Multi-Agent Reinforcement Learning for Simulating Pedestrian Navigation

Francisco Martinez-Gil
Departament d'Informàtica.
Universitat de València
Vicent Andrés Estellés s/n
46100, Burjassot, Valencia, Spain
Francisco.Martinez-Gil@uv.es

Miguel Lozano
Departament d'Informàtica.
Universitat de València
Vicent Andrés Estellés s/n
46100, Burjassot, Valencia, Spain
Miguel.Lozano@uv.es

Fernando Fernández
Computer Science Dpt.
Universidad Carlos III de Madrid
Avd. de la Universidad 30
28911 Leganés, Madrid, Spain
ffernand@inf.uc3m.es

ABSTRACT

In this paper we introduce a Multi-agent system that uses Reinforcement Learning (RL) techniques to learn local navigational behaviors to simulate virtual pedestrian groups. The aim of the paper is to study empirically the validity of RL to learn agent-based navigation controllers and their transfer capabilities when they are used in simulation environments with a higher number of agents than in the learned scenario. Two RL algorithms which use Vector Quantization (VQ) as the generalization method for the space state are presented. Both strategies are focused on obtaining a good vector quantizier that represents adequately the state space of the agents. We empirically state the convergence of both methods in our navigational Multi-agent learning domain. Besides, we use validation tools of pedestrian models to analyze the simulation results in the context of pedestrian dynamics. The simulations carried out, scaling up the number of agents in our environment (a closed room with a door through which the agents have to leave), have revealed that the basic characteristics of pedestrian movements have been learned.

Categories and Subject Descriptors

I.2.11, I.2.6, I.6.5 [**Multiagent Systems, Learning, Simulation and Modeling**]:

General Terms

Experimentation

Keywords

Reinforcement learning, pedestrian simulation, state generalization

1. INTRODUCTION

Controlling the movement of virtual agents groups to provide simulations with behavioral quality is an active research problem that has mainly attracted Artificial Intelligence and Computer Graphics techniques and methods. Multi-agent systems are a natural framework for this problem. A Multi-agent system is composed of autonomous entities named agents that interact each other sharing a common environment which they represent through a state and upon which they act with actions. In the simulation field they can be used in simulating virtual crowds or group-level behaviors for computer games, training systems and for studying architectural and urban designs. They constitute a local or agent-based approach to the problem opposite to macroscopic approaches in which the state of the system is described by mass densities and a corresponding locally averaged velocity [13]. In local approaches, the complexity of the problem, the dynamic environment or the possibility that unforeseen situations occur, make the solutions based on a priori design (like rule-based systems), difficult to tune. Besides, the replication of the same rule set in all the agents can create unrealistic simulations. In this context, Multi-agent learning systems, where each agent learns individually from its own experience, are an interesting alternative.

A RL-based agent learns by interacting with the environment. In response to the actions of the agent, the environment provides it with a reward signal that models the task to be learned. In the value-based family of RL algorithms, rewards are used by the agent to estimate the value of the decisions that it is taking in specific states. In this paper we focus on Temporal Difference Methods (TD Methods) [16] which have proven useful in a variety of domains.

Markov games are the natural extension of the single RL problem for Multi-agent RL systems (MARL). This framework allows to define the whole range of collective situations from fully-cooperative to non-cooperative games including general-sum games (see [10] for a review). Markov games use the joint actions (the cartesian product of the agents' actions) as part of the definition of the state-action space. Unfortunately, the exponential dependence in the number of agents and the necessity of converging to equilibrium as a basic stability requirement of these games, increase considerably the computational cost of the learning process. On the other hand, Multi-agent systems where the agents are independent learners have been studied in several works. In [11] and [14] independent RL processes are associated to robots in a group for grasping and navigation problems. [3] empirically shows that convergence is possible in cooperative settings for a Multi-agent system with independent RL processes. Recently, a small case study that applies independent learning in a Multi-agent RL problem for crowd simulation has been presented [18].

In this paper we study the feasibility of building a complex MARL oriented to learn realistic behaviors of virtual agents for pedestrian simulation. The aim is to introduce RL as a useful technique to find an agent-based control model to cope with the problem of simulating virtual agents that behave as groups of pedestrians. Realistic behavior means that agents appear to behave as pedestrians but they do not need necessarily conform to the characteristics of the models of real pedestrians. However, we use validation tools used in pedestrian models to quantify the overlaps between these models and our results.

Learning how to navigate in a continuous space towards a goal in an environment with other agents and using collision detection is not a trivial task. We propose two different learning approaches based on the Vector Quantization for Q-Learning(VQQL) [4] algorithm. These approaches are focused on finding a good state generalization as a key point to get realistic behaviors for the virtual agents. Also we study the scalability of the system respect to the number of agents. The strategy consists on learning the navigational problem with a moderate number of agents, and then transfer the value functions [17] to scale up to many agents.

The remainder of the paper is organized as follows. Section 2 describes the domain and the problem modeling. Section 3 describes the state generalization method. Section 4 describes the two algorithmic approaches to the learning problem. Section 5 focuses on the learning experiments. Section 6 shows the simulation and scaling results. Section 7 concludes and suggests future work.

2. THE DOMAIN

The scenario consist of a group of agents inside a closed room with a door. The agents have to learn to reach the door and leave the room. The agents detect collisions with other agents and walls which are relevant in the learning process. In order to resemble the model of pedestrians, the agents are constrained to move on the plane with a maximum velocity of 2.5 m/s. The environment is modeled like a two dimensional continuous plane where the room, defined with five walls, is placed. The cinematic module of the environment moves the agents across the plane using the velocity vector of each agent. The cinematic module actuates following a configurable clock signal so that the user can specify the number of decisions per second that the agent must take.

The definition of the states that the agents sensorize follows a deictic representation approach. The central premise underlying a deictic representation is that the agent only registers information about objects that are relevant to the task at hand [1] [19]. The selection of features that represent the state for the agent is critic for the success of learning. We have chosen features that provide local information about the agent cinematic state, the neighbor agents and the nearest walls, modeling the real situation of a pedestrian inside a group. As a result, the state for each agent is described by the features showed in Figure 1 and Table 1.

The number of sensorized neighbor agents and neighbor objects is configurable. In our evaluation, the number of sensorized neighbors is 7 and the number of sensorized static objects (walls) is 2. Therefore, in the evaluation, the state space has 28 features.

The agents' actions consist on modifying its vector velocity. The agent must set two values in each decision: the change ratio of the velocity module (increasing or reducing) and the change ratio of the angle (positive or negative) to modify the vector velocity. There are 8 different ratios plus the ‘no operation” option for both the module and the angle resulting in 81 possible actions.

Sag	Velocity module of the agent.
Av	Angle of the velocity vector relative to the reference line.
$Dgoal$	Distance to the goal.
$Srel_i$	Relative scalar velocity of the i-th nearest neighbor.
Dag_i	Distance to the i-th nearest neighbor.
Aag_i	Angle of the position of the i-th nearest neighbor relative to the reference line.
Dob_j	Distance to the j-th nearest static object (walls).
Aob_j	Angle of the position of the j-th nearest static object relative to the reference line.

Table 1: Description of the features of the agent's state. The reference line joins the agent's position with its goal position.

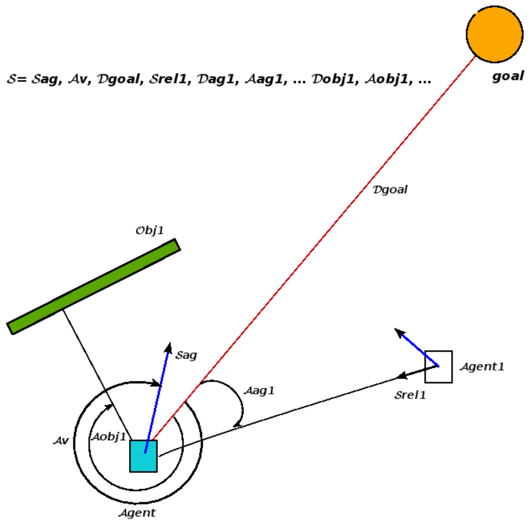

Figure 1: Agent's state description

3. STATE SPACE GENERALIZATION

The states are generalized using Vector Quantization (VQ), which has demonstrated to be an accurate approach for state space generalization and transfer learning [6]. A vector quantizier V_Q of dimension K and size N is a mapping from a vector space (in this paper the state space) in the K-dimensional euclidean space, R^k, into a finite set C containing N states. A sensorized state is aggregated to its nearest state in C, also named its prototype. Thus given C and a state $x \in R^k$ then $V_Q(x) = \arg\min_{y \in C}\{dist(x, y)\}$. The prototypes, that is, the members of C, are found using the Generalized Lloyd Algorithm (K-Means) and together with the euclidean metric, define the Voronoi regions of the state space [4, 6]. Vector Quantization makes possible the use of a table for representing the value function and therefore the use of classic TD algorithms like Q-learning or Sarsa.

Vector Quantization for Q-Learning(VQQL) [4] is a learning schema that uses VQ as the generalization method for states and the tabular version of Q-Learning for the learning process. Tabular Q-Learning uses a table (named Q) to represent the value function and takes as entries, a prototype and an action. For each entry of Q, the expected accumulated reward of being in state s and doing action a is stored. The process of updating the Q table with a new immediate reward r_t at instant t is named credit assignment operation, and it is performed using Equation 1.

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a \{Q(s_{t+1}, a)\} - Q(s_t, a_t)] \tag{1}$$

Where γ models the importance of the future reward and α is the learning rate. In VQQL, given a sensorized state s_t and a selected action a_t, the Q table entry to be updated is $(V_Q(s_t), a_t)$.

The use of VQ introduces two problems. The first one is to decide the number of prototypes to use, or the resolution of the state space. Typically, a very coarse discretization composed of a reduced number of prototypes has not enough expressiveness to represent optimal value functions. Too many states introduces again the generalization problem, although with a finite number of states. Therefore, VQQL has proven in most of the domains tested that intermediate values of the number of prototypes are more accurate than low or high values. In our experiments, different number of prototypes have been proved ($k = 512, 1024, 2048, 4096, 8192, 16384$). The best results were achieved with 4096 prototypes, and this configuration is used in all the experiments (despite better results may be obtained with different values).

The second problem of VQQL is how to generate the training data to learn the prototypes. The most straightforward way to get them is by generating random movements of the learning agents. However, in many domains, like crowd navigation, random movements of the agents generate biased data which are not representative enough to learn accurate value functions, as will be demonstrated empirically later. To deal with this problem we have defined two different learning strategies. Iterative VQQL (IT-VQQL) strategy and Incremental VQQL (IN-VQQL) strategy, which are described next.

4. INCREMENTAL VQQL AND ITERATIVE VQQL

The Iterative VQQL strategy, shown in Figure 2 is inspired in the adaptive K-Means family of algorithms. In adaptive K-Means, given a set of patterns or a density of probability that generate them, the problem is to define an optimal criterion that bias the centroids towards an optimal partition [2]. In our approach, the dataset generation procedure or the density of probability that generates the data is biased towards a better model of the problem by using a better learned policy. In IT-VQQL, we fix a number of agents (specifically 20) and the learning task is refined in each iteration of the learning process. We use the value functions learned in iteration step i to build a simulation and gather a new dataset. With the new dataset, the K-Means algorithm is used and a new set of prototypes is found, therefore a new V_Q^{i+1} is implemented. In the next iteration, the agents learn from scratch using the new vector quantizier V_Q^{m+1}, and so on. In the first iteration, the agents make a random walk, since the value functions are initialized to zero for all the state and action pairs. The IT-VQQL strategy ends when a maximum number of iterations are performed.

```
Multi-agent IT-VQQL

Entry: The number of learning agents, $p$, a deictic representation of the state space $S \in \mathcal{R}^k$, and a finite action space $A$.

1. Set $Q_0^1, \ldots, Q_0^p = 0$, $\forall s \in \mathcal{R}^k$, $\forall a \in A$
2. Set $i = 0$
3. Repeat:
   (a) Set $i = i + 1$
   (b) Generate a new vector quantizer, $V_Q^i$:
       • Generate a training set, $T^i$, by recording states visited by the agents when following and $\epsilon$-greedy exploration strategy over $Q_{i-1}^1, \ldots, Q_{i-1}^p$
       • Learn $V_Q^i$ by executing GLA algorithm over $T^i$
   (c) Learn the new Q tables for each agent, $Q_i^1, \ldots, Q_i^p$, following the Q-Learning algorithm
4. Until end condition is satisfied

Return: $Q_r^1, \ldots, Q_r^p$ and $V_Q^i$
```

Figure 2: Iterative VQQL Algorithm

IT-VQQL is a general algorithm that could be used to learn action policies in many domains. However, crowd navigation has an additional challenge, which is the difficulty to solve the problem from scratch. The Incremental VQQL strategy is based on a classic approach of transfer learning in which the problem to be learned is approximated by solving easier problems. The problem of finding a good set of prototypes that model the state space of a domain with a high number of agents (specifically 20) is tackled solving successive problems with less agents. Therefore, when using IN-VQQL, learning experiments are incremental in the number of agents. IN-VQQL, shown in Figure 3, can be seen as an adaptation of IT-VQQL, where the number of agents in the environment is increased in each iteration.

If the state representation of an agent includes features regarding with the neighbor agents, the IN-VQQL algorithm has the additional difficulty that the state spaces in the incremental learning processes have different dimensionalities. When the problem has been learned with m agents, and the next incremental problem with $m+1$ agents uses a state representation that sensorizes more neighbor agents, we need to use transfer learning techniques as performed for transfer learning in domains like Keepaway [5]. Specifically, a projection is used in order to get a new dataset in the new $m+1$-agents problem state space included in R^r. A projection can be understood as a selection of features $\Gamma : R^r \rightarrow R^s$ where $r > s$. The projection makes possible to use the vector quantizier V_Q^s and the value functions $Q_m^1, \ldots, Q_m^m$ learned in the m-agents problem, with the new higher-dimensional state space to collect data. Besides, the learned value functions are replicated to be used by the new set of agents. After the new dataset is obtained, a new set of prototypes using

```
Multi-agent IN-VQQL

Entry: The number of learning agents, $p$, a deictic representation of the state space $S \in \mathcal{R}^k$, and a finite action space $A$.

1. Set $Q^1_0$, $\forall s \in \mathcal{R}^k$, $\forall a \in A$
2. Set $i = 0$
3. Repeat:
   (a) Set $i = i + 1$
   (b) Generate a new vector quantizer, $V^i_Q$:
       • Generate a training set, $T^i$, by recording states visited by the current learning
         agents when following and $\epsilon$-greedy exploration strategy over
         $Q^1_{i-1}, \ldots, Q^{i-1}_{i-1}, Q^i_i = Q^j_{i-1} \quad j \in [1, i-1]$.
       • Learn $V^i_Q$ by executing GLA over $T^i$
       • Set $V^i_Q = V^i_Q \cup V^{i-1}_Q$
   (c) Learn the new Q tables for each agent, $Q^1_i, \ldots, Q^i_i$,
       following the Q-Learning algorithm
4. Until $i = r$

Return: $Q^1_r, \ldots, Q^p_r$ and $V^r_Q$
```

Figure 3: Incremental VQQL Algorithm

VQ is calculated and, therefore, a new vector quantizier V^r_Q is implemented to be used in a new learning process from scratch.

5. LEARNING EXPERIMENTS

In our Multi-agent learning system, the agents learn simultaneously. This means that the learning process is divided in episodes or trials and in each point of the process, all the agents are in the same trial. Besides, considering each trial of the learning process divided in discrete decision slots, all active agents take their decisions in the same decision slot before going to the next one. These characteristics warrant that the environment varies softly along the process, a desiderable property for the convergence of the learning process. In general, the number of decisions in a trial is different for each agent. An agent ends its trial when it reaches the door or after a fixed number of decisions have been taken.

The virtual environment is a 60x100 rectangle with an aperture that represents the door in the center of one of the small sides. The limits of the rectangle are defined by five walls. The agents are placed in the center of a bounding circumference with radius 0.4 meters that represents the area occupied by the "body" of the agent. The environment has collision detection; therefore the agents can crash against the walls and with other agents. In a collision, the agent stops and the other object or agent cannot go into the bounding circumference of the agent. The cinematic module moves each agent in the room according to its velocity. The simulation is divided in cycles limited by decision steps. The number of decisions per second is a parameter of the system. The state space is the same for all the agents. As stated in Section 2, the maximum number of sensorized neighbors is 7 and the fixed number of sensorized walls is 2. There is not a maximum distance of perception.

The behavior of the agents is modeled according to the immediate rewards listed in Table 2. As it can be seen, the payoff function reinforces the crash situations because the prevention of collisions is the main task that a navigation controller must take into account. Our model is related to pedestrian models that pay special attention to interactions between pedestrians like the Social-Force [7] and the Optimal-velocity models [12]. In these models, the velocity vector of a pedestrian is modified using forces parameterized by a desired velocity and the proximity to other pedestrians. In our model, where most of the state features are related with the sensorization of neighbor agents and walls, the negative immediate rewards provides information to learn the mentioned forces in terms of selecting an adequate action.

Crash against other agent	-4.0
Crash against a wall	-15.0
Reach the goal	+100.0
Default	0.0

Table 2: Inmediate rewards

5.1 Learning experiment for IT-VQQL

We have fixed a number of 20 learning agents for our experiments. It is a figure that trades-off the complexity of the problem to learn, and the necessity of a minimum density of agents to characterize the variety of possible interactions that can appear in a group. The dataset is gathered using a ϵ-greedy policy with $\epsilon = 0.07$ to palliate overfiting problems. Before using the K-Means algorithm, the collected data are standardized (each feature has zero mean and standard deviation equal to 1). We have detected empirically that our vector quantiziers did not give satisfactory results when the number of active agents became less than 5, mainly in the earlier iterations of the learning process. This can be explained considering the scarce number of occurrences for these configurations compared with data gathered from higher number of agents (mainly 20). The datasets obtained in simulation have few data with these configurations, creating a bad representation of the state space. Although the solution of this problem is a question of performing more iterations until we get a suitable dataset, we have improved the speed of learning by filling the void features of the state representation for these cases with random valid values. In this case, the vector quantizier always works with states with a full set of features. Thus, the behavior of these situations can be improved biasing the filling values using domain knowledge. The IN-VQQL algorithm has not this problem because the final vector quantizier is the union of quantiziers specialized in a specific number of agents. The curves and simulation results for the IT-VQQL in this paper have been performed using this approach. The performance curves for the iterative learning processes of IT-VQQL are displayed in Figure 4. The number of trials is low in the first curves because the goal is to get a better dataset for the vector quantizier. The learning processes uses a ϵ-greedy policy as the exploratory policy. The configuration of the main parameters of Q-learning for the curve 14 (the highest) are shown in Table 3. We use as a reference the learning curve of the basic VQQL algorithm with the same parameters shown in Table 3 excepting the α parameter that has the value $\alpha = 0.25$, consistent with the high number of trials carried out by this algorithm (see Figure 5).

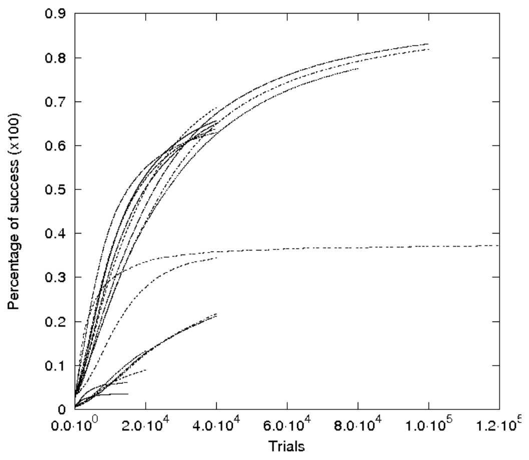

Figure 4: IT-VQQL performance curves for a 20 agents learning process. The longest curve corresponds to the reference curve for the VQQL algorithm. The rest of the curves, from down to up looking at the end of the curve: iterations $1, 2, 3, 5, 6, 4, 7, 8, 9, 11, 10, 12, 13, 15, 14$. The curves are averages of the data for 20 learning processes

Importance of future rewards (γ)	0.91
Initial rate for Exploratory policy (ϵ)	0.4
Learning rate (α)	0.35
Decitions per second	1

Table 3: Q-Learning parameters for IN-VQQL and IT-VQQL

The whole plot of the iterative learning process is displayed in Figure 5. These curves are those displayed in Figure 4. Note the improvement in performance along the increasing number of trials. The saw tooth pattern of the plot is due to the fact that learning in each iteration of IT-VQQL is performed from scratch, without transferring the value function from iteration to iteration [1].

5.2 Learning experiment for IN-VQQL

In the incremental approach the state space is variable in number of dimensions at different stages of the learning process (i.e. given a learning setting of 20 agents, at the beginning the state space will include the features to describe 7 neighbor agents, but when only one agent remains, its state space does not have features for the description of neighbors).

[1]Performing a value function transfer from each iteration to the following could be an interesting idea. However, given the vector quantizer used in each iteration is different, such transfer is not trivial

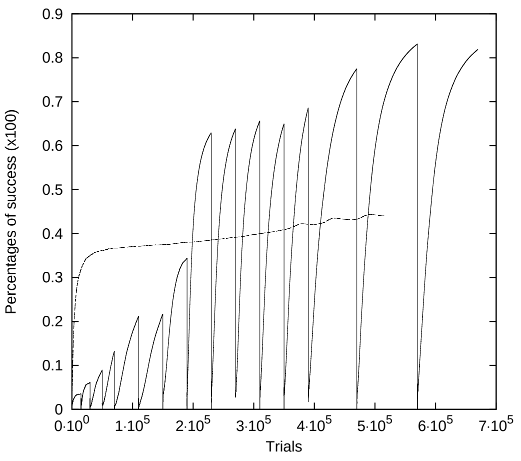

Figure 5: The whole learning process for the IT-VQQL strategy. The curves are sorted by iteration number inside the learning process. The dashed curve is for the VQQL algorithm.

In our incremental learning setting, the sequence of experiments performed has the following number of agents: $1, 2, 3, 4, 5, 6, 7, 8, 10$ and 20. The learning performance curves are plotted in Figure 6 together with the reference curve of VQQL (the lower curve). Note that, given a finite number of trials, the performance decreases with increasing the number of agents. It is caused by the increment of complexity of the problem to be learned. Therefore, the number of trials of the curves is incremented gradually with the number of learning agents. Besides, it is not necessary to await the asymptotic behavior of the curve, when the actual goal is to find a good (not optimal) vector quantizier that improves what already exists.

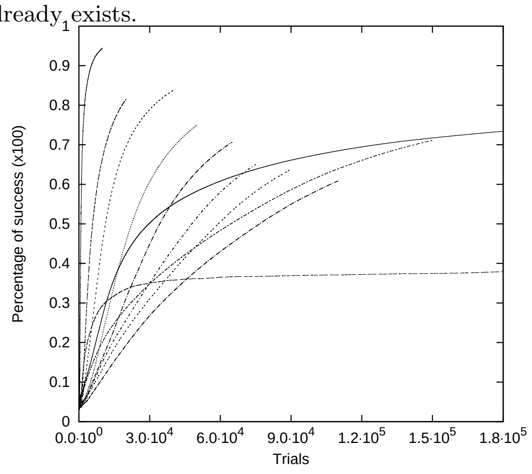

Figure 6: IN-VQQL performance curves for 20 agents learning process. From up to down with less than $1.5\,10^5$ trials, curves with number of agents $1, 2, 3, 4, 5, 6, 7, 8$. From down to up, with number of trials greater than $1.2\,10^5$, the curves for $10, 20$ agents. The dashed curve of final value near 0.4, corresponds to the VQQL algorithm. The curves are averages of the data for 20 learning processes.

The whole plot of the incremental learning process is displayed in Figure 7 and, also, the curve for the VQQL algorithm is plotted as a reference. Note the difference in the number of trials with Figure 5. Each element of the saw tooth pattern is a learning process with different number of agents. Although it seems to be an increment of performance from curve 8 to 9, it does not probably occur in the asymptotic regime.

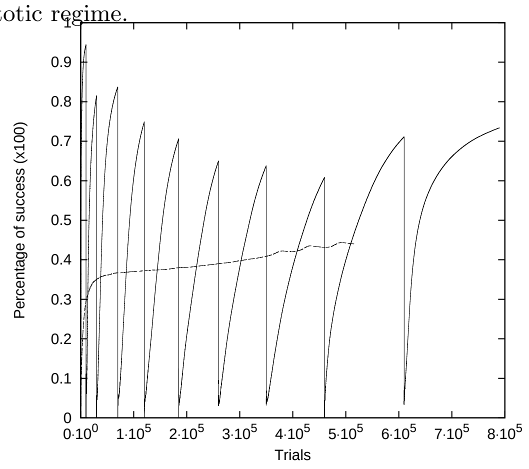

Figure 7: The whole iterative learning process for the IN-VQQL strategy. The curves are sorted by iteration number inside the learning process. The dashed curve is for the VQQL algorithm.

The configuration of the main parameters of Q-learning for the curve 10 corresponding to 20 agents is the same that the iterative curve number 14 and the VQQL reference algorithm and it is shown in Table 3.

6. SIMULATION RESULTS

In this section, we show the fundamental diagrams used in pedestrian dynamics to analyze the simulated behavior obtained by the RL agents. Pedestrian dynamics models usually focus on the qualitative reproduction of empirically observed collective phenomena, like the dynamical formation of lanes, bottlenecks, etc. In this sense, the main quantitative characteristics for the description of pedestrian streams are flow and density. Therefore, the main diagrams are derived from these functions. According to the definition shown in [8], the local density is obtained by averaging over a circular region of radius R. The local density at place $\vec{r} = (x, y)$ and time t was measured as

$$\rho(r,t) = \sum_j f(\vec{r_j}(t) - \vec{r}) \tag{2}$$

where $\vec{r_j}(t)$ are the positions of the pedestrians j in the surrounding of $\vec{r}$ and

$$f(\vec{r_j}(t) - \vec{r}) = \frac{1}{\pi R^2} exp[-||\vec{r_j} - \vec{r}||^2/R^2] \tag{3}$$

is a Gaussian, distance-dependent weight function. The local speeds have been defined via the weighted average

$$\vec{S}(\vec{r},t) = \frac{\sum_j \vec{v_j} f(\vec{r_j}(t) - \vec{r})}{\sum_j f(\vec{r_j}(t) - \vec{r})} \tag{4}$$

while the flow has been determined according to the fluid-dynamic formula

$$\vec{Q}(\vec{r},t) = \rho(\vec{r},t)\vec{S}(\vec{r},t) \tag{5}$$

Figure 8 shows this fundamental diagram for both IN/IT-VQQL in a simulation with 100 agents randomly placed in the environment. The first diagram (left column) shows the average of the local speeds $\vec{S}(\vec{r},t)$ as a function of the local density $\rho(r,t)$ for both cases. We have measured the local density capturing the positions of the agents in a circumference (R = 1) near the exit, so we guarantee a minimum flow during the simulation.

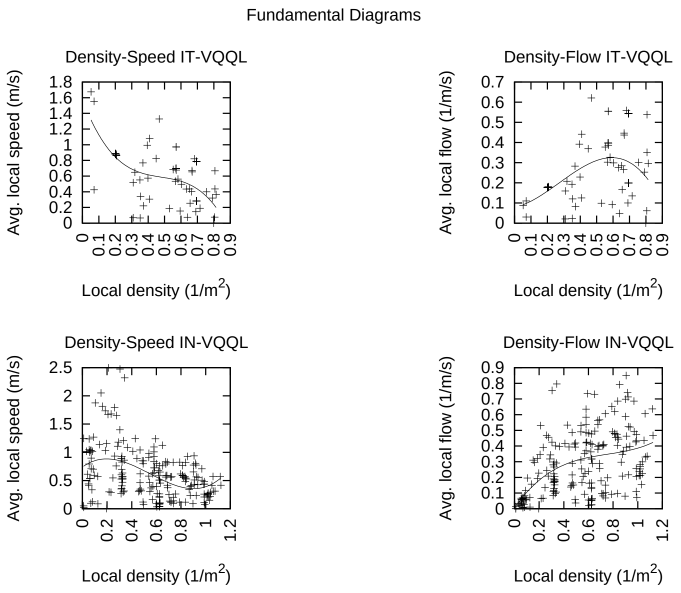

Figure 8: Fundamental diagrams

The low density values offered ($\rho < 1.2$) are likely due to the same effect described in [15] which states that when the initial positions of the pedestrians are not near from the bottleneck center, the density will decrease due to the movement from the initial position to this point, resulting in a smaller density. Furthermore, our aim here is neither a deep characterization of our agent nor a comparison with the other pedestrian models/data, but to analyze the simulation results from a behavioral point of view when scaling up the models learned. The scalability problem (increasing the number of agents without losing behavioral quality during the simulation) involves a good generalization of the learned model, as it must face new situations, that properly managed, will lead it to reach its goal.

The first column on Figure 8 shows how the RL controllers (IT/IN) have learned to reduce their speed according to the density perceived. In both cases, the data plotted indicate that different kind of speed-reduction behaviors can be produced while the fitting functions (used only for clarity purpose) let us to observe that the shape of the curves and their tendency can be considered as reasonable.

However, there are several differences among the RL models shown in this column. Firstly, the IT model shows a reduced number of points comparing with the IN model. The points plotted here can be viewed as different navigational decisions (speed regulations) which lead the agents to reach their goal. In this sense, the IN learned controllers seem to be able to better generalize and scale the problem. On the other side, the IT model results at this diagram are possibly indicating that an overfiting situation may be happening, due to an excessive dependency of the situations learned. To confirm this hypothesis we have measured the number of agents that finally evacuate the environment in both methods. The results are shown in tables 4 and 5.

Agents	Fails (%)	σ	$\overline{v}$ (m/s)	σ
20	21.5	5.7	1.64	0.82
40	13.95	3.99	1.57	0.83
60	13.63	4.42	1.40	0.83
80	19.13	3.67	1.27	0.81
100	26.75	4.35	1.19	0.79

Table 4: Performance results and average velocity with the number of agents for the IN-VQQL algorithm. Data are averages of 20 agents and 100 trials.

Agents	Fails (%)	σ	$\overline{v}$(m/s)	σ
20	9.25	2.68	1.99	0.73
40	25.32	3.62	1.84	0.77
60	39.1	5.25	1.69	0.79
80	46.42	4.59	1.62	0.78
100	56.98	4.8	1.54	0.77

Table 5: Performance results and average velocity with the number of agents for the IT-VQQL algorithm. Data are averages of 20 agents and 100 trials.

A better performance of IT-VQQL vs. IN-VQQL has been observed in the learning case (20 agents). The performance of IT-VQQL is degradated faster than in the IN-VQQL when the number of agents grows, and generalization capabilities are needed. Obviously, the VQ state generalizer has a decisive influence in these results. Also note the higher averaged velocity of the agents that use the IT-VQQL learned controllers that can produce problems when scaling up the number of agents.

The second column on Figure 8 shows the relation among the simulated densities and flows. The diagram reveals that for the densities considered, the maximum flow is still no reached so the growing trend of the curve has not ended.

We have also calculated the density maps associated to these simulations. Here, the plane is tiled with a grid and the number of agents per tile unit is counted. Therefore, it is a histogram that represents the number of agents per tile during the simulation. It gives the information of the level of occupation of different zones of the plane so it is interesting to know the bottleneck shape (Figures 9 and 10). These figures show the typical concentration around the exit and a continuous increasing flow towards the door, represented as a gray degradation. An interesting thing to see is that isolated dark grey tiles can be interpreted as places where crashes occur and, therefore, where the agents are stopped. Note that the greatest concentration of these isolated tiles are near the walls, where crashes are more likely. The comparison between both RL models reveals the area where the IT-VQQL model crashes frequently near the goal. This can prevent the IT-VQQL model agents from reaching the goal.

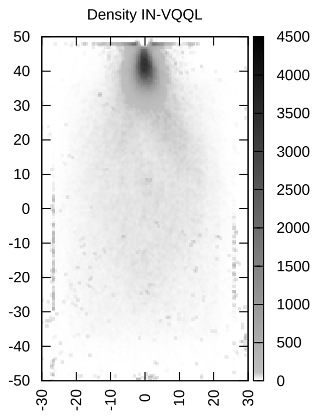

Figure 9: IN-VQQL Density Map for 100 agents. Points display data of 100 simulations.

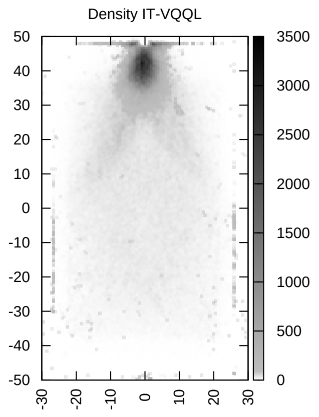

Figure 10: IT-VQQL Density Map for 100 agents. Points display data of 100 simulations.

7. CONCLUSIONS AND FUTURE WORK

- The experiments show that the Multi-agent navigation problem can be faced using reinforcement learning algorithms. The results have revealed that important characteristics, like the speed control, remain when scaling to a larger number of agents without additional learning.

- The results indicate that the IT-VQQL learning schema learns faster than the IN-VQQL schema. However, when scaling up the number of agents, the IT-VQQL schema overfits the learning problem giving worse simulation results than the IN-VQQl schema. This could be caused by the successive refinements over the same learning scenario in IT-VQQL.

- Classic TD single-agent algorithms like Q-learning have been proven to converge in the limit with discrete state and action spaces and stationary environments [9]. Convergence in the limit means in practice that the learned value functions are suboptimal. This fact does not need to be necessarily a handicap in pedestrian simulations because, in real life, people's behaviors do not use optimality as the main criteria. On the other hand, Multi-agent learning systems are inherently non-stationary. The convergence is a domain property that needs to be studied case-by-case. With our results we have proved empirically that RL techniques give sufficient quality in this domain and, likely, its use could be extended to other pedestrian scenarios.

- Future work:

  It is possible to unify the two learning schemas in a single algorithmic schema. Based on the IN-VQQL algorithm it is possible to consider each incremental problem subjected to a refining process. Considering the results exposed above, a trade-off should be applied in this scheme between adaptation and specialization capabilities. Besides, classic strategies of transfer learning could also be applied for the VQ state generalizer and for the learned value functions in different steps of this unified schema. Other aspect of interest is the use of other state generalization methods (i.e. tile coding) to compare the results.

  On the other hand it is necessary to study the response in simulation with a learning scenario with more agents. That is, to study the performance when the number of learning agents are 40, 80, etc. It is plausible to expect an asymptotic behavior in the scaling capabilities in this context. Other interesting subject is the study of the capability of RL in the emergence of collective pedestrian behaviors. There are several classic well studied self-organization phenomenon that appear in pedestrian groups inside certain scenarios (like the zipper effect in front of a bottleneck or the formation of lanes inside a corridor) that could be studied.

Acknowledgements

This work has been supported by the Spanish MEC under grant TIN2009-14475-C04-04.

8. REFERENCES

[1] P. Agre and D. Chapman. Pengi: An implementation of a theory of activity. In *Proceedings of the Sixth National Conference on Artificial Intelligence*, pages 268–272. Morgan Kaufmann, 1987.

[2] C. Chinrungrueng and C. Sequin. Optimal adaptive k-means algorithm with dynamic adjustment of learning rate. *Neural Networks, IEEE Transactions on*, 6(1):157 –169, Jan. 1995.

[3] C. Claus and C. Boutilier. The dynamics of reinforcement learning in cooperative multiagent systems. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pages 746–752. AAAI Press, 1998.

[4] F. Fernández and D. Borrajo. Two steps reinforcement learning. *International Journal of Intelligent Systems*, 23(2):213–245, 2008.

[5] F. Fernández, J. García, and M. Veloso. Probabilistic policy reuse for inter-task transfer learning. *Robotics and Autonomous Systems*, 58(7):866–871, 2010.

[6] J. García, I. López-Bueno, F. Fernández, and D. Borrajo. *A Comparative Study of Discretization Approaches for State Space Generalization in the Keepaway Soccer Task. In Reinforcement Learning: Algorithms, Implementations and Aplications.* Nova Science Publishers, 2010.

[7] D. Hebing and P. Molnár. Social force model for pedestrian dynamics. *Physics Review E*, pages 4282–4286, 1995.

[8] A. Johansson, D. Helbing, and P. K. Shukla. Specification of the social force pedestrian model by evolutionary adjustment to video tracking data. *Advances in Complex Systems*, 10(2):271–288, 2007.

[9] L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: A survey. *Int. Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[10] R. B. L. Busoniu and B. D. Schutter. A comprehensive survey of multi-agent reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics Part C: Applications and Reviews*, 38:156–172, 2008.

[11] M. J. Mataric. Learning to behave socially. In *From Animals to Animats: International Conference on Simulation of Adaptive Behavior*, pages 453–462. MIT Press, 1994.

[12] A. Nakayama, Y. Sugiyama, and K. Hasebe. Instability of pedestrian flow and phase structure in a two–dimensional optimal velocity model. *Pedestrian and Evacuation Dynamics 2005*, pages 321–332, Springer Verlag 2007.

[13] A. Schadschneider, W. Klingsch, H. Klüpfel, T. Kretz, C. Rogsch, and A. Seyfried. Evacuation dynamics: Empirical results, modeling and applications. In *Encyclopedia of Complexity and Systems Science*, pages 3142–3176. 2009.

[14] S. Sen and M. Sekaran. Multiagent coordination with learning classifier systems. In *IJCAI95 Workshop on Adaptation and Learning in Multiagent Systems*, pages 218–233. Springer Verlag, 1996.

[15] A. Seyfried, O. Passon, B. Steffen, M. Boltes, T. Rupprecht, and W. Klingsch. New insights into pedestrian flow through bottlenecks. *Transportation Science*, 43(3):395–406, August 2009.

[16] R. S. Sutton. Learning to predict by the methods of temporal differences. In *Machine Learning*, pages 9–44. Kluwer Academic Publishers, 1988.

[17] M. E. Taylor and P. Stone. Behavior transfer for value-function-based reinforcement learning. In *The Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, July 2005.

[18] L. Torrey. Crowd simulation via multi-agent reinforcement learning. In *Proceedings of the Sixth AAAI Conference On Artificial Intelligence and Interactive Digital Entertainment.* AAAI Press, Menlo Park, California, 2010.

[19] S. D. Whitehead and D. H. Ballard. Learning to perceive and act by trial and error. *Machine Learning*, pages 45–83, 1991.