2013 12th International Conference on Machine Learning and Applications

Coordinated Reinforcement Learning Agents in a Multi-Agent Virtual Environment

William Sause, Ph.D.
Graduate School of Computer and Information Sciences
Nova Southeastern University
Fort Lauderdale, USA
sause@nova.edu

***Abstract*—This research presents a framework for coordinating multiple intelligent agents within a single virtual environment. Coordination is accomplished via a "next available agent" scheme while learning is achieved through the use of the Q-learning and Sarsa temporal difference reinforcement learning algorithms. To assess the effectiveness of each learning algorithm, experiments were conducted that measured an agent's ability to learn tasks in a static and dynamic environment while using both a fixed (FEP) and variable (VEP) ϵ-greedy probability rate. Results show that Sarsa, on average, outperformed Q-learning in almost all experiments. Overall, VEP resulted in higher percentages of successes and optimal successes than FEP, and showed convergence to the optimal policy when measuring the average number of time steps per episode.**

Keywords—Reinforcement learning, virtual environments, intelligent agents

I. INTRODUCTION

The growing popularity of online virtual communities demands the presence of intelligent agents to assist users in their daily online activities (e.g., exploring, shopping, and socializing). As these virtual environments become more crowded, multiple agents are needed to support the increasing number of users. Multi-agent environments, however, can suffer from the problem of resource competition among agents. It is therefore necessary that agents within multi-agent environments include a coordination mechanism to prevent unrealistic behaviors (such as performing the same tasks at the same time). Moreover, it is essential that these agents exhibit some form of intelligence, or the ability to learn, to support realism as well as to eliminate the need for developers to write separate scripts for each task the agents are required to perform. This research presents a coordinated reinforcement learning framework which can be used to develop task-oriented intelligent agents in multi-agent virtual environments. The framework contains a combination of a "next available agent" coordination model and a reinforcement learning model consisting of existing temporal difference reinforcement learning algorithms. Furthermore, the framework supports evaluations of reinforcement learning algorithms to determine which methods are best suited for task-oriented intelligent agents in dynamic, multi-agent virtual environments.

II. COORDINATION

Coordination is accomplished via a "next available agent" model. A database is utilized to obtain information regarding the agent such as the agent's identification and start location. This information is used to ensure the maximum number of agents in the environment is not exceeded and that each agent is placed in its proper location. Also stored in the database is the state of the agent (i.e., busy or available). This is necessary since the agent that had arrived first may be busy working on another task; therefore, another agent will need to take on the assigned task if the need arises. This ensures that all agents work cooperatively and thus behave more realistically.

III. LEARNING

Learning is achieved by use of a reinforcement learning model. Reinforcement learning is the learning of a behavior through trial-and-error interactions within a dynamic environment consisting of a set of states, a set of actions, and a set of reward signals. There are several reinforcement learning algorithms available; however, this present research will focus on those algorithms that fall under the category of temporal difference learning. Temporal difference learning can be broken down into two categories: on-policy and off-policy. A policy is used to select an action that an agent should take from any given state. The goal of a reinforcement learning agent is to find an optimal policy π^*. A policy is said to be optimal if that policy yields the highest expected return. On-policy methods attempt to find the optimal policy and follow this policy while exploring the environment. In contrast, off-policy methods learn an optimal policy which may be different from the policy being explored [1].

A. Sarsa

Sarsa is an example of an on-policy temporal difference learning algorithm. The Sarsa algorithm learns an optimal policy by approximating the optimal action-value function Q^* [2]. The action-value function $Q(s, a)$ returns the value of selecting action a in state s, and is calculated after every transition from a nonterminal state [1]. The action value of the state-action pair to which the agent is transitioning $Q(s', a')$ is used when calculating the action value of the current state-action pair:

$$Q(s, a) \leftarrow Q(s, a) + \alpha\,[r + \gamma\, Q(s', a') - Q(s, a)] \qquad (1)$$

B. Q-learning

An example of an off-policy temporal difference learning algorithm is the Q-learning algorithm. Similar to the Sarsa algorithm, the action-value function $Q(s, a)$ is updated at each

978-0-7695-5144-9/13 $31.00 © 2013 IEEE
DOI 10.1109/ICMLA.2013.46

step of an episode as the agent transitions from a nonterminal state. In contrast to Sarsa, a Q-learning agent will always utilize the maximum action value of the next state-action pair $Q(s', a')$ when computing the action value of the current state-action pair $Q(s, a)$ even if the action a' that returns the maximum action-value is not necessarily the action taken by the agent. Therefore, the agent can continue to explore (i.e., selecting actions at random) while an optimal policy is found:

$$Q(s, a) \leftarrow Q(s, a) + \alpha\,[r + \gamma \max_{a'} Q(s', a') - Q(s, a)] \quad (2)$$

C. Learning Parameters

Learning parameters utilized by the action-value function include the learning rate α, the reward value r, and the discount rate γ. The learning rate α is a value between 0 and 1 that indicates how much the action-value for a state is adjusted when an action is performed. A state's reward r is the value used in deciding which action is most desirable from a given state. Temporal difference methods typically define rewards using a rule-based representation that comprises a set of conditions about the subsets of the state space and numerical values for states in those subsets [3]. For this research, the agent is assigned a reward for a given state based on the current task being performed and the resource currently being utilized to complete the task:

$$R_{(t)} = \begin{cases} 1 & \text{if } S_{(t)} \text{ contains a resource required to complete task} \\ n & \text{if } S_{(t)} \text{ is goal state and resources were visited in order} \\ 0 & \text{otherwise} \end{cases}$$

where n denotes a large number (e.g., $n >= 100$). Lastly, the discount rate γ is a value between 0 and 1 that specifies an agent's interest in future rewards.

A common element that threads through these algorithms is the ϵ-greedy factor. An agent utilizing ϵ-greedy action selection has ϵ chance of selecting a random action rather than the current best action from any state [4].

IV. Experiments

Three tasks were presented to the agents in this research: (1) a basic sandwich making (BSM) task which included a plate, bread, and meat; (2) a complex sandwich making (CSM) task that consisted of the BSM resources plus an additional resource of ketchup; and (3) a dynamic sandwich making (DSM) task that contained the same resources as CSM. The ketchup resource for DSM was added after 5000 episodes, by which time the agent realized the optimal policy for BSM a significant number of times (i.e., at least about 100 times in all experiments). An episode is a sequence of steps taken by the agent from the start state to the goal state. The optimal policy for BSM includes exactly four steps in a prescribed order. These steps are: (1) pick up a plate from the dish rack, (2) get bread from the breadbox, (3) get meat from the grill, and (4) deliver the plate with the completed sandwich to the customer's table. The optimal policies for the CSM and DSM tasks comprise of the same steps as BSM, with an additional step of applying ketchup to the sandwich after step (3). Using the two temporal difference reinforcement learning algorithms included in this study, the goal of the agent was to learn the optimal policy, thus prompting the agent to visit each resource required to make the sandwich in the correct order.

The Q-learning and Sarsa reinforcement learning algorithms were evaluated using three forms of measurement: (1) the percentage of successful episodes (i.e., the sandwich making task was completed), (2) the percentage of successful episodes where the optimal policy was reached (i.e., the sandwich making task was completed in exactly 4 time steps for BSM, and exactly 5 time steps for CSM and DSM), and (3) the average number of time steps taken to complete a successful episode. The algorithms used to generate the results utilized two ϵ-greedy policies for action selection: a fixed ϵ-greedy policy (FEP) and a variable ϵ-greedy policy (VEP). Applying the ϵ-greedy policy to the Q-learning and Sarsa algorithms enabled the agent to explore its environment with a small percentage of probability. For FEP, this probability was fixed at 0.1 which afforded the agent 10% exploration at each step in the learning process. Results show that this probability helped the agent reach the optimal policy early; however, once the agent was able to select all necessary actions to complete the sandwich making task with certainty, only a slight improvement was at times shown in its performance. This is due to the ϵ-greedy factor which drives the agent to continue learning after reaching the optimal policy. To encourage improvement in the agent's performance after the optimal policy was found, it was necessary to apply VEP. With VEP, ϵ was initialized to 0.1 and decreased by 0.01 at each episode where the optimal policy was reached. Reducing the value of ϵ as the agent learns enables major exploration at the beginning learning phase and more exploitation of the acquired knowledge during the advanced learning phase [5]. The minimum threshold at which ϵ was no longer reduced was 0. When ϵ is equal to 0, learning no longer occurs and the agent continues to follow the learned policy. It was therefore necessary in the research to reset the value of ϵ to 0.1 when the additional resource of ketchup was added in the DSM task. This afforded the agent the ability to adapt to the changing environment after it had achieved the optimal policy a significant number of times while performing BSM.

A series of 10 trial runs were performed with each run consisting of 5000 episodes for BSM. Likewise, 10 trial runs were performed for CSM and DSM; however, these experiments consisted of 50000 episodes. Results were generated using the cumulative average of the number of successful episodes performed by an agent, the cumulative average of the number of optimally successful episodes performed by an agent, and the average number of time steps taken to complete a successful episode. Each experiment used a learning rate α of 0.5 and discount rate γ of 0.5. There is no standard way to define these parameters to guarantee the convergence of learning since they are closely connected with the particular application [5]. For the application of dynamic virtual environments, it was found that the value of 0.5 for both learning parameters resulted in the highest percentage of successful episodes and the minimum number of time steps per successful episode.

V. RESULTS

The graphs in Figures 3 through 8 illustrate the results of the experiments. Tests conducted for this research demonstrated that, on average, a Sarsa agent completes more episodes successfully and in less time steps than that of a Q-learning agent. The algorithms were compared when learning the BSM task and the more complex task of sandwich making with the additional resource of ketchup. This complex task was performed both from scratch (CSM) as well as after reaching the optimal policy a significant number of times for the BSM task (DSM). The experiments that included the additional resource required a tenfold increase in the number of episodes per trial run to generate results. This was a result of the exponential growth of the state space when adding a single resource to a virtual environment. Adding the resource of ketchup created an additional action for each existing state (i.e., go to the ketchup bottle from the current state and apply the ketchup) as well as an additional state-action pair consisting of each action from the ketchup state. Results showed that the agent performs worse at the beginning of the learning process when learning DSM than it does with CSM. This was due to the agent following the learned policy for BSM while the DSM task was learned. As a result, DSM required more episodes than CSM before the agent was confident in performing the task with the additional resource of ketchup. Overall, experiments utilizing VEP resulted in higher percentages of successes and optimal successes than that of FEP, and showed convergence to the optimal policy when measuring the average number of time steps per successful episode.

VI. CONCLUSION

The framework presented in this research offers an effective means of coordinating multiple learning agents within a single virtual environment. Using a "next available agent" coordination model, the agents exhibited realistic behavior by working in coordination even when faced with multiple orders (i.e., greater than the number of agents) simultaneously. Figures 1 and 2 illustrate this coordination in a virtual environment consisting of four agents. The learning aspect of the framework enabled the agents to learn and perform an assigned task by utilizing either the Q-learning or Sarsa reinforcement learning algorithm. Results showed that, in general, the Sarsa algorithm is more suitable for task-oriented intelligent agents in dynamic virtual environments than the Q-learning algorithm. This is due to Sarsa's on-policy control strategy in which the actions chosen by the agent are selected from the learned policy. The Sarsa agent therefore, in most instances, learned the optimal policy in fewer episodes than the Q-learning agent. This efficiency is a desirable feature in multi-user virtual environments such as the café environment developed in this research where the user's expectation is to be served promptly upon placing an order. Results also illustrated that Sarsa is most effective in dynamic virtual environments that contain complex tasks with fewer than four resources. Adding additional resources increases the state space exponentially and, in turn, hinders learning. Applying a technique such as a neural network function approximator to solve the issue of large state spaces is a future goal of this research. This method would entail replacing the lookup table with a neural network with state s and action a as input variables, and the Q value as the target output [6].

REFERENCES

[1] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. Cambridge, MA: The MIT Press, 1998.

[2] K. Merrick, "Modelling motivation for experience-based attention focus in reinforcement learning," Unpublished doctoral dissertation, University of Sydney, Australia, 2007.

[3] K. Merrick and M. Maher, *Motivated reinforcement learning: Curious characters for multiuser games*. Berlin, Germany: Springer, 2009.

[4] L. M. Zamstein et al., "Koolio: Path planning using reinforcement learning on a real robot platform," *Florida Conference on Recent Advances in Robotics*, Miami, Florida, 2006.

[5] E. Menegatti et al., "Explicit knowledge distribution in an omnidirectional distributed vision system," *Proceedings of the International Conference on Intelligent Robots and Systems*, Sendai, Japan, 2004, pp. 2743-2749.

[6] W. Ertel, *Introduction to artificial intelligence*. London: Springer, 2011.

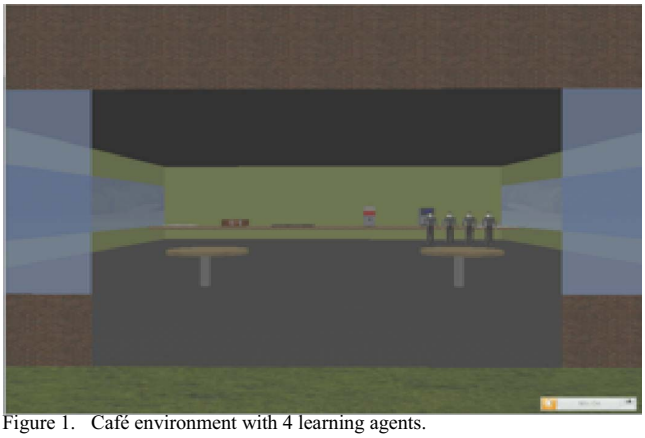

Figure 1. Café environment with 4 learning agents.

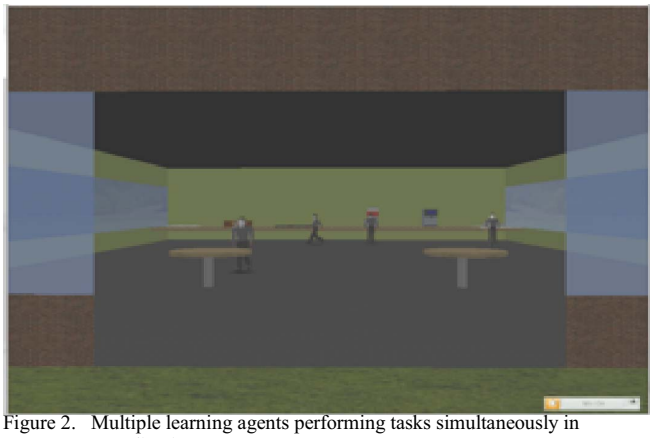

Figure 2. Multiple learning agents performing tasks simultaneously in coordination.

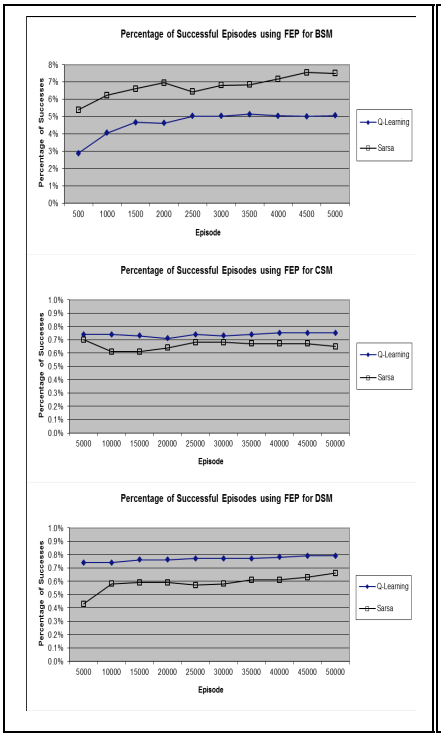

Figure 3. Results for percentage of successful episodes using FEP.

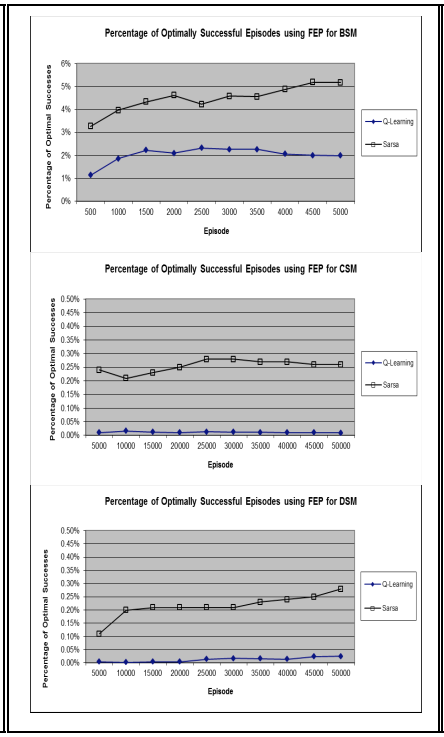

Figure 4. Results for percentage of optimally successful episodes using FEP.

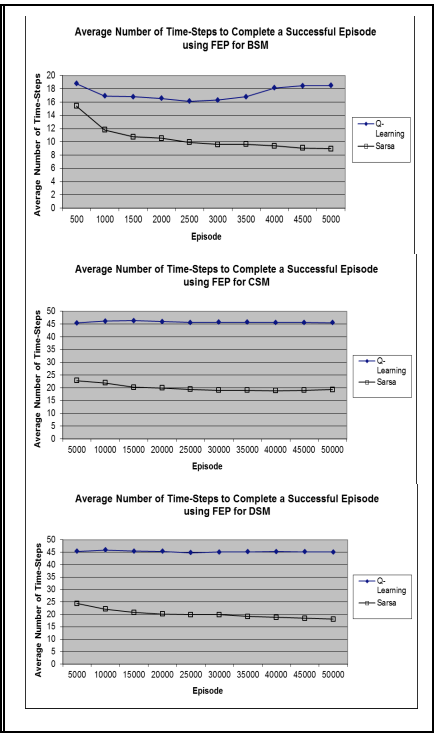

Figure 5. Results for average number of time step steps to complete a successful episode using FEP.

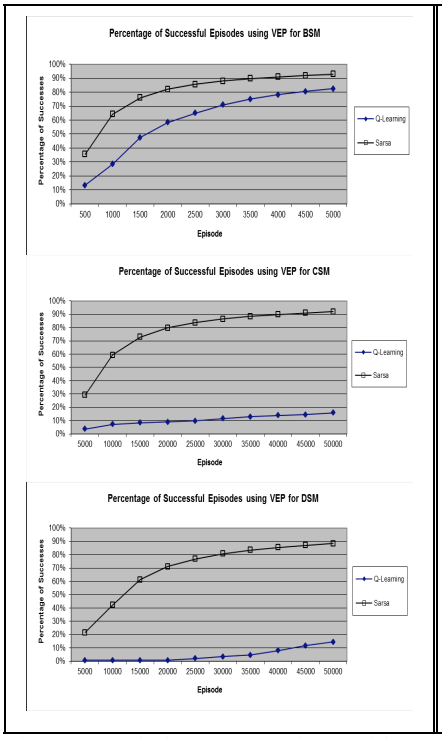

Figure 6. Results for percentage of successful episodes using VEP

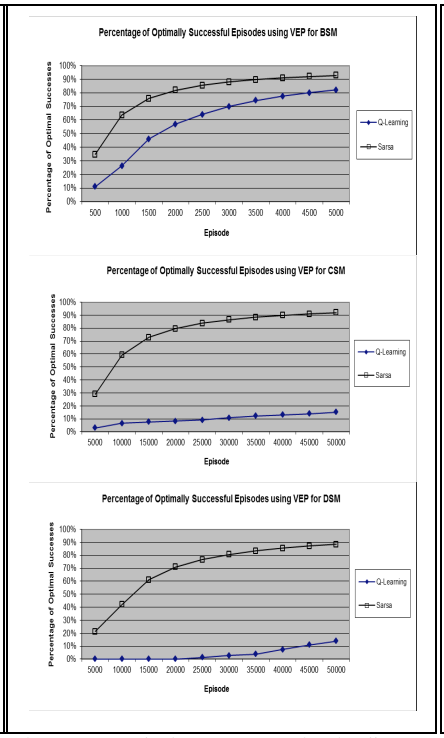

Figure 7. Results for percentage of optimally successful episodes using VEP.

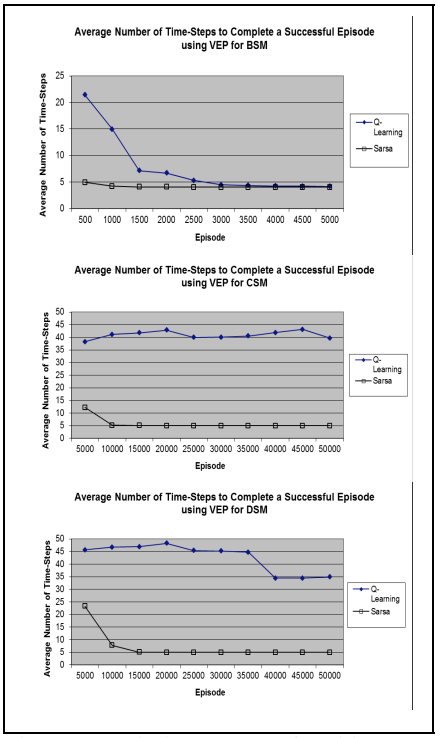

Figure 8. Results for average number of time steps to complete a successful episode using VEP.