can then observe and compare algorithms under these s scenarios and we can measure a variable of interest for all d algorithms in all s traffic scenarios. Two-way analysis of variance (ANOVA) can be employed to test for the presence of significant differences. When there are significant differences, though two-way ANOVA [23] does not identify them explicitly, one can employ a significance test such as Tukey Honestly Significant Difference (HSD) test [23] to discern significant pairwise differences.

We would like to measure system performance. In the elevator industry, *mean wait time* is the prevalent measure. This is the time from when a user enters the system (hallway) until that user can board a car. We use this mean wait time as one, of two, experimental measurement variables. We employ a second experimental variable often called *mean service time*. For this, the time commences when a user enters the system in a hallway, and concludes when that user exits a car at the intended destination. Service time is a good measure of throughput, i.e., how quickly users are moved through the system. Lower service times mean that users are generally wasting less time in the system, and that greater capacity is available.

B. Questions and Experiments

Below we list a set of eight questions, with experiments designed to provide answers. Detailed explanations of the variant algorithms used in the experiments are explained later.

- Question: Does continuous updating assignments help?
  Experiment: Compare MV10 to MV10noUpdate.
- Question: How many hall calls should be pitched in local search > 3?
  Experiment: Compare MV10 and to MV10pitch5.
- Question: How many hall calls should be pitched in local search < 3?
  Experiment: Compare MV10 and to MV10pitch1.
- Question: Does it help to model individual users in the system?
  Experiment: Compare MV10 to MV10noAdjustDripsFuture.
- Question: Does it help to estimate probabilities?
  Experiment: Compare MV10 to MV10noProb.
- Question: Can we get close to omniscient? Could we do better?
  Experiment: Compare MV10 to Omniscient.
- Question: How does MV10 compare to ETA?
  Experiment: Compare MV10 to ETA.
- Question: How does MV10 compare to ETD?
  Experiment: Compare MV10 to ETD.

When asking many questions, one will make many performance comparisons, which broaches the statistical problem of multiple comparisons. One must use a mass significance test that is formulated to avoid false significance when multiple comparisons are performed. For clarity, we pose each question, along with a single experiment designed to provide the answer. However, when it comes time to perform the set of experiments,

we recast the individual experiments as a single experiment of the design discussed above for implied multiple comparisons. We collapse these eight comparisons (nine algorithms) into a single experiment that identifies significant difference between algorithms based on the chosen performance measure.

1) Building and Elevator Configuration: The building, elevator and passenger data that we use in our experiments is hypothetical but realistic. There are 18 possible landings in the building. The ground floor is separated from the next possible landing by an express zone of 91 ft and the interfloor distance for the remaining landings is 13 ft. When a car opens its doors at the ground floor it must dwell there for at least 8 s. No such restriction on dwell time applies at the other floors.

There are six elevators in the group, each with a capacity of 15 riders. The maximum speed of 20 ft/s, maximum acceleration of 3.5 ft/s^2, maximum deceleration of 4 ft/s^2 and jerk of 6 ft/s^3 are the same for all cars as well as preflight delay of 1 s, door opening time of 1.5 s, door closing time of 2.4 s, and passenger on or off time of 1 s. An elevator door must remain open for at least 3 s for a hall call and 1 s for a car call. A door can reopen at a given floor only after 1 s.

2) Traffic Scenarios: We ran the experiments with what we call up, down and meeting traffic patterns and each of these in both heavy and light mode. A description of these 30 minute experiments and how each was generated follows.

In the up pattern 75% of all hall calls originate at the ground floor and all destinations are equally possible. The remaining 25% of calls follow an interfloor pattern where there is no preference expressed for pick up floor or dropoff floor. In light traffic the mean arrival interval, which is the same for every floor/direction, is 5 s and in heavy traffic it is 2.5 s. Given these parameters, it is possible to generate a distribution table and a 30 min instance of a traffic scenario. It should be recalled that in a traffic scenario there is a random element in generating hall calls from probability distributions.

In the down pattern, 75% of all calls are to be dropped off at the ground floor and requests originate with equal probability from all upper floors. The remaining 25% of calls follow an interfloor pattern. Again the mean average arrival interval for light and heavy traffic are 5 s and 2.5 s, respectively.

Finally, the meeting traffic pattern starts with 10 min of interfloor traffic, all requests and destinations being equally probable. This is followed by 15 min where 50% of all requests are coming from either floor 8 or floor 11 and all of the riders are headed to floor 13. The remaining 50% of calls are equally distributed. The last 5 min of the traffic reverts again to the interfloor pattern. The mean arrival interval of light and heavy traffic are 5 s and 2.5 s, respectively.

3) Algorithms: The following is a more detailed explanation of the algorithms referred to in the list of questions with experiments.

- MV10: All of the features described as part of our algorithm are employed.
- MV10noUpdate: There is no continuous update of assignments. Once the best car is assigned to a hall call, this is the car that will carry the riders behind that call to their destinations.
- MV10pitch1: When updating assignments, only one current hall call is randomly selected and assigned by the objective function to the best car in the local search for an improved assignment set.
- MV10pitch5: When updating assignments, five current hall calls are randomly selected and assigned in search.
- Omniscient(OMN): This is the offline omniscient case in which, unlike real-time elevator dispatching, an unlimited amount of computer time is available for search and all users are known at the outset. It is described in Section VI below.
- MV10noProb: This version does not continuously update probabilities based on past events. There is no continuously updated expectation of where hall calls are likely to originate which will affect idle car placement nor where an inferred user is likely to exit.
- MV10noAdjustDripsFuture(MV10noADF): This version does not adjust the estimate of the number of riders in a car, nor the estimate of the number of people behind each hall call. It also does not anticipate future hall calls.
- ETA: This is the ETA algorithm, the industry standard, which is described in Section V-C below.
- ETD: This is our implementation of the ETD algorithm [10]. It optimizes passenger time to destination, reevaluating assignments each time new information is available. Destinations are inferred and the booster option is not included. The cost of assigning a new hall call to a candidate car is $CC = ETD + \sum_{k=1}^{n} SDF_k$, where ETD is the estimated time to destination of the new call and where SDF is the delay each user (either in the car or assigned to it) will experience if the car accepts the hall call. Because the authors do not say how they arrive at any estimates, we make all destinations equally likely and infer that there is one user behind each hall call.

4) Performance Measures and Comparisons: We measure performance with mean passenger wait time and mean passenger service time. This is the mean of all passenger wait or service times in a chosen time period. To compare the algorithms, we generated ten distinct 30 minute instances for each of the six traffic scenarios. The same ten instances of a particular traffic scenario were inputs to each of the nine dispatching algorithms. A traffic scenario instance is, in fact, a traffic file consisting of a list of the riders' time of entry, floor of entry and exit floor, generated according to the probability distribution and arrival rate of the traffic scenario. In this type of experiment one can find and subtract any variability that comes from preexisting differences among the instances themselves. This is because we found that the effects of the dispatching algorithms are consistent between instances as measured by positive correlation coefficients of the same magnitude between pairs of instances. We therefore did a two-way ANOVA with replicates [24], [25].

5) Experiments to Answer Questions: We want to answer the questions that were put forth. To do this, we examine the effects of the two independent variables, traffic scenario and dispatching algorithm, concurrently and apply two-way analysis of variance. The dependent variable is one of the

TABLE III
MEAN SERVICE/WAIT TIME FOR EACH SCENARIO-ALGORITHM PAIR

	MV10	noUpdate	MEAN noProb	SERVICE noADF	TIME pitch1	pitch5	ETA	ETD	Omniscient
lightU	44.49	46.77	46.61	45.58	44.72	44.57	49.78	48.56	32.50
heavyU	78.44	86.27	78.95	83.84	82.45	78.11	90.20	94.88	57.60
lightD	51.03	55.42	55.55	50.76	54.20	50.68	50.16	57.08	36.33
heavyD	74.36	82.24	79.70	74.77	77.27	74.34	83.55	83.35	64.00
lightM	32.53	34.55	32.60	32.76	34.02	32.83	35.54	34.82	22.62
heavyM	60.88	68.10	60.98	60.29	63.48	61.15	71.41	68.36	48.31

	MV10	noUpdate	MEAN noProb	WAIT noADF	TIME pitch1	pitch5	ETA	ETD	Omniscient
lightU	8.50	9.57	9.25	8.77	8.60	8.38	9.84	10.34	7.01
heavyU	21.87	27.26	21.66	24.28	24.92	21.46	26.91	33.69	21.50
lightD	24.29	28.24	29.76	23.11	27.59	24.05	18.85	30.07	12.07
heavyD	39.09	45.20	45.93	37.65	41.77	39.28	35.26	45.35	27.71
lightM	13.87	15.46	14.00	14.08	15.11	14.12	13.12	15.83	8.17
heavyM	33.83	39.07	34.03	33.56	35.71	34.13	33.18	39.75	23.17

performance measures. For the ANOVA test, each column represents one of the nine dispatching algorithms and each row one of the traffic scenarios. The ten values in each cell are the mean passenger wait or mean passenger service times of the ten separate scenario-algorithm instances.

First, consider *service time*. The ANOVA results show that there is a significant difference between the means of the traffic scenarios ($F(5, 477) = 3587$; $p < 0.00001$), and also between the means of the dispatching algorithms ($F(8, 477) = 253$; $p < 0.00001$). There is also significant interaction between scenario and algorithm means ($F(40, 477) = 9.69$; $p < 0.00001$).

To answer our questions, further analysis of dispatching algorithms is needed. We use the Tukey HSD test (0.05). The results are represented below. For this representation one initially sorts the algorithm means and then underlines all pairs of means that differ by less than a common critical value (in this case 1.73) which is determined by the Tukey test. Any two means that are not underscored by a common line are significantly different. Note that these are the means of the algorithms over all scenarios. Here, the MV10 prefix is omitted (see Tukey HSD test at the bottom of the page).

These service-time Tukey test results show that omniscient dispatching, OMN, is significantly better than MV10 and that MV10 is significantly better than all others except MV10pitch5 and MV10noAdjustDripsFuture. When we say one algorithm is significantly better than another, we mean that the difference in the results, according to the performance measure, is statistically significant.

We can investigate even further and compare cell means since there is significant interaction between scenario and algorithm means. The cell means, where each is the mean of the 10 instances, are displayed in Table III. Here, U stands for up traffic, D for down and M for meeting. We can compare all pairs of cell means with the Tukey HSD test (0.05). The Tukey test determined that the difference in any pair of cell means must be greater than the critical number 0.765 for this difference to be significant [26]. With this information one can analyze traffic to see where each of the components of MV10 is particularly effective in influencing low service time. We see that for service time:

1) Omniscient is significantly better than MV10 is all traffic scenarios.
2) MV10 is significantly better than MV10noUpdate and ETD in all traffic scenarios.
3) MV10 is significantly better than ETA in all traffic scenarios except light down traffic.
4) MV10 is significantly better than MV10noADF in up traffic.
5) MV10 is significantly better than MV10noProb in down traffic and light up traffic.
6) MV10 is significantly better than MV10pitch1 in all scenarios except in light up traffic, while MV10pitch5 gives no significant improvement over MV10 in any traffic scenario.

Next, we consider *wait time* as the measure of performance. The two-way ANOVA shows that there is a significant difference between the wait-time means of dispatching algorithms ($F(8, 477) = 134$; $p < 0.00001$) as well as between the means of traffic scenarios ($F(5, 477) = 2172$; $p < 0.00001$). There is also significant interaction between scenario and algorithm means ($F(40, 477) = 13.3$; $p < 0.00001$).

To compare algorithms and answer our original questions with respect to wait time, we applied the Tukey HSD test (0.05)

Tukey HSD Test, Service - time means in seconds

OMN	pitch5	MV10	noADF	noProb	pitch1	noUpdate	ETA	ETD
43.56	56.95	56.96	58.00	59.06	59.36	62.23	63.44	64.51

which determined that any pair of means must differ by more than 1.37 s for this difference to be significant (see results of the Tukey HSD test at the bottom of the page).

The results of the Tukey test show the following. Omniscient is significantly better than MV10. MV10 is significantly better than its variants MV10pitch1, MV10noProb, and MV10noUpdate as well as ETD. Although ETA gives lower mean wait time than MV10, the difference is not significant.

Again we can investigate further and compare wait-time cell means (Table III). We applied the Tukey HSD test (0.05) on these cell means which determined that the difference in any pair of cell means must be greater than the critical number, 0.6059, to be significant. The investigation of cell means for wait-time performance reveals the following.

1) Omniscient is significantly better than MV10 in all traffic scenarios, except in heavy up traffic.
2) MV10 is significantly better than MV10noUpdate and ETD in all traffic scenarios.
3) MV10 is significantly better than MV10noProb in down traffic.
4) ETA is significantly better than MV10 in down traffic, as well as in meeting traffic.
5) MV10 is significantly better than MV10Pitch1 in all traffic scenarios except light up traffic. MV10pitch5 gives no significant improvement over MV10 in any scenario.
6) MV10 is significantly better than ETA in up traffic.
7) MV10 is significantly better than MV10noADF in heavy up traffic.

Looking back at our original questions, we answer them with regard to service-time and wait-time performance.

- It does help to continually update assignments in real time for both measures of performance.
- Three or more hall calls should be pitched in local search. Search is too narrow when the reassignment of only one randomly selected hall call is considered at each reevaluation, whereas nothing is gained by considering five hall calls rather than three.
- Modeling users, and anticipating future hall calls helps significantly to minimize service time in the up traffic scenarios. Modeling users does not seem to help overall performance as measured by wait time, except in heavy up traffic.
- It does help to estimate changing probabilities of user destinations and hall call origin, particularly in down traffic according to both performance measures.
- The large significant difference in means between MV10 and OMN shows that there is lots of room for MV10 to improve if possible, regardless of the performance measure.
- MV10 is a better algorithm than ETA when minimizing service time. No statistically significant conclusion can be reached overall concerning which is the better algorithm when minimizing wait time, however in this case in down and meeting traffic ETA is significantly better than MV10. In up traffic MV10 has significantly lower wait times. We will demonstrate in Section V-D that when a system is stressed, MV10 is the better algorithm regardless of the performance measure.
- MV10 is a better dispatching algorithm than ETD.

These conclusions are reached on the basis of our chosen building, group of elevators and traffic data. Some of the above conclusions warrant further discussion.

When traffic follows a distinct pattern, MV10 dispatching responds to the pattern, lowering times. For instance, often there is large passenger demand at a particular floor, as in up traffic, and the dispatcher adjusts. Modeling users and future users is particularly effective in up traffic where the number of people waiting to go up accumulates. When there are a number of inferred users behind a hall call button at a landing, each contributes to the cost, calculated in the objective function, and the more inferred users there are behind a hall call the more important it becomes to answer the call in good time to lower cost. On the other hand, when there is a distinct user-destination pattern as in down traffic, MV10 responds with better performance. Estimating destination probabilities is particularly effective in down traffic. However, in the meeting traffic scenarios the patterns are not distinct. Consequently, modeling users and their destinations does not significantly improve performance.

That ETA has a low mean wait time at the expense of a low service time is not surprising since its objective function is based on passenger hallway wait time. This is particularly the case in down traffic where the ETA wait-time scores outperform those of MV10. Another reason for the good performance of ETA in this case might be that in down traffic, the actual number of users in the system closely matches the number of hall call buttons pushed. Consequently, it might be a benefit in down traffic that the ETA algorithm does not infer users. However, when the focus is to lessen time waiting in the hallway, a passenger's service time may lengthen as a consequence. For instance, in heavy down traffic the wait-time mean is significantly lower for ETA than that for MV10, but the ETA mean service time of heavy down traffic is unexpectedly high, indicating that the system will optimize its pick ups at the expense of delivering riders promptly. This shows that optimizing just the wait time does not optimize system throughput, in fact it may work against it. Users may arrive at their destinations later than necessary, reducing capacity of the elevator group. These results give insight into the self-limiting nature of the industry standard of optimizing wait time.

Tukey HSD Test, Wait - time means in seconds

OMN	ETA	pitch5	noADF	MV10	pitch1	noProb	noUpdate	ETD
16.61	22.86	23.57	23.57	23.58	25.62	25.77	27.47	29.17

There are several reasons contributing to the poor performance of the ETD algorithm. Because the authors [10] have not described their method for estimating hall call destinations nor their method of inferring the number of users behind a hall call, our implementation of ETD does not include these estimates. In addition ETD has no provision to prioritize the pick up of hall calls that have long waiting periods. Also, as with classic ETA there is no variation in the search order of assignments. Thus, not only do the two objective functions of MV10 and ETD differ, but also how and how often each is called differ. A comparison of ETD and MV10 performance in meeting traffic, where traffic does not follow a distinct and sustained pattern, more closely reveals how the two different objective functions affect results. These results show that the MV10 algorithm significantly outperforms the ETD algorithm.

C. Barney Comparison

In a set of experiments we compare the MV10 dispatching algorithm to the ETA algorithm on the case studies reported in Barney [4]. Building data, lift data and passenger data are available in the book which contains a complete analysis of five traffic scenarios in a 31 story building.

The comparisons are done on what the book refers to as the Upper Zone simulation. There are ten possible stops at the upper floors, from 21 to 30. The elevators can also stop at floor 7 and the main terminal, floor 0. The distance from the main terminal to the 21st floor is 98.8 m, and from the main terminal to floor 7 is 36.9 m. The interfloor distance is 4.2 m.

There are 8 elevators in the group each with the capacity of 20 persons. The maximum speed 6 m/s, maximum acceleration 1 m/s^2 and jerk 1.5 m/s^3 are identical for all elevators as well as door closing time 2.6 s, door opening time 1.3 s, and motor start delay 0.3 s. Passenger transfer time is 1 s.

Barney ran and analyzed simulations for uppeak, down peak, interfloor, and mid day traffic, as well as high demand mid day traffic. Each simulation lasts for one hour and is divided into twelve 5 min periods.

It is possible to duplicate, although not exactly, a traffic simulation since the book provides a table of rates of arrival, number of calls per 5 min periods for each floor, and a bias for each call's destination is contained in a table with a relative bias for each floor. These can be translated into arrival rates and probability distributions of (pf, df) pairs that determine the traffic scenarios.

To verify that we have reproduced accurate kinematics the flight times for each jump level should match those reported in the book. The flight time is measured from when the car doors close until the car is level at the destination floor and includes motor start delay. When one runs the GOSIMPLE program suite, found in Appendix Two of the book, and calculates the lift system dynamics, adding the preflight delay the flight times are as follows. Jump(levels) 1, 2, 3, 4, 5, 6, 7, 8 correspond to flight times 5.1, 6.8, 8.1, 9.19, 10.16, 11.03, 11.83, 12.58 s. The flight times in our simulations match those of GOSIMPLE exactly.

Our motivation in running Barney's simulation case studies is to determine how the MV10 dispatching algorithm compares to the ETA supervisory system used in the book. First we must be able to duplicate closely the book's results using our implementation of ETA dispatching and thus verify that any comparison of algorithms is meaningful. We implemented a version of the ETA algorithm, following the ETA traffic control system described in Barney [4]. When a new hall call button is pushed, it is assigned to the car that takes the shortest amount of time to reach the call while still honoring its preassigned car calls and hall calls. A quick simulation determines this cost value. Reallocation of assignments is done every second. The reassignment of hall calls relies on considering each in unchanging order (chronologically with the oldest first) one at a time. A new car is assigned to a hall call only if it can reach the call in less time than the previously assigned car.

Since the exact ETA algorithm (many of which are proprietary) used in the case studies is not given and different implementations differ in their additional features, our implementation will most likely not be exactly the same as that used in Barney's case studies. Most ETA algorithms have a provision for giving a priority to answering calls that have not been answered for a long time and ours does also. We designate a hall call that has been waiting longer than ten times the average waiting time of the last six answered calls as a priority call and it is considered first in reassigning hall calls. In this case the elevator taking the least time to arrive to the call is chosen to make a special run, only honoring its car calls on the way. ETA algorithms also often designate high-activity floors as those to which cars should be sent. Consequently, our implementation insures that a car is parked at such a floor when possible.

Our ETA implementation does not include many of the enhancements that are part of the MV10 system. There is no provision to update probability tables or to model inferred present and future users. Idle cars are parked in the same way as in the MV10 implementation (without probability updates), mainly because there is no clear indication of how parking is handled. However, as mentioned, we do park cars at high-activity floors.

Performance data for each simulation are reported by Barney. What is particularly interesting are the hourly average passenger wait and journey times. Passenger journey time is measured from hall call registration until the passenger can leave the elevator at the destination floor. Note that this measure is slightly different from service time in the previous experiments. Barney also reports the total number of answered calls in each simulation which is analogous to our report of the number of passengers entering the system.

We generated ten distinct one-hour instances of each traffic scenario using the arrival rates and biases published by Barney. For each scenario we found the instance that most closely resembles the one analyzed in the book to verify that we were closely repeating Barney's experiments. Below is a description of the five simulations and a table, Table IV, where for each single simulation we see the number of picked up calls, average wait time, and average journey time reported by Barney. We also see the number of passengers, average wait time, and average journey time of the scenario instance whose performance

TABLE IV
COMPARISONS BARNEY SIMULATION AND SCENARIO INSTANCE

	Barney num calls	Barney avg user wait	Barney avg user journey	instance num users	instance avg user wait	instance avg user journey
Up	1248	6 s	86 s	1256	6 s	84 s
Dp	1665	20 s	69 s	1664	22 s	71 s
I	510	7 s	29 s	517	8 s	30 s
Md	1170	13 s	53 s	1146	13 s	56 s
Mdhd	1756	20 s	70 s	1738	18 s	73 s

and number of passengers most closely matches the Barney simulation.

- *uppeak*: (Up) The number of simulated users arriving during the 60 minutes of uppeak traffic gradually increases from 20 calls per 5 min period to 240 calls per 5 min period and then falls back to 20. All requests originate at the main floor and are destined for floors from 21 to 30 with equal probability except for the facilities floor, 7, which is half as probable a destination as any of the others.
- *down peak*: (Dp) All requests in the down peak scenario are from the upper floors going to the main floor with half as many calls originating from floor 7 as from each floor from 21 to 30. The pattern of the rates of arrival is level at 8 calls per floor per 5 min period, peaks to 34 calls per floor per period and returns to level.
- *interfloor*: (I) In the interfloor traffic simulation requests come from all floors with destinations to all floors, although passengers are twice as likely to want to go to floor 0 and 7 as other floors. The rates of arrival are the same throughout the twelve 5 min periods at about 4 calls per 5 min for any given floor except for the main and facilities floor where the rate is 2 calls per 5 min period.
- *mid day*: (Md) In the mid day traffic scenario there is a bias for going to and returning from the seventh floor. Arrival rates vary over 5 min periods and differ depending on the floor. A complete table of passenger data is given in Barney [4], Table CS21.2.
- *mid day higher demand*: (Mdhd) The final simulation is mid day traffic with higher demand. This traffic pattern is the same as the mid day only the arrival rate increases by half again as many requests per each 5 min interval.

Looking at Table IV, there is evidence that our implementation of the ETA algorithm gives comparable results to those reported in the book. On the basis of this evidence, our implementation of ETA is used for a comparison of the ETA and MV10 algorithms. In each case the number of passengers and the average waiting and journey times reported by Barney are close to those of one of our generated traffic scenario instances.

As mentioned, ten distinct instances were generated for each of the traffic scenarios. Two dispatching algorithms were used on each. One is our version of the Barney ETA algorithm and the other is the MV10 algorithm. Two-way analysis of variance with replicates was performed to investigate the differences in passenger journey-time means and the differences in passenger wait-time means due to each algorithm. Thus for every traffic scenario the identical ten instances were input to each system in identical order.

TABLE V
MEAN JOURNEY AND WAIT TIME FOR EACH ETA, MV10 PAIR

	journey-time means		wait-time means	
	MV10	ETA	MV10	ETA
uppeak	76.8	83.7	6.17	7.12
down peak	69.5	70.9	27.51	21.57
interfloor	30.5	31.2	8.92	8.53
mid day	56.8	57.9	17.11	13.49
mid dayhd	71.4	75.7	23.13	20.53

The results show that there is significant difference between the means of the MV10 and ETA algorithms according to each performance measure (journey time: $F(1, 81) = 58.7$; $p < 0.00001$ and wait time: $F(1, 81) = 85.6$; $p < 0.00001$). The means of the five traffic scenarios are also significantly different (journey time: $F(4, 81) = 2187$; $p < 0.00001$ and wait time: $F(4, 81) = 782$; $p < 0.00001$).

Further analysis with the Tukey HSD test (0.05) reveals that the journey-time mean, 60.99 s, is significantly less when the dispatching decisions are made by the MV10 algorithm than it is, 63.85 s, when decisions are made by ETA. The difference in means is greater than the critical value, 0.74, required for statistical significance. However, the Tukey test shows that wait time is significantly less when dispatching with ETA where a statistically significant difference must exceed 0.50 s. The wait-time mean for ETA is 14.25 s and for MV10 is 16.57 s.

Because there is significant interaction between scenario means and algorithm means in both cases (journey time: $F(4, 81) = 10.1$; $p < 0.00001$ and wait time: $F(4, 81) = 23.3$; $p < 0.00001$), we can analyze further with the Tukey test and compare cell means. The difference in any pair of journey-time cell means must be greater than 0.85 for statistical significance. We see journey time is significantly better with MV10 dispatching in all traffic scenarios except interfloor, Table V. The Tukey test determined that the difference in any pair of wait-time cell means must be greater than 0.58 for significance. Looking at differences in wait-time cell means in Table V, we see that the ETA dispatching algorithm is significantly better in the mid day, mid day high demand and down peak traffic. No conclusion can be reached about interfloor traffic. MV10 is a significantly better algorithm in uppeak wait-time traffic.

In general, in the context of Barney's building, elevator, and traffic data it appears that ETA dispatching yields better wait-time performance, although further investigation shows this to be true in three of five traffic scenarios. Recall that the ETA algorithm minimizes wait time. The MV10 dispatching system is preferable overall when minimizing journey time and further investigation shows this to be true in all scenarios except interfloor. Looking at mid day and mid day higher demand traffic, ETA wait-time means are significantly better than those of MV10. However, ETA journey-time means are significantly worse in these cases. This indicates again that improvements in anticipating user demand in the hallway can result in significant increase of journey time and degradation of system throughput.

D. Stressing the System

We want to know what happens when traffic increases and the system becomes heavily loaded. To find answers, we compared

MV10 with ETA on the previous heavy traffic scenarios and building parameters described in Section V-B2 and removed one car.

The two-way ANOVA analysis of service time and wait time for this five car heavy-traffic case demonstrates that there is significant difference between means due to dispatching algorithms (service time: $F(1, 45) = 68.8$; $p < 0.00001$ and wait time: $F(1, 45) = 30.15$; $p < 0.00001$). There is also significant difference between means due to traffic scenarios (service time: $F(2, 45) = 62.2$; $p < 0.00001$ and wait time: $F(2, 45) = 8.41$; $p = 0.00079$).

The results of applying the the Tukey test to service time revealed a large difference of means, 26.33 s, where the difference had to be greater than 6.41 to be statistically significant. The MV10 service-time mean is 90.54 s while the ETA service-time mean is 116.87 s. What is noteworthy is that when the Tukey test was applied to wait time, the MV10 mean, 45.16 s, is significantly less than the ETA mean, 61.99 s. The difference of 16.83 s is greater than the 6.18 s needed for significance. Most of this difference is due to the poor performance of ETA dispatching in heavy up traffic. When ETA does the dispatching, passengers will be waiting on average much more than the industry's acceptable 30 s for a car to arrive.

We can conclude that when the system becomes stressed, there can be a significant disadvantage to using ETA dispatching irrespective of performance measure. The wait-time advantages of the ETA algorithm disappear in a heavily loaded system.

VI. Omniscient System

It is instructive to attempt to determine the best performance that one could possibly achieve for a given elevator group and a particular instance of a traffic scenario. This has the potential to offer insights. How close are we coming to this performance optimum? To this end, we have created a group dispatcher and simulator, separate from MV10, called the omniscient system (OMN).

The essential aspect of OMN is that it is an offline algorithm. All of the users of the system are known at the outset. For each user, the exact time that it enters the system, the landing at which it enters the system, and the landing at which it exits are known. All of the issues related to anticipating unknown future user demands are removed entirely.

Even knowing the future, it is expensive to find a provably globally optimal solution. Instead, OMN optimizes in a somewhat local manner. The system takes a single problem file as its input. We use the same scenario instance input files that were used in Section V-B2.

The optimization process initializes a sliding window with the first k (we use $k = 15$) users who enter the system and assigns each greedily to the locally best car according to the objective function. Optimization of the assignments is then done within the window until 200 consecutive attempts to improve the value of the objective function have failed to do so. The window is then advanced by one user, the earliest user drops out, and optimization within the new window commences. Assignments for users to the left of the window are not changed and users to the right are not considered at all. This process of optimizing, and sliding the window forward continues until there are no more users. Finally, the window is set to include all of the users. Optimization continues in this all-encompassing window until there have been 25 000 consecutive attempted optimization steps without achieving any improvement. An optimization step within the window is done in the same way as it is for MV10. As in MV10, the OMN system simulates all the cars as they serve all the known users. Vexation is the same superlinear function of wasted time used for MV10.

Table III shows that the Omniscient system reduces both wait time and service time below what we expect any dispatching algorithm, subject to real-life uncertainties, can achieve.

VII. Conclusion

We have presented a new and dynamic system for handling a group of elevators. It consists of a dispatcher which makes decisions in real time and communicates with a controller which controls simulated or real hardware. Our dispatcher's assignment of cars to calls is sensitive to current conditions and changes when better assignments are discovered. We have found that it helps to model individual users as well as their entry and destination floors. The number of users and each user's intention is often unknown, but by our providing an accurate model, built with sound inferences, we are able to make good dispatching choices. We have found that our algorithm's maintenance of dynamic probability tables based on recent history can anticipate demand. Our optimization method with a cost function that does rapid simulation searches the space of assignment sets effectively and uses all cpu power available to find improvement. We have shown through experiments that dispatching with the MV10 dispatcher algorithm results in very good performance, even in stressful conditions. We have also investigated, through statistical analysis, the utility of the various components of the algorithm including identifying when a given component is most effective. We have statistically compared the algorithm's results to previously published results and have found that MV10 performance is particularly good when it is measured by service time. Finally, experiments demonstrate that a cost function which minimizes wasted time makes good sense since optimizing wait time can be misguided.

Acknowledgment

The authors would like to thank Prof. V. Lesser and the reviewers for helpful inputs.

References

[1] A. Valdivielso and T. Miyamoto, "Multicar-elevator group control algorithm for interference prevention and optimal call allocation," *IEEE Trans. Syst., Man, Cybern. A, Syst., Humans*, vol. 41, no. 2, pp. 311–322, Mar. 2011.

[2] G. C. Barney and S. M. dos Santos, *Elevator Traffic Analysis, Design and Control*. London, U.K.: Peregrinus, 1985.

[3] G. R. Strakosch, *Vertical Transportation: Elevators and Escalators*. Hoboken, NJ: Wiley, 1983.

[4] G. C. Barney, *Elevator Traffic Handbook Theory and Practice*. New York: Spon Press, 2003.

[5] A. Rong, H. Hakonen, and R. Lahdelma, "Estimated Time of Arrival (ETA) based elevator group control algorithm with more accurate estimation," Turku Centre Comput. Sci., Turku, Finland, TUCS Tech. Rep. No 584, Dec. 2003.

[6] J. Bittar and K. Thangavelu, "Contiguous floor channeling elevator dispatching," U.S. Patent 4 804 069, Feb. 12, 1988.

[7] *Miconic 10 a Revolution in Elevator Technology*, 2002. [Online]. Available: http://www6.schindler.com/SEC/websecen.nsf/pages/elev-MHR-Mic10-01

[8] J. Koehler and D. Ottiger, "An AI-based approach to destination control in elevators," *AI Mag.*, vol. 23, no. 3, pp. 59–78, 2002.

[9] R. D. Peters, "Understanding the benefits and limitations of destination dispatch," in *Proc. ELEVCON 16*, Helsinki, Finland, 2006.

[10] R. Smith and R. Peters, "ETD algorithm with destination dispatch and booster options," *Elevator World*, vol. 50, no. 6, pp. 136–142, Jul. 2002.

[11] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*. Reading, MA: Addison-Wesley, 1989.

[12] P. Cortés, J. Larrañeta, and L. Onieva, "Genetic algorithm for controllers in elevator groups: Analysis and simulation during lunchpeak traffic," *Appl. Soft Comput.*, vol. 4, no. 2, pp. 159–174, May 2004.

[13] R. H. Crites and A. G. Barto, "Elevator group control using multiple reinforcement learning agents," *Mach. Learn.*, vol. 33, no. 2/3, pp. 235–262, Nov./Dec. 1998.

[14] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press, 1998.

[15] D. L. Pepyne and C. G. Cassandras, "Optimal dispatching control for elevator systems during uppeak traffic," *IEEE Trans. Control Syst. Technol.*, vol. 5, no. 6, pp. 629–643, Nov. 1997.

[16] D. L. Pepyne and C. G. Cassandras, "Design and implementation of an adaptive dispatching controller for elevator systems during uppeak traffic," *IEEE Trans. Control Syst. Technol.*, vol. 6, no. 5, pp. 635–650, Sep. 1998.

[17] M. L. Siikonen, "Elevator group control with artificial intelligence," KONE Corp., Helsinki, Finland, Tech. Rep. Research Report A67, 1997.

[18] M. L. Siikonen, "Planning and control models for elevators in high-rise buildings," KONE Corp., Helsinki, Finland, Tech. Rep. Research Report A68, 1997.

[19] D. Nikovski and M. Brand, "Exact calculation of expected waiting times for group elevator control," *IEEE Trans. Autom. Control*, vol. 49, no. 10, pp. 1820–1823, Oct. 2004.

[20] D. Nikovski and M. Brand, "Marginalizing out future passengers in group elevator control," Mitsubishi Elect. Res. Labs., Cambridge, MA, Tech. Rep. TR2003-62, Jun. 2003.

[21] H. Hoos and T. Stutzle, *Stochastic Local Search: Foundations and Applications*. San Francisco, CA: Morgan Kaufmann, 2005.

[22] P. Chenais, "Method and apparatus for assigning calls entered at floors to cars of a group of elevators," U.S. Patent 5 612 519, Mar. 18, 1997.

[23] R. G. Steel, J. H. Torrie, and D. A. Dickey, *Principles and Procedures of Statistics: A Biometrical Approach*, 3rd ed. New York: McGraw-Hill, 1997.

[24] R. Lowry, 1999–2009. [Online]. Available: http://faculty.vassar.edu/lowry/ch15pt1.html

[25] J. E. Freund, P. E. Livermore, and I. Miller, *Manual of Experimental Statistics*. Englewood Cliffs, NJ: Prentice-Hall, 1960.

[26] H. L. Harter, *Tables of Range and Studentized Range*. [Online]. Available: http://projecteuclid.org/euclid.aoms/1177705684

Paul E. Utgoff (M'85) received the bachelor's degree in music from Oberlin College Conservatory of Music, Oberlin, OH, in 1979 and the Dr. Phil. degree in computer science from Rutgers, The State University of New Jersey, in 1984.

He was a Full Professor in the Department of Computer Science at the University of Massachusetts Amherst. He is recognized internationally as a pioneer and leader in the area of machine learning. His work on incremental decision tree learning, multivariate decision trees and problem representation are notable. He served as an action editor for the Machine Learning journal from 1991 to 1995. He also served in may organizational roles for the International Conference on Machine Learning.

Margaret E. Connell received the B.A. degree in mathematics from Smith College, Northampton, MA, and the M.S. degree in computer science from the University of Massachusetts Amherst, in 1987.

She was a Research and Systems Programmer in the Department of Computer Science. In 2005, she retired from The Center for Intelligent Information Retrieval, part of the computer science department, where she worked on the discovery of language models of text databases, cross-lingual information retrieval, and Topic Detection and Tracking from newswire data.

Mrs. Connell is a member of Association for the Advancement of Artificial Intelligence.