Speeding up Tabular Reinforcement Learning Using State-Action Similarities

Ariel Rosenfeld
Bar-Ilan University
arielros1@gmail.com

Matthew E. Taylor
Washington State University
matthew.e.taylor@wsu.edu

Sarit Kraus
Bar-Ilan University
sarit@cs.biu.ac.il

ABSTRACT

This paper proposes a novel method to speed up temporal difference learning by using state-action similarities. These hand-coded similarities are tested in three well-studied domains, demonstrating our approach's benefits. Additionally, a human subjects study with 16 programmers shows that the proposed approach can reduce the engineering effort of human designers. These results combine to show that our novel method is not only effective, but can be efficiently used by non-expert designers.

CCS Concepts

•**Computing methodologies** → **Artificial intelligence;**

Keywords

Reinforcement Learning, Human Study

1. INTRODUCTION

Reinforcement Learning (RL) [29] has had many successes, solving complex, real-world problems. When tackling such problems, the designer must face the question about how much human knowledge to inject to the system. From the AI or research perspective, the more that can be learned autonomously, the more interesting and beneficial the agent. From the engineering or more practical perspective, more human knowledge is desirable as it can help improve learning, but only if it is practical to gather or leverage, and only as long as it does not cause the agent to be limited to sub-optimal solutions after training.

Many successful RL applications have traditionally used highly engineered state features (e.g., 'the distance between the simulated robot soccer player with the ball to its closest opponent' and 'the minimal angle with the vertex at the simulated robot soccer player with the ball between the closest teammate and any of the opponents' [26]). With the recent successes of DeepRL [18], convolutional neural networks were shown to successfully learn features directly from pixel-level representations. However, such features are not necessarily optimal, significant amounts of human time is necessary to define the deep neural network's architecture, and significant amounts of data is required to learn the features. This paper proposes a different, and potentially complimentary, approach, in a 'shallow RL' setting.

In order to minimize confounding factors, this paper considers the simplest representation for temporal difference RL algorithms, a tabular representation of an action-value function, with variants of the well-studied Q-learning algorithm [34]. Our novel approach, which we name *SASS*, standing for State Action Similarity Solutions, allows the generalization of knowledge across state-action values in the action-value function table by leveraging hand-coded heuristics. While there are many ways of leveraging human knowledge in an RL learner by leveraging demonstrations or direct human knowledge (e.g., inverse reinforcement learning [21, 27], learning from demonstration [2, 33], learning from advice [15], etc.), SASS focuses on allowing users to specify *state-action similarities* in a given domain. In addition to showing that such similarities effectively improve learning, we also provide results from a human subject study, showing that such similarities can be provided by non-expert, RL-knowledgeable participants *in practice*.

Our approach and its integration with the Q-learning framework provide several desired properties that compare favorably with existing RL methods:

1. SASS is able to significantly speed up the agents' learning process in terms of sample efficiency.
2. Unlike various other generalization techniques, SASS retains the convergence and near-optimal guarantees of the original Q-learning algorithm.
3. SASS is based on an arbitrary designer-defined similarity function that does not assume any specific functional form, as other generalization techniques often do. Empirical evaluation with 16 participants showed that our approach improved learning in practice and did not require specific knowledge by the algorithm designers (the authors).

This paper focuses on showing how state-action similarities can be leveraged by very simple algorithms, allowing us to demonstrate the success of our ideas without confounding factors. We also believe these successes will extend to other, more complex, RL algorithms and that these similarities can be integrated with traditional function approximation techniques like neural networks, but these two extensions will be left to future work.

We evaluate our methodology in three RL tasks of varying complexity:

1. The "toy" task of simple robotic soccer, providing a basic setting for the evaluation of the proposed approach.
2. A large grid-world task named Pursuit, showing the scale-up of our approach.
3. The popular game of Mario, exemplifying our approach's benefits in a task with billions of states.

2. PRELIMINARIES AND BACKGROUND

An RL agent generally learns how to interact with an unfamiliar environment [29]. We define the RL task using the standard notation of a Markov Decision Process (MDP) [35]. An MDP is defined as $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma \rangle$ where

- $\mathcal{S}$ is the state-space;
- $\mathcal{A}$ is the action-space;
- $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ is a transition function where $\mathcal{T}(s, a, s')$ is the probability of making a transition from state s to state s' using action a;
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ is the reward function;
- $\gamma \in [0, 1]$ is the discount factor, which represents the significance of future rewards compared to present rewards.

In an RL task, $\mathcal{T}$ and $\mathcal{R}$ are initially unknown to the agent. In discrete time tasks, the agent interacts in a sequence of time steps. On each time step, the agent observes its state $s \in \mathcal{S}$ and is required to select an action $a \in \mathcal{A}$. The agent then receives a reward r according to the unknown $\mathcal{R}$ function and arrives at a new state s' according to the unknown $\mathcal{T}$ function. We define an agent's *experience* as a tuple $\langle s, a, r, s' \rangle$ where action a is taken in state s, resulting in reward r and a transition to next state s'. The agent's objective is to maximize the accumulated discounted rewards throughout its lifetime. Namely, the agent seeks to find a policy $\pi : \mathcal{S} \mapsto \mathcal{A}$ that maximizes the expected total discounted reward (i.e., expected return) from following it.

Temporal difference RL algorithms such as Q-learning [34] approximate an action-value function $Q : \mathcal{S} \times \mathcal{A} \mapsto \mathcal{R}$, which maps state-action pairs to the expected real-valued discounted return. Using an estimated Q function, which continues updating as long as the agent is operating in the environment, such RL algorithms specify how the agent should select which action to perform, i.e., which policy π the agent should follow, and how should the agent change its knowledge as a result of its experience. For each experience $\langle s, a, r, s' \rangle$, Q-learning updates the Q-value estimation according to the temporal difference update rule

$$Q(s, a) = Q(s, a) + \alpha e \tag{1}$$

where α is the learning rate and e is the temporal difference error term defined as

$$e \leftarrow r + \gamma max_{a'} Q(s', a') - Q(s, a) \tag{2}$$

If both $\mathcal{S}$ and $\mathcal{A}$ are finite sets, the Q function can be easily represented in a table, namely in an $|\mathcal{S}| \times |\mathcal{A}|$ matrix, where each state-action pair is saved along with its discounted return estimation. In this case, the convergence of Q-learning has been proven in the past (under standard assumptions). See Sutton and Barto [29] for more details.

In this work we focus on the Q-learning algorithm with a tabular representation of the Q function. This scheme is, perhaps, the most commonly applied in RL tasks. As mentioned before, tabular representation suffers from low convergence rates. To address this problem, designers infuse domain knowledge into the agent's learning process in different ways. A few recent examples include the incorporation of human provided feedback [22] or demonstrations [5] from which the agent can deduce knowledge, transferred policies from similar domains [4], or by using potential base reward shaping (PBRS) to bias the agent's actions [9].

In this paper we present a new approach to infuse prior knowledge using *similarity*. The notion of similarity allows the human designer to bias the generalization of the learning agent's experience across the state-action space according to the designer's prior knowledge and beliefs using a theoretically grounded technique. The generalization of knowledge over the state-action space is also possible using existing methods, such as Function Approximation (FA) [7] or neural networks (including those used in Deep Reinforcement Learning (DeepRL) [17]). These two approaches require significant effort on the part of the human designer. When using FA, the designer needs to abstract the state-action space in a sophisticated manner such that the assumed similarities will be modeled while avoiding unwanted coarse similarities. On the other hand, when using neural networks or DeepRL, the designer can avoid such feature extraction of the problem yet she must engineer an appropriate neural network architecture that will leverage state-action similarities in the problem (or her implicit prior knowledge). The success of the learning agent in such cases critically depends on the designer's effort.

The notion of *similarity* is also common in other techniques that allow the learning agent to provide predictions for unseen or infrequently visited states based on their similarity to other states. For example, the Texplore algorithm [10] uses supervised learning techniques (i.e., decision trees) to generalize the effects of actions across different states. The assumption is that actions are likely to have similar effects across states. Tamassia et al. [31] suggest a different approach by dynamically selecting state-space abstraction by which different states that share the same abstraction features are consider similar.

Brys et al. [5] hand code a distance metric over the state-space prior to the agent's learning and Taylor et al. [32] learn a distance metric on-line, based on gathered transition data. In both cases, the authors use the metrics to bias learning to improve learning by biasing the policy towards actions that have been demonstrated by people in similar states and by generalizing across state features based on the metric, respectively. Sequeira et al. [25] and Girgin et al. [8] have presented variations of this notion by online identifying associations between different states in order to define a state-space metric or equivalence relation.

Prior knowledge can also be exploited by various FA methods. For example, a designer can abstract the state-space such that similar states would be represented using the same meta-state. The trade-off is clear: if states are represented in a rich format (i.e., using a very detailed description), it will take the agent considerable time to learn the expected return associated with the many state-action pairs, especially if outcomes are stochastic [1]. This phenomenon is at its worse if the agent considers every feature making up the state-space. Unfortunately, the success of FA and DeepRL in practice heavily depends on parameter choices. Moreover, desired properties such as convergence and near-optimality are often lost in the process.

Updating more than a single table entry per experience has been also proposed in Dyna [28], an RL algorithm that learns both from direct environmental interaction and by planning over an approximate model learned from experience. Dyna attempts to learn more quickly, reducing envi-

ronment interactions (data complexity) by trading off time (computational complexity). This approach is most effective when the environment can be quickly modeled.

QS-learning

In order to integrate our approach within the Q-learning framework we adopt a previously introduced technique [24], where Q-learning is combined with a spreading function that "spreads" the estimates of the Q-function in a given state to neighboring states, exploiting an assumed spatial smoothness of the state-space. Formally, given an experience $\langle s, a, r, s' \rangle$ and a spreading function $\sigma : \mathcal{S} \times \mathcal{S} \mapsto [0, 1]$ that captures how *"close"* states s and s' are in the environment, a new update rule is used:

$$Q(\tilde{s}, a) = Q(\tilde{s}, a) + \alpha\sigma(s, \tilde{s})e \quad (3)$$

where e is the temporal difference error term (Eq. 2). The update rule in Eq. 3 is applied to all states in the environment after each experience. The resulting variation is denoted as QS-learning (where S stands for spreading).

Algorithm 1 QS-learning Algorithm

Require: State-space $\mathcal{S}$, Action-space $\mathcal{A}$, discount factor γ, learning rate α, similarity function $\sigma(s, a, s', a')$
  initialize Q arbitrarily (e.g. $Q(s, a) = 0$)
  for t=1,2,... **do**
    s is initialized as the *starting* state
    repeat
      choose an action $a \in A(s)$ based on an exploration strategy
      perform action a
      observe the new state s' and receive reward r
      $e \leftarrow r + \gamma \cdot \max_{a' \in A} Q(s', a') - Q(s, a)$
      for each $\tilde{s} \in \mathcal{S}$ **do**
        $Q(\tilde{s}, a) \leftarrow Q(\tilde{s}, a) + \alpha\sigma(s, \tilde{s})e$
      end for
      $s \leftarrow s'$
    until s' is a terminal state
  end for

To the best of our knowledge, this method was not extended or evaluated using state-action similarities nor was it tested in domains of larger complexity than simple grid worlds.

We now turn to discuss a few theoretical properties of the QS-learning algorithm which we will use later, in Section 3, in instantiating our approach to the Q-learning framework.

PROPOSITION 1. *Standard Q-learning is a special case of QS-learning.*

PROOF. By setting the function σ to the Kronecker delta ($\delta(\alpha, \beta) = 1$ if $\alpha = \beta$, otherwise $\delta(\alpha, \beta) = 0$), Eq. 3 is equivalent to Eq. 1. □

PROPOSITION 2. *QS-learning converges to the optimal policy given the standard condition for convergence of Q-learning and a similarity function σ that converges to the Kronecker delta over the state-action space at least as quickly as the learning rate α converges to zero.*

PROPOSITION 3. *QS-learning converges to a near-optimal policy given the standard condition for convergence of Q-learning with a σ which is fixed in time. The accuracy of the QS-learning depends on the accuracy of σ in hindsight.*

PROOF. Proposition 2 was proven in [23] and Proposition 3 was proven in [30]. Both propositions were proven for the update rule of Eq. 3 without loss of generality, and therefore the provided proofs apply to the QS-learning update rule of Eq. 5 as well. □

3. THE SASS APPROACH

We first formally define the similarity function:

DEFINITION 1. *Let $\mathcal{S}$, $\mathcal{A}$ be a state-space and an action-space, respectively. A similarity function $\sigma : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \times \mathcal{A} \mapsto [0, 1]$ maps every two state-action pairs in $\mathcal{S} \times \mathcal{A}$ to the degree to which we expect the two state-action pairs to have a similar expected return.*

σ is considered *valid* if for all state-action pairs s, a it satisfies

$$\sigma(s, a, s, a) > 0 \quad (4)$$

In this study we assume that the similarity function is defined by the designer. Automatic identification of similarities will be explored in future work.

Similarity functions can be defined in multiple ways to capture various assumptions about the state-action space. In this section we define, discuss and exemplify three similarity notions that can be easily instantiated to different domains. In Section 4, we provide a detailed instantiation of the proposed similarity notions to three RL tasks and in Section 4.2 we discuss a human subjects study showing our approach's practicality and advantage with human designers.

1: Representational similarity.

Representational similarity arises from the representation of the tasks' state-action space. FA is perhaps the most prominent example of the use of this technique. The function approximator (e.g., tile coding, neural networks, abstraction, etc. [29]) approximates the Q-value instead of using a tabular representation and therefore implicitly force a generalization over the feature space used by the approximator. A common method in this realm is using a factored state-space representation, where each state is represented by a vector of features which capture different characteristics of the state-space. Using such abstraction, one can easily define similarities using a metric over the factored state-action (e.g., [25, 5]). Defining representational similarities introduces a major engineering concern of choosing the right abstraction method or function approximator that would work well across the entire state-action space. That is, we wish to avoid (or at least minimize) the generalization of experiences in the state-action space where we believe such a generalization is incorrect, yet allow such a generalization in other regions. Representational similarity is the most basic form of similarity and it has repeatedly shown its benefit in real world applications. However, no one-size-fits-all method exists for efficiently representing the state-action space. See Figure 1 (a) for an illustration.

2: Symmetry similarity.

RL agents learn faster in smaller environments. Therefore, whenever possible, a designer seeks to consolidate state-action pairs that are identical or completely symmetrical in order to avoid redundancies. Zinkevich and Balch [36] formalized the concept of symmetry in MDPs and proved that if

such consolidation of symmetrical state-action is performed accurately the optimal Q function and the optimal policy are not altered. However, automatically identifying symmetries is computationally complex [19], especially when the symmetry is only *assumed*. For example, in the Pursuit domain one may consider the 90°, 180° and 270° transpositions of the state around its center (along with the direction of the action) as being similar (see Figure 1 (b)). However, as the predators do not know the prey's policy, which may be biased towards some absolute direction, they can only assume such symmetry exists.

3: Transition similarity.

As previously discussed, defining a metric over the state-space is a common practice in RL tasks. In deterministic environments, where the transition function does not impose any uncertainty, one can define state-action similarities by a reduction to the goal states' similarity. For example, given that states s and $\tilde{s}$ are similar according to some state-space similarity, then all state-action pairs that result in either s or $\tilde{s}$ should be considered similar to some extent. Another interpretation of the notion is to consider the *relative effects* of actions in different states. A relative effect is the change in the state's features caused by the execution of an action. Exploiting relative effects to speed up learning was proposed in [11, 13] in the context of model learning. For example, in the Mario domain, if Mario *walks right* or *runs right*, outcomes are assumed to be similar as both actions induce similar relative changes to the state (see Figure 1 (c)). In environments where the transition model cannot be a priori assumed, the above notions are not easy to instantiate.

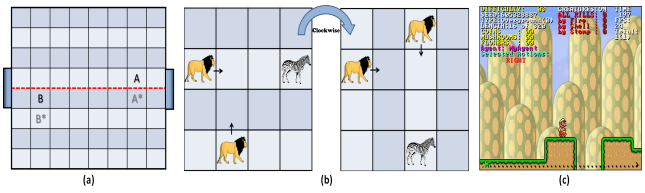

Figure 1: (a) Initial positioning of virtual players (A and B) in a simple robotic soccer task. The state in which both players are artificially placed one grid-cell down from their original position (marked as A* and B*) should be considered similar to the original one. (b) Two possible state-action pairs in the 4×4 Pursuit domain. Both state-action pairs are assumed to be similar. (c) A state in the Mario AI task. If Mario decides to walk right or run right, outcomes are assumed to be similar (falling into the gap).

SASS in the Q-learning Framework

For the purpose of incorporating similarities within Q-learning with tabular representation, we assume a similarity function $\sigma(s, a, s', a')$ is given by the designer. We then use this similarity function σ instead of the spreading function needed by the QS-learning in Algorithm 1. In words, for each experience $\langle s, a, r, s' \rangle$ that the agent encounters, depending on the similarity function σ, we seek to update more than a single $\langle s, a \rangle$ entry in the Q table. Therefore, we perform multiple updates, one for each entry $\langle \tilde{s}, \tilde{a} \rangle$ for which $\sigma(s, a, \tilde{s}, \tilde{a}) > 0$.

We use the following update rule:

$$Q(\tilde{s}, \tilde{a}) = Q(\tilde{s}, \tilde{a}) + \alpha\sigma(s, a, \tilde{s}, \tilde{a}) \left(r + \gamma \cdot \max_{a' \in A} Q(s', a') - Q(s, a) \right) \quad (5)$$

which, as discussed in Section 2, does not compromise the theoretical guarantees of the unadorned Q-learning algorithm.

The update rule states that as a consequence of experiencing $\langle s, a, r, s' \rangle$, an update is made to other pairs $\langle \tilde{s}, \tilde{a} \rangle$ as if the real experience actually was $\langle \tilde{s}, \tilde{a}, r, s' \rangle$ (discounted by the similarity function).

In order to avoid a time complexity of $O(|S||A|)$ per step, QS-learning should be restricted to update state-action pairs for which the similarity is larger than $\epsilon \geq 0$, where ϵ is a user chosen threshold. In our experiments (see Section 4), we found that only a minor increase in time-complexity was experienced in practice.

For the interest of clarity, from this point forward, we will use the term QS-learning using the above Q-learning-with-$SASS$ interpretation. Namely, using a designer-defined similarity function σ and the update rule of Eq. 5, we will modify the classic QS-learning algorithm yet keep its original name due to their inherent resemblance.

4. EVALUATION

We evaluate our approach in two experiments. First, we evaluate our approach against regular Q-learning, Q-learning combined with state-space abstraction and the *Dyna* algorithm. Second, we evaluate our approach in a human study in which we examine the human designers' engineering effort in instantiating our method to a simple grid-world task.

4.1 Speeding-up RL

We examine the proposed state-action similarities (Section 3) along with the QS-algorithm (Section 2) in three RL tasks of varying complexity: A "toy" task named Simple Robotic Soccer, a discrete grid-world task named Pursuit and the popular Mario AI game. Each task is first described and discussed. Then, the task is evaluated in the following manner: First, we instantiate the state-action similarities to the specific domain and examine the agent's learning performance over time (denoted QS). Then, we compare the learning performance to that of a Q-learning agent that uses state-space abstraction (denoted QA), a basic Q-learning agent (denoted Q) and a Dyna agent (denoted *Dyna*). Each task provides us with a unique opportunity to examine different similarity notions and combine our approach with existing ones: In the Simple Robotic Soccer task, presented in Section 4.1.1, we evaluate our approach's benefits in a basic settings which allows us to isolate the effect of our approach on the agent's learning performance and the human designer's effort. In the Pursuit task, presented in Section 4.1.2, we evaluate a more realistic task where symmetries can play a crucial role in alleviating the agent's learning effort. Last, we evaluate our approach in the Mario AI task, presented in Section 4.1.3, which emphasize the importance of transition similarities which were not effectively used in the first two tasks. All technical parameters used in this study (learning rates, exploration type, eligibility trace parameter, etc.) in the three tasks are fully specified in the code and are available in `http://www.biu-ai.com/RL`.

In all tasks we used basic PBRS [20] to avoid long learning periods under all conditions.[1] Note that PBRS allows one to modify the reward function of an MDP without altering the desired theoretical properties of Q-learning and QS-learning algorithms.

In the Pursuit and Mario AI tasks, we use $Q(\lambda)$-learning and $QS(\lambda)$-learning, which are slight variations of the Q-learning and QS-learning algorithms that use eligibility traces. The addition of eligibility traces to the evaluation was carried out as done by the authors of the recent papers from which the implementations have been taken. This provides us with an opportunity to compare SASS with recently provided solutions without altering their implementation. The interested reader can find the formal definition and discussion on eligibility traces elsewhere [29].

The evaluation of actual running times was done on a personal computer with 16 GB RAM and a CPU with 4 cores, each operating at 4 GHz.

4.1.1 *Simple Robotic Soccer.*

Task Description: Proposed in [14], the task is performed on an 8×8 grid world, defining the state-space S. Two simulated robotic players occupy distinct cells on the grid and can take one of 5 possible actions: North (N), South (S), East (E), West (W) or Idle (I). The simulated robots are designed to play a simplified version of soccer: At the beginning of each game players are positioned according to Figure 1 and possession of the ball is assigned to one of the players (either the learning agent or the fixed, handcoded-policy opponent[2]). During each turn, both players select their actions at the same time and the actions are executed in random order. When the attacking player (the player with the ball) executes an action that would take it to a square occupied by the other player, possession of the ball goes to the defender player (the player without the ball) and the move does not take place. A goal is scored when the player with the ball enters the goal region of the other player. Once a goal is scored the game is won; the agent who scored receives 1 point and the other agent receives -1 point, and the game is reset. The discount factor was set to 0.9 as done in the original paper. Our Python code (along with other codes used in this study) is available at `http://www.biu-ai.com/RL` for future research. The same code is also used in the human experiment specified in Section 4.2.

We used a basic state-space representation as done in [16], which is, to the best of our knowledge, the most recent investigation of the game. A state s is represented as a 5-tuple $\langle x_A, y_A, x_B, y_B, b\rangle$ where x_i and y_i indicate player i's position on the grid and $b \in \{A, B\}$ indicates which player has the ball. The action-space is defined as a set of 5 actions as specified above. Overall, the state-action space consists of approximately 41,000 state-action pairs.

Abstraction: We define the state-space abstraction, used by QA-learning using a simple distance-based approach, which represented each state according to the learning agent's distance to its opponent and goal.

Similarities: We define the state-action similarities, used by QS-learning, with respect to the similarity notions presented in Section 3: As *representational similarities*, the agent artificially moves *both players* together across the grid, keeping their original relative distance (see Figure 1). As the players are moved further and further away from their original positions, the similarity estimation gets exponentially lower. *Symmetry similarities* were identified using an imaginary horizontal line dividing the grid in half. Experiences in the upper half of the field are mirrored in the bottom part by mirroring states and actions with respect to the Y-axis and vice-versa. *Transition similarities* were not defined for this task.

Results: Four learning agents (QS-learning, QA-learning, Q-learning and *Dyna*) were trained for 1000 games. After each batch of 50 games, the learning was halted and 10,000 test games were played during which no learning occurred. The process was repeated 10,000 times. The results show that under the QS-learning condition, our agent learns significantly faster and outperforms both the QA-learning, Q-learning and *Dyna* conditions from the first batch onwards. See Figure 2 for a graphical representation of the learning process.

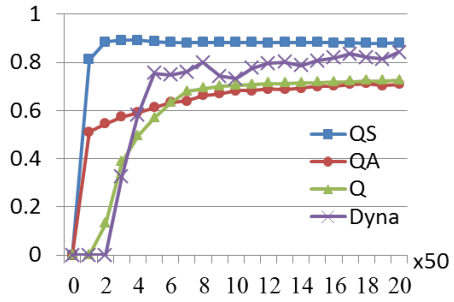

Figure 2: The QS-learning agent outperforms both the QA-learning, Q-learning and *Dyna* agents in Simple Robotic Soccer. The X-axis is the learning duration (in games) and Y-axis is the winning rate of the agent.

On average, the QS-learning agent updated 66 entries per iteration. However, the run-time for QS-learning agent was only slightly elevated compared to the Q-learning and QA-learning conditions. While the Q-learning and QA-learning complete their training (1,000 games each) in 0.03-0.04 of a second on average (with no significant difference between them), the QS-learning and *Dyna* agents complete the same training in 0.06 and 0.08 of a second, respectively.

4.1.2 *Pursuit.*

Task Description: The Pursuit task, which is also known as the Chase task or the Predator/Prey task, was proposed by Benda et al. [3]. For our evaluation we use the recently evaluated instantiation of Pursuit implemented in [6]. According to the authors' implementation, there are two predators and one prey, each of which can move in the four cardinal directions as well as choose to stay in place (5 actions each) on a 20×20 grid world. The prey is caught when a predator moves onto the same grid cell as the prey. In that case, a reward of 1 is given to the predators, 0 otherwise.

[1]The PBRS were defined as presented in the original papers from which each task was taken.

[2]The opponent was given a handcoded policy, similar to that used in the original paper, which instructs it to avoid colliding with the other player while it has the ball and to attempt to score a goal. While defending, the agent merely chases its opponent and tries to steal the ball.

We refer the reader to the original paper for the complete description of the underlining MDP and parameters. The authors use $Q(\lambda)$-learning, a slight variation of Q-learning, and therefore, so do we.

Abstraction: We use Brys et al. [6]'s implemented tile-code approximation for the QA-learning agent deployment.

Similarities: In our implementation, each state is represented as $\langle \Delta_{x_1}, \Delta_{y_1}, \Delta_{x_2}, \Delta_{y_2} \rangle$ where Δ_{x_i} (Δ_{y_i}) is the difference between predator i's x-index (y-index) and the prey's x-index (y-index), thereby we set a similarity of 1 for all states in which the relative positioning of the prey and predators is the same. Symmetry similarities were defined using 90°, 180° and 270° transpositions of the state around its center (along with the direction of the action). Symmetry similarities were also identified using imaginary horizontal and vertical lines dividing the grid in half. Experiences in the upper (left) half of the field are mirrored in the bottom (right) part by mirroring states and actions with respect to the Y-axis (X-axis) and vice-versa. Transition similarities were defined for all state-action pairs that are expected to result in the same state.

Results: Four learning agents (QS, QA, Q and $Dyna$) were trained for 10,000 games. After each batch of 500 games, the learning was halted and 10,000 test games were played during which no learning occurred. The process was repeated 10,000 times. The results show that in the QS-learning condition, our agent learns significantly faster and outperforms the Q-learning condition from the first batch onwards and the QA-learning and $Dyna$ conditions from the third batch onwards. See Figure 3 for a graphical representation of the learning process.

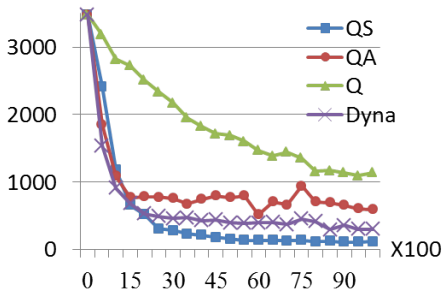

Figure 3: The QS-learning agent outperforms both the QA-learning and Q-learning agents in Pursuit.

On average, the QS-learning agent updated 12 entries per iteration. However, while the Q-learning and QA-learning agents complete their training (10,000 games each) in 8.5 seconds on average (with no significant difference between the two), the QS-learning agent completes the same training in 17.5 seconds on average. $Dyna$ performed significantly worse, averaging 87 seconds for its training. Unlike other conditions, $Dyna$ also introduced extreme memory requirements due to its model-based approach.

4.1.3 Mario AI.

Task Description: Super Mario Bros is a popular 2-D side-scrolling video game developed by the Nintendo Corporation. In this work we use the popular Mario AI version of the game, often used for the evaluation of RL techniques [12]. We use the recently evaluated formulation of the Mario AI task proposed by Suay et al. [27]. The authors use a 27-dimensional discrete state-variables representation of the state-space and model 12 actions that Mario can take. We refer the reader to the original paper for the complete description of the underlining MDP and parameters. Given the authors' abstraction of the state-space, the size of the state-action space is over 100 billion.[3] Due to the huge state-action space, Q-learning without the authors' abstraction will not be evaluated. Due to extreme memory requirements in run-time, the authors were unable to evaluate the $Dyna$ condition properly. Note that the authors use $Q(\lambda)$-learning and, therefore, so do we.

Similarities: As the original authors have already abstracted the state-space, we consider their provided agent to be a QA-learning agent.

Building upon the authors' abstraction, we define our similarities. We noticed that each state representation indicates whether Mario can jump or shoot using 2 Boolean variables. Given a state-action pair in which Mario does not jump or shoot, we define all respective states with the four variations of these two Boolean variables as similar to the original pair. Namely, if Mario walks right, then regardless of Mario's ability to shoot or jump, the state-action pair is considered similar to the original one. Symmetry similarities are defined using the mirroring of the state-actions across an imaginary horizontal line that divides the environment in half, with Mario in the middle. As illustrated in Figure 1, regardless of specific state, performing action a (e.g., move right) is similar to using action a+"run" (e.g., run right). That is, we assume each action introduces a similar relative change to a state whether it is executed with or without the running option.

Results: Two learning agents (QS-learning and QA-learning) were trained for 20,000 games. After each batch of 1000 games, the learning was halted and 1000 test games were played during which no learning occurred. The process was repeated 100 times. The two agents are compared against human performance level as evaluated in [27]. The results show that the QS-learning agent surpasses human performance level significantly faster than the QA-learning agent. See Figure 4 for a graphical representation of the learning process.

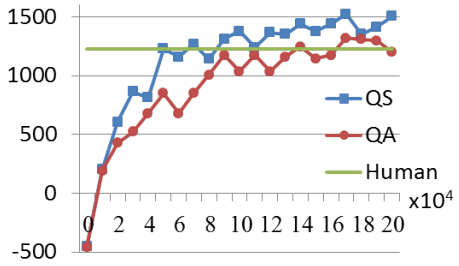

Figure 4: The QS-learning agent surpasses human performance level significantly faster than the QA-learning agent in Mario.

On average, the QS-learning agent updated 33 entries per

[3]Some of the states are never encountered in reality. For example, it is impossible to have Mario trapped by enemies from all directions at the same time.

iteration. However, the QA-learning agent requires 63.4% of the time as QS-learning agents to complete it's training (20,000 games each, 33 vs. 55 seconds) on average.

4.2 Human Study

A popular competing approach to the one proposed in this paper is using state-space abstraction. By using abstraction, a designer changes the state-space representation (usually reduces its size) and thereby speeds up Q-learning. We speculate that in most domains, with thoughtful and trial-and-error, a designer may find both good state-space abstractions and state-action similarities that will speed up the RL processes. However, in the following experiment we will show that given a limited amount of time, an RL-knowledgeable designer could benefit more from using the similarity approach which is presented in this study.

In the following we describe a first of its kind human study aimed at estimating *the engineering effort* needed by designers to speed up RL using the above two approaches in the Simple Robotic Soccer task (see Section 4.1.1). We will first describe our experimental setup and then discuss the results.

4.2.1 Experimental Design

We recruited 16 Computer Science grad-students (4 PhD students and 12 Master students, ranging in age from 23 to 43, 10 male and 6 female) who are majoring in AI to participate in the experiment and act as engineers of our RL agents. All students have prior knowledge of RL from advanced AI courses. The students are majoring in Machine Learning (7), Robotics (4) and other computational AI subfields (5).

Prior to the experiment, all of the participants participated in an hour-long tutorial reminding them of the basics of Q-learning and explaining the Simple Robotic Soccer task's specification. Participants were given two python codes: First, an implemented QA-learning algorithm in which participants had to design and implement a state-space abstraction. Specifically, the participants were requested to implement a single function that translates the naïve representation to their own state-space representation. Second, participants were given a QS-learning algorithm in which they had to implement a similarity function. Both codes already implemented all the needed mechanisms of the game and the learning agents and are available in `http://www.biu-ai.com/RL`.

We used a within-subjects experimental design where each participant was asked to participate in the task twice, a week apart. In both sessions, the participants' task was to design a learning agent that will outperform a basic Q-learning agent in terms of asymptotic performance and/or average performance (one would suffice to consider the task successful) by using either abstraction or similarities, in no more than 45 minutes of work.[4] That is, participants were asked to implement both the abstraction and the similarity function (only one in each session) that will allow the agent learn a good policy as quickly as possible. Participants were counter-balanced as to which function they had to implement first. We then tested the participant's submitted agents against the same hand-coded opponent against whom they trained. During each session, participants were able to test the quality of their designed agent at any time. This was done using the same procedure used in Section 4.1.1. Namely, by running the testing procedure the designed agent was trained for 1,000 games. After each batch of 50 games, the learning was halted and 10,000 test games were played during which no learning occurred. The winning ratio for these 10,000 test games was presented to the designer after each batch. Given a 'reasonable' number of updates per step, the procedure does not consume more than a few seconds on a standard PC.

After all agents were submitted, we analyzed the submitted agents using the same procedure used in Section 4.1.1. Namely, each agent received two scores: one for its average performance of during its learning period and one for the asymptotic performance of the agent, i.e., its performance after the training completed.

In order to allow designers to compare their agents success to a basic Q-learning agent (the benchmark agent they were requested outperform), each designer was presented with a report on a basic Q-learning that was trained and tested prior to the experiment using the same procedure described above.

4.2.2 Results

In the QS-learning condition, participants defined similarity functions. A similarity function is considered beneficial if it helps a QS-learning agent fulfill her task – outperforming basic Q-learning. Otherwise, we say that the similarity function is flawed (since Q-learning is a private case of QS-learning, see Proposition 1). Namely, flawed similarity functions introduce similarities that are far from correct in hindsight, and thus hinder the agent's learning.

When analyzing the average performance of the submitted agents, we see that out of the 16 QS-learning submitted agents, 12 (75%) successfully defined a beneficial similarity function. However, only 3 (19%) of the QA-learning agents outperformed Q-learning. The average winning ratio recorded for the QS-learning agents along their training was 68.2% compared to the 42.7% averaged by the QA-learning agent and 60.8% averaged by the benchmark Q-learning agent.

Asymptotically, 13 out of the 16 QS-learning agents (81%) asymptotically outperformed or matched the basic Q-learning performance. None of the QA-learning agents asymptotically outperformed Q-learning. On average, under the QS-learning condition, participants designed agents that asymptotically achieves an average winning ratio of 74.5%. The QA-learning condition which achieved 47.7% and 72.5% of the benchmark Q-learning agent.

The QS-learning agents' advantage over the QA-learning agents is increased when each QS-learning agent is analyzed with respect to its designer's QA-learning agent. Interestingly, *all 16 participants* submitted QS-learning agents which perform better than their submitted QA-learning agents both in their terms of average learning performance and asymptotic performance. Furthermore, for all participants, the QS-learning agent outperforms the QA-learning agent from the 3^{rd} test (the 150^{th} game) onwards. For 9 of the 16 participants (56%), the QS-learning agent outperformed the QA-learning agent from the first test onwards.

We further analyzed the types of similarities participants

[4]Ideally, we want participants to take as much time as they need. However, given that each participant had to dedicate about 3 hours for the whole process (1 hour tutorial + 1.5 hours of programming and half an hour of logistics) we could not ask participants for more than 45 minutes per condition.

defined under the QS-learning condition. This was done manually by examining the participants' codes and trying to reverse-engineer their intentions. Fortunately, due to the task's simple representation and dynamics, distinguishing between the different similarity notions was possible. It turns out that representational and symmetry similarity notions were the most prevalent among the submitted agents. In 8 out of the 16 QS-learning agents, representational similarities were instantiated, mainly using different variants of the idea presented before — moving one or both the virtual players across the grid. Symmetry similarities were used by 7 out of the 16 participants. All 7 of these agents used the idea of mirroring, where the state and action where mirrored across an imaginary horizontal line dividing the grid in half. Some of them also defined mirroring across an imaginary vertical line dividing the grid in half, with an additional change of switching ball position between the players. While we were able to show that each of these ideas is empirically beneficial on its own, we did not find evidence that combining them together brings about a significant change. Transitional similarities were only defined by 2 designers. Both designers tried to consider a more strategical approach and defined state-action similarities according to their strategical means. For example, moving towards the opponent while of defense is similar regardless of the initial state. It turns out that both provided transitional similarities were not beneficial in the way they were defined.

Only 4 out of the 16 participants (25%) used more than a single similarity notion while defining the similarity function. Interestingly, the two best performing QS-learning agents combined 2 notions in their similarity function. We speculate that combining more than a single similarity notion can be useful for most designers, yet in the interest of keeping with the task's tight time frame, participants refrained from exploring 'too many different directions' and focused on the ones they believed to be the most promising according to their initial tries.

Recall that 4 participants (25%) submitted flawed similarity functions. Although these participants were unable to find a beneficial similarity function, the submitted agents were not considerably worse than the basic Q-learning. The average performance (during training) for these 4 agents was 56.9% (compared to 60.8% of the basic Q-learning) and the average asymptotic score was 61.5% (compared to 72.5% of the basic Q-learning).

5. CONCLUSIONS

In this paper we proposed and extensively evaluated a novel approach for speeding up Q-learning agents using the notion of state-action similarities. Our theoretical discussion and empirical evaluation shows that the use of state-action similarities can: 1) Significantly speed up an agent's learning process; 2) Use inaccurate and sub-optimal similarities and still provide beneficial results; 3) Accommodate different similarity notions that can be instantiated easily by programmers and 4) Retain the desired theoretical properties of Q-learning. We hope this work will inspire other researchers to investigate their approach in human studies with actual programmers.

In this work we assumed a similarity function is defined by the human designer prior to the agent's learning. For future work we will tackle the challenge of on-line identification of similarities with and without a designer's input. In addition, we will investigate extending the proposed approach to other RL algorithms, linear function approximation, and DeepRL.

REFERENCES

[1] D. Andre and S. J. Russell. State abstraction for programmable reinforcement learning agents. In *AAAI/IAAI*, pages 119–125, 2002.

[2] B. D. Argall, S. Chernova, M. Veloso, and B. Browning. A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5):469–483, 2009.

[3] M. Benda. On optimal cooperation of knowledge sources. *Technical Report BCS-G2010-28*, 1985.

[4] R. A. Bianchi, L. A. Celiberto Jr, P. E. Santos, J. P. Matsuura, and R. L. de Mantaras. Transferring knowledge as heuristics in reinforcement learning: A case-based approach. *Artificial Intelligence*, 226:102–121, 2015.

[5] T. Brys, A. Harutyunyan, H. B. Suay, S. Chernova, M. E. Taylor, and A. Nowé. Reinforcement learning from demonstration through shaping. In *IJCAI*, pages 3352–3358, 2015.

[6] T. Brys, A. Nowé, D. Kudenko, and M. E. Taylor. Combining multiple correlated reward and shaping signals by measuring confidence. In *AAAI*, pages 1687–1693, 2014.

[7] L. Busoniu, R. Babuska, B. De Schutter, and D. Ernst. *Reinforcement learning and dynamic programming using function approximators*, volume 39. CRC press, 2010.

[8] S. Girgin, F. Polat, and R. Alhajj. Positive impact of state similarity on reinforcement learning performance. *IEEE Transactions on Cybernetics*, 37(5):1256–1270, 2007.

[9] A. Harutyunyan, S. Devlin, P. Vrancx, and A. Nowé. Expressing arbitrary reward functions as potential-based advice. In *AAAI*, pages 2652–2658, 2015.

[10] T. Hester and P. Stone. Texplore: real-time sample-efficient reinforcement learning for robots. *Machine learning*, 90(3):385–429, 2013.

[11] N. K. Jong and P. Stone. Model-based function approximation in reinforcement learning. In *AAMAS*, page 95. ACM, 2007.

[12] S. Karakovskiy and J. Togelius. The mario AI benchmark and competitions. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):55–67, 2012.

[13] B. R. Leffler, M. L. Littman, and T. Edmunds. Efficient reinforcement learning with relocatable action models. In *AAAI*, volume 7, pages 572–577, 2007.

[14] M. L. Littman. Markov games as a framework for multi-agent reinforcement learning. In *ICML*, volume 157, pages 157–163, 1994.

[15] R. Maclin, J. Shavlik, L. Torrey, T. Walker, and E. Wild. Giving advice about preferred actions to reinforcement learners via knowledge-based kernel regression. In *Proceedings of the National Conference on Artificial intelligence*, volume 20, page 819. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2005.

[16] M. F. Martins and R. A. Bianchi. Heuristically-accelerated reinforcement learning: A comparative analysis of performance. In *Conference Towards Autonomous Robotic Systems*, pages 15–27. Springer, 2013.

[17] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[18] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[19] S. M. Narayanamurthy and B. Ravindran. On the hardness of finding symmetries in markov decision processes. In *ICML*, pages 688–695, 2008.

[20] A. Y. Ng, D. Harada, and S. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*, volume 99, pages 278–287, 1999.

[21] A. Y. Ng and S. J. Russell. Algorithms for inverse reinforcement learning. In *Icml*, pages 663–670, 2000.

[22] B. Peng, J. MacGlashan, R. Loftin, M. L. Littman, D. L. Roberts, and M. E. Taylor. A need for speed: Adapting agent action speed to improve task learning from non-expert humans. In *AAMAS*, pages 957–965, 2016.

[23] C. Ribeiro and C. Szepesvári. Q-learning combined with spreading: Convergence and results. In *Procs. of the ISRF-IEE International Conf. on Intelligent and Cognitive Systems (Neural Networks Symposium)*, pages 32–36, 1996.

[24] C. H. Ribeiro. Attentional mechanisms as a strategy for generalisation in the q-learning algorithm. In *Proceedings of ICANN*, volume 95, pages 455–460, 1995.

[25] P. Sequeira, F. S. Melo, and A. Paiva. An associative state-space metric for learning in factored mdps. In *Portuguese Conference on Artificial Intelligence*, pages 163–174. Springer, 2013.

[26] P. Stone, G. Kuhlmann, M. E. Taylor, and Y. Liu. Keepaway soccer: From machine learning testbed to benchmark. In I. Noda, A. Jacoff, A. Bredenfeld, and Y. Takahashi, editors, *RoboCup-2005: Robot Soccer World Cup IX*, volume 4020, pages 93–105. Springer Verlag, Berlin, 2006.

[27] H. B. Suay, T. Brys, M. E. Taylor, and S. Chernova. Learning from demonstration for shaping through inverse reinforcement learning. In *AAMAS*, pages 429–437, 2016.

[28] R. S. Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *ACM SIGART Bulletin*, 2(4):160–163, 1991.

[29] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 1998.

[30] C. Szepesvári and M. L. Littman. A unified analysis of value-function-based reinforcement-learning algorithms. *Neural computation*, 11(8):2017–2060, 1999.

[31] M. Tamassia, F. Zambetta, W. Raffe, F. Mueller, and X. Li. Dynamic choice of state abstraction in q-learning. In *ECAI)*, 2016.

[32] M. E. Taylor, B. Kulis, and F. Sha. Metric Learning for Reinforcement Learning Agents. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, May 2011.

[33] M. E. Taylor, H. B. Suay, and S. Chernova. Integrating reinforcement learning with human demonstrations of varying ability. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 617–624. International Foundation for Autonomous Agents and Multiagent Systems, 2011.

[34] C. J. C. H. Watkins. *Learning from delayed rewards*. PhD thesis, University of Cambridge England, 1989.

[35] C. C. White III and D. J. White. Markov decision processes. *European Journal of Operational Research*, 39(1):1–16, 1989.

[36] M. Zinkevich and T. Balch. Symmetry in markov decision processes and its implications for single agent and multi agent learning. In *ICML*, 2001.