2011

Reinforcement learning of competitive and cooperative skills in soccer agents

Jinsong Leng
Edith Cowan University

Chee Lim

10.1016/j.asoc.2010.04.007

Manuscript Number: ASOC-D-08-00557R1

Title: Reinforcement Learning of Competitive and Cooperative Skills in Soccer Agents

Article Type: Full Length Article

Keywords: Reinforcement Learning, Temporal Difference Learning, On-policy and off-policy, Eligibility Traces, Performance, Convergence

Corresponding Author: Dr. Jinsong Leng, PhD

Corresponding Author's Institution: Edith Cowan University

First Author: Jinsong Leng, PhD

Order of Authors: Jinsong Leng, PhD; Chee Peng Lim, PhD

Abstract: The main aim of this paper is to provide a comprehensive numerical analysis on the efficiency of various Reinforcement Learning (RL) techniques in an agent-based soccer game. The SoccerBots is employed as a simulation testbed to analyze the effectiveness of RL techniques under various scenarios. A hybrid agent teaming framework for investigating agent team architecture, learning abilities, and other specific behaviours is presented. Novel RL algorithms to verify the competitive and cooperative learning abilities of goal-oriented agents for decision-making are developed. In particular, the tile coding (TC) technique, a function approximation approach, is used to prevent the state space from growing exponentially, hence avoiding the curse of dimensionality. The underlying mechanism of eligibility traces is evaluated in terms of on-policy and off-policy procedures, as well as accumulating traces and replacing traces. The results obtained are analyzed, and implications of the results towards agent teaming and learning are discussed.

Response to Reviewer Comments

I. RESPONSE TO THE QUERIES OF REVIEWERS

1) As properly suggested by the reviewer, we reconstructed the Section 2 by deleting some parts in Section 2.1, and merging the previous Section 2.3 and Section 2.4. Please see the Section 2 for details.

2) As per the suggestion of the reviewer, we revised the Section 3 and deleted the previous Section 3.1. Please see Section 3 for revision.

3) As properly suggested by the reviewer, we merged the previous two algorithms in Fig. 5 and Fig. 11. Particularly, we added the pseudo-code to the algorithms in Fig. 5 and Fig.8. Please see details in Fig.5 and Fig. 8. In addition, two additional paragraphs were added to explain the two algorithms in Section 5 and Section 6, respectively.

4) As properly suggested by the reviewer, we deleted some contents in Section 4.1 and made some revision in Section 4, 5, 6, 7. Please see details in Section 4-7.

II.ADDITIONAL REVISION

1) The subtle changes have been made in Abstract and Keywords.

2) Some changes of references.

Reinforcement Learning of Competitive and Cooperative Skills in Soccer Agents

Jinsong Leng[a,*], Chee Peng Lim[b]

[a]*School of Security and Information Science, Edith Cowan University, 2 Bradford Street, Mount Lawley, WA 6050, Australia*
[b]*School of Electrical and Electronic Engineering, University of Science Malaysia, 14300 Nibong Tebal, Penang, Malaysia*

Abstract

The main aim of this paper is to provide a comprehensive numerical analysis on the efficiency of various Reinforcement Learning (RL) techniques in an agent-based soccer game. The SoccerBots is employed as a simulation testbed to analyze the effectiveness of RL techniques under various scenarios. A hybrid agent teaming framework for investigating agent team architecture, learning abilities, and other specific behaviours is presented. Novel RL algorithms to verify the competitive and cooperative learning abilities of goal-oriented agents for decision-making are developed. In particular, the tile coding (TC) technique, a function approximation approach, is used to prevent the state space from growing exponentially, hence avoiding the curse of dimensionality. The underlying mechanism of eligibility traces is evaluated in terms of on-policy and off-policy procedures, as well as accumulating traces and replacing traces. The results obtained are analyzed, and implications of the results towards agent teaming and learning are discussed.

Keywords: Reinforcement Learning, Temporal Difference Learning, On-policy and off-policy, Eligibility Traces, Performance, Convergence.

1. Introduction

Artificial intelligence (AI) is the science of making machines to perform tasks that would require intelligence if conducted by humans. In this aspect, intelligence may consist of many aspects, such as perceiving, reasoning, planning, learning, and communication. The agent-based systems normally possess much of these characteristics of intelligence, and are classified under the category of distributed AI.

An agent is a hardware and/or software-based computerized system displaying the properties of autonomy, social adeptness, reactivity, and proactivity [29]. Agent-based systems usually operate in real-time, stochastic, and dynamic environments. The ability to act under uncertainty without human or other intervention (autonomy) is the key feature of an intelligent agent. Intelligent agents are required to adapt and learn in uncertain environments via communication and

[*]Corresponding author. Tel.: +61 8 93706332, fax: +61 8 93706100
Email address: `j.leng@ecu.edu.au (Jinsong Leng)`, `Cplim@eng.usm.my` (Chee Peng Lim)

collaboration (in both competitive and cooperative situations). Owing to the inherent complexity, however, it is difficult to verify the properties of the complex and dynamic environments a *priori*. As such, learning without external instruction becomes one of the fundamental abilities of intelligent agents.

Reinforcement learning (RL) has appeared as a popular learning approach that is conducted with minimal supervision. RL maps states to actions and attempts to maximize the long-term rewards during learning. The goal of RL is to compute a value function so as to find the optimal or near optimal action for a given state. The learning environment consists of two elements: the learning agent(s) and the dynamic process. An agent pursues the goals and builds up knowledge to control the process optimally while interacting with the environment. At every time step, the agent perceives the process state via sensors, makes a decision, and takes an action on the environment via actuators.

Most stochastic systems have mathematical roots that can be modeled as Markov Decision Processes (MDPs) [1, 6]. Computer simulation of stochastic game systems is useful to replicate an environment for testing and verifying the efficiency of learning and optimization techniques. For example, learning competitive and/or cooperative behaviors has been widely investigated in computer games such as Soccer [32, 31] and Unreal Tournament (UT) [30]. In this domain, RL has been used to learn both competitive and cooperative skills in the RoboCup simulation system using different kinds of learning algorithms and state space representations [17]. The success of RL critically depends on an effective function approximation, a facility for representing the value function concisely, and the parameter choices used [27].

Temporal difference learning (TD) [21, 26] is a popular computational RL technique. The TD method is a combination of Dynamic Programming technique (DP) [1, 6] and Monte Carlo method (MC) [13]. This technique has been used to deal with prediction problems [21], as well as to solve optimal control problems [26, 19]. The difficulty associated with TD learning is how to solve a temporal credit assignment problem, i.e., to apportion credit and blame to each of the states and actions. Trial-and-error, delayed reward, and trade-off between exploration and exploitation are the most important features in TD techniques. As with many learning techniques, convergence analysis of TD is of paramount importance from both theoretical and practical points of view. Unfortunately, convergence of TD has been proved only under some strict conditions, such as tabular state representation and with a small learning rate [4, 3, 2]. It is unclear whether a TD algorithm would converge with any approximation function. Furthermore, the use of eligibility traces in TD adds another dimension to its convergence.

In this paper, a computer game called SoccerBots [32] is used as a simulation environment for investigating individual and cooperative learning abilities of goal-oriented agents. SoccerBots provides a real-time, dynamic, and uncertain environment with continuous state-action spaces. TD(λ) is adopted to learn competitive and cooperative skills of soccer agents in the SoccerBots environment. The focus of this paper is on the analysis of TD(λ) algorithmic problems in a practical domain. The TD(λ) algorithm could be applied to individual agent learning as well as be extended to multi-agent co-operative learning. An agent architecture is introduced for conducting an empirical study of the TD technique with an approximation function. The objectives of this work are summarized as follows:

1. To perform an investigation of agent-based games with RL. This involves
    - Abstraction of a minimal team for the chosen test-suite (SoccerBots), and identification of a minimal set of agent behaviors - run to the ball, stop the ball from crossing the line, etc.

    - Experiments using TD(λ) with approximation functions to obtain performance metrics.
    - Efficiency analysis of the on-policy and off-policy RL algorithms.
2. To investigate the mechanism of eligibility traces in TD. This involves
    - Investigation of the on-policy and off-policy algorithms, and comparison between accumulating traces (AT) and replacing traces (RT).
    - Analysis of the efficiency of bootstrapping, and recommendations for choosing the 'appropriate' value of λ.

This paper aims to provide a comprehensive numerical analysis on the efficiency of various RL techniques. The major contributions of this paper are outlined as follows. Firstly, a hybrid agent teaming framework is presented for investigating agent team architecture, learning abilities, and other specific behaviors. Secondly, novel RL algorithms to verify the competitive and cooperative learning abilities of goal-oriented agents for decision-making are developed. In addition, the tile coding (TC) technique, a function approximation approach, is used to prevent the state space from growing exponentially, hence avoid the curse of dimensionality. Thirdly, the underlying mechanism of eligibility traces is analyzed in terms of on-policy and off-policy algorithms, accumulating traces and replacing traces.

The paper is organized as follows: Section 2 discusses the related work about RL techniques and agent systems. Section 3 gives an overview of TD(λ) techniques. An agent architecture and simulation environment are described in Section 4. Section 5 details an investigation of TD(λ) performance. Analysis of on-policy and off-policy algorithms are presented in Section 6. Section 7 examines the mechanism of eligibility traces. Finally, conclusions and suggestions for future work are presented in Section 8.

2. Related Work

2.1. RL and Soccer Agents

The major purpose of RoboCup soccer is to provide a simulation environment for studying research problems related to machine learning and intelligence, such as learning algorithms and state space reduction techniques. RL has been used to learn both competitive and cooperative skills in RoboCup simulation systems using different learning algorithms and state space representations [17].

In particular, RL algorithms have been applied to various tasks in RoboCup soccer. Stone et al. proposed a Team-Partitioned, Opaque-Transition RL algorithm called TPOT-RL, which was successfully used to train soccer agents for learning some competitive and cooperative skills [18]. The learning method in [18] was based on a Monte Carlo approach, and a look-up table in conjunction with the state space abstraction technique was used to store the Q-values. In [5, 10], different RL algorithms were used to train different aspects of soccer skills. A ball interception method was proposed using Q-learning and a Grid- and Memory-Based function approximator. Another related work was proposed in [7], in which the shooting goal skill was learned via Sarsa(λ) with tile coding techniques. However, the discussion on the algorithms and the way tile coding used was limited in that paper.

Many approaches use RL with a linear function approximation. In [17], the application of episodic MDP Sarsa(λ) with a tile-coding approximation function to learn higher-level decisions in a keepaway subtask was introduced. The parameters were fixed without a sensitivity analysis of the parameters towards performance. The experimental results were compared based on different exploration strategies (Random, Hold, and Hand-coded).

2.2. *On-policy and Off-policy Approaches*

There are a few comparative studies of eligibility traces with on-policy and off-policy algorithms. The most extensive comparative studies were conducted by Rummery [14]. He compared the convergence properties in both tabular representation and approximation function to solve robot control problems. The following algorithms are considered in that paper [14]: (1). Watkins Q(λ) with the eligibilities zeroed whenever a non-greedy action is performed. (2). Modified Q(λ), i.e., Sarsa(λ). (3). Peng's Q(λ). (4). Standard Q(λ) where the eligibilities are not zeroed, i.e., *naive Q*(λ).

In [14], the experimental results indicated that Modified Q(λ) showed the best performance with the least computational overhead. Peng's Q(λ) ranked second, while Watkins Q(λ) was the last. Another comparative study was made by Singh and Sutton, using accumulating traces and replacing traces [16]. The experiments in [16] provided evidence for two key points: (1). that replace-trace methods can perform much better than conventional, accumulate-trace methods, particularly at long trace lengths; (2). that although long traces may help substantially, the best performance is obtained when the traces are not infinite, that is, when intermediate predictions are used as targets rather than actual sample returns. However, they claimed that more empirical studies are needed with trace mechanisms before a definitive conclusion can be drawn about their relative effectiveness, particularly when function approximators are used.

2.3. *Problems of Temporal Difference Learning*

The major difficulty in TD learning is how to solve a temporal credit assignment problem, i.e., to apportion credit and blame to each of the states and actions. TD(λ) was proposed by Sutton [21] in 1988, but without providing a convergence analysis. TD(λ) for discounted problems converges with unit probability under the standard stochastic convergence constraints on the learning rate and linear state representation such as look-up table or linear function [3, 25, 24]. However, convergence of TD(λ) has only been proved for linear networks and linearly independent sets of input patterns. In a more general case, the algorithm may not converge even to a locally optimal solution, let alone to the globally optimal solution [22].

There are no universal methods available for analyzing convergence, nor for finding the optimal parameter values to improve the learning performance. For example, in [12] an approximate estimator is used for choosing the parameter values to increase the convergence rate in some numerical experiments. However, the approach in [12] was only tested on a limited number of systems and cannot guarantee to improve convergence in different domains.

As with many learning techniques, convergence analysis is of paramount importance from both theoretical and practical points of view. Unfortunately, convergence has been proved only under some strict conditions, as explained earlier. It is unclear whether TD(λ) would converge with any approximation function. Furthermore, the use of eligibility traces leads to an additional problem. In summary, the major challenges of TD(λ) are outlined as follows:

1. Mechanism of eligibility traces: The aim of eligibility traces is to assign credit or blame to the eligible states or actions. From a mechanistic point of view, eligibility traces can be implemented using a memory associated with each state to record the occurrence of an

event, i.e., the visiting of a state or the taking of an action. However, the underlying mechanism of eligibility traces with an approximation function has not been well understood, either from the theoretical or practical point of view.

2. Parametric tuning problem: The TD(λ) algorithm comes with some adjustable parameters including the learning rate α, the discount rate γ, and the decay rate λ. A number of parametric optimization techniques can be used to find the optimal parameter values. However, the performance analysis is very expensive, because the analysis metrics are obtained only by running an extensive set of experiments with different parameter values. There is no standard benchmark on a performance measure from either the theoretical or practical point of view.
3. Convergence and divergence: A rigorous proof is given that TD(0) converges to the optimal predictions for linearly independent feature vectors [21]. Convergence with unit probability for general λ was proved too [3, 23]. Convergence has not been proved with function approximation for $0 \leq \lambda \leq 1$ [20, 28]. Such an in-depth convergence analysis relies on empirical studies and related performance evaluations [3, 2, 12, 17, 20].

So far, there are no standard criteria for evaluating convergence either from theoretical or practical perspectives. The performance metrics are often used to analyze convergence. However, the performance metrics can only be obtained by conducting a number of experiments. In this aspect, a simulation system is helpful to investigate the issues such as convergence analysis, parameter tuning, and analysis of the efficiency of bootstrapping.

3. Temporal Difference Learning

RL is about learning to map situations to actions for maximizing a numerical reward signal [20]. RL is the general name for sample-backup techniques, including Dynamic Programming (DP) and Monte Carlo methods. RL problems can be solved in different forms, for instance, the actor-critic method that is based on the predictive state representation and gradient descent techniques. Basically, RL problems can be characterized in terms of optimal control of MDPs. RL also adopts the same optimization technique under the formalism of MDP.

Although RL is a powerful and effective methodology with theoretical foundations, learning in dynamic and distributed environments is a difficult task owing to large, continuous state-action spaces. Normally, the RL approach comprises a trial-and-error search, delayed reward, and exploration versus exploitation. RL has also to balance the trade-off between exploration and exploitation.

3.1. Temporal Difference Learning Techniques

The prerequisite of DP is that an explicit model of transition probabilities is needed. For most stochastic and dynamic systems, it is infeasible or impractical to obtain such a model in advance. For Monte Carlo methods, it is required that an episode must terminate. The ideas of DP, i.e., policy evaluation, the state value function computation, then policy improvement, can be combined with the Monte Carlo method when an explicit model of transition probabilities is not available. Instead of using a model-based approach, the so-called model-free RL techniques are used to probe the environment, thereby computing the value functions. Temporal Difference Learning (TD) methods bootstrap the state value function as with DP, and update the value function at every step in a fully incremental fashion. TD techniques, on the other hand, overcome the drawback of the Monte Carlo method that state value is updated at the end of each episode.

The TD method updates the state value at every step and learns directly from experience like the Monte Carlo approach (model-free) but employs bootstrapping like DP. In short, TD methods overcome the disadvantages of DP and Monte Carlo methods: it is not necessary to formulate a model of the environment and transition probabilities a *priori*, and not necessary to estimate the state value function until the end of an episode. The Monte Carlo method uses sample backups, which is based on sample experience.

State value function $V(s_0)$ in a state s_0 can be estimated using Equation (1):

$$V(s_0) = V(s_0) + \alpha[r(s_0) + \gamma r(s_1) + \gamma^2 r(s_2) + \cdots - V(s_0)] \tag{1}$$

where α is the learning rate, and γ is the discount rate.

The successive TD error δ_t is as follows:

$$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \tag{2}$$

Then Equation (1) can be written as follows:

$$V(s_0) = V(s_0) + \alpha[\delta_0 + \gamma\delta_1 + \gamma^2\delta_2 + \dots] \tag{3}$$

The successive TD error in a state s at time t is derived from Equation (3):

$$\delta_t = \max_a [r_{t+1} + \gamma V_t(s_{t+1}) - V_t(s_t)] \tag{4}$$

TD techniques have become the most popular approach for solving learning problems. Trial-and-error and discounted rewards are the two most important features in TD learning. The TD algorithms fall into two main classes: (1). On-policy learning – action selection based on the learned policy; (2). Off-policy learning – action selection based on a greedy policy.

Sarsa (initially known as modified Q-learning [14]) is an on-policy approach. The state-action value function $Q(s_t, a_t)$ at the time t is updated as follows:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\delta_t \tag{5}$$

While Q-learning is an off-policy method in that an action selection uses a hypothetical action and is independent of the policy being followed:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \max_a \gamma Q_t(s_{t+1}, a) - Q_t(s_t, a_t)] \tag{6}$$

3.2. Eligibility Traces

Eligibility traces are another basic mechanism for improving the speed of TD learning. An eligibility trace is a temporary log to record the occurrence of an event, for example, the visiting of a state or the taking of an action. From a theoretical viewpoint, eligibility traces build a bridge from TD to Monte Carlo methods [20].

The TD(λ) algorithm can be understood as one particular way of averaging-step backups. This average contains all the step backups, each weighted in a proportional manner to $0 \leq \lambda \leq$. A normalization factor of λ ensures that the weights sum to 1 [20]. From the forward point of view, the resulting backup is derived from Equation (7), called the λ-return, as follows:

$$V_\pi^\lambda(s, t) = (1 - \lambda)E_\pi\{\sum_{k=0}^{\infty}(\gamma\lambda)^k r(s_{t+k+1})\big|s_t = s\} \tag{7}$$

The main advantage of using λ is to decay the reward according to the geometrical distribution in every episode, which can enhance convergence properties.

The related state value function in Equation (3) can be defined by:

$$V(s_0) = V(s_0) + \alpha[\delta_0 + \gamma\lambda\delta_1 + (\gamma\lambda)^2\delta_2 + \dots] \tag{8}$$

TD(λ) is difficult to implement from forward point of view. Instead, TD(λ) can be implemented from backward point of view. The purpose of eligibility traces is to assign the credit or blame to the eligible states or actions. Traces ($e_t(s, a)$) can be accumulated (accumulating traces) by ($e_t(s, a) = \gamma\lambda e_{t-1}(s, a) + 1$) or replaced by 1 (replacing traces).

Accumulating traces can be defined as:

$$e_t(s, a) = \begin{cases} \gamma\lambda e_{t-1}(s, a), & \text{if } s \neq s_t \\ \gamma\lambda e_{t-1}(s, a) + 1, & \text{if } s = s_t \end{cases} \tag{9}$$

whereas replacing traces use $e_t(s, a) = 1$ for the second update.

The successive TD(λ) error in a state s at time t is derived from Equation (4):

$$\delta_t = \max_a [r_{t+1} + \gamma V_t(s_{t+1}) - V_t(s_t)]e_t(s, a) \tag{10}$$

3.3. Generalization and Approximation Function

The curse of dimensionality [1] is an important practical issue; the performance depends on the abstract or approximate method. Without abstraction or approximation techniques, RL does not scale up well for real-time, dynamic systems. As mentioned above, convergence theorems of DP and TD(λ) are investigated under the assumption that the state space is with a look-up table representation or a linear approximation function. However, most RL problems have large, dynamic and continuous state spaces. It is important to find some techniques, which can reduce the complexity and, meanwhile, ensure acceptable performance during learning. State abstraction techniques have been used for RL problems [15, 24]. Function approximation techniques are the most popular ways to represent the state space using fewer parameters than classification features.

Tile coding is one of most popular function approximation methods for RL problems. The tilings are overlapped over the state space. Each tiling has only one cell that can be activated. The denser the tiling, the finer and more accurately the desired function can be approximated, but the greater the computational overhead [20]. The linear approximation function with tile coding is illustrated in Fig. 1.

The approximate function, V_t is represented as a parameterized function, thereby updating parameters instead of entries in a table. This is represented as:

$$V_t = \overrightarrow{\theta}_t^T \overrightarrow{\phi}_s = \sum_{i=1}^{n} \theta_t(i)\phi_s(i) \tag{11}$$

In the linear function, ($\overrightarrow{\theta}_t$) is the parameter vector, and ($\overrightarrow{\phi}_s$) is a corresponding column vector of features for each state. The complexity is related to the size of feature θ rather than the size of state space.

The tile coding method splits the state space into tilings. In tile coding, the representation is determined by the shape of tile, and the resolution of final approximation is determined by the

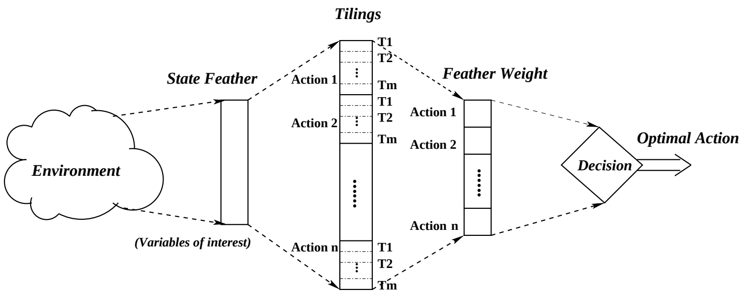


Figure 1: The Linear Approximation Function with Tile Coding

number of tilings [20]. The tilings partition the state space into cells. Each tiling has only one tile that can be activated for every input state, and the active cells of a state are called features. The receptive fields of the features are grouped into partitions (also called tiling). The value of a state is calculated as ($\sum_{i=1}^{n} \theta(i)\phi(i)$). The performance may be sensitive to the specifics of partition and the number of tilings.

4. Agent Architecture and Simulation Environment

4.1. Simulation System

SoccerBots is part of TeamBots(tm) software package [32], which is developed in Java. Fig. 2 shows the SoccerBots simulation system.

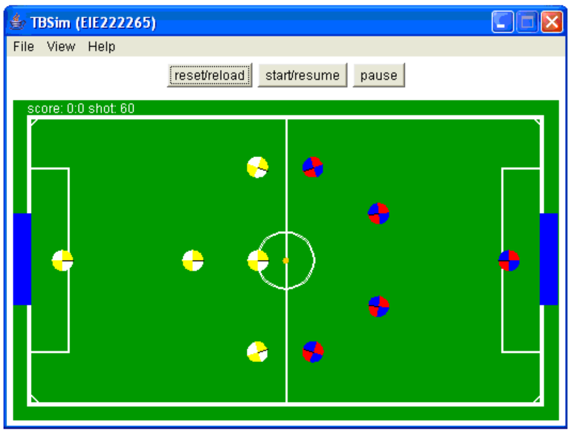


Figure 2: The SoccerBots Simulation System

SoccerBots consists of three core components:

1. The description file — the TBSim simulator reads in a description file to know what sorts of objects are in the environment and their current status.
2. The simulation kernel — which runs the simulator at each step by drawing the objects. Each object in the simulation includes two components: a drawing method and a dynamic

simulation method. The vision scope of the soccer players and the noise in the system can be defined.

3. Robot control system — which controls the soccer players. This system can be modified by adding some soccer control and learning strategies. This is the key component for evaluating TD(λ) algorithms.

The soccer game is a real-time, noisy, adversarial domain, which has been regarded as a suitable environment for multi-agent system simulations [18].

The competitive skills, scoring goal and intercepting ball, are illustrated in Fig. 3.

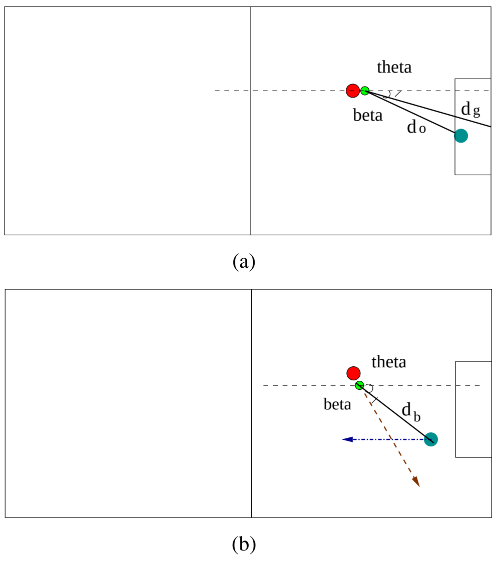


Figure 3: (a) Shooting. (b) Intercepting.

The shooting and intercepting behaviors are the most basic individual skills in the soccer game, as shown in Fig 3. The shooting problem is to find the optimal kicking direction toward the goal. The intercepting problem is to compute a direction in order to intercept an approaching ball in the shortest possible time.

4.2. Agent Architecture

In order to evaluate the most realistic performance of RL, the small-sized soccer league SoccerBots, which is a collection of application of TeamBots [32], is used. Each soccer team can have 2 to 5 players (see Fig 4). Each player can observe the behaviors of other objects such as the ball, a teammate, an opponent, and their locations and motions via a sensor. The soccer game players have to interact, coordinate, and cooperate with each other. The ball's direction is influenced by environment noise (at random). In the real-time environment, performance is a high priority to be considered to follow the changes of motion of the ball and the players. The action that is sent back to each player is the deliberative outcome of the whole team.

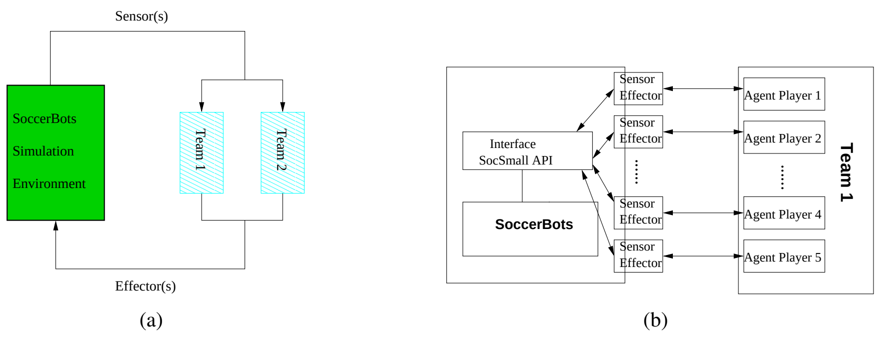


Figure 4: The SoccerBots and Agents Team. (a) SoccerBots. (b) Agent Team Architecture

5. Learning Competitive Skills

In this study, we extend our previous work [8, 9]. The detailed Sarsa (λ) control algorithm with replacing traces and accumulating traces in Fig. 5, where $\vec{e}$ and $\vec{\theta}$ are matrices to represent trace feature and related feature weights, and $\mathcal{F}$ is the feature set to trace the eligible state process. The ϵ-greedy policy is the exploration strategy, i.e., the agent takes a random action with probability ϵ and takes best action learned with probability (1 - ϵ).

In Fig. 5, the Sarsa(λ) Control Algorithm is divided three stages for every episode. At the first stage, it is in the initial state s_0 of an episode, the state-action value function and the feature set $\mathcal{F}$ are updated. For each internal step of an episode, the action is chosen based on ϵ-greedy policy. The state-action value function is computed in line 7, and the related trace feature from $\mathcal{F}$ is obtained in line 8. Accordingly, the temporal difference δ is calculated, as shown in line 9. The feature weights $\vec{\theta}$ and the trace feature $\vec{e}$ are updated in line 10. For the action following the policy, the related feature is updated based on either accumulating traces or replacing traces, as shown in line 11. If the state is final state of an episode, the related feature weights $\vec{\theta}$ is updated, as shown in line 13.

A four-dimensional tiling approach with the continuous variables that divides the space into $8 \times 8 \times 10 \times 10$ tiles is used. All tilings are offset at random variables.

5.1. Learning for Scoring Goal

The scenario to learn shooting behavior for a player is given in Fig. 3 (a). An attacker with a ball is placed in front of goal, and a goalie is at the goalmouth moving north or south along the vertical axis. The shooting behavior is influenced by the position of the ball and the goalie. In this case, four parameters are taken into account:

1. The distance from the ball to the goal.
2. The distance from the ball to the goalie.
3. The angle between the ball to the goal.
4. The angle between the ball to the goalie.

The reward function is defined as follows:

$$Reward(s) = \begin{cases} 100, & \text{if the ball goes into the goal;} \\ -1, & \text{for each step;} \\ 0, & \text{if the ball misses the goal.} \end{cases} \tag{12}$$

```
StartEpisode ($s_0$):
1.   Initialize $\vec{e} = 0$. /* Initialize the trace feature */
2.   Get action $a_{LastAction}$ from $s_0$ using $\epsilon$-greedy policy. /* Find the action based on policy */
3.   Calculate $Q_{LastAction}$ using $s_0$, $a_{LastAction}$. /* Calculate the sate-action value */
4.   For all i $\in \mathcal{F}(s_0, a_{LastAction})$ /* Update trace feature using the feature set */
         $e_i \leftarrow e_i + 1$ (For accumulating traces)
         $e_i \leftarrow 1$ (For replacing traces)
ExecuteStep(s):
5.   $\delta$ = reward - $Q_{LastAction}$. /* Calculate the difference between the reward and state-action value*/
6.   Get action $a_{NewAction}$ from s using $\epsilon$-greedy. /* Find the action based on policy */
7.   Calculate $Q_{NewAction}$ using s, $a_{NewAction}$. /* Compute the state-action value */
8.   Get $\mathcal{F}(s, a_{NewAction})$ using s and $a_{NewAction}$. /* Find the related feature set */
9.   $\delta \leftarrow \delta + \gamma * Q_{NewAction}$. /* Calculate the temporal difference */
10.  Update all $\vec{\theta}$ and $\vec{e}$
         $\vec{\theta} \leftarrow \vec{\theta} + \alpha * \delta * \vec{e}$. /* Update the feature weights */
         $\vec{e} \leftarrow \gamma * \lambda * \vec{e}$. /* Update the trace feature */
11.  For all i $\in \mathcal{F}(s, a_{NewAction})$ /* Update the trace feature based on feature set */
         $e_i \leftarrow 1 + \gamma * \lambda * e_i$ (For accumulating traces).
         $e_i \leftarrow 1$ (For replacing traces)
StopEpisode (s'):
12.  $\delta$ = reward - $Q_{LastAction}$. /* Calculate the difference between the reward and state-action value*/
13.  $\vec{\theta} \leftarrow \vec{\theta} + \alpha * \delta * \vec{e}$. /* Update the feature weights */
14.  End episode. /* The end of an episode */.
```

Figure 5: Sarsa(λ) Control Algorithm with Replace Traces and Accumulating Traces

The kicking direction is defined as a set of angles in degrees:

$$\{ 27, 24, \cdots, 3, 0, -3, \cdots, 24, 27 \}.$$

Fig. 6 indicates that the speed of convergence is quicker for bigger learning rate α, but smoother for smaller learning rate α.

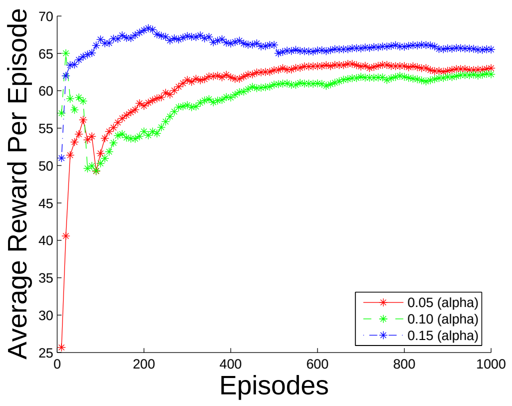


Figure 6: The Diagram of Reward and Episodes α: 0.05, 0.10, 0.15

5.2. *Learning Ball Interception*

The scenario to learn skill for the player is given in Fig. 3 (b). The ball and the player are placed at fixed points at the beginning of each episode, so as to verify and investigate the performance and convergence easily. By running each episode, the player learns how to take action in the given state. For ball interception, the ball is kicked at a certain angle and speed. The player is away from the ball at a certain distance and angle to ball.

Four parameters are considered:

1. The distance from the ball to the player.
2. The angle from the ball to the player.
3. The velocity of the ball.
4. The angle of the ball.

The first two parameters are obtained directly from the system, and the last two parameters are obtained by calculating the ball's positions at certain time period.

The reward function is defined as follows:

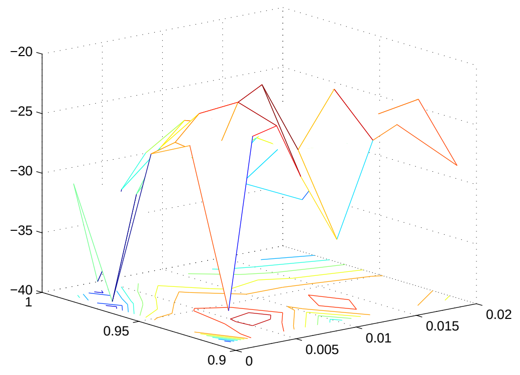
$$Reward(s) = \begin{cases} 0, & \text{if the ball is intercepted;} \\ \text{-0.1}, & \text{for each step.} \end{cases}$$

The relationship between parameters in the algorithm is analyzed. The values of α and γ are tuned while fixing the values of the other parameters. The λ is not considered because the performance is relatively independent of it. The data set as illustrated in Table 1 is obtained:

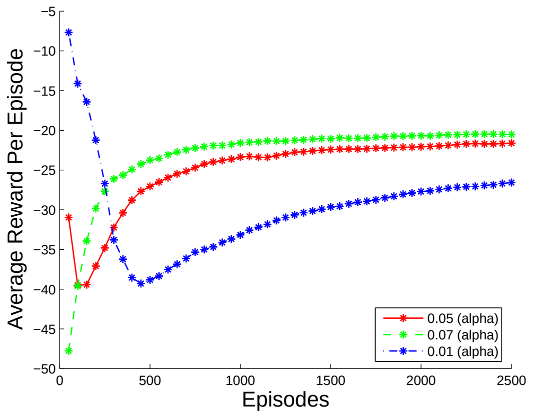
The results between α and γ are shown in Fig. 7 (a). The diagram illustrates the influence of α ($\gamma = 0.93$, $\lambda = 0.9$ and $\epsilon = 0.1$) in Fig. 7 (b) .

Table 1: The α, γ and Average Rewards

α, γ	0.001	0.003	0.005	0.007	0.01	0.013	0.015	0.02
0.99	-30.58	-39.18	-31.84	-29.17	-29.47	-32.48	-33.25	-35.64
0.97	-39.5	-30.89	-27.36	-25.47	-29.06	-28.11	-29.24	-30.49
0.95	-26.14	-25.57	-23.54	-24.82	-24.46	-24.95	-25.01	-26.15
0.93	-24.48	-25.98	-21.62	-20.53	-26.58	-22.09	-25.17	-24.3
0.91	-37.37	-23.11	-22.63	-27.27	-33.14	-25.4	-24.48	-28.89



(a)



(b)

Figure 7: (a) The Relationship between α and γ. (b) α at 0.005, 0.007, 0.01

The optimal values of α, γ are 0.005 – 0.007, 0.92 — 0.94 respectively, as shown in Fig. 7 (a). The value of α is adjusted by setting $\epsilon = 0.1$, $\gamma = 0.93$, and $\lambda = 0.9$. Fig. 7 (b) indicates that the stability and speed of convergence are improved when α is 0.07.

6. Analysis on On-policy and Off-policy Algorithms

6.1. Details of On-policy and Off-policy Algorithms

TD learning techniques can be categorized as on-policy learning (Sarsa) and off-policy learning (Q-learning). The major difference between them is the method for updating the state function. The eligibility traces are considered as a mechanism to remember the events during a training episode. The state values are updated at every step, and weighted by the traces-decay parameter λ. For on-policy Sarsa(λ) learning in Fig 5, the traces are recorded according to the action selection policy. For off-policy Q(λ) learning, however, there are several ways to utilize the traces [20]. Since a non-greedy action selection does not follow the greedy policy being followed, the main difference is in what kind of strategies to be taken when an exploratory action is taken. Sutton [20] described the major Q(λ) learning algorithms include Watkins's Q(λ) [26], Peng's Q(λ) [11] and *naive* Q(λ).

Watkins's Q(λ) records the traces whenever a greedy action is taken; otherwise stops the traces by setting the trace to 0. Suppose that the action a^* ($Q_t(a^*) = max_a Q_t(a)$) is the greedy action in the state s_t, the accumulating traces are updated as follows:

$$e_t(s,a) = \begin{cases} \gamma\lambda e_{t-1}(s,a) + 1, & \text{if } s = s_t, a^* = a_t \\ 0, & \text{if } s = s_t, a^* \neq a_t \\ \gamma\lambda e_{t-1}(s,a), & \text{otherwise} \end{cases} \tag{13}$$

whereas replacing traces use $e_t(s,a) = 1$ for the first update.

Another Q-learning algorithm with another traces strategy was proposed by Peng [11], in which all state values are updated like Watkins'Q(λ) except that the traces are not terminated for exploratory actions, and the existing state value is further adjusted by calculating an additional difference of this step. Peng's Q (λ) learning algorithm follows neither on-policy nor off-policy [11]. The drawback of Peng's Q (λ) learning algorithm is that it is difficult to implement, and it does not guarantee convergence under some circumstances [20].

This paper presents three learning algorithms (Sarsa(λ), Watkins'Q(λ) and *naive* Q(λ)) with different eligibility traces, so as to analyze the effects of these traces strategies. A linear approximation function known as Tile Coding is used to represent the state space. The trade-off of exploration and exploitation is balanced. The action selection policy called ϵ-greedy policy is adopted, i.e., the agent takes a random action with probability ϵ to explore the unknown states and takes the best action following the learned knowledge with probability (1 - ϵ). For simplicity, the traces strategy modified from Watkins'Q(λ) called *naive* Q (λ) [20], in which the traces is not terminated for exploratory actions, are considered. The Watkins'Q(λ) and *naive* Q(λ)) are detailed in Fig. 8.

The major differences between Sarsa(λ) in Fig. 5 and Watkins'Q(λ) (*naive* Q(λ)) are twofold: The action to be taken at state s is based greedy policy, as shown in line 7 in Fig. 8; The feature weights $\vec{\theta}$ and the trace feature $\vec{e}$ in line 12 are different from Sarsa(λ).

```
StartEpisode ($s_0$):
1.  Initialize $\vec{e} = 0$. /* Initialize the trace feature */

2.  Get action $a_{LastAction}$ from $s_0$ using $\epsilon$-greedy policy. /* Find the action based on policy */

3.  Calculate $Q_{LastAction}$ using $s_0$, $a_{LastAction}$. /* Calculate the sate-action value */

4.  For all i $\in \mathcal{F}(s_0, a_{LastAction})$ /* Update trace feature using the feature set */
        $e_i \leftarrow e_i + 1$. (For accumulating traces)
        $e_i \leftarrow 1$. (For replacing traces)
ExecuteStep(s):
5.  $\delta$ = reward - $Q_{LastAction}$. /* Calculate the difference between the reward and state-action value*/
6.  Get action $a_{NewAction}$ from s using $\epsilon$-greedy. /* Find the action based on policy */
7.  Get action $a_{GreedyAction}$ from s. /* Find the best action based on greedy policy */
8.  Calculate $Q_{NewAction}$ using s, $a_{NewAction}$. /* Compute the state-action value using new action */
9.  Calculate $Q_{GreegyAction}$ using s, $a_{GreedyAction}$. /* Compute the state-action value using greedy action */
10.   Get $\mathcal{F}(s, a_{NewAction})$ using s and $a_{NewAction}$. /* Find the feature set using new action */
11.   $\delta \leftarrow \delta + \gamma * Q_{GreedyAction}$. /* Calculate the temporal difference using greedy action*/
12.   (For Watkins'Q($\lambda$))
        Update all $\vec{\theta}$ and $\vec{e}$.
          $\vec{\theta} \leftarrow \vec{\theta} + \alpha * \delta * \vec{e}$. /* Update the feature weights */
          if $a_{NewAction} = a_{GreedyAction}$)
            $\vec{e} \leftarrow \gamma * \lambda * \vec{e}$. /* Update the trace feature */
          else
            $\vec{e} \leftarrow 0$. /* Set the trace feature to 0*/
        ( or for naive'Q($\lambda$))
        Update all $\vec{\theta}$ and $\vec{e}$.
          $\vec{\theta} \leftarrow \vec{\theta} + \alpha * \delta * \vec{e}$. /* Update the feature weights */
          $\vec{e} \leftarrow \gamma * \lambda * \vec{e}$. /* Update the trace feature */
13.   For all i $\in \mathcal{F}(s, a_{NewAction})$
          $e_i \leftarrow e_i + 1$. (For accumulating traces)
          $e_i \leftarrow 1$. (For replacing traces)
StopEpisode (s'):
14.   $\delta$ = reward - $Q_{LastAction}$. /* Calculate the difference between the reward and state-action value*/

15.   $\vec{\theta} \leftarrow \vec{\theta} + \alpha * \delta * \vec{e}$. /* Update the feature weights */
16.   End episode. /* The end of an episode */
```

Figure 8: Watkins'Q(λ) (*naive* Q(λ)) Control Algorithm

6.2. Performance Comparison

The values of α, γ for Sarsa(λ), Watkins's Q(λ) and *naive* $Q(\lambda)$ are 0.007 and 0.93, 0.01 and 0.93, 0.005 and 0.93 respectively. The convergence of three algorithms is drawn in Fig. 9.

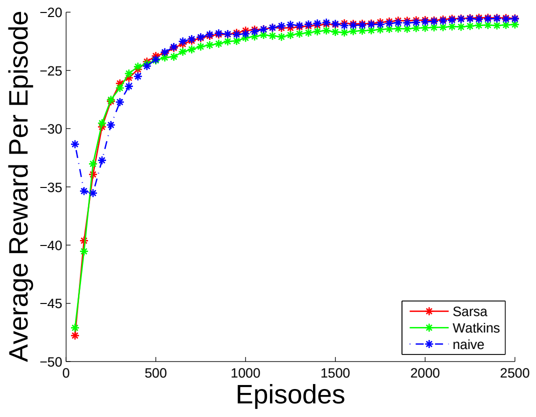


Figure 9: Comparison of Three Algorithms

In Fig. 9, the speed of convergence to optimality is compared. The empirical results indicate that all algorithms work well and eventually converge to optimality. By analyzing the convergence diagram, the following differences among Sarsa(λ), Watkins's Q(λ) and *naive* $Q(\lambda)$ are noted:

- Sarsa(λ) and *naive* $Q(\lambda)$ have almost the same performance both in speed of convergence and in eventual convergence reward values.
- Sarsa(λ) performs almost the same as *naive* $Q(\lambda)$, especially after 1000 episodes.
- The empirical results have also shown *naive* $Q(\lambda)$ performs better than Watkins's Q(λ).

Sarsa(λ) performs more smoothly than Watkins's Q(λ) and *naive* $Q(\lambda)$ because it selects the action using the fixed ϵ-greedy policy. The different eligibility traces strategies are the reason that *naive* $Q(\lambda)$ performs better than Watkins's Q(λ). As argued by Sutton [20], cutting off traces every time an exploratory action is taken loses much of the advantage of using eligibility. In addition, Rummery and Peng [11, 14] argued that zeroing the effect of subsequent reward prior to a non-greedy action is likely to be more of a hindrance than a help in converging to optimal policies since $max_a Q(s, a)$ may not provide the best estimate of value of states.

7. Analysis on Eligibility Traces

7.1. Details of Sarsa(λ) Algorithms with Different Eligibility Traces Strategies

The aim of eligibility traces is to assign credit or blame to the eligible states or actions. From a mechanistic point of view, eligibility traces can be implemented using a memory associated with each state $e_t(s)$ to record the occurrence of an event, i.e., the visiting of a state or the taking of an action. There are several ways for evaluating the traces, particularly accumulating traces and replacing traces. In a conventional accumulating trace, the trace augments each time the state is entered. In a replacing trace, on the other hand, each time the state is visited, its trace is reset to 1 regardless of the value of the prior trace [16].

Accumulating traces can be defined as:

$$e_t(s,a) = \begin{cases} \gamma\lambda e_{t-1}(s,a), & \text{if } s \neq s_t \\ \gamma\lambda e_{t-1}(s,a) + 1, & \text{if } s = s_t \end{cases} \tag{14}$$

Replacing traces use $e_t(s,a) = 1$ for the second update in (14), such that:

$$e_t(s,a) = \begin{cases} \gamma\lambda e_{t-1}(s,a), & \text{if } s \neq s_t \\ 1, & \text{if } s = s_t \end{cases} \tag{15}$$

If λ is 0,

$$e_t(s,a) = \begin{cases} 0, & \text{if } s \neq s_t \\ 1, & \text{if } s = s_t \end{cases} \tag{16}$$

The mechanism of eligibility traces is illustrated in Fig 10.

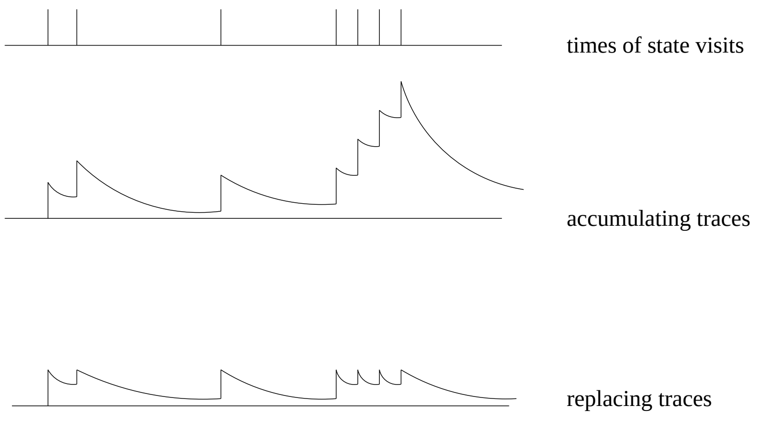


Figure 10: Accumulating and Replacing Traces (taken from [20])

Importantly, no theoretical analysis is available on how the eligibility traces affect the rate of convergence, and on what kind of traces are the best in large, non-deterministic, and dynamic environments.

Sarsa(λ) is an on-policy control algorithm, and the increment is updated synchronously. The Sarsa(λ) algorithm with different eligibility traces is detailed in Fig. 5. The ϵ-greedy policy is the exploration strategy, i.e. the agent takes a random action with probability ϵ and takes the current best action with probability (1 - ϵ).

7.2. Comparison and Discussion

The optimal values of α, γ in Sarsa(λ) for non-traces, replacing traces, and accumulating traces are 0.005 and 0.93, 0.007 and 0.93, 0.007, and 0.91, respectively. Convergence (the speed of convergence and final convergence to optimality) of three cases, as shown in Fig. 11, is compared.

By analyzing the convergence diagram in Fig. 11, the following findings are observed:

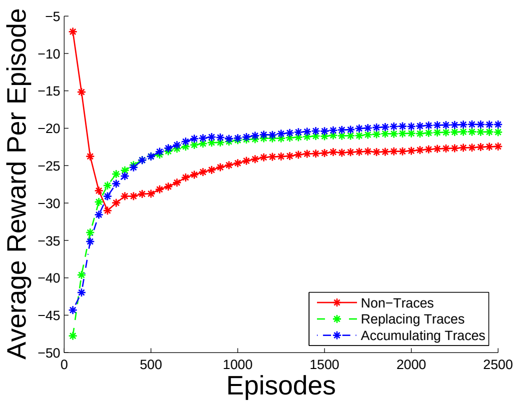


Figure 11: Comparison of Three Algorithms

- Sarsa(λ) can converge in all cases.
- The use of eligibility traces can cause significant improvement in the rate of convergence.
- The empirical results have also shown the performance with accumulating traces performs better than that with replacing traces.

The experimental results are different from those of Singh and Sutton [16, 20]. Singh and Sutton provide evidence that replacing-trace methods can perform much better than conventional, accumulating-trace methods, particularly at long trace lengths [16, 20]. The reason is that their case study was based on an MDP, which meant that the same state is re-visited frequently, and this drives the accumulating traces to be greater than 1. So, the accumulating traces are likely to be more of a hindrance than a help in finding the best estimate of value. However, in this work, the test environment is modeled as a SMDP, meaning that the system does not jump back to the same state. So the use of replacing traces or accumulating traces depends on the stochastic mature of the application domains.

8. Conclusions and Future Work

This paper demonstrates the use of RL in learning competitive and cooperative skills in SoccerBots agents. This research addresses the agent teaming architecture and some algorithmic problems of approximate TD(λ). The empirical results have validated some findings related to RL, i.e.,:

1. A novel RL algorithm is implemented and integrated with a stochastic and dynamic environment, so as to investigate agent teaming architecture, learning abilities, and other specific behaviors. The SoccerBots game is utilized as the simulation environment to verify goal-oriented agents' competitive and cooperative learning abilities for decision making. To deal with a large and continuous state space, the approximation function technique known as tile coding is applied to avoid the state space from growing exponentially with the number of dimensions.

2. The on-policy and off-policy algorithms with different strategies of eligibility traces have been investigated. In addition, the mechanism of utilizing eligibility traces are analyzed and investigated. The efficiency of accumulating traces and replacing traces has also been examined. The major findings of this paper include that the use of accumulating traces and replacing traces depends on the nature of stochastic processes, and without cutting off traces for off-policy algorithms can significantly improve the performance.

The analysis on convergence of TD(λ) and sensitivity between parameters is of paramount importance. This paper investigates the learning of individual agent skills and the related performance metrics are obtained via the process of learning competitive and cooperative skills. The future work includes the issues on agent architecture for agent teaming and cooperative learning, and adaptive selection of parameter values.

References

[1] R. Bellman. A Markovian Decision Process. *Journal of Mathematics and Mechanics*, 6, 1957.

[2] J. A. Boyan. Technical Update: Least-Squares Temporal Difference Learning. *Machine Learning*, 49(2-3):233–246, 2002.

[3] Peter Dayan and Terrence J. Sejnowski. TD(λ) Converges with Probability 1. *Machine Learning*, 14(1):295–301, 1994.

[4] Peter Dayan. The Convergence of TD(λ) for General λ. *Machine Learning*, 8:341–362, 1992.

[5] Thomas Gabel and Martin A. Riedmiller. Learning a Partial Behavior for a Competitive Robotic Soccer Agent. *KI*, 20(2):18–23, 2006.

[6] R. A. Howard. *Dynamic Programming and Markov Processes*. MIT Press, Cambridge, 1960.

[7] A. Kleiner, M. Dietl, and B. Nebel. Towards a Life-Long Learning Soccer Agent. In *Proc. Int. RoboCup Symposium 02*, pages 119–127, Fukuoka, Japan, 2002.

[8] Jinsong Leng, Colin Fyfe, and Lakhmi Jain. Teamwork and Simulation in Hybrid Cognitive Architecture. In *Proceeding in 10th Knowledge-Based Intelligent Information and Engineering Systems*, volume 4252 of *LNCS*, pages 472–478. Springer-Verlag Berlin Heidelberg, 2006.

[9] Jinsong Leng, Colin Fyfe, and Lakhmi Jain. Reinforcement Learning of Competitive Skills with Soccer Agents. In *Proceeding in 11th Knowledge-Based Intelligent Information and Engineering Systems*, volume 4692 of *LNAI*, pages 572–579. Springer-Verlag Berlin Heidelberg.

[10] A. Merke and M. Riedmiller. Karlsruhe Brainstormers – A Reinforcement Learning Approach to Robotic Soccer. In *RoboCup 2001*, volume 2377 of *LNAI*, pages 435–440. Springer Berlin / Heidelberg, 2002.

[11] J. Peng and R. Williams. Incremental Multi-Step Q-learning. *Machine Learning*, 22:283–290, 1996.

[12] A. Potapov and M. K. Ali. Convergence of Reinforcement Learning Algorithms and Acceleration of Learning. *Physical Review E*, 67, Issue 2, 2003.

[13] Reuven Y. Rubinstein. *Simulation and the Monte Carlo method*. New York : Wiley, 1981.

[14] G. A. Rummery. *Problem Solving with Reinforcement Learning*. PhD thesis, Cambridge University, 1995.

[15] S. P. Singh, T. Jaakkola, and M. I. Jordan. Reinforcement Learning with Soft State Aggregation. In *Advances in Neural Information Processing Systems: Proceedings of the 1994 Conference*, pages 359–368. Cambridge, MA. MIT Press, 1995.

[16] S. P. Singh and R. S. Sutton. Reinforcement Learning with Replacing Eligibility Traces. *Machine Learning*, 22(1–3):123–158, 1996.

[17] Peter Stone, Richard S. Sutton, and Gregory Kuhlmann. Reinforcement Learning for RoboCup-Soccer Keepaway. *Adaptive Behavior*, 13(3):165–188, 2005.

[18] Peter Stone and Manuela Veloso. Team-Partitioned, Opaque-Transition Reinforcement Learning. In *RoboCup-98: Robot Soccer World Cup II*, volume 1604 of *LNCS*, pages 261–272, Berlin, 1999. Springer Verlag.

[19] R.S. Sutton, A.G. Barto, and R.J Williams. Reinforcement Learning is Direct Adaptive Optimal Control. *Control Systems Magazine, IEEE*, 12:19–22, 1992.

[20] R.S. Sutton and A.G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[21] R. S. Sutton. Learning to Predict by the Method of Temporal Differences. *Machine Learning*, 3:9–44, 1988.

[22] G. Tesauro. Practical Issues in Temporal Difference Learning. *Machine Learning*, 8(3-4):257–277, 1992.

[23] John N. Tsitsiklis. Asynchronous Stochastic Approximation and Q-learning. *Machine Learning*, 16(3):185–202, 1994.

[24] J. N. Tsitsiklis and B. Van Roy. An Analysis of Temporal-Difference Learning with Function Approximation. *IEEE Transactions on Automatic Control*, 42(5):674–690, 1997.

[25] J. N. Tsitsiklis. Asynchronous Stochastic Approximation and Q-learning. *Machine Learning*, 16(1):185–202, 1994.

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

[27] Shimon Whiteson and Peter Stone. Evolutionary Function Approximation for Reinforcement Learning. *Journal of Machine Learning Research*, 7:877–917, 2006.

[28] Marco A. Wiering. Convergence and Divergence in Standard and Averaging Reinforcement Learning. In *Proceedings of the 15th European Conference on Machine Learning (ECML'04)*, volume 3201 of *LNCS*, pages 477–488. Springer-Verlag Berlin Heidelberg, 2004.

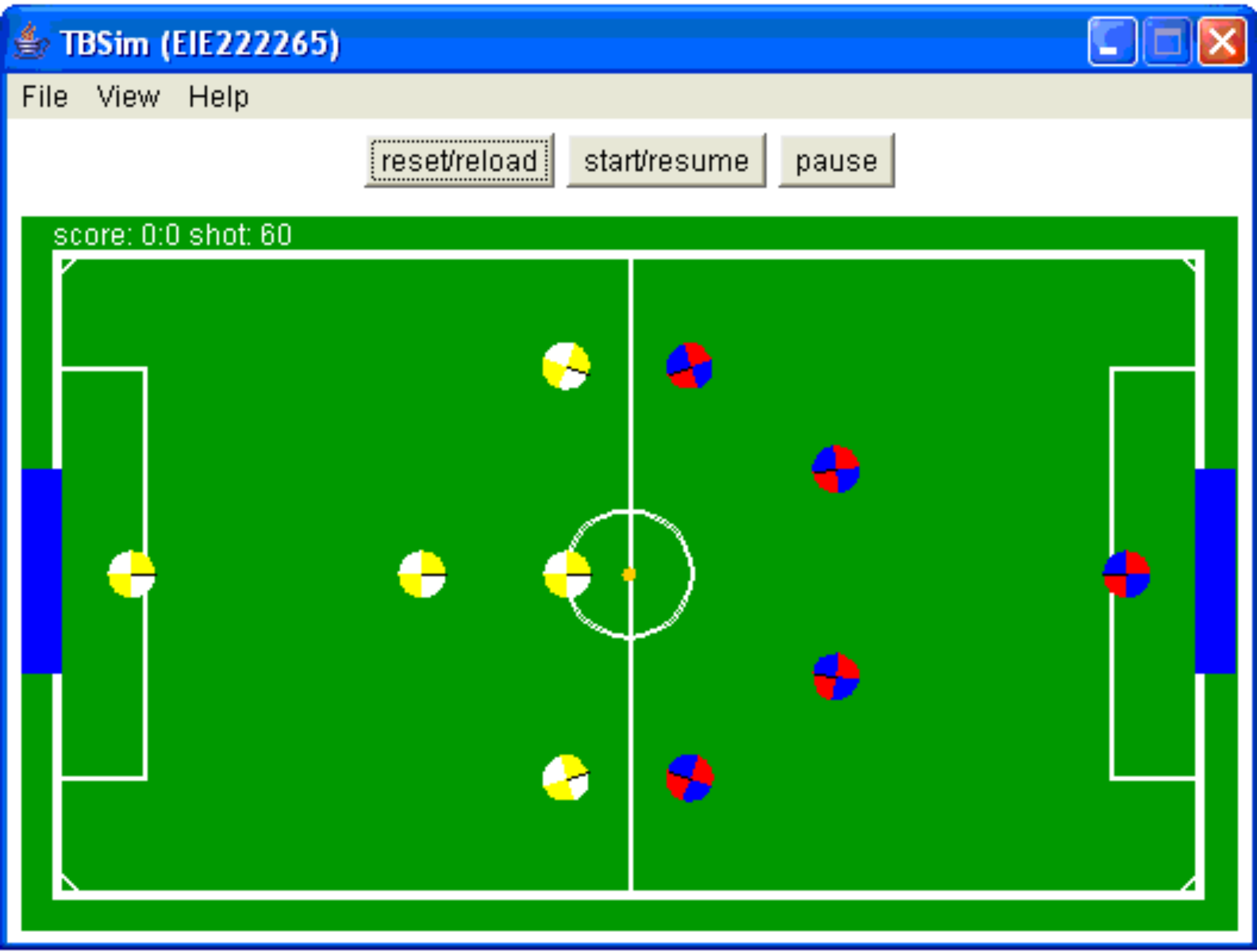
[29] Michael Wooldridge and Nick Jennings. Intelligent Agents: Theory and Practice. *Knowledge Engineering Review*, 10(2):115–152, 1995.

[30] InfoGrames Epic Games and Digital Entertainment. Technical report, Unreal Tournament Manual, 2000.

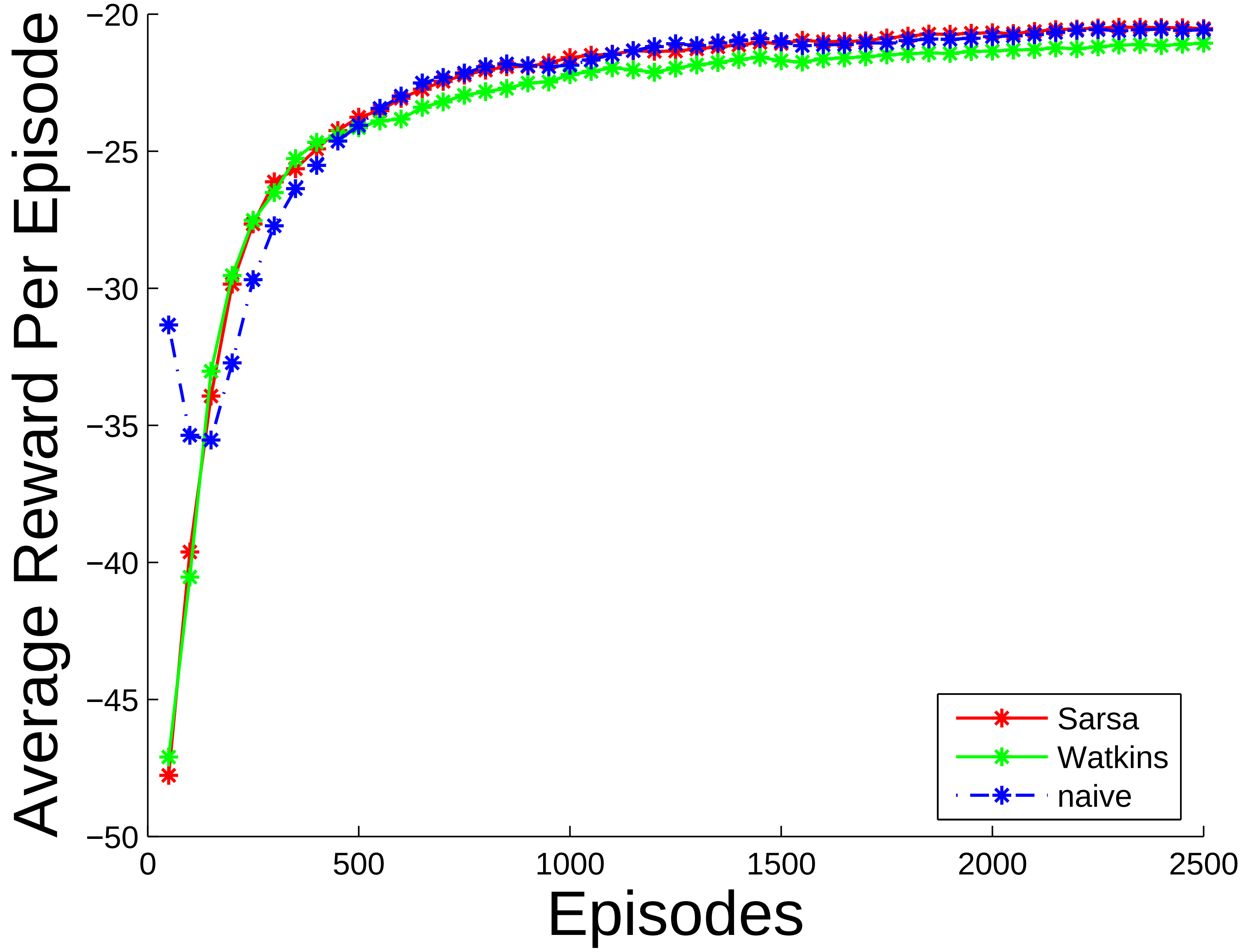
[31] Humanoid Kid and Medium Size League, Rules and Setup for Osaka 2005. Technical report, Robocup, 2005.

[32] *TeamBots^{TM} Domain: SoccerBots*.
`http://www-2.cs.cmu.edu/~trb/TeamBots/Domains/SoccerBots/`.

Figure

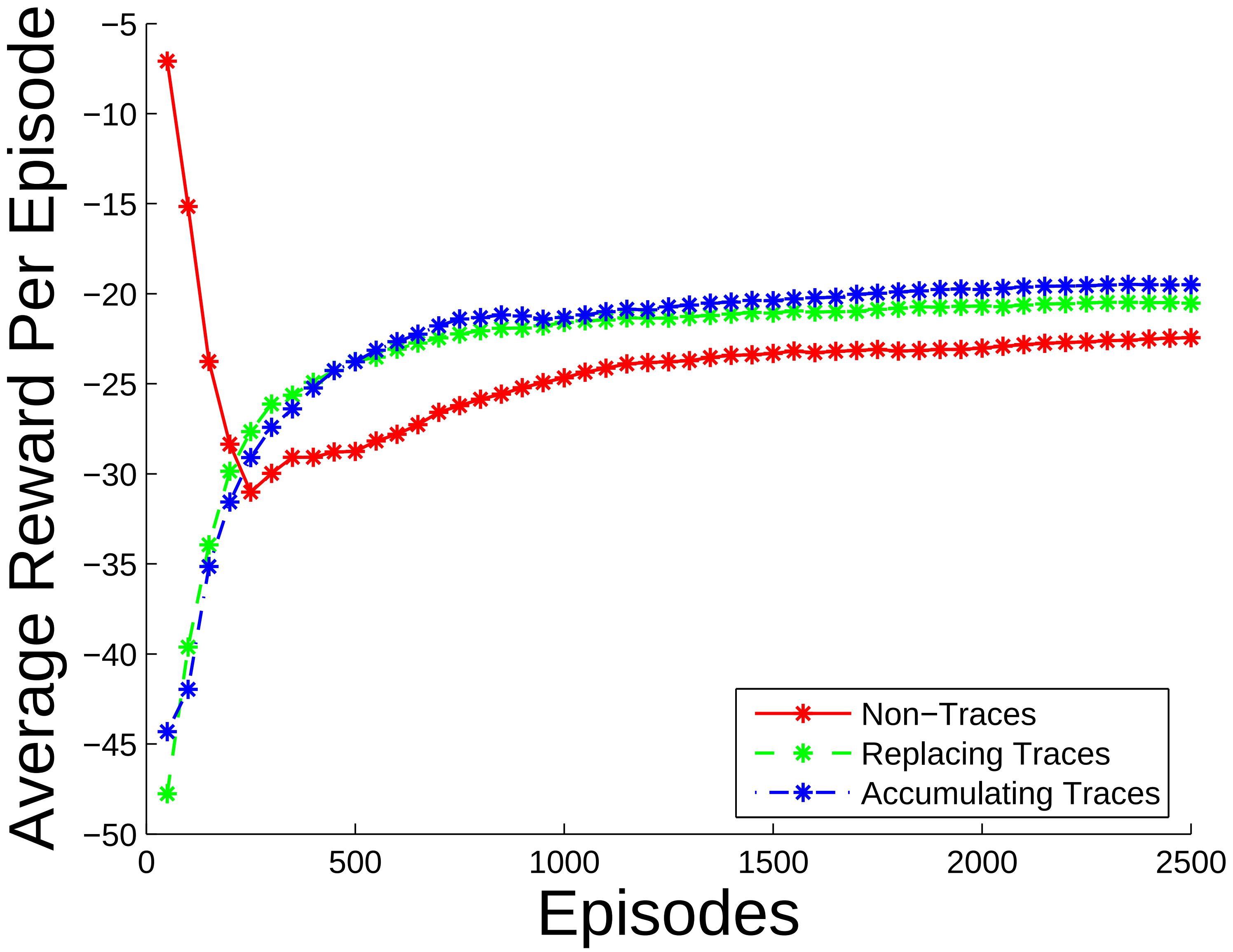

TBSim (EIE222265)
File
View
Help
reset/reload
start/resume
pause
score: 0:0 shot: 60

Figure



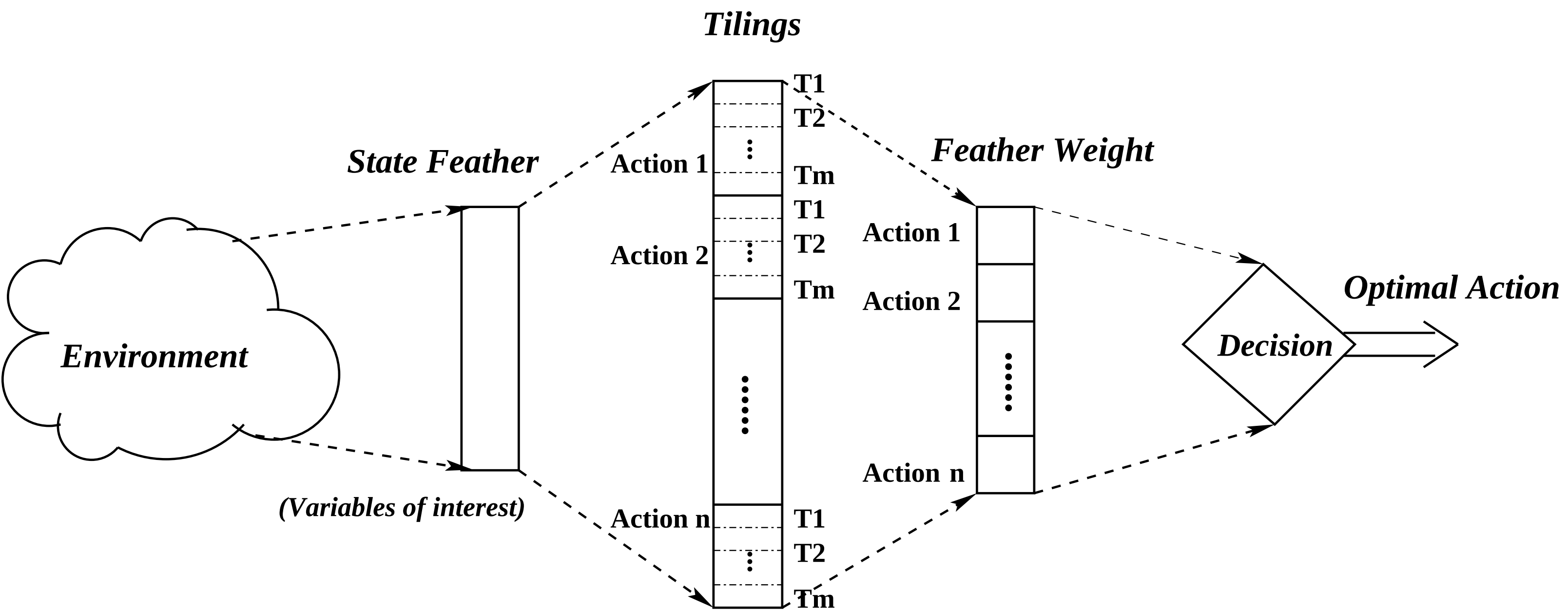
−20
−25
−30
−35
−40
−45
−50
0
500
1000
1500
2000
2500
Average Reward Per Episode
Episodes
Sarsa
Watkins
naive

Figure

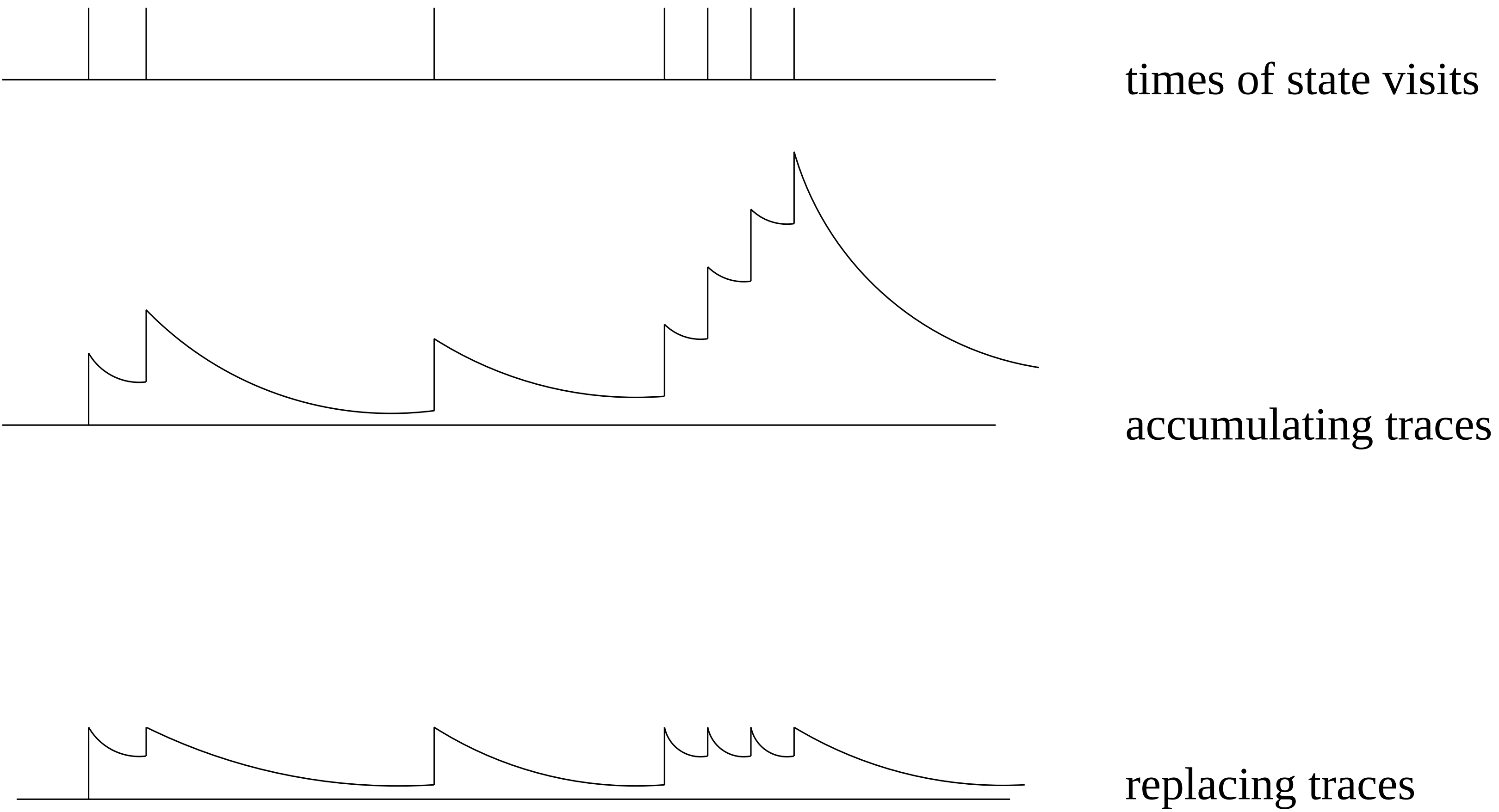


Average Reward Per Episode
−5
−10
−15
−20
−25
−30
−35
−40
−45
−50
0
500
1000
1500
2000
2500
Episodes
Non−Traces
Replacing Traces
Accumulating Traces

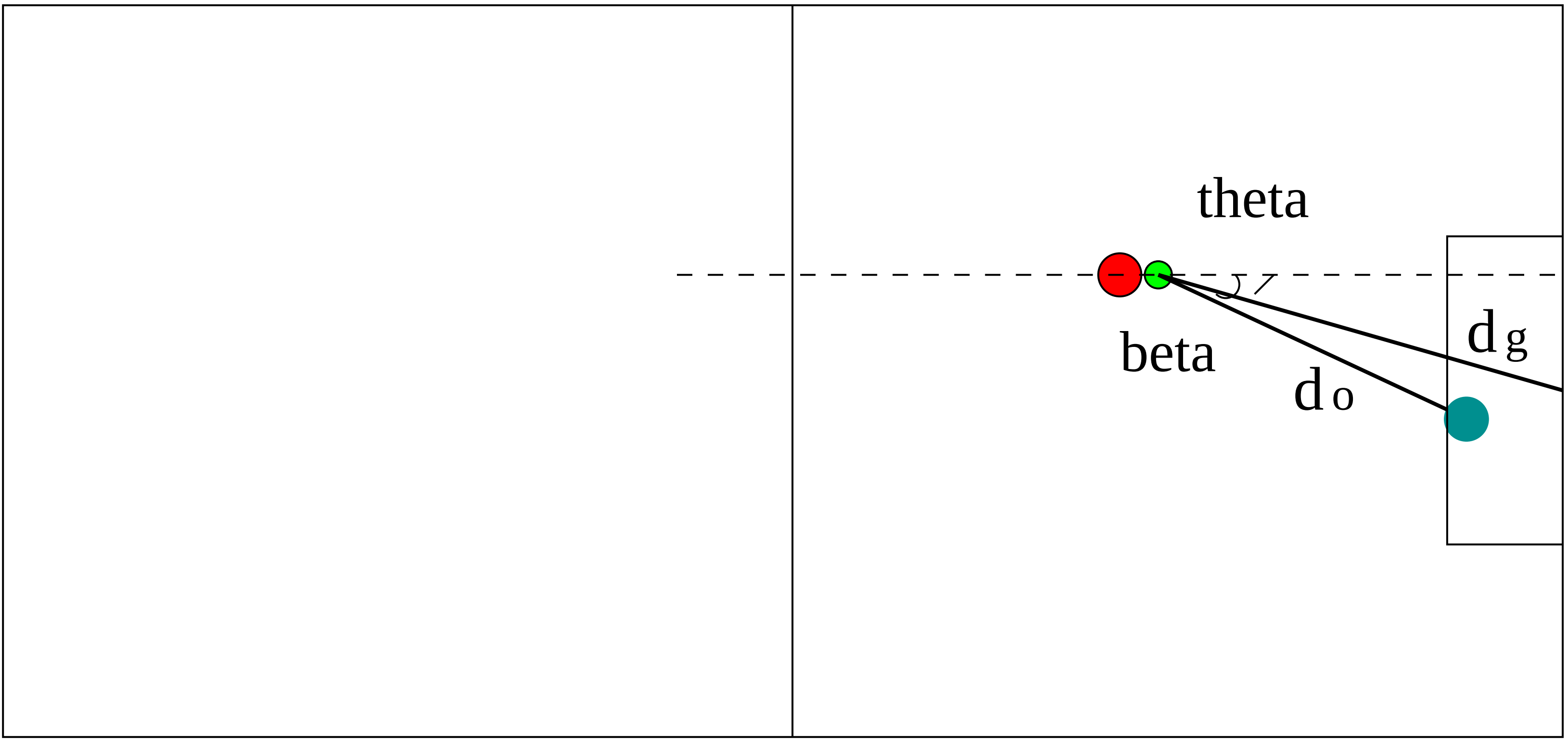
Figure


Tilings
State Feather
Feather Weight
Environment
(Variables of interest)
Action 1
Action 2
Action n
T1
T2
Tm
Action 1
Action 2
Action n
Decision
Optimal Action

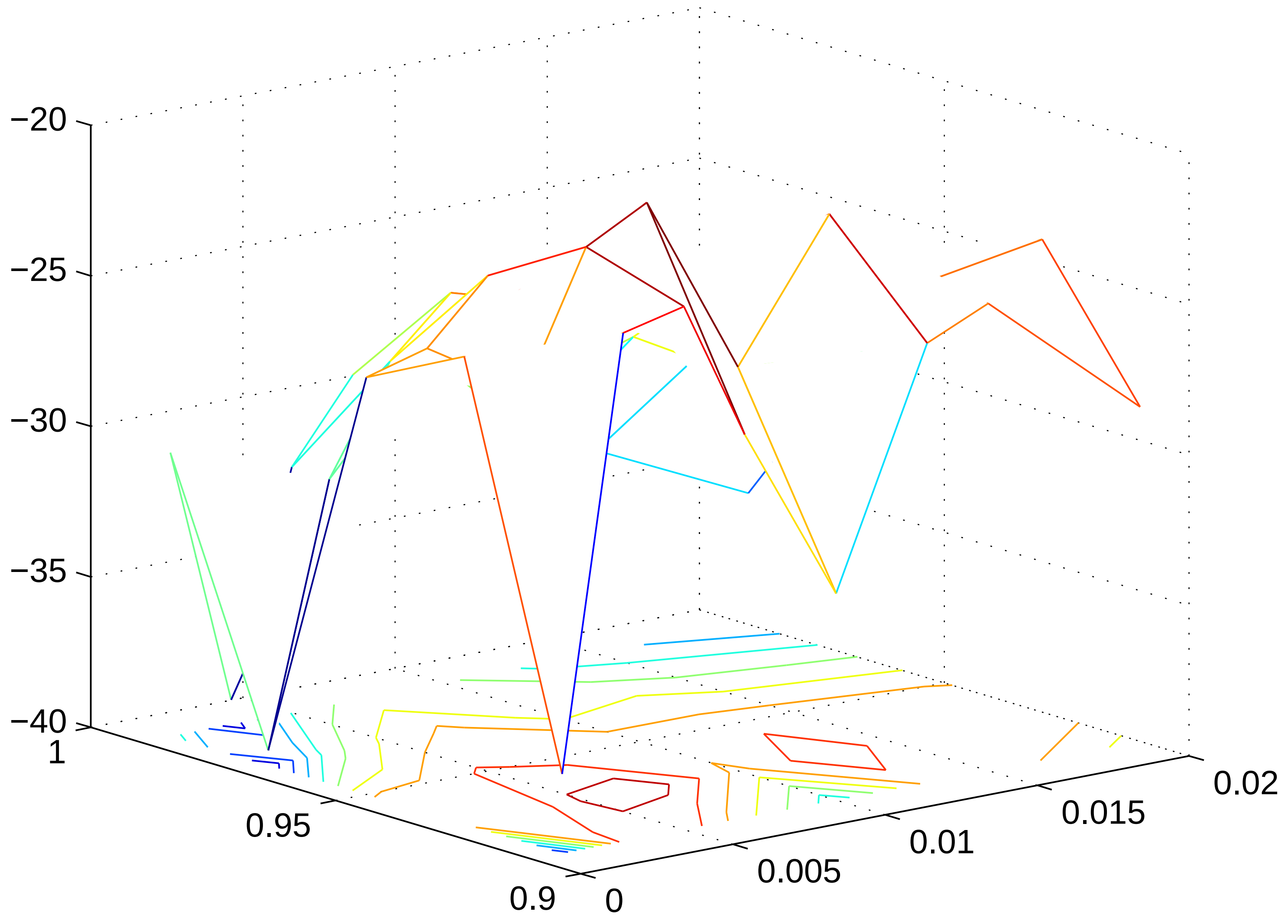
Figure


times of state visits
accumulating traces
replacing traces

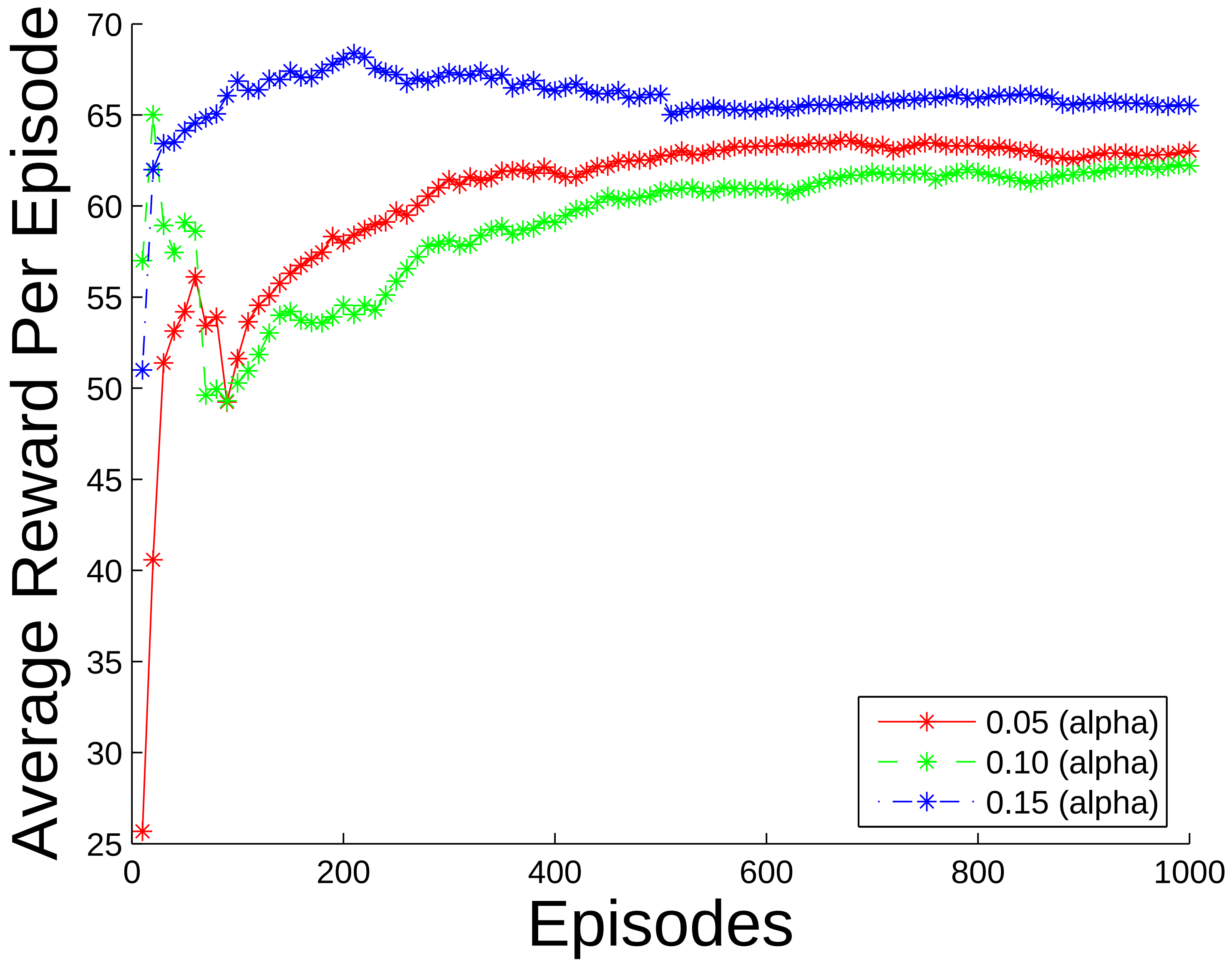
Figure



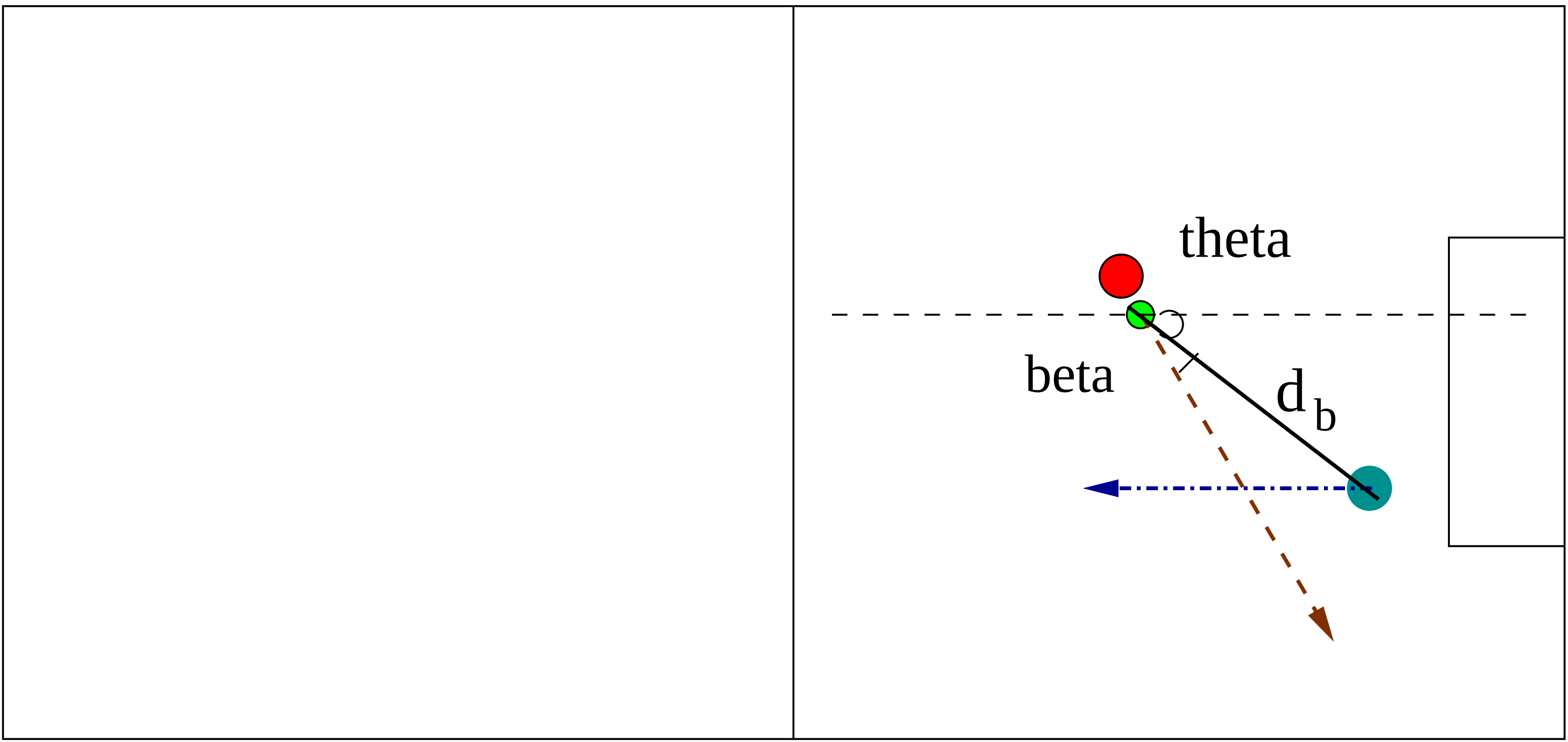
Figure



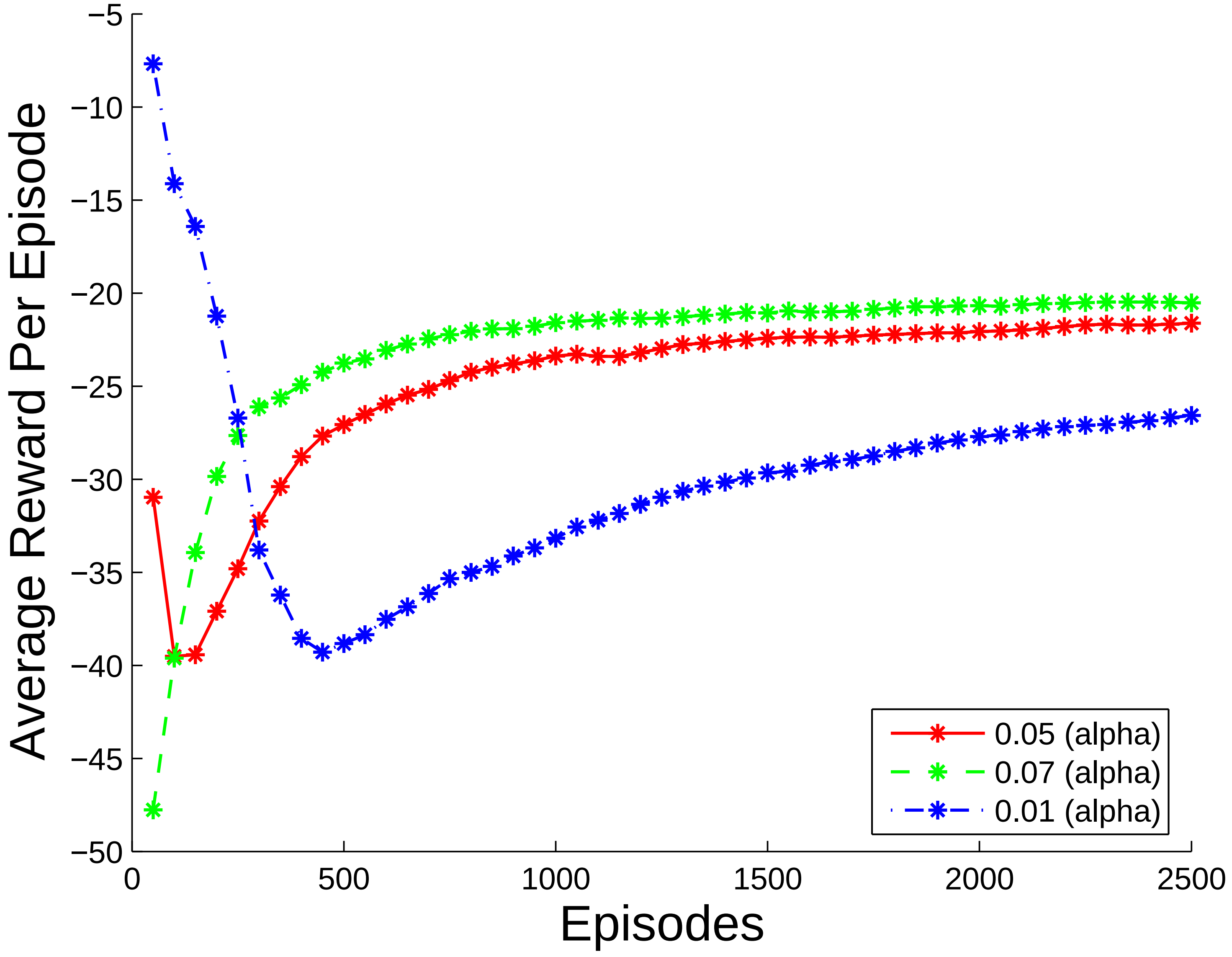
Figure


Average Reward Per Episode
Episodes
0.05 (alpha)
0.10 (alpha)
0.15 (alpha)
25
30
35
40
45
50
55
60
65
70
0
200
400
600
800
1000

Figure

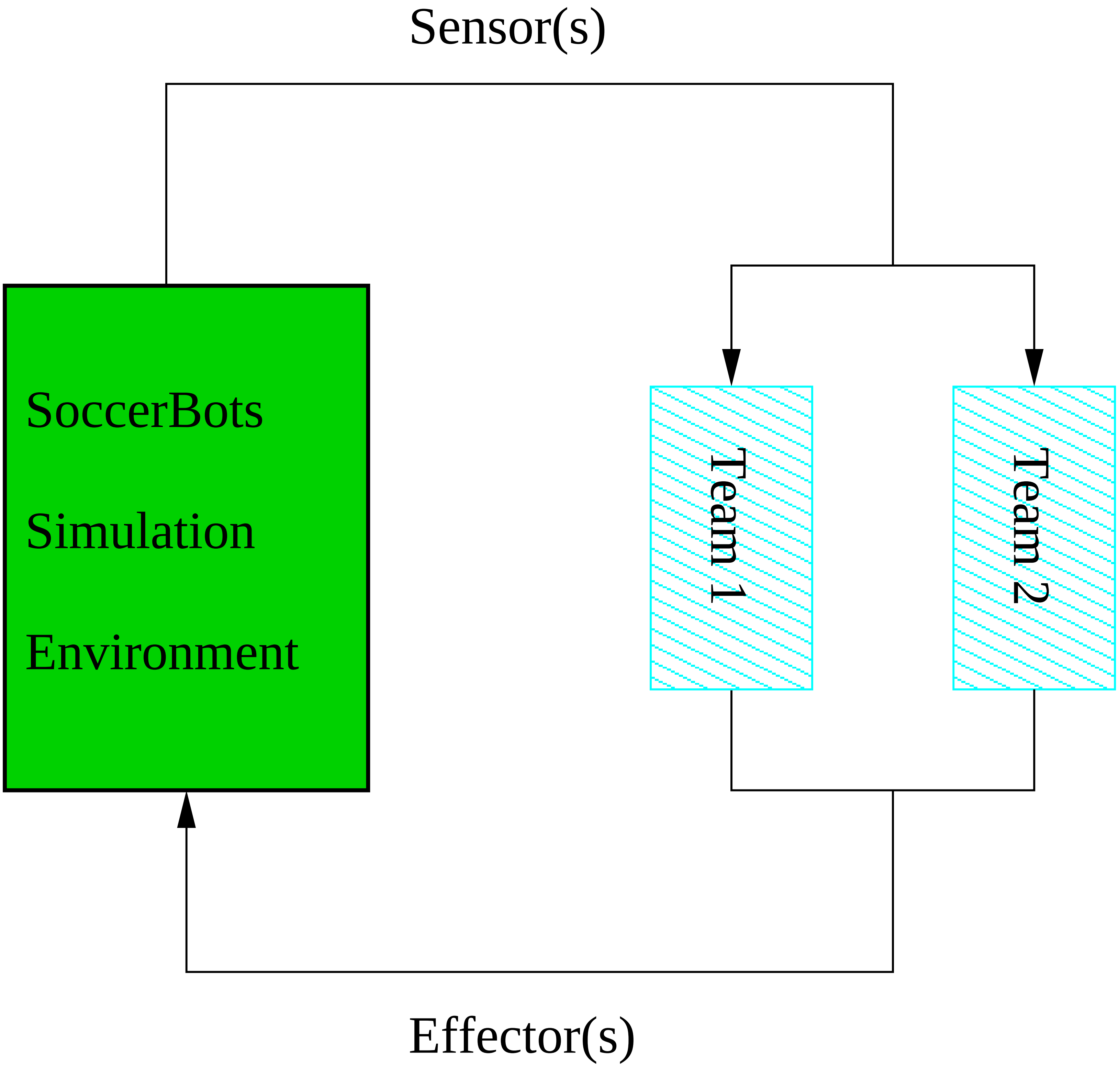


Figure

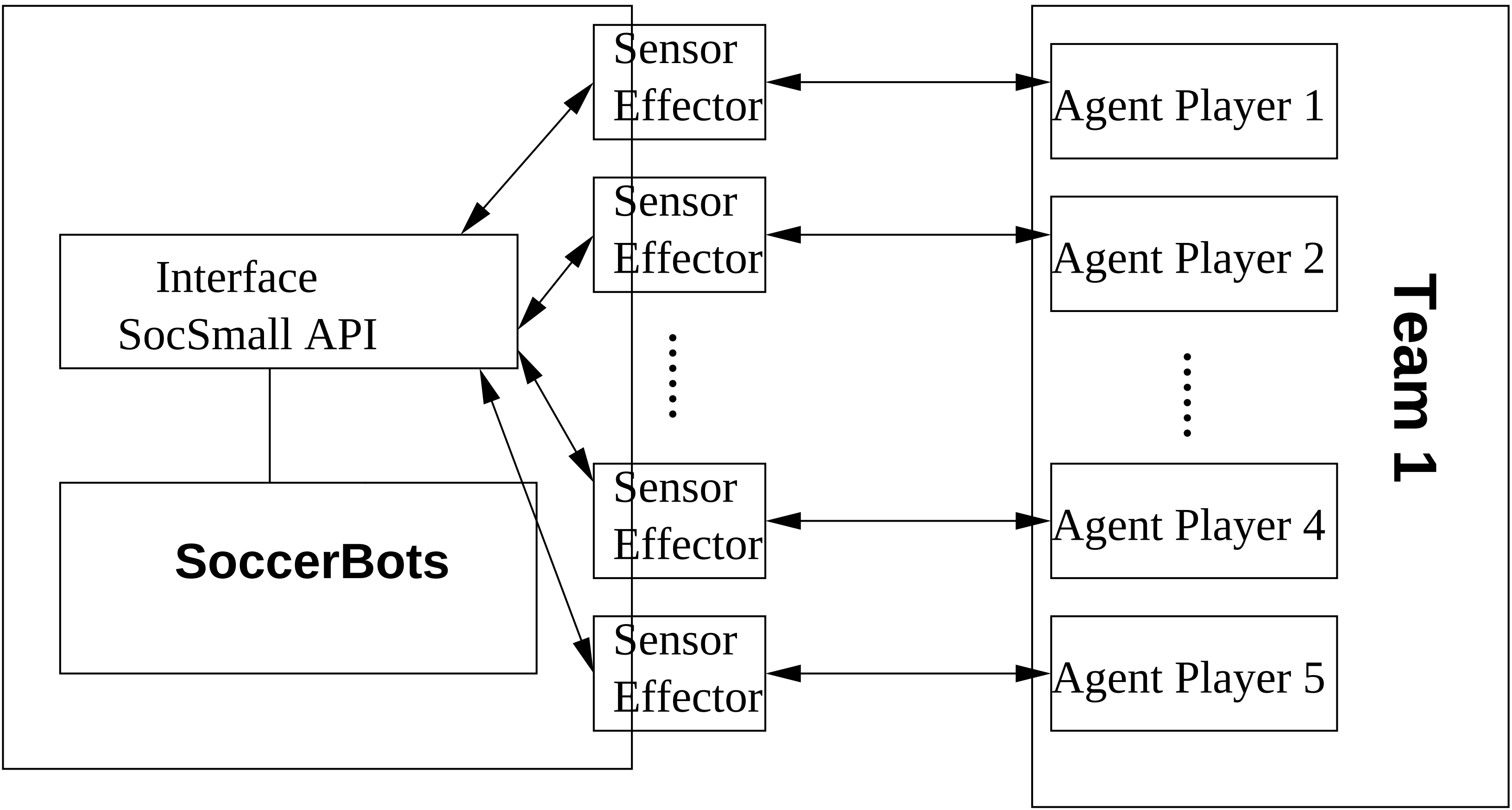


Average Reward Per Episode
Episodes
0.05 (alpha)
0.07 (alpha)
0.01 (alpha)
−5
−10
−15
−20
−25
−30
−35
−40
−45
−50
0
500
1000
1500
2000
2500

Figure


Sensor(s)
SoccerBots
Simulation
Environment
Team 1
Team 2
Effector(s)

Figure


Sensor
Effector
Sensor
Effector
Sensor
Effector
Sensor
Effector
Interface
SocSmall API
SoccerBots
Agent Player 1
Agent Player 2
Agent Player 4
Agent Player 5
Team 1