



DEGREE PROJECT IN ELECTRONICS AND COMPUTER
ENGINEERING,
FIRST CYCLE, 15 CREDITS
STOCKHOLM, SWEDEN 2019

Learning comparison: Reinforcement Learning vs Inverse Reinforcement Learning

How well does inverse reinforcement learning perform in simple markov decision processes in comparison to reinforcement learning?

PABLO IZQUIERDO AYALA

Jämförelse mellan förstärkningsinlärning och inverterad förstärkningsinlärning

PABLO IZQUIERDO AYALA

Bachelor in Computer Science

Date: June 12, 2019

Supervisor: Jörg Conradt

Examiner: Örjan Ekeberg

School of Electrical Engineering and Computer Science

Abstract

This research project elaborates a qualitative comparison between two different learning approaches, Reinforcement Learning (RL) and Inverse Reinforcement Learning (IRL) over the Gridworld Markov Decision Process. The interest focus will be set on the second learning paradigm, IRL, as it is considered to be relatively new and little work has been developed in this field of study. As observed, RL outperforms IRL, obtaining a correct solution in all the different scenarios studied. However, the behaviour of the IRL algorithms can be improved and this will be shown and analyzed as part of the scope.

Abstrakt

Denna studie är en kvalitativ jämförelse mellan två olika inlärningsangreppssätt, "Reinforcement Learning" (RL) och "Inverse Reinforcement Learning" (IRL), om använder "Gridworld", en "Markov Decision-Process". Fokus ligger på den senare algoritmen, IRL, eftersom den anses relativt ny och få studier har i nuläget gjorts kring den. I studien är RL mer fördelaktig än IRL, som skapar en korrekt lösning i alla olika scenarier som presenteras i studien. Beteendet hos IRL-algoritmen kan dock förbättras vilket också visas och analyseras i denna studie.

Contents

1	Introduction	6
1.1	Research Question	7
1.1.1	Motivation	7
1.2	Approach	7
2	Background	8
2.1	Markov Decision Process (MDP)	8
2.1.1	Gridworld	9
2.2	Reinforcement Learning (RL)	10
2.2.1	Model based	10
2.2.2	Model free	11
2.2.3	Value Function	11
2.2.4	Action-Value Function	11
2.2.5	Bellman Equation	12
2.3	Inverse Reinforcement Learning (IRL)	12
2.3.1	Linear Programming	13
2.3.2	Maximum Entropy	13
3	Method	15
3.1	Model description and selection criteria	15
3.2	Implementation	16
3.3	Evaluation	17
4	Results and Analysis	18
4.1	First Experiment	18
4.1.1	IRL - Linear Programming	19
4.1.2	IRL - Maximum Entropy	20
4.1.3	RL - Value Iteration	21
4.1.4	Running time between IRL algorithms	21
4.2	Second Experiment	22

4.2.1	3x3 grid	22
4.2.2	6x6 grid	23
4.2.3	9x9 grid	24
5	Discussion	25
5.1	Hypothesis	25
5.2	First Experiment - Discussion	25
5.3	Second Experiment - Discussion	28
6	Conclusion	31

Chapter 1

Introduction

In a world with infinite different scenarios, being able to generate an accurate model that represents a real problem is not a simple task. Sometimes, the information at hand might not suffice, whilst others, the complexity of the model will make it so that an excess of information is irrelevant. This leads to our first thought, how can an accurate model be elaborated? Can a behaviour be mimicked, can it be parametrized?

This is one of many challenges faced in the Machine Learning field of study and, as such, different methodologies have been used throughout the years, some with success. One of them is presented as Reinforcement Learning, which at the same time defines a whole field of study, and will comprise the first part of this research project:

Reinforcement learning (RL) stems from the idea that, given a scenario, we know the rewards that will derive from performing certain actions and we want to discover what the total reward will be. The problem then becomes an iterative task of trying to maximize this overall reward through an explicitly given reward function. Through trial and error, our agent will learn what the optimal policy is in the given environment. In the last years, this learning paradigm has proven extremely useful and has been applied to many different areas, i.e. classical board games (the chinese game of Go was beaten in late 2016 by AlphaGo, developed by Deepmind [1]) or optimizing memory control. [2][3].

However, elaborating a model that can provide an accurate representation of a scenario is not always possible or relevant. Sometimes it is not the output but the underlying behaviour what is interesting. Sometimes the scenario as

such might be too complex to replicate. To face these challenges, a relatively new sub-field of Machine Learning (and directly linked with Reinforcement Learning) has been described, Inverse Reinforcement Learning:

Inverse Reinforcement learning (IRL), as the name might suggest, stems from the complementary concept to RL. Instead of having a reward function and trying to maximize its outcome, we try to obtain a reward function based on the behaviour of an agent that follows an optimal policy in the given environment. By attempting to understand the reasons behind this underlying behaviour that drives the agent into performing different actions, we may be able to reach a deeper perception of the interaction of our agent in the environment.

In this research project, different versions of both learning approaches will be implemented and tested over the same environment. To be analyzed is how they will both behave and which conclusions can be drawn from this process.

1.1 Research Question

The research question that will be answered in this paper is:

How well does inverse reinforcement learning perform in simple markov decision processes in comparison to reinforcement learning?

1.1.1 Motivation

The field of Inverse Reinforcement Learning is relatively new. Deepening my understanding in Reinforcement Learning and being able to broaden my knowledge in an upcoming study field, as well as elaborating a comparison that might be used in the future, is the main motivation behind this project choice and research question.

1.2 Approach

Initially, different versions of both learning approaches will be implemented and tested on the same environment. Each of these algorithms will output a value map which will be used to analyze and compare the results obtained.

Chapter 2

Background

In order to understand this research project, three different terms introduced previously must be formally defined:

- **Markov Decision Process (MDP)**
- **Reinforcement Learning (RL)**
- **Inverse Reinforcement Learning (IRL)**

(Most of these concepts can be found in the book by R.S.Sutton and A.G.Barto.[4])

2.1 Markov Decision Process (MDP)

In both Reinforcement Learning and Inverse Reinforcement learning, our agent will act inside an environment, known as a Markov Decision Process or MDP. This environment can be described following formal notation[5] as a tuple $(\mathcal{S}, \mathcal{A}, \{P_{ss'}^a\}, R, \mathcal{R}, \gamma)$ where:

- $t \in N_{\geq 0}$ is a time step.
- \mathcal{S} is the set of possible states.
- \mathcal{A} is the set of possible actions the agent can perform.
- $\{P_{ss'}^a\}$ is the set of transition probabilities. Each transition probability is defined as $P_{ss'}^a = P(s, a, s')$ ($s \in \mathcal{S}, a \in \mathcal{A}, s' \in \mathcal{S}$), which can be formally described as the chance of reaching a state s' from a state s given an action a .

- R is the reward function. It evaluates how much reward should be obtained by taking a specific action a from a given state s .
- \mathcal{R} is the set of rewards, where $\mathcal{R}_i = R(s_i)$
- γ is the reward discount parameter. This value determines the size of a reward based on the time moment it is obtained. As an example, given a discount of $\gamma = 0.5$, obtaining a reward in t_1 of 2 is equivalent to obtaining a reward in t_2 of 4.

The total reward (or long-term reward) is the sum of all rewards obtained from t_0 to t_n . Several other definitions that are relevant for this research project are:

- A policy is the term that defines the behaviour of the agent on the MDP. Under a policy and given a state s , the agent will perform an action a . We consider an optimal policy π^* to be the policy that maximizes the long-term reward.
- A policy may create different paths. A path is a list of tuples (state,action) which describes a possible route of action an agent may follow inside the MDP.
- A state can be described as a feature vector. To do so, a feature map is created, $\phi : \mathcal{S} \rightarrow \mathbb{R}^D$, where the set of states \mathcal{S} is mapped into D dimensions (D being a natural number). $\phi(s)$ is then the representation of state s in this D -dimensional space.

/

2.1.1 Gridworld

All the experiments that we performed in this project were run over the Gridworld MDP. This MDP is an $N \times N$ grid, $N \in \mathbb{N}$, where every cell is a possible state (See figure 2.1). An agent can perform 4 different actions: Moving up, down, left, right. The agent also has a chance of moving to a random cell. If the agent tries to perform a movement that sets itself off the grid, it will instead remain on its place. The objective of the agent in this Gridworld is to reach an end-state, which is a cell that contains a positive reward. This end-state can be placed in any cell on the grid and a grid may contain more than 1 end-states.

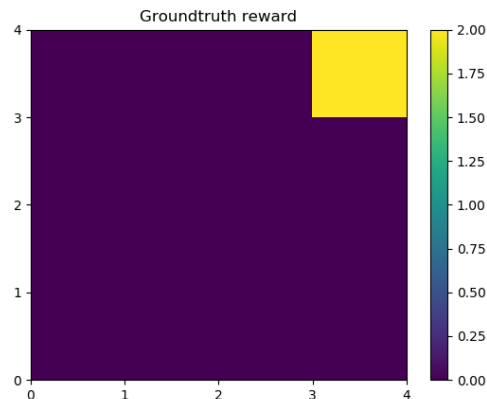


Figure 2.1: *Example of a 4x4 GridWorld with 1 end-state. The right indicator is a colour code to show the obtained reward on each state. All the states are set to zero except for the end-state located on the top right corner with a reward of positive 2.*

2.2 Reinforcement Learning (RL)

Reinforcement Learning is a learning approach that tries to model how an agent should behave in an environment in order to maximize a final reward. This agent does not know which actions it should take, but instead discovers the optimal ones through trial and error. It is important to note that, in order to perform an action, the agent must be able to identify which state he is in inside the given environment and what its goal is, making an MDP the ideal testing environment.

In order to find the optimal policy, there are two different types of algorithms: Model based and Model free. There are also 3 key concepts that must be defined in order to understand the basics of Reinforcement Learning: Value Function, Action-Value Function and Bellman Equation. These will be described in the following sections.

2.2.1 Model based

A model based algorithm performs over an estimate of the environment. This means that the possible actions (and the probability of each action) an agent can perform on each state are explicit or can be estimated.

This definition is meaningful, as it implies that the optimal value function (see subsection 2.2.3) and action value function (see subsection 2.2.4) can both be obtained for every state.

2.2.2 Model free

Analogously to Model based, a Model free algorithm does not rely on a previously defined model. (An example is Q-learning, where the algorithm estimates the optimal Q-values of each action in each state in order to find the optimal Q-function[6].)

2.2.3 Value Function

The value function is formally defined as:

$$V^\pi(s) = E_\pi[\sum_{i=0}^{\infty} \gamma^i R(s_{t+i+1}) | s_t = s]$$

Given a policy π , this function returns the total reward that can be expected by an agent that follows said policy from a starting state s (this reward contains the discount factor γ described in section 2.1) This function fulfills the Bellman equation¹ for any policy and state.

The optimal value function V_* can be defined as

$$V_*(s) = \max_\pi v_\pi(s)$$

for all $s \in S$

2.2.4 Action-Value Function

The action-value function is formally defined as:

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

Similar to the value function, the action-value function returns the total discounted reward expected by an agent that takes action a from state s and then follows the policy π .

The optimal action-value function can now be defined as

$$Q_*(s, a) = \max_\pi q_\pi(s, a)$$

¹This equation was formulated by R.Bellman in 1957 and is the basis of dynamic programming. See 2.2.5

for all $s \in S$ and $a \in A(s)$

The action-value function can be expressed in terms of the value function described previously as

$$Q^\pi(s, a) = E_\pi[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a]$$

2.2.5 Bellman Equation

The Bellman equation is a recursive function that is key to solve an MDP. Through it we can express the values of a state as values of posterior states, making it possible to approach this problem as an iterative question. Furthermore, the previously described equations fulfill this recursive relationship and can be expressed as:

$$V^\pi(s) = R(s) + \gamma \sum_{s'} P_{s\pi(s)}(s') V^\pi(s')$$

for value iteration and

$$Q^\pi(s, a) = R(s) + \gamma \sum_{s'} P_{sa(s)}(s') V^\pi(s')$$

for action value iteration.

In this project we will be utilizing a Model based algorithm defined as Value iteration. This algorithm uses the Bellman equation to calculate the optimal value for each cell in the MDP and then outputs the obtained value map.

2.3 Inverse Reinforcement Learning (IRL)

Inverse Reinforcement Learning is a learning approach which consists in trying to find the optimal reward function given observed optimal behaviour. It can be compared to Apprenticeship Learning (AL), as both learning approaches are based on the understanding of the behaviour of an optimal agent. However, IRL seeks to obtain the implicit reward function to then develop the correct optimal policy, whereas AL's goal is to obtain that optimal policy directly from the observed patterns.

As described by Ng. and Russel[7], IRL is extremely useful in certain fields such as econometrics or theoretical biology. An example would be the pollination process: In order to model the behaviour of bees, an RL approach assumes that a reward, consisting of the amount of nectar, is set on each flower. In reality, the accurate reward function is much more intricate, as a bee might

consider other values such as flight distance, time, risk from wind, etc, as part of its own reward function in order to take its decisions. It is also more relevant, as knowing how many flowers can be polinized by a bee might not be as important for a biologist as understanding the underlying motivation that leads the bee into taking such actions.

There are three main algorithms that follow this learning approach. In this research project we will cover two of them, Linear Programming and Maximum Entropy. In both algorithms, the optimal policy will be obtained from a set of sampled paths, sampled from an initial state, s_0 .

2.3.1 Linear Programming

This variant of IRL was first introduced in the paper cited earlier by Ng. and Russell[7]. In the original paper, the authors describe two different versions of the algorithm, one for finite state spaces and a different one for infinite or large state spaces. In this research project we will only cover the first version. This implementation requires an optimal policy π^* as well as the transition probabilities $P_{ss'}^a$.

In their paper, the full algorithm is described and proven. In order to reduce the complexity of this research project, the key concepts behind it will be simplified and described. Their findings can be defined as follows:

This problem can be treated as a linear programming problem. To do so, several constraints must be set over the reward domain. First, the reward domain must be restricted, as there are infinite possible rewards that would still validate the proof. This multiplicity problem is first addressed by forcing the reward to be positive and later on by adding an L1-norm penalty term to prioritize "simpler" rewards over more complex ones. Once this is established, the execution of the algorithm consists in trying to deviate as little as possible from the optimal policy, whilst obtaining the maximum possible total reward. (*Full proof can be found in section 3 of their paper.*)

2.3.2 Maximum Entropy

This second IRL algorithm approaches the same problem in a different manner. This approach will be simplified and the key aspects will be explained:

First, and as described in the MDP section (see section 2.1), a state can be

mapped into a feature vector of D dimensions. A path ζ can then be described as the sum of all the feature vectors of the states that form it.

$$\phi_\zeta = \sum_{(s_i, a_i) \in \zeta} \phi(s_i)$$

This is formally known as feature counts. When having multiple paths, this feature counts can be averaged, obtaining a new term, the feature expectations.

$$\tilde{\phi} = \sum_{\text{all paths}} P(\zeta) \phi_\zeta$$

To initialize this algorithm, multiple paths are sampled. The authors first demonstrate that the feature expectations of an observed set of paths should match the feature expectations of a true optimal behaviour. Then they include the principle of Maximum Entropy to increase the likelihood that paths yielding a higher reward are chosen, as there may be many paths that can match the feature expectations without being optimal. This problem can now be approached as a dynamic programming problem, the output being the expected visitation frequency of each state.

(Full proof and description of the algorithm can be found in their paper [8].)

Chapter 3

Method

3.1 Model description and selection criteria

The environment used for this project is the Gridworld MDP described in the previous section (see 2.1.1).

Two different experiments will be performed over this environment:

The **first experiment** will be performed in an increasing size grid, from a small representation (2x2) to a much more bigger version (10x10). The objective of this MDP will be reaching the top right cell, (N-1,N-1), which will contain a single positive reward. All the remaining cells will contain a reward of zero. The idea behind this experiment is trying to understand the difference between short term and long term goals. In real life, having a goal that is far away from our current stand point might result in us acting in a different way. This behaviour should be mimicked by this experiment.

The **second experiment** will consist in using different discount parameters of increasing size over different grid configurations. The grid configurations were chosen evenly to obtain a uniform example. The aim of this experiment can be explained by making an analogy to the real life scenario described for experiment 1. If a goal is further away from our current stand point, it might be harder for us as agents to identify the optimal move. By increasing the discount parameter, we increase the penalty caused by an incorrect move or a sub-optimal decision. This should make paths that were not that different from one another to diverge greatly, as some of them may contain decisions that could be defined as trivial in experiment 1 but are now key to obtaining

an optimal result.

The algorithms will have a set of fixed parameters.

- Random factor of 0.1
- γ of 0.01 (First experiment)
- Length of the sampled paths of $2 \times$ size of the grid for the IRL algorithms
- Reward on the top-right square will always be set to 1.
- Learning rate of 0.01

It is relevant to note that comparing RL and IRL is not a simple task, as both algorithms have different objectives. However, both learning approaches can perform over the same environment, so the research question then pivots into trying to understand what motivates the output given by these algorithms and if it can be improved. In the case of IRL, a first step will be evaluating whether the algorithms are capable of extracting the underlying reward function and value map, as well as if this value map can be influenced and in which way. In the case of RL, by increasing the size of the grid and the discount factor, we might also observe a change in the output of the given algorithm that might be interesting for the discussion.

3.2 Implementation

This project is mainly created using versions 2.7 and 3.6 of python.

There are several algorithms that belong to the RL paradigm. The value iteration algorithm was chosen, as the value map output is compatible with the type of output that will be used for the latter comparison made in this project. The implementation of both the MDP and the value iteration algorithm was based on the book by Sutton and Barto[4].

IRL is a considerably recent field of study, as the initial reference paper is dated on the year 2000. Therefore there is no abundance of resources. There are three main algorithms that belong to this paradigm: Linear Programming, Maximum Entropy and Deep Maximum Entropy. Due to the complexity of these algorithms, only the Linear Programming version and the classical Maximum Entropy version were used. The implementation of the Linear Programming version is based on the paper by Ng. and Russell[7], the Maximum

Entropy algorithm is based on the paper written by several members of the Carnegie Mellon University[8]. After testing different frameworks unsuccessfully (minor errors were found in the code that would invalidate our results), one consistent framework was found and will be used as the basis for the development of this research. This framework is provided by M.Alger[9] and satisfies the requisites for this project (it has a consistent output, the code is well written and understandable and can be scaled properly)

3.3 Evaluation

The evaluation will be qualitative. The value map for the RL algorithm will be plotted. Originally, the IRL algorithms output a reward map and a transition probability distribution map for the MDP. In order to make an homogeneous comparison, this transition probability map will then used to generate the corresponding equivalent value map that will be shown for the analogous IRL versions over increasing sized grids. After an initial hypothesis, the results will be described and explained.

Afterwards, a second experiment will be conducted. In this experiment the discount factor will vary. The evaluation will again be qualitative, understanding the difference between plots and the reasons behind it, as well as the impact it has in the outcome of the algorithms.

In both cases, the value map shown will contain a colour grading. The colour of a cell represents the value of that specific cell. This value is part of a value gradient that can be seen on the right hand side of every grid. The lighter the colour, the higher the value. However, the value per se is not relevant, as we will be analyzing the gradient pattern that should be obtained as the output of the different algorithms. This is part of our hypothesis and will be described in detail in the upcoming section.

As an important side note, both IRL algorithms perform over different time spans. This will be measured by timing the execution of both algorithms and reported at the end of the first experiment. The objective is illustrating that, even though both algorithms follow a similar principle, under our implementation it might be more useful to employ one of them due to time constraints.

Chapter 4

Results and Analysis

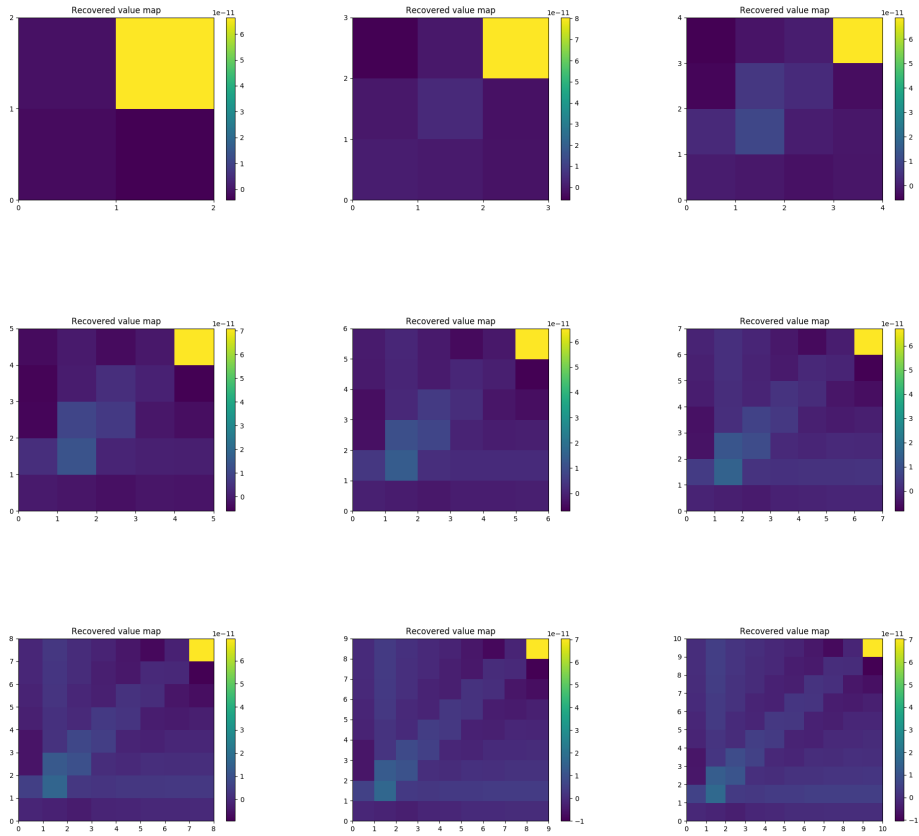
4.1 First Experiment

The varying factor will be the size of the $N \times N$ grid. For this experiment we will iterate from $N=2$ to $N=10$ on steps of 1.

The results will be displayed as a coloured value map grid. The colour of a cell defines the value mapped to that specific cell. The value gradient can be found on the right side of each grid, going from 0 as a dark blue to a light yellow for the highest value. The grids will be distributed in blocks of 9 (9 grids per page). From first grid on the top left will belong to the 2×2 grid, the last grid on the bottom right side will belong to the 10×10 grid.

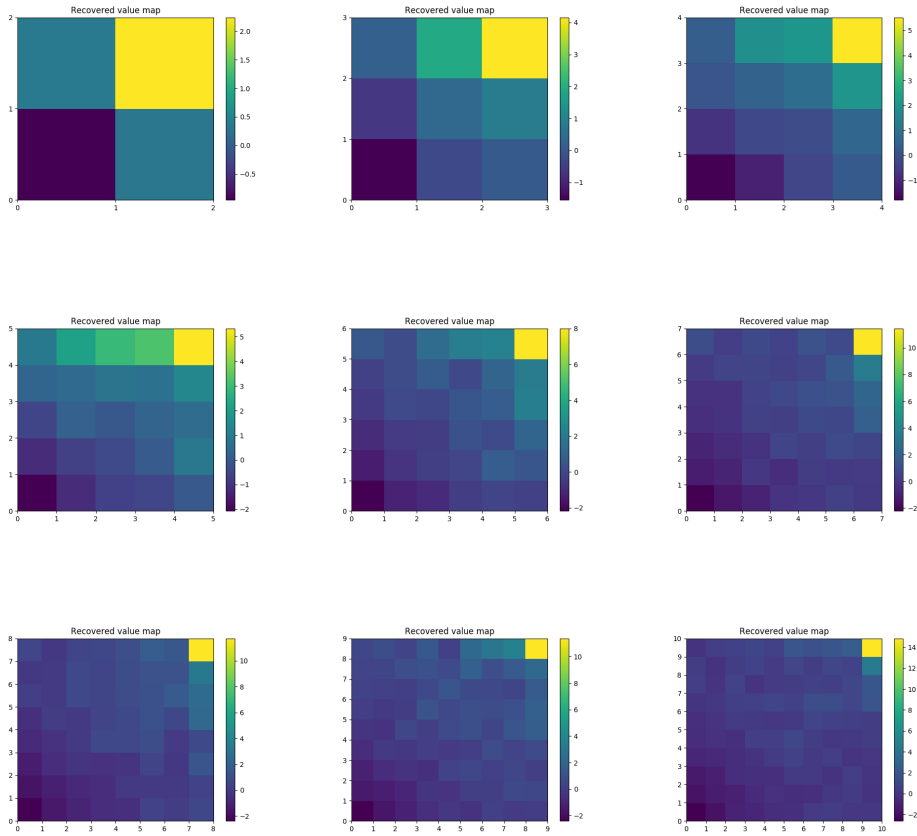
4.1.1 IRL - Linear Programming

Representation of the value map of different sized grids, ordered by increasing size.



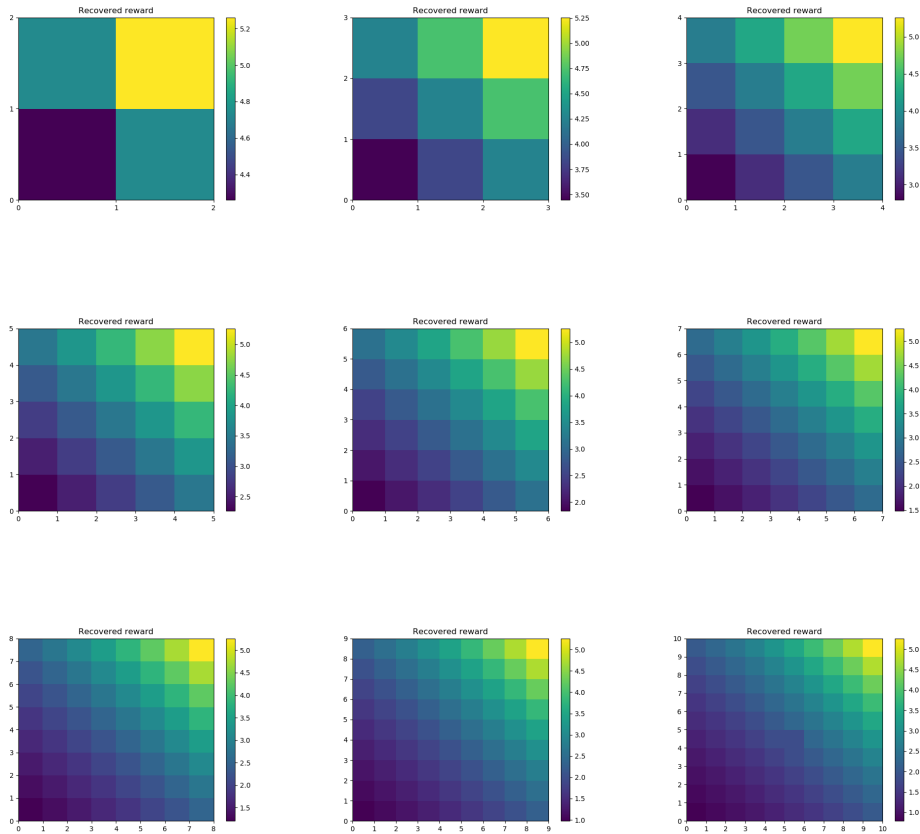
4.1.2 IRL - Maximum Entropy

Representation of the value map of different sized grids, ordered by increasing size.



4.1.3 RL - Value Iteration

Representation of the value map of different sized grids, ordered by increasing size.



4.1.4 Running time between IRL algorithms

Time difference in seconds between both IRL algorithms, run over 200 epochs.

	2x2	3x3	4x4	5x5	6x6	7x7	8x8	9x9	10x10
Linear	0.01	0.01	0.02	0.03	0.05	0.07	0.13	0.17	0.24
Max.Ent.	0.11	0.44	1.81	6.06	13.52	28.23	54.43	92.88	159.5

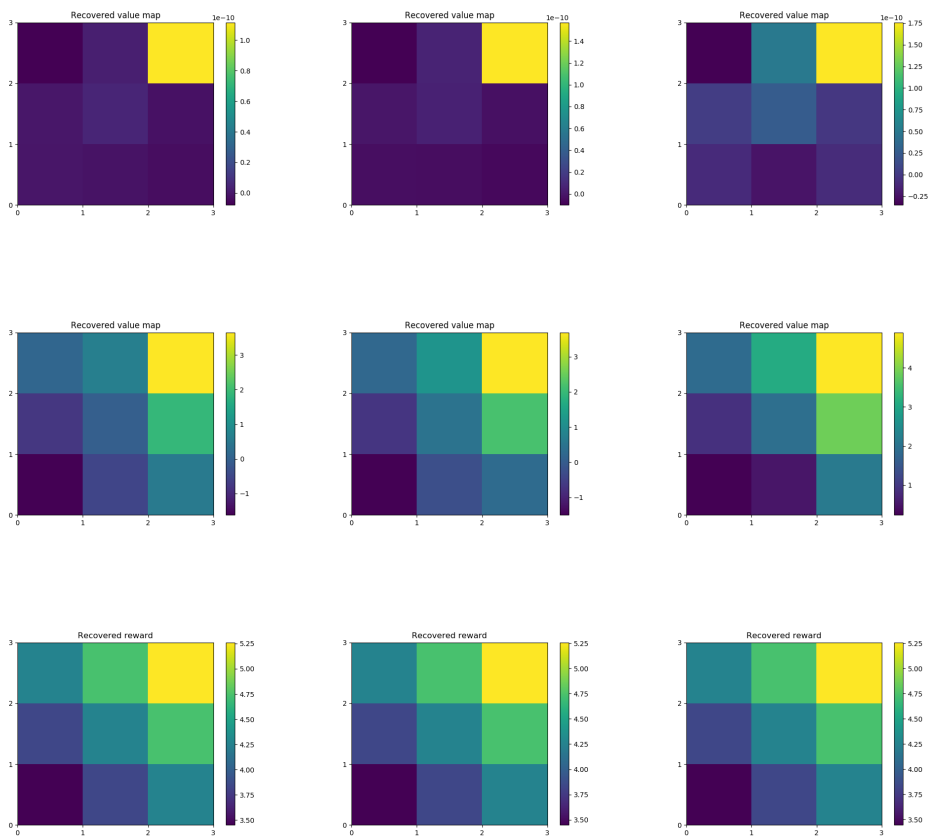
4.2 Second Experiment

In this second experiment, we will proceed to modify the discount parameter to observe how it impacts the behaviour of the different algorithms. To do so, we will employ 3 different sized grids, 3x3, 6x6 and 9x9 and will iterate over varying discount parameters, 0.05, 0.1, 0.5. The remaining parameters will be set as described in the previous section.

4.2.1 3x3 grid

Each row corresponds to a different algorithm. From top to bottom: Linear Programming, Maximum Entropy and Value Iteration.

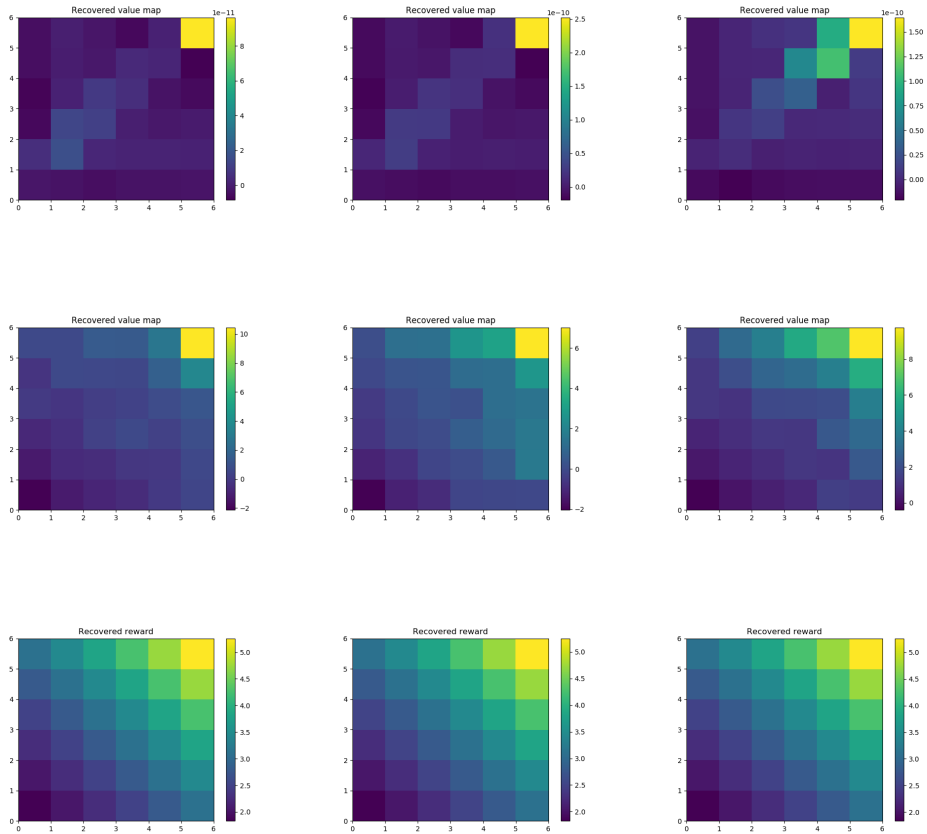
Each column corresponds to a different discount value. From left to right: 0.05, 0.1, 0.5.



4.2.2 6x6 grid

Each row corresponds to a different algorithm. From top to bottom: Linear Programming, Maximum Entropy and Value Iteration.

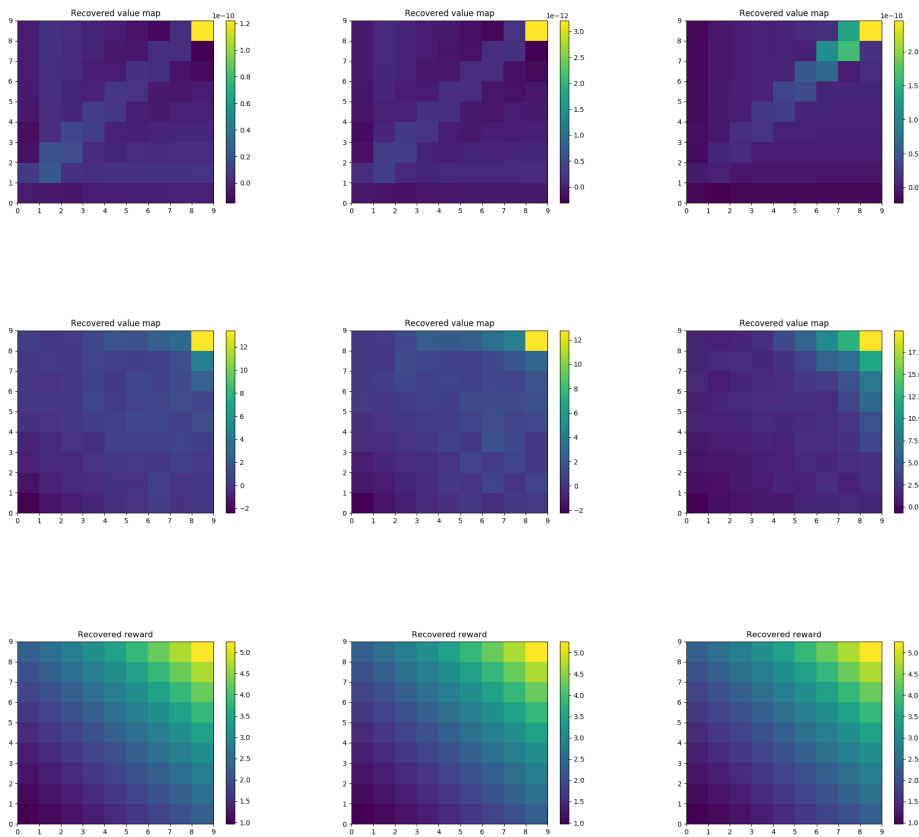
Each column corresponds to a different discount value. From left to right: 0.05, 0.1, 0.5.



4.2.3 9x9 grid

Each row corresponds to a different algorithm. From top to bottom: Linear Programming, Maximum Entropy and Value Iteration.

Each column corresponds to a different discount value. From left to right: 0.05, 0.1, 0.5.



Chapter 5

Discussion

5.1 Hypothesis

Regarding the first experiment, our initial hypothesis is that the ideal result should be a diagonal gradient, starting in cell $(0,0)$ and finishing in cell $(N-1,N-1)$ (where N is the size of the grid). Being the discount factor constant, every cell being crossed by a diagonal line is at equal distance from the final reward. This is easily illustrated through figure 5.1. It is also expected that Reinforcement Learning performs better than Inverse Reinforcement Learning, specially on increasing size grids. It should converge faster and the output value map should be more accurate than the value map given as the output of the IRL algorithm. This is due to the nature of the RL algorithm. It will work as a value propagation from the $(N-1,N-1)$ to the cell located at $(0,0)$, generating a clean value map.

No hypothesis is set for the second experiment, it is yet to see what the results will yield.

5.2 First Experiment - Discussion

The first experiment yields really interesting results, as each algorithm performs in a different way.

The Reinforcement Learning algorithm follows the hypothesis formulated in the previous section. The gradient created remains stable throughout the different grid sizes. All the cells located on the same diagonal contain the exact same values. This is an expected behaviour that can be simply described. The

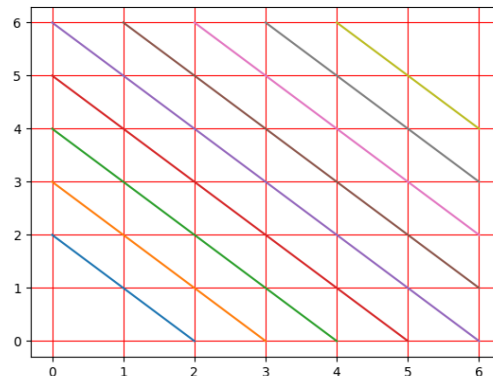


Figure 5.1: *Hypothesis, where every diagonal line represents all cells that are equidistant from the final reward*

value-iteration algorithm propagates the values from the reward cell towards the other cells. On each step, equidistant cells are updated, forming the gradient described (see figure 5.2)

The Maximum Entropy algorithm follows a different pattern (see figure 5.3). On smaller grids, it resembles the Reinforcement Learning algorithm, but as the size of the grid increases, the gradient pattern becomes barely visible, specially in the middle section of it. This performance can be explained through the maximum entropy concept: Paths with higher total reward have a higher exponential chance to be elected. This is exactly how it performs, yet equally rewarded paths may cross the grid through completely different cells, as an equal reward will always be obtained when performing the same amount of steps. This generates a value gradient close to the $(0,0)$ cell and the reward cell, as there are less possible cells a path can go through, but it gets distorted in the center, where the number of possible cells increases.

The Linear Programming algorithm obtains the most unique outcome (see figure 5.4). Instead of generating a diagonal gradient, it creates a diagonal path that crosses from cell $(0,0)$ to the reward cell. This path is also optimal, as the number of steps required to reach the end goal from any of the cells that are part of the path is minimal. Initially, the reasoning behind this specific pattern might be a product of the L1-Norm employed in the algorithm. However, in the original paper [7], the authors relate this behaviour to the mirror-like equivalence of the diagonal cells. As these cells have equivalent values, the algorithm

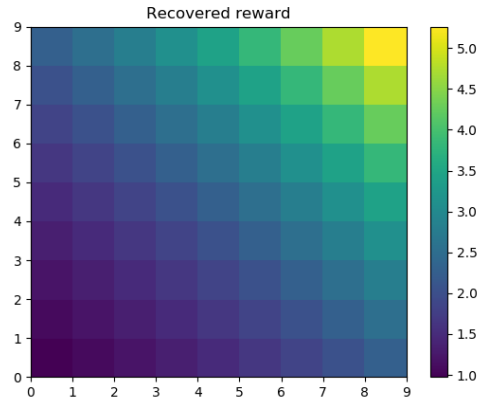


Figure 5.2: *Extracted example of the value map obtained from the RL algorithm*

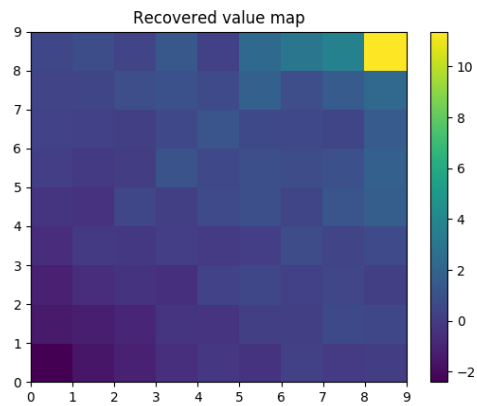


Figure 5.3: *Extracted example of the value map obtained from the Maximum Entropy algorithm*

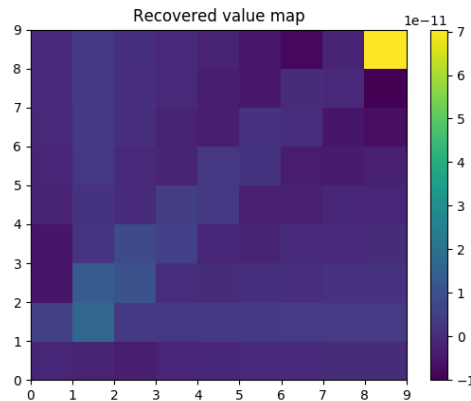


Figure 5.4: *Extracted example of the value map obtained from the Linear Programming algorithm*

simply prioritizes this central path. The authors also describe that, by using the Lasso regularization, we reduce the sparsity of our outcome, shrinking all the values and generating a more homogeneous outcome.

It is worth noting that there is a relevant cost difference between both IRL algorithms. The temporal cost and growth is significantly higher for the Maximum Entropy algorithm. This is due to the amount of matrix operations needed for it to operate.

5.3 Second Experiment - Discussion

This second experiment yields a more generic result.

On one side, the Reinforcement Learning algorithm's performance on all 3 grid sizes does not change. The diagonal gradient is totally visible and remains constant, as a modification on the discount factor does not change the way the algorithm works, generating the value map by propagating the values from the cell that contains the reward to the remaining cells, one step per iteration (see figure 5.5).

On the other side, an increasing discount factor does have an impact on the Maximum Entropy algorithm. With a discount factor of 0.5, the cells that are closer to the end reward cell have a significantly higher value than the previous

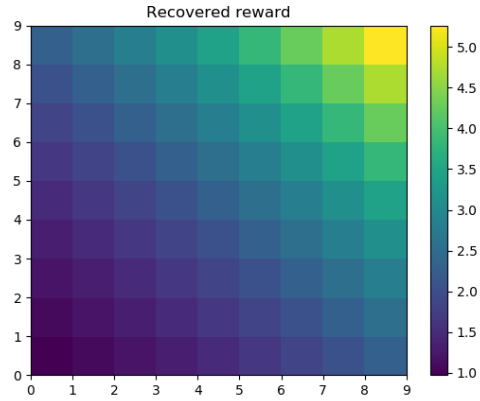


Figure 5.5: *Extracted example of the value map obtained from the RL algorithm, 9x9 grid with discount factor of 0.5*

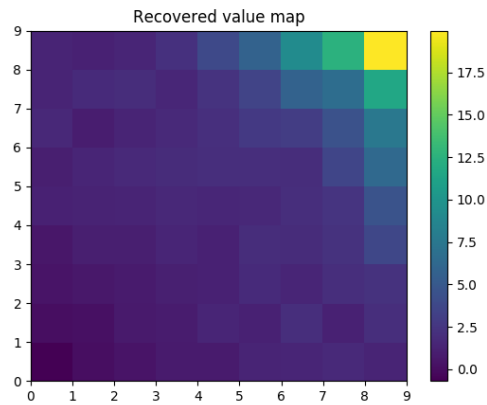


Figure 5.6: *Extracted example of the value map obtained from the Maximum Entropy algorithm, 9x9 grid with discount factor of 0.5*

versions with a smaller γ . Furthermore, a higher discount factor improves the quality of the result for the Maximum Entropy algorithm. As we can observe in the 9x9 grid, the last value output does show a clear gradient pattern close to the end reward cell (see figure 5.6). To explain this behaviour one must first quote the purpose of this parameter (see section 2.1):

The discount factor determines the size of a reward based on the time moment it is obtained.

As an example, given a discount of $\gamma = 0.5$, obtaining a reward in t_1 of 2 is equivalent to obtaining a reward in t_2 of 4. This means that if two agents take the same steps in almost identical environment, where the only difference is the discount factor, the agent that operates with a smaller discount factor will obtain a higher reward. This reduces the noise of our value map for the Maximum Entropy output, as the cells with a theoretical higher value will now be more likely to be chosen, as they will be more distinguishable from those with a theoretical lower value.

In the Linear Programming algorithm, it is worth mentioning that the cells closer to the end reward cell show an increment in their value and the overall gradient is noticeably smoother. However, this is not significant, as the underlying path that was observed in the first experiment is still present and it does not affect the overall performance of the algorithm (if an agent were to start at s_0 , the path that it would follow would be equivalent to the path followed by that same agent in a Gridworld with a lower discount factor).

Chapter 6

Conclusion

To summarize the results obtained in this research project, Reinforcement Learning does perform more consistently and qualitatively better. However, it is really interesting to observe how different algorithms from the Inverse Reinforcement Learning paradigm perform over the same environment.

One of the conclusions that can be understood from this paper is how both learning approaches behave on short term vs long term oriented models. RL is capable of generating an effective value map, yet IRL performs much worse the more sparse the environment is, the more states it has and the bigger the average distance from a state to the end reward is. Also observing that a higher discount factor improves the quality of the result on the IRL output is meaningful, as it can lead to other ways of trying to effectively enhance the result, such as adding artificial intermediate rewards with an overall higher discount factor to reduce the noise in the middle section of the grid.

It is yet to see how other parameters, such as the length of the sample paths or the total number of sample paths for IRL or the random jump variable, affect the algorithms. Future work could involve said experiments.

Glossary

AL Apprenticeship Learning. 31

IRL Inverse Reinforcement Learning. 31

MDP Markov Decision Process. 31

RL Reinforcement Learning. 31

Bibliography

- [1] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.
- [2] Andrew G Barto, PS Thomas, and Richard S Sutton. Some recent applications of reinforcement learning. In *Proceedings of the 18th Yale Workshop on Adaptive and Learning Systems*, 2017.
- [3] Engin Ipek, Onur Mutlu, José F Martínez, and Rich Caruana. Self-optimizing memory controllers: A reinforcement learning approach. In *ACM SIGARCH Computer Architecture News*, volume 36, pages 39–50. IEEE Computer Society, 2008.
- [4] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.
- [5] Philip S Thomas and Billy Okal. A notation for markov decision processes. *arXiv preprint arXiv:1512.09075*, 2015.
- [6] Francisco S Melo. Convergence of q-learning: A simple proof. *Institute Of Systems and Robotics, Tech. Rep*, pages 1–4, 2001.
- [7] Andrew Y Ng, Stuart J Russell, et al. Algorithms for inverse reinforcement learning. In *Icml*, volume 1, page 2, 2000.
- [8] Brian D Ziebart, Andrew L Maas, J Andrew Bagnell, and Anind K Dey. Maximum entropy inverse reinforcement learning. In *Aaai*, volume 8, pages 1433–1438. Chicago, IL, USA, 2008.
- [9] Matthew Alger. *Inverse reinforcement learning*, 2016.

TRITA-EECS-EX-2019:400