, , 1–32 ()
© Kluwer Academic Publishers, Boston. Manufactured in The Netherlands.

Elevator Group Control Using Multiple Reinforcement Learning Agents

ROBERT H. CRITES AND ANDREW G. BARTO crites—barto@cs.umass.edu
Department of Computer Science, University of Massachusetts, Amherst, MA 01003

Received Feb 1, 1997

Editor: Michael Huhns and Gerhard Weiss

Abstract. Recent algorithmic and theoretical advances in reinforcement learning (RL) have attracted widespread interest. RL algorithms have appeared that approximate dynamic programming on an incremental basis. They can be trained on the basis of real or simulated experiences, focusing their computation on areas of state space that are actually visited during control, making them computationally tractable on very large problems. If each member of a team of agents employs one of these algorithms, a new *collective* learning algorithm emerges for the team as a whole. In this paper we demonstrate that such collective RL algorithms can be powerful heuristic methods for addressing large–scale control problems.

Elevator group control serves as our testbed. It is a difficult domain posing a combination of challenges not seen in most multi-agent learning research to date. We use a team of RL agents, each of which is responsible for controlling one elevator car. The team receives a global reinforcement signal which appears noisy to each agent due to the effects of the actions of the other agents, the random nature of the arrivals and the incomplete observation of the state. In spite of these complications, we show results that in simulation surpass the best of the heuristic elevator control algorithms of which we are aware. These results demonstrate the power of multi-agent RL on a very large scale stochastic dynamic optimization problem of practical utility.

Keywords: Reinforcement learning, multiple agents, teams, elevator group control, discrete event dynamic systems

1. Introduction

Interest in developing capable learning systems is increasing within the multi-agent and AI research communities (e.g., Weiss & Sen, 1996). Learning enables systems to be more flexible and robust, and it makes them better able to handle uncertainty and changing circumstances. This is especially important in multi-agent systems, where the *designers* of such systems have often faced the extremely difficult task of trying to anticipate all possible contingencies and interactions among the agents ahead of time. Much the same could be said concerning the field of decentralized control, where policies for the control stations are developed from a global vantage point, and learning does not play a role. Even though *executing* the policies depends only on the information available at each control station, the policies are *designed* in a centralized way, with access to a complete description of the problem. Research has focused on what constitutes an optimal policy under a given information pattern but not on how such policies might be *learned* under the same constraints.

Reinforcement learning (RL) (Barto & Sutton, forthcoming; Bertsekas & Tsitsiklis, 1996) applies naturally to the case of autonomous agents, which receive sensations as inputs, and take actions that affect their environment in order to achieve their own goals. RL is based on the idea that the tendency to produce an action should be strengthened (*reinforced*) if it produces favorable results, and weakened if it produces unfavorable results. This framework is appealing from a biological point of view, since an animal has certain built-in preferences (such as pleasure or pain), but does not always have a teacher to tell it exactly what action it should take in every situation.

If the members of a group of agents each employ an RL algorithm, the resulting collective algorithm allows control policies to be learned in a decentralized way. Even in situations where centralized information is available, it may be advantageous to develop control policies in a decentralized way in order to simplify the search through policy space. Although it may be possible to synthesize a system whose goals can be achieved by agents with *conflicting* objectives, this paper focuses on *teams* of agents that share *identical* objectives corresponding directly to the goals of the system as a whole.

To demonstrate the power of multi-agent RL, we focus on the difficult problem of elevator group supervisory control. Elevator systems operate in high-dimensional continuous state spaces and in continuous time as discrete event dynamic systems. Their states are not fully observable and they are non-stationary due to changing passenger arrival rates. We use a team of RL agents, each of which is responsible for controlling one elevator car. Each agent uses artificial neural networks to store its action value estimates. We compare a *parallel* architecture where the agents share the same networks with a *decentralized* architecture where the agents have their own independent networks. In either case, the team receives a global reinforcement signal which is noisy from the perspective of each agent due in part to the effects of the actions of the other agents. Despite these difficulties, our system outperforms all of the heuristic elevator control algorithms known to us. We also analyze the policies learned by the agents, and show that learning is relatively robust even in the face of increasingly incomplete state information. These results suggest that approaches to decentralized control using multi-agent RL have considerable promise.

In the following sections, we give some additional background on RL, introduce the elevator domain, describe in more detail the multi-agent RL algorithm and network architecture we used, present and discuss our results, and finally draw some conclusions. For further details on all these topics, see Crites (1996).

2. Reinforcement Learning

Both symbolic and connectionist learning researchers have focused primarily on supervised learning, where a “teacher” provides the learning system with a set of training examples in the form of input-output pairs. Supervised learning techniques are useful in a wide variety of problems involving pattern classification and function approximation. However, there are many situations in which training examples are costly or even impossible to obtain. RL is applicable in these more difficult situations, where the only help available is a "critic" that provides a scalar evaluation of the output that was selected, rather than specifying the best output or a direction of how to change the output. In RL, one faces all the difficulties of supervised learning combined with the additional difficulty of exploration, that is, determining the best output for any given input.

RL tasks can be divided naturally into two types. In non-sequential tasks, agents must learn mappings from situations to actions that maximize the expected immediate payoff. In sequential tasks, agents must learn mappings from situations to actions that maximize the expected long-term payoffs. Sequential tasks are more difficult because the actions selected by the agents may influence their future situations and thus their future payoffs. In this case, the agents interact with their environment over an extended period of time, and they need to evaluate their actions on the basis of their long-term consequences.

From the perspective of control theory, RL techniques are ways of finding approximate solutions to stochastic optimal control problems. The agent is a controller, and the environment is a system to be controlled. The objective is to maximize some performance measure over time. Given a model of the state transition probabilities and reward structure of the environment, these problems can be solved in principle using dynamic programming (DP) algorithms. However, even though DP only requires time that is polynomial in the number of states, in many problems of interest, there are so many states that the amount of time required for a solution is infeasible. Some recent RL algorithms have been designed to perform DP in an incremental manner. Unlike traditional DP, these algorithms do not require *a priori* knowledge of the state transition probabilities and reward structure of the environment and can be used to improve performance *on-line* while interacting with the environment. This on-line learning focuses computation on the areas of state space that are actually visited during control. Thus, these algorithms are a computationally tractable way of approximating DP on very large problems.

The same focusing phenomenon can also be achieved with *simulated* online training. One can often construct a simulation model without ever explicitly determining the state transition probabilities for an environment (Barto & Sutton, forthcoming; Crites & Barto, 1996). (For an example of such a simulation model, see section 3.3.) There are several advantages to this use of a simulation model if it is sufficiently accurate. It is possible to generate huge amounts of simulated experience very quickly, potentially speeding up the training process by many orders of magnitude over what would be possible using actual experience. In addition, one need not be concerned about the performance level of a simulated system during training. A successful example of simulated online training is found in Tesauro's TD-Gammon system (1992, 1994, 1995), which used RL techniques to learn to play strong master–level backgammon.

2.1. Multi-Agent Reinforcement Learning

A variety of disciplines have contributed to the study of multi-agent systems. Many researchers have focused on top-down approaches to building distributed systems, creating them from a global vantage point. One drawback to this top-down approach is the extraordinary complexity of designing such agents, since it is extremely difficult to anticipate all possible interactions and contingencies ahead of time in complex systems.

Other researchers have recently taken the opposite approach, combining large numbers of relatively unsophisticated agents in a bottom-up manner and seeing what emerges when they are put together into a group. This amounts to a sort of iterative procedure: designing a set of agents, observing their group behavior, and repeatedly adjusting the design and noting its effect on group behavior. Although such groups of simple agents often exhibit interesting and complex dynamics, there is little understanding as yet how to create bottom-up designs that can achieve complex *pre-defined* goals.

Multi-agent RL attempts to combine the advantages of both approaches. It achieves the simplicity of a bottom-up approach by allowing the use of relatively unsophisticated agents that learn on the basis of their own experiences. At the same time, RL agents adapt to a top-down global reinforcement signal, which guides their behavior toward the achievement of complex pre-defined goals. As a result, very robust systems for complex problems can be created with a minimum of human effort (Crites & Barto, 1996).

Research on multi-agent RL dates back at least to the work of the Russian mathematician Tsetlin (1973) and others from the field of learning automata, see Narendra & Thathachar (1989). A number of theoretical results have been obtained in the context of non-sequential RL. Certain types of learning automata will converge to an equilibrium point in zero-sum and non-zero-sum repeated games. See Narendra & Thathachar (1989) for details. For teams, an equilibrium point is a local maximum (an element of the game matrix that is the maximum of both its row and its column). However, in more general non-zero-sum games, equilibrium points often provide poor payoffs for all players. A good example of this is the Prisoner's Dilemma, where the only equilibrium point produces the lowest total payoff (Axelrod, 1984).

Starting in approximately 1993, a number of researchers began to investigate applying sequential RL algorithms in multi-agent contexts. Although much of the work has been in simplistic domains such as grid worlds, several interesting applications have appeared that have pointed to the promise of sequential multi-agent RL.

Markey (1994) applies parallel Q-learning to the problem of controlling a vocal tract model with 10 degrees of freedom. He discusses two architectures equivalent to the distributed and parallel architectures described in section 4.4. Each agent controls one degree of freedom in the action space, and distinguishes Q-values based only on its own action selections.

Bradtke (1993) describes some initial experiments using RL for the decentralized control of a flexible beam. The task is to efficiently damp out disturbances of a beam by applying forces at discrete locations and times. He uses 10 independent adaptive controllers distributed along the beam. Each controller attempts to minimize its own local costs and observes only its own local portion of the state information.

Dayan & Hinton (1993) propose a managerial hierarchy they call *Feudal RL*. In their scheme, higher-level managers set tasks for lower level managers, and reward them as they see fit. Since the rewards may be different at different levels of the hierarchy, this is not a team. Furthermore, only a single action selected at the lowest level actually affects the environment, so in some sense, this is a hierarchical architecture for a single agent.

Tan (1993) reports on some simple hunter-prey experiments with multi-agent RL. His focus is on the sharing of sensory information, policies, and experience among the agents.

Shoham & Tennenholtz (1993) investigate the social behavior that can emerge from agents with simple learning rules. They focus on two simple n-k-g iterative games, where n agents meet k at a time (randomly) to play game g.

Littman & Boyan (1993) describe a distributed reinforcement learning algorithm for packet routing based on the asynchronous Bellman-Ford algorithm. Their scheme uses a single Q-function, where each state entry in the Q-function is assigned to a node in the network which is responsible for storing and updating the value of that entry. This differs from most other work on distributed RL, where an entire Q-function, not just a single entry, must be stored at each node.

In addition to the multi-agent RL research concerned with team problems, a significant amount of work has focused on zero-sum games, where a single agent learns to play against an opponent. One of the earliest examples of this is Samuel's (1963) checker-playing program. A more recent example is Tesauro's TD-Gammon program (1992, 1994, 1995), which has learned to play strong Master level backgammon. These types of programs are often trained using self-play, and they can generally be viewed as single agents. Littman (1994, 1996) provides a detailed discussion of RL applied to zero-sum games, both in the case where the agents alternate their actions and where they take them simultaneously.

Very little work has been done on multi-agent RL in more general non-zero-sum games. Sandholm & Crites (1996) study the behavior of multi-agent RL in the context of the iterated prisoner's dilemma. They show that Q-learning agents are able to learn the optimal strategy against the fixed opponent Tit-for-Tat. In addition, they investigate the behavior that results when two Q-learning agents face each other.

3. Elevator Group Control

This section introduces the problem of elevator group control, which serves as our testbed for multi-agent reinforcement learning. It is a familiar problem to anyone who has ever used an elevator system, but in spite of its conceptual simplicity, it poses significant difficulties. Elevator systems operate in high–dimensional continuous state spaces and in continuous time as discrete event dynamic systems. Their states are not fully observable and they are non-stationary due to changing passenger arrival rates. An optimal policy for elevator group control is not known, so we use existing control algorithms as a standard for comparison. The elevator domain provides an opportunity to compare parallel and distributed control architectures where each agent controls one elevator car, and to monitor the amount of degradation that occurs as the agents face increasing levels of incomplete state information.

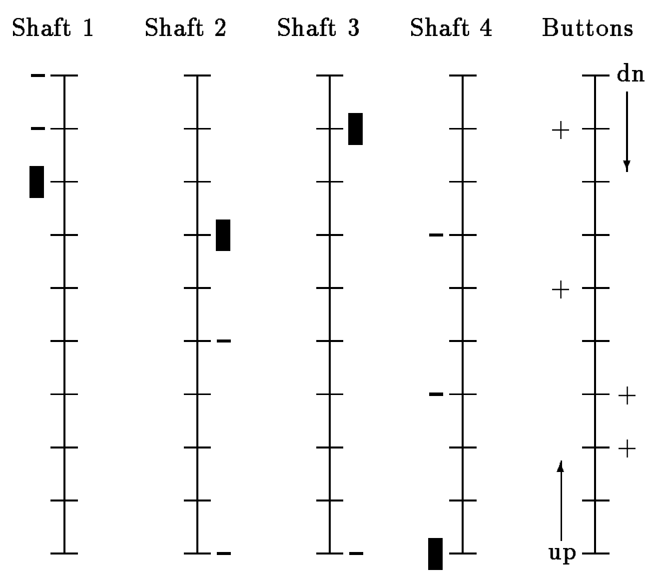

Figure 1. Elevator system schematic diagram.

A schematic diagram of an elevator system (Lewis, 1991) is presented in figure 1. The elevators cars are represented as filled boxes in the diagram. '+' represents a *hall call* or someone wanting to *enter* a car. '−' represents a *car call* or someone wanting to *leave* a car. The left side of a shaft represents upward moving cars and calls. The right side of a shaft represents downward moving cars and calls. Cars therefore move in a clockwise direction around the shafts.

Section 3.1 considers the nature of different passenger arrival patterns, and their implications. Section 3.2 reviews a variety of elevator control strategies from the literature. Section 3.3 describes the particular simulated elevator system that will be the focus in the remainder of this paper.

3.1. Passenger Arrival Patterns

Elevator systems are driven by passenger arrivals. Arrival patterns vary during the course of the day. In a typical office building, the morning rush hour brings a peak level of up traffic, while a peak in down traffic occurs during the afternoon. Other parts of the day have their own characteristic patterns. Different arrival patterns have very different effects, and each pattern requires its own analysis. Up-peak and down-peak elevator traffic are not simply equivalent patterns in opposite directions, as one might initially guess. Down-peak traffic has many arrival floors and a single destination, while up-peak traffic has a single arrival floor and many destinations. This distinction has significant implications. For example, in light up traffic, the average passenger waiting times can be kept very low by keeping idle cars at the lobby where they will be immediately available for arriving passengers. In light down traffic, waiting times will be longer since it is not possible to keep an idle car at every upper floor in the building, and therefore additional waiting time will be incurred while cars move to service hall calls. The situation is reversed in heavy traffic. In heavy up traffic, each car may fill up at the lobby with passengers desiring to stop at many different upper floors. The large number of stops will cause significantly longer round-trip times than in heavy down traffic, where each car may fill up after only a few stops at upper floors. For this reason, down-peak handling capacity is much greater than up-peak capacity. Siikonen (1993) illustrates these differences in an excellent graph obtained through extensive simulations.

Since up-peak handling capacity is a limiting factor, elevator systems are designed by predicting the heaviest likely up-peak demand in a building, and then determining a configuration that can accomodate that demand. If up-peak capacity is sufficient, then down-peak generally will be also. Up-peak traffic is the easiest type to analyze, since all passengers enter cars at the lobby, their destination floors are serviced in ascending order, and empty cars then return to the lobby. The standard capacity calculations (Strakosch, 1983; Siikonen, 1993) assume that each car leaves the lobby with M passengers (80 to 100 percent of its capacity) and that the average passenger's likelihood of selecting each destination floor is known. Then probability theory is used to determine the average number of stops needed on each round trip. From this one can estimate the average round trip time τ. The *interval* $I = \frac{\tau}{L}$ represents the average amount of time between car arrivals to the lobby, where L is the number of cars. Assuming that the cars are evenly spaced, the average waiting time is one half the interval. In reality, the average wait is somewhat longer.

The only control decisions in pure up traffic are to determine when to open and close the elevator doors at the lobby. These decisions affect how many passengers will board an elevator at the lobby. Once the doors have closed, there is really no choice about the next actions: the car calls registered by the passengers must be serviced in ascending order and the empty car must then return to the lobby. Pepyne & Cassandras (1996) show that the optimal policy for handling pure up traffic is a threshold-based policy that closes the doors after an optimal number of passengers have entered the car. The optimal threshold depends upon the traffic intensity, and may also be affected by the number of car calls already registered and by the state of the other cars. Of course, up traffic is seldom completely pure. Some method must be used for assigning any down hall calls.

More general two way traffic comes in two varieties. In two way *lobby* traffic, up-moving passengers arrive at the lobby and down-moving passengers depart at the lobby. Compared with pure up traffic, the round trip times will be longer, but more passengers will be served. In two way *interfloor* traffic, most passengers travel between floors other than the lobby. Interfloor traffic is more complex than lobby traffic in that it requires almost twice as many stops per passenger, further lengthening the round trip times.

Two way and down-peak traffic patterns require many more decisions than does pure up traffic. After leaving the lobby, a car must decide how high to travel in the building before turning, and at what floors to make additional pickups. Because more decisions are required in a wider variety of contexts, more control strategies are also possible in two way and down-peak traffic situations. For this reason, a down-peak traffic pattern was chosen as a testbed for our research. Before describing the testbed in detail, we review various elevator control strategies from the literature.

3.2. Elevator Control Strategies

The oldest relay-based automatic controllers used the principle of *collective control* (Strakosch, 1983; Siikonen, 1993), where cars always stop at the nearest call in their running direction. One drawback of this scheme is that there is no means to avoid the phenomenon called *bunching*, where several cars arrive at a floor at about the same time, making the interval, and thus the average waiting time, much longer. Advances in electronics, including the advent of microprocessors, made possible more sophisticated control policies.

The approaches to elevator control discussed in the literature generally fit into the following categories, often more than one category. Unfortunately the descriptions of the proprietary algorithms are often rather vague, since they are written for marketing purposes, and are specifically not intended to be of benefit to competitors. For this reason, it is difficult to ascertain the relative performance levels of many of these algorithms, and there is no accepted definition of the current state of the art (Ovaska, 1992).

3.2.1. Zoning Approaches

The Otis Elevator Company has used zoning as a starting point in dealing with various traffic patterns (Strakosch, 1983). Each car is assigned a zone of the building. It answers hall calls within its zone, and parks there when it is idle. The goal of the zoning approach is to keep the cars reasonably well separated and thus keep the interval down. While this approach is quite robust in heavy traffic, it gives up a significant amount of flexibility. Sakai & Kurosawa (1984) of Hitachi describe a concept called *area control* that is related to zoning. If possible, it assigns a hall call to a car that already must stop at that floor due to a car call. Otherwise, a car within an area α of the hall call is assigned if possible. The area α is a control parameter that affects both the average wait time and the power consumption.

3.2.2. Search-Based Approaches

Another control strategy is to search through the space of possible car assignments, selecting the one that optimizes some criterion such as the average waiting time. *Greedy* search strategies perform immediate call assignment, that is, they assign hall calls to cars when they are first registered, and never reconsider those assignments. Non-greedy algorithms postpone their assignments or reconsider them in light of updated information they may receive about additional hall calls or passenger destinations. Greedy algorithms give up some measure of performance due to their lack of flexibility, but also require less computation time. In western countries, an arriving car generally signals waiting passengers when it begins to decelerate (Siikonen, 1993), allowing the use of a non-greedy algorithm. The custom in Japan is to signal the car assignment immediately upon call registration. This type of signalling requires the use of a greedy algorithm.

Tobita et al (1991) of Hitachi describe a system where car assignment occurs when a hall button is pressed. They assign the car that minimizes a weighted sum of predicted wait time, travel time, and number of riders. A fuzzy rule-based system is used to pick the coefficients and estimating functions. Simulations are used to verify their effectiveness.

Receding horizon controllers are examples of non-greedy search-based approaches. After every event, they perform an expensive search for the best assignment of hall calls assuming no new passenger arrivals. Closed-loop control is achieved by re-calculating a new open-loop plan after every event. The weaknesses of this approach are its computational demands, and its lack of consideration of future arrivals. Examples of receding horizon controllers are Finite Intervisit Minimization (FIM) and Empty the System Algorithm (ESA) (Bao et al, 1994). FIM attempts to minimize squared waiting times and ESA attempts to minimize the length of the current busy period.

3.2.3. Rule-Based Approaches

In some sense, all control policies could be considered rule-based: IF situation THEN action. However, here we are more narrowly considering the type of production systems commonly used in Artificial Intelligence. Ujihara & Tsuji (1988) of Mitsubishi describe the AI-2100 system. It uses expert-system and fuzzy-logic technologies. They claim that experts in group-supervisory control have the experience and knowledge necessary to shorten waiting times under various traffic conditions, but admit that expert knowledge is fragmentary, hard to organize, and difficult to incorporate. They created a rule base by comparing the decisions made by a conventional algorithm with decisions determined by simulated annealing. The discrepancies were then analyzed by the experts, whose knowledge about solving such problems was used to create fuzzy control rules. The fuzziness lies in the IF part of the rules. Ujihara & Amano (1994) describe the latest changes to the AI-2100 system. A previous version used a fixed evaluation formula based on the current car positions and call locations. A more recent version considers future car positions and probable future hall calls. For example, one rule is IF (there is a hall call registered on an upper floor) AND (there are a large number of cars ascending towards the upper floors) THEN (assign one of the ascending cars on the basis of estimated time of arrival). Note that this is an immediate call allocation algorithm, and the consequent of this particular rule about assigning cars on the basis of estimated time of arrival bears some similarity to the greedy search-based algorithms described above.

3.2.4. Other Heuristic Approaches

The Longest Queue First (LQF) algorithm assigns upward moving cars to the longest waiting queue, and the Highest Unanswered Floor First (HUFF) algorithm assigns upward moving cars to the highest queue with people waiting (Bao et al, 1994). Both of these algorithms are designed specifically for down-peak traffic. They assign downward moving cars to any unassigned hall calls they encounter. The Dynamic Load Balancing (DLB) algorithm attempts to keep the cars evenly spaced by assigning contiguous non-overlapping sectors to each car in a way that balances their loads (Lewis, 1991). DLB is a non-greedy algorithm because it reassigns sectors after every event.

3.2.5. Adaptive and Learning Approaches

Imasaki et al (1991) of Toshiba use a fuzzy neural network to predict passenger waiting time distributions for various sets of control parameters. Their system adjusts the parameters by evaluating alternative candidate parameters with the neural network. They do not explain what control algorithm is actually used, what its parameters are, or how the network is trained.

Hitachi researchers (Fujino et al, 1992; Tobita et al, 1991) use a greedy control algorithm that combines multiple objectives such as wait time, travel time, crowding, and power consumption. The weighting of these objectives is accomplished using parameters that are tuned online. A module called the learning function unit collects traffic statistics and attempts to classify the current traffic pattern. The tuning function unit generates parameter sets for the current traffic pattern and tests them using a built-in simulator. The best parameters are then used to control the system. Searching the entire parameter space would be prohibitively expensive, so heuristics are used about which parameter sets to test.

Levy et al (1977) use dynamic programming (DP) offline to minimize the expected time needed for completion of the current busy period. No discount factor is used, since it is assumed that the values will all be finite. The major difference between this and Q-learning is that it must be performed offline since it uses a model of the transition probabilities of the system and performs sweeps of the state space. The trouble with using DP to calculate an optimal policy is that the state space is very large, requiring drastic simplification. Levy et al use several methods to keep the size of the state space manageable: they consider a building with only 2 cars and 8 floors, where the number of buttons that can be on simultaneously is restricted, the state of the buttons are restricted to binary values (i.e., elapsed times are discarded), and the cars have unlimited capacity. Construction of the transition probability matrix is the principle part of the procedure, and it assumes that the intensity of Poisson arrivals at each floor is known. Value iteration or policy iteration is then performed to obtain the solution.

Markon et al (1994) have devised a system that trains a neural network to perform immediate call allocation. There are three phases of training. In phase one, while the system is being controlled by an existing controller (the FLEX-8820 Fuzzy/AI Group Control System of Fujitec), supervised learning is used to train the network to predict the hall call service times. This first phase of training is used to learn an appropriate internal representation, i.e., weights from the input layer to the hidden layer of the network. At the end of the first phase of training, those weights are fixed. In phase two, the output layer of the network is retrained to emulate the existing controller. In phase three, single weights in the output layer of the network are perturbed, and the resulting performance is measured on a traffic sample. The weights are then modified in the direction of improved performance. This can be viewed as a form of non-sequential reinforcement learning. The single-stage reward is determined by measuring the system's performance on a traffic sample.

Their input representation uses 25 units for each car, and their output representation uses one unit for each car. Hall calls are allocated to the car corresponding to the output unit with the highest activation. They also describe a very clever way of incorporating the permutational symmetry of the problem into the architecture of their network. As they say, "If the states of two cars are interchanged, the outputs should also be interchanged." This is done by having as many sets of hidden units as there are cars, and then explicitly linking together the appropriate weights.

Their system was tested in a simulation with 6 cars and 15 floors. In a "typical building", trained on 900 passengers per hour, there was a very small improvement of around 1 second in the average wait time over the existing controller. In a more "untypical" building with uniformly distributed origin and destination floors and 1500 passengers per hour, the improvement in average wait time was almost 4 seconds.

One advantage of this system is that it can maintain an adequate service level from the beginning since it starts with a pre-existing controller. On the other hand, it is not clear whether this also may trap the controller in a suboptimal region of policy space. It would be very interesting to use this centralized, immediate call allocation network architecture as part of a *sequential* reinforcement learning algorithm.

3.3. The Elevator Testbed

The particular elevator system we study in this paper is a simulated 10-story building with 4 elevator cars. The simulator was written by Lewis (1991). Passenger arrivals at each floor are assumed to be Poisson, with arrival rates that vary during the course of the day. Our simulations use a traffic profile (Bao et al, 1994) which dictates arrival rates for every 5-minute interval during a typical afternoon down-peak rush hour. Table 1 shows the mean number of passengers arriving at *each* of floors 2 through 10 during each 5-minute interval who are headed for the lobby. In addition, there is inter-floor traffic which varies from 0% to 10% of the traffic to the lobby.

Table 1. The down–peak traffic profile.

Time	00	05	10	15	20	25	30	35	40	45	50	55
Rate	1	2	4	4	18	12	8	7	18	5	3	2

3.3.1. System Dynamics

The system dynamics are approximated by the following parameters:

- Floor time (the time to move one floor at maximum speed): 1.45 secs.
- Stop time (the time needed to decelerate, open and close the doors, and accelerate again): 7.19 secs.
- Turn time (the time needed for a stopped car to change direction): 1 sec.
- Load time (the time for one passenger to enter or exit a car): random variable from a $20th$ order truncated Erlang distribution with a range from 0.6 to 6.0 secs and a mean of 1 sec.
- Car capacity: 20 passengers.

The simulator is quite detailed, and is certainly realistic enough for our purposes. However, a few minor deviations from reality should be noted. In the simulator, a car can accelerate to full speed or decelerate from full speed in a distance of only one half of a floor, while the distances would be somewhat longer in a real system. Thus, the simulated acceleration and deceleration times are always the same, but in a real system, they will vary depending on the speed of the elevator. For example, an express car descending from the tenth floor at top speed will take longer to decelerate at the first floor than a car that is descending from the second floor. The simulator also allows the cars to commit to stopping at a floor when they are only one half of a floor away. Though this is not realistic for cars moving at top speed, the concept of making decisions regarding the next floor where the car could commit to stopping is valid.

Although the elevator cars in this system are homogeneous, the learning techniques described in this paper can also be used in more general situations, e.g., where there are several express cars or cars that only service some subset of the floors.

3.3.2. State Space

The state space is continuous because it includes the elapsed times since any hall calls were registered, which are real–valued. Even if these real values are approximated as binary values, the size of the state space is still immense. Its components include 2^{18} possible combinations of the 18 hall call buttons (up and down buttons at each landing except the top and bottom), 2^{40} possible combinations of the 40 car buttons, and 18^4 possible combinations of the positions and directions of the cars (rounding off to the nearest floor). Other parts of the state are not fully observable, for example, the exact number of passengers waiting at each floor, their exact arrival times, and their desired destinations. Ignoring everything except the configuration of the hall and car call buttons and the approximate position and direction of the cars, we obtain an extremely conservative estimate of the size of a discrete approximation to the continuous state space:

$$2^{18} \cdot 2^{40} \cdot 18^4 \approx 10^{22} \text{ states.}$$

3.3.3. Control Actions

Each car has a small set of primitive actions. If it is stopped at a floor, it must either "move up" or "move down". If it is in motion between floors, it must either "stop at the next floor" or "continue past the next floor". Due to passenger expectations, there are two constraints on these actions: a car cannot pass a floor if a passenger wants to get off there and cannot turn until it has serviced all the car buttons in its present direction. We also added three additional heuristic constraints in an attempt to build in some primitive prior knowledge: a car cannot stop at a floor unless someone wants to get on or off there, it cannot stop to pick up passengers at a floor if another car is already stopped there, and given a choice between moving up and down, it should prefer to move up (since the down-peak traffic tends to push the cars toward the bottom of the building). Because of this last constraint, the only real choices left to each car are the stop and continue actions. The actions of the elevator cars are executed asynchronously since they may take different amounts of time to complete.

3.3.4. Performance Criteria

The performance objectives of an elevator system can be defined in many ways. One possible objective is to minimize the average *wait* time, which is the time between the arrival of a passenger and his entry into a car. Another possible objective is to minimize the average *system* time, which is the sum of the wait time and the travel time. A third possible objective is to minimize the percentage of passengers that wait longer than some dissatisfaction threshold (usually 60 seconds). Another common objective is to minimize the average *squared* wait time. We chose this latter performance objective since it tends to keep the wait times low while also encouraging fair service. For example, wait times of 2 and 8 seconds have the same average (5 seconds) as wait times of 4 and 6 seconds. But the average *squared* wait times are different (34 for 2 and 8 versus 26 for 4 and 6).

4. The Algorithm and Network Architecture

This section describes the multi-agent reinforcement learning algorithm that we have applied to elevator group control. In our scheme, each agent is responsible for controlling one elevator car. Each agent uses a modification of Q-learning for discrete-event systems. Together, they employ a collective form of reinforcement learning. We begin by describing the modifications needed to extend Q-learning into a discrete-event framework, and derive a method for determining appropriate reinforcement signals in the face of uncertainty about exact passenger arrival times. Then we describe the algorithm, the feedforward networks used to store the Q-values, and the distinction between parallel and distributed versions of the algorithm.

4.1. Discrete-Event Reinforcement Learning

Elevator systems can be modeled as *discrete event* systems (Cassandras, 1993), where significant events (such as passenger arrivals) occur at discrete times, but the amount of time between events is a real-valued variable. In such systems, the constant discount factor γ used in most discrete-time reinforcement learning algorithms is inadequate. This problem can be approached using a variable discount factor that depends on the amount of time between events (Bradtke & Duff, 1995). In this case, the cost–to–go is defined as an integral rather than as an infinite sum, as follows:

$$\sum_{t=0}^{\infty} \gamma^t c_t \quad \text{becomes} \quad \int_0^{\infty} e^{-\beta\tau} c_\tau d\tau,$$

where c_t is the immediate cost at discrete time t, c_τ is the instantaneous cost at continuous time τ (the sum of the squared wait times of all currently waiting passengers), and β controls the rate of exponential decay. $\beta = 0.01$ in the experiments described in this paper. Since the wait times are measured in seconds, we scale down the instantaneous costs c_τ by a factor of 10^6 to keep the cost–to–go values from becoming exceedingly large.

Because elevator system events occur randomly in continuous time, the branching factor is effectively infinite, which complicates the use of algorithms that require explicit lookahead. Therefore, we employ a discrete event version of the Q-learning algorithm since it considers only events encountered in actual system trajectories and does not require a model of the state transition probabilities. The Q-learning update rule (Watkins, 1989) takes on the following discrete event form:

$$\Delta\hat{Q}(x,a) = \alpha \cdot [\int_{t_x}^{t_y} e^{-\beta(\tau - t_x)} c_\tau d\tau + e^{-\beta(t_y - t_x)} \min_b \hat{Q}(y,b) - \hat{Q}(x,a)],$$

where action a is taken from state x at time t_x, the next decision is required from state y at time t_y, α is the learning rate parameter, and c_τ and β are defined as above. $e^{-\beta(t_y - t_x)}$ acts as a variable discount factor that depends on the amount of time between events.

Bradtke & Duff (1995) consider the case where c_τ is constant between events. We extend their formulation to the case where c_τ is quadratic, since the goal is to minimize squared wait times. The integral in the Q-learning update rule then takes the form:

$$\int_{t_x}^{t_y} \sum_p e^{-\beta(\tau - t_x)} (\tau - t_x + w_p)^2 d\tau,$$

where w_p is the amount of time each passenger p waiting at time t_y has already waited at time t_x. (Special care is needed to handle any passengers that begin or end waiting between t_x and t_y. See section 4.2.1.)

This integral can be solved by parts to yield:

$$\sum_p e^{-\beta w_p}[\frac{2}{\beta^3} + \frac{2w_p}{\beta^2} + \frac{w_p^2}{\beta}] - e^{-\beta(w_p + t_y - t_x)}[\frac{2}{\beta^3} + \frac{2(w_p + t_y - t_x)}{\beta^2} + \frac{(w_p + t_y - t_x)^2}{\beta}].$$

A difficulty arises in using this formula since it requires knowledge of the waiting times of all waiting passengers. However, only the waiting times of passengers who press hall call buttons will be known in a real elevator system. The number of subsequent passengers to arrive and their exact waiting times will not be available. We examine two ways of dealing with this problem, which we call *omniscient* and *online* reinforcement schemes.

The simulator has access to the waiting times of all passengers. It could use this information to produce the necessary reinforcement signals. We call these *omniscient* reinforcements, since they require information that is not available in a real system. Note that it is not the controller that receives this extra information, however, but rather the *critic* that is evaluating the controller. For this reason, even if omniscient reinforcements are used during the design phase of an elevator controller on a simulated system, the resulting trained controller can be installed in a real system without requiring any extra knowledge.

The other possibility is to train using only information that would be available to a real system online. Such *online* reinforcements assume only that the waiting time of the first passenger in each queue is known (which is the elapsed button time). If the Poisson arrival rate λ for each queue is known or can be estimated, the Gamma distribution can be used to estimate the arrival times of subsequent passengers. The time until the n^{th} subsequent arrival follows the Gamma distribution $\Gamma(n, \frac{1}{\lambda})$. For each queue, subsequent arrivals will generate the following expected costs during the first b seconds after the hall button has been pressed:

$$\sum_{n=1}^{\infty} \int_0^b \text{(prob } n^{th} \text{ arrival occurs at time } \tau) \cdot \text{(cost given arrival at time } \tau)\, d\tau$$

$$= \sum_{n=1}^{\infty} \int_0^b \frac{\lambda^n \tau^{n-1} e^{-\lambda\tau}}{(n-1)!} \int_0^{b-\tau} w^2 e^{-\beta(w+\tau)} dw\, d\tau \;=\; \int_0^b \int_0^{b-\tau} \lambda w^2 e^{-\beta(w+\tau)} dw\, d\tau.$$

This integral can also be solved by parts to yield expected costs. A general solution is provided in section 4.2.2. As described in section 5.4, using online reinforcements produces results that are almost as good as those obtained with omniscient reinforcements.

4.2. Collective Discrete-Event Q-Learning

Elevator system events can be divided into two types. Events of the first type are important in the calculation of waiting times and therefore also reinforcements. These include passenger arrivals and transfers in and out of cars in the omniscient case, or hall button events in the online case. The second type are car arrival events, which are potential decision points for the RL agents controlling each car. A car that is in motion between floors generates a car arrival event when it reaches the point where it must decide whether to stop at the next floor or continue past the next floor. In some cases, cars are constrained to take a particular action, for example, stopping at the next floor if a passenger wants to get off there. An agent faces a decision point only when it has an unconstrained choice of actions.

4.2.1. Calculating Omniscient Reinforcements

Omniscient reinforcements are updated incrementally after every passenger arrival event (when a passenger arrives at a queue), passenger transfer event (when a passenger gets on or off of a car), and car arrival event (when a control decision is made). These incremental updates are a natural way of dealing with the discontinuities in reinforcement that arise when passengers begin or end waiting between a car's decisions, e.g., when another car picks up waiting passengers. The amount of reinforcement between events is the same for all the cars since they share the same objective function, but the amount of reinforcement each car receives between its *decisions* is different since the cars make their decisions asynchronously. Therefore, each car i has an associated storage location, $R[i]$, where the total discounted reinforcement it has received since its last decision (at time $d[i]$) is accumulated.

At the time of each event, the following computations are performed: Let t_0 be the time of the last event and t_1 be the time of the current event. For each passenger p that has been waiting between t_0 and t_1, let $w_0(p)$ and $w_1(p)$ be the total time that passenger p has waited at t_0 and t_1 respectively. Then for each car i,

$$\Delta R[i] = \sum_p e^{-\beta(t_0 - d[i])}(\frac{2}{\beta^3} + \frac{2w_0(p)}{\beta^2} + \frac{w_0^2(p)}{\beta}) - e^{-\beta(t_1 - d[i])}(\frac{2}{\beta^3} + \frac{2w_1(p)}{\beta^2} + \frac{w_1^2(p)}{\beta}).$$

4.2.2. Calculating Online Reinforcements

Online reinforcements are updated incrementally after every hall button event (signaling the arrival of the first waiting passenger at a queue or the arrival of a car to pick up any waiting passengers at a queue) and car arrival event (when a control decision is made). We assume that online reinforcements caused by passengers waiting at a queue end immediately when a car arrives to service the queue, since it is not possible to know exactly when each passenger boards a car. The Poisson arrival rate λ for each queue is estimated as the reciprocal of the last inter-button time for that queue, i.e., the amount of time from the last service until the button was pushed again. However, a ceiling of $\hat{\lambda} \leq 0.04$ passengers per second is placed on the estimated arrival rates to prevent any very small inter-button times from creating huge penalties that might destabilize the cost–to–go estimates.

At the time of each event, the following computations are performed: Let t_0 be the time of the last event and t_1 be the time of the current event. For each hall call button b that was active between t_0 and t_1, let $w_0(b)$ and $w_1(b)$ be the elapsed time of button b at t_0 and t_1 respectively. Then for each car i,

$$\begin{aligned}\Delta R[i] \;=\; & e^{-\beta(t_0 - d[i])} \sum_b \{ \frac{2\hat{\lambda}_b(1 - e^{-\beta(t_1 - t_0)})}{\beta^4} + \\ & (\frac{2}{\beta^3} + \frac{2w_0(b)}{\beta^2} + \frac{w_0^2(b)}{\beta}) - e^{-\beta(t_1 - t_0)}(\frac{2}{\beta^3} + \frac{2w_1(b)}{\beta^2} + \frac{w_1^2(b)}{\beta}) + \\ & \hat{\lambda}_b[(\frac{2w_0(b)}{\beta^3} + \frac{w_0^2(b)}{\beta^2} + \frac{w_0^3(b)}{3\beta}) - e^{-\beta(t_1 - t_0)}(\frac{2w_1(b)}{\beta^3} + \frac{w_1^2(b)}{\beta^2} + \frac{w_1^3(b)}{3\beta})]\}.\end{aligned}$$

4.2.3. Making Decisions and Updating Q-Values

A car that is in motion between floors generates a car arrival event when it reaches the point where it must decide whether to stop at the next floor or continue past the next floor. In some cases, cars are constrained to take a particular action, for example, stopping at the next floor if a passenger wants to get off there. An agent faces a decision point only when it has an unconstrained choice of actions. The algorithm used by each agent for making decisions and updating its Q-value estimates is as follows:

1. At time t_x, observing state x, car i arrives at a decision point. It selects an action a using the Boltzmann distribution over its Q-value estimates:

$$Pr(stop) = \frac{e^{Q(x,cont)/T}}{e^{Q(x,stop)/T} + e^{Q(x,cont)/T}},$$

   where T is a positive "temperature" parameter that is "annealed" or decreased during training. The value of T controls the amount of randomness in the selection of actions. At the beginning of training, when the Q-value estimates are very inaccurate, high values of T are used, which give nearly equal probabilities to each action. Later in training, when the Q-value estimates are more accurate, lower values of T are used, which give higher probabilities to actions that are thought to be superior, while still allowing some exploration to gather more information about the other actions. As discussed in section 5.3, choosing a slow enough annealing schedule is particularly important in multi-agent settings.

2. Let the next decision point for car i be at time t_y in state y. After *all* cars (including car i) have updated their $R[\cdot]$ values as described in the last two sections, car i adjusts its estimate of $Q(x, a)$ toward the following target value:

$$R[i] + e^{-\beta(t_y - t_x)} \min_{\{stop,cont\}} \hat{Q}(y, \cdot).$$

   Car i then resets its reinforcement accumulator $R[i]$ to zero.

3. Let $x \leftarrow y$ and $t_x \leftarrow t_y$. Go to step 1.

4.3. The Networks Used to Store the Q-Values

Using lookup tables to store the Q-values was ruled out for such a large system. Instead, we used feedforward neural networks trained with the error backpropagation algorithm (Rumelhart et al, 1986). The networks receive some of the state information as input, and produce Q-value estimates as output. The Q-value estimates can be written as $\hat{Q}(x, a, \phi)$, where ϕ is a vector of the parameters or weights of the networks. The exact weight update equation is:

$$\Delta\phi = \alpha[R[i] + e^{-\beta(t_y - t_x)} \min_{\{stop,cont\}} \hat{Q}(y, \cdot, \phi) - \hat{Q}(x, a, \phi)] \bigtriangledown_\phi \hat{Q}(x, a, \phi),$$

where α is a positive learning rate or stepsize parameter, and the gradient $\bigtriangledown_\phi \hat{Q}(x, a, \phi)$ is the vector of partial derivatives of $\hat{Q}(x, a, \phi)$ with respect to each component of ϕ.

At the start of training, the weights of each network are initialized to be uniform random numbers between -1 and $+1$. Some experiments in this paper use separate single-output networks for each action–value estimate, while others use one network with multiple output units, one for each action. Our basic network architecture for pure down traffic uses 47 input units, 20 hidden sigmoid units, and 1 or 2 linear output units. The input units are as follows:

- 18 units: Two units encode information about each of the nine down hall buttons. A real–valued unit encodes the elapsed time if the button has been pushed and a binary unit is on if the button has not been pushed.
- 16 units: Each of these units represents a possible location and direction for the car whose decision is required. Exactly one of these units will be on at any given time. Note that each car has a different egocentric view of the state of the system.
- 10 units: These units each represent one of the 10 floors where the other cars may be located. Each car has a "footprint" that depends on its direction and speed. For example, a stopped car causes activation only on the unit corresponding to its current floor, but a moving car causes activation on several units corresponding to the floors it is approaching, with the highest activations on the closest floors. No information is provided about *which one* of the other cars is at a particular location.
- 1 unit: This unit is on if the car whose decision is required is at the highest floor with a waiting passenger.
- 1 unit: This unit is on if the car whose decision is required is at the floor with the passenger that has been waiting for the longest amount of time.
- 1 unit: The bias unit is always on.

In section 4, we introduce other representations, including some with more restricted state information.

4.4. Parallel and Distributed Implementations

Each elevator car is controlled by a separate Q-learning agent. We experimented with both parallel and decentralized implementations. In parallel implementations, the agents use a central set of shared networks, allowing them to learn from each other's experiences, but forcing them to learn identical policies. In totally decentralized implementations, the agents have their own networks, allowing them to specialize their control policies. In either case, none of the agents is given explicit access to the actions of the other agents. Cooperation has to be learned indirectly via the global reinforcement signal. Each agent faces added stochasticity and nonstationarity because its environment contains other learning agents.

5. Results and Discussion

5.1. Basic Results Versus Other Algorithms

Since an optimal policy for the elevator group control problem is unknown, we measured the performance of our algorithm against other heuristic algorithms, including the best of which we were aware. The algorithms were: SECTOR, a sector–based algorithm similar to what is used in many actual elevator systems; DLB, Dynamic Load Balancing, attempts to equalize the load of all cars; HUFF, Highest Unanswered Floor First, gives priority to the highest floor with people waiting; LQF, Longest Queue First, gives priority to the queue with the person who has been waiting for the longest amount of time; FIM, Finite Intervisit Minimization, a receding horizon controller that searches the space of admissible car assignments to minimize a load function; ESA, Empty the System Algorithm, a receding horizon controller that searches for the fastest way to "empty the system" assuming no new passenger arrivals. FIM is very computationally intensive, and would be difficult to implement in real time in its present form. ESA uses queue length information that would not be available in a real elevator system. ESA/nq is a version of ESA that uses arrival rate information to estimate the queue lengths. For more details, see Bao et al (1994). RLp and RLd denote the RL controllers, parallel and decentralized. The RL controllers were each trained on 60,000 hours of simulated elevator time, which took four days on a 100 MIPS workstation. The results for all the algorithms were averaged over 30 hours of simulated elevator time to ensure their statistical significance. The average waiting times listed below for the trained RL algorithms are correct to within ± 0.13 at a 95% confidence level, the average squared waiting times are correct to within ± 5.3, and the average system times are correct to within ± 0.27. Table 2 shows the results for the traffic profile with down traffic only.

Table 3 shows the results for the down–peak traffic profile with up and down traffic, including an average of 2 up passengers per minute at the lobby. The algorithm was trained on down–only traffic, yet it generalizes well when up traffic is added and upward moving cars are forced to stop for any upward hall calls.

Table 4 shows the results for the down–peak traffic profile with up and down traffic, including an average of 4 up passengers per minute at the lobby. This time there is twice as much up traffic, and the RL agents generalize extremely well to this new situation.

Table 2. Results for down–peak profile with down traffic only.

Algorithm	AvgWait	SquaredWait	SystemTime	Percent>60 secs
SECTOR	21.4	674	47.7	1.12
DLB	19.4	658	53.2	2.74
BASIC HUFF	19.9	580	47.2	0.76
LQF	19.1	534	46.6	0.89
HUFF	16.8	396	48.6	0.16
FIM	16.0	359	47.9	0.11
ESA/nq	15.8	358	47.7	0.12
ESA	15.1	338	47.1	0.25
RLp	14.8	320	41.8	0.09
RLd	14.7	313	41.7	0.07

Table 3. Results for down–peak profile with up and down traffic.

Algorithm	AvgWait	SquaredWait	SystemTime	Percent>60 secs
SECTOR	27.3	1252	54.8	9.24
DLB	21.7	826	54.4	4.74
BASIC HUFF	22.0	756	51.1	3.46
LQF	21.9	732	50.7	2.87
HUFF	19.6	608	50.5	1.99
ESA	18.0	524	50.0	1.56
FIM	17.9	476	48.9	0.50
RLp	16.9	476	42.7	1.53
RLd	16.9	468	42.7	1.40

Table 4. Results for down–peak profile with twice as much up traffic.

Algorithm	AvgWait	SquaredWait	SystemTime	Percent>60 secs
SECTOR	30.3	1643	59.5	13.50
HUFF	22.8	884	55.3	5.10
DLB	22.6	880	55.8	5.18
LQF	23.5	877	53.5	4.92
BASIC HUFF	23.2	875	54.7	4.94
FIM	20.8	685	53.4	3.10
ESA	20.1	667	52.3	3.12
RLd	18.8	593	45.4	2.40
RLp	18.6	585	45.7	2.49

One can see that both the RL systems achieved very good performance, most notably as measured by system time (the sum of the wait and travel time), a measure that was not directly being minimized. Surprisingly, the decentralized RL system was able to achieve as good a level of performance as the parallel RL system.

5.2. Analysis of Decentralized Results

In view of the outstanding success of the decentralized RL algorithm, several questions suggest themselves: How similar are the policies that the agents have learned to one another and to the policy learned by the parallel algorithm? Can the results be improved by using a voting scheme? What happens if one agent's policy is used to control all of the cars? This section addresses all of these questions.

First the simulator was modified to poll each of the four decentralized Q-network agents as well as the parallel Q-network on every decision by every car, and compare their action selections. During one hour of simulated elevator time, there were a total of 573 decisions required. The four agents were unanimous on 505 decisions (88 percent), they split 3 to 1 on 47 decisions (8 percent), and they split evenly on 21 decisions (4 percent). The parallel network agreed with 493 of the 505 unanimous decisions (98 percent). For some reason, the parallel network tended to favor the STOP action more than the decentralized networks, though that apparently had little impact on the overall performance. The complete results are listed in table 5.

Table 5. Amount of agreement between decentralized agents.

Agents Saying STOP	Agents Saying CONTINUE	Number of Instances	Parallel Says STOP	Parallel Says CONT
4	0	389	386	3
3	1	29	27	2
2	2	21	17	4
1	3	18	11	7
0	4	116	9	107

While these results show considerable agreement, there are a minority of situations where the agents disagree. In the next experiment the agents vote on which actions should be selected for all of the cars. In the cases where the agents are evenly split, we examine three ways of resolving the ties: in favor of STOP (RLs), in favor of CONTINUE (RLc), or randomly (RLr). The following table shows the results of this voting scheme compared to the original decentralized algorithm (RLd). The results are averaged over 30 hours of simulated elevator time on pure down traffic.

These results show no significant improvement from voting. In the situations where the agents were evenly split, breaking the ties randomly produced results that were almost identical to those of the original decentralized algorithm. This seems to imply that the agents generally agree on the most important decisions,

Table 6. Comparison with several voting schemes.

Algorithm	AvgWait	SquaredWait	SystemTime	Percent>60 secs
RLc	15.0	325	41.7	0.09
RLs	14.9	322	41.7	0.10
RLr	14.8	314	41.7	0.12
RLd	14.7	313	41.7	0.07

and disagree only on decisions of little consequence where the action values are very similar.

In the next experiment the agent for a single car selects actions for all the cars. RL1 uses the agent for car 1 to control all the cars, RL2 uses the agent for car 2, and so on. The following table compares these controllers to the original decentralized algorithm (RLd). The results are averaged over 30 hours of simulated elevator time on pure down traffic.

Table 7. Letting a single agent control all four cars.

Algorithm	AvgWait	SquaredWait	SystemTime	Percent>60 secs
RL1	14.7	315	41.6	0.15
RL2	15.0	324	41.9	0.10
RL3	15.0	333	41.9	0.26
RL4	15.0	324	41.8	0.15
RLd	14.7	313	41.7	0.07

While agent 1 outperformed the other agents, all of the agents performed well relative to the non–RL controllers discussed above. In summary, it appears that all the decentralized and parallel agents learned very similar policies. The similarity of the learned policies may have been caused in part by the symmetry of the elevator system and the input representation we selected, which did not distinguish among the cars. For future work, it would be interesting to see whether agents with input representations that *did* distinguish among the cars would still arrive at similar policies.

5.3. Annealing Schedules

One of the most important factors in the performance of the algorithms is the annealing schedule used to control the amount of exploration performed by each agent. The slower the annealing process, the better the final result. This is illustrated in table 8 and figure , which show the results of one training run with each of a number of annealing rates. The temperature T was annealed according to the schedule: $T = 2.0 * (Factor)^h$, where h represents the hours of training completed.

Once again, the results were measured over 30 hours of simulated elevator time. Even though they are somewhat noisy due to not being averaged over multiple training runs, the trend is still quite clear.

Each of the schedules that we tested shared the same starting and ending temperatures. Although the annealing process can be ended at any time with the current Q-value estimates being used to determine a control policy, if the amount of time available for training is known in advance, one should select an annealing schedule that covers a full range of temperatures.

Table 8. The effect of varying the annealing rate.

Factor	Hours	AvgWait	SquaredWait	SystemTime	Pct>60 secs
0.992	950	19.3	581	44.7	1.69
0.996	1875	17.6	487	43.2	1.17
0.998	3750	15.8	376	42.0	0.28
0.999	7500	15.3	353	41.8	0.30
0.9995	15000	15.7	361	42.4	0.17
0.99975	30000	15.1	335	41.9	0.12
0.999875	60000	14.7	313	41.7	0.07

While gradual annealing is important in single–agent RL, it is even more important in multi–agent RL. The tradeoff between exploration and exploitation for an agent now must also be balanced with the need for other agents to learn in a stationary environment and while that agent is doing its best. At the beginning of the learning process, the agents are all extremely inept. With gradual annealing they are all able to raise their performance levels in parallel. Tesauro (1992, 1994, 1995) notes a slightly different but related phenomenon in the context of zero–sum games, where training with self-play allows an agent to learn with a well–matched opponent during each stage of its development.

5.4. Omniscient Versus Online Reinforcements

This section examines the relative performance of the omniscient and online reinforcements described in section 4.1, given the same network structure and temperature and learning rate schedule. As shown in table 9, omniscient reinforcements led to slightly better performance than online reinforcements. This should be of little concern regarding the application of RL to a real elevator system, since one would want to perform the initial training in simulation in any case, not only because of the huge amount of experience needed, but also because performance would be poor during the early stages of training. In a real elevator system, the initial training could be performed using a simulator, and the networks could be fine-tuned on the real system.

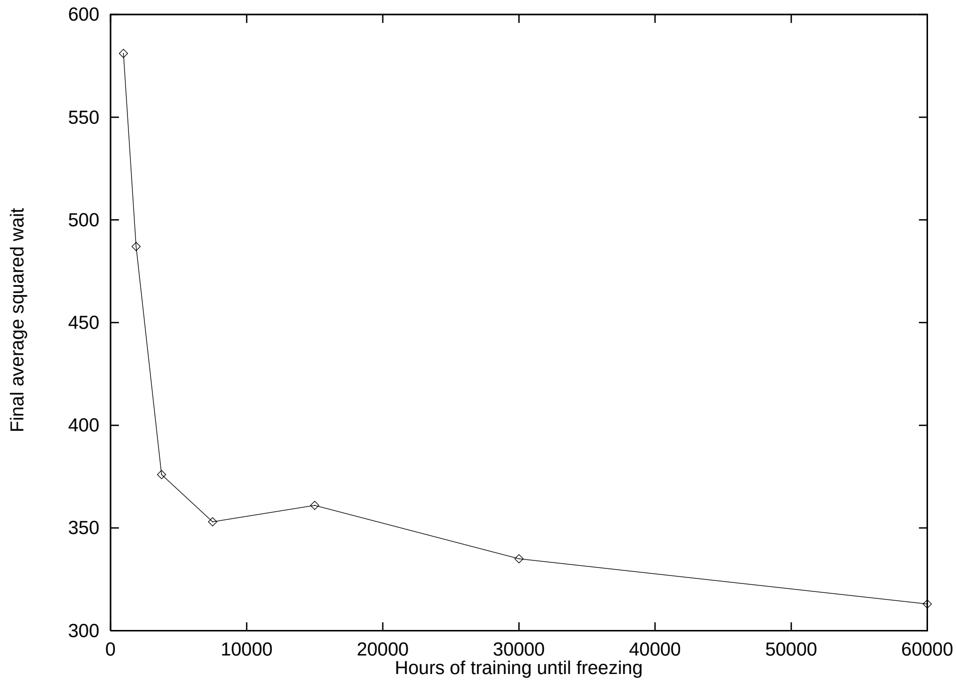

Figure 2. The effect of varying the annealing rate.

Table 9. Omniscient versus online reinforcements.

	AvgWait	SquaredWait	SystemTime	Pct>60 secs
Omniscient	15.2	332	42.1	0.07
Online	15.3	342	41.6	0.16

5.5. Levels of Incomplete State Information

If parallel or decentralized RL were to be implemented in a real elevator system, there would be no problem providing whatever state information was available to all of the agents. However, in a truly decentralized control situation, this might not be possible. This section looks at how performance degrades as the agents receive less state information.

In these experiments, the amount of information available to the agents was varied along two dimensions: information about the hall call buttons, and information about the location, direction, and status of the other cars.

The input representations for the hall call buttons were: REAL, consisting of 18 input units, where two units encode information about each of the nine down hall buttons. A real-valued unit encodes the elapsed time if the button has been pushed and a binary unit is on if the button has not been pushed; BINARY, consisting of 9 binary input units corresponding to the nine down hall buttons; QUANTITY, consisting of two input units measuring the number of hall calls above and below the current decision-making car, and NONE, with no input units conveying information about the hall buttons.

The input representations for the configuration of the other cars were: FOOTPRINTS, consisting of 10 input units, where each unit represents one of the 10 floors where the other cars may be located. Each car has a "footprint" that depends on its direction and speed. For example, a stopped car causes activation only on the unit corresponding to its current floor, but a moving car causes activation on several units corresponding to the floors it is approaching, with the highest activations on the closest floors. Activations caused by the various cars are additive; QUANTITY, consisting of 4 input units that represent the number of upward and downward moving cars above and below the decision-making car; and NONE, consisting of no input units conveying information about the hall buttons.

All of the networks also possessed a bias unit that was always activated, 20 hidden units, and 2 output units (for the STOP and CONTINUE actions). All used the decentralized RL algorithm, trained for 12000 hours of simulated elevator time using the down–peak profile and omniscient reinforcements. The temperature T was annealed according to the schedule: $T = 2.0 * (.9995)^h$, where h is the hours of training. The learning rate parameter was decreased according to the schedule: $LR = 0.01 * (.99975)^h$.

The results shown in table 10 are measured in terms of the average squared passenger waiting times over 30 hours of simulated elevator time. They should be considered to be fairly noisy because they were not averaged over multiple training runs. Nevertheless, they show some interesting trends.

Table 10. Average squared wait times with various levels of incomplete state information.

Hall Buttons	Location of Other Cars		
	Footprints	Quantity	None
Real	370	428	474
Binary	471	409	553
Quantity	449	390	530
None	1161	778	827

Clearly, information about the hall calls was more important than information about the configuration of the other cars. In fact, performance was still remarkably good even without any information about the other cars. (Technically speaking, some information was always available about the other cars because of the constraint that prevents a car from stopping to pick up passengers at a floor where another car has already stopped. No doubt this constraint helped performance considerably.)

When the hall call information was completely missing, the network weights had an increased tendency to become unstable or grow without bound and so the learning rate parameter had to be lowered in some cases. For a further discussion of network instability, see section 5.7.

The way that information was presented was important. For example, being supplied with the number of hall calls above and below the decision-making car was more useful to the networks than the potentially more informative binary button information. It also appears that information along one dimension is helpful in utilizing information along the other dimension. For example, the FOOTPRINTS representation made performance much worse than no car information in the absence of any hall call information. The only time FOOTPRINTS outperformed the other representations was with the maximum amount of hall call information.

Overall, the performance was quite good except in the complete absence of hall call information (which is a significant handicap indeed), and it could be improved further by slower annealing. It seems reasonable to say that the algorithm degrades gracefully in the presence of incomplete state information in this problem.

In a final experiment, two binary features were added to the REAL/FOOTPRINTS input representation. They were activated when the decision-making car was at the highest floor with a waiting passenger, and the floor with the longest waiting passenger, respectively. With the addition of these features, the average squared wait time decreased from 370 to 359, so they appear to have some value.

5.6. Practical Issues

One of the biggest difficulties in applying RL to the elevator control problem was finding the correct temperature and learning rate parameters. It was very helpful to start with a scaled down version consisting of 1 car and 4 floors and a lookup table for the Q-values. This made it easier to determine rough values for the temperature and learning rate schedules.

The importance of focusing the experience of the learner into the most appropriate areas of the state space cannot be overstressed. Training with trajectories of the system is an important start, but adding reasonable constraints such as those described in section 3.3.3 also helps. Further evidence supporting the importance of focusing is that given a choice between training on heavier or lighter traffic than one expects to face during testing, it is better to train on the heavier traffic. This type of training gives the system more experience with states where the queue lengths are long and thus where making the correct decision is crucial.

5.7. Instability

The weights of the neural networks can become unstable, their magnitude increasing without bound. Two particular situations seem to lead to instability. The first occurs when the learning algorithm makes updates that are too large. This can happen when the learning rate is too large, or when the network inputs are too large (which can happen in very heavy traffic situations), or both. The second occurs when the network weights have just been initialized to random values, producing excessively inconsistent Q-values. For example, while a learning rate of 10^{-2} is suitable for training a random initial network on moderate traffic (700 passengers/hour), it very consistently brings on instability in heavy traffic (1900 passengers/hour). However, a learning rate of 10^{-3} keeps the network stable even in heavy traffic. If we train the network this way for several hundred hours of elevator time, leading to weights that represent a more consistent set of Q-values, then the learning rate can be safely raised back up to 10^{-2} without causing instability.

5.8. Linear Networks

One may ask whether nonlinear function approximators such as feedforward sigmoidal networks are necessary for good performance in this elevator control problem. A test was run using a linear network trained with the delta rule. The linear network had a much greater tendency to be unstable. In order to keep the weights from blowing up, the learning rate had to be lowered by several orders of magnitude, from 10^{-3} to 10^{-6}. After some initial improvement, the linear network was unable to further reduce the average TD error, resulting in extremely poor performance. This failure of linear networks lends support to the contention that elevator control is a difficult problem.

6. Discussion

Both the parallel and distributed multi-agent RL architectures were able to outperform all of the elevator algorithms they were tested against. The two architectures learned very similar policies. Gradual annealing appeared to be a crucial factor in their success. Training was accomplished effectively using both omniscient and online reinforcements. The algorithms were robust, easily generalizing to new situations such as added up traffic. Finally, they degraded gracefully in the face of increasing levels of incomplete state information. Although the networks became unstable under certain circumstances, techniques were discussed that prevented the instabilities in practice. Taken together, these results demonstrate that multi–agent RL algorithms are very powerful techniques for addressing very large scale stochastic dynamic optimization problems.

A crucial ingredient in the success of multi-agent RL is a careful control of the amount of exploration performed by each agent. Exploration in this context means trying an action believed to be sub-optimal in order to gather additional information about its potential value. At the beginning of the learning process, each RL agent chooses its actions randomly, without any knowledge of their relative values, and thus all the agents are extremely inept. However, in spite of the noise in the reinforcement signal caused by the actions of the other agents, some actions will begin to appear to be better than others. By gradually annealing (or lowering) the amount of exploration performed by the agents, these better actions will be taken with greater frequency. This gradually changes the environment for each of the agents, and as they continue to explore, they all raise their performance levels in parallel. Even though RL agents in a team face added stochasticity and non-stationarity due to the changing stochastic policies of the other agents on the team, they display an exceptional ability to cooperate with one another in learning to maximize their rewards.

There are many areas of research in both elevator group control and general multi-agent RL that deserve further investigation. Implementing an RL controller in a real elevator system would require training on several other traffic profiles, including up-peak and inter-floor traffic patterns. Additional actions would be needed in order to handle these traffic patterns. For example, in up-peak traffic it would be useful to have actions to specifically open and close the doors or to control the dwell time at the lobby. In inter-floor traffic, unconstrained "up" and "down" actions would be needed for the sake of flexibility. The cars should also have the ability to "park" at various floors during periods of light traffic.

It would be interesting to try something other than a uniform annealing schedule for the agents. For example, a coordinated exploration strategy or round-robin type of annealing might be a way of reducing the noise generated by the other agents. However, such a coordinated exploration strategy may have a greater tendency to become stuck in sub-optimal policies.

Theoretical results for sequential multi-agent RL are needed to supplement the results for non-sequential multi-agent RL described in section 2.1. Another area that needs further study is RL architectures where reinforcements are tailored to individual agents, possibly by using a hierarchy or some other advanced organizational structure. Such local reinforcement architectures have the potential to greatly increase the speed of learning, but they will require much more knowledge on the part of whatever is producing the reinforcement signals (Barto, 1989). Finally, it is important to find effective methods of allowing the possibility of explicit communication among the agents.

7. Conclusions

Multi-agent control systems are often required because of spatial or geographic distribution, or in situations where centralized information is not available or is not practical. But even when a distributed approach is not required, multiple agents may still provide an excellent way of scaling up to approximate solutions for very large problems by streamlining the search through the space of possible policies.

Multi-agent RL combines the advantages of bottom-up and top-down approaches to the design of multi-agent systems. It achieves the simplicity of a bottom-up approach by allowing the use of relatively unsophisticated agents that learn on the basis of their own experiences. At the same time, RL agents adapt to a top-down global reinforcement signal, which guides their behavior toward the achievement of complex specific goals. As a result, very robust systems for complex problems can be created with a minimum of human effort.

RL algorithms can be trained using actual or simulated experiences, allowing them to focus computation on the areas of state space that are actually visited during control, making them computationally tractable on very large problems. If each of the members of a team of agents employs an RL algorithm, a new *collective* algorithm emerges for the group as a whole. This type of collective algorithm allows control policies to be learned in a decentralized way. Even though RL agents in a team face added stochasticity and non-stationarity due to the changing stochastic policies of the other agents on the team, they display an exceptional ability to cooperate with one another in maximizing their rewards.

In order to demonstrate the power of multi-agent RL, we focused on the difficult problem of elevator group supervisory control. We used a team of RL agents, each of which was responsible for controlling one elevator car. Results obtained in simulation surpassed the best of the heuristic elevator control algorithms of which we are aware. Performance was also very robust in the face of increased levels of incomplete state information.

Acknowledgments

We thank John McNulty, Christos Cassandras, Asif Gandhi, Dave Pepyne, Kevin Markey, Victor Lesser, Rod Grupen, Rich Sutton, Steve Bradtke, and the ANW group for assistance with the simulator and for helpful discussions. This research was supported by the Air Force Office of Scientific Research under grant F49620–93–1–0269.

References

R. M. Axelrod. *The Evolution of Cooperation.* Basic Books, New York, 1984.

G. Bao, C. G. Cassandras, T. E. Djaferis, A. D. Gandhi, and D. P. Looze. Elevator dispatchers for down peak traffic. ECE Department Technical Report, University of Massachusetts, 1994.

A. G. Barto. From chemotaxis to cooperativity: Abstract exercises in neuronal learning strategies. In R. Durbin, C. Miall, and G. Mitchison, editors, *The Computing Neuron.* Addison-Wesley, Wokingham, England, 1989.

A. G. Barto and R. S. Sutton. *Learning By Interaction: An Introduction to Modern Reinforcement Learning.* Forthcoming.

D. P. Bertsekas and J. N. Tsitsiklis. *Neuro–Dynamic Programming.* Athena Scientific Press, Belmont, MA, 1996.

S. J. Bradtke. Distributed adaptive optimal control of flexible structures, 1993. Unpublished manuscript.

S. J. Bradtke and M. O. Duff. Reinforcement learning methods for continuous-time Markov decision problems. In G. Tesauro, D. Touretzky, and T. Leen, editors, *Advances in Neural Information Processing Systems 7.* MIT Press, Cambridge, MA, 1995.

C. G. Cassandras. *Discrete Event Systems: Modeling and Performance Analysis.* Aksen Associates, Homewood, IL, 1993.

R. H. Crites. *Large-Scale Dynamic Optimization Using Teams of Reinforcement Learning Agents.* PhD thesis, University of Massachusetts, September 1996.

R. H. Crites and A. G. Barto. Forming control policies from simulation models using reinforcement learning. In *Proceedings of the Ninth Yale Workshop on Adaptive and Learning Systems.* 1996.

R. H. Crites and A. G. Barto. Improving elevator performance using reinforcement learning. In D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, editors, *Advances in Neural Information Processing Systems 8.* MIT Press, Cambridge, MA, 1996.

P. Dayan and G. E. Hinton. Feudal reinforcement learning. In S. J. Hanson, J. D. Cowan, and C. L. Giles, editors, *Advances in Neural Information Processing Systems 5.* Morgan Kaufmann, San Mateo, CA, 1993.

A. Fujino, T. Tobita, and K. Yoneda. An on-line tuning method for multi-objective control of elevator group. In *Proceedings of the International Conference on Industrial Electronics, Control, Instrumentation, and Automation*, pages 795–800, 1992.

N. Imasaki, J. Kiji, and T. Endo. A fuzzy neural network and its application to elevator group control. In T. Terano, M. Sugeno, M. Mukaidono, and K. Shigemasu, editors, *Fuzzy Engineering Toward Human Friendly Systems.* IOS Press, Amsterdam, 1992.

D. Levy, M. Yadin, and A. Alexandrovitz. Optimal control of elevators. *International Journal of Systems Science*, 8:301–320, 1977.

J. Lewis. *A Dynamic Load Balancing Approach to the Control of Multiserver Polling Systems with Applications to Elevator System Dispatching.* Ece department, University of Massachusetts, September 1991.

M. Littman and J. Boyan. A distributed reinforcement learning scheme for network routing. Technical Report CMU-CS-93-165, Carnegie Mellon University, 1993.

M. L. Littman. Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the Eleventh International Conference on Machine Learning.* Morgan Kaufmann, 1994.

M. L. Littman. *Algorithms for Sequential Decision Making.* PhD thesis, Brown University, March 1996.

K. L. Markey. Efficient learning of multiple degree-of-freedom control problems with quasi-independent Q-agents. In M. C. Mozer, P. Smolensky, D. S. Touretzky, J. L. Elman, and A. S. Weigend, editors, *Proceedings of the 1993 Connectionist Models Summer School.* Erlbaum Associates, Hillsdale, NJ, 1994.

S. Markon, H. Kita, and Y. Nishikawa. Adaptive optimal elevator group control by use of neural networks. *Transactions of the Institute of Systems, Control, and Information Engineers*, 7:487–497, 1994.

K. S. Narendra and M. A. L. Thathachar. *Learning Automata: An Introduction.* Prentice-Hall, Englewood Cliffs, NJ, 1989.

S. J. Ovaska. Electronics and information technology in high-range elevator systems. *Mechatronics*, 2:89–99, 1992.

D. L. Pepyne and C. G. Cassandras. Optimal dispatching control for elevator systems during uppeak traffic. Submitted to CDC-96.

D. E. Rumelhart, J. L. McClelland, and the PDP Research Group. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition.* MIT Press, Cambridge, MA, 1986.

Y. Sakai and K. Kurosawa. Development of elevator supervisory group control system with artificial intelligence. *Hitachi Review*, 33:25–30, 1984.

A. L. Samuel. Some studies in machine learning using the game of checkers. In E. Feigenbaum and J. Feldman, editors, *Computers and Thought.* McGraw-Hill, New York, 1963.

T. W. Sandholm and R. H. Crites. Multiagent reinforcement learning in the iterated prisoner's dilemma. *Biosystems*, 37:147–166, 1996.

Y. Shoham and M. Tennenholtz. Co-learning and the evolution of coordinated multi-agent activity. 1993.

M. L. Siikonen. Elevator traffic simulation. *Simulation*, 61:257–267, 1993.

G. R. Strakosch. *Vertical Transportation: Elevators and Escalators.* Wiley and Sons, New York, 1983.

M. Tan. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the Tenth International Conference on Machine Learning*, 1993.

G. Tesauro. Practical issues in temporal difference learning. *Machine Learning*, 8:257–277, 1992.

G. Tesauro. TD-Gammon, a self-teaching backgammon program, achieves master-level play. *Neural Computation*, 6:215–219, 1994.

G. Tesauro. Temporal difference learning and TD-Gammon. *Communications of the ACM*, 38:58–68, 1995.

T. Tobita, A. Fujino, H. Inaba, K. Yoneda, and T. Ueshima. An elevator characterized group supervisory control system. In *Proceedings of IECON*, pages 1972–1976, 1991.

M. L. Tsetlin. *Automaton Theory and Modeling of Biological Systems.* Academic Press, New York, NY, 1973.

H. Ujihara and M. Amano. The latest elevator group-control system. *Mitsubishi Electric Advance*, 67:10–12, 1994.

H. Ujihara and S. Tsuji. The revolutionary AI-2100 elevator-group control system and the new intelligent option series. *Mitsubishi Electric Advance*, 45:5–8, 1988.

C. J. C. H. Watkins. *Learning from Delayed Rewards.* PhD thesis, Cambridge University, 1989.

G. Weiss and S. Sen. *Adaptation and Learning in Multi-Agent Systems.* Springer Verlag, Berlin, 1996. Lecture Notes in Artificial Intelligence, Volume 1042.

Received Date
Accepted Date
Final Manuscript Date