MC-AIXI-CTW Report

Elliot Catt, Suraj Narayanan, Arie Slobbe

May 27, 2017

Contents

1 Introduction **2**
- 1.1 The AIXI agent . . . 2
- 1.2 Monte Carlo Tree Search . . . 4
- 1.3 Context Tree Weighting . . . 4

2 Overview of implemented domains **6**
- 2.1 Kuhn Poker . . . 6
- 2.2 Biased Rock Paper Scissors . . . 7
- 2.3 Coin Flip . . . 7
- 2.4 Extended Tiger . . . 7
- 2.5 Partially Observable Pacman . . . 8
- 2.6 Tic Tac Toe . . . 8

3 Overview of agent program **8**
- 3.1 Software Architecture . . . 10
- 3.2 How to compile and run . . . 11

4 Experimental Results **11**
- 4.1 Biased Rock Paper Scissors . . . 13
- 4.2 Kuhn Poker . . . 14
- 4.3 True Kuhn Poker . . . 15
- 4.4 TicTacToe . . . 17
- 4.5 Extended Tiger . . . 18
- 4.6 Pacman . . . 20
- 4.7 Cross Domain-Kuhn Poker and BRPS . . . 22

5 Discussion **24**

6 Conclusion **24**

1 Introduction

We implement and test an approximation of the AIXI model (Hutter 2005). The AIXI model is a mathematical "solution" to the general reinforcement learning problem. That is, an AIXI agent solves the problem of maximising expected reward in an unknown computable environment that is potentially stochastic and partially observable. Unfortunately, the AIXI model is incomputable. A downscaled (i.e. computable) version of AIXI is presented in (Veness et al. 2011) which is called Monte Carlo AIXI Context Tree Weighting, or MC-AIXI-CTW. Our work implements the agent design of Veness and colleagues with the aim to reproduce their experimental results on a variety of reinforcement learning domains.

The following subsections briefly introduce the three main components of the MC-AIXI-CTW agent. A complete description of the AIXI agent can be found in (Hutter 2005). The Monte Carlo component is described in (Kocsis and Szepesvári 2006), and the Context Tree Weighting method in (Willems, Shtarkov, and Tjalkens 1995). Section 2 describes a variety of domains on which we tested our agent, section 3 describes our implementation of MC-AIXI-CTW and how to compile and run the code, and section 4 presents our experimental results on a variety of problem domains, and compares them with the results originally obtained by (Veness et al. 2011).

1.1 The AIXI agent

AIXI is a Bayesian ("on-average") optimality notion for general reinforcement learning agents. AIXI interacts with an unknown environment in cycles. In each cycle, the agent executes an action and in turn receives an observation and a reward. (when the environment is fully observable at all times, an observation is called a state; see figure 1). AIXI aims to choose actions that will on average lead to the best possible outcome. More precisely, AIXI maximises the expected sum of rewards over its lifetime.

Each time the agent takes an action, the environment responds by returning an observation reward pair. The agent processes the new observation-reward information and the next iteration begins with another action by the agent. In principle, the agent has access to its entire history of action-observation-reward cycles, and unlimited time to compute the next best action.

Suppose the AIXI agent interacts with an environment in cycles $k = 1, 2, \ldots, m$. The agent chooses action a_k, subsequently the environment "chooses" (produces) an observation reward pair $o_k r_k$, and the next cycle begins. Over time, a history sequence $h = a_1 o_1 r_1 \ldots a_m o_m r_m$ is formed.

Agents are distinguished by how well they choose their actions in light of rewards received from the environment. In cycle k, the full AIXI model chooses

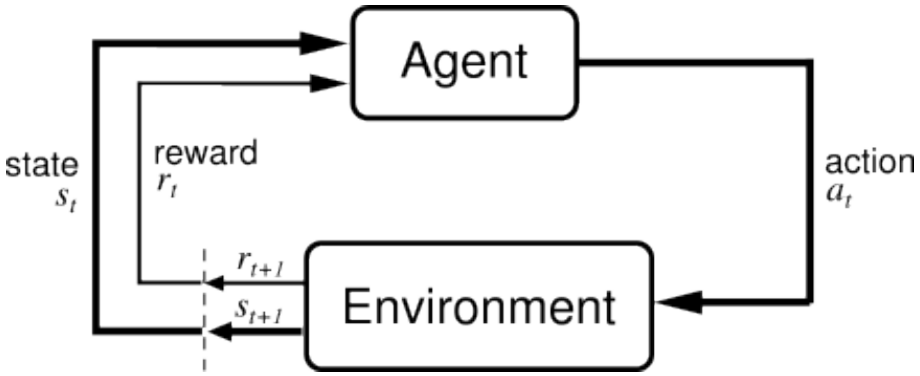


Figure 1: (Sutton and Barto 1998)

its next action a_k according to the following equation:

$$a_k := \arg\max_{a_k} \sum_{o_k r_k} \ldots \max_{a_m} \sum_{o_m r_m} (r_k + \ldots + r_m)\xi(o_1 r_1 \ldots o_m r_m \mid a_1 \ldots a_m),$$

where $\xi(o_1 r_1 \ldots o_m r_m \mid a_1 \ldots a_m)$ is a mixture environment model over a model class of chronological probability distributions. See (Hutter 2005) for the full details of this model. Intuitively, the agent considers the sum of the total reward over all possible futures up to m steps ahead, weighs each of them by the complexity of (environments) consistent with the agent's past that can generate the future, and then picks the action that maximises expected future rewards (Veness et al. 2011).

The alternating sum and max operators execute the (finite-horizon) expectimax operation, which is a classic result from sequential decision theory. Expectimax is responsible for planning a best course of action.

The mixture environment model is responsible for learning to predict future observations and rewards based on past experience. The technique is inspired by Solomonoff's universal induction scheme. Initially, the mixture assigns greater weights to simple environment models. Each interaction cycle generates new evidence as to which models are more likely to be the "true" model of the environment. Models consistent with the evidence acquire increased weight (i.e. importance) in the decision procedure while inconsistent models are pruned away.

The AIXI agent requires an overwhelming number of computations to find its next action. The expectimax operation runs in exponential time with respect to lookahead m and is computationally infeasible for all but the smallest values. Worse yet, the mixture environment model is not even finitely computable because the mixture is taken over an infinite set of environments. To deal with

these issues, Veness et al. developed a computable approximation of AIXI. This approximation uses Monte Carlo Tree Search (MCTS) to approximate the expectimax operation and Context Tree Weighting (CTW) to maintain a mixture environment model.

1.2 Monte Carlo Tree Search

Expectimax requires us to consider every possible future that can result from taking an action in the present. Again, this is computationally infeasible. Instead of considering every path through the future, Monte Carlo Tree Search (MCTS) samples as many paths as time and computation resources permit. An action with highly rewarding (simulated) futures is considered better than an action for which every sample returns nothing.

MCTS collects the outcomes of simulations in an iteratively expanding search tree. (Kocsis and Szepesvári 2006) developed a powerful heuristic which helps the agent focus its resources on useful parts of the tree. Actions with more simulations have better value estimates, so we must carefully consider which actions we care most about investing computational resources in. We prefer to spend resources on actions with high uncertainty about their real value, and/or with high probability of being the best action. These competing demands – exploration of new actions, and exploitation of known good actions – are balanced by taking a weighted average in such a way that total "regret" is minimised (see paper).

To illustrate, consider the problem of finding the best opening move in a game of tic-tac-toe. MC-AIXI-CTW runs a number of simulations to up to a pre-specified "search depth" into the future. The Context Tree Weighting method simulates the environment by taking prospective actions from the agent and return observation reward pairs, much like the environment would. In aggregate, the simulations yield an estimated win rate for each opening move. The agent subsequently chooses the opening move that had the highest win rate in its simulations.

1.3 Context Tree Weighting

The Context Tree Weighting (CTW) method was initially proposed as a sequential universal compression method. CTW learns to identify patterns in a bit-sequence by performing Bayesian model averaging over all prediction suffix trees of maximum depth D. If the sequence generated by an k-order Markov process (that is, the probability of the next bit being 1 is conditioned on the last k bits that were observed) and $k \leq D$ then CTW will converge to optimal prediction.

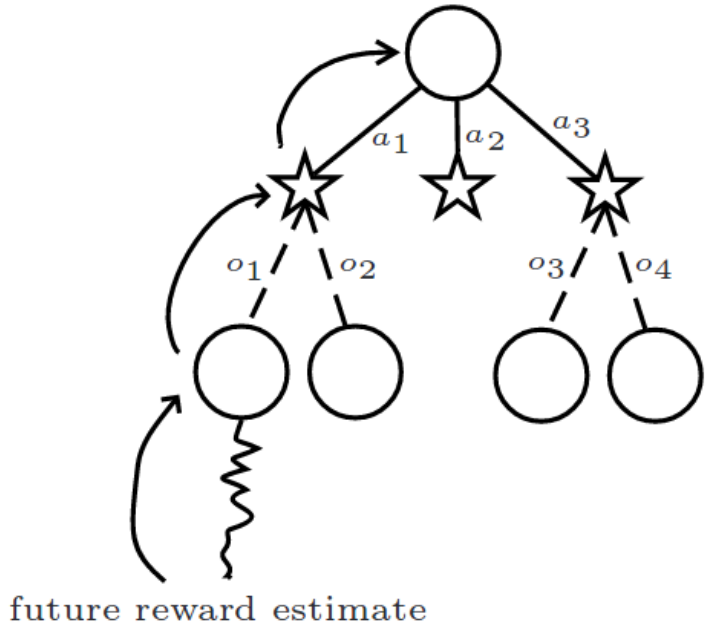


Figure 2: (Veness et al. 2011). Monte Carlo Tree Search collects the outcomes of simulations in an iteratively expanding search tree. The tree is composed of decision nodes (circles) and chance nodes (stars). These correspond to the alternating max and sum operations in the expectimax operation.

Several attributes make CTW a suitable approximator of universal Solomonoff induction:

- like Solomonoff induction, CTW maintains a mixture model over environments. The mixture converges to the true environment as evidence accumulates;
- CTW assigns higher prior probability to simple models;
- CTW covers the huge class of D-order Markov processes which contains many interesting and important environments; and
- The running time of CTW is linear in the length of the input sequence, compared to exponential in the length of more naïve methods.

Each time the agent simulates taking an action, CTW is used to predict the subsequent observation reward pair. In the MC-AIXI-CTW implementation, every action, observation and reward is encoded as a string of bits. So in practice, to generate an observation reward pair, a number of steps are involved. CTW is asked to predict the next bit b given the history sequence h, and subsequently the agent appends the predicted bit to the history sequence by setting

$h \leftarrow hb$. The agent predicts subsequent bits in this manner until they have appended to the original history h a sequence of bits of length corresponding to an obseration reward pair.

In this manner, the MC-AIXI-CTW agent uses the CTW method to simulate paths through the future in order to plan a best course of action. By interacting with the environment, MC-AIXI-CTW acquires the training data for CTW to learn over time how to emulate the environment, making it an increasingly good predictor of the future.

This convergence to the true environment works as follows. Let C_D denote the class of all prediction suffix trees of depth up to D, which are a form of variable order Markov models. We use C_D as our environment class and we assume that the "true" environment E^* is contained in C_D. To each $E \in C_D$ we assign a code with length $CL(E)$. We can think of this code as a compressed representation of the environment; simple environments have simple (short) codes and complex environments have complex (long) codes.

The mixture environment model ξ from the AIXI model is approximated by

$$\begin{aligned}\xi(o_1r_1 \ldots o_mr_m \mid a_1 \ldots a_m) &\approx P_{CTW}(o_1r_1 \ldots o_mr_m \mid a_1 \ldots a_m) \\ &= \sum_{E \in C_D} P(E) P(o_1r_1 \ldots o_mr_m \mid E, a_1 \ldots a_m) \\ &= \sum_{E \in C_D} 2^{-CL(E)} P(o_1r_1 \ldots o_mr_m \mid E, a_1 \ldots a_m).\end{aligned} \tag{1}$$

The $2^{-CL(E)}$ factor causes simple environments to have an outsize contribution on the output value P_{CTW}. The factor $P(o_1r_1 \ldots o_mr_m \mid E, a_1 \ldots a_m)$ approaches zero when the "real" environment E^* produces observation reward pairs $o_1r_1 \ldots o_mr_m$ that we would not likely observe if E were the true environment. Hence, over time the $P(o_1r_1 \ldots o_mr_m \mid E^*, a_1 \ldots a_m)$ term that corresponds to the real environment $E*$ will dominate the sum.

2 Overview of implemented domains

2.1 Kuhn Poker

Kuhn Poker is a simplified version of poker that was introduced in (Kuhn 1950). It involves a deck of 3 cards (King, Queen, Jack). First the cards will be given out, and each player bets 1, then the opponent will choose to bet or pass, after looking at his card, the agent will then choose to bet or pass. If the actions taken are equal (both bet or both pass) then the player with the higher card wins the round (King>Queen>Jack). If instead the agent chose to pass, the opponent wins the round. If the opponent passed then the agent bet 1, the

opponent then has the choice to bet or pass again, if they pass the agent wins, if they bet then the player with the highest card wins.

The opponent will always play the Nash Strategy. To represent this at the start of the round the opponent chooses an $\alpha \in [0, \frac{1}{3}]$ at random. If they drew a Jack, they will bet with probability α. If they drew a queen they will bet with probability 0. If they drew a king they will bet with probability 3α. If there is a second round of betting, they will choose to bet with probability $\alpha + \frac{1}{3}$ regardless of their card.

If the agent wins, they get reward equal to the number of chips in play. If the agent loses they will get a reward equal to the negative of the number of chips they bet.

We suspected that the Kuhn Poker described above, which is presented as mentioned in (Veness et al. 2011), is not the true version of the game because it is not zero sum. We took their description of the game literally and implemented it as the domain Kuhn poker. Additionally we implemented the True Kuhn Poker domain (that is, the Kuhn Poker exactly as described in (Kuhn 1950)). In True Kuhn Poker, that is exactly as detailed in (Kuhn 1950) the reward for the agent is the number of chips the opponent put in.

2.2 Biased Rock Paper Scissors

Biased Rock Paper Scissors is very similar to regular rock paper scissors, with an exception in how the opponent chooses to play. Both the agent and the opponent will choose either Rock, Paper or Scissors. As per usual, Rock beats Scissors, Paper beats Rock, Scissors beats Paper. If the agent beats the opponent, the agent will receive a reward of 1. If the agent loses it will receive a reward of -1. If it is a draw, meaning both opponent and agent make the same choice, then the agent receives a reward of 0. The opponent repeats its choice of the previous round if it resulted in a win, and picks a random move otherwise.

2.3 Coin Flip

The Coin Flip is a very simple environment where the agent will choose heads or tails. Then a biased coin with probability p for heads and $1 - p$ for tails will be flipped. If the agent guesses correctly it will receive reward of 1, if not it will receive a reward of 0.

2.4 Extended Tiger

Extended tiger is a simple game where the agent begins sitting on a chair, in front of two doors. Behind one door is a tiger, behind the other is gold. While sitting the agent has the choice to stand or listen to the door. If the agent listens

they will make a correct observation about which door the tiger is behind with a probability of 0.85, and observe the wrong door with probability 0.15. Once standing the agent can open door 1 or open door 2. If the agent opens the door with the gold they receive a reward of 30, and if they open the door with the tiger they receive a reward of -100. If the agent stands up while sitting down or listens while sitting down they will receive a reward of -1. Any invalid action the agent performs yields a reward of -10.

2.5 Partially Observable Pacman

Partially observable pacman (POcman) is a version of the 17x19 pacman game where the agent controlling Pacman cannot observe the whole space. Instead, pacman can detect: if there are walls around him; if he can see a ghost in the four directions around him; if there are is a food pellet within 2, 3, and 4 of his location; if he can see a food pellet in any of the four directions; and if he has the power pill or not. The ghosts will move randomly, unless they are within a manhattan distance of 5 or less, in which case they will try to move towards him. The agent receives a reward of -1 for every move it makes, -10 for hitting a wall, +10 for collecting a food pellet, -50 if touched by a ghost, and +100 when it completes the level by collecting all pellets.

2.6 Tic Tac Toe

Tic Tac Toe is an adversarial fully observable deterministic game played on a 3 by 3 board. It begins with the agent placing a 'X' on one of the places on the board, the opponent then places an 'O' on another place. This continues until there is a horizontal, vertical or diagonal line of 3 'X's or 'O's on the board, at which point the game is over and the player who played the line wins. If the board is full and there is no winner, then the game is a draw. If the agent attempts to place an 'X' on a place that already has a piece he will penalized by -3 reward. If the agent loses the game, they get a reward of -2. If the agent wins the game they get a reward of 2, and if the agent draws the game it will get a reward of 1. The opponent will always play in a random available spot.

3 Overview of agent program

The MC-AIXI-CTW agent program is composed of several components. Each is briefly presented in the following overview. More comprehensive documentation is provided in the source files themselves.

- **AIXI** The AIXI module has four main functions:
  - **(d)encode**: Decodes and encode the reward, actions and percepts of the agent with the environment.

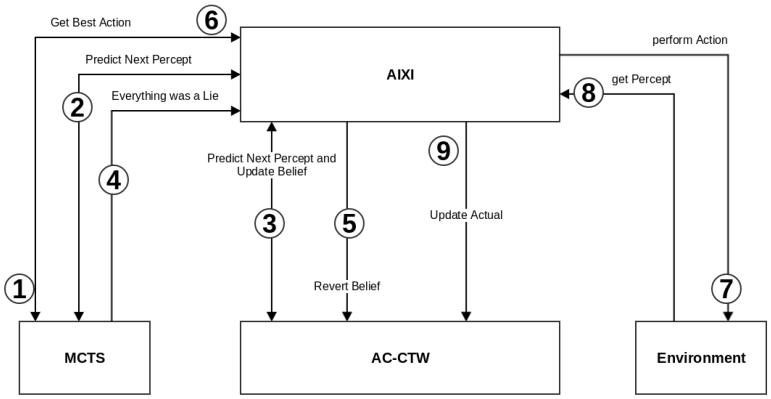
- **update**: Updates the model based on the updated and generated percepts.
- **initialisation**: Initialises the agents parameters: History size, number of simulations/actions, number of observation and reward bits.
- **planning**: The planner from the search tree returns a planned action.

- **MCTS** The MCTS module has three main functions:
  - **search**: Estimates the best action by simulation playouts and collecting the results in an iteratively expanding search tree.
  - **sample**: grows the search tree by one node by simulating a playout and storing the results.
  - **prune**: retains useful computations from the previous agent-environment interaction cycle while reclaiming memory allocated to nodes that are not necessary anymore.

- **CTW**
  The CTW module has two main functions:
  - **update**: Takes care of updating and/or reverting the context tree. Update handles both by means of a flag variable that indicates the operation that need to be performed.
  - **genNextSymbolsAndUpdate**: Generates the symbols for the next percept and updates the tree with them.

  Detailed documentation of the other functions are given in the header file.

  CTW Implementation Enhancements
  - **Numerical Stability:** All the probability related calculations are done in log-space to avoid under-flow issues.
  - **Efficient Node Creation:** Nodes are created dynamically when requested. This allows for slow memory growth.
  - **Efficient Node Deletion:** Nodes created during MCTS simulations are deleted, thereby reclaiming memory allocated during MCTS search.
  - **Generic Update:** The update method is equiped with an operation flag so that the same infrastructure can be re-used to perform *revert* operation as well.
  - **Built-in Sanity Checks:** The code has built-in sanity-check that take palce every update/revert operation. The code checks for invalid probability values, and invalid reverts.



3.1 Software Architecture

Our implementation of MC-AIXI-CTW was designed keeping modularity and extensibility in mind. The figure above shows the components and the interaction between them. Logically the system is split into four parts.

- **Agent:** AIXI
- **Planner:** MCTS
- **Model-learner:** CTW
- **Environment:** TicTacToe/Pacman/BiasedRPS/...

The interaction between the modules is designed in such a way that each component is loosely coupled. This allows any component adhering to AIXI's interface to be plugged in without any change being effected in other module.

Agent-Environment Interaction Workflow

1. AIXI asks MCTS for the best action.
2. MCTS in turn asks AIXI to predict what the next percept would be.
3. AIXI asks CTW for the next percept, and updates the CTW with the percept. AIXI sends the percept to MCTS.
4. MCTS tells the agent that it has finished planning, and its time to revert the prediction.
5. AIXI tells CTW to revert the prediction.
6. MCTS returns the next best action to AIXI.
7. AIXI performs the action on the Environment.

8. Environment returns a percept to AIXI.

9. AIXI updates the CTW with the percept.

3.2 How to compile and run

Linux/Mac OS

Extract the source files into a folder

```
~/mc-aixi-ctw
```

and execute in the terminal:

```
cd ~/mc-aixi-ctw
make all
./bin/aixi -c /conf/coinflip.conf
```

Windows (Cygwin)

Extract the source files into a folder for example

```
/cygwin/c/mc-aixi-ctw
```

and execute the command (in Cygwin)

```
cd mc-aixi-ctw
make all
/bin/aixi.exe -c /conf/coinflip.conf
```

The file coinflip.conf may be replaced with any other file in the conf folder. After execution, the two files logfile.log and logfile.csv will contain data on the agent's actions and performance. Additionally, the command

```
./bin/aixi -h
```

or if using windows (cygwin)

```
/bin/aixi.exe -h
```

can be executed to find for more details on how to run the program.

4 Experimental Results

Our objective is to test the agent's learning capability in an environment with no prior knowledge. The learnt policy is evaluated at the end of the training phase. The environments Biased Rock-Paper-Scissors and Kuhn-Poker are considered easy. The other environments TicTacToe, Pacman, Extended Tiger are harder to learn. In each experiment in each domain we chose a CTW Depth, a Horizon, an Exploration, and Exploration Decay and a Number of Cycles. For each set of chosen parameters we ran 4 experiments of our agent on the given domain with the given parameters. We indexed each parameter set by experiment ID.

The parameters were chosen so that we had a balance between the parameters used in (Veness et al. 2011) and computational efficiency. The exploration and exploration decay were chosen so that after the number of cycles the exploration would be 10^{-5}. For most environments we fixed the horizon and increased the CTW depth. This was done to see what influence the choice of CTW depth had on the results.

We use the following notation:

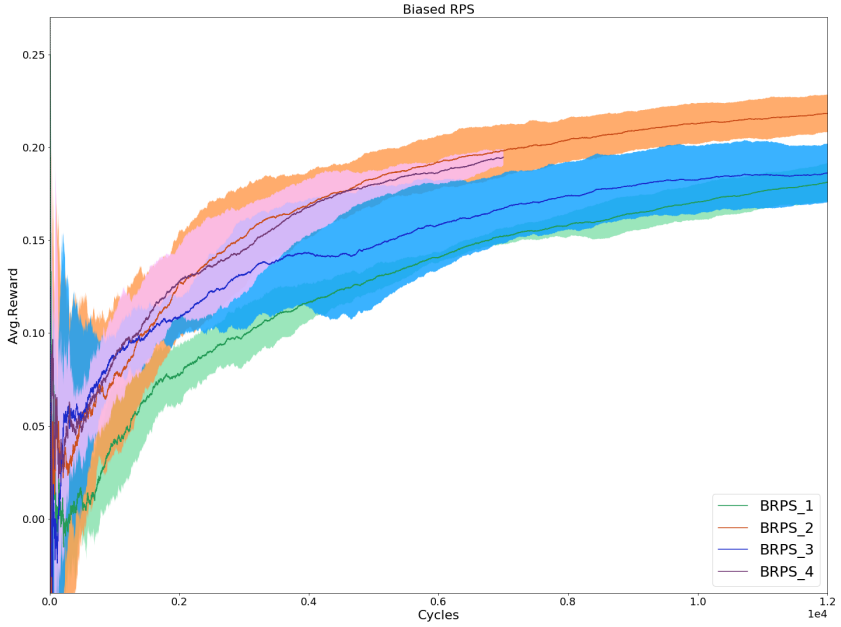
$$\begin{aligned} D &= \text{ CTW Depth} \\ m &= \text{ Horizon} \\ \epsilon &= \text{ Exploration} \\ \gamma &= \text{ Exploration Decay} \\ \text{Cycles } \times 10^3 &= \text{ Number of cycles} \\ \text{E\_} &= \text{ Intermediate Evaluation} \end{aligned}$$

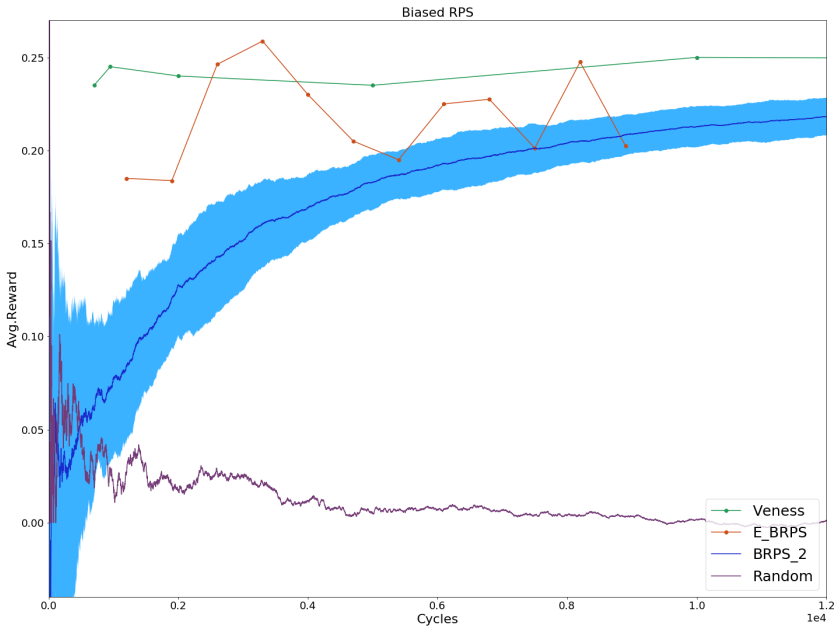
In the following subsections, we showcase the experiments conducted for each domain. For each domain we conducted multiple experiments with different parameter values, which are presented in tables within each corresponding section. We present two graphs for each domain. The first graph compares the learning curves between the experiments conducted. The second graph compares our best experiment with the agent of Veness, an agent acting randomly, and also shows the results of an experiment in which we performed intermittent evalutation. Intermittent evaluation means that every 500 cycles, exploration is paused for 200 cycles and performance is evaluated. As such, the results from intermittent evaluation are not weighed down by exploration moves from the agent. In each graph the shaded region represents the standard deviation from the 4 experiments that were run for that chosen set of parameters.

4.1 Biased Rock Paper Scissors

Experiment ID	D	m	ϵ	γ	Cycles	Average Reward
BRPS-1	32	4	0.999	0.9995	30	0.2293375
BRPS-2	96	8	0.999	0.9991	12	0.2419375
BRPS-3	256	4	0.999	0.9991	13	0.21425
BRPS-4	256	8	0.999	0.999	7	0.246525
E_BRPS	256	8	0.999	0.999	9	N/A

In biased rock-paper-scissors, we achieve optimal performance after the first 3000 training cycles. We expected this for two reasons. The CTW module is learning a 1^{st} order Markov process, and the agent simply has to learn the correlation in its opponent's actions. Moreover, the observation space and action space are relatively small. This allows CTW with shallow depth to model the environment in fewer cycles. Also, due to the small branching factor, MCTS is able to plan ahead more effectively. Therefore, increased CTW depth and longer horizon did not cause significantly faster converge to optimal play. Our agent performed similarly to that of Veness et al. 2011, and outperformed the random agent.

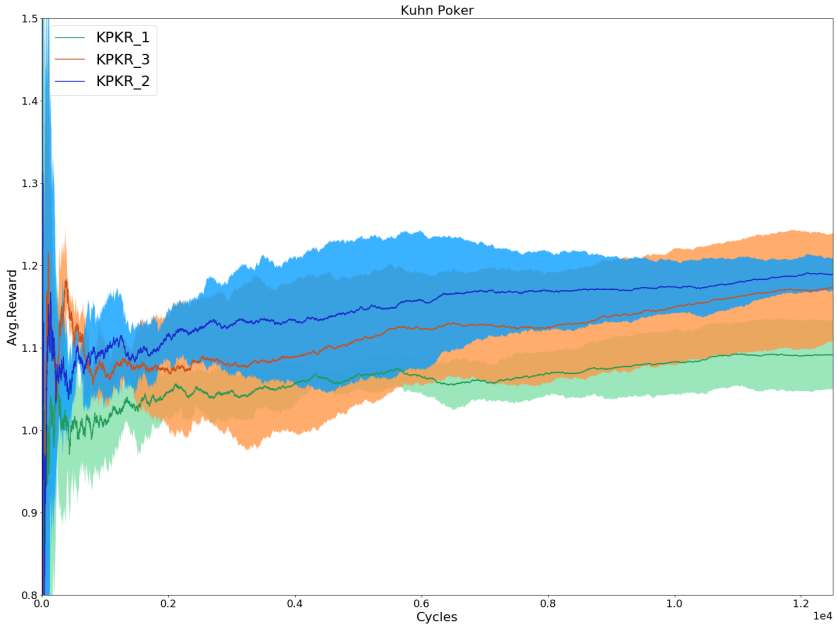


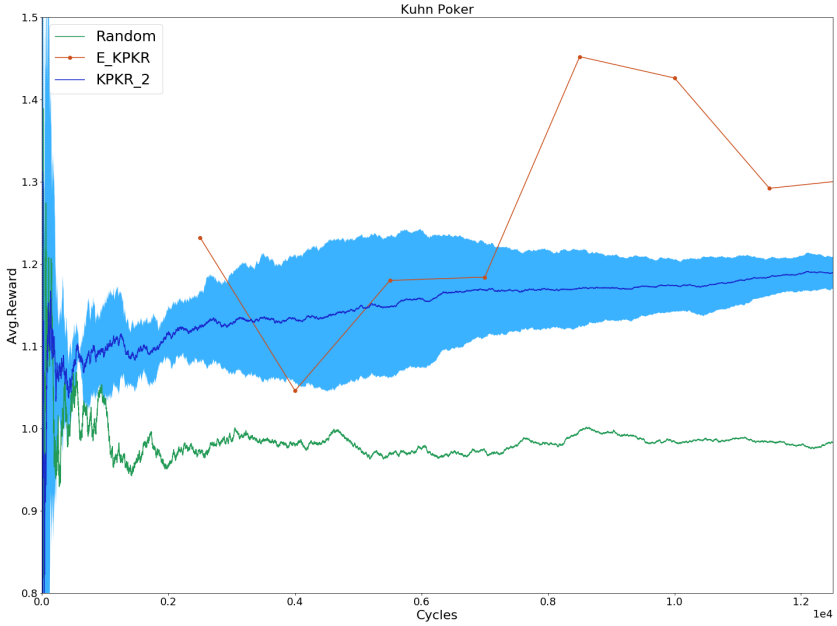


4.2 Kuhn Poker

Experiment ID	D	m	ϵ	γ	Cycles	Average Reward
KPKR-1	42	2	0.99	0.9999	2000	1.2024625
KPKR-2	96	4	0.9999	0.9995	24	1.258875
KPKR-3	256	4	0.999	0.9991	13	1.23595
E_KPKR	96	4	0.9999	0.9995	30	N/A

Due to the simplicity of the game, the optimal policy in Kuhn Poker is easy to learn. However, since the variety in possible rewards is so high the optimal policy will perform far better then the random policy. This was demonstarted by our experiment, where our agent learnt the near optimal polciy and out performed the random agent. We did not compare with Veness et al. 2011 since they used True Kuhn Poker.

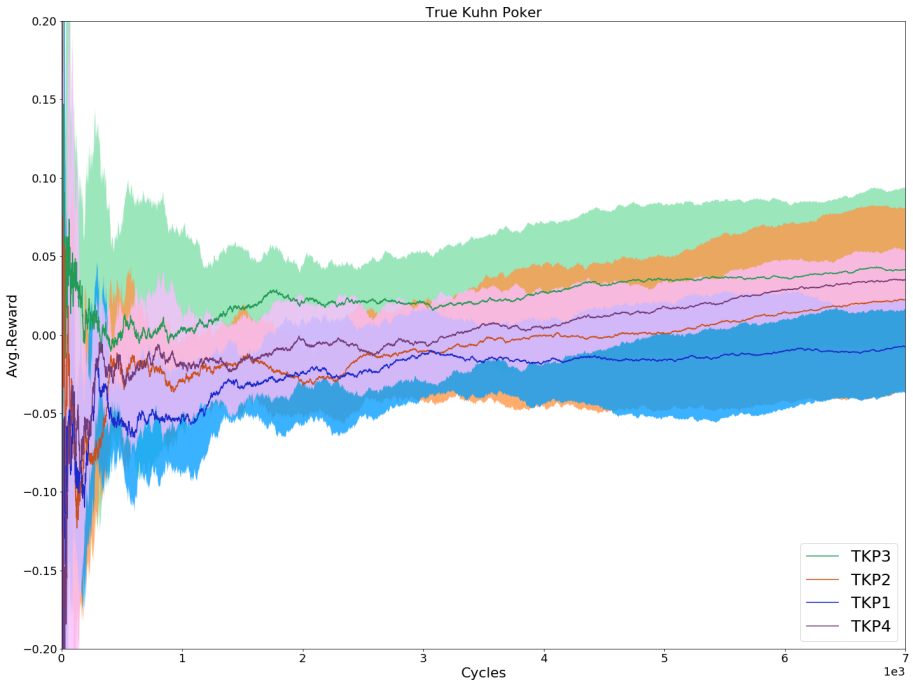


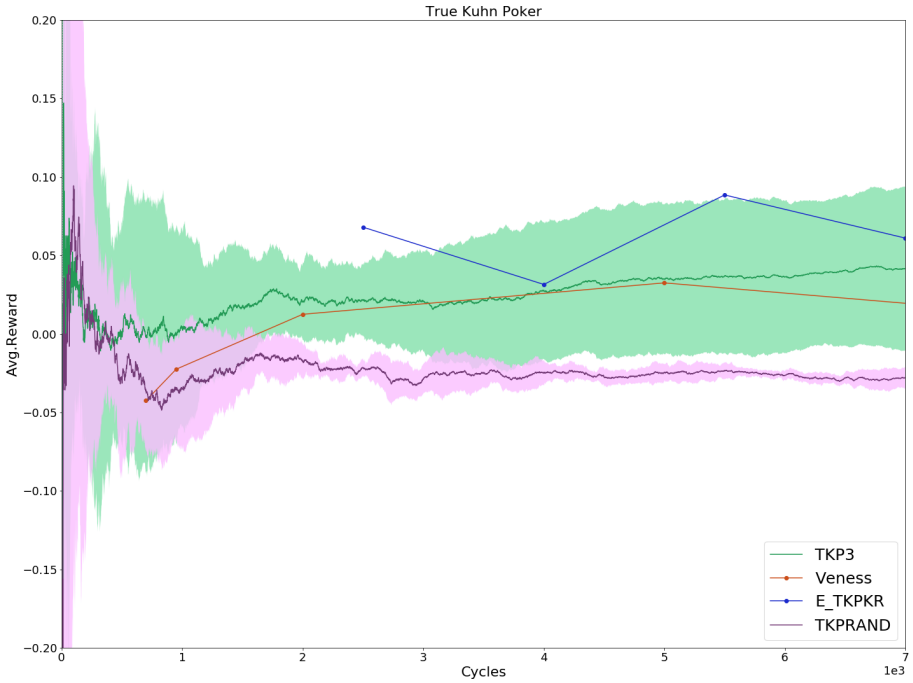


4.3 True Kuhn Poker

Experiment ID	D	m	ϵ	γ	Cycles	Average Reward
TKPKR-1	42	2	0.99	0.9999	2000	0.0719
TKPKR-2	96	4	0.9999	0.9995	24	0.01415
TKPKR-3	256	4	0.999	0.9991	13	0.0611
TKPKR-4	512	8	0.9999	0.999	7	0.056
E_TKPKR	96	4	0.9999	0.9995	30	N/A

True Kuhn Poker has a small observation, action and reward space. This allows the agent to quickly learn the optimal policy. For the optimal policy the average reward per cycle is $\frac{1}{18}$. In comparison to the random agent, our agent was able to achieve a superior performance. In comparison to Veness et al. 2011, which is near optimal, our agent was able achieve very a similar performance.

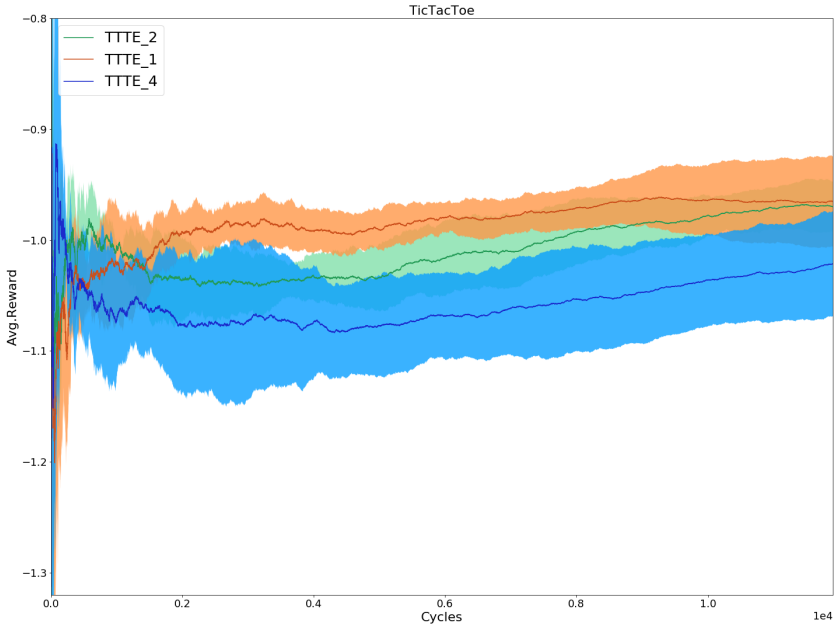


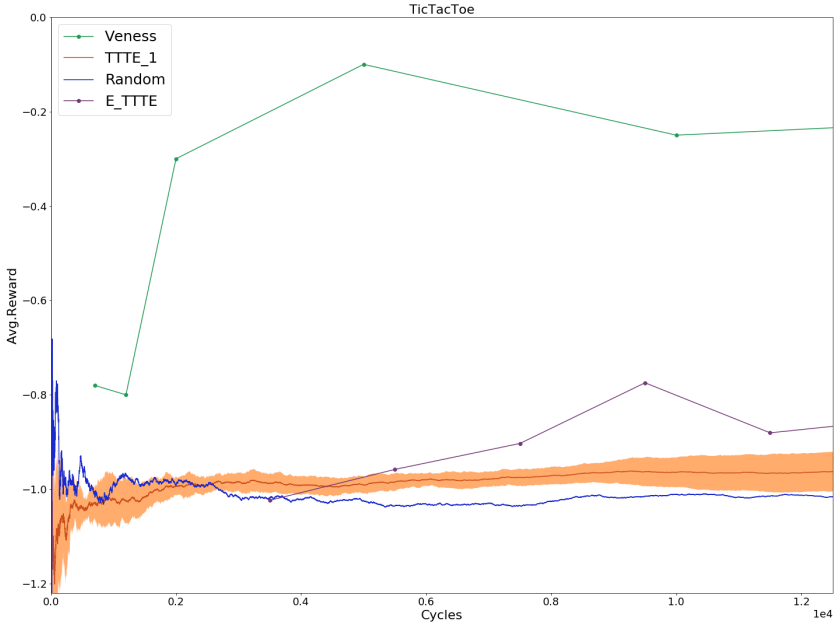


4.4 TicTacToe

Experiment ID	D	m	ϵ	γ	Cycles	Average Reward
TTTE-1	64	9	0.9999	0.99991	145	0.0661625
TTTE-2	96	9	0.9999	0.9995	24	-0.636375
TTTE-4	128	10	0.9999	0.9991	12	-0.83726875
E_TTTE	64	9	0.9999	0.99991	180	N/A

TicTacToe is a hard game because it involves a large observation space. It should be noted that Veness's agent took more than 3 weeks of training to get optimal results. In experiment TTTE_1 our final evaluation receives better a score than Veness. We think this is an anomaly because its associated training curve does not validate the result. We could not redo the same experiment again due to the large number of cycles it required, and time-constraints. Our training graph shows that we have a rising trend and a narrowing standard-deviation. Therefore we hypothesise that, given more time, we should see better results on TicTacToe. Compared to the random agent, our agent performed reasonably better. Our agent performed worse compared to Veness et al. 2011.

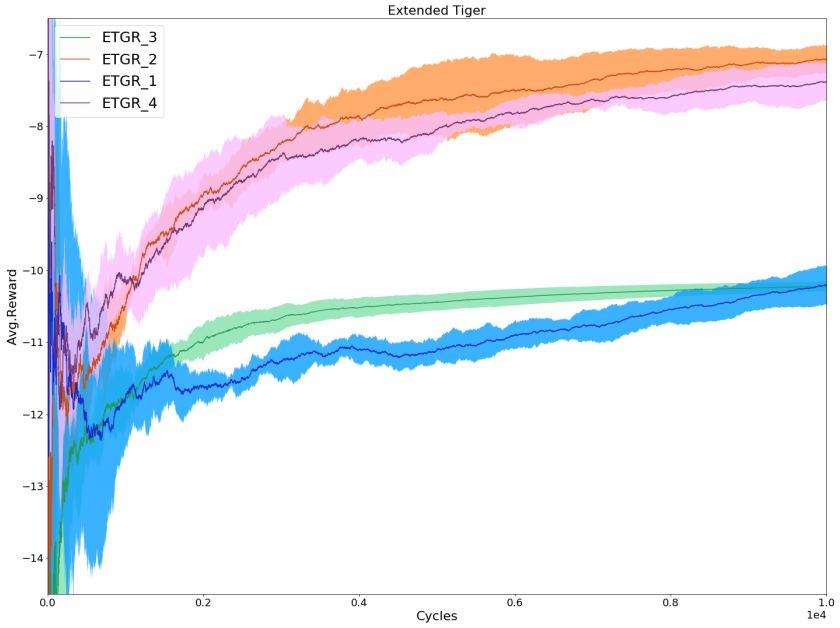


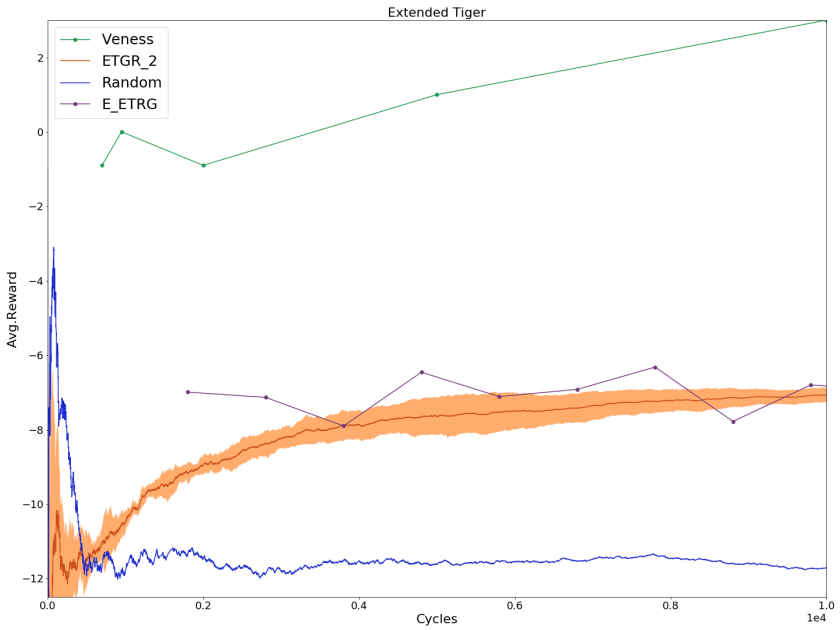


4.5 Extended Tiger

Experiment ID	D	m	ϵ	γ	Cycles	Average Reward
ETGR-1	96	4	0.999	0.99991	130	-7.77525
ETGR-2	64	4	0.999	0.999	12	-7.08699
ETGR-3	42	8	0.999	0.9991	13	-10
ETGR-4	256	4	0.999	0.9991	13	-7.519
E_ETGR	64	4	0.999	0.999	15	N/A

Extended Tiger is a hard game to learn because the configuration of the game (the respective location of the gold and the tiger) changes with each episode. In addition, the observations are stochastic. Since the agent gets large penalties for opening the door, it prefers not to open it at all. The agent is happy to just get rewards for doing valid actions. Given the extremely negative reward for opening the door with the tiger, our agent is relatively close to the optimal value, so our agent performed better than random. However, our agent performed worse compared to Veness et al. 2011.

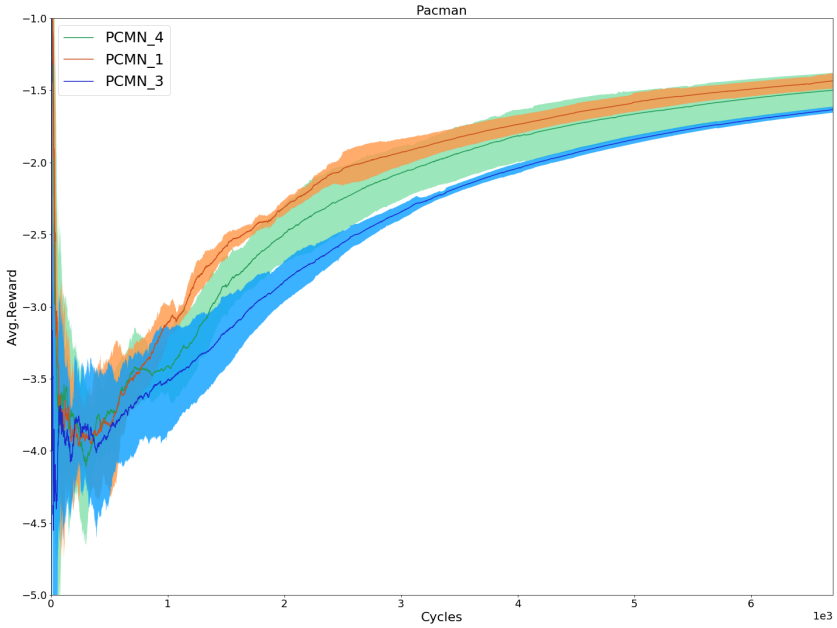


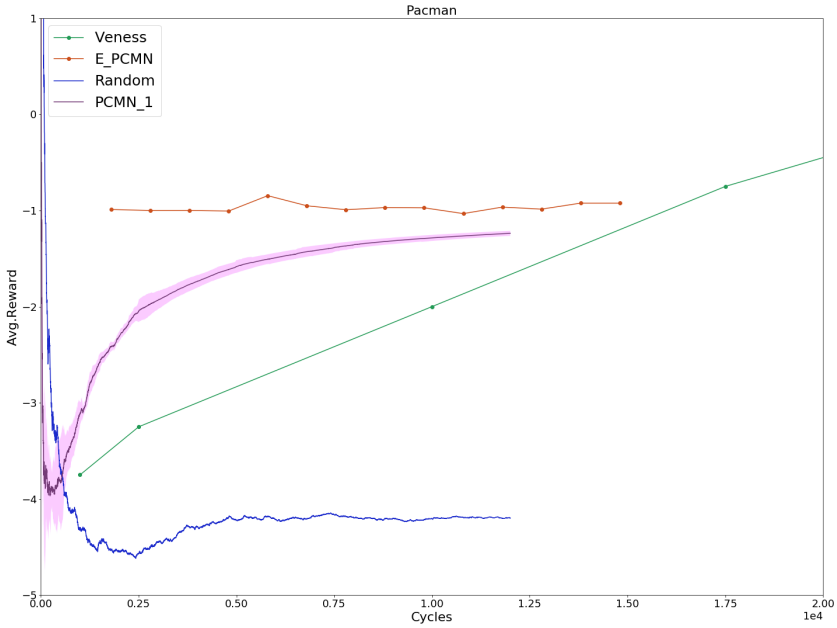


4.6 Pacman

Experiment ID	D	m	ϵ	γ	Cycles	Average Reward
PCMN-1	256	8	0.999	0.999	12	-0.994
PCMN-3	256	4	0.999	0.9991	13	-1.0055625
PCMN-4	92	4	0.999	0.999	7	-1.000225
E_PCMN	256	8	0.999	0.999	15	N/A

There are 10^{60} states in Pacman, making this the most difficult game in our set of experiments. Our results suggests that the agent learns to not run into walls and ghosts. The agent of Veness et al. 2011 trained for $250,000$ cycles and improved continuously over time. In a cycle-by-cycle comparison our agent performed better, but by the time our experiment ended our agent's performance was well below that of the fully trained agent of Veness et al. 2011.


Pacman
PCMN_4
PCMN_1
PCMN_3
Avg.Reward
Cycles
1e3

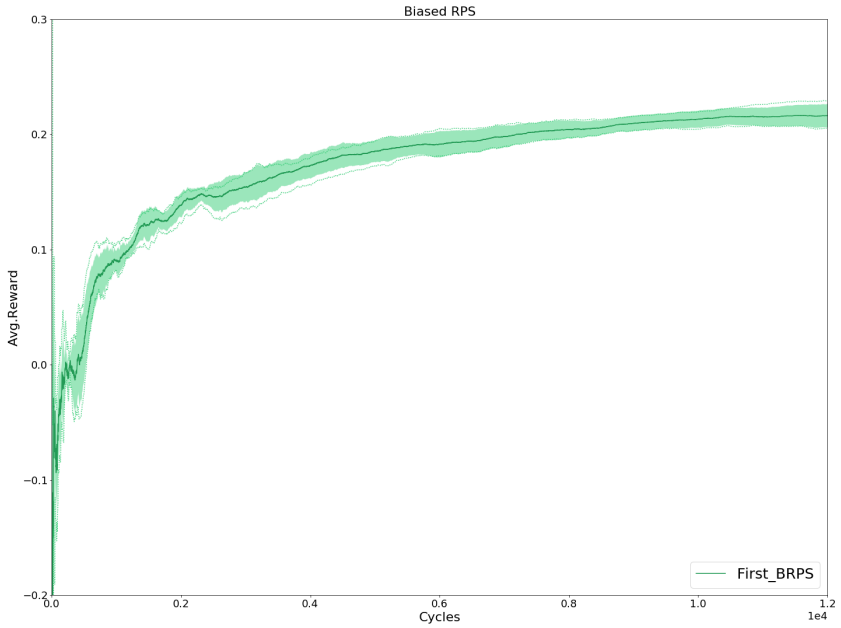

Pacman
Veness
E_PCMN
Random
PCMN_1
Avg.Reward
Cycles
1e4

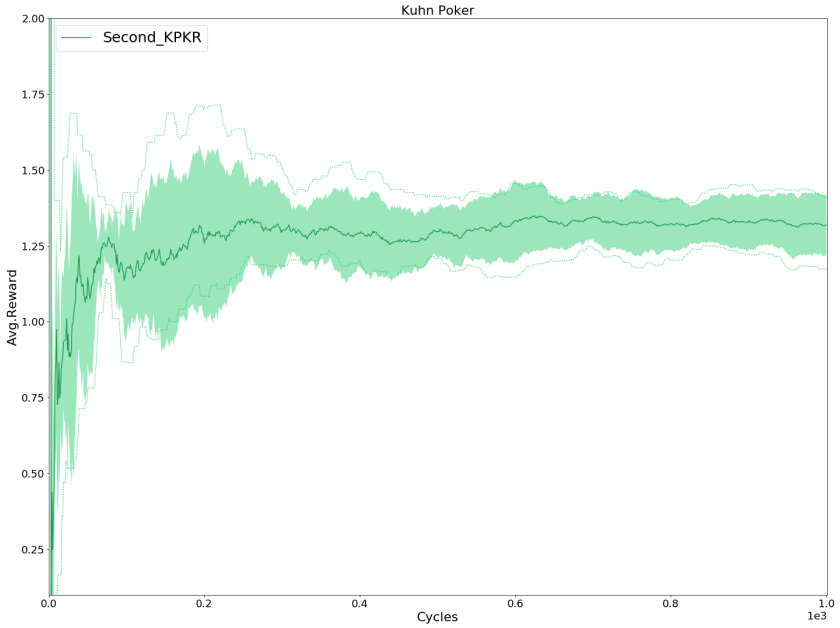
4.7 Cross Domain-Kuhn Poker and BRPS

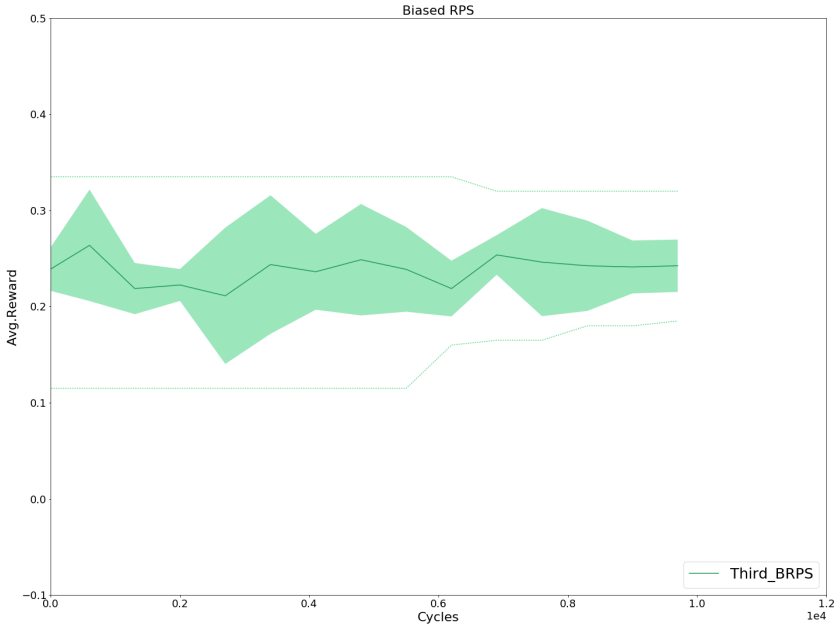
Experiment ID	D	m	ϵ	γ	Cycles (KP)	Cycles (BRPS)
BRPS_KPKR	96	8	0.9999	0.9991	12	10

We performed a cross-domain experiment on Biased Rock-Paper-Scissors (BRPS) and Kuhn Poker. We chose these domains because they are simple and because we achieved optimal performance on BRPS. The agent was first evaluated on BRPS, obtaining optimal performace as expected. Moving to Khun Poker, the agent struggled initially but later obtained the expected performance. This means that the CTW would have re-learnt the model in a short period of time. Given that the domains are entirely different, this is a remarkable result. When we moved back to BRSP our evaluation scores show that the agent performed well without any issue. This is perhaps because BRSP is really a simple game. The evidence from Khun poker might not have affected the statistics for simpler models in the CTW. Regardless, we were not able to conclude whether any information that was learned in one domain was useful in the other.

First_BRPS is the initial training. Second_KPKR is the cross domain running on Kuhn poker. Third_BRPS is the evaluation for again running on Biased Rock Paper Scissors.




Kuhn Poker
Second_KPKR
Avg.Reward
Cycles
1e3


Biased RPS
Third_BRPS
Avg.Reward
Cycles
1e4

5 Discussion

The main goal of our implementation of MC-AIXI-CTW was to achieved comparable results to (Veness et al. 2011) on our chosen domains. Due to computational constraints we did not use the exact same parameters for our agents in each environment. Regardless we were able to achieve our goal, and in each domain our agent had comparable performance.

Another one of the goals with our implementation of MC-AIXI-CTW was modularity. With this in mind we managed to create a loosely coupled system in which each part can be replaced easily. For example, if we wished to use Context Tree Switching (Veness et al. 2012) in the place of Context Tree Weighting then the implementation required is minimal.

As was mentioned in (Veness et al. 2011), the agent environment interaction history need not be in base 2. In fact, if base 3 was used then the observation space $|\mathcal{O}|$ of tictactoe would be reduced from $2^{18} = 262144$ to $3^9 = 19683$. Reducing the observation space so much would lead to a much smaller history, and ultimately a speedup for the computation. Another possible speedup could come from reward scaling. In the case of extended tiger, we could scale rewards from 0,90,99,130 to 0,9,10,13. This would reduce the number of reward bits from 8 to 4 and cause an improvement in computation time.

Lastly, we were able to implement our agent in the RoboCup simulation environment. However, in this environment our agent was unable to outperform a random agent. This was likely due to the large state space of the environment. One way in which we may we able to reduce the size of the state space of RoboCup simulation (and possibly other environments) is through state aggregation. State aggregation refers to a procedure that merges or aggregates equivalent or similar states into one. State aggregation reduces the observation (and reward) space, thereby reducing search space, thereby reducing computation time.

6 Conclusion

We implemented and tested a version of the MC-AIXI-CTW agent presented by (Veness et al. 2011). The results of Veness and colleagues were not reproduced exactly, but our results are nonetheless similar enough to substantiate their claim that MC-AIXI-CTW can learn their domains.

In all the environments we tested, except for RoboCup, our MC-AIXI-CTW agent was able to outperform a random agent. Near optimal to optimal solutions were achieved on on small domains. However, our agent ran into difficulties on larger domains, where care must be taken to manage the memory and initialize data structures as you go. This came into play especially for TicTacToe, Pac-

man and Extended tiger.

MC-AIXI-CTW is more than anything a proof of concept. The agent design may readily be improved by various methods. Our implementation of MCTS has been relatively bare-bones in that it failed to leverage the rich literature of improvements that has been developed since the technique's inception. Applying multithreading is another obvious candidate. As mentioned in the discussion, future work could also look into using Context Tree Switching in place of CTW, which we expect would lead to similar results with less computation time.

Using our modular implementation of MC-AIXI-CTW as a basis, the next step would be to develop the ideas of Veness and colleagues to their full potential as a solution to the general reinforcement learning problem.

References

[1] Marcus Hutter. *Universal AI*. 2005.

[2] Levente Kocsis and Csaba Szepesvári. "Bandit based monte-carlo planning". In: *European conference on machine learning*. Springer. 2006, pp. 282–293.

[3] Harold W Kuhn. "A simplified two-person poker". In: *Contributions to the Theory of Games* 1 (1950), pp. 97–103.

[4] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. Vol. 1. 1. MIT press Cambridge, 1998.

[5] Joel Veness et al. "A monte-carlo aixi approximation". In: *Journal of Artificial Intelligence Research* 40.1 (2011), pp. 95–142.

[6] Joel Veness et al. "Context tree switching". In: *Data Compression Conference (DCC), 2012*. IEEE. 2012, pp. 327–336.

[7] Frans MJ Willems, Yuri M Shtarkov, and Tjalling J Tjalkens. "The context-tree weighting method: Basic properties". In: *IEEE Transactions on Information Theory* 41.3 (1995), pp. 653–664.