Interactive Reinforcement Learning; Two successful solutions for handling an abundance of positive feedback

Jasper S. van der Waa[a,*], Maurits Kaptein[a], Khiet Truong[a]

[a] *Radboud University Nijmegen; Artificial Intelligence Department, Netherlands*

Abstract

The field of interactive reinforcement learning focuses on creating a learning method where users can teach an agent how to solve a task by providing feedback on the agent's behavior in an intuitive way. The goal of these agents is to find a behavior that maximizes the positive feedback it receives. As users provide positive feedback for almost every step towards the task's goal, the agent learns that some set of actions result in more positive feedback than others. This set becomes a positive circuit: a set of actions and situations for which the agent learned to expect relatively much positive feedback. The problem here is that the agent will exploit a positive circuit until corrected by the user, even though the circuit may not necessarily solve the task. In this study we propose two novel solutions to this positive circuits problem. Both solutions are new in that they focus on forcing the agent to explore more actions and situations instead of simply exploiting a found positive circuit. The first solution generalizes the feedback given for an action in some situation to situations similar to that one situation. If this feedback is positive, it will motivate the agent to perform this action again, even in unknown situations. The second solution uses a method to detect any repetitive behavior and a method to detect high-risk situations likely to elicit such undesired behavior. If one of these methods triggers, the agent is forced to perform the most recent, best assessed action. Both solutions were tested individually by comparing each to a baseline agent with none of the solutions implemented. Interaction between the two solutions was tested by combining them in one agent. Tests were performed in a grid environment with a simple navigation task. The results showed that both solutions caused the agent to solve the task more often and faster than the baseline agent. The first solution also allowed the agent to learn a task solution that was optimal and effective, independent of were it started the navigation task. Finally, results showed that the forced exploration action of the second solution aids the first solutions generalization in finding such optimal task solutions. This study proved that improving exploratory behavior in an interactive reinforcement learning agent is a valid approach to solve the positive circuits problem.

Keywords: Interactive reinforcement learning, Positive ciruits problem, Human teachers, machine learning, Function approximation, feedback-based exploration

*Corresponding author

Email address: jaspervanderwaa@gmail.com (Jasper S. van der Waa)

1. Introduction

We humans behave in a way that is accepted by others, at least most of the time, otherwise we can even adapt our behavior [1, 2]. A similar skill would be very beneficial for a robot as well, because it is nearly impossible to implement a predetermined optimal behavior for every possible situation [3]. There are just too many situations and behaviors possible. This would be unnecessary if the robot could adjust its behavior based on the user's assessment of that behavior, a teaching method similar to how humans teach each other [4, 5]. We constantly provide each other with small rewards and penalties in our social interactions [1, 6]. These rewards and penalties help us define and improve our behavior. If a robot could learn in a similar way from its user, the robot could improve its interaction and even learn how to solve new tasks. Such a skill can become invaluable as robots start to play an increasingly important role in our daily lives [7, 8, 9, 10].

Reinforcement learning (RL) is the machine learning analog of human behavioral learning through rewards and penalties [6, 11]. It is a method for unsupervised learning that allows agents to make control decisions that maximize some reward [6]. In other words, RL allows an agent to learn and adjust its behavior such that it results in the most rewards. With the increasing popularity of robotics, RL agents and robots are being designed to handle user feedback as their rewards [12]. The result is a relatively new learning method, referred to as *interactive reinforcement learning*.

Interactive reinforcement learning (IRL) allows the user to specify whether the agent's behavior is desired or undesired through a reward signal. Instead of a predefined reward function, the reward is defined by the user himself during learning [13, 14]. This feedback is translated into a reward signal and used in RL algorithms to optimize the agent's future behavior. Users can consequently convey their task knowledge in a way that requires no programming skills or complex instructions [13]. In contrast to imitation learning, IRL is performed during training with an easy to use interface such as a keyboard or voice commands [15, 16, 17, 18, 19]. IRL was successfully applied in several common RL problem domains such as grid worlds, mountain car and pole balancing [3]. Besides these domains, IRL has been applied in physical and virtual robots [20, 21, 3], a prosthetic arm [22], the game Tetris [23], a cooking simulation [4], and in a chat bot [24].

The long term goal of IRL research is to develop agents that can learn complex tasks that conform with their user's teachings. The first step towards this goal is to design an agent that learns the task reliably and fast, without requiring the user to change his teaching method [3, 13].

However, users prove to be very optimistic and provide agents with more positive than negative feedback which inhibits the agent's task performance [3, 4]. A high frequency of positive feedback signals with only a few negative reward signals results in the agent learning to exploit this bias, because it wants to maximize reward. This exploitation takes the form of a repetitive behavior, as there is no better way to maximize reward than to repeat the most rewarding behavior. Until another behavior proves to be more rewarding, the behavior is repeated. The actions of such a repetitive behavior are referred to as a positive circuit. Due to the user's high frequency of positive feedback signals, an abundance of positive circuits will occur before the agent even reaches

the task's goal for the first time [4].

If the repetitive behavior is not what the user wants to teach, negative feedback is required to force the agent into a different behavior. This delays training and forces the user to adjust his teaching strategies to give less positive feedback signals [3, 4, 5]. This problem will be referred here as the Positive Circuits Problem, or the PCP in short.

Since the PCP is caused by one behavior being more rewarding than the others, the actual sum of rewards does not need to be positive. A negative reward sum will still be exploited if all other known behaviors result in even more negative sums. This makes the solution of simply subtracting a constant from the feedback signals useless, as it would leave the ratio between the reward sums intact. There is also no immediate method to determine from which signals to subtract such a constant. The solution of removing positive feedback prevents the user from signaling successful task completion. While the solution of decreasing the positive feedback signal, will make it harder for the user to correct mistakenly given negative feedback.

Several effective solutions to the PCP have been explored. One solution is to decrease the discount parameter [13]. This parameter determines how far the agent should "look into the future" when defining the reward an actions causes over time. Subsequently, the influence of future rewards on the rewards sum which the agent tries to maximize is effectively reduced. If this discount parameter is low, the agent may never become aware of the existence of a positive circuit and hence never repeat one indefinitely. However, decreasing the discount parameter also prevents the agent from learning a task solution that is effective in every possible situation and can withstand small changes to the environment [6, 3]. Another solution is to update all or multiple reward predictions for every possible situation [25, 3]. This allows the agent to learn that some actions result in higher rewards than others. The agent is then more likely to learn the task, as completing the task is assumed to result in the highest rewards possible. However, this solution is not suitable for domains with a lot of possible situations or when the agent does not know all possible situations. Another tested solution to PCP is to only use IRL agents in continuous domains (domains without explicit learning episodes) [13]. This is a promising solution, especially combined with a RL algorithm that updates all or several locally biased reward predictions. These solutions will be discussed in greater detail in Section 2.

The current study proposes two new solutions to the PCP to improve an IRL agent's task performance. These two solutions differ from existing solutions in that they require no special learning conditions such as low discount parameters or only continuous task domains. Nor do they require any prior knowledge about the task domain such as all possible situations the agent can come across. Instead, both proposed solutions treat the PCP as an imbalance between the exploitation of learned reward sums and the exploration of new sums. In doing so, they try to motivate the agent to explore more promising actions in terms of reward, instead of exploiting a found positive circuit.

The first proposed solution aims to provide this motivation by generalizing received feedback to similar situations. The purpose of this generalization is to prevent the agent from learning the presence of positive

circuits. The second proposed solution forces the agent to explore the consequences of an action when a positive circuit is about to be exploited or is already being exploited. This action is selected based on the user's feedback, making it easier for the user to fix and avoid repetitive behavior. The first solution will from here on be referred to as the preventive solution and the second as the symptom solution.

The performed experiments test the effect of the two solutions on the agent's task performance. As a baseline we used a state-of-the-art IRL agent developed and tested in previous studies and based on the TAMER framework [13]. The baseline agent functioned as a baseline that we could compare the effects of our proposed solutions to. We chose this agent as it proved to be a reliable and an effective way to implement an IRL agent [3, 13]. Also, it allowed us to test whether the proposed solution can further improve state-of-the-art agents in IRL. The second and third agent tested in this study, each have one of the proposed solutions implemented. One with the preventive solution and one agent with the symptom solution. A fourth agent was constructed with both of the solutions implemented. This was possible because the preventive solution targets a change in learning while the symptom solution targets a change in the way of selecting an action. The combination of the two solutions in one agent allowed us to measure the interaction between the solutions.

We hypothesize that the agent with the preventive solution shows no or little repetitive behavior as it will not learn the presence of any positive circuit. From the agent with the symptom solution we expect that it will show less repetitive behavior than the baseline agents. This will be reflected in an increased task performance as the agent spends less time with such undesired behaviors and explores and learns the task solution instead. Finally, we hypothesize, that all four agents are able to learn a task solution that can solve the task from any situation due to a high discount parameter, even when the environment is slightly changed.

The main contribution of this study is to construct and analyze exploration as new approach to the positive circuits problem. The results of this study can prove whether exploration is a viable method to reduce the undesired exploitation of positive circuits and as such increase task performance. Also, the proposed solutions are not limited to small problems as previously proposed solutions were and can be implemented in robots and real-world problems without much effort. Finally, this study further investigates the positive circuits problem and the results from this can be used to develop new solutions to the problem.

This article is structured as follows. Section 2 provides the background and definitions in reinforcement learning and interactive reinforcement learning needed to understand the proposed solutions. This section also explains the positive circuits problem in detail. Section 3 discusses the implementation of the used IRL agent and the two proposed solutions in detail. Section 4 describes the performed experiment. Section 5 provides the results on the measurements that show that the two proposed solutions sucessfully solve the PCP and increase task performance. Section 6 discusses the results as well as possible improvements to the proposed solutions. Finally, the study is summarized in Section 7.

2. Background and definitions

This section begins with a brief description of reinforcement learning with a focus on the terms important for this study. For a more detailed description of reinforcement learning, we refer to the supplementary materials. Interactive reinforcement learning is discussed in Section 2.2. The Positive Circuits Problem (PCP) is discussed in detail in Section 2.3 and Section 2.4 discusses the existing solutions to the PCP.

2.1. Reinforcement Learning

Section 1 introduced reinforcement learning as the machine learning analog of behavioral learning through rewards and penalties. This section briefly explains the concept of reinforcement learning and introduces several important definitions used in this study.

Reinforcement learning algorithms learn to perform actions that maximize a sum of rewards over time [6]. Problems in reinforcement learning are often described as Markov Decision Processes (MDPs), denoted as $\{S, A, T, R, \gamma\}$. The S and A are the sets of possible states and actions. The T is the transition function that describes the probability that the agent transitions from one state to another if a specific action is performed, such that $T(s_t, a_t, s_{t+1} = P(s_{t+1} \mid s_t, a_t)$. The R is the reward function; $R : S \times A \times S \rightarrow \mathbb{R}$. The reward function provides the agent with a reward for every state transition. The γ is the discount factor that controls the amount of valued future reward.

RL algorithms learn a *policy* [6]. A policy determines the action that needs to be performed in each state. Only deterministic policies are used in this study, which map one action per state; $\pi : S \rightarrow A$. The algorithm's goal is to maximize the sum of discounted rewards over time; the *return*. This return is expressed as $Q^\pi(s, a)$. The return is defined as the sum of a specific discounted action sequence starting in s and with action a while following π afterwards:

$$Q^\pi(s, a) = \sum_{t=0}^{\infty} E_\pi \left[\gamma^t R(s_t, a_t, s_{t+1})\right] \tag{1}$$

The Q function, the action-value function, provides the agent with state-action values. A state-action value, is the value of the return the agent expects to receive after performing a in s. These expectations are learned and based on the actual experienced rewards. The purpose of a RL algorithm is to update Q such that it provides state-action values that match the return with greater accuracy. An optimal Q provides the agent with the exact return as defined by Eq. 1.

From Q, a policy can be derived by choosing the action with the largest state-action value; $\pi(s) = maxarg_a Q(s, a)$. Such a policy can be followed to immediately exploit the found expected –by definition– highest sum of rewards, but some exploration is necessary. Some explorations allow the agent to experience the learned state-action values and whether these are indeed correct, in some cases the Q may be indeed inaccurate and a different policy may prove to be even better. A widely used and accepted exploration method is the ϵ-greedy action selection mechanism [6]. This method is also used in this study and means that the agent will

choose a random action with ϵ probability and with $1 - \epsilon$ probability the action from π. In this study the ϵ is kept constant over time, this means that even when Q and its derived policy π are optimal, there is always an ϵ probability that the agent chooses a different action. This ϵ-greedy action selection facilitates some (naive) exploration as it allows the agent to sample or experience different rewards than stated by π. If the agent would have infinite time to learn, this action selection mechanism would cause the agent to learn a fully accurate Q and derive an optimal policy π from it [6].

The discount factor plays an important role in reinforcement learning [6]. This parameter defines with how much the future rewards are valued. A discount factor of zero, $\gamma = 0$, creates a *myopic agent* that only values the immediate reward. A discount factor of one, $\gamma = 1$, creates a *non-myopic agent* that values future rewards equally as the immediate reward. This results in infinite returns, hence γ is limited to the range $[0; 1)$ [6].

A high discount factor, for example $\gamma = 0.9$, allows the RL algorithm to define a policy that aims to collect as much current and future reward as possible. However, rewards in the distant future are weighted less important than more immediate rewards. A high discount factor provides the agent with a general understanding of how actions affect return. In other words; a high discount factor allows the agent to better understand how actions affect its capability to solve the task that is described in terms of rewards.

2.2. Interactive Reinforcement Learning

Section 1 explained *interactive* reinforcement learning (IRL) as a method to teach a new behavior to an agent based on user feedback that assesses the agent's current behavior. This section briefly discusses IRL, for a complete overview of the possibilities of IRL we refer to Knox [3].

Interactive reinforcement learning is a method based on RL that allows a user to interactively shape the agent's behavior. The behavior takes the form of a policy and the experienced rewards are based on user feedback that assesses the actions performed according to the policy. The most straightforward method to create an IRL agent is to substitute the reward function R with the user feedback F. This substitution is done in several studies, with success [4, 20, 22, 26].

However, there are several issues with this approach that distinct it from regular RL. User feedback is not static as the rewards from a predefined reward function. User feedback can suffer from a delay due to human reaction time when actions are performed in quick succession [3, 22]. Also, feedback may not be given for every state-action pair [4]. Finally, multiple users have different teaching strategies resulting in different feedback values for the same state-action pairs [4]. A simple substitution of R with F may cause sub-optimal learning and task performance due these issues.

Although several studies proved that this naive substitution is a valid approach to implement IRL, other studies showed that learning performance can be improved. Two of such improvements are the modeling of F and treating F as a delayed reward [3, 22]. The study done by Pilarski et al. [22] distributed received feedback values over several past state-action pairs using a decay approach. This was necessary because the robotic-arm,

their agent, chose actions in the order of milliseconds and users provided feedback in the order of seconds. For similar reasons Knox [3] distributed feedback values over past state-action pairs using a credit assignment method based on the probability mass of a predefined probability distribution over past time. In addition this study by Knox [3] modeled these distributed feedback values with a regression model [27]. The rewards for a state-action pair were modeled into one reward value which acted as a moving average. This created a more static reward signal for state-action pairs compared to simply using F as the reward function. The moving average effect maintained some of the adaptivity of user feedback that is so useful to an IRL agent [5].

The modeling of feedback values into a more static reward reduces the violation of a non-static reward for state-action pairs. This assumption of static rewards applies for most RL algorithms used in IRL: An IRL agent learns to predict the return of performing some action in a state, if this return keeps changing it is not guaranteed that the RL agent will converge into any policy [6]. An accurately learned model can also provide the agent with meaningful rewards even when no feedback is given. However, as we will discuss in Section 2.3, the modeling and distribution of the feedback values makes the agent more susceptible to the positive circuits problem.

Another approach to IRL is to combine feedback with a predefined reward function; the agent receives rewards from feedback values F and the predefined R [28, 29, 30]. This allows the designer of the agent to implement a negative or positive reward for states that should either be avoided or reached. The user can still teach the agent some behavior, but the agent will try to learn a trade off between what the user wants the agent to learn and what it should learn according to the agent's designer.

The above described methods on how to use feedback signals as reward signals, focus on adjusting RL algorithms such that an IRL agent can learn the task. These methods only implicitly deal with the general problem of IRL; how to deal with the inconsistent, biased and ever-changing reward signal coming from a human user. This study has a similar focus but we do point out that the field of IRL is closely related to the field of Contextual Multi-armed Bandit problems (CMAB). Both fields try to learn decisions or actions that maximize some sum of rewards and both deal with non-static rewards. The main difference between the two fields is that IRL also deals with human-machine interaction issues, but besides this the learning goals and problem descriptions are generally the same.

The relatively older field of CMAB problems can offer a source of methods on how to deal with user feedback to the new field of IRL. For example, an IRL agent can learn a probabilistic model for state-action values given the state-action pair instead of a point estimate. Such methods can be used to create effective exploration methods. For example an action selection mechanism based on some uncertainty score based on the learned probabilistic method. If such a model for a state-action value has a lot of uncertainty, it may be useful to explore this action further to learn a more accurate estimation of the expected return.

2.3. The positive circuits problem

The positive circuits problem (PCP) was first identified by Knox [3]. This is the problem when an agent learns to exploit the positive bias on feedback by performing an undesired repetitive behavior. A sequence of state-action pairs that results in a higher return than other sequences is referred to as a positive circuit. Short circuits with high returns are present during almost every step towards the task's goal, because of the user's tendency to provide positive feedback for every of those steps. Since user feedback is also variable and not given for every action, some circuits result in higher returns than others and create positive circuits.

The exploitation of such a positive circuit takes the form of repeatedly performing the actions of that circuit. This repetition allows the agent to collect the learned expected return as often as the actions are repeated. This repetitive behavior can only be stopped if the expected return is reduced as it will motivate the agent to perform a different behavior. This can be done by the user by providing enough negative feedback to force the agent to adjust its expectations to a lower return for the circuit it currently exploits. In other words; the agent will exploit one positive circuit until its return is lowered and another circuit becomes more positive, only when the found positive circuit solves the task the user can stop correcting the agent. The necessity of these corrections forces the user to adjust his teaching method. Also, these corrections lengthen the time an agent needs to learn the task and as such decrease the agent's task performance. These corrections may even cause the agent to learn some strange behavior in those states, which the user may need to correct again when the agent comes across those states again. This is the problem of positive circuits; they reduce task performance and force the user to adjust its teaching method.

Although the PCP can thus be corrected by the user himself, it requires him to adjust his teaching method and decreases the task performance. However, high discount parameters reduce the users capacity of actually fixing the PCP while high discount parameters are required to create a non-myopic agent as discussed in Section 2.1. A high discount parameter will cause the agent to weigh future rewards almost the same as current rewards. This makes it difficult for the user to stop the repetitive behavior as a few negative feedback signals are unlikely to outweigh the expected high return. Since this return includes many future (positive) rewards which are weighted almost as much as those current few negative rewards. The user will need to provide more negative feedback signals to stop the repetitive behavior. The user may even need to provide negative feedback to the desirable actions that caused this positive circuit in the first place. This will increase the risk of the agent learning a strange and sub-optimal behavior in those states that will only be discovered if the agent experiences those states again.

Recently developed improvements for IRL agents also reduce the users capacity of actually fixing the PCP. One of these is the modeling of feedback values to create more robust reward and the distribution of feedback over multiple past state-action pairs. A reward model will reduce the impact of the provided negative feedback because the model will most likely learn the positive bias on feedback as well. If this is the case then the positive bias will also be applied to negative feedback values, which reduces the impact on the agent's expected

high return. Another improvement to IRL agents that reduces the users capacity for fixing the PCP himself is the distribution of feedback over past state-action pairs. This improvement may cause negative feedback to be applied incorrectly. The model behind this distribution may prove to be correct under normal circumstances, but as behavior is repeated the user learns to provide feedback more accurately making the distribution unnecessary and perhaps even undesired. Both of these improvements aim to make an IRL agent that can handle the variable and inconsistently given feedback, but make the agent also more susceptible to the effects of the PCP.

2.4. Existing solutions to the Positive Circuits Problem

Several promising solutions to the PCP are already explored in different studies. One solution is to lower the discount parameter. This creates a more myopic agent that learns less positive circuits at it only uses the most immediate rewards to base its action selection on. If these rewards would be negative the actions that cause these are avoided, even though they would result in high rewards later on. This also allows the user to stop any repetitive behavior more easily as discussed in Section 2.3. However, it forces the user to adjust his teaching method and prevents the now myopic agent from learning a more robust task solution as discussed in Section 2.1.

Another solution to the positive circuits problem is to apply IRL only to continuous MDPs as the problem was assumed to be caused by the episodic nature of a task [13]. In episodic MDPs, the end state would force the agent away from the states that provide positive reward, whether this is the goal or a positive circuit. In continuous MDPs there is no state that returns the agent back to its starting position. The method proposed by Knox and Stone [13] changed an episodic MDP to a continuous MDP. This was done by assigning the state features of the starting state to the end state. This makes the MDP circular for the agent and creates a positive circuit that can be exploited.

Whereas, the continuous MDP solution proved to be effective, it was not optimal as some users were still not able to teach the agent the task [13]. Also, this solution does not prevent the positive circuits problem nor its main symptom; undesired repetitive behavior. The IRL agent might still learn short positive circuits before it finds the actual task solution. Hence, the prevention of PCP or its main symptom may further improve the task performance.

3. The implementation of the agent

This section discusses the implementation of the four agent types. This includes an explanation of the two proposed solutions.

3.1. Human feedback to rewards; TAMER Framework

All the implemented agents in this study distribute and model the user feedback similarly as the TAMER framework [3]. This means that;

1. Our agents can receive feedback values at any time of any value;
2. Agents distribute these values over past state-action pairs based on a probability distribution over those pairs;
3. Agents learns a reward model from these distributed values;
4. The output of the reward model is used as the reward for the RL algorithm the agent uses.

With this approach, our agents benefit from the advantages of distributing and modeling user feedback as discussed in Section 2.2. It also allows us to test the two proposed solutions on state of the art IRL agents. Also, as mentioned in Section 2.3, the distribution and modeling of feedback creates an agent more susceptible to the effects of the PCP. If the two proposed solutions prove to be effective for agents that use such improvements, it is likely that they also apply to the more simple IRL agents.

In this study, the distribution of the feedback signal was done according to a uniform probability distribution over the past state-action pairs, similar as in the TAMER framework [3]. The range of this distribution was adjusted to target all state-action pairs between 0.2 and 0.8 seconds into the past. The offset of 0.2 seconds was based on the average response time for humans in visual tasks [31]. The time window of 0.6 seconds was chosen such that the distribution could target at most two (partial) state-action pairs. See Figure 1 for an example of this feedback distribution.

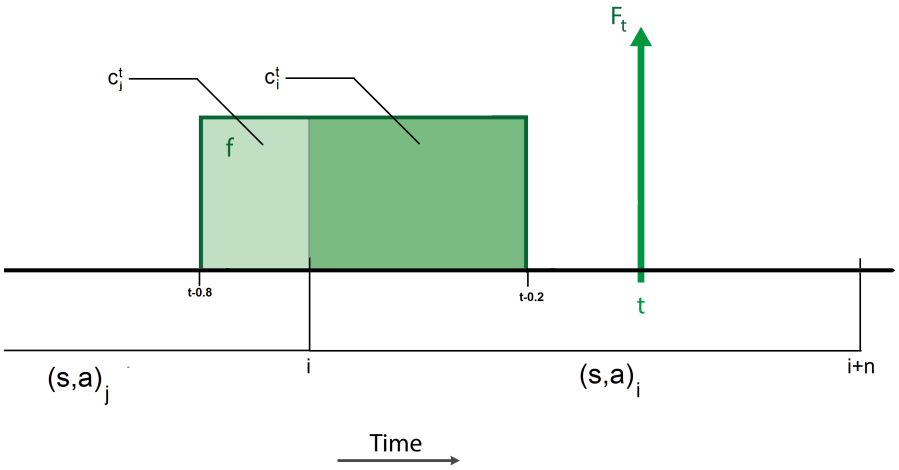


Figure 1: The uniform probability density function over past state-action pairs. This function was used by the implemented agents to distribute feedback signals as proposed in the TAMER framework [3]. The figure shows two state-action pairs; $(s,a)_j$ and $(s,a)_i$ with i and j as their starting times and a discrete time-step size of $n = 0.8$. At time t the user provided a feedback signal F_t. The probability distribution is used to calculate the amount of credit each of the two state-action pairs receives. These credits, c_j^t and c_i^t are the surfaces of the overlapping time period each state-action pair has with the uniform probability density function.

The distributed feedback values were modeled by the same model used in the TAMER framework. The model used incremental gradient descent –a simple single perceptron– to learn an average feedback value for

each state-action pair. These modeled values were used as the reward signals for the RL algorithm that learned the state-action values. The learning rate of the incremental gradient descent algorithm was set to 0.05 and the same state features were used as those discussed in Section 4.1.

3.2. First solution; Prevention through generalization

The first proposed solution to the positive circuits problem tries to prevent the agent from learning the presence of positive circuits and instead learn what actions are overall the best in which sets of states. This section explains the general idea behind this solution and the used algorithm to implement the solution. We refer to the supplementary materials for an in depth explanation of this algorithm.

Most studies in interactive reinforcement learning used an RL algorithms based on look-up tables [3, 5, 21, 23, 24, 32]. Look-up tables allow the IRL agent to store a state-action value with perfect accuracy and retrieve it when needed. However, in such a look-up table all never experienced and unfamiliar state-action pairs and their values remain zero or some other initial value. This initial value is only changed when the agent experiences the state-action pair and all state-action values are treated as fully accurate unless proven otherwise by such an experience.

These unfamiliar state-action pairs, with their initial or inaccurate values, inhibit the agent's exploration of these very same state-action pairs. The reason for this is that the familiar state-action pairs have a high reward due to the positive bias on user feedback. The problem is that the initial or inaccurate values of unfamiliar state-action pairs do not outweigh the high return of the familiar state-action pairs that form a positive circuit. Hence, the agent has no incentive to explore these unfamiliar state-action pairs.

The first solution we propose in this study tries to generalize the experienced return of a state-action pair to other state-action pairs with the same action and as a function of the similarity between states. This reduces the number of inaccurate values of the unfamiliar state-action pairs without the need for the agent to actually experience these unfamiliar pairs first. A well assessed action in one state would also become a profitable action in similar states, and thus provide an incentive for the agent to explore these actions and improve their predictions further.

The generalization of experienced return is done with function approximation [6, 33]. The look-up table of Q is approximated by a linear function of state features and a set of adjustable weights. Function approximation allows the agent to store the look-up table as a linear function. The agent updates state-action values by adjusting the weights to reduce some error. Weights can be shared over several or all state-action pairs to facilitate generalization. For example, if one weight of a state-feature is shared by all state-action pairs, increasing this weight results in increasing all state-action values that contains that specific feature.

In this proposed solution, weights are shared by all state-action pairs with that have the same action. This means that each possible action has its own unique set of weights. This allows the agent to generalize a value update over all other state-action values with the same actions, where the equality between states determines

how a value is updated [33]. In practice, this approach allows the agent to learn what return can be expected in what kind of states when performing a specific action. This as opposed to an update in a look-up table which is simply the update of one specific state-action value.

To illustrate this solution imagine a non-myopic agent that needs to navigate across a hallway and starts at one side. If the agent moves forward, it's user provides positive feedback as this is a desired action. If the agent moves backwards, negative feedback is provided. In the case of the look-up table, the agent will learn that a forward step from its starting position results in positive feedback. This knowledge is not generalized to other forward steps in different states. Since the non-myopic agent does not know what kind of reward it will receive if it goes further, the agent steps backwards such that it can step forward again. This positive circuit is repeated as it results in positive infinite return as long as the positive feedback outweighs the negative feedback. By generalizing the positive feedback for a forward step to state-action values with states similar to the starting state, the agent will have some idea of the return it might expect in other –never experienced– states. In this case, the starting state and the agent's state after one forward step, are quite similar as they are close to each other. This means that the agent has some incentive to step forwards again, due to this generalization. Since the agent never stepped backwards, the value for that action in any state is still some initial value and is outweighed by the generalized value for stepping forwards. As states become more distant from the starting state, the generalized value decreases until it does not outweigh the initial value of other actions any longer. The behavior we then perceive is an agent that keeps stepping forward. If the user provides negative feedback the generalized forward step can be interrupted. As opposed to any positive feedback from the user results that will strengthen this forwards behavior. This kind of behavior is much more in line with what the user expects than the repetitive behavior of moving forwards and backwards again if the agent would use a look-up table without generalization.

The function approximation technique cannot be simply added to *direct* RL algorithms such as Q-Learning or Value Iteration [33]. These algorithms are designed to directly update one state-action value. If these methods would be used to to update the shared linear weights, any convergence guarantee is lost [33]. Direct algorithms define the error of a predicted return as the difference between the experienced reward plus the predicted future return and the actually predicted current return. If this error would be used to update the linear weights, all state-action values with the same action are affected. This naive update causes divergence in all updated state-action values [33].

The RL algorithm we propose to perform the generalization is the TDC Learning algorithm [34]. This algorithm incorporates the fact that changing the shared weights affects more than one state-action value. A direct algorithm would simply update the weights to improve the accuracy of the current state-action value. The TDC Learning algorithm updates the weights just enough to improve the current state-action value while minimizing the impact on other predictions. This allows the TDC Learning algorithm to converge to the optimal function approximation solution without a learning speed to direct RL algorithms [34].

We hypothesize that the agent does not learn the presence of a positive circuit while generalizing its state-action value updates and, as a result, learns the task faster. The generalization will provide the agent with an incentive to continue exploring the well assessed actions, even in unfamiliar states. In contrast to this, an agent without generalization will find a positive circuit in every set of familiar states that causes a high return due to the user's positive bias on feedback. Then the positive circuit is repeated because the state-action values of familiar states will outweigh the initial value of the unfamiliar state-action pairs.

3.3. Second solution; A symptom fix through user guided exploration

The second proposed solution tries to prevent the *repetitive behavior* and helps the user fix it when it occurs. Note that unlike the first solution this solution does aim to prevent the agent from finding a positive circuit. The solution tries to limit the behavioral symptom when the agent does find a positive circuit; an undesired repetitive behavior that exploits the positive circuit.

This solution consists of two parts to limit the repetitive behavior and improve task performance. The first part tries to prevent the agent from returning to a positive circuit simply because an action to a more unfamiliar state proved not to give the expected reward. The second part tries to detect when the agent repeats some undesired behavior and tries to fix it. In both cases the learned action is ignored and an exploratory action is chosen based on recently given feedback. The proposed solution is a combination of these two solutions to the repetitive behavior because both are merely partial solutions. Preventing the agent from returning to a positive circuit because the state-action value in some unfamiliar state is not accurate, will not prevent the agent from actually finding and exploiting a positive circuit. This first part will only motivate the agent to continue exploring unfamiliar states. Also, the second part only provides a method to stop the repetitive behavior and does not prevent the agent from returning to it again after the exploratory action that broke the repetition.

The first part of this solution tries to *prevent* any repetitive behavior by ignoring the learned policy in unfamiliar states. Instead, the agent chooses an exploratory action based on the recently received feedback. This allows the agent to explore state-action pairs even when its learned policy tells the agent to return and exploit a positive circuit. This, we hypothesize, helps to prevent the agent from exploiting a positive circuit and instead explore state-action pairs with a possible, but unknown, high return. Here we classify an unfamiliar state with perhaps inaccurate valued actions by a simple thresholding method. If the agent experienced some state less than a set number of times, the state is classified as unfamiliar. This threshold is denoted here as k.

If the repetitive behavior does occur, the second part of this solution tries to *fix* the repetition by ignoring the learned policy. Instead, the agent starts to explore based on the user feedback. Since the learned policy tells the agent to repeat a positive circuit, ignoring the policy briefly may break the repetition. The feedback-based exploration in such a case causes the agent to even break the repetition with a sensible action. To do this, the agent needs to detect its own repetitive behavior. In this solution the agent keeps track of a short-term memory, referred to as M, that contains recently experienced state-action pairs and the feedback values distributed to

these pairs. The number of pairs stored in M is denoted by the parameter n. If the agent's current state was experienced some specific number of times in its short-term memory, the learned policy is classified as repetitive. This threshold is denoted with the parameter l.

For both the prevention and fix of the repetitive behavior the learned policy is ignored and an action is chosen based on the user's feedback. This action is chosen based on a probability distribution that favors actions if they were recently assessed with better feedback as compared to the other performed actions. The distribution disfavors actions that were assessed with the worst feedback compared to the other performed actions. All possible actions that were not performed recently or assessed by the user (not in M), receive an equal probability. These recent actions and states are retrieved from the agent's short term memory M. The selected action is referred in this study as the "feedback-based action". Note that the feedback-based action is based on the (distributed) feedback values and not on the modeled reward. This allows the second solution to quickly react to the user without the reward model interfering with its learned positive bias and slowly changing output.

The selection of the feedback-based action allows the agent to perform a sensible action with respect to the user feedback. When repetitive behavior is detected, the agent will, instead of continuing the repetition, select the action favored the most by the user independent of the state the agent is in. Even if this action is equal to the learned action, the feedback-based action selection will trigger again since another –also often experienced– state is experienced again.

If the agent comes across an unfamiliar state it is likely that the agent has not yet learned the action desired by the user and that it will return to more familiar states to start to exploit a positive circuit (as discussed in Section 3.2). This returning behavior is not beneficial for learning, as it inhibits the exploration necessary for the agent to find the policy that solves the task. Instead of choosing the learned action and to facilitate exploration, a feedback-based action is selected. This allows the agent to explore the actions that were recently well assessed by the user. The feedback-based action only favors the well assessed actions, other actions can still be chosen. This allows the agent to still sample the returns of other actions independent on the set parameters k, l and n that determine how often a feedback-based action is chosen.

It is important for this solution that the parameters k, l and n are reasonable values given the task domain. The parameter k determines how often the feedback-based action is used as an exploratory action in unfamiliar states. The parameter l determines when a set of actions is repeatedly performed and thus an undesired repetitive behavior. Finally, the parameter n defines the size of the short-term memory that is used to determine both repetitive behavior and the feedback-based action itself. If the l is relatively high compared to the n, as in this study, a behavior is only repetitive when it performs a small set of actions in quick succession with little variation in its experienced states.

The novel method used to select the feedback-based exploratory action uses the feedback of the user to create a probability mass function over all actions. The agent's short-term memory M and a constant c that

denote the minimum probability for every action function as parameters for this probability distribution. This distribution is constructed each time the two parts of the solutions trigger an exploratory action, as M changed. This probability is based on an action-score that determines how well each action was assessed relative to each other according to M. This action-score, denoted as $\mu(a)$ for action a, is calculated as follows;

$$\mu(a) = \frac{1}{\mid M(a) \mid} \sum_{F(s,a) \in M(a)} F(s,a) \qquad \text{if } \mid M(a) \mid \neq 0 \text{ and } F(s,a) \in \mathbb{R} \tag{2}$$

$$\tilde{\mu}(a) = \frac{\mu(a)}{\mid \sum_{a' \in A} \mu(a') \mid} \tag{3}$$

Equation (2) calculates the average reward for action a based on the recently received user feedback values. The $M(a)$ is the set of all rewards stored in the short-term memory M caused by the action a. More formally; $M(a) = \{(\ldots, a') \in M \mid a' = a\}$, with $\mid M(a) \mid$ as the number of state-action pairs in M with action a. The $F(s,a)$ is the feedback value distributed to the state-action pair (s,a) which are stored in M. If a is not performed in the agent's short-term memory, the average received feedback for a is zero.

Equation (3) normalizes the average feedback for every action into an *action value*. The range of an action value $\tilde{\mu}(a)$ is $[-1;\ 1]$. An action has a value of -1 or 1 when it is the only assessed action in M that caused respectively a negative or positive reward, all other action values are zero. If multiple actions are assessed and performed these $\tilde{\mu}(a)$ for each action a represent their relative score with respect to the reward they caused on average in M. For example when action a_1 and a_2 have the following values; $\tilde{\mu}(a_1) = -0.5$ and $\tilde{\mu}(a_2) = 0.25$, then a_1 caused an average negative reward twice as large as the average positive reward caused by a_1.

These calculated action-scores are then used to construct a probability distribution over all possible actions. The variable for this distribution is the action set A and its outcome is the probability that some action a will be chosen as the feedback-based action. The distribution takes the parameter M to determine the action-score for a and a constant $c \in \mathbb{R}$ that defines the minimum probability for a. The distribution is constructed with the following equation;

$$P(a \mid M, c) = \frac{1}{Z} \left(\frac{1}{\mid A_{possible} \mid} + \frac{\tilde{\mu}(a)}{\mid A_{possible} \mid} + c \right) \tag{4}$$

$$\text{where } Z = \sum_{a' \in A_{possible}} \frac{1}{\mid A_{possible} \mid} + \frac{\tilde{\mu}(a)}{\mid A_{possible} \mid} + c$$

The Equation (4) uses the action values to construct a probability distribution. This distribution is constructed such that each action starts out with a probability of $1/\mid A_{possible} \mid$, where $\mid A_{possible} \mid$ is the number

of all possible actions in the agent's current state. In this study the maximum number of actions will be 4, but in most states only 2 or 3 actions will be possible. In other words, each action starts out with its uniform probability based on the current number of possible actions. These uniform probabilities are adjusted such that an action with a positive action value has a higher probability, up to twice the uniform probability plus c. This step also lowers the uniform probability of an action with a negative action value, to a minimum of c.

The normalization constant $\mathbb{Z}$ ensures that the probability distribution is a correct one. It multiplies the adjusted uniform probability of a with the inverse of the sum of all adjusted uniform probabilities. This assures that $\sum_{a \in A} P(a \mid M, c) = 1$ independent of M or c.

This parameter c ensures that even the worst assessed action has a small probability of being selected. This parameter determines how "flat" the resulting probability mass function will be. The higher c is, the less difference there will be between the best and worst assessed actions although there will always be a small probability. In this study $c = 0.05$ was used, which flattened the probability mass function only little. This choice allowed an action probability between $1/60$ and $1/2$ for an action, depending on the number of possible actions, how many were assessed and how well these assessments were. The designer of an IRL agent can determine which c is best suited for his agents, just as in deciding the ϵ for the ϵ-greedy action selection mechanism.

We hypothesize that this solution will reduce the time the agent spends in repeating a positive circuit. This does not necessarily mean the agent will learn how to solve the task faster than an agent without the solution. Therefore the agent with this solution selects a feedback-based action to facilitate the exploration of state-action pairs with the highest potential according to the user's feedback. Since the user feedback affects the probabilities for actions, exploratory action is implicitly determined by the user himself; his past and current feedback signals determine the future exploration to prevent and fix any repetitive behavior. With this user-guided exploration to prevent and fix repetitive behavior caused by positive circuits, we hypothesize that the agent will indeed learn the task faster.

4. The experiment

The effect of the two proposed solutions on the agent's performance was tested in a 2×2 between subject design. The two solutions in were the binary solution factors Preventive and Symptom denoting whether the agent used the respective first and second proposed solution. Table 1 shows an overview of the experimental design and the agents created by the two factors.

Several settings were needed for the algorithms and solutions of the agents in each of the four conditions. The learning rate of the Q-Learning algorithm was set to 0.05 and the two learning rates for the TDC Learning algorithm was set to 0.05 and 0.0125 for its respective primary and secondary set of weights. The learning rate of the reward model, using the Incremental Gradient Descent algorithm, was set to 0.05. The ϵ for the ϵ-greedy action selection of every learning algorithm was set to 0.1. Note that the symptom solution could override this exploration behavior by the feedback-based action. The symptom solution had several parameters as well. The

	Without preventive solution	**With preventive solution**
Without symptom solution	**Baseline**: Q-Learning algorithm	**Preventive agents**: State-action value generalization
With symptom solution	**Symptom**: Q-Learning algorithm with repetitive behavior fix and prevention	**Combined**: State-action value generalization with repetitive behavior fix and prevention

Table 1: An overview of the experimental design. The two factors of the 2×2 between subject design determine which of the two proposed solutions are implemented in the agents.

threshold to determine unfamiliar states k was set to 5. The threshold for a repetitively experienced state l was set to 5 as well. The size of the agent's short-term memory referred to as n was set to 9. With these parameters a behavior was classified as repetitive if a state was experienced five times out of ten (the nine previous state-action pairs and the current pair). Finally the constant c of the symptom solution was set to 0.05. All agents had a discrete time-step size of 0.8 seconds. This created a relatively fast moving agent to prevent the user from trying to (successfully) assess every action. Finally the discount parameter was set to 0.9 in all agents, making them non-myopic.

The following sections discuss the performed experiment, the problem domain and the general workings of the agent. This section concludes with an introduction to the used measurements and performed analysis. Section 5 shows the main results for each performed measurement and Section 6 provides a complete discussion of these results.

4.1. Experiment and task description

Participants could download the experiment from a website. On the website participants could read some general information about this study. The task was to teach an agent called Peter to find the exit in a grid-based maze. The first steps of the experiment explained and introduced the grid world, how Peter navigated the maze and what the user should teach and how to teach. After this step-wise explanation the participant had to go through an interactive demonstration. A detailed overview of these experimental steps can be found in the supplementary materials.

After the explanation and the demonstration, a short summary the experiment was provided and the actual task started. When the task was completed a short questionnaire followed after which all data was processed and send to the server.

The participant's task was to teach the agent how to reach the blue position in the maze for five times within seven minutes. Figure 2 shows this maze, the blue goal position and the agent at its start position. If the agent reached the goal position it was reset to its original starting position at $(0, 0)$. This task is episodic and chosen intentionally as the PCP occurs mostly in episodic tasks [13]. If the agent reached the goal position for the fifth time, the task ended and a questionnaire was presented. If the user was unable to teach the agent how to reach the goal position for five times the task would end if the seven minutes had passed.

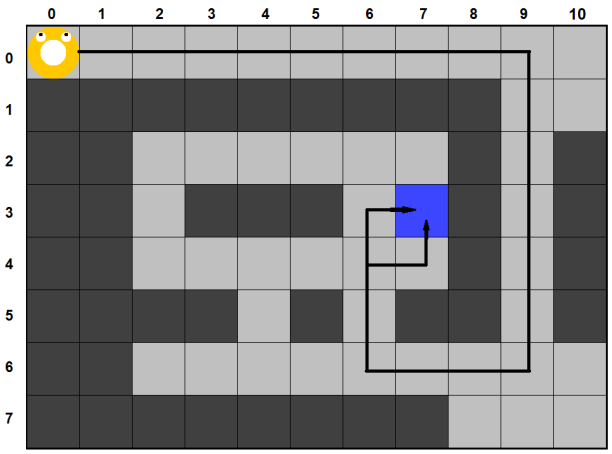


Figure 2: The maze used in the experiment. States were described as maze positions and their coordinates. The agent is shown at its starting position and the goal position was marked in blue. The black arrow shows the shortest paths to the goal from the start. The eyes of the agent pointed towards the direction of the last performed action.

In this task the agent's environment was a grid world of size 11×8. The set of states, S, consisted of all grid positions with the exception of the obstructed positions due to the maze's walls. The set of all actions, A, consisted of all horizontal and vertical movement steps; $NORTH, EAST, SOUTH, WEST$. The starting position of the agent was always at $(0, 0)$. The goal position was marked as blue for the participant, however this position did not have any additional state features for the agent.

The agent's state feature vector consisted of Radial Basis Function (RBF) values [35]. These RBFs were evenly distributed over the grid world, 40 for each dimension for a total of 1600. Every RBF had a two dimensional Gaussian form the coordinates of the RBF as its means and standard deviations of 0.8. Most features were near zero except those with RBFs close to the agent's center. These RBF features allowed us to define state similarity as the distance between states. This method also provided enough features, and thus weights, for the linear regression methods that were used to learn predictions for feedback values (by the reward model) or state-action value (by the preventive solution).

The Q-Learning algorithm used a look-up table for every possible s in S. The reward model and TDC Learning algorithm both used the RBF feature vector. Most of the 1600 RBF values were small but several would be near 1.0. These high state features were shared by neighboring states. The large size of the feature

vector also gave the algorithms the freedom to define the function they had to learn. As mentioned, the agents used one weight set per possible action. All agents used a reward model that used four sets of weights; one for each action. The agents of the third and fourth type also used the TDC Learning algorithm that used its own four sets of weights, one for each action. In addition to the RBFs features each of the weights also had a bias feature of 0.1 to help model the positive bias on feedback and rewards.

The participant could interact with the agent and the task in several ways. The teaching itself was performed through the use of two buttons on a standard US keyboard; 'z' and '/', for negative and positive feedback respectively. The feedback values were either 1 or -1 before being distributed over past state-action pairs. These values were chosen with a one to one ratio to allow the user to correct any mistakenly given feedback signal with just one or two extra button presses. Also, the positive feedback signal was a positive value to allow the user to signify the importance of a state-action pair compared to its initially zero value.

In addition to the two feedback buttons, the user was able to reset the agent back to its starting position with the use of a button. This caused only a position reset, it would not reset any learned state-action values. A second button allowed the user to manually quit the experiment. This button was introduced to prevent any loss in data if the participant would decide to close the experiment before finishing it correctly. This button also allowed the participant to give up teaching if it was too frustrating instead of simply stop giving feedback.

4.2. Participants

A total of 60 participants successfully completed the online experiment; 39 men, 21 women. Each subject group had 15 participants, except the group with the third agent type who had one participant removed due to data corruption. The mean age was 25 ± 9, the youngest was 18 and the oldest 58 years old. The questionnaire asked if the participant knew the meaning of the terms 'machine learning' and 'reinforcement learning'. A total of 48 participants knew these terms. Most of the participants were recruited from an Artificial Intelligence study.

The aim of the study was to solve the PCP which only occurs if there is a positive bias on all feedback. The participants had an average positive to negative feedback ratio of 2.58 ± 1.45 with a minimum of 0.39 and a maximum of 7.6727. This large difference in the feedback ratio between participants showed that participants had different teaching strategies but all gave more positive than negative feedback. Therefore, all tested agents could potentially suffer from the effects of the PCP.

On average participants assessed 64% of all performed actions with at least one feedback value. The smallest percentage was 10% and the highest 97%. These extremes again show that different participants can have different teaching strategies. A low percentage did not necessarily meant that the agent was also not able to learn the task, and we chose not to exclude these participants.

4.3. Measurements and analysis

A total of four measurements are reported in this article:

1. The number of times the was goal reached during training.
2. The number of actions the agent required to reach the goal each subsequent time.
3. The number of states the agent experienced repeatedly in quick succession.
4. The agent's capability of reaching the goal from any maze position.

The first measurements could determine for each condition whether the agent managed to reach the goal reliably within the experiment's time constraint. The lower this measurement the worse the agent's task performance. Do note that the maximum number of times an agent could reach the goal was five.

The second measurement supplemented the first. This measurement allows us to analyze for each condition how quickly its agents reached the goal every time. Only those agents that managed to reach the goal for five times were included in this measurement. This second measurement was chosen to provide an additional view on the agents task performances in terms of efficiency. If agents from a specific condition would require more actions to reach the goal each subsequent time, these agents learned the task slower than the agents of other conditions. This could be due to often performed repetitive behaviors caused by positive circuits or exploration.

The third measurement was a measure of the main symptom of the PCP; the repetitive behavior. The higher this measurement, the more states the agents of a condition experienced while repeating some undesired behavior. These undesired repetitive behavior were classified as such when the agent experienced at most three unique consecutive state more than once in a row. In light of the task this was an undesired behavior as it would mean that the agent moved away and towards the goal repeatedly in quick succession. This classification counted all the same behavior as repetitive as the symptom solution under the settings described in Section 4.

The fourth measurement was a score for each position in the maze. This score described the number of actions an agent required to reach the goal from that position compared to the (optimal) number of actions of the shortest path from that position to the goal. This score was determined for every position in the regular maze used for training as well as a slightly different maze. This second maze had the shortest path from the starting position to the goal blocked, as can be seen in Figure 3. This fourth measurement in both of these mazes allowed us to measure the learned task solution's optimality and whether this solution could withstand an environmental change. The calculation of this score is explained in the following paragraphs.

The score of the fourth measurement was based on the number of actions an agent required to reach the goal from some arbitrary but accessible maze position. This number was collected by simulating the agent's behavior based on the stored state-action values the agent learned when training ended. This simulation was performed for every accessible maze position in both the training and obstructed mazes. If the agent did not reached the goal within 525 actions, the attempt was marked as a failure. If the goal was reached, the score was calculated by dividing the optimal number of actions by the number of actions the agent required. This ensured that the score's range was $[0,\ 1)$. If the score was 1, the agent learned the optimal path. The lower this score, the more actions the agent required to reach the goal on top of the optimal number.

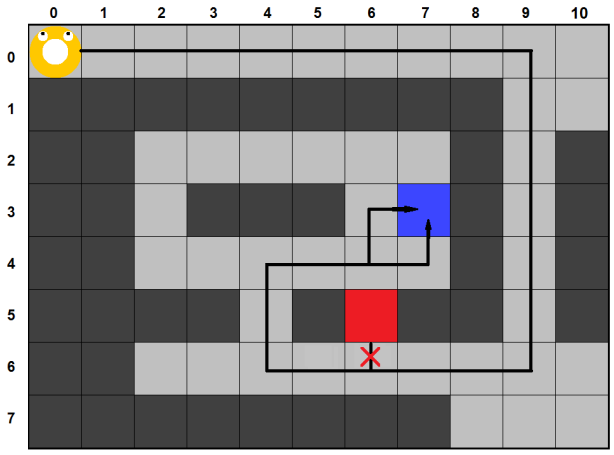


Figure 3: The obstructed maze. The red maze position is the added wall that obstructed the shortest path to the goal. It redefined the optimal solution to the task.

Section 5 provides the results on these four measurements. Two-way ANOVAs were performed on the first and third measurement, with the proposed solutions as the independent variables. Each tested measurement had four means, one for each condition, which were compared with the Tukey-Kramer multi-comparison method [36]. Finally, all error bars shown in the bar and interaction plots consist of the standard error.

5. Results

This section presents the results on the four measurements conducted for each of the four agents. These results are briefly discussed to point out any interesting differences or possible abnormalities. A full discussion of all the results is given in Section 6.

5.1. Number times the goal was reached

Figure 4 shows the number of goals each of the four agents reached on average. Both the preventive and symptom solutions caused a significant improvement as compared to the baseline agent (preventive; $p<0.01$, $\mu=3.67$, SE=0.41, d.f.=1, F=10.82 and symptom; $p<0.01$, $\mu=4.36$, SE=0.43, d.f.=1, F=24.27). There was no significant interaction between the two solutions ($p>0.05$, d.f.=1, F=3.12). The combination of solutions caused an additive effect, meaning that the agent with both solutions reached the goal more often compared to the agents with one of the solutions implemented. However, according to the mean comparison test between the mean of the combination of solutions and the other conditions, this was not a significant increase which may have been caused by a ceiling effect.

Figure 5 provides the distribution of how many trained agents never reached the goal, reached it at least once or for the maximum number of five times. This bar plot gives more insight in the improvement caused by the solutions. Half of the baseline agents never reached the goal. This was decreased by the proposed solutions

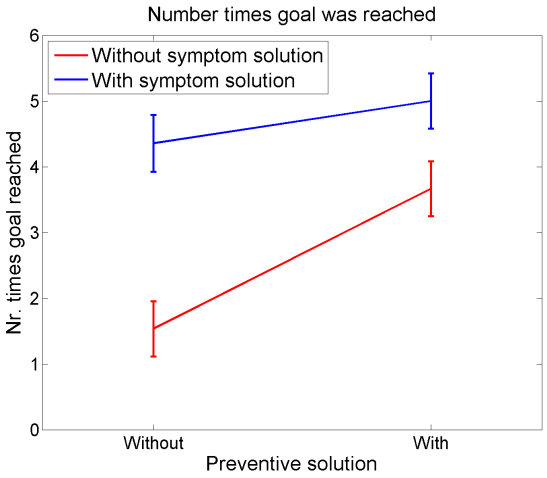


Figure 4: The number of times each of the four agents reached the goal on average. The proposed solutions both caused an significant increase in the number of times the goal was reached compared to the baseline agent. The error bars resemble the standard errors.

with a factor of five as more agents reached the goal for five times. The agents with the combined solution, always reached the goal for at least five times.

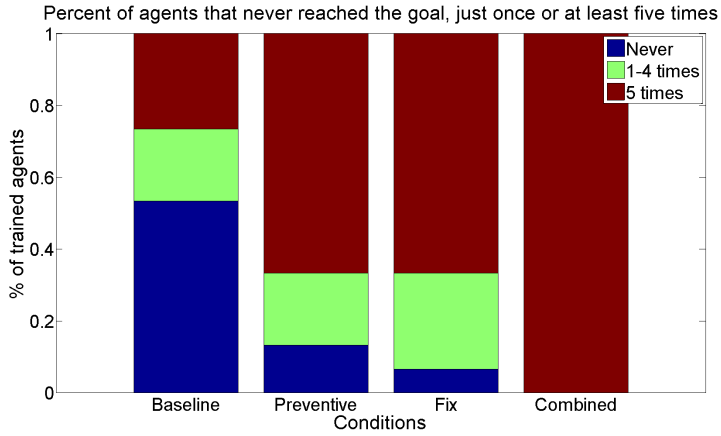


Figure 5: The distribution of the number of trained agents that never reached the goal, reached the goal at least once, and those that reached the goal for the maximum number of five times.

5.2. Number of actions required to reach the goal

Figure 6 shows the number of actions the agents required on average to reach each subsequent goal. This bar plot shows that all agents of each condition spend more actions to find the goal for the first time then for the next times. This comes as no surprise as the agents have no idea what is expected of them when training just started. What is interesting, is that the baseline and the preventive solution conditions both required relatively few steps to find the goal after finding it for the first time. The symptom solution and combined solution

conditions required relatively more time to find the goal the second time. Finally, the results show that these agents who learned to reach the goal for five times during training, reached it for the fifth time with nearly the shortest path.

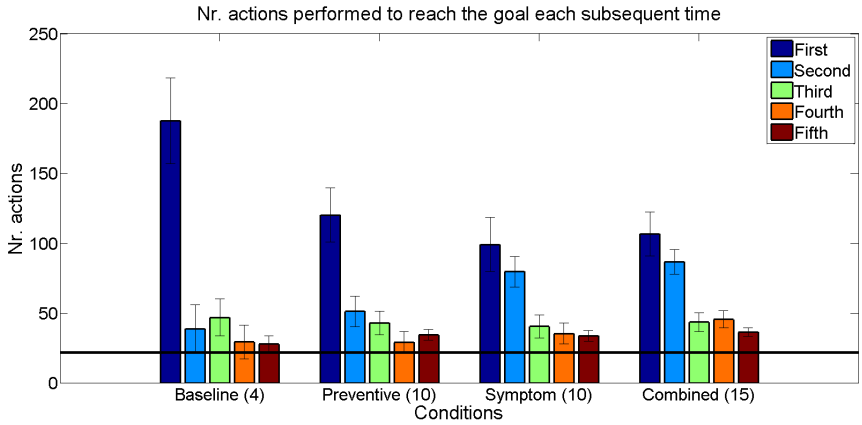


Figure 6: The average number of actions required for each agent to reach the goal for the first, second, third, fourth and fifth time. The horizontal line shows the number of actions (22) required for the shortest/optimal path to the goal from the starting positions. Only the agents that reached the goal for five times were included. On the x-axis the number of agents that achieved this was noted. The error bars resemble the standard errors.

5.3. The number of states repeatedly experienced in succession

Figure 7 shows how many states the agents experienced on average for each condition while repeating an undesired behavior. The analysis showed that the preventive solution caused the agents to significantly spent less time in positive circuits ($p<0.05$, $\mu=67.93$, SE=11.07, d.f.=1, F=12.27). The symptom solution however, caused no significant improvement compared to the baseline agents. Also, there is a significant interaction between the two proposed solutions ($p<0.05$, d.f.=1, F=5.05).

The symptom solution had no conclusive effect on the time the agent spent in positive circuits. It is neither significantly worse than the agents with the preventive solution nor is it significantly better than the baseline agents.

5.4. The agent's capability to reach the goal from anywhere

The fourth measurement analyzed how well the agents were capable in reaching the goals from any position in the maze. Figure 8 shows an overview of the average number of positions from which each type of agent managed to reach the goal. A total of 45 positions could be reached by the agent, excluding the goal position.

This figure shows that in the actual training maze the combination of solutions caused the agents to reach the goal from almost every possible position. The preventive solution showed a slight decrease compared to the baseline agents. The symptom solution however, showed a slight increase. The plotted error bars (based on the standard error) do overlap, making these results inconclusive.

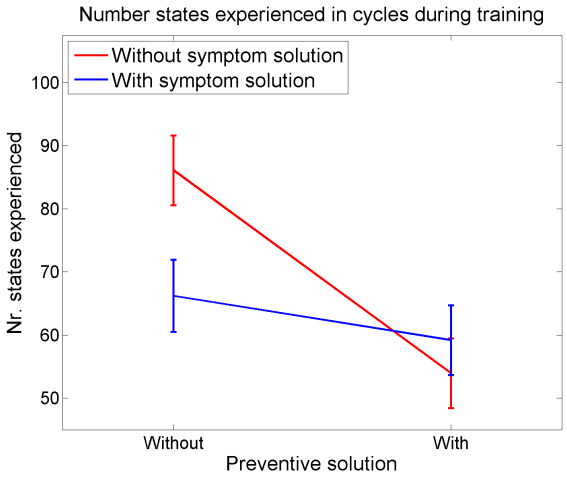


Figure 7: The total number of states experienced while exploiting positive circuits with a maximum length of three states. The results show that the preventive solution caused a significant decrease in this number.

The same measurement was performed on the obstructed maze and shows an interesting effect. The baseline agents show little decrease. This in contrast with the agents with one or both proposed solutions, these agents show a more substantial decrease. A slight decrease was expected as the environment slightly changed. However, this large decrease shows that the proposed solutions inhibit the agent's capability to find the goal from any position.

The next two sections examine the efficiency of each agent type in reaching the goal from every position. These results are shown in the maze itself. Each position includes a number of how many agents reached the goal from that position. The color of every position visualized the efficiency of the found path to the goal; the brighter the color, the more optimal the path is.

5.4.1. The training maze

Figure 9 shows how optimal each found path to the goal was for each of the four agent types. As discussed in Section 4.3, each position accessible by the agent shows the number of agents that reached the goal from that position. The color of each position shows how optimal the found paths were on average; the brighter the color, the more closely the found path was to the optimal path in terms of the number of actions required.

Figure 9 clearly shows that the baseline agents were often capable in finding the goal (high numbers in each position), but were not efficient in doing so (dark/reddish tints in each position). The top row of positions had only few agents that reached the goal, but did so with an efficient path. All other positions show that baseline agents were not able to reach the goal efficiently on average.

The agents with the symptom solution found more optimal paths than the baseline agents. However, the maze is fairly checkered, meaning that at some positions the agent was not able to learn a fairly optimal path. The agents with the preventive solution however, shows that these agents learned very efficient paths to the

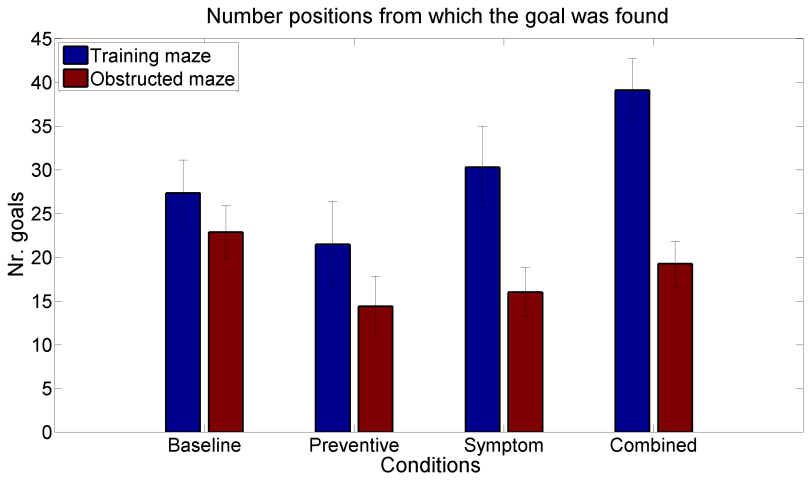


Figure 8: The average number of positions from which each agent was capable to reach the goal position. This bar plot shows this number for all 45 accessible positions in the training maze, as for all 44 accessible positions in the slightly obstructed maze. The combined solutions show a large increase. However in the obstructed maze, the solutions seem to inhibit the how often the agents were capable of reaching the goal compared to the baseline agents. The error bars resemble the standard errors.

goal. From Figure 8 we can conclude that the baseline agents and the agents with either of the two solutions implemented, find the goal just as often from any position. The agents with both solutions show the same efficiency as the agents with only the preventive solution but more agents also actually learned these optimal paths. Figure 8 also concluded that the combined solutions caused the agents to find the goal more often in the training maze. However, Figure 9d shows that these agents were also very efficient in doing so.

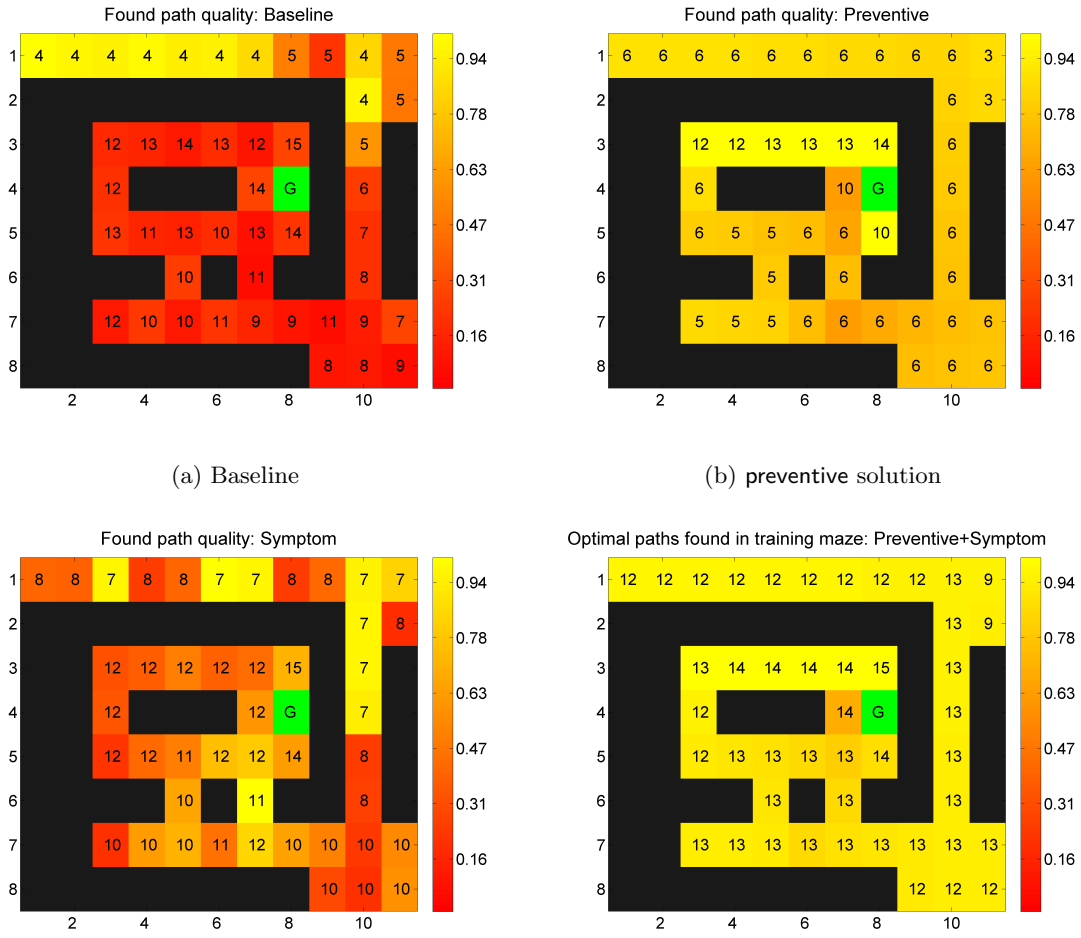
A final interesting effect we can conclude from Figure 9, is that the further the position from the maze, the less often the agents were capable of finding the goal unless they combined both proposed solutions. In this case, almost all agents were still capable of finding the goal even from the most distant positions.

5.4.2. The obstructed maze

Figure 10 shows the efficiency of each agent type to find the goal in the obstructed maze (see Figure 3). These figures show similar effects as in described in Section 5.4.1 from Figure 9 for the maze used for training. Although a new effect is the decrease in the number of agents that were able to reach the goal, especially from the positions more distant from this goal. The positions past the obstructions are the same as expected, but the position before the obstruction are effected. This is especially the case for the agents that use either solution or both of them, as also shown in Figure 8.

6. Discussion

The purpose of this study was to develop and analyze two possible solutions to the positive circuits problem. The PCP was treated as a problem of too much exploitation versus exploration, and both proposed solutions



(a) Baseline

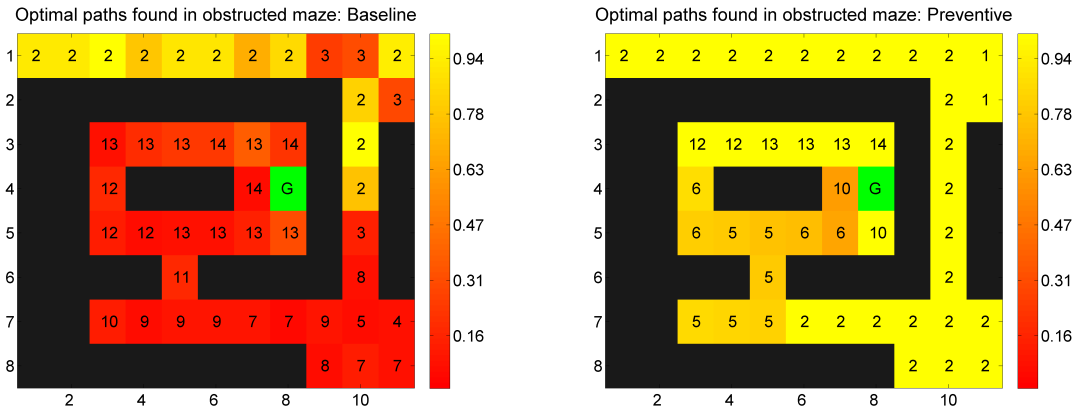
(b) preventive solution

(c) symptom solution

(d) Both solutions combined

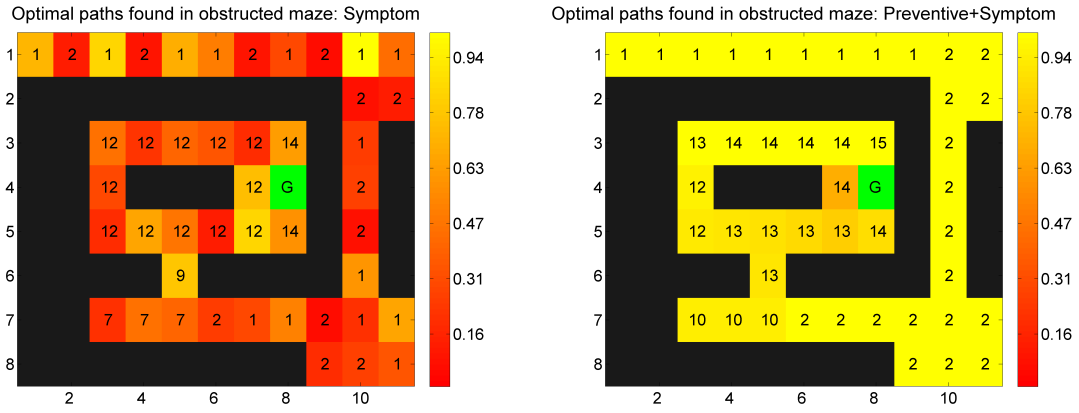
Figure 9: An overview of how optimal the learned paths to the goal were for each agent type in the training maze. Each accessible position in the maze is colored; the brighter the color the more optimal the fond path is in terms of the number of actions required to reach the goal position (a value of 1 represents the optimal path). The number in each maze position shows how many agents reached the goal from that position. These four figures show that the baseline agents and symptom solutions have several positions with very optimal paths and others with very sub-optimal paths (a checkered maze). The agents with the preventive solution and combined solutions show that a great number of agents found the goal very efficiently.

tried to motivate the agent to explore more in their own unique way. The two proposed solutions are the first solutions to solve the PCP and improve the IRL agent's task performance through exploration. The first solution did this by generalizing state-action values to motivate the agent to keep performing the most valued action in neighboring states. For this solution we hypothesized that it would prevent the agent from learning the presence of positive circuits, and as a consequence learn the task faster and more reliable. The second solution tried to improve exploration based on the (distributed) user feedback. This second solution introduced explicit thresholds for detecting repetitive behavior and states with a high risk of initiating such repetitive behavior.



(a) Baseline

(b) preventive solution



(c) symptom solution

(d) Both solutions combined

Figure 10: An overview of how optimal the learned paths to the goal were for each agent type in the obstructed maze. Each accessible position in the maze is colored; the brighter the color the more optimal the found path is in terms of the number of actions required to reach the goal position (a value of 1 represents the optimal path). The number in each maze position shows how many agents reached the goal from that position. These figures show similar effects as in Figure 9, although in the obstructed maze agents have trouble to reach the goal from distant positions. Also, the solutions inhibit the agent's capability of finding the goal.

Instead of a learned action, a feedback-based action was selected to explore the return of the relatively best assessed action. Agents with this solution, we hypothesized, would still learn the presence of positive circuits. However, the agent would spent less time repeating behavior as this behavior would be avoided or otherwise fixed by the user without additional effort. The solutions were combined to measure the interaction between the solutions. We expected that if the first preventive solution would be optimal, the second symptom solution would have no additional affect on the measurements.

The measurements analyzed the task performance of the baseline agents, agents with the preventive solution,

agents with the symptom solution, and the agents with both solutions. In the sections below we first discuss the improvements the solutions caused in the agent's task performance. Second, we discuss the issues that arise when using either or both of the two proposed solutions. Here we also provide possible future work to resolve these issues. Table 2 shows an overview of the results to substantiate the conclusions in the next sections.

	Baseline	**Preventive**	**Symptom**	**Combined**
Nr. times goal was reached	1.53	3.67	4.36	**5.00**
% of agents reached goal once	53.3%	13.3%	6.7%	**0**%
% of agents reached goal 2-4 times	20.0%	20.0%	26.7	**0**%
% of agents reached goal 5 times	26.7%	66.7%	66.7%	**100**%
Nr. actions to reach goal 1st time	187.5	120.2	**99.0**	106.7
Nr. actions to reach goal 2nd time	**39.0**	51.4	79.8	86.7
Nr. actions to reach goal 3rd time	47.0	43.0	**40.6**	43.7
Nr. actions to reach goal 4th time	29.5	**29.4**	35.6	45.7
Nr. actions to reach goal 5th time	**28.0**	34.6	34.0	36.6
Nr. states experienced with repetitive behavior	132.2	**67.9**	92.4	78.4
Nr. positions goal was found; Training maze	27.3	21.5	30.3	**39.1**
Nr. positions goal was found; Obstructed maze	**22.9**	14.4	16.0	19.3

Table 2: An overview of the results. This table shows the means or percentages for each of the measurements presented in Section 5. The best performances on a measurements are shown in bold.

6.1. Improvements caused by the proposed solutions

The first out of four main improvements is that the two solutions both caused the agent to reach the goal more often during training. Both the symptom and preventive solutions caused the agents to always find the goal as opposed to the baseline agents. The combination of the solutions even caused all agents to reach the goal for the maximum number of five times in the limited training time.

Second, the two solutions caused agents to reach the goal for the first time with fewer actions compared to the baseline agents. Also, both solutions did not prevented the agents from finding a short path to the goal just as the baseline agents. The slightly higher number of actions to reach the goal for the fifth time for the three solution conditions can be explained by the additional exploration performed by those conditions.

Third, the baseline agents proved to experience more states while repeating undesired behaviors than the agents with the preventive solutions. This indicates that the preventive solution spends less time in repeating behaviors and more time on learning a solution to the task. The results from the symptom solution were not significantly different than those of the baseline or the preventive solution.

Finally, the preventive solution caused agents to find the goal from more positions in the training maze. Figure 9b shows that these paths are also among the shortest or most optimal paths. When the preventive solution was combined with the symptom solution, nearly all agents could find the goal from any position in the training maze. Figure 9d shows that these learned paths are also optimal.

These four improvements show that the explorations caused by the proposed solutions increase the task performance of an IRL agent. The results indicate that the preventive solution indeed prevents the agent from learning the presence of positive circuits. Instead of exploiting such positive circuits, the agent learns a (near) optimal solution to the task independent in which state it starts. However, the symptom solution also improved the task performance, especially when combined with the preventive solution. This combination allowed more agents to find similar (optimal) task solutions as those learned by the agents with only the preventive. This compared to the baseline agents, who were unable to learn even a sub-optimal task solution most of the time due to the exploitation of positive circuits that became part of those solutions.

6.2. Issues with the proposed solutions

Although the two solutions cause improvements there are also some issues according to the results. First, the expected interaction is absent in the measurements. We expected that if the preventive solution indeed prevented agents from learning the presence of positive circuits, the symptom solution would have no positive effect when the two would be combined. However, even though the interaction is absent, the results from 9 show that the preventive solution prevents the agent from learning the presence of positive circuits. We believe that the positive effect of the symptom solution is caused by its feedback-based exploratory action in unfamiliar states. When triggered, this exploratory action overrides any exploratory action from the preventive solution. It is possible that this action explores more often or more useful state-action pairs than the actions of the preventive solution that explore more implicitly. This is a likely scenario as Figure 9b shows that the preventive solution prevents the agent from learning the presence of positive circuits, but when combined with the symptom solutions, agents learn to reach the goal from more positions, as shown in Figure 8. Future research can prove explicit exploratory actions in unfamiliar states is indeed an improvement on top of the already successful preventive solution.

A second issue is that the symptom solution seems to inhibit an agent's capacity to quickly find the goal again after finding it for the first time. This opposed to the baseline agents and agents with the preventive solution who manage to quickly find the goal a second time. This effect is also present when the solutions are combined in an agent. This may indicate that the feedback-based action in unfamiliar states inhibits the agent from quickly finding the goal again, instead the agent unnecessarily explores more state-action pairs. In other words, the symptom solution's exploratory action in unfamiliar states triggers exploration instead of exploiting an already effective task solution. This unnecessary exploration is most likely caused by a too high threshold in the naive thresholding method to determine unfamiliar states. The main reason to classify a state as unfamiliar was because it is unlikely that the agent learned accurate values for all actions in that state. The classification

can be improved by, for example, defining a confidence interval over the learned action values in a state. If this interval is large, the agent may not be certain of the learned values and exploration may be beneficial otherwise choose the learned action.

Finally we expected our non-myopic agents to learn a task solution that could withstand a small change to the environment. However, Figure 8 showed otherwise. In fact this figure indicates that the baseline agents can better withstand the environmental change than the agents with the proposed solutions. However, a closer look at Figure 9a shows that the baseline agents found the goal because their sub-optimal paths did not traversed through the obstructed state. From the results it is unclear why the non-myopic agents were not able to adjust their task solutions when the environment changed. The study performed by Knox and Stone [13] showed that non-myopic IRL agents should be able to do this. Future research may prove whether this is mere a matter of extending the learning time or a matter of an even higher discount parameter.

Future studies may focus on further investigating exploration as a method to solve the PCP and, with it, improve the IRL agent's task performance. The field of multi-armed bandit problems may prove to be useful in this, when feedback signals are treated as sampled rewards from an unknown distribution. Also, other approaches to solve the PCP can be studied. In the introduction we discussed why subtracting a constant from feedback values is not an option, as it is the ratio between rewards that causes positive circuits to emerge, not the fact that a positive feedback signal is indeed a (high) positive value. However, RL methods that use a reference reward may be able to neutralize the positive bias on user feedback [6]. These methods weigh a reward against an adaptive reward value, the reference reward, to determine whether the reward should be treated as positive given the past. As a more advanced version of these methods, Actor-Critique algorithms may also prove to be very useful for neutralizing the positive bias on user feedback and as such prevent the PCP on a feedback level [37, 6].

7. Conclusion

The purpose of this study was to construct and analyze two new solutions to the Positive Circuits Problem. These two solutions are the first of their kind as they solve the Positive Circuits Problem by improving the exploration of an interactive reinforcement learning (IRL) agent. Both solutions enforced exploration differently. The first solution explored more implicitly by generalizing state-action values over similar state-action pairs to motivate the agent to choose actions in states similar to each other. The second solution selected explicit exploratory actions based on user feedback in states classified as either part of a repetitive behavior or being a high-risk of trigging such repetitive behavior.

An experiment was conducted with a non-myopic agent in an episodic task. This agent was based on a state-of-the-art IRL agent, as it could handle delayed feedback signals and learned to provide rewards even when no feedback was given. This created an agent susceptible to the Positive Circuits Problem and allowed us to test the effects of the solutions on a state-of-the-art IRL agent.

The results showed that the proposed solutions increased the agent's task performance. Agents with the solutions found the goal more often and faster during training. Also, the used generalization technique of the first solution, function approximation, allowed agents to learn the optimal task solution. Function approximation successfully prevented the agent from learning the presence of positive circuits, this gave the agent more time to explore and learn a solution. The agents with the feedback-based exploration did learn the presence of those circuits but successfully improved the agent's capacity to learn a task solution.

The combination of the two proposed solutions showed that both solutions can still be improved. Agents that use function approximation may benefit from an additional exploration method to further improve both initial exploration and the exploration for different task solutions after finding such a solution. Agents with feedback-based exploration, could benefit from improved methods to classify high-risk states that could induce repetitive behavior.

Future work can focus on improving the suggested solutions or find new solutions that utilize exploration. However, an important next step in IRL is to identify and solve the Positive Circuits Problem in real world problems with actual robots. The two proposed solutions require little adjustments to test their effectiveness in the real world, as they are not limited to problems with small discrete state and action sets nor do they need complete knowledge of the problem.

This study proved to be another step towards a future where agents and robots learn new tasks based on a natural interaction with their users.

References

[1] J. E. Ormrod, K. M. Davis, Human learning, Merrill, 2004.

[2] L. Rendell, L. Fogarty, W. J. Hoppitt, T. J. Morgan, M. M. Webster, K. N. Laland, Cognitive culture: theoretical and empirical insights into social learning strategies, Trends in cognitive sciences 15 (2011) 68–76.

[3] W. B. Knox, Learning from human-generated reward (2012).

[4] A. L. Thomaz, C. Breazeal, Reinforcement learning with human teachers: Evidence of feedback and guidance with implications for learning performance, in: AAAI, volume 6, 2006, pp. 1000–1005.

[5] A. L. Thomaz, C. Breazeal, Teachable robots: Understanding human teaching behavior to build more effective robot learners, Artificial Intelligence 172 (2008) 716–737.

[6] A. G. Barto, Reinforcement learning: An introduction, MIT press, 1998.

[7] I. G. Alonso, Service robotics, in: Service Robotics within the Digital Home, Springer, 2011, pp. 89–114.

[8] J. Z. Yan, Z. L. Wang, Y. Yan, Research on household intelligent robot based on artificial psychology, Advanced Materials Research 898 (2014) 586–589.

[9] M. Vagaš, M. Hajduk, J. Semjon, L. Koukolová, R. Jánoš, The view to the current state of robotics, in: Advanced Materials Research, volume 463, Trans Tech Publ, 2012, pp. 1711–1714.

[10] F. Tajti, G. Szayer, B. Kovacs, B. Daniel, P. Korondi, Crm tc covering paper-robotics trends, in: Industrial Electronics Society, IECON 2013-39th Annual Conference of the IEEE, IEEE, 2013, pp. 48–53.

[11] J. Kober, J. A. Bagnell, J. Peters, Reinforcement learning in robotics: A survey, The International Journal of Robotics Research (2013) 0278364913495721.

[12] H. Cuayáhuitl, M. van Otterlo, N. Dethlefs, L. Frommberger, Machine learning for interactive systems and robots: a brief introduction, in: Proceedings of the 2nd Workshop on Machine Learning for Interactive Systems: Bridging the Gap Between Perception, Action and Communication, ACM, 2013, pp. 19–28.

[13] W. B. Knox, P. Stone, Framing reinforcement learning from human reward: Reward positivity, temporal discounting, episodicity, and performance, Artificial Intelligence 225 (2015) 24–50.

[14] W. B. Knox, B. D. Glass, B. C. Love, W. T. Maddox, P. Stone, How humans teach agents, International Journal of Social Robotics 4 (2012) 409–421.

[15] S. Schaal, Is imitation learning the route to humanoid robots?, Trends in cognitive sciences 3 (1999) 233–242.

[16] S. Schaal, et al., Learning from demonstration, Advances in neural information processing systems (1997) 1040–1046.

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

[18] C. Breazeal, B. Scassellati, Robots that imitate humans, Trends in cognitive sciences 6 (2002) 481–487.

[19] C. Breazeal, B. Scassellati, f 4 challenges in building robots that imitate people, Imitation in animals and artifacts (2002) 363.

[20] A. León, E. F. Morales, L. Altamirano, J. R. Ruiz, Teaching a robot to perform task through imitation and on-line feedback, in: Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications, Springer, 2011, pp. 549–556.

[21] H. B. Suay, S. Chernova, Effect of human guidance and state space size on interactive reinforcement learning, in: RO-MAN, 2011 IEEE, IEEE, 2011, pp. 1–6.

[22] P. M. Pilarski, M. R. Dawson, T. Degris, F. Fahimi, J. P. Carey, R. S. Sutton, Online human training of a myoelectric prosthesis controller via actor-critic reinforcement learning, in: Rehabilitation Robotics (ICORR), 2011 IEEE International Conference on, IEEE, 2011, pp. 1–7.

[23] W. B. Knox, P. Stone, Training a tetris agent via interactive shaping: a demonstration of the tamer framework, in: Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1, International Foundation for Autonomous Agents and Multiagent Systems, 2010, pp. 1767–1768.

[24] C. L. Isbell Jr, M. Kearns, S. Singh, C. R. Shelton, P. Stone, D. Kormann, Cobot in lambdamoo: An adaptive social statistics agent, Autonomous Agents and Multi-Agent Systems 13 (2006) 327–354.

[25] W. B. Knox, P. Stone, Learning non-myopically from human-generated reward, in: Proceedings of the 2013 international conference on Intelligent user interfaces, ACM, 2013, pp. 191–202.

[26] N. A. Vien, W. Ertel, T. C. Chung, Learning via human feedback in continuous state and action spaces, Applied intelligence 39 (2013) 267–278.

[27] L. Bottou, Online learning and stochastic approximations, On-line learning in neural networks 17 (1998) 25.

[28] W. B. Knox, P. Stone, Reinforcement learning from simultaneous human and mdp reward, in: Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 1, International Foundation for Autonomous Agents and Multiagent Systems, 2012, pp. 475–482.

[29] W. B. Knox, P. Stone, Combining manual feedback with subsequent mdp reward signals for reinforcement learning, in: Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1, International Foundation for Autonomous Agents and Multiagent Systems, 2010, pp. 5–12.

[30] W. B. Knox, P. Stone, Augmenting reinforcement learning with human feedback, in: ICML 2011 Workshop on New Developments in Imitation Learning (July 2011), 2011.

[31] W. E. Hockley, Analysis of response time distributions in the study of cognitive processes., Journal of Experimental Psychology: Learning, Memory, and Cognition 10 (1984) 598.

[32] A. Lockerd, C. Breazeal, Tutelage and socially guided robot learning, in: Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on, volume 4, IEEE, 2004, pp. 3475–3480.

[33] L. Baird, Residual algorithms: Reinforcement learning with function approximation, in: Proceedings of the twelfth international conference on machine learning, 1995, pp. 30–37.

[34] R. S. Sutton, H. R. Maei, D. Precup, S. Bhatnagar, D. Silver, C. Szepesvári, E. Wiewiora, Fast gradient-descent methods for temporal-difference learning with linear function approximation, in: Proceedings of the 26th Annual International Conference on Machine Learning, ACM, 2009, pp. 993–1000.

[35] D. S. Broomhead, D. Lowe, Radial basis functions, multi-variable functional interpolation and adaptive networks, Technical Report, DTIC Document, 1988.

[36] R. Lowry, Concepts and applications of inferential statistics, 2014. URL: `http://vassarstats.net/textbook/ch14pt2.html`.

[37] M. T. R. A. G. BARTO, J. 4 supervised actor-critic reinforcement learning, Handbook of learning and approximate dynamic programming 2 (2004) 359.