Leveraging Human Knowledge in Tabular Reinforcement Learning: A Study of Human Subjects

Paper #3019

Abstract

Reinforcement Learning (RL) can be extremely effective in solving complex, real-world problems. However, injecting human knowledge into an RL agent may require extensive effort on the human designer's part. To date, human factors are generally not considered in the development and evaluation of possible approaches. In this paper, we propose and evaluate a novel method, based on human psychology literature, which we show to be both effective and efficient, for both expert and non-expert designers, in injecting human knowledge for speeding up tabular RL.

1 Introduction

Reinforcement Learning [Sutton and Barto, 1998] (RL) has had many successes, solving complex, real-world problems. However, unlike supervised machine learning, there is no standard framework for non-experts to easily try out different methods (e.g., Weka [Witten *et al.*, 2016]). Another barrier to wider adoption of RL methods is the fact that injecting human knowledge, which can significantly improve the speed of learning, can be difficult for a human designer. In order for RL methods to move beyond requiring developers to fully understand the "black arts" of generalization, approximation, and biasing, it is critical that the community better understand if and how non-expert humans can provide useful information. This paper, rather than requiring RL experts to provide such biases, focuses on participants who have some background in AI and coding, but little experience in RL.

There are many approaches for leveraging human knowledge in an RL learner. Perhaps the most prominent one is function approximation [Busoniu *et al.*, 2010] (FA), which allows a designer to inject his knowledge by abstracting the domain appropriately and thereby allowing the agent to generalize its experience quickly. Many successful RL applications have used highly engineered state features to bring about successful learning performance (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' [Stone *et al.*, 2006]). While different methods of leveraging human knowledge in RL learners have been thoroughly investigated with respect to their theoretical properties and empirical performance in various settings, their deployment often requires extensive engineering and expertise on the designers' part, which is generally not considered (e.g., methods are not evaluated in terms of the amount of time a developer must invest to fine-tune parameters, select appropriate state representations, etc.).

The baseline approach in this paper is to allow no generalization: an agent's interactions with its environment will immediately affect only its current state (in a tabular representation). We will compare this baseline with a novel approach, which we name *SASS* (standing for State Action Similarity Solutions), for speeding up temporal difference learning using state-action similarities. Our approach is based on the well-established psychological theory of constructivism [Bruner, 1957], by which a designer can define and refine both complex and simplistic constructs (or *similarities*) in the state-action space, according to the designer's abilities, knowledge, and beliefs, using hand-coded similarities to simplify the RL agent's task. To avoid confounding factors, we consider a simple temporal difference RL algorithm: Q-learning [Watkins, 1989] with a tabular representation.

We test our proposed approach in a human study consisting of three experts (highly experienced programmers with an RL background, but not co-authoring this paper) and 16 non-expert graduate students. To that end, three RL tasks of varying complexity are considered, showing that SASS is effective for both expert and non-expert designers alike.

This paper argues that in order to bring about a wider adoption of RL techniques, and specifically of generalization techniques, it is essential to both investigate and develop techniques appropriate for both expert and non-expert designers.

2 Preliminaries and Background

An RL agent generally learns how to interact with an unfamiliar environment [Sutton and Barto, 1998]. We define the RL task using the standard notation of a Markov Decision Process (MDP). 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; $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ defines the transition probability; $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function; and $\gamma \in [0, 1]$ is the discount factor.

We assume $\mathcal{T}$ and $\mathcal{R}$ are initially unknown to the agent and the agent seeks to learn a policy $\pi : \mathcal{S} \mapsto \mathcal{A}$ that maximizes the expected total discounted reward (i.e., the expected return) while interacting with the environment.

Temporal difference RL algorithms such as Q-learning [Watkins, 1989] approximate an action-value function $Q : \mathcal{S} \times \mathcal{A} \mapsto \mathcal{R}$, mapping state-action pairs to the expected real-valued discounted return. Q-learning updates the Q-value estimation according to the temporal difference update rule

$$Q(s,a) = Q(s,a) + \alpha(r + \gamma max_{a'} Q(s',a') - Q(s,a))$$

where α is the learning rate. When $\mathcal{S}$ and $\mathcal{A}$ are finite, Q can be represented in a table. In this case, the convergence of Q-learning has been proven in the past (under standard assumptions [Sutton and Barto, 1998]).

RL in general, and tabular representations in particular, can suffer from slow learning rates. To address this problem, designers infuse domain-specific knowledge into the agent's learning process in different ways, enabling better generalization across small numbers of samples. The most common approach is to use FA. When using FA, the designer needs to abstract the state-action space in a sophisticated manner such that the (presumed) similar states or state-action pairs will be updated together and dissimilar states or state-action pairs are not. With the recent successes of DeepRL [Mnih *et al.*, 2015], convolutional neural networks were shown to successfully learn features directly from pixel-level representations. However, such features are not necessarily optimal; significant amounts of designer time is necessary to define the deep neural network's architecture, and significant amounts of data is required to learn the features.

Our proposed method, SASS, leverages a human designer's constructivism [Bruner, 1957]. Constructivism is a well-established psychological theory where people make sense of the world (situations, people, etc.) by making use of constructs (or clusters), which are perceptual categories used for evaluation by considering members of the same construct as *similar*. It has been shown that people who have many different and abstract constructs have greater flexibility in understanding the world, and are usually more robust against inconsistent signals. The SASS approach is inspired by constructivism, allowing a designer to define both complex as well as simplistic constructs of similar state-action pairs, according to one's knowledge, abilities and beliefs, and refine them as more experience is gained. Unlike more complex types of generalization (e.g., specifying the width of a tile, the number of tiles, and the number of tilings in a CMAC [Albus, 1981] or specifying the number of neurons, number of layers, and activation functions in a deep net).

The notion of generalization through *similarity* is also common in other techniques that allow the learning agent to provide predictions for unseen or infrequently visited states. For example, Texplore [Hester and Stone, 2013] uses supervised learning techniques 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. [2016] suggest a different approach by dynamically selecting state-space abstraction by which different states that share the same abstraction features are considered similar. Sequeira et al. [2013] and Girgin et al. [2007] have presented variations of this notion by online identifying associations between different states in order to define a state-space metric or equivalence relation. However, all of these methods assume that an expert RL designer is able to iteratively define and test the required similarities without explicit cost.

Another related line of research investigates providing direct biasing from non-expert humans, such as incorporating human-provided feedback [Knox and Stone, 2010; Peng *et al.*, 2016] or demonstrations [Brys *et al.*, 2015]. In this paper we focus on a complementary approach, leveraging human knowledge about *similarity*, rather than requiring a user to teleoperate the agent or provide on-line feedback.

Lastly, alternative updating approaches such as eligibility traces [Sutton and Barto, 1998], where multiple states can be updated based on time since visitation, and Dyna [Sutton, 1991], where the agent learns both from direct environmental interaction and by planning over an approximate model learned from experience, are popular as well. For ease of analysis, this paper does not incorporate such methods, but we note that our SASS approach is compatible with these and other additional speed-up techniques.

QS-learning

In order to integrate our generalization approach within the Q-learning framework we adopt a previously introduced technique [Ribeiro, 1995], 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:

$$Q(\tilde{s},a) = Q(\tilde{s},a) + \alpha\sigma(s,\tilde{s})\delta \quad (1)$$

where δ is the temporal difference error term $(r + \gamma max_{a'} Q(s',a') - Q(s,a))$. The update rule in Eq. 1 is applied to all states in the environment after each experience. The resulting variation is denoted as QS-learning (S stands for spreading). This method was only tested with author-defined spreading functions in simple grid worlds.

Note that standard Q-learning is a special case of QS-learning by setting the function σ to the Kronecker delta ($\delta(\alpha,\beta) = 1$ if $\alpha = \beta$, otherwise $\delta(\alpha,\beta) = 0$).

Proposition 1. *QS-learning converges to the optimal policy given the standard condition for convergence of Q-learning and either: 1) σ which is fixed in time; or 2) σ that converges to the Kronecker delta over the state-action space at least as quickly as the learning rate α converges to zero.*

Proof. The proposition is a combination of two proofs available in [Szepesvári and Littman, 1999] and [Ribeiro and Szepesvári, 1996]. Both were proven for the update rule of Eq. 1 without loss of generality, and therefore apply to the QS-learning update rule of Eq. 2 as well. □

3 The SASS Approach

According to constructivism literature, and specifically personal construct psychology [Kelly, 1955], while designing and testing an RL agent, the human designer himself learns the traits of the domain at hand by identifying patterns and domain-specific characteristics. To accommodate both prior knowledge and learned insights (which may change over time), it is necessary to allow the designer to easily explore and refine different similarity hypotheses. For example, a designer may have an initial belief that the state-action pair s, a has the same expected return as s', a'. Using FA, this can be easily captured by mapping both pairs into a single meta state-action pair. However, after gaining some experience in the domain, the designer refines his belief and presumes that the two pairs are merely *similar* (they would have close expected returns if they were to be modeled separately).

In this study we assume that the similarity function is defined and refined by a human designer during the development of the RL agent as follows.

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 $\forall \langle s, a \rangle$, $\sigma(s, a, s, a) > 0$.*

Similarity functions can be defined in multiple ways to capture various assumptions and insights about the state-action space. As shown in constructivism literature [Bruner, 1957], some people may use simplistic, crude similarities that allow quick generalizing of knowledge across different settings. Others may use complex and sophisticated similarity functions that will allow a more fine-grained generalization. Although people can easily identify similarities in real-life, they are often incapable of articulating sophisticated rules for defining such similarities. Therefore, in the following, we identify and discuss three notable similarity notions that were encountered repeatedly in our human study (Section 4), covering the majority of human-designed similarity functions.

1) Representational Similarity from the tasks' state-action space. FA is perhaps the most popular example of the use of this technique. The function approximator (e.g., tile coding, neural networks, abstraction, etc.) approximates the Q-value and therefore implicitly forces a generalization over the feature space. A common method is using a factored state-space representation, where each state is represented by a vector of features that capture different characteristics of the state-space. Using such abstraction, one can define similarities using a metric over the factored state-action (e.g., [Sequeira *et al.*, 2013; Brys *et al.*, 2015]). Defining representational similarities introduces the major engineering concern of choosing the right abstraction method or FA that would work well across the entire state-action space, while minimizing generalizing between dissimilar state-actions. Representational similarity has repeatedly shown its benefit in real world applications, but no one-size-fits-all method exists for efficiently representing the state-action space. See Figure 1 (a) for an illustration.

2) Symmetry Similarity seeks to consolidate state-action pairs that are identical or completely symmetrical in order to avoid redundancies. Zinkevich and Balch [2001] formalized the concept of symmetry in MDPs and proved that if such consolidation of symmetrical state-actions is performed accurately, then the optimal Q function and the optimal policy are not altered. However, automatically identifying symmetries is computationally complex [Narayanamurthy and Ravindran, 2008], 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 (potentially biased) policy, they can only assume such symmetry exists.

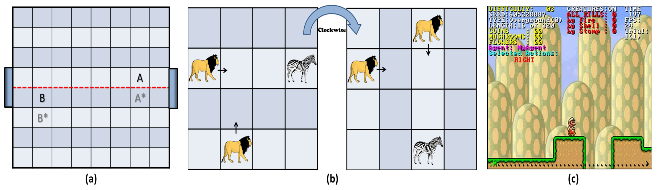


Figure 1: (a) Players in the simple robotic soccer task are A and B; one cell down (A* and B*) are considered similar. (b) Two similar state-action pairs in the Pursuit domain. (c) A state in the Mario AI task where walking or running right are similar (i.e., falling into the gap).

3) Transition Similarity can be defined based on the idea of *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 [Jong and Stone, 2007; Leffler *et al.*, 2007] 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 with complex or non-obvious transition models, it can be difficult to intuit this type of similarity.

SASS in the Q-learning Framework

We use the designer-provided similarity function $\sigma(s, a, s', a')$. In words, for each experience $\langle s, a, r, s' \rangle$ that the agent encounters, depending on the similarity function σ, we potentially update more than a single $\langle s, a \rangle$ entry in the Q table. Multiple updates, one for each entry $\langle \tilde{s}, \tilde{a} \rangle$ for which $\sigma(s, a, \tilde{s}, \tilde{a}) > 0$, are performed using the following update:

$$Q(\tilde{s}, \tilde{a}) = Q(\tilde{s}, \tilde{a}) + \alpha\sigma(s, a, \tilde{s}, \tilde{a})\delta \qquad (2)$$

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

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 0. In our experiments (see Section 4) we found only a minor increase in time-complexity for most human designers.

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. 2, we will modify the classic QS-learning algorithm yet keep its original name due to their inherent resemblance.

4 Evaluation

We consider two subject groups: experts and non-experts.

4.1 Expert Study

We recruited 3 highly experienced, expert programmers with Masters degrees in Computer Science and proven experience in RL. None of the experts co-author this paper. Each expert was asked to implement 4 RL agents: a basic Q-learning agent (denoted Q), QS-learning agent (denoted QS), a Q-learning agent that uses state-space abstraction (denoted QA) and a Dyna agent (denoted $Dyna$). Each was given a different RL task: a "toy" task named Simple Robotic Soccer, a grid-world task named Pursuit, and the Mario AI game.

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 allows us to compare SASS with recently provided solutions without altering their implementation.

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. All technical parameters used in the three tasks in this study (learning rates, exploration type, eligibility trace parameter, etc.) are fully specified in the code and are available at `http://RemovedForBlindReview`.

Simple Robotic Soccer

Proposed in [Littman, 1994], 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 either move in the four cardinal directions or stay in place (5 actions each). 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[1]). 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.

We used a basic state-space representation as done in [Martins and Bianchi, 2013], which is, to 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.

QA used a simple distance-based approach, which represented each state according to the learning agent's distance to its opponent and goal.

QS used two major similarity notions: First, *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. Second, *symmetry similarities* – 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 by the expert for this task.

Results: Each 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 process was repeated 10,000 times. The results show that under the QS condition, the agent learns significantly faster and outperforms the QA, Q and $Dyna$ conditions from the first batch onwards. See Figure 2a for a graphical representation of the learning process.

On average, QS updated 66 entries per iteration. The runtime for QS was only slightly elevated compared to Q and QA conditions. Over 1,000 games each, Q and QA averaged 0.03 and 0.04 seconds (not statistically significant), while QS and $Dyna$ took 0.06 and 0.08 seconds, respectively.

Pursuit

The Pursuit task (also known as Chase or Predator/Prey task), was proposed by Benda et al. [Benda, 1985]. For our evaluation we use the recently evaluated instantiation of Pursuit implemented in [Brys *et al.*, 2014]. 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 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. 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.

QA was already defined by Brys et al. [2014] who implemented tile-code approximation.

QS was defined based on linear differences and angular rotations. 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 a similarity of 1 was set 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). Furthermore, experiences in the upper (left) half of the field are mirrored in the bottom (right) part by mirroring

[1]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 attempts to score a goal. While defending, the agent chases its opponent and tries to steal the ball.

states and actions. Transition similarities were defined for all state-action pairs that are expected to result in the same state.
Results: Each agent was 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 the QS condition learns significantly faster and outperforms the Q condition from the first batch onwards and the QA and $Dyna$ conditions from the third batch onwards. See Figure 2b for a graphical representation of the learning process.

On average, QS updated 12 entries per iteration. However, while Q and QA-learning agents complete their training (10,000 games each) in 8.5 seconds on average (with no significant difference between the two), QS 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.

Mario AI

We use the popular Mario AI game, often used for the evaluation of RL techniques [Karakovskiy and Togelius, 2012]. We use the recently evaluated formulation of the Mario AI task proposed by Suay et al. [2016]. 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, though 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. 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.
QA was implicitly defined by the original authors as they already abstracted the state space.
QS was defined on top of the authors' abstraction. It turns out 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, all respective states (with the four variations of these two Boolean variables) were defined 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 assumed similar to using action a+"run" (e.g., run right).
Results: Each agent was trained for 20,000 games. After each batch of 1,000 games, the learning was halted and 1,000 test games were played during which no learning occurred. The process was repeated 100 times. The two agents are also compared against human performance level as evaluated in [Suay *et al.*, 2016]. The results show that the QS-learning agent surpasses human performance level significantly faster than the QA-learning agent. See Figure 2c for a graphical representation of the learning process.

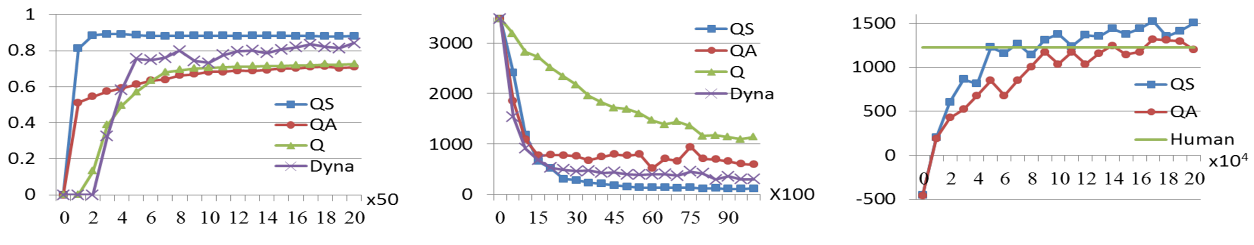
On average, the QS updated 33 entries per iteration. However, the QA requires 63.4% of QS's runtime to complete its training (20,000 games each, 33 vs. 55 seconds).

4.2 Non-Expert Study

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 designers for two RL agents. All of the students have knowledge of RL from advanced AI courses. The students are majoring in Machine Learning (7), Robotics (4) and other computational AI sub-fields (5).

Prior to the experiment, all 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 agent 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 agent 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 they are available in `http://RemovedForBlindReview`.

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 would 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. Ideally, we want participants to take as much time as they need, as the experts did. However, given that each participant had to dedicate about 3 hours for the experiment (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. 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 as used in Section 4.1. 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. In order to allow designers to compare their agents' success to a basic Q-learning agent (the benchmark agent they were requested to outperform), each designer was given a report on a basic Q-learning that was trained and tested prior to the experiment using the same procedure described above.



(a) Simple Robotic Soccer (b) Pursuit (c) Mario AI

Figure 2: The QS-learning agent outperforms QA-learning, Q-learning and $Dyna$ agents in all three domains.

After all agents were submitted, each agent was tested and received two scores: one for its average performance during its learning period and one for the asymptotic performance of the agent, i.e., its performance after the training is completed. Results: under the QS condition, participants defined similarity functions. A similarity function is beneficial if it helps the QS outperform Q-learning. Otherwise, we say that the similarity function is "flawed," in that it hinders learning.

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

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

Interestingly, *all 16 participants* submitted QS-learning agents which perform better than their submitted QA-learning agents both in terms of average learning performance and asymptotic performance. Namely, the QS agents' advantage over the QA agents is most apparent when examining each desginer separately. Furthermore, for all participants, the QS agent outperforms the QA 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 that participants defined under the QS-learning condition. This phase was done manually by the authors, 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 agents, representational similarities were instantiated, mainly using different variants of the idea presented by the expert in Section 4.1 — moving one or both of 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 were 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 brings about a significant change. Transitional similarities were only defined by 2 designers. Both designers tried to consider a more strategical approach. For example, moving towards the opponent while on defense is considered similar regardless of the initial position. It turns out that neither of the provided transitional similarities were beneficial on their own as 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 agents combined 2 notions in their similarity function. Therefore, we speculate that combining more than a single similarity notion can be useful for some 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 initially believed to be the most promising.

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 for these 4 agents was 56.9% compared to 60.8% for the basic Q-learning and their average asymptotic score was 61.5% compared to 72.5% for the basic Q-learning.

5 Conclusions

We conducted a first-of-its-kind human study that explores human aspects in speeding up RL agents. We focused on the task of injecting human knowledge into an RL learner using the notion of similarity and generalization. Our presented approach, SASS, is shown to be both effective and efficient, for both expert and non-expert designers. We hope this work will inspire other researchers to investigate their approach in human studies with actual programmers. For future work, we will tackle the challenge of 1) on-line identification and refinement of similarities with and without a designer's input, and 2) extending the proposed approach to other RL algorithms (e.g., linear function approximation and DeepRL).

References

[Albus, 1981] James Sacra Albus. *Brains, Behavior and Robotics*. McGraw-Hill, Inc., New York, NY, USA, 1981.

[Benda, 1985] Miroslav Benda. On optimal cooperation of knowledge sources. *Tech.l Report BCS-G2010-28*, 1985.

[Bruner, 1957] Jerome S Bruner. Going beyond the information given. *Contemporary approaches to cognition*, 1(1):119–160, 1957.

[Brys *et al.*, 2014] Tim Brys, Ann Nowé, Daniel Kudenko, and Matthew E Taylor. Combining multiple correlated reward and shaping signals by measuring confidence. In *AAAI*, pages 1687–1693, 2014.

[Brys *et al.*, 2015] Tim Brys, Anna Harutyunyan, Halit Bener Suay, Sonia Chernova, Matthew E Taylor, and Ann Nowé. Reinforcement learning from demonstration through shaping. In *IJCAI*, pages 3352–3358, 2015.

[Busoniu *et al.*, 2010] Lucian Busoniu, Robert Babuska, Bart De Schutter, and Damien Ernst. *Reinforcement learning and dynamic programming using function approximators*, volume 39. CRC press, 2010.

[Girgin *et al.*, 2007] Sertan Girgin, Faruk Polat, and Reda Alhajj. Positive impact of state similarity on reinforcement learning performance. *IEEE Transactions on Cybernetics*, 37(5):1256–1270, 2007.

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

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

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

[Kelly, 1955] George Kelly. *Personal construct psychology*. New York: Norton, 1955.

[Knox and Stone, 2010] W.B̃radley Knox and Peter Stone. Combining manual feedback with subsequent MDP reward signals for reinforcement learning. In *Proc. of AAMAS*, May 2010.

[Leffler *et al.*, 2007] Bethany R Leffler, Michael L Littman, and Timothy Edmunds. Efficient reinforcement learning with relocatable action models. In *AAAI*, 2007.

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

[Martins and Bianchi, 2013] Murilo Fernandes Martins and Reinaldo AC Bianchi. Heuristically-accelerated reinforcement learning: A comparative analysis of performance. In *Conference Towards Autonomous Robotic Systems*, 2013.

[Mnih *et al.*, 2015] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 2015.

[Narayanamurthy and Ravindran, 2008] Shravan Matthur Narayanamurthy and Balaraman Ravindran. On the hardness of finding symmetries in markov decision processes. In *ICML*, pages 688–695, 2008.

[Peng *et al.*, 2016] Bei Peng, James MacGlashan, Robert Loftin, Michael L Littman, David L Roberts, and Matthew E Taylor. A need for speed: Adapting agent action speed to improve task learning from non-expert humans. In *AAMAS*, pages 957–965, 2016.

[Ribeiro and Szepesvári, 1996] Carlos Ribeiro and Csaba Szepesvári. Q-learning combined with spreading: Convergence and results. In *Proc. of Intelligent and Cognitive Systems*, 1996.

[Ribeiro, 1995] Carlos HC Ribeiro. Attentional mechanisms as a strategy for generalisation in the Q-learning algorithm. In *Proceedings of ICANN*, 1995.

[Sequeira *et al.*, 2013] Pedro Sequeira, Francisco S Melo, and Ana Paiva. An associative state-space metric for learning in factored mdps. In *Portuguese Conference on Artificial Intelligence*, pages 163–174. Springer, 2013.

[Stone *et al.*, 2006] Peter Stone, Gregory Kuhlmann, Matthew E. Taylor, and Yaxin Liu. Keepaway soccer: From machine learning testbed to benchmark. In *RoboCup-2005: Robot Soccer World Cup IX*. 2006.

[Suay *et al.*, 2016] Halit Bener Suay, Tim Brys, Matthew E Taylor, and Sonia Chernova. Learning from demonstration for shaping through inverse reinforcement learning. In *AAMAS*, pages 429–437, 2016.

[Sutton and Barto, 1998] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 1998.

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

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

[Tamassia *et al.*, 2016] Marco Tamassia, Fabio Zambetta, William Raffe, Florian Mueller, and Xiaodong Li. Dynamic choice of state abstraction in q-learning. In *ECAI)*, 2016.

[Watkins, 1989] Christopher John Cornish Hellaby Watkins. *Learning from delayed rewards*. PhD thesis, University of Cambridge England, 1989.

[Witten *et al.*, 2016] Ian H Witten, Eibe Frank, Mark A Hall, and Christopher J Pal. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2016.

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