Generalised Discount Functions applied to a Monte-Carlo AIμ Implementation

Sean Lamont*[1], John Aslanides†[1], Jan Leike‡[2], and Marcus Hutter§[1]

[1]Research School of Computer Science, Australian National University
[2]Google Deepmind, London
[2]Future of Humanity Institute, University of Oxford

Abstract

In recent years, work has been done to develop the theory of General Reinforcement Learning (GRL). However, there are few examples demonstrating the known results regarding generalised discounting. We have added to the GRL simulation platform AIXIjs the functionality to assign an agent arbitrary discount functions, and an environment which can be used to determine the effect of discounting on an agent's policy. Using this, we investigate how geometric, hyperbolic and power discounting affect an informed agent in a simple MDP. We experimentally reproduce a number of theoretical results, and discuss some related subtleties. It was found that the agent's behaviour followed what is expected theoretically, assuming appropriate parameters were chosen for the Monte-Carlo Tree Search (MCTS) planning algorithm.

Keywords— Reinforcement Learning, Discount Function, Time Consistency, Monte Carlo

1 Introduction

Reinforcement learning (RL) is a branch of artificial intelligence which is focused on designing and implementing agents that learn how to achieve a task through rewards. Most RL methods focus on one specialised area, for example the Alpha-Go program from Google Deepmind which is targeted towards the board game Go [12]. General Reinforcement Learning (GRL) is concerned with the design of agents which are effective in a wide range of environments. RL agents use a *discount function* when choosing their future actions, which controls how heavily they weight future rewards. Several theoretical results have been proven for arbitrary discount functions relating to GRL agents [8].

We present some contributions to the platform AIXIjs[1] [1][2], which enables the simulation of GRL agents for gridworld problems. Being web-based allows this platform to be used as an

*sean.a.lamont@outlook.com
†john.stewart.aslanides@gmail.com
‡leike@google.com
§marcus.hutter@anu.edu.au

[1]For a thorough introduction to AIXIjs, `aslanides.io/docs/masters_thesis.pdf`

educational tool, as it provides an understandable visual demonstration of theoretical results. In addition, it allows the testing of GRL agents in several different types of environments and scenarios, which can be used to analyze and compare models. This helps to showcase the different strengths and weaknesses among GRL agents, making it a useful tool for the GRL community in terms of demonstrating results. Our main work here is to extend this platform to arbitrary discount functions. Using this, we then compare the behaviour induced by common discount functions and compare this to what is theoretically expected.

We first provide the necessary background to understand the experiments by introducing the RL setup, agent and planning algorithms, general discounting, and AIXIjs. We then present details of the environment and agent implementation used for the analyses. Finally, we present the experiments and the results, along with a discussion for each function.

2 Background

2.1 Reinforcement Learning Setup

RL research is concerned with the design and implementation of goal-oriented agents. The characteristic approach of RL is to associate *rewards* with the desired goal and allow the *agent* to learn the best strategy for gaining rewards itself through trial and error [14]. The agent interacts with an *environment* by producing an action a, and the environment responds with an observation and reward pair $(o, r) = e$ which we call a *percept*. The *history* up to interaction cycle k is given by the string of all actions and percepts, $a_1e_1.....a_{k-1}e_{k-1}$. To simplify notation, this is written as $æ_{<k}$. Mathematically, an agent's *policy* is a stochastic function mapping a history to an action, $\pi : (\mathcal{A} \times \mathcal{E}) \rightsquigarrow \mathcal{A}$, while an environment is a stochastic map from a history and an action to a percept, $\mu : (\mathcal{A} \times \mathcal{E})^* \times \mathcal{A} \rightsquigarrow \mathcal{E}$, where $\rightsquigarrow$ is a stochastic mapping. In the context of adaptive control, Bellman [3] first introduced equations for expressing optimal policies in both deterministic and stochastic environments, including infinite state spaces. Also introduced was the idea of a *value function*. A value function is how an agent assigns value to an environment *state* (or a state-action pair), where value is a measure of the expected future discounted reward sum. To solve the Bellman equations, it is necessary to assume a fully observable *Markovian* environment (a *Markov Decision Process*, or a *MDP*). In an MDP, the agent can observe all relevant information from the environment at any time, without needing to remember the history. Although useful for MDPs, many problems of interest lack the necessary assumptions to tractably solve the Bellman equations. The problem of scaling RL to non-Markovian and *partially observable* real world domains provides the motivation for General Reinforcement Learning.

In such cases, it is useful to express the value function in terms of the agent's history, with the value of a policy π with history $_{<t}$ and environment μ given by the equation:

$$V_\mu^\pi(_{<t}) := \mathbb{E}_\mu^\pi \left[\sum_{k=t}^{\infty} \gamma_k r_k |_{<t} \right] \tag{1}$$

Where r is the reward and γ is a discount function [9]. This equation gives the μ-expected utility for a policy π . If we are in a MDP, then we can replace the history by the current state, and rewrite this as a Bellman Equation [3].

2.2 AIμ

The GRL agent AIμ [4] is purposed to find the optimal reward in a known environment. There are no other assumptions made about the environment, so this agent extends to partially observable cases. AIμ is simply defined as the agent which maximises the value function given by (1). Specifically, for any environment μ,

$$\pi^{AI\mu} \in \arg\max_{\pi} V_{\mu}^{\pi} \tag{2}$$

As there is usually no way to know the true environment, the main purpose of AIμ is to provide a theoretical upper bound for the performance of an agent for a given environment. As we wish to isolate the effect of discounting, AIμ is the agent used for our experiments to remove uncertainty in the agent's model.

2.3 Generalised Discounting

A discount function is used to weight rewards based on their temporal position relative to the current time. There are several motivations for using a discount function to determine utility, as opposed to taking an unaltered sum of rewards. In practice, a discount function allows the agent's designer to decide how it would like the agent to value rewards based on how far away they are. A discount function also serves to prevent the utility from diverging to infinity, as is the case when using undiscounted reward sums.

Samuelson [11] first introduced the model of discounted utility, with the utility at time k given by the sum of discounted future rewards:

$$V_k = \sum_{t=k}^{\infty} \gamma_{t-k} r_t \tag{3}$$

This model is the most commonly used in both RL and other disciplines, but has several issues. These include that the discount function cannot change over time, and that the value of an action is independent of the history. Hutter and Lattimore [8] address several issues with this model first by using the GRL framwork to allow decisions which consider the agent's history. They also generalise the setting to allow a change in discounting over time. Specifically, they define a discount vector γ^k for each time step k, with the entries in the vector being the discount applied at each time step $t > k$. Replacing γ_{t-k} with γ^k in (3) gives a more general model of discounted utility, as it allows the discount function to change over time by using different vectors for different time steps.

Using this model, Hutter and Lattimore [8] provide a general classification of *time inconsistent* discounting. Qualitatively, a policy is time consistent if it agrees with previous plans and time inconsistent if it does not. For example, if I plan to complete a task in 2 hours but then after 1 hour plan to do it after another 2 hours, my policy will be time inconsistent. Formally, an agent using discount vectors γ^k is time consistent iff:

$$\forall k,\ \exists a_k > 0 \ \textit{such that}\ \ \gamma_t^k = a_k \gamma_t^1, \quad \forall t \geq k \in \mathbb{N} \tag{4}$$

Which is to say, the discount applied from the current time k to the reward at time t is equal to some positive scalar multiple of the discount used for t at time 1.

Also presented in their work is a list of common discount functions and a characterisation of which of these are time consistent. These form the basis for our experiments and we present a taxonomy below:

Given the current time k, future time $t > k$, and a discount vector γ, we have:

Geometric Discounting: $\gamma_t^k = g^t, g \in (0,1)$. Geometric discounting is the most commonly used discount function, as it provides a straightforward and predictable way to value closer rewards higher. It is also convenient as for $\gamma \in (0,1)$ it ensures the expected discounted reward (i.e. value) will always be bounded, and therefore well defined in all instances. Geometric discounting is always time consistent, which is apparent when considering the definition in (4).

Hyperbolic Discounting: $\gamma_t^k = \frac{1}{(1+\kappa(t-k))^\beta}, \kappa \in \mathbb{R}^+, \beta \geq 1$. Hyperbolic discounting has been thought to accurately model human behaviour, with some research suggesting humans discount this way when deciding actions [15]. Hyperbolic discounting is time inconsistent, which is much of the reason why it is considered to model many irrational human behaviour patterns. It is clear that hyperbolic discounting is time inconsistent, as it is not possible to factor the above expression in a way which satisfies (4). Hyperbolic discounting is most commonly seen for $\beta = 1$, with $\beta > 1$ ensuring the discounted reward sum doesn't diverge to infinity.

Power Discounting: $\gamma_t^k = t^{-\beta}, \beta > 1$. Power discounting is of interest because it causes a *growing effective horizon.* This in effect causes the agent to become more far sighted over time, with future rewards becoming relatively more desirable as time progresses. This is flexible as there is no need to assign an arbitrary fixed effective horizon, it will instead grow over time. Hutter and Lattimore [8] point out that this function is time consistent, which combined with the growing effective horizon makes it an effective means of agent discounting.

2.4 Monte-Carlo Tree Search with ρUCT

Monte-Carlo Tree Search (MCTS) is a planning algorithm designed to approximate the expectimax search tree generated by (1), which is usually intractable to fully enumerate. UCT [7] is a MCTS algorithm which is effective for Markovian settings. Veness et al. [16] extend this to general environments with the ρUCT algorithm. The algortithm generates a tree comprised of two types of nodes, 'decision' nodes and 'chance' nodes. A decision node reflects the agents possible actions, while chance nodes represent the possible environment responses. A summary of the algorithm is as follows: First, plan forward using standard Monte-Carlo simulation. Then select an action in the tree using the UCB action policy; Define a search horizon m, maximum and minumum reward β and α, value estimate V', and history h, with $T(ha)$ being the number of visits to a chance node, and $T(h)$ the number of visits to a decision node. Then, for $T(ha) > 0$:

$$a_{UCB} = \arg\max_a \frac{1}{m(\beta-\alpha)} V'(ha) + C\sqrt{\frac{\log(T(h))}{T(ha)}} \tag{5}$$

If $T(ha) = 0$ then the best action will default to a. The parameter C is an exploration constant, which can be modified to control the likelihood that an agent will take an exploratory action. Veness et al. [16] remark that high values of C lead to 'bushy' and short trees, compared to low values yielding longer and more discerning trees. Once the best action is selected, the values for each node are updated backwards to the root to reflect the new action. The primary

strength of this algorithm is that it allows for history based tree search, by using ρ as the current environment model and planning based on that.

2.5 AIXIjs

We implement our experiments using AIXIjs, a JavaScript platform designed to demonstrate GRL results. AIXIjs is structured as follows: There are currently several GRL agents which have been implemented to work in different (toy) gridworld and MDP environments. Using these, there are a collection of demos which are each designed to showcase some theoretical result in GRL and are presented on the web page. Once a demo is selected, the user can choose to alter some default parameters and then run the demo. This then begins a batch simulation with the specified agent and environment for the selected number of time cycles (a batch simulation runs the whole simulation as one job, without any interference). The data collected during the simulation is then used to visualise the interaction. The API allows for anyone to design their own demos based on current agents and environments, and for new agents and environments to be added and interfaced into the system. It also includes the option to run the simulations as experiments, collecting the data relevant to the simulation and storing it in a JSON file for analysis.

The source code can be accessed on: `https://github.com/aslanides/aixijs`
While the demos can be found at: `http://aslanides.io/aixijs/`
or `http://www.hutter1.net/aixijs/`

There has been some related work in adapting GRL results to a practical setting. In particular, the Monte-Carlo AIXI approximation [16] successfully implemented a AIXI model using the aforementioned ρUCT algorithm. This agent was quite successful, even within a "challenge domain" (a modified Pac-Man game with 1060 possible states) with the agent learning several key tactics for the game and consistently improving. This example demonstrated that it is possible to effectively adapt GRL agents to a practical setting, and is the basis for the approximation of AIμ presented here.

Related to the AIXIjs platform is the REINFORCEjs web demo by Karpathy [6]. This demo implements Q-Learning [17] and SARSA [10] RL methods in a grid world scenario, as well as deep Q-Learning for two continuous state settings. The limitation of this example is its restriction to a small set of environments, with Q-Learning and SARSA being defined only for Markovian environments. These algorithms do not extend to more complicated environments or agents, which is addressed by AIXIjs.

3 Technical Details

3.1 AIμ Implementation

The agent used for experiments is an MCTS approximated version of AIμ. By using AIμ, we are removing any potential uncertainty in the agent's model which facilitates a more accurate analysis of the effect of discounting. This agent knows the true environment, so for a fixed discount this implies that its policy $\pi(s)$ will stay the same for any particular state, assuming a Markovian environment.

Although we will not be using very large tree depth, enumerating the expectimax by solving equation (1) is not generally feasible. We instead use MCTS to approximate the search tree, specifically the ρUCT algorithm introduced in the background. Although UCT would suffice in our deterministic setting, ρUCT is the default search algorithm incorporated into AIXIjs and as such was used without modification.

3.2 Agent Plan and Time Inconsistency Heuristic

We determine the agent's plan at time step k by traversing the tree created by ρUCT, first selecting the highest value decision node and then choosing the corresponding chance node with the most number of visits. In the case of the environment used here, each decision node has only one chance child as it is deterministic. The process is then repeated up to the maximum horizon reached by the search, and the sequence of actions taken are recorded as the agent's plan. The plan is recorded as a numeric string representing the sequence of actions the agent plans to take. For example, a recorded plan of 000111 indicates the agent plans to first take action '0' three times in a row and then take '1' three times.

If the action at cycle k is not equal to the action predicted by the plan at time $k-1$ then we consider this time inconsistent. Formally, if the following equation is satisfied then the action at t is time inconsistent:

$$\pi_{\gamma^{k-1}}(S_k) \neq \pi_{\gamma^k}(S_k)$$

Where $\pi_{\gamma^t}(S_t)$ is a policy π using discount vector γ^t in state S at time k and $\pi_{\gamma^{k-1}}(S_k)$ is the same policy using an older discount vector γ^{k-1}.

If this is true, then the action will be time inconsistent. If it is not true, the action may still be time inconsistent in regards to older plans. This method is used to prevent false positives, as the agent plan deep in the search tree is often not representative due to the cutoff at the horizon.

3.3 Environment Setup

The environment we use is a deterministic fully observable finite state MDP, represented by figure 1. This environment is structured to provide a simple means to differentiate myopic and far-sighted agent policies. The idea behind the environment is to give the agent the option of receiving an instant reward at any point, which it will take if it is sufficiently myopic. The other option gives a very large reward only after following a second action for N steps. If the agent is far-sighted enough, it will ignore the low instant reward and plan ahead to reach the very large reward in N time steps. Formally, the agent has 2 actions from state S_i : The first is to go to S_0 and receive a small instant reward r_I. The other takes the agent to S_{i+1} (where $i \in \mathbb{Z}/(N+1)\mathbb{Z}$) and gives very low reward $r_0 < \frac{1}{N} r_I$, and a large reward $r_L > N r_I$ for $i = N-1$. In figure 1 the straight lines represent the first action a_0 while the other lines representing the second action a_1.

4 Experiments

4.1 Overview

In this section we present the experiments for the discount functions, which were conducted using the AIXIjs experiment API mentioned in the background. In particular, we will investigate the effect of geometric, hyperbolic and power discounting on the ρUCT AIμ model. The environment used was the instance of figure 1 parametrised by $N = 6, r_I = 4, r_0 = 0, r_L = 1000$. We use average reward as our metric for agent performance. We avoid using total reward as, in this environment, it is monotonically increasing with respect to time. This would affect the scale of graphs, which could obscure an agent's behaviour. We now present the MCTS parameters, after which we detail two specific policies prior to the experiments which comprise the rest of the section. We introduce these policies to avoid unnecessary overlap in the analysis of geometric and hyperbolic discounting, as they displayed very similar behaviours.

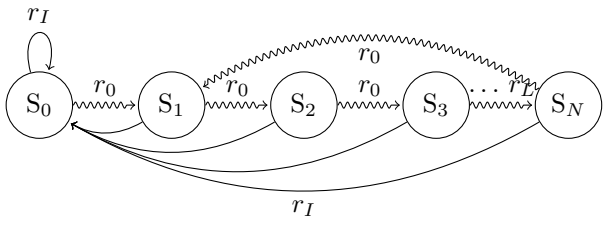

Figure 1: MDP used to conduct discounting experiments

4.2 MCTS Parameters

	Horizon	UCB Parameter	Samples
Geometric	10	0.01	10 000
Hyperbolic	10	0.01	10 000
Power	7	0.001	100 000

Figure 2: MCTS Parameters used for each discount function

It was necessary to increase the samples and lower the exploration constant for power discounting because over time, the discount factor becomes exponentially lower with respect to β. A high exploration constant would overpower the UCB expression in (5) and result in erratic policies as there is no clear better action. Given the large number of samples, it was also necessary to reduce the horizon to shorten the depth of the tree. 7 is the minimum required to see far enough into the future to notice the delayed reward.

4.3 Far-Sighted Policy

With reference to figure 1, this policy takes action a_1 (the alternating arrow) for every time step. The total reward for the far-sighted policy in 200 time cycles is 33 000, given a delayed reward of 1000 and a reward interval of 6 time steps. Figure 3 presents a plot of the average reward of this policy in our environment.

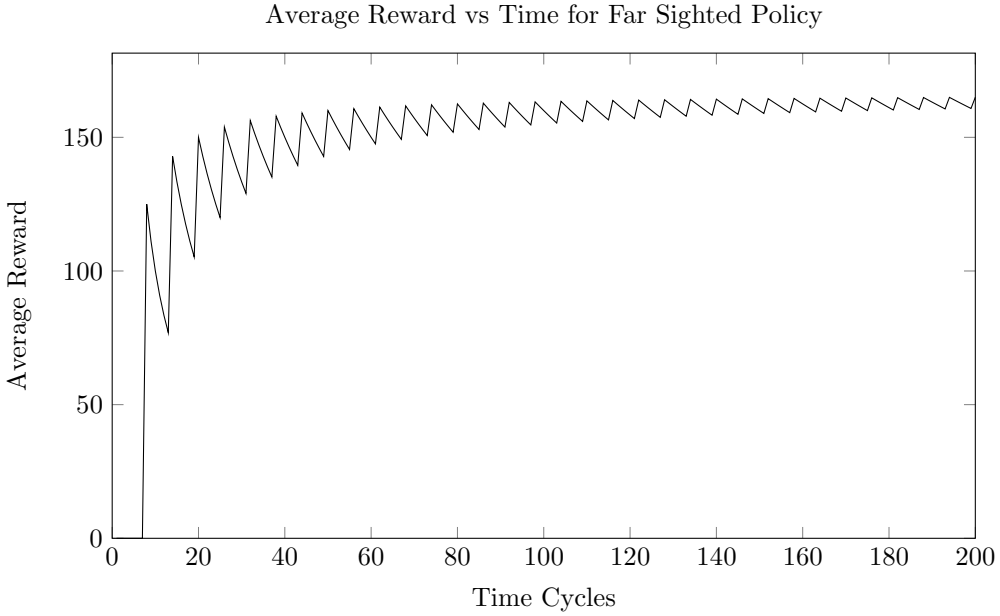

Figure 3: Average reward versus time cycles for a far-sighted agent policy

The periodic nature of the delayed reward is reflected in the zig zag shape of the average reward graph. As this policy is consistently taking a_1, the time between spikes is constant.

4.4 Short-Sighted (Myopic) Agent

The second policy takes action a_1 (solid arrow in figure 1) for every time step. The total reward for this policy is 792, given an instant reward of 4. Figure 4 presents a plot of the average reward of this policy in our environment.

The graph reflects the initial reward of 0 as the agent starts off, and then the constant reward of 4 every following time cycle.

4.5 Geometric Discounting

We ran experiments by altering γ in increments of 0.1, ranging from 0.1 to 1.0. We found that in all test runs, the number of time inconsistent actions given by our heuristic was 0. We found that for $\gamma \leq 0.4$ the agent followed exactly the myopic policy from the previous subsection,

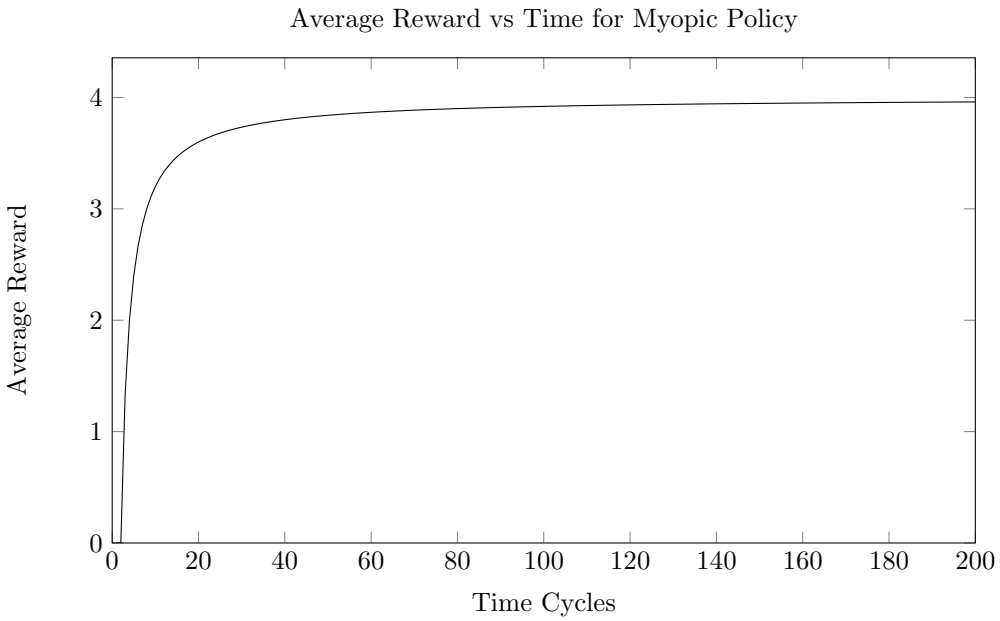

Figure 4: Average reward versus time cycles for a myopic agent policy

receiving a total reward of 792. For $\gamma \geq 0.6$ the agent behaved as described in the far-sighted policy subsection, achieving the optimal reward of 33 000. For $\gamma = 0.5$, the agent behaved somewhat erratically, occasionally altering its behaviour between both policies. As this value lies between the γ which cause strict far/short sightedness, there would be a small difference in weighted rewards between both policies. It is therefore likely the erratic behaviour is caused by the MCTS struggling to find the best decision, given there is a degree of innacuracy in the tree search. The agent plan enumeration gave a consistent plan of 0000000000 for $\gamma \leq 0.4$ and varied between 1111100000 and 1111111111 for $\gamma \geq 0.6$, where '0' and '1' are shorthand for a_0 and a_1 respectively. This variation is due to the horizon cutoff at 10, since at some points the agent won't see far ahead enough to plan for 2 far-sighted actions.

4.6 Hyperbolic Discounting

We varied κ between 1.0 and 3.0 in increments of 0.2, and kept β constant at 1.0. We found that only $\kappa = 1.8$ yielded non-zero time inconsistent actions, with the total number of such actions recorded as 200. We found for $\kappa \leq 1.8$ that the agent followed exactly the behaviour from the myopic policy subsection, receiving a total reward of 792. For $\kappa > 1.8$ the agent behaved as described in the far-sighted policy subsection, achieving the optimal reward of 33 000. The plan for $\kappa = 1.8$ was 0111111000 at every time step. For $\kappa > 1.8$ the plan stayed as 1111111111, and $\kappa \leq 1.8$ gave a constant plan of 0000000000.

In the interest of reproducibility, the experiments for hyperbolic discounting were performed on commit 3911d73 on the provided github link. The results can also be replicated with recent

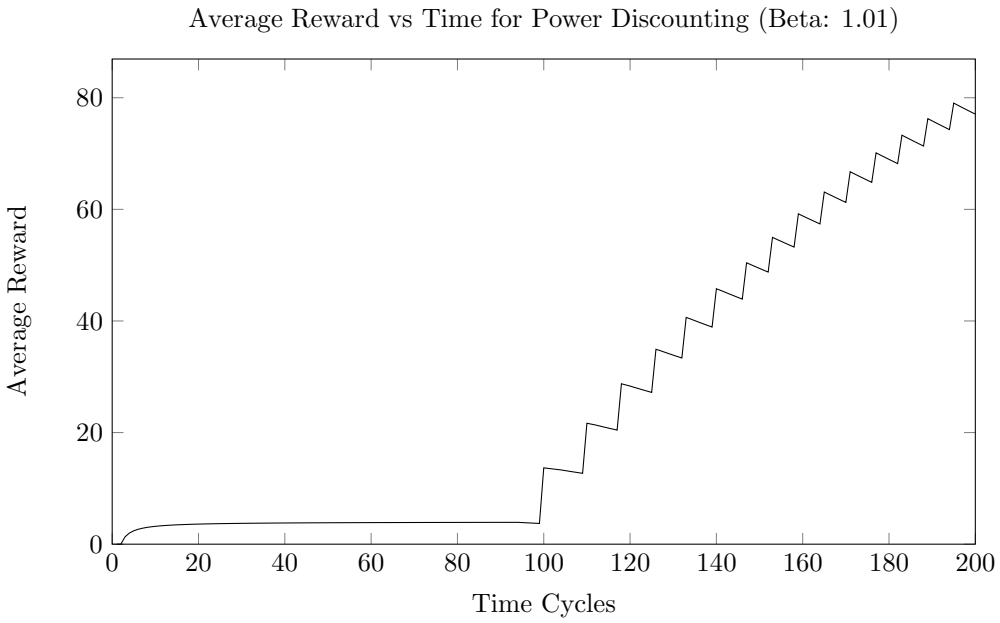

Figure 5: Average reward versus time cycles for power discounting

and future versions, however the MCTS parameters may need to be adjusted.

4.7 Power Discounting

We only used a single β value in this case, with β= 1.01. We note that any change in β would result in similar behaviour, with only the length and time between these stages changing (hence we need only present the results of one β value). No time inconsistent actions were detected for this function. The total reward obtained by the agent was 15412.

The behaviour in this circumstance follows three stages: For around 100 time steps, the agent behaves completely myopically, reflected by the small continuous rise in the first half of the graph. The discount function then reaches a stage where distant rewards are weighted high enough so that the agent decides to act far-sightedly. For several time steps, the agent collects the delayed reward then goes to the instant one for a few cycles. The number of cycles it stays there gradually decreases until it strictly follows the far-sighted policy. This can be seen in the graph, as the intervals between peaks are larger from cycles 100-150 than 150-200 when the agent acts completely far-sighted.

5 Discussion

In regards to time inconsistent agent behaviour, the results were consistent with theoretical predictions. Geometric discounting was, for all instances of γ, time consistent as expected.

Somewhat suprisingly, hyperbolic discounting was time consistent for all measured κ except 1.8 when it was continuously acting inconsistently. The results of power discounting also lacked any time inconsistent actions which is expected.

The hyperbolic agent plan of 0111111000 for $\kappa = 1.8$ reflects some interesting behaviour. We can see the agent is planning to stay at the instant reward for the next time step, and then move off to collect a delayed reward. But as this plan is the same for all time steps, the agent continuously stays on the instant reward planning to do the better long term action later. In effect, the agent is eternally procrastinating. The fact that this behaviour can be induced with this function also supports the claim that hyperbolic discounting can model certain irrational human behaviour. We note the trailing 0s are an artifact of the horizon being too low to see far ahead enough to notice another delayed reward, given the horizon was only 10.

The results of power discounting clearly demonstrate how a growing effective horizon can effect an agent's policy. Initially the agent is too short sighted to collect the delayed reward, but over time this reward becomes more heavily weighted compared to the instant reward. After some time the agent starts to collect the delayed reward and soon is fixed to a far-sighted policy. This shows that a growing effective horizon can cause an agent to collect distant rewards only after some time has passed, which again reflects what is theoretically predicted.

There will continue to be new results proven for GRL, so an avenue for future work would be to demonstrate those results in a similar fashion to the work presented here. Our contributions to the AIXIjs framework would allow for this to be done easily for results pertaining to agent discounting. Other future work would be the development of practical GRL systems which extend beyond a toy environment, and which can be used for non-trivial tasks.

6 Summary

We have adapted the platform AIXIjs to include arbitrary discount functions. Using this, we were able to isolate time inconsistent behaviour and illustrate the effect of the discount function on an agent's farsightedness. We were able to show it is possible to use power discounting in a concrete setting to observe the impact of a growing effective horizon, which influenced the time at which an agent chose to collect distant rewards. We also demonstrated that hyperbolic discounting can induce procrastinating behaviour in an agent. Our current framework now permits a larger class of experiments and demos with general discounting, which will be useful for future research on the topic.

References

[1] J Aslanides. AIXIjs: A software demo for general reinforcement learning, Australian National University, 2016.

[2] J Aslanides and M. Hutter and J. Leike., General Reinforcement Learning Algorithms: Survey and Experiments, 2016. http://www.hutter1.net/official/bib.htm#grlsurexp

[3] R Bellman. Dynamic programming. *Princeton, NJ: Princeton University Press*, 1957.

[4] M. Hutter. A theory of universal artificial intelligence based on algorithmic complexity. *ISDIA-14-00, ISDIA, arXiv:cs.AI/0004001*, 2000.

[5] M. Hutter. Universal artificial intelligence: Sequential decisions based on algorithmic probability. *Springer*, 2005.

[6] A. Karpathy. Reinforcejs, 2015. https://cs.stanford.edu/people/karpathy/reinforcejs /index.html.

[7] L. Kocsis and C. Szepesvari. Bandit based Monte-Carlo planning. *Euro. Conf. Mach. Learn. Berlin, Germany : Springer, pp. 282-293.*, 2006.

[8] T. Lattimore and M. Hutter. General time consistent discounting. *Theoretical Computer Science, 519:140-154*, 2014.

[9] J. Leike. What is AIXI? - An Introduction to General Reinforcement Learning, 2015. https://jan.leike.name/AIXI.html.

[10] G. A. Rummery and M. Niranjan. On-line Q-learning using connectionist systems. *Technical Report CUED/F-INFENG/TR 166, Cambridge University Engineering Department*, 1994.

[11] P. Samuelson. A note on measurement of utility. *The Review of Economic Studies, 4(2) : 155-161*, 1937.

[12] D. Sliver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, N. Kalchbrenner, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature 529, 484-489*, 2016.

[13] R. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning, 3, 9-44.*, 1988.

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

[15] R. Thaler. Some empirical evidence on dynamic inconsistency. *Economics Letters, 8(3) : 201 - 207*, 1981.

[16] J. Veness, M. Hutter, W. Uther, D. Silver, and K. S. Ng. A Monte-Carlo AIXI Approximation. *Journal of Artificial Intelligence Research 40: 95-142*, 2011.

[17] C. Watkins and P. Dayan. Q-learning. *Machine Learning, 8, 279-292*, 1992.