Agent-Based Elevator Control

Clark Moody
CSCE 631, Project 2

March 29, 2011

1 Introduction

This project seeks to find an agent-based controller that enhances the performance of a multi-elevator system. A common simulation framework simplifies the problem and makes results easily comparable between implementations. This report describes an agent that plans its path through the floors of the building, communicates its intentions with other agents, and bids on new tasks based on a time of arrival cost function. The path-planner is compared with a simple, non-cooperative rule-based controller.

The simulation environment allows for an arbitrary number of floors and cars. Both time and space are discrete in units of seconds and feet, respectively. The dynamics of each car is deterministic and discrete. An onboard control system will bring each car to its desired goal floor, and this functionality is built in to the provided simulator. The arrival rate of passengers is uniform across all floors and is a simple probability that a new passenger will be created on each time step. This value is also called the load factor. Each passenger chooses a random floor within the building as its destination. The elevator car has a maximum capacity, in terms of the number of passengers it may hold.

The details of the simulation environment differ from the dynamics of real elevator systems mainly in the passenger load model. Real-world passenger arrival patterns are not only non-uniform in floor but also in time of day. Morning time traffic tends to originate on the ground floor and choose multiple destinations, while afternoon traffic originates on multiple floors and has a common destination: the ground floor. Crites & Barto [1] describe many common approaches to this problem domain.

Despite the differences mentioned above, both the Simple and Enhanced elevator controllers may still be compared directly within their common task domain. As in the real-world situation, cars see hall calls as a simple up or down request on a certain floor. The decision engine is unaware of the number of passengers waiting on a given floor. Car calls give the elevator knowledge of passenger destinations once they enter the car.

2 Methods

The elevator simulator allows for limited decision making in each car: once per time step, if the car is stopped with no current goal, it may decide on a new goal floor. The car is not allowed to change this goal during the trip. The perceived shortcomings of this decision-making approach are twofold. First, the car is unable to pick up new calls while en route to its current goal. Another car must dispatch to this new call regardless of the efficiency of such a decision. Second, a car cannot prevent traveling to the goal floor even if the hall call button is reset on that floor, implying that another car has answered the call. It will simply arrive at the floor, open its doors, and close them again when no passengers load. The lack of any communication mechanism between cars gives rise to this particular shortfall.

The Simple car algorithm is very basic - it accumulates internal passenger floor requests as well as external hall calls into a single collection and heads to the nearest floor among them.

Additional logic is required in order to ensure that potential destinations are compatible with the car's indicated direction. Algorithm 1 presents the basic functionality of the Simple car DECIDE() function.

Algorithm 1 The Simple DECIDE()

```
D ← my passenger destinations
C ← outstanding hall calls
d ← my direction ∈ {−1, 0, 1}
f ← my floor
sortDescending? ← (d == −1)

if this.atCapacity then
  arraySort(D, sortDescending?)
  GoTo(D[0])
  return
end if

for i ← 0 to |C| do
  c_f ← C[i].floor
  c_d ← C[i].direction
  if d ≠ 0 and d == c_d then
    if d > 0 and f <= c_f or d < 0 and
    f >= c_f then
      arrayPush(D, c_f)
    end if
  else if d == 0 then
    arrayPush(D, c_f)
  end if
end for

arraySort(D, sortDescending?)
GoTo(D[0])
```

The Enhanced car agent seeks to alleviate the perceived shortcomings of the Simple car. First, each elevator maintains a plan of the path it must take to fulfill its commitments. Each node in the path includes a floor number, call direction, requested action, estimated passenger count, estimated time of arrival, and the priority number of the event. Path actions are either load or unload, and plan call directions may be up or down. The estimated passenger count is computed based on the order in which the car services load and unload requests, assuming one passenger per node. The case of multiple passengers boarding is discussed later. Since the dynamics of the simulator are known, the agent may accurately estimate rest-to-rest travel times between floors and even whether the agent can stop in time to service a call given its current position, velocity, and acceleration. The priority number of the path node is simply the order the call entered the system. Lower-numbered nodes are serviced with priority, as they entered the system earlier than higher-numbered nodes.

The agent maintains consistency in its planned path by recalculating the estimated parameters whenever a new node is added to the path. If at any point in the path the passenger count exceeds the car's capacity, the car recursively removes load request nodes until the path is valid. This is the mechanism by which the car copes with loading more passengers than anticipated during stops. The agent informs the others of its removal of a load request from its path through the agent communication mechanism: the blackboard.

The mechanism for preventing conflicting agent goals is the blackboard, a common memory space shared between agents. Once per time step, the blackboard is checked for consistency. New hall calls are added to the blackboard and bid upon immediately by all agents. Knowing their current path, along with estimated time of arrival to each node, each agent may estimate whether it can logically fit the new call into its path as well as when it would be able to service the call. The bid submitted by each agent is the time it would take it to reach the new call plus the time from that call to the next call in its path. This bidding format is essentially a greedy search in time to reach the call and resume the current course.

Sometimes, all agents are unable to bid on a new call. In this case, a wait signal may be placed on that call so that it is not bid upon again until the wait signal is cleared. Since the state of car paths remains unchanged during travel, cars will not be able to bid again until a state transition occurs. Thus, when a car reaches its destination, removing a node from its

path, it resets the wait signal on all outstanding calls posted on the blackboard, allowing for all cars bid on these calls once more.

When an agent wins a bid for a new call, it marks that call as claimed on the blackboard. This is the method by which an agent communicates its intentions. Finding the path insertion point for the new node requires a bit of expert logic. For instance, an up call on floor five should not come before an up call on floor three if the car is below floor three and moving up. Much of the effort in designing this system is tied up in fixing edge cases in the path insertion logic.

Due to the communication and path planning requirements of the Enhanced agent, as well as to the reporting and evaluation requirements, some modifications were made to the simulation framework.

- Passengers record when the car arrived to pick them up.
- The global controller keeps a list of all passengers who have completed their trips.
- The controller maintains a list of hall calls in order of occurrence and a flag denoting new calls.
- The time step logic for the car is expanded to allow for decision making while en route.
- The Load and Discharge functions add and remove items from the agent path and blackboard.

3 Results

Simulation results were obtained for a variety of situations and address various metrics of performance. As with any multi-variable problem, some variables are held constant while others change. The first set of results holds passenger arrival rate and floor count constant and varies the number of elevator shafts in the building. A building architect with this type of data could determine the number of elevators to install for a known building design. The pickup time of the passenger is the time difference between calling for an elevator and loading into the car. The percentage of trips completed represents the ability of the system to handle the passenger arrival load; it is the proportion of total passengers in the system that reached their destinations before the end of simulation.

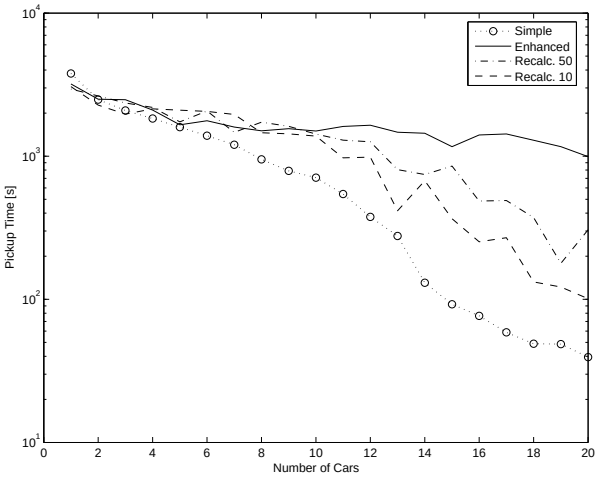

Figure 1: Average pickup time versus number of cars (floors=30, load=0.02, t=10000)

Initial simulation results showed the Simple agent vastly outperformed the Enhanced agent in most situations. In an effort to confront a potential over-commitment problem on the part of the Enhanced agent, a recalculate interval was added. At the end of a specified interval, each car discards its current hall calls (while keeping the car calls of onboard passengers) and all agents rebid on all hall calls, in the order of call priority. This extra computation step shows significant improvement in the performance of the Enhanced agent for all metrics tested.

The figures present results for a thirty floor building after 10000 seconds of simulation time. The load factor is 0.02, and the number of cars is the horizontal axis variable. Figure 1 shows the average pickup time over the simulation interval versus the number of elevator shafts, and Figure 2 shows the percentage of trips completed by the agents, also plotted against the number of cars. The effectiveness of the Enhanced agent in arriving at hall calls quickly is surprisingly poor, as this is the metric bid upon in its cost function. The Recalculate-50 and Recalculate-

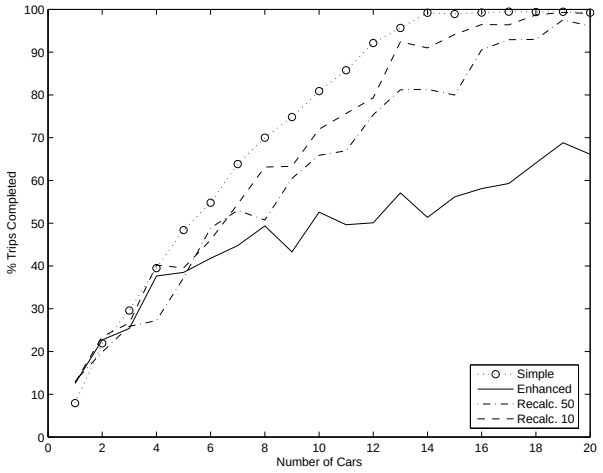

Figure 2: Percentage of completed trips versus number of cars (floors=30, load=0.02, t=10000)

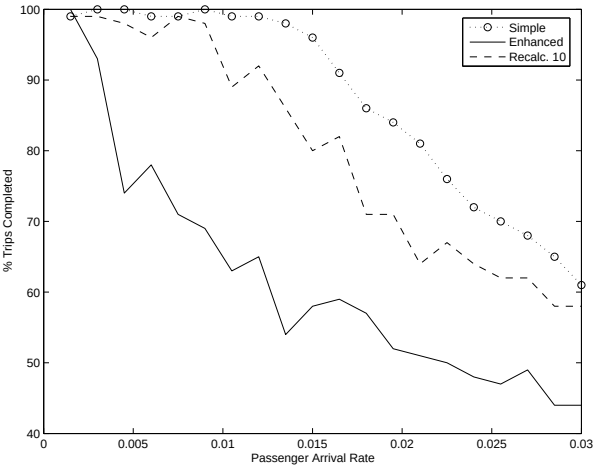

Figure 4: Percentage of completed trips versus passenger load (floors=30, cars=10, t=10000)

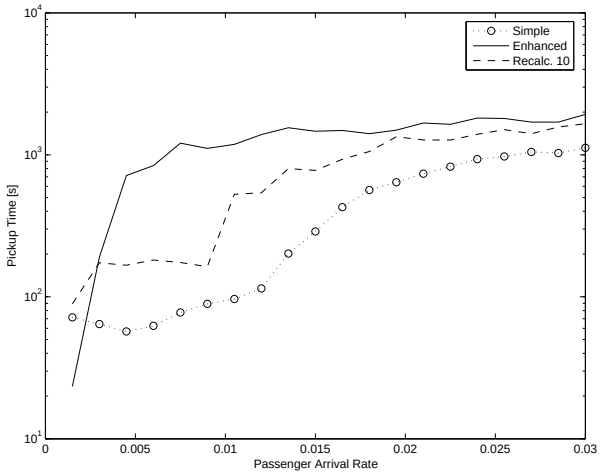

Figure 3: Average pickup time versus passenger load (floors=30, cars=10, t=10000)

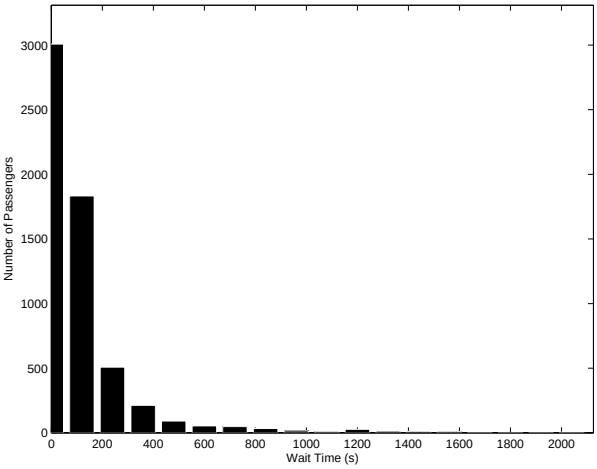

Figure 5: Simple agent, pickup time distribution (floors=30, cars=14, load=0.02, t=10000)

10 agents are simply Enhanced agents that perform a recalculate step at 50 and 10 seconds, respectively. Both agents improve upon the Enhanced agent.

Figures 1 and 2 reveal that with 14 cars, the Simple agent system achieves a near 100% completion rate and a drastic marginal improvement in pickup time. Starting from this data point, Figures 3 and 4 show the effects of varying the arrival rate of the passengers upon completion rate and pickup time.

The distribution of pickup times is important because it helps to estimate the expected wait time for a passenger. While Figures 1 and 3 plot average pickup times, Figures 5 through 8 expand the same 14 car data point to show the distribution of all pickup times for the various agent systems. All the distributions have fat tails, indicating that there is a significant portion of the population experiencing extremely long wait times. The Simple agent (Figure 5) maintains its dominance over all forms of the Enhanced agent, but there remain a few outliers in its pickup time distribution. The Recalculate-10 (Figure 8) agent improves upon the maximum wait time of the Enhanced agent (Figure 6) by a factor of at least two. The Recalculate-50 agent in Figure 7 also shows a striking improvement over the Enhanced agent system.

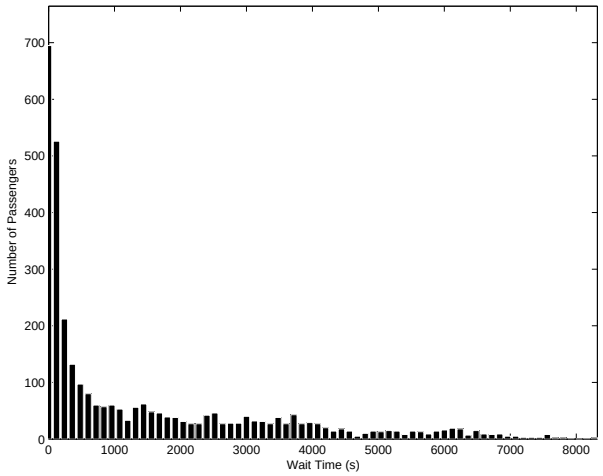

Figure 6: Enhanced agent, pickup time distribution (floors=30, cars=14, load=0.02, t=10000)

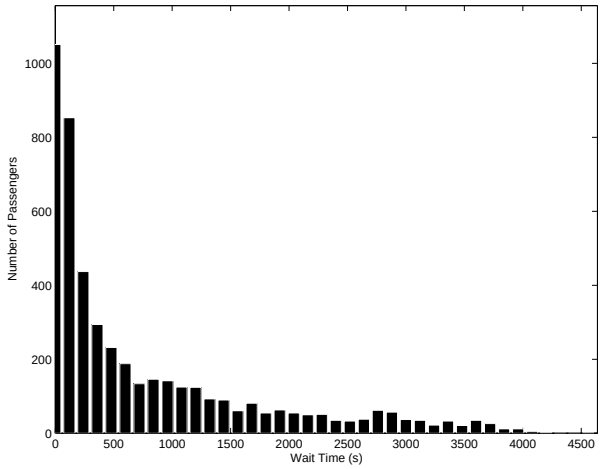

Figure 7: Recalculate-50 agent, pickup time distribution (floors=30, cars=14, load=0.02, t=10000)

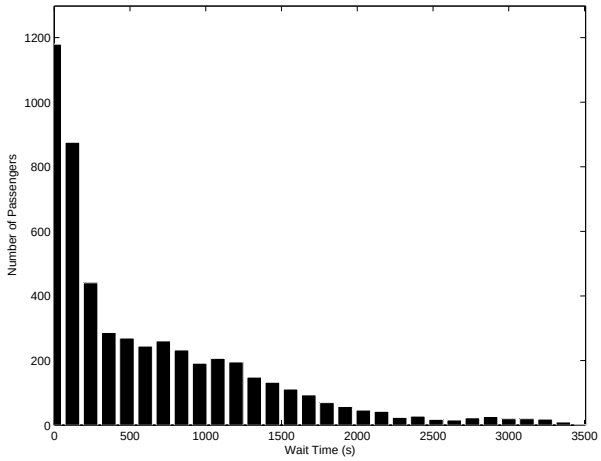

Figure 8: Recalculate-10 agent, pickup time distribution (floors=30, cars=14, load=0.02, t=10000)

4 Discussion

The results of this experiment reveal that the Simple agent is superior in most respects to the much more complicated Enhanced agent. The computational complexity, code size, and run time of the Simple agent are also better than those of the Enhanced agent, in all its forms. The goal of the project, therefore, was not met, but some valuable insight has been gained.

Concerning the commitment problem, the Enhanced agent reveals much. Path planning in a dynamic, stochastic environment is fraught with pitfalls. Plans must be laid carefully, but the planner must be able to abandon old plans in favor of better ones. In carefully planning of its course, the Enhanced agent limits its long term performance in that it choses its planned path over the most efficient choices of the moment. The Simple agent has no such commitment; it simply fulfills its car calls and answers the nearest hall calls.

The Recalculate agents behave much more like the Simple agent in that they make no permanent plans. Each time the recalculate phase is performed, the agents are allowed to reallocate the hall calls in the most efficient way. The notion of bidding an agent's estimated arrival time makes more sense when those arrival times happen sooner in the agent's plan. Long-term estimates prove to be grossly inaccurate and inefficient in this domain, as evidenced by the results in the previous section. The effect of shorter plans is evident even between the two Recalculate agents, as the agent with the shorter time horizon outperforms that with a longer outlook.

The way forward could be to incorporate the known length of the recalculate phase into the bidding cost function of the agent. Knowing that it will not be able to keep a long-term commitment could help the agent bid more accurately on new hall calls. The Simple agent's performance is surprising, but there must be more

efficient methods of solving the multi-elevator control problem.

Issues not addressed by this simulation framework, such as time-varying passenger arrival rates and non-uniform destination floor selection, are of great importance to the elevator control community. It would be rather informative to compare the Simple agent with some more complex control methodologies like those listed in [1].

The question of non-homogeneous agent teams also arises in the elevator domain. Express elevators and zone covering are two common examples, but a system of Simple agents with a few Recompute-10 agents mixed in could perform better than a homogeneous system.

In all, this project raises more questions than it answers, and the task domain proves to be very rich. There are many areas of further investigation nested within the framework of this task. Perhaps the author will revisit those topics in future work.

References

[1] Crites and Barto (1998). *Elevator Group Control Using Multiple Reinforcement Learning Agents.* Machine Learning, 33:235-262