Simulation with Learning Agents [*]

Erol Gelenbe, *Fellow IEEE*, Esin Şeref, Zhiguang Xu
School of Electrical Engineering and Computer Science
University of Central Florida
Orlando, FL 32816
{erol,esin,zgxu}@cs.ucf.edu

March 29, 2001

Abstract

We propose that learning agents be incorporated into simulation Environments in order to model the adaptive behavior of humans. These Learning agents adapt to specific circumstances and events during the simulation run. They would select tasks to be accomplished among a given set of tasks as the simulation progresses, or synthesize tasks for themselves based on their observations of the environment and on information they may receive from other agents. We investigate an approach in which agents are assigned goals when the simulation starts, and then pursue these goals autonomously and adaptively. During the simulation, agents progressively improve their ability to accomplish their goals effectively and safely. Agents learn from their own observations, and from the experience of other agents with whom they exchange information. Each learning agent starts with a given representation of the simulation environment from which it progressively constructs its own internal representation and uses it to make decisions. This paper describes how learning neural networks can support this approach, and shows that goal based learning may be used effectively used in this context. An example simulation is presented in which agents represent manned vehicles; they are assigned the goal of traversing a dangerous metropolitan grid safely and rapidly using goal based reinforcement learning with neural networks, and compared to three other algorithms.

Keywords: Simulation, Adaptive Entities, Learning Agents, Goal Based Adaptation, Random Neural Networks, Reinforcement Learning

1 Introduction

Typically, in a discrete event simulation the time and nature of future events is computed in a predetermined fashion from the list of past events which have occurred. This is typically achieved by creating a dynamic event list which maintains all future events whose occurrence is currently known as well as their date of occurrence; current time then moves to the earliest future event, and specific algorithms are used to process the consequences of this event, and updating the future event list in the course of this process. Additional variables may be driven by random number generators to create an element of apparent unpredictability and probabilistic variability in the simulation. Thus the designer of a simulation will typically pre-specify all possible transitions, and will not leave the definition of state transitions to entities which are not fully pre-determined. This conventional approach to simulation design has the advantage of providing the designer with full control over events. However it also has some major disadvantages. For instance, the behavior of living "agents" or of agents which are controlled by human beings (e.g. manned vehicles) does not just obey well-known physical laws. Human beings will reach a variety of decisions which are not fully dictated by current physical circumstances, and changes in human behavior can occur under the effect of observation, experience and learning, stress, fear and many other factors [8]. Thus it would be very useful to introduce agents in a simulation whose behavior is determined by specific circumstances and through adaptive behavior.

[*]This research was supported by a grant from the U.S. ARMY SIMULATION AND TRAINING COMMAND.

The alternative we propose is that, in addition to the usual attributes of a discrete event simulation (such as event lists and possibly random number generator driven events), a simulation should contain a certain number of learning agents. These learning agents store experience during a simulation, and use it to modify their behavior during that same simulation or during distinct simulation runs. Thus the agent's response, and hence the simulation's detailed outcome to identical event lists can change over time, both within the same simulation run and during successive distinct runs, to reflect learning from experience. In addition to allowing us to explicitly include learning and adaptive behavior, this approach is simple to implement: it is easier to describe rules of adaptation using neural networks or similar techniques, than to specify the behavior of an individual under all possible cases that might arise.

We [12, 13] have recently applied these ideas to the well known existing military simulation tool ModSAF, in which we have enhanced the "mine-field breaching" component of tank platoons which is currently based on fixed doctrine, to obtain an approach which modifies the behavior of each individual tank to take into account the on-going fraction of damaged tanks during the breaching manoeuver. Of course, a greater level of adaptation in a simulation system may result in more complexity in the simulation, and hence longer simulation runs. However the purpose of our work is to propose an approach that provides greater simulation realism to represent adaptive behavior; it appears clear that this can lead to greater computational cost.

There is another advantage to being able to avoid having to completely pre-specify a simulation: this relates to the economy of producing simulation software. For instance if we could introduce exactly the same code in many of the active entities of a simulation (simply by replicating the code), and then provide each distinct entity just with a goal or objective which could be specified in a compact form but which might vary from entity to entity, one would be able to save programming time and also reduce the number of programming errors. As an example from a simulation of entities which travel in a geometric terrain, the basic code for each entity might be the same, but some entities would be instructed to "Go from point A to point B by following the roads", while another entity with cross-country capabilities might be instructed to "Go from point C to point D by following the shortest path". Thus the different goals would use the same code in either entity, and we would be saved from having to write different high-level decision programs for each of the entities.

In this paper we propose simulation environments in which intelligence can be constructed into Learning Agents (LA), rather than just in centralized control. In the proposed approach, LAs learn from their own observations about the simulation environment and from the experience of other agents; they rely minimally on centralized control. We will illustrate these concepts with simulations of LAs which use three learning paradigms: adaptive finite-state machines, feedforward neural network learning and reinforcement learning.

In simple terms a LA, in the context of the simulation environment we consider, is an algorithmic entity which models and object which is relevant to this simulation: for instance an automobile in a traffic simulator. A LA receives inputs and produces outputs; for instance an automobile is informed of its current position and directional velocity, and about the cars which are surrounding it: these are its inputs. It then reaches a decision to maintain or modify its velocity and perhaps use certain traffic signaling conventions: these are its outputs. One could algorithmically specify how the car's velocity changes as a function of these inputs: this is the conventional approach to simulation. Or one could replace this algorithm with a learning neural network which has been trained with observations of real cars in similar conditions. At this level, we are not yet dealing with a learning agent since we have just stored an input-output function in a neural network which controls the entity. However, if in addit ion we could insert into the entity the ability to learn specifically from the entity's environment in real time, then we would be dealing with a LA, according to the ideas we develop in this paper. To pursue our illustration of how a LA may embody the behavior of a car in a traffic simulation, consider the following possibilities which allow a LA to embody some aspects of driver behavior. During the simulation every car can decelerate by braking. When road or weather conditions deteriorate, the effect of the car's braking becomes less effective. The LA learns that braking is less effective at this time and lowers the speed of the vehicle. Later, when the conditions change and braking becomes again more effective, the

LA increases the speed of the car again. This learning behavior could be rendered quite complex by introducing information coming from other vehicles, such as noticing that other vehicles pass very fast on the left: then the current vehicle either moves as far right as possible (cautious driver) , or the current vehicle also enters the left lane and accelerates (imitating other drivers), depending on such things as the driver's "goal" (do I prefer to arrive safely or to arrive as soon as possible?). The degree to which the driver would be cautious or less cautious could change also as a function of the number of accidents that the driver has observed on the road that day, and so on.

The rest of this paper is organized as follows. Section 2 presents a description of learning agents as we define them in the context of simulation. To illustrate these ideas, in Section 3 a specific simulation environment is described in which agents represent manned vehicles traversing a dangerous urban grid, and we discuss adaptive algorithms which learning agents may use. In Section 5 we present simulation results to illustrate the effectiveness of these ideas and compare the performance of the various learning algorithms we discuss earlier. Conclusions are summarized in Section 6. Some background on the neural network model used in this paper is given in the Appendix.

2 Adaptation by Learning Agents

In the approach we propose, each learning agent starts with a representation of the simulation environment. From this information it then progressively constructs its own internal representation or Cognitive Map (CM) and uses it to make decisions. these decisions are implemented as conventional asynchronous or synchronous transitions in the discrete event simulation. Each learning agent progressively refines its own model of the simulation environment as the simulation progresses, and it uses adaptation to make decisions. In the most extreme case, a learning agent might build an internal "private" representation of the state of the complete simulation as it pertains to its own needs.

Mailboxes can be used to share information among agents. Learning algorithms are used by the agents to update their CM based on the information in mailboxes and on their observation of the other variables in the simulation. An agent then reaches decisions using its prior experience and all other available input. Each agent is assigned a Goal before simulation begins, and the learning agent uses the goal to determine its course of action each time it has to make a decision. Goals can be modified by other agents, or can also be modified by the agent itself, based on its own experience. Learning agents can also be hierarchically structured so that a group of agents share the same goals, use similar sets of rules, and make use of each other's experience. If we are considering a set of LAs simulating entities which have to obey hierarchical orders, some of these LA will simulate the leaders and they will assign or modify the goals of the LAs who report to them. For instance suppose we consider LAs which simulate a tank which is part of a tank platoon. A typical goal assigned to an agent might be: "Go from some origin S to a destination D in minimum time". A more sophisticated goal could be: "Go from S to D in minimum time, but do not overtake the leader of your group of agents". Yet another goal may simply be: "Go from S to D without getting destroyed".

In our work, such goals are translated into numerical quantities (e.g. delay values, probabilities of being destroyed, weighted combinations of such numerical quantities) which are obtained during the simulation from observation of the events which occur. They are then used directly by the LA for adaptive learning and control.

For decision making, the LA will rely on its CM, on its own observations of the environment, on information it may receive from other agents, and will then translate this information into a course of action which best allows it to attain its goals. A LA may also refer to "reality" (or the temporal state of the simulation) to update its own learning rules and improve its efficiency and accuracy of the learning algorithm it uses to build its internal representation or Cognitive Memory (CM) and to make decisions. We believe that in future simulation environments, LAs will routinely inhabit the simulation environment at the same time as other conventional objects.

2.1 Adaptation Paradigms

A very simple approach to adaptation is to change one's course of action so as to respond to information resulting from the the most recently available observations or data. A slightly more sophisticated approach would be to integrate this past information over a certain time frame. Thus a LA which simulates a vehicle which is on its way from point A to point B, and whose essential goal is not to become a casualty (i.e. not to be destroyed) would simply head back if it encountered danger. If the goal is a weighted combination of the delay (or time to get to the destination) and of the probability of being destroyed, then the decision would be to head back or to persevere toward the objective, depending on a more refined estimate by the agent. Thus, as goals become more complex, simple reactions to external information are not sufficient for decision making, and more sophisticated techniques for integrating external observations and information are needed. An individual LA may find it necessary to integrate the cumulated experience of hundreds or of thousands of other LAs which may have encountered similar situations. In this section we discuss the learning paradigms that we propose for the LAs. These paradigms allow the LAs to integrate observations and the experience of other agents. The resulting performance in a test simulation environment is evaluated and discussed in the Section 3.

The LA regularly updates its CM with information from mailboxes (coming from other LAs), and from its own observations.

The simplest decision paradigm we will consider is the case where the LA makes the decision which is most advantageous (lowest cost or highest reward) based on the instantaneous expected value of this metric. We will call this approach the *Bang-Bang Algorithm*. This approach has the advantage of being simple and explicit. Its disadvantage is that it does not allow us to easily integrate the values of the cost and of the reward as it is perceived or integrated by other agents over time, and that it requires a priori knowledge about the lowest cost that should be expected among a set of possible decisions. In order to take into account a greater wealth of information, we will also investigate two other learning paradigms for the agents:

- *Learning feed-forward random neural networks (LF RNN)* [6, 10]. These networks update their internal representation (the weights) using a gradient based algorithm to improve either their predictive capabilities, or to improve their ability to reach decisions which elicit the maximum reward. The LF RNN is used as a learning tool and then as an "oracle". When a decision needs to be taken, the network is presented with the external data and is used to produce a prediction of the cost or reward of different outcomes. The outcome offering the lowest cost or the highest reward is then selected.

- *Random neural networks with reinforcement learning (RNN RL)* [6, 14]. In this case a network is used both for storing the CM and making decisions. The weights of the network are updated so that decisions are reinforced or weakened depending on whether they have been observed to contribute to increasing or decreasing the accomplishment of the declared goal. After an update has been effected, the network explicitly recommends the outcome which appears to it to be the most beneficial.

A description of the basic neural network model (RNN) and of the related learning algorithms used in this paper is given in the Appendix.

Adaptive Stochastic Finite-State Machines (ASFSM) [3, 4, 7] are another class of models which we will investigate in future work. It is known that by introducing randomness in a finite state machine's transitions, one can obtain a computational model which is as powerful as a Turing machine [1]. Such models can be easily implemented using deterministic finite-state machines and random number generators [2]. SFSM's can implement complex computational decision processes, and learning is achieved by updating the state transition probabilities so as to maximize expected rewards associated with decisions. State transitions then occur probabilistically using these updated transition rules.

3 Simulating Vehicle Traffic in a Dangerous Urban Grid

In order to study the impact of adaptation on the behavior of the agents, we wrote a simulation program in which LAs travel through a dangerous urban grid. The purpose of the simulation experiments was to examine how the adaptive algorithms which we have described will affect the performance of the learning agents. Both a very simple decision algorithm (the Bang-Bang algorithm described below), and the more sophisticated neural network algorithms using learning, were tested. A single simulation program was written, and four simulations were run separately under identical input conditions. One simulation was run with no adaptation at all. Each of the remaining three simulations were run with the learning agents using three different paradigms for adaptation.

The system being simulated is an urban grid: it is composed of streets and intersections. In this particular simulation there are ten streets running vertically from North to South and vice-versa, and ten streets running East to West and vice-versa. This leads to an urban grid with 100 intersections. This grid is represented schematically in Figure 2.

Each section of street between two intersections is of the same constant length for each street section, and we assume that vehicles proceed at *constant average speed* between any two consecutive intersections. Time is normalized so that *average unit time* is the time it takes a vehicle (agent) to travel between two consecutive intersections (either horizontally or vertically). Streets are "two-way" but there is only one lane in each direction. Thus vehicles can only proceed in a single queue in each direction. We have simplified things so that street sections are long enough to contain all vehicles which have entered them: there is never any blockage back from a street into an intersection.

3.1 Vehicle (Agent) Motion and Traffic

Agents (vehicles) are allowed to enter and leave the simulation "vertically" either from the top ten intersections or the bottom ten intersections. Thus any vehicle will enter from one of the top or bottom ten intersections, and will leave from an intersection at the bottom or top. Each vehicle entering from an intersection (top or bottom) will be assigned a fixed exit intersection at the opposite end (bottom or top) which it will try to reach. The destination is selected by drawing it at random with equal probability among all possible 10 destinations.

At any intersection, a vehicle can either go Forward through the intersection (i.e. continue in the same direction as it came) or go Left or Right. It may also Wait, but it cannot go back the way it arrived. The only exception is at intersections on the East or West edge of the grid: at the East edge, vehicles cannot go East and cannot go back, while at the West edge they cannot go West and cannot go back.

In order to build time variation and randomness into the simulation, several parameters are chosen to be random variables. The arrival of traffic at the input North and South intersections is taken to be a Poisson process with rate Λ vehicles/unit-time at each input point. This means that the external inter-arival times of vehicles to each input point are exponentially distributed random variables of average value Λ^{-1}. By varying Λ we can vary the traffic load in the simulation. Throughout the simulation experiments, the average time it takes an agent to move from an intersection to another adjacent intersection is set to 1 in order to normalize all other quantities in the simulation experiments. The actual time it takes each agent to move from an intersection to an adjacent intersection is chosen to be random, with an exponential distribution. However agents moving in a given direction down the same street are queued up behind each other; for an agent to go down a street, the agent just in front of it must have completed its motion and reached the end of the street. This constraint has been introduced to allow us to simulate congestion very simply.

3.2 Destruction of Vehicles or Agents

The simulation represents a "dangerous" urban grid by introducing a probability of vehicle destruction. Destruction of a vehicle occurs at random only at intersections (the street's themselves are "safe"). We also make the simplifying assumption that a vehicle which is destroyed does not block other vehicles. At each intersection (i, j), there is a probability $p(i, j)$ that a vehicle crossing that intersection is destroyed. Some intersections will be relatively safe with a small value of this probability, while others may have a high value and be very unsafe. Vehicle destruction was simulated by selecting a small fixed probability of destruction throughout all but a few specific intersections; at those selected few there is a high probability of vehicle (agent) destruction.

3.3 The Agents' Goals

All the agents are assigned a common goal, which is to minimize a weighted combination of average delay (W) and probability of destruction (L). This common goal can be expressed mathematically as follows:

$$G = \alpha W + \beta L, \tag{1}$$

where α and β are used so that the loss probability is commensurate with the delay, and to provide a relative weight to each of these two factors. For example, if we were to select $\alpha = 1$, we can take some value of β which would make βL commensurate with the delay. For instance, in the simulation example at hand, the minimum average delay is simply the time it would take a vehicle to reach its expected destination point without any queuing effects, and its value can be easily calculated to be 15. Since $0 \leq L \leq 1$, we can select $\beta = 150$ if the typical loss values we wish to address are of the order of $L = 0.1$. A smaller value of β implies that we are giving relatively less importance to loss, and vice-versa.

The agents make decisions concerning what to do when they reach an intersection. They are allowed to use three items of information which they can observe directly, or which they learn by communicating with other agents:

- (1) The length of the local queues (congestion) in each direction at the intersection,
- (2) Recent values of the downstream delays experienced by agents having the same final destination which have previously gone through the same intersection, selected a possible direction, and who have reached their destinations; call these W_{IDd1}, W_{IDd2}, ... where I refers to the intersection, d refers to the direction taken at that intersection, and D refers to the final destination.
- (3) the destruction or loss rate of agents having the same final destination which have passed through the same intersection and selected one of the possible directions; call these L_{IDd1}, L_{IDd2}, ...

Both the Learning Feed-Forward RNN (LF RNN), and the Reinforcement Learning RNN (RL RNN), will use these values directly in updating their internal representations and thus in making their decisions.

In the simulation, collecting information about the exact state of the urban grid, about vehicle losses and vehicle delays is simple since one can conceive a program where all these variables are shared with all agents. If we were dealing with the "real world situation" we have described, the position of the vehicles could be available to themselves via GPS (global positioning system). This position could be sent by wireless to other vehicles. Wireless communications could be used to interrogate each vehicle about its status. Presumably, a vehicle that does not answer would be assumed to have been damaged or destroyed, and its most recent known "safe" position and direction would provide information about where it has been destroyed. Similarly, some damaged vehicles may still have the ability to communicate and would then inform the other vehicles where they have been damaged. Furthermore, once a vehicle arrives to destination, it can inform all other vehicles about the respective delays it incurred thro ughout its path to destination. Thus the data needed for learning can be available both in the "real" and the simulated environment.

Note that for the different algorithm which were tested, the LA which models each individual vehicle will incorporate the algorithm. Thus, when a neural network is being used, each LA will contain a distinct set of network weights For that particular vehicle and at a particular time in the simulation. Similarly for the Bang-Bang algorithm, the algorithm will be stored in each LA corresponding to each vehicle, and will use different data depending on where it is located.

4 The Decision Algorithms

All the decision algorithms used by each LA will have to use an estimate of the current goal value G. To do that, they will have to estimate current delay and loss values. This will be done by gathering information from what has happened to vehicles which have previously traveled through the urban grid.

Let W^n_{IDd} be the most recently available downstream delay value experienced by some vehicle going from I to D known for the direction d. A LA whose final destination is D will estimate the delay W_{IDd} it can experience (from intersection I to destination D) if it makes the choice of direction d at its current intersection I. Each agent will construct an estimate of the risk or probability L_{IDd} of being destroyed if it takes direction d at intersection I as it heads for its destination D. Let a be a constant with $0 < a < 1$: a small value of a will give greater weight to recent values of delay (or of loss, when it is used to update the loss estimate), and vice-versa. The estimates are computed as weighted running averages of the form:

$$W_{IDd} \leftarrow aW_{IDd} + (1-a)W^n_{IDd}, L_{IDd} \leftarrow aL_{IDd} + (1-a)L^n_{IDd}.$$

All three algorithms will use these quantities in making decisions.

We will use the "fixed-shortest-path" algorithm as a straightforward benchmark for comparison. This algorithm obviously uses full a priori knowledge of the topology of the urban grid to compute shortest-paths, in number of hops or intersections that need to be traversed by a vehicle, for each source-destination pair in advance of the simulation. As such, it has high initial computational cost but is of low computational cost after each route is set up since we know in advance for each origin-destination pair which direction will be selected from each intersection. We will see that this algorithm is relatively ineffective compared to some of the other algorithms. In any case, it would be impractical as a way of implementing intelligent adaptive behavior on the part of individual LAs because it assumes that everything is known in advance and pre-planned by the user of the simulation.

For the remaining algorithms, the following characteristics can be outlined:

- **Bang-Bang** This algorithm uses the estimates W_{IDd} and L_{IDd}, but additionally makes use of the relative position of I and D to determine reasonable estimates of the actual delay and loss that it should incur. In other words, it assumes that the topology of the grid is known in advance. As such, it assumes that more information is available than the two neural network based decision algorithms.

- **RL RNN** This algorithm just makes use of the values of W_{IDd} and L_{IDd}, and combines the decision of the direction to take at each intersection with the learning process. The amount of computation needed is comparable to that of the Bang-Bang algorithm but it does not assume prior knowledge of the topology of the grid. Interestingly enough, it provides a level of performance which is comparable to the Bang-Bang algorithm. Therefore it is superior to Bang-Bang since it gets the same results with less information. Both Bang-Bang and RL RNN yield much better performance than the fixed-shortest-path algorithm.

- **FF RNN** This algorithm uses gradient-descent based learning in the RNN as described in [6]. It is the most computationally costly of the three learning algorithms since learning is undertaken at each intersection. The algorithm uses the estimates W_{IDd} and L_{IDd} to update network weights. Then it provides its own estimate of the delay and loss to be incurred in each direction d. Finally the direction which it estimates will result in the smallest loss is selected. We see that, of the three, it is the least effective. However, like RL RNN, it does not use a priori information about the topology of the grid.

4.1 The Bang-Bang Algorithm

The Bang-Bang algorithm uses a simple approach. It is a switching control policy which changes its choice of the current outgoing direction d whenever it "thinks" it should be doing better in terms of loss probability or delay to destination. The simple idea here is to try another outgoing direction at random, if the current one appears to provide poor performance with respect to some known performance target.

The agent then makes an assessment of what a “reasonable” value of downstream Loss L^*_{ID} and Delay W^*_{ID} should be to its destination D, based on the intersection I where it currently is, and using its distance to the destination.

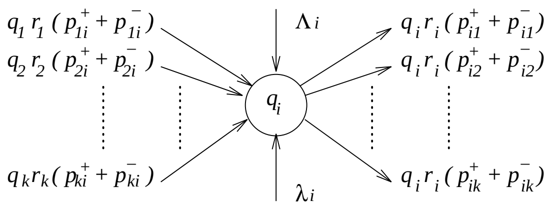
The delay estimate W^*_{ID} is obtained as follows. The local queue length of vehicles is considered and used to estimate the average time through the intersection. This time is then multiplied by the row distance from I to D and also by a small constant larger than one to introduce a tolerance factor; the result is used for W^*_{ID}. The loss estimate L^*_{ID} Is obtained as follows: from I and D we know the minimum number of intersections that will be traversed, and this number is multiplied by a small constant greater than one. We then use the estimate of the loss probability at each intersection to compute the estimate of the total loss probability. In the simulation results we have presented, the “small constant” is taken to be 3.

The LA first selects the direction d used by the previous LA which was at I and was headed for D. If either $W^*_{ID} < W_{IDd}$ or $L^*_{ID} < L_{IDd}$, then the agent judges that an alternate outgoing direction should be tried, and selects it at random among all directions which are on a path to destination D.

Note that the Bang-Bang algorithm, though quite simple, makes use of a priori information about the structure of the grid through the Loss L^*_{ID} and Delay W^*_{ID}. If these estimates are not accurate, then the algorithm cannot be effective. Neither the FF RNN nor the RL RNN make use of such a priori information, and each LA only relies on information which is gathered on-line from other LAs (vehicles).

4.2 The Random Neural Network (RNN) Based Algorithms

For the reinforcement learning approach, as well as in the feed-forward neural network predictor, we have used the RNN [6]. This is an analytically tractable spiked neural network model which is schematically shown in Figure 1. Each neuron $i$in this network can be interconnected to other neurons via excitatory or inhibitory weights as shown in the figure; it can also receive excitatory or inhibitory external signals. As a result, each of the $i = 1,$,..., n neurons of the resulting interconnected network will have an internal state value given by equation (2).



$q_1 r_1 (p^+_{1i} + p^-_{1i})$
$q_2 r_2 (p^+_{2i} + p^-_{2i})$
$q_k r_k (p^+_{ki} + p^-_{ki})$
Λ_i
q_i
λ_i
$q_i r_i (p^+_{i1} + p^-_{i1})$
$q_i r_i (p^+_{i2} + p^-_{i2})$
$q_i r_i (p^+_{ik} + p^-_{ik})$

Λ_i	arrival rate of exogenous excitatory signals
λ_i	arrival rate of exogenous inhibitory signals
$q_j r_j p^+_{ji}$	arrival rate of excitatory signals from neuron j
$q_j r_j p^-_{ji}$	arrival rate of inhibitory signals from neuron j
q_i	probability that the neuron is excited

Figure 1. Schematic Representation of a connected neuron in the RNN

The state q_i of the $i - th$ neuron is the probability that it is excited. These quantities satisfy the following system of non-linear equations:

$$q_i = \lambda^+(i)/[r(i) + \lambda^-(i)], \tag{2}$$

where

$$\lambda^+(i) = \sum_j q_j w^+_{ji} + \Lambda_i, \quad \lambda^-(i) = \sum_j q_j w^-_{ji} + \lambda_i \tag{3}$$

and

$$r_i = \sum_j w_{ij}^+ w_{ij}^- \tag{4}$$

Here w_{ij}^+ is the rate at which neuron i sends "excitation spikes" to neuron j when i is excited, and w_{ij}^- is the rate at which neuron i sends "inhibition spikes" to neuron j when i is excited. Thus r_i is the total outgoing rate of spikes for neuron I when it is excited. For an n neuron system, the neural network parameters are these n by n "weight matrices" $\mathbf{W}^+ = \{w^+(i,j)\}$ and $\mathbf{W}^- = \{w^-(i,j)\}$ which need to be "learned" from input data.

Various techniques for learning may be applied to the RNN. These include Hebbian learning, which will not be discussed here since it is slow and relatively ineffective with small neural networks, Reinforcement Learning and Gradient Based Learning. The Reinforcement Learning algorithm we have used here is described below. The Gradient Based Learning algorithm can be found in [6]. The general function aproximation capabilities of the RNN are discussed in [10].

4.3 Reinforcement Learning for the RNN

There are many different ways to introduce Reinforcement Learning in the RNN model. In this paper we have used an extension of simple approach [14] which was originally suggested for navigation in a maze. The simulations in the next section have revealed that this simple approach appears effective for autonomous routing of the learning agents. In the sequel we will refer interchangeably to probability of Loss of a vehicle or of Destruction of a vehicle.

Given some Goal G that the learning agent has to achieve as a function to be to be minimized (i.e. Transit Delay or Probability of Loss or Destruction, or a weighted combination of the two), we formulate a reward R which is simply $R = G^{-1}$. Successive measured values of the R are denoted by $R_l, l = 1, 2,$ .. These are first used to compute a decision threshold:

$$T_l = bT_{l-1} + (1-b)R_l, \tag{5}$$

where b is some constant $0 < b < 1$, typically close to 1.

Then a RNN with as many neurons as there are different outcomes to a decision is constructed. Let the neurons be numbered $1, \ldots, n$. Thus for each alternate decision i, there is some neuron i. Decisions in this RL algorithm with the RNN are taken by selecting the decision j for which the corresponding neuron is the most excited, i.e. the one with has the largest value of q_j. Note that the $l-th$ decision may not have contributed directly to the $l-th$ observed reward because of time delays between cause and effect.

Suppose that we have taken the $l-th$ decision which corresponds to neuron j, and that we have measured the $l-th$ reward R_l. Let us denote by r_i the firing rates of the neurons before the update takes place.

What we do is first to determine whether the most recent value of the reward is larger than the previous "smoothed" value of the reward which we call the threshold T_{l-1}. If that is the case, then we increase very significantly the excitatory weights going into the neuron that was the previous winner (in order to reward it for its new success), and make a small increase of the inhibitory weights leading to other neurons. If the new reward is not better than the previously observed smoothed reward (the threshold), then we simply increase moderately all excitatory weights leading to all neurons, except for the previous winner, and increase significantly the inhibitory weights leading to the previous winning neuron (in order to punish it for not being very successful this time). This is detailed in the algorithm given below.

We first compute T_{l-1} and then update the network weights as follows for all neurons $i \neq j$:

- If $T_{l-1} \leq R_l$
  - $w^+(i,j) \leftarrow w^+(i,j) + R_l,$

  - $w^-(i,k) \leftarrow w^-(i,k) + \frac{R_l}{n-2},\ if\ k \neq j.$

- Else
  - $w^+(i,k) \leftarrow w^+(i,k) + \frac{R_l}{n-2}, k \neq j,$
  - $w^-(i,j) \leftarrow w^-(i,j) + R_l.$

Then we re-normalize all the weights by carrying out the following operations, to avoid obtaining weights which indefinitely increase in size. First for each i we compute:

$$r_i^* = \sum_{i=1}^{n}[w^+(i,m) + w^-(i,m)], \tag{6}$$

and then re-normalize the weights with:

$$w^+(i,j) \leftarrow w^+(i,j) * \frac{r_i}{r_i^*},$$
$$w^-(i,j) \leftarrow w^-(i,j) * \frac{r_i}{r_i^*}\ .$$

Finally, the probabilities q_i are computed using the non-linear iterations (2), (3), leading to a new decision based on the neuron with the highest probability of being excited.

5 Simulation Experiments

The learning algorithms we have simulated allow each individual learning agent to carry out a separate computation and make its own individual decisions. Agents arrive into the simulation via the intersections which are at the "north" and "south" end of the urban grid shown in Figure 2. Each agent has a destination intersection which is at the diametrically opposite end of the grid: an agent entering at the north end will exit at the south end, and vice-versa. Exits are impossible from any other point in the grid. The only other manner for an agent to leave the simulation is through its destruction at one of the intersections. The goal of the agents is to reach their destination point quickly (minimum delay) and safely (minimum loss).

The purpose of the simulations is to compare the learning agents' routing behavior and routing effectiveness through the grid against a fixed routing policy (marked "No Control" on the figures given below). In the static policy, the agent is routed along a static shortest path to the output layer of intersections, and then horizontally to its destination, without taking into account either congestion or the probability of destruction.

We have varied the arrival rates of vehicles or agents to each input ("north" or "south") intersection between 0.1 and 0.9, in steps of 0.1 or 0.05, so as to simulate the system under different "load" conditions. The arrival rate is interpreted as the average number of agents entering an input intersection from the outside world.

The probability that an agent is destroyed is set to 0.1 at most of the intersections, while this probability is 0.5 at a few specific intersections which are unkown to the agents. Actually, none of these loss rates are known to the agents. The "high destruction or high loss" areas are in four contiguous intersections given in (x,y) coordinates as (2,0), (2,1), (2,2) and (2,3), and also in four other intersections (7,7), (7,8), (7,9) and (7,10). Both the 0.1 and 0.5 loss values are very high in practical terms, but are selected so as to be able to illustrate the effect of learning by the agents.

The first simulation results compare the Bang-Bang approach to the fixed path routing of all agents; they are summarized in Figures 3, and 4. We see that when the goal is to minimize the **delay** (Figures 3, 4), the average delay of agents with control is lower than without control (Figure 3), while we have the opposite effect on destruction (Figure 4) as would be expected. In the next two figures, we base decisions on a combination of **destruction and delay** and compare the Bang-Bang algorithm we Reinforcement

Learning control; the particular values chosen of α and β lead to the results observed in Figures 5, 6 where average delay is just a little better with RL than with Bang-Bang control, while there is close to $70-80\%$ reduction in the rate of agent destruction. This shows that the learning agents are autonomously avoiding regions of high agent destruction and also providing improvements in delay.

Figures 7 and 8 present simulation results under the same conditions as in the four previous figures. Here we compare the RNN with Reinforcement Learning (RL) and Bang-Bang when the Goal includes both Loss and Delay. We see that RL and the Bang-Bang algorithm provide essentially equivalent performance. We have noticed this in many other simulations which we do not paper here.

We then examine the use of a FF RNN which is used to predict the Delay and Loss quantities. In each agent, recent samples of these quantities are used to train a FF RNN, and the network is then used to make decisions. The FF RNN is trained to predict the numerical value of the Goal for each possible option. The outcome is selected by choosing the neuron whose output which has the smallest numerical value of the Goal. This method is "indirect" in that it trains and uses the RNN as a predictor, while the RL algorithm trains the RNN directly as the decision making element. Figures 9 and 10 present results of the simulation runs when learning agents use this approach to control routing. We clearly see that this approach does improve performance over the simulations with "No Control", but that it is not as effective as the RL RNN or the Bang-Bang algorithm which both bypass the construction of a predictor for decision making. The RL RNN algorithm bypasses the use of a predictor for selecting the decision, while the prediction errors that may be made by the FF RNN appear to reduce the effectiveness of the latter method.

Note that the Bang-Bang algorithm, though quite simple, makes use of a priori information about the structure of the grid through the Loss L^*_{ID} and Delay W^*_{ID}. If these estimates are not accurate, then the algorithm will not be effective. Neither the FF RNN nor the RL RNN make use of such a priori information, and each LA only relies on information which is gathered on-line from other vehicles which have preceded it. Thus we see that the RL RNN is in fact the superior algorithm because it makes decisions which are clearly superior to the "shortest path" algorithm, and because its performance is equivalent to the Bang-Bang algorithm which uses a priori information contrary to RL RNN which uses only on-line estimated values.

6 Conclusions and Future Work

In discrete event simulation the time and nature of future events is typically selected in a pre-determined fashion from the list of past events which have occured. Random number generators are often used to introduce randomness or non-determinism in a seqence of events which would otherwise be strictly predictable. This conventional approach to simulation provides the designer with full control of events.

In this paper we have proposed simulation environments in which intelligence can be constructed into Learning Agents (LA), rather than just in centralized control. The LAs are provided with Goals, but their actions are not dictated in advance and they make choices so as to best achieve their goals. In the proposed approach LAs learn from their own observations about the simulation environment and from the experience of other agents; they rely minimally on centralized control.

This approach can provide a basis for simulating complex environments which include the behavior of living "agents" or of agents which are controlled by human beings (e.g. manned vehicles). Human behavior does not simply obey physical laws or circumstances, and changes in human behavior can occur under the effect of observation, learning, stress, fear and many other factors [8]. Another advantage of our approach is that it allows us to avoid having to completely pre-specify a simulation; this can potentially reduce the cost of writing simulation software. For instance if we could introduce exactly the same code in many of the active entities of a simulation (simply by replicating the code), and then provide each distinct entity just with an objective which is specified in a very compact manner and which may vary from one agent to the other, we may be able to save programming time and also reduce the number of programming errors.

We illustrate these concepts with simulations of LAs using three learning paradigms: the so-called bang-bang control, feed-forward neural network learning, and reinforcement learning. The simulation experiments we describe address routing of LAs representing vehicles moving in a dangerous urban grid. They show that goal based adaptive learning algorithms which are carried out independently by each individual agent without centralized control, can provide effective performance.

Of all the algorithms we have designed and evaluated, the reinforcement learning based RL RNN algorithm appears to be the best approach to the problem, since it provides the best performance without a priori knowledge of the Structure of the urban grid through which vehicles are being routed. The Bang-Bang algorithm is equivalent in performance but does require information about the grid's topology, and the loss probabilities. The shortest-path algorithm is unable to deal with uncertainty in the grid and provides unsatisfactory performance. Finally the FF RNN does respond to uncertainty and does not use any a priori information, however it is computationally costly and less effective than both Bang-Bang and RL RNN.

In our approach using the RL RNN, the agents have a lot of knowledge. One can Ask whether it realistic to assume that each agent will know of the experiences of other agents. Clearly, such information may only be partially available, or statistically available with some associated error probability. On the other hand, in many practical applications of simulation such as tactical simulations (military simulations), it is reasonable to assume that a given unit has a good sense of what is happening to all of its members. For instance, all tanks of a tank platoon can be in visual or radio contact with each other and can find out if anyone is hit. All aircraft in a flight formation are in radio, visual or radar contact with each other, and so on. Location information is now commonly available via GPS in many vehcles, and can be broadcast automatically to other members of a group. However information may only be partially available or correct. Thus in future work one could study the effect of incomplete information, of wrong information, of statistical errors and of delays incurred by the Las in acquiring information.

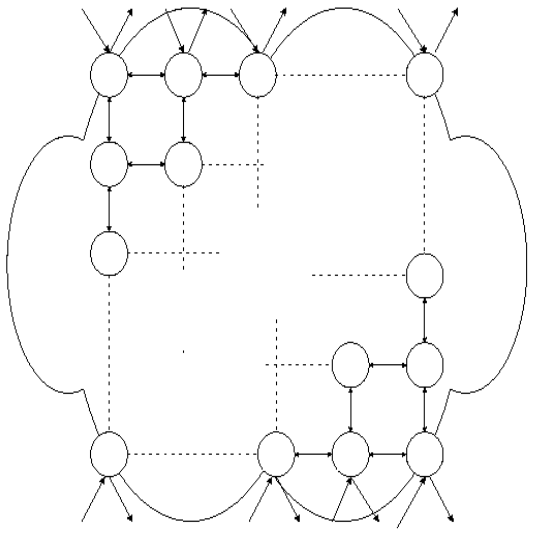


Figure 2. The Simulated Urban Grid

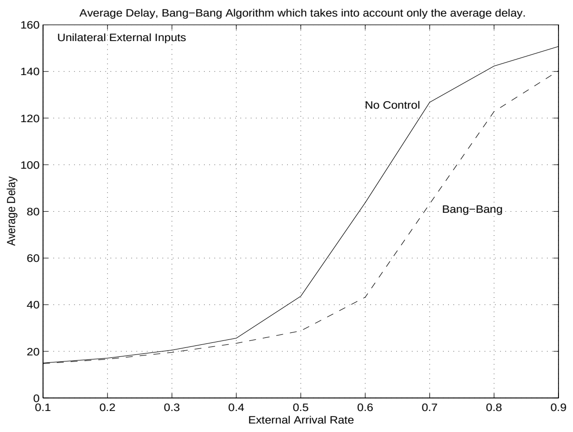


Figure 3. Comparison of Average Delay from Source to Destination for Bang-Bang Control using Estimated Delay in the Goal, and Fixed Routing

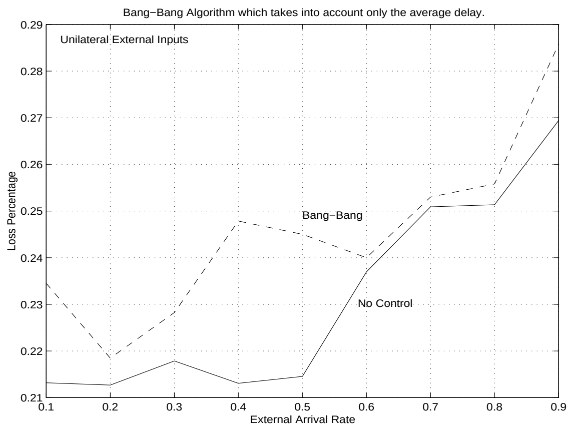


Figure 4. Comparison of Average Loss from Source to Destination for Bang-Bang Control using Estimated Delay in the Goal, and Fixed Routing

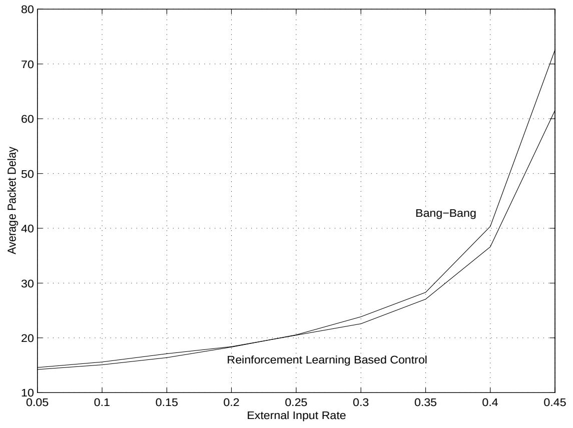


Figure 5. Comparison of Average Delay from Source to Destination for Bang-Bang Control and Reinforcement Based Control, using Estimated Delay and Estimated Loss in the Goal

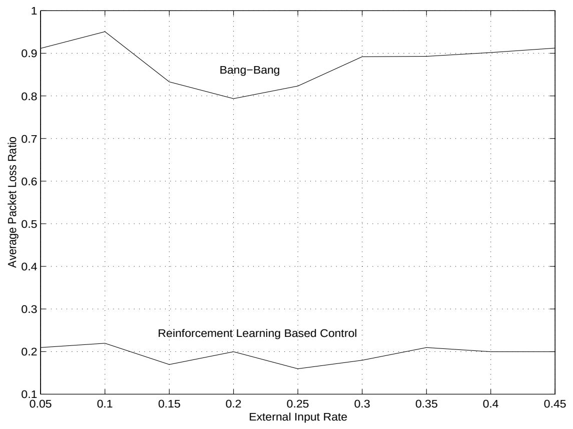


Figure 6. Comparison of Average Loss from Source to Destination for Bang-Bang Control and Reinforcement Based Control, using Estimated Delay and Estimated Loss in the Goal

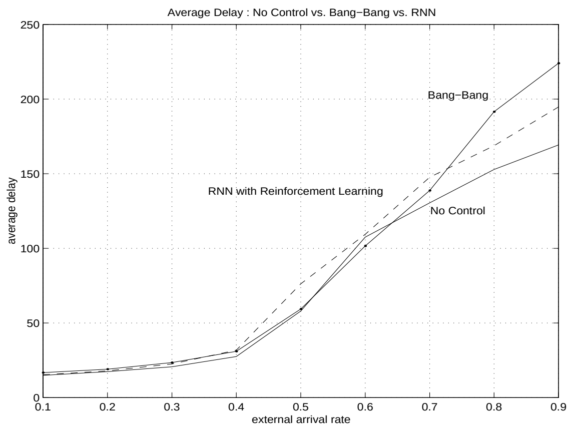


Figure 7. Reinforcement Learning Based Control using Delay and Loss as the Goal. Comparison of Average Transit Delay through the Urban Grid

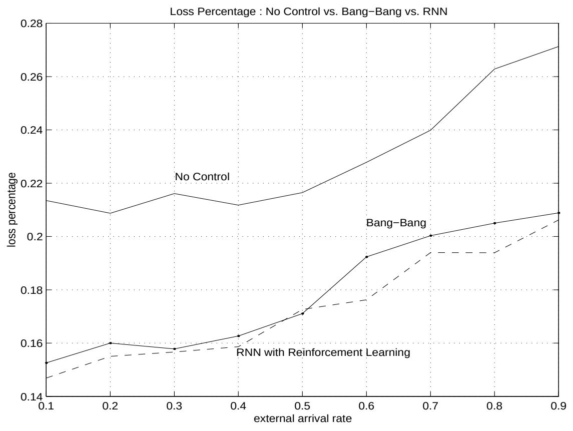


Figure 8. Reinforcement Learning Based Control using Delay and Loss as the Goal. Comparison of Average Loss through the Urban Grid

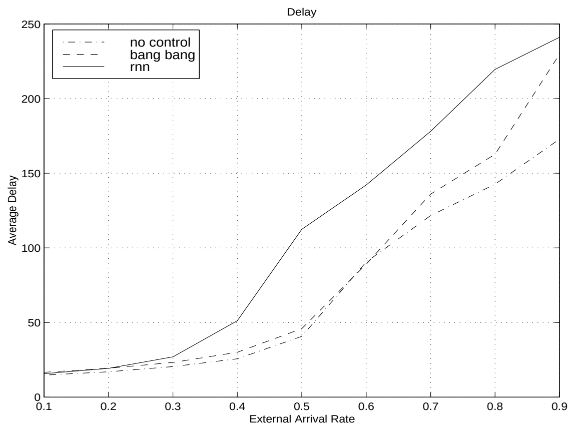


Figure 9. Feedforward Neural Network Learning Based Control using Delay and Loss as the Goal. Comparison of Average Transit Delay through the Urban Grid

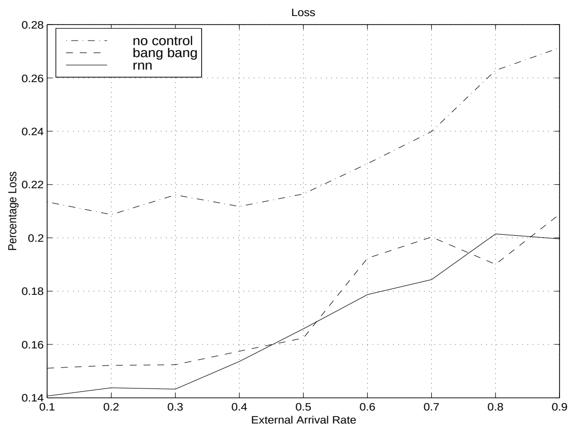


Figure 10. Feedforward Neural Network Learning Based Control using Delay and Loss as the Goal. Comparison of Average Loss through the Urban Grid

REFERENCES

[1] E. Gelenbe “Probabilistic automata with structural restrictions”, SWAT 1969 (IEEE Symp. on Switching and Automata Theory), also appeared as *On languages defined by linear probabilistic automata*, Information and Control, Vol. 18, February 1971.

[2] E. Gelenbe *A realizable model for stochastic sequential machines*, IEEE Trans. Computers, Vol. C-20, No. 2, pp. 199-204, February 1971.

[3] R. Viswanathan and K.S. Narendra *Comparison of expedient and optimal reinforcement schemes for learning systems*, J. Cybernetics, Vol. 2, pp 21-37, 1972.

[4] K.S. Narendra and P. Mars, *The use of learning algorithms in telephone traffic routing - a methodology*, Automatica, Vol. 19, pp. 495-502, 1983.

[5] R.S. Sutton “Learning to predict the methods of temporal difference”, *Machine Learning*, Vol. 3, pp. 9-44, 1988.

[6] E. Gelenbe (1993) “Learning in the recurrent random neural network”, *Neural Computation*, Vol. 5, No. 1, pp. 154-164, 1993.

[7] P. Mars, J.R. Chen, and R. Nambiar, *Learning Algorithms: Theory and Applications in Signal Processing, Control and Communications* , CRC Press, Boca Raton, 1996.

[8] R.W. Pew and A.S. Mavor (Editors) “Representing Human Behavior in Military Situations: Interim Report”, National Academy Press, Washington, D.C., 1997.

[9] Proceedings of the 7th Conference on Computer Generated Forces and Behavioral Representation, 1998.

[10] E. Gelenbe, Zhi-Hong Mao, Y. Da-Li (1999) “Function approximation with spiked random networks” *IEEE Trans. on Neural Networks*, Vol. 10, No. 1, pp. 3–9, 1999.

[11] E. Gelenbe “Modeling CGF with learning stochastic finite-state machines”, *Proc. 8–th Conference on Computer Generated Forces and Behavioral Representation, Simulation Interoperability Standards Organization*, pp. 113-115, Orlando, May 11-13, 1999.

[12] W. Foss, E. Gelenbe, P. Franceschini, M. Petty “Integrating Reinforcement Learning into Computer Generated Forces”, Technical Report, School of Computer Science and Institute for Simulation and Training, University of Central Florida, Orlando, September 1999.

[13] E. Gelenbe, W. and R. Franceschini, “Reinforcement Learning in ModSAF Reactive Behaviors”, *Proc. 9–th Conference on Computer Generated Forces and Behavioral Representation, Simulation Interoperability Standards Organization*, pp. 389–397, May 16-18, 2000.

[14] U. Halici, “Reinforcement learning with internal expectation for the random neural network”, *European Journal of Operations Research*, Vol. 126, No. 2, pp. 288-307, 2000.