UTRECHT UNIVERSITY

BACHELOR THESIS

Model-based VS Model-free Approaches for Argument-based Inquiry

Author:
Arja Mentink

Supervisor:
dr. B.J.G (Bas) Testerink
Second Supervisor:
dr. G.A.W. (Gerard) Vreeswijk

A 7.5EC thesis for the Bachelor of Science in Artificial Intelligence

June 20, 2020

Universiteit Utrecht

Abstract

Arja Mentink

Model-based VS Model-free Approaches for Argument-based Inquiry

The Dutch National Police researches different A.I. techniques for (semi-) autonomous decision making. One of those techniques involves the use of structured argumentation [4]. It guides an agent in an information search process which we call argument-based inquiry. This technique is for example used in the handling of international communication [8] and the intake of crime reports [1].
Reinforcement learning can be used in a wide variety of decision making tasks where the goal of an agent is to maximize its expected future reward. In this field there are two main approaches: model-free methods and model-based methods.
Though model-free methods are very generic, model-based methods offer better insight in the decision making process. Since transparency and explainability are important for the police, it would be beneficial if a model-based approach is used in argument-based inquiry rather than a model-free approach. However, the use of a model-based method should not impact the performance negatively. Hence it is worth investigating model-based methods for finding a policy for argument-based inquiry.
In this report we research whether model-based methods perform similar to model-free methods for argument-based inquiry. In Uijlen [9] multiple model-free methods have been implemented and tested on argument-based inquiry. We will implement model-based methods and compare the performance.
Comparing the theory on model-based and model-free methods led to the hypothesis that model-based methods would indeed perform similarly in our domain. This was shown to also hold empirically for our implementation of the model-based methods. The experimentation also showed some potentially relevant trade-offs that may occur when opting for model-based approaches.

Keywords: Reinforcement learning; Model-based; Model-free; Argument-based inquiry

Contents

Chapter 1

Introduction

The Dutch National Police does various A.I. related research, for example techniques for (semi-) autonomous decision making. These techniques can be used for various purposes to support police work. One application of these techniques is processing online fraud complaints automatically [1]. This way, less people need to be involved, and the process can be completed more quickly and with improved quality.
In 2019 Uijlen [9] researched how to train an agent to handle online fraud complaints. To handle a fraud complaint, the agent needs to perform inquiry to determine whether the case is a case of fraud or not.

In an inquiry task, an agent searches for information about a specific topic. The agent wants to learn whether the given topic is true or false. Inquiry can be done in different ways. In this thesis we will use an argument-based inquiry method [7]. In this method the agent creates arguments for and against the topic by matching argumentation rules with observations from the environment. These observations are gathered by querying the environment. With these arguments the agent can make a decision about the topic. For example, in the case of online fraud complaints, the agent wants to determine whether the specific case is a case of fraud. It does this by asking questions to the complainant and creating arguments with the answers.
In this thesis and in [9] this argument-based inquiry method is used to determine the status of the topic and which information is relevant at each time-step. Reinforcement learning techniques are then used to learn an agent which of the relevant questions to ask in each circumstance. This way, an agent can learn how to perform inquiry as efficiently as possible.

In reinforcement learning an agent learns which actions to take by trying to optimize an expected future reward. A reward is a simple numerical value. The agent is not told what to do, but must learn which actions lead to the highest reward. Reinforcement learning problems can often be formalized as a Markov-decision process (MDP), including the argument-based inquiry learning problem [7]. An MDP is a representation of the decision process, it defines the behavior of the environment by defining the environments one-step dynamics. Markov-decision processes will be explained in chapter 2, but the basic idea is to capture the most important aspects of the problem in their simplest form.
Solving the learning problem now means solving the MDP. Solving an MDP means finding the best action for each state, this is called the optimal policy. This way, the agent knows what to do in any state. In the inquiry domain the optimal policy returns the best query to execute. MDP's can be solved using various reinforcement learning methods. These methods can be divided between two approaches: model-based and model-free methods.
The model-based algorithms try to solve the underlying MDP by approximating a

model of the environment and solving that model. In reinforcement learning the model is trying to capture two functions, the transition function (the probabilities of ending up in a next state) and the reward function. In the Model-free approaches, the agent learns the state values (how good each state is), and the policy, directly from experiences.
A popular reinforcement learning algorithm is Q-learning. It has for example been used in AlphaGo. AlphaGo is the first computer program to defeat the top human players in a game of go [5]. In Q-learning a function is used to learn the Q-values per state-action pair, which is the expected reward of doing that action in that state. An optimal policy is then determined from these values. In [9] Q-learning was used with a model-free approach to solve the inquiry learning problem. In this thesis we will use it with a model-based approach, and compare the performance.

There are reasons why model-free approaches are very popular. A reason for this is that they are very generic. Moreover, they are computationally cheaper than model-based methods [10]. However, a problem in model-free Q-learning is that the function is not clearly explanative, it is unclear what makes an action good. The Q-function takes the immediate reward, and the future reward into account. These different values are not easily determined from the Q-function itself, they are embedded into the function. Because of this, it is unclear whether an agent chooses the action because it gives a high immediate reward, or because it prevents a high penalty in the long run. With a model-based approach, in which there is a model for the transition probabilities and the reward function, we can use that model to look ahead and figure out what makes an action good or bad. In a model-based approach the actions of the agents will thus be more explanative. In the case of online fraud this is preferred since transparency and explainability are important for the police.

For this research we are interested in comparing model-based and model-free learning for argument-based inquiry. We answer the question: *Does model-based q-learning perform similar to model-free approaches for argument-based inquiry?*
If the performance is the same, then the case can be made that model-based approached are preferred since it allows us to create explanative q-learning agents. The research question will be answered by means of the following subquestions:

- Subquestion 1: Under what circumstances is model-free or model-based preferred according to literature?
- Subquestion 2: Does the domain of argument-based inquiry fit these criteria? I.e.: which one should theoretically work best?
- Subquestion 3: With empirical examples; can we observe our theoretical hypothesis?

For subquestion 1 we will conduct literature research. For subquestion 2 we analyse the MDP constructed for argumentation-based inquiry for the relevant properties. For subquestion 3 we develop a test framework and run simulations of inquiries.

The structure of this thesis will be as follows. Reinforcement learning and Markov-decision Processes will be explained in chapter 2. We will also explain the method used to conduct argument-based inquiry, and define the MDP for our problem. In chapter 3 we will answer subquestion 1 and 2 by comparing model-free with model-based approaches, first in general, and then for our specific MDP. The implemented algorithms will be explained in chapter 4. Chapter 5 contains the results, chapter 6 the discussion and chapter 7 the conclusion.

Chapter 2

Reinforcement Learning and Markov-decision Processes

In this chapter the concepts reinforcement learning and Markov-decision processes will be explained, based on the textbooks "Reinforcement learning: An introduction" [6] and "Reinforcement Learning" [10]. Additionally, it will describe how the domain of argument-based inquiry can be modelled as a Markov-decision process.

2.1 Reinforcement learning

In reinforcement learning an agent learns what to do as to maximize a numerical reward. The agent learns to achieve such a goal by interacting with the environment. The environment is everything outside the agent. The agent and environment interact continually in discrete time steps. At each time step the agent receives a representation of a state, $S_t \in S$, from the environment. A state is a unique representation of everything that is important in a state of the problem. For example, in chess, a state is a complete configuration of all board pieces. After receiving the state, the agent selects an action that is available in that state, $A_t \in A(s)$. The action moves the agent to a new state. The agent receives feedback from the environment after the action is performed. This feedback consists of a representation of the new state and a reward. Rewards are numerical values that the agent wants to maximize over time. In Figure 2.1 you see a representation of the agent-environment interaction.

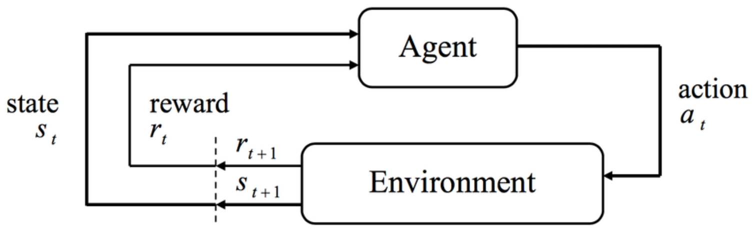


FIGURE 2.1: A representation of the agent-environment interaction

2.1.1 Rewards

What does it mean for the agent to maximize the expected reward? At each time step the agent receives a simple numeric reward. However, the goal is to maximize the total amount of reward, the cumulative reward in the long run. How can we define this in a formal way? A sequence of future rewards is denoted as $R_{t+1}, R_{t+2}, ...$ The total amount of future reward can be denoted as:

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + ... + R_T$$

With G_t the expected future reward at time step t, and T the final time step. Calculating the expected reward this way is possible if the interaction between the agent and environment breaks naturally into subsequences, called episodes, for example a play of a game is an episode. Each episode ends in a final state, and is then reset to a starting state. However, many agent-environment interactions are continuous, they do not break into episodes. One such task is an on-going process-control task. In these cases, the previous function cannot be used to calculate the expected reward, since T = ∞, and the reward could become infinite as well.
To handle these cases we need the concept *discounting*. In this approach, the agent chooses an action to maximize the expected *discounted reward*:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ...$$

Where $0 \leq \gamma \leq 1$, is called the discount rate. The discount rate determines the value of future rewards. A reward receives 2 steps from now is only worth γ^2 times what is would be worth if it were received immediately. With a discount rate close to 0, the agent prefers immediate rewards. With a discount rate close to 1 the future rewards are taken into account more strongly. With discounting immediate rewards are preferred over future rewards.

2.2 What are Markov-decision processes?

Reinforcement learning problems are often modelled as a Markov-decision process (MDP). An MDP can be used to represent the states of the problem, and the transitions between the states. In this section, we will describe an MDP, and explain how it can be solved.

An MDP is defined by its set of states, set of actions, rewards and transition function. The reward is the numerical value the agent receives when moving from state s to state s' by performing action a, it is denoted by $R^a_{s,s'}$. An MDP has a transition function because it is stochastic, which means that given a state and an action, the next state is non-deterministic. The transition function tells us the probability of ending up in state s$'$ after doing action a in state s, denoted by $P^a_{s,s'}$. These probabilities, together with the rewards, define the dynamics of the MDP.
A reinforcement learning problem can only be modelled as an MDP if it has the Markov property. The Markov property states that the probability of reaching state s' is only dependent on the preceding state s and action a, and not on the earlier states. It can be seen as a restriction on the state, the state must include all information of the past that is relevant for the future. In real world scenarios we do not always have the Markov property. In that case, the states can often be described in such a way that the Markov property does hold, by including all relevant information in each state.

2.2.1 Solving a Markov-decision process

Solving a Markov-decision process means finding an optimal policy (π^*). A policy gives an action for each state. An optimal policy is a policy that maximizes the

expected reward. Most learning algorithms compute the optimal policy by learning value functions. A value function estimates the value of each state, which is a measure of how good it is for the agent to be in that state. The value of a state is expressed in terms of the expected reward. The optimal policy is found by maximizing the value function.
Computing the value function and optimal policy can be done using different algorithms. Some of these algorithms, called dynamic programming, use a model of the MDP to calculate the value function. A model of an MDP consists of the transition probabilities for each state and the rewards. These algorithms assume that the model is known. However, the difficulty is that in real life problems, these probabilities are often not known. Reinforcement learning algorithms solve the MDP by interacting with the environment. These algorithms do not need a model of the environment. Reinforcement learning algorithms can broadly be divided into two groups, model-free and model-based algorithms. Model-free algorithms use samples of the state transitions and reward to estimate state-action value functions. On the other hand, model-based algorithms use the samples to estimate a model. After a reasonable model has been learned, the MDP can be solved using dynamic programming algorithms.

Difficulties when formulating MDP's
First of all, a difficulty in MDP's is defining your states, actions and reward function in such a way that you get the desired behavior. Not all learning problems can be formulated as an MDP in a trivial way. There needs to be time spend on defining your states in such a way that the Markov Property is valid. Furthermore, the reward function needs to be defined in such a way that the agent learns the right behavior in the end. When solving a learning problem, the first obstacle is thus defining your MDP. In section 2.4 we will define the MDP for this thesis.
Another obstacle that is encountered when solving the MDP with reinforcement learning lies in choosing the right parameters for the algorithm. The learning algorithms have various parameters that impact the algorithm's performance. Choosing these parameters is not a trivial task and time should be spend on trying to find values that make the algorithm work as intended.
One such parameter is the exploitation factor, epsilon. This parameter influences whether the agent chooses the best action, or a suboptimal action in any state. This is a trade-off between exploration and exploitation. Exploration means trying new unexplored actions, while exploitation means choosing the best action. To obtain a reward, an agent has to exploit the part of the environment that it already knows. However, to discover which actions lead to the highest reward, an agent has to explore, so that it can make better action decisions in the future. The difficulty is that the agent can neither explore nor exploit exclusively without failing to find the optimal policy. The agent needs to balance exploration and exploitation.

To conclude, to define an MDP we need states, actions, rewards and the transitions function. To solve an MDP we need a learning algorithm and values for the parameters. In this thesis the focus is on comparing the algorithms from a proof-of-concept perspective. Because of this, no extensive hyper-parameter tuning will be carried out.

2.3 Argument-based inquiry

In this thesis we will focus on a Markov-decision process regarding argument-based inquiry. The argument-based inquiry task is first described before discussing how this can be modelled as an MDP. The method for argument-based inquiry determines which information is still relevant in each state. The reinforcement learning techniques will then be used to determine which relevant information to query next. The inquiry method will be explained in the following section. This section is based on Testerink et. al [7].

When an agent is performing an inquiry, it gathers information from the environment so that it can form an opinion on a specific topic. As mentioned in the introduction, there are different ways an agent can perform inquiry. In this method, an agent creates arguments for and against a particular topic by matching argumentation rules with observations gathered by querying the environment. These arguments can then be used to determine the status of the topic.
The inquiry method uses argumentation to provide transparency about the 'opinion of the system'. In argumentation theory there exists a distinction between structured argumentation and abstract argumentation. Structured argumentation is used for constructing the arguments. Our method uses a simplified version of ASPIC+ [4] for structured argumentation. Abstract argumentation is used to select the acceptable arguments. Our method uses Dung's grounded semantics [2] for abstract argumentation.

Now that the basis of argumentation is explained, let us look at how the agent actually performs the inquiry. For an agent to perform inquiry, it needs knowledge. In this method the knowledge of the agent is modeled as a structured argumentation setup, similar to ASPIC+ [4]. The concepts of queryable literals (observations that can be made in the future) and a topic are added to the concepts of ASPIC+. The topic is a special literal for which the agent aims to get a stable opinion. The structured argumentation setup is then defined as follows:

- $\mathcal{L}$: A logical language consisting of propositional literals
- $\mathcal{R}$: A set of defeasible rules
- $\mathcal{Q}$: A set of non-negated queryable literals
- $\mathcal{K}$: A knowledge base of observations which is a consistent set of literals (what the agent knows)
- τ: A topic

The agent can determine the value of the queryable literals by querying the environment to extend his knowledge base. With these literals and the arguments that follow from the current knowledge base, the status of the topic, and which observations could change this status, can be determined.
To avoid making unnecessary queries, the agent needs to determine if the acceptability status of the topic can change given more information. This is done with the notion of stability, where a structured argumentation setup is stable if executing more queries will not change the status of the topic. To define stability the notion of all future setups is used, future setups are all those setups where the queryable literals are put in the current knowledge base. An argumentation setup is stable if any of the following holds:

- *Unsatisfiable*: For all future setups there is no argument for the topic.
- *Defended*: For all future setups, there is an argument for the topic in the grounded extension.
- *Out*: For all future setups there is an argument for the topic, but all arguments are attacked by an argument in the grounded extension.
- *Blocked*: For all future setups there is an argument for the topic that is not in the grounded extension and there is at least one argument that is not attacked by an argument from the grounded extension.

Determining stability is computationally very expensive, since it requires hypothesizing over all future argumentation setups. Therefore, a less complex approximation algorithm is used, which is a sound approximation of stability. The idea is that rules and literals are labelled, where the labels relate to the cases of stability.
Given these labels, the relevant literals will be determined. A literal is relevant if it could change the status of the topic and if it is unlabelled. If there is a relevant literal, a query should be asked to the environment to determine the value of the relevant literal. Which query to ask will be determined by the reinforcement learning algorithm. When the topic is labelled, or when there is no relevant literal, the argumentation setup is stable, and the status of the topic can be determined by its label.

2.4 Our Markov-decision process

In this section we will describe our MDP for argument-based inquiry. In [7] it is explained why our problem could be modelled as an MDP. First we will define the states, actions, transition function and reward function. After which the characteristics of the MDP are described, these are important for formulating our theoretical hypothesis.

2.4.1 MDP definition

As mentioned in section 2.1 an MDP is defined by a state-space, actions, the transition function and a reward function. Here, we will consider each in turn. Furthermore, to help understand the MDP for the inquiry problem, we give an example of an MDP for an argument-based inquiry domain.

State-space
In this MDP, a state is a pair of observables and the available queries. The set of observables represents the knowledge base of the agent. The set of observables is empty when the agent has no information. The available queries show which actions are available in that state. By formulating the state-space this way, the Markov property is achieved. All relevant information of the past, namely, which actions have already been done, is available in the state itself. In this case, the agent knows which questions he has asked, by looking at the available questions.
A final state is a state in which the agent has enough information to determine the status of the topic. This means that the state is stable, as explained in the previous section.

Actions
An action is a possible query that the agent can ask. In this thesis we assume that there is a specific query for every observable. This is done for clarity but in general this is not essential. Which actions are available depends on the state the agent is in, i.e. on the observables and the available queries. The agent can only execute those actions that yield observables that the agent is missing and that are relevant. Which queries are relevant is determined by the method explained in the previous section. This is done because it is useless for the agent to ask queries about observables that it already knows. An action leading back to an already visited state are not allowed either. These actions do not yield observables that the agent is missing, since the set of observables in a previous state can only be smaller.

The transition function
The transition function tells us with what possibility we end up in state s' after asking query q in s. The transition probability for illegal transitions is set to zero. An example of an illegal transition is a transition that leads to an already visited state. In our MDP, the transition function is not known. It is either explicitly learned with a model, or implicitly learned.

Reward function
A positive reward is given after reaching a final state, a stable argumentation setup, from an unstable setup. In this case, the agent receives $+n$ with n the number of available actions in that state. A negative reward (-1) is given for transitions between unstable setups because a query was executed and the goal is to get to a stable setup after as few queries as possible. The optimal policy is to execute that action/query that maximizes the expected reward. With a negative reward for each step, the number of queries will be minimized before reaching a stable setup.

Example 1: *Defining an MDP*
In this example we will define an MDP for an argument-based inquiry domain, and show how the expected reward can be calculated. To define an MDP we need to define the state-space, the actions, the reward function and the transition function. We will define each in turn. After which we will show an example of calculating the expected reward.
Here we will use a synthetic example. This example will also be used as the test for our experiment. By using a synthetic example we can produce a larger example for more detail in the experiment. In [7] and [9] examples of MDP's for the fraud-domain have been developed.

In this example we have four observable literals (a, b, c and d, and additionally their negations), four possible corresponding actions (A, B, C and D) and one topic t.
The state-space of this MDP represents the information that the agent knows. The state-space comprises of those states with a set of zero or more of the observable literals, either true or false. For example $\{a, \neg d\}$ is a state. The set of literals in a state represent the knowledge base of the agent in that state. We represent a state with the knowledge base instead of a pair of the knowledge base and the available actions, because the available actions follow directly from the knowledge base in this example.
To get to another state we need actions. There are four actions; A, B, C and D. Action A leads to knowledge about literal a, action B about literal b etcetera. In each

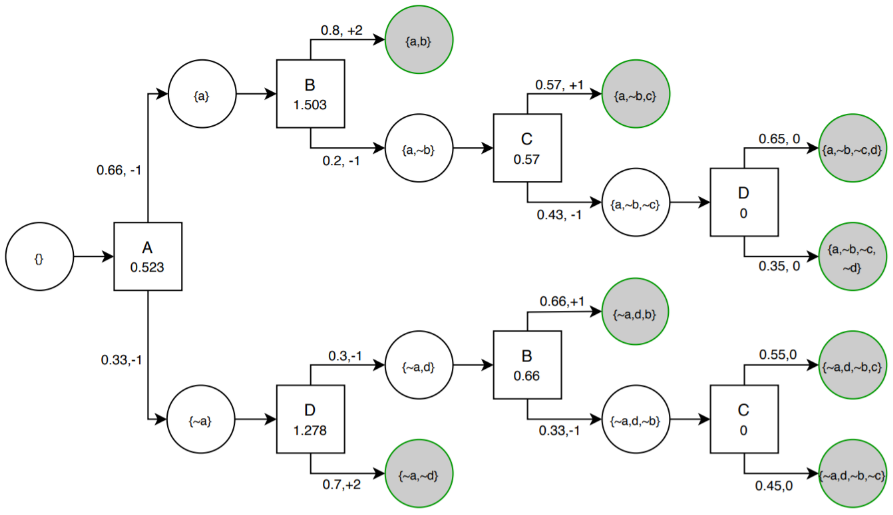


FIGURE 2.2: Diagram of optimal path agent for our example MDP

state, an agent can perform an action that leads to a state with more information. The possible actions per state are those actions that are relevant in that state. If we are in state $\{a\}$, action A is irrelevant since the value of literal a is known. Thus in each state, only those actions whose literal is unknown can be performed.
To make the state-space and actions more clear, let us look at 2.2. This diagram represents part of the MDP. For the purpose of readability we do not give the entire MDP. Instead, 2.2 shows what an optimal agent would do in each state. The circles represent the states, and the squares represent the actions. The filed in circles represent the final states. In these states, the status of the topic t can be determined. For example, after doing action A, and ending up in $\{a\}$, the agent could do actions B, C and D. However, the best course of action in this MDP, to receive an expected maximum reward, is to do action B.
Next, we define a reward function ($R^a_{s,s'}$). A reward function can be defined in multiple ways, depending on the goal of the task. In this example we will use the same reward function as explained above in section 2.4.1. This function was as followed; a positive reward is given after reaching a final state. For the final MDP this was $+n$, with n the number of available actions in that state. The agent receives a negative reward for transitions between non-final states, since we want to reach the final state after as few steps as possible. The negative reward will be -1. The reward is shown in 2.2 on each arrow (second number).
Now that we have defined the state-space, actions and reward function, only the transition function is left. For this example we will define the transition probabilities $P^a_{s,s'}$. However, this will be unknown in the experiment.
We define the transition function only for those transitions in 2.2. This is done because defining the function for all possible transitions does not make it easier to understand the example. Furthermore, for explaining how the agent learns a policy, we just need the probabilities for these transitions.
When the knowledge base of the agent is empty, it will perform action A. This leads to two possible next states $\{a\}$ and $\{\neg a\}$. Assume that the chance to end up in state $\{a\}$ is 0.66, the probability to end up in $\{\neg a\}$ will then be $1 - 0.66 = 0.33$. The probabilities for each transition will be defined the same way, the result is shown in 2.2. The number on the arrow represent the probability of that transition. The first number is the probability, the second is the reward.

Now that we have defined the MDP, let us look at calculating the expected reward of a state given an action. This can be done by using the following function:

$$V(s) = \sum_{s'} P^a_{s,s'}[R^a_{s,s'} + \gamma V(s')]$$

Where γ is the discount factor, a the best action in state s and s' a possible new state given s and a. All possible next states per action are taken into account, since the agent might end up in any of them with a certain probability (P). Let us look at an example. Let us look at what the expected reward is in state $s = \{a, \neg b, \neg c\}$ taking action D and let us take $\gamma = 0.9$.

$$V(s) = 0.65 * (0 + 0.9 * 0) + 0.35 * (0 + 0.9 * 0) = 0$$

Since both next states give a reward of 0, state s has an expected reward of zero. Now that we know the expected reward of $\{a, \neg b, \neg c\}$, we can calculate the expected reward of $s = \{a, \neg b\}$ doing action C.

$$V(s) = 0.57 * (1 + 0.9 * 0) + 0.43 * (0 + 0.9 * 0) = 0.57$$

Here, one of the next states does lead to a positive reward, which means that there is a positive expected reward. The expected reward for each action is shown in 2.2. Calculating the expected reward with this function is the basis for dynamic programming and reinforcement learning algorithms. They use the expected reward to determine the optimal policy. The algorithms used in our experiment will be further explained in chapter 4.

2.4.2 MDP characteristics

Reinforcement learning algorithms tend to be specified for arbitraty MDP's. However, as was already shown by Uijlen [9] certain characteristics of a specifiek MDP may help learning algorithms. The following characteristics are inherent to argumentation-based inquiry MDP's.

1. An agent cannot choose an action that leads to an already visited state during one episode. This is called forward progression.

2. The actions do not change the environment, it only changes the state of the agent. Whether or not the counterparty sent a product does not change due to an action. The information that the agent obtained earlier is not changed by executing an action. Executing an action only leads to the agent getting more information. Since the environment is static, it means that reinforcement learning is theoretically a good fit.

3. Our MDP is finite, there is a finite set of states and a finite set of actions. It is also non-cyclic (follows from 1 and 2). This means that each episode ends after a finite number of iterations. An episode can only last a certain number of moves, the number of actions in the initial state, which is finite.

4. Given the observables an action can reveal, the possible next states can directly be determined given the state and action. For example, the action A can reveal the observable a, which can be true or false. The possible next states are thus the previous state, plus the observables a or $\neg a$. This means that for a model based approach the transition model can be greatly simplified and only needs to model one probability per state-action pair.

5. Given a state-action-state triple, the reward is known. For a model-based approach this means that the reward model is already given and perfect.

6. Though the MDP is finite and non-cyclic, it still has an exponential number of states. This is why techniques such as dynamic programming are theoretically perfect but practically infeasible.

Chapter 3

Model-Based and Model-Free Reinforcement Learning

In reinforcement learning, the problem is often modelled as a Markov-decision process. However, in reinforcement learning problems, the MDP to solve is not always known. This means that in those problems, we do not know the transition function, nor the reward function. There are broadly two ways of dealing with this uncertainty; model-based and model-free methods. In this section we will compare these methods and see in which circumstances one approach might be better. For this section the textbooks "Reinforcement learning: An introduction" [6] and "Reinforcement Learning" [10] were used.

3.1 Model-based approaches

In model-based methods, the MDP is solved by approximating a model of the environment and solving that model. The first step is to learn the model. This is done by doing actions in the environment. The internal model is then used to solve the MDP in a way as if the learned model is correct. Model-based methods thus find the optimal policy indirectly, through a model.
In reinforcement learning a model is anything that an agent can use to predict how the environment will respond to the agent's actions. Given a state and an action, a model predicts the resultant next state and reward. A lot of different models are possible. Each learning problem is different and requires a model that fits that problem. Models are therefore often specific, and applicable to a small number of problems. However, there are also models that are more generic, but those models often fit the problem less well, resulting in a lower performance. There is a gradient in model-based methods from specific to more generic models. Generic models often take longer to solve the MDP, but can be used for multiple domains, while specific models are faster, but can be used only on a small number of domains.

3.2 Model-free approaches

Model-free methods work without a model of the environment. The agent learns the state values directly from experiences. These methods find the optimal policy directly by maximizing over the state values. A model-free method simplifies the learning problem, since no model is necessary. Furthermore, they are very generic, they can be used on multiple different domains.

3.3 Model-based VS Model-free

Model-free and model-based approaches differ in the way they use the experience to learn an optimal policy, one learns the policy directly, and the other indirectly. Both approaches have advantages and disadvantages. In this section we will look at certain (dis)advantages of one approach over the other. These results will be used to say something about which approach to use in certain circumstances.

An advantage of the model-based methods, as mentioned in the introduction, is the explanabilility of those methods. The learned model can be used to look ahead and see what makes an action good or bad. On the other hand, in model-free methods, this information is embedded into the policy function and cannot easily be extracted. Another advantage of model-based methods is that they often have an improved sample efficiency over model-free methods. This means that model-based methods often need fewer actions to learn. However, this is only the case if a model can be learned fast enough. While learning a model, the agent has to act in the environment just as with model-free methods. The model can be used to plan an optimal policy without using additional experiences, only after learning is done. Learning a model as fast as possible can be achieved by using samples. The model can be used to predict future states, thereby making it possible to perform extra sample transitions, speeding up learning. Another option is to drive the agent to explore states that are unvisited or states which the agent is uncertain about. The possibility to plan exploration trajectories is another advantage of model-based methods.
However, this higher sample efficiency does come at the cost of more computational and space complexity for learning the model and planning a policy. This means that even efficient algorithms may take too many actions to be practical for larger and more realistic problems. In addition, it is also difficult to use model-based methods on continuous state and action spaces. While model-free approaches can fairly easily be extended to continuous domains.
Another disadvantage of the model-based approach is that the model could be incorrect. When a model is incorrect, it could generate false transitions, which is disastrous for finding the optimal policy. A number of causes can lead to an incorrect model. For example, only a limited number of samples have been observed in a stochastic environment, or the model was learned using function approximation that has generalized imperfectly. Another cause could be that the environment has changed, and the model has not yet observed its new behaviour.

Comparing the model-free and model-based approach raises the following question; Under what circumstances is model-free or model-based preferred?

- If knowledge about the target MDP's dynamics can be captured in the model, then unseen transitions might be correctly provided by the model through inference over past experiences. Meaning that with a correct model new transitions can be created, leading to a higher sample efficiency than model-free methods. Because of this a model is useful in circumstances where experience is costly.

- Model-based methods are thought to be most useful in large domains with a limited amount of possible actions. When a problem has large or even continuous state and action spaces model-based methods become impractical due to the computational complexity.

- Model-free methods might be preferred when the difficulty in solving a problem is constructing an accurate model of the environment. If no accurate model can be constructed, false transitions will be created by the model, leading to a suboptimal policy.
- Model-free methods can be extended to continuous domains using function approximation, however, model-based methods are not so easily extended to continuous domains. They would have to learn a continuous model, plan in a continuous state and explore a continuous state-space. Model-free methods thus work better than model-based methods in continuous domains.
- Since model-based method are more explanative, they are preferred when insight into the decision making process is important.

To conclude, if you have a problem with a discrete domain and a limited amount of available actions, and if you have an idea about what a realistic model for the problem looks like, and you have enough data to generate the model, then model-based methods would work well, and even better than model-free methods like standard Q-learning.

3.3.1 Model-based VS Model-free for our MDP

In the previous section we looked at under what circumstances each approach should work best. If we combine this with the characteristics of our MDP as described in section 2.4.2, we can see which approach should theoretically works best for the MDP.
First of all, model-based methods were preferred in three different circumstances; when experience is costly, when we have a large domain with limited actions, and when explanation is required. To start, in our domain explanation is important, since transparency is important for the police. Second, we indeed have a large domain with limited actions. As explained in section 2.4.2, there is an exponential number of states, however, the available actions are limited. In our case experience is not necessarily costly, which means that model-based does not have an advantage over model-free on that point, but it is also not a disadvantage.
Model-free methods are preferred when it is difficult to construct an accurate model. However, in our case, we can learn a perfect model. The reward function is already known and perfect (section 2.4.2). For the transition function we know that we only need to learn one probability per state-action pair (section 2.4.2). This probability can be learned by counting how many times an action resulted in one of the two next states. If we can do enough transitions, the count will eventually converge to the true probability, giving us the perfect model. Furthermore, our domain is not continuous, so model-based approaches do not experience any difficulty on that front.
To conclude, we have a discrete domain with a limited amount of available actions. Furthermore, we have an idea for a perfect model of the environment, and if we train the algorithm for enough rounds, we have enough transitions to generate the model. This knowledge, together with conclusion of section 3.3, leads to the hypothesis that a model-based approach should theoretically work best for our argument-based inquiry domain.

Chapter 4

Implemented Algorithms

In this chapter the three algorithms used in our experiment are explained. The model-free method that will be used is Q-learning. A model-free method with some model-based characteristics called replay is also used. Lastly, we have model-based Q-learning. For this method we implement our own model.

4.1 Q-learning

Q-learning is a popular model-free method for reinforcement learning. In this section we will explain how it works. The Q-learning algorithm was implemented by Uijlen [9]. This code will be reused.
Q-learning is model-free, which means that it will learn a policy directly from experience. To do this, the agent learns Q-values for state-action pairs ($Q(s,a)$). These Q-values estimate the expected reward of each action in a state, and can be used to derive a policy. The Q-values are updated based on the received reward and next state.
In Q-learning the agent follows these steps:

1. The Q-values $Q(s,a)$ or initializes to zero for each state-action pair (s,a).
2. An action is selected and reward received.
3. Use the reward to make an estimate of $Q(s,a)$: $r + \gamma \text{max}_{a'} Q(s',a')$
4. Update $Q(s,a)$ by moving the values slightly towards the estimate.
   $Q(s,a) = Q(s,a) + \alpha(r + \gamma \text{max}_{a'} Q(s',a') - Q(s,a))$

The agent will repeat step 2-4 for a number of rounds. Eventually the Q-values converge. The optimal policy can then be found by doing action a with the highest Q-value. An underspecified element above is the choice of action in step 2. There are several schemes proposed for this in literature. For simplicity we use 'epsilon-greedy'. This is a scheme where a parameter epsilon gives probability that a random action is chosen and 1-epsilon is the probability that the action that currently has the highest Q-value is chosen. When implementing Q-learning, one needs to think about which values to use for the hyper-parameters ϵ, the discount factor γ and the adaptation factor α. The adaptation factor influences how much the Q-values move towards the estimate in step 4. In this thesis we will not carry out extensive hyper-parameter tuning. Instead we will use values that are commonly used for these parameters, and that are also used in [9]. These are as followed: $\epsilon = 0.15, \gamma = 0.9$ and $\alpha = 0.03$. These parameters are used for all our algorithms, since part of the Q-learning algorithm will be used for these as well. Below the pseudo-code for Q-learning is shown.

Algorithm 1 Q-learning

```
 1: Input: policy $\pi$ [a]
 2: Input: mdp [b]
 3: for $iteration = 1, 2, \ldots, R$ do
 4:     Generate state: $s \leftarrow$ state-generator
 5:     for $interaction = 1, 2, \ldots, S$ do
 6:         Generate action: $a \leftarrow \pi(s)$
 7:         Generate reward, next state: $r, s' \leftarrow$mdp$(s, a)$
 8:         $\pi \leftarrow UpdateAgent(s, a, r, s')$ [c]
 9:         update current state: $s = s'$
10:         if $s$ is final [d] then
11:             break
12:         end if
13:     end for
14: end for
```

[a] π is the policy used to determine the next action.
[b] mdp is a function that generates a new transition.
[c] The *UpdateAgent* function corresponds to step 4 above, it updates the Q-values.
[d] s is final when a stable argumentation setup has been achieved (section 2.3).

4.2 Replay

Replay is a model-free algorithm with some model-based characteristics. Replay does not use a model, but it does use additional computation to improve the agent's policy [3]. In our implementation, replay uses Q-learning to learn a policy and to update the Q-values. However, each observed transition is stored. After every transition, a sample will be taken from those stored transitions. Every transition in the sample will be replayed, updating the Q-values. This way, the predictions and policy of the agent receive additional updates without interaction with the environment.
The advantage of replay is thus that uses previous experience more efficiently than Q-learning. This is key for situations where experience is costly. With the additional updates in replay, the Q-values can converge more quickly, needing less experience. Furthermore, when training Neural Networks, it is important to make sure old information is not forgotten. With replay, each time you create a new sample, you also train on old samples, this way the old information is not lost. Replay is thus most useful in those situations where it is important to make sure that old information is not forgotten.
Just as with Q-learning, one needs to think about the values for the hyper-parameters used in replay. These parameters are the same as in Q-learning plus the parameter denoting the sample-size. The sample-size determines how many transitions will be sampled in each iteration. For the hyper-parameters ϵ, γ and α, we use the values mentioned above. We will use a sample-size of 100.

Algorithm 2 Replay

```
Input: policy $\pi$ [a]
Input: mdp [b]
for $iteration = 1, 2, \dots, R$ do
  Generate state: $s \leftarrow$ state-generator
  for $interaction = 1, 2, \dots, S$ do
    Generate action: $a \leftarrow \pi(s)$
    Generate reward, next state: $r, s' \leftarrow$mdp$(s, a)$
    Store $s, a, r, s'$
    Sample $\leftarrow GenerateSample(s, a, r, s')$ [c]
    for $s, a, r, s'$ in Sample do
      $\pi \leftarrow UpdateAgent(s, a, r, s')$ [d]
    end for
    update current state: $s = s'$
    if $s$ is final [e] then
      break
    end if
  end for
end for
```

[a] π is the policy used to determine the next action.
[b] mdp is a function that generates a new transition.
[c] The *GenerateSample* function generates a sample of transitions from the stored transitions.
[d] The $UpdateAgent$ function updates the Q-values.
[e] s is final when a stable argumentation setup has been achieved (section 2.3).

4.3 Model-based Q-learning

In model-based Q-learning Q-learning is combined with a model. Using this model, extra transitions are created. These transitions are then used to update the Q-values. It differs from replay in that it does not just sample old transitions, but it can also generate new transitions.
When making a model, the choices are infinite. For each problem, multiple models could be used, and each model can be implemented in different ways. In this thesis we assume the most ideal situation; that we know what the structure of the probability function is. In addition, for the scale of the experiments, we assume that these are the independent, not dependent, probabilities per proposition. Even with these assumptions, there are multiple ways to make the model. In the next paragraph we explain how we created our model.
In our MDP, the reward function is known. This means that only a model for the transition probabilities is required. In our domain, each action can only lead to two possible states, one where the observable is true and one where it is false. Because of this, the transition probability $P^a_{s,s'}$ where s' is the state where the observable is true, is the same as the probability of s' being true. The transition probability $P^a_{s,s'}$ where s' is the state where the observable is false, is then the same as the probability of s' being false, or in other words, one minus the probability of s' being true. In order to learn the transition probabilities, we thus only need to learn the probabilities of the observables.
Our model learns these probabilities by counting how often an action is executed, and counting the number of times that action resulted in the next state where the observable is true. This model will eventually converge to the correct probabilities,

meaning it is a perfect model and that it can find the optimal policy.
The probabilities of the observables can be determined by us. By using a weighted case-generator, we can generate cases where the probabilities of the observables are the same as the weights.
Just as with Q-learning and replay, one needs to think about the values for the hyper-parameters. These parameters are the same as in replay, here we will use the same values as used for replay.

Algorithm 3 Model-based Q-learning

```
Input: policy $\pi$ [a]
Input: mdp [b]
Input: model $m$
for $iteration = 1, 2, \dots, R$ do
    Generate state: $s \leftarrow$ state-generator
    for $interaction = 1, 2, \dots, S$ do
        Generate action: $a \leftarrow \pi(s)$
        Generate reward, next state: $r, s' \leftarrow$mdp$(s, a)$
        $m \leftarrow UpdateModel(s, a, r, s')$ [c]
        Sample $\leftarrow GenerateSampleByModel(s, a, r, s')$ [d]
        for $s, a, r, s'$ in Sample do
            $\pi \leftarrow UpdateAgent(s, a, r, s')$ [e]
        end for
        update current state: $s = s'$
        if $s$ is final [f] then
            break
        end if
    end for
end for
```

[a] π is the policy used to determine the next action.
[b] mdp is a function that generates a new transition.
[c] This function updates the probabilities for the transition function.
[d] This function generates a sample of transitions generated by the model.
[e] The *UpdateAgent* function updates the Q-values.
[f] s is final when a stable argumentation setup has been achieved (section 2.3).

4.4 The experiment

In our experiment we are going to compare the three algorithms mentioned above. This will be done by using the setup of example 1 in section 2.4.1. In this section the precise setup of the experiment will be explained.

First of all, the experiment will consist of a baseline and an adverse scenario. In the baseline scenario we will compare the performance of the algorithms based on the reward they eventually receive and with which speed the algorithms learn their policy. This scenario thus looks at the intuitive measurements of performance.
Our research questions pertains the performance of model-based reinforcement learning. Performance can be measured in different ways. To highlight this, we also use an adverse scenario in which we will look at another way to measure performance, recoverability. Recoverability look at how well each algorithm can recover once the

weights change and a different policy needs to be learned. Recoverability is chosen as another measure to compare performance because this quality is important in the domain of online fraud and in other real world scenarios. In the real world, probabilities change. The algorithm needs to be able to recover when that happens, otherwise, it will be useless after one change. In the case of online-fraud the probabilities could change for example when a lot more people start to commit online fraud. Because of this, we also look at the recoverability of the algorithms.

Let us first look at the baseline scenario. In this scenario we use the MDP that was used in example 1, this MDP is representative for the real-world. The algorithms will be trained on this MDP to see the performance of each algorithm. In order for us to be able to train the algorithms, we need to know the observables, the topic, the queries and the rules. The first three are mentioned in example 1. The rules were not yet explicitly written down. The rules are determined by looking at figure 2.2. We know that in each end-state, the topic is known. So to be able to determine the rules, we just need to look at which observables are in the knowledge base in those states. The observables are then the premises of the rule, while the topic is the conclusion. This leads to the following setup:

- rules: $[(a, b => t), (a, \neg b, c => t), (a, \neg b, \neg c, d => t), (a, \neg b, \neg c, \neg d => t), (\neg a, \neg d => t), (\neg a, d, b => t), (\neg a, d, \neg b, c => t), (\neg a, d, \neg b, \neg c => t)]$
- observable = [a, b, c, d]
- topics = [t]
- queries = [A,B,C,D]

With this setup we can do the experiment. In the baseline scenario the weights will stay the same throughout the rounds. Each algorithm will be trained with two sets of weights.

In the adverse scenario we will use different sets of weights as in the baseline scenario, and a different rule set. With these weights and rules we can best test recoverability. Given the rules and weight set, we know that there is one correct order of actions that the agent should learn. After a certain number of rounds, there will be a switch in which set of weights is used. In this new set, the order of actions is the reversed. The agent then has to learn a completely different policy. This way, we can see how well the algorithms can recover, how quickly they find the new optimal policy. We will run multiple adverse scenarios. At first, the weights will only be switched once, then twice and lastly they will be switched three times. The following rules and weights will be used:

- rules: [(a=>t), (b=>t), (c=>t), (d=>t)]
- weights 1: [.2, .4, .6, .8]
- weights 2: [.8, .6, .4, .2]

The weights correspond to the observables in the following order: $[a, b, c, d]$. Thus with weights 1, observables a has a 0.2 percent change of being true. As mentioned, these weights lead to one correct order of actions regardless of the truth-value of the observable, for weights 1 this is $[D, C, B, A]$. With these rules and weights, we know that the algorithms should have the same performance with both weight sets.

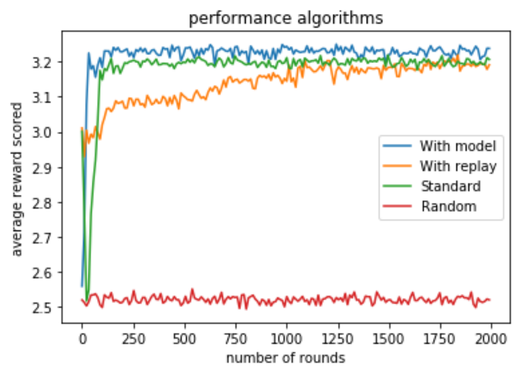
Furthermore, there is a big change in policy. This way, the recoverability of the algorithms can best be tested. With the rules of the baseline scenario, there is no two sets of weights which give the same score. This way, changing the weights leads to switching to another average reward scored, and the recoverability would be less clear.

Chapter 5

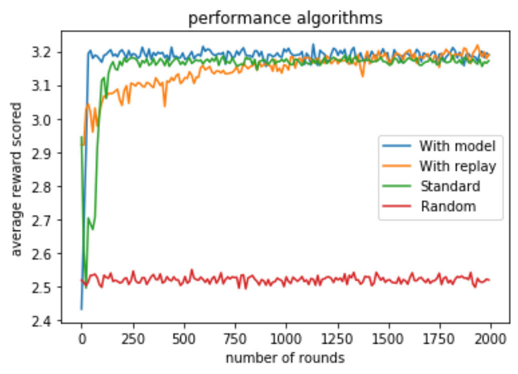
Results

5.1 Performance algorithms

First, we tested the algorithms on two different weight sets. The results of which are shown in figure 5.1. The reward scored is shown on the y-axis. This is the average reward scored over a 1000 iterations. The number of rounds are shown on the x-axis. So for instance if $x = 100$ then the y value is the result of freezing the agent after training for 100 episodes, running consequently the agent for 1000 episodes and then the y value is the average reward over those 1000 episodes. Figure 5.1 shows that all three algorithms learn the optimal policy, eventually they all receive the same average reward. However, not all algorithms need the same amount of time for training before they have found the optimal policy. In both 5.1a and 5.1b we see that model-based Q-learning quickly learns the optimal policy. They also show that replay progresses faster than standard Q-learning at first. However, after a couple of rounds, the progress of replay declines, and standard Q-learning reaches the optimal policy faster.



(A) Performance of the algorithms with weights: $[0.4, 0.66, 0.5, 0.7]$, iterations = 2000.



(B) Performance of the algorithms with weights: $[0.7, 0.4, 0.66, 0.5]$, iterations = 2000.

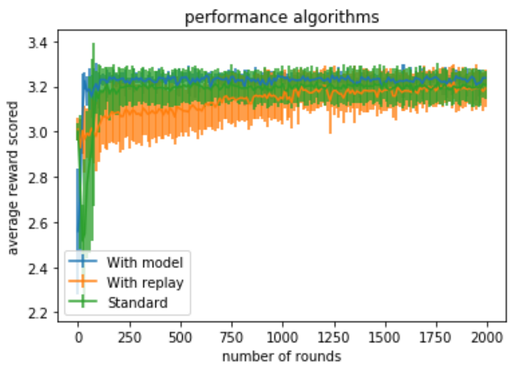
FIGURE 5.1: The baseline performances with two different weight sets.

In table 5.1 the standard deviation for each algorithm is shown per weight set. In figure 5.2 the error bars are added to the graphs. The random algorithm has a large standard deviation, for this reason it is not included in figure 5.2. Replay and Q-learning both have a larger standard deviation than Model-based. Model-based has a small standard deviation, which means that at the reward scored after training is stable. The agent receives about the same reward each time. The standard deviation of Replay and standard Q-learning is a bit bigger. The reward scored is thus less

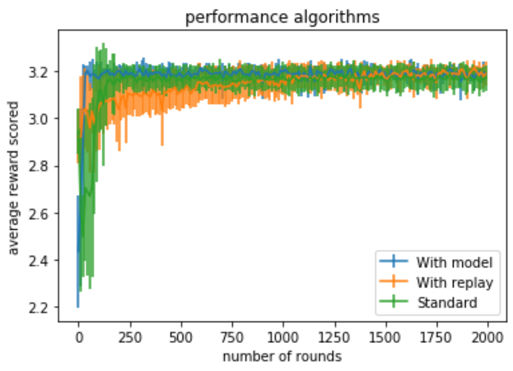
Algorithm	sd with first weight set	sd with second weight set
Model-based	0.037	0.040
Replay	0.102	0.070
Standard Q-learning	0.087	0.069
Random	0.661	0.661

TABLE 5.1: The average standard deviation for each algorithm per weight set

stable, for example, the agent sometimes receives a reward of 3.2 and in another test-round a reward of 3.0.



(A) Performance of the algorithms with weights: [0.4, 0.66, 0.5, 0.7], iterations = 2000.

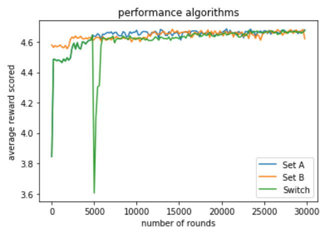


(B) Performance of the algorithms with weights: [0.7, 0.4, 0.66, 0.5], iterations = 2000.

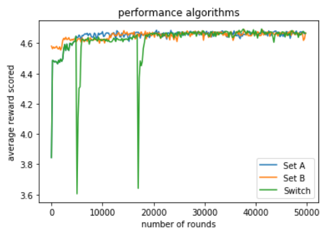
FIGURE 5.2: The baseline performances with two different weight sets and the standard deviation.

5.2 Recoverability

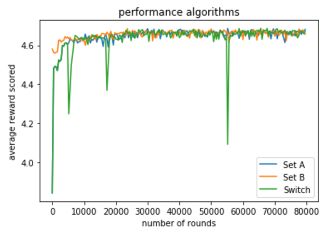
In figures 5.3, 5.4, 5.5 the recoverability of each algorithm is shown. The average reward scored is again shown on the y-axis, and the number of rounds on the x-axis. In figure 5.3 the recoverability of standard Q-learning is shown. In figure 5.3a the weights are switched one time, in figure 5.3b twice and in 5.3c three times. These graphs show that standard Q-learning recovers quite fast after a change in weights. Furthermore, after each switch it takes about the same amount of time to recover. Is does not take longer after more iterations.
If we look at figure 5.4 we see the recoverability of replay. Here we also have figures for one switch, two switches and three switches. Replay takes quite a while longer to recover than standard Q-learning. Furthermore, after each switch it takes longer to recover the optimal policy.
Model-based Q-learning, as shown in figure 5.5 recovers similar to replay. It also takes a lot longer than standard Q-learning, and each time it takes longer to recover.



(A) Standard Q-learning with one switch: from weights [.2, .4, .5, .8] to [.8, .6, .4, .2], iterations = 30000.

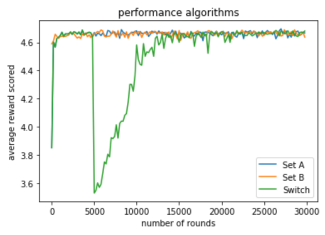


(B) Standard Q-learning with two switches: from weights [.2, .4, .5, .8] to [.8, .6, .4, .2] and back, iterations = 50000.

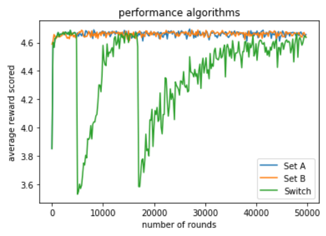


(C) Standard Q-learning with three switches: from weights [.2, .4, .5, .8] to [.8, .6, .4, .2] and back, and to [.8, .6, .4, .2] again, iterations = 80000.

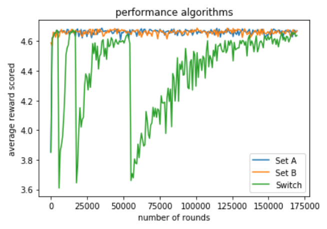
FIGURE 5.3: The recoverability of standard Q-learning



(A) Replay with one switch: from weights [.2, .4, .5, .8] to [.8, .6, .4, .2], iterations = 30000.

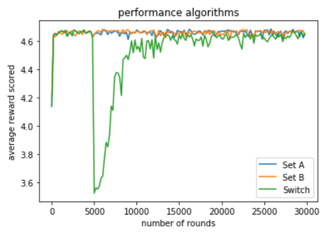


(B) Replay with two switches: from weights [.2, .4, .5, .8] to [.8, .6, .4, .2] and back, iterations = 50000.

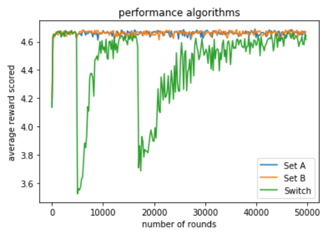


(C) Replay with three switches: from weights [.2, .4, .5, .8] to [.8, .6, .4, .2] and back, and to [.8, .6, .4, .2] again, iterations = 17000.

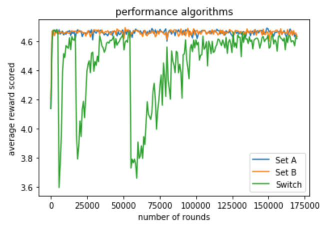
FIGURE 5.4: The recoverability of Replay



(A) Model-based with one switch: from weights [.2, .4, .5, .8] to [.8, .6, .4, .2], iterations = 30000.



(B) Model-based with two switches: from weights [.2, .4, .5, .8] to [.8, .6, .4, .2] and back, iterations = 50000.



(C) Model-based with three switches: from weights [.2, .4, .5, .8] to [.8, .6, .4, .2] and back, and to [.8, .6, .4, .2] again, iterations = 17000.

FIGURE 5.5: The recoverability of Model-based Q-learning

Chapter 6

Discussion

Let us first discuss the results of the baseline performances.
In chapter 5 we saw that all three algorithms find the optimal policy. However, they did not find the optimal policy at the same speed. Theoretically, this is what was expected. Since we have the perfect model, we know that it will find the optimal policy. For the other two algorithms we know that if they have enough time to train, the Q-values will converge, and they too will find the optimal policy. Theoretically it was thus expected that all three would find the optimal policy. Furthermore, the fact that the speed at which the policy is found differ, can be explained as well.
With model-based Q-learning the agent can look ahead and create new transitions, because of this, it can find the optimal policy very fast. In small MDPs like the one used here, it is often the case that the agent has to do the right action once, after which the reward is a lot higher. Since model-based Q-learning creates multiple new transitions with every move, it can find that action very quickly, and with that, the optimal policy.
Standard Q-learning is not far behind model-based Q-learning. It also finds the optimal policy quite rapidly. For standard Q-learning it is also the case that it just needs to find the right action. This can happen fast in a small MDP. However, since the agent cannot create new transitions, it does take a bit longer than model-based Q-learning. In a bigger MDP the difference would be more clear. In a larger MDP, more transitions are needed for the Q-values to converge to the optimal policy. In standard Q-learning, only one transition is done in each move that leads to updates to the Q-values. However, in model-based Q-learning, multiple transitions are done per move, each resulting in updates to the model and Q-values. Because of this, model-based Q-learning can learn the optimal policy in less rounds.
Replay rises fast in the beginning as well but the curve flattens after a while. Replay learning cannot create new transitions like model-based Q-learning can. Because of this, replay is slower to find the optimal policy. The figure also shows that replay is slower than standard Q-learning. This is a bit more surprising. Replay uses additional computation to update the Q-values more often. However, it seems that in our MDP this does not improve the performance. It could be that replay keeps sampling old transitions. This way, the Q-values of the newer transitions are not updated, or not very often. The agent then has trouble finding the one right action. This way, it takes a long time to find the optimal policy for the last few steps in the MDP. In the beginning, there are few transitions, because of this the newer ones will also be updated. However, when there are more transitions stored, this chance diminishes. This explains why replay is fast at first, but declines after a while.

If we look at the recoverability of the algorithms, we can see that standard Q-learning recovers far quicker than the other two algorithms. This is because replay and model-based Q-learning are data methods. They save data and use it to learn the

policy. However, when the weights change, the policy changes. The stored data at that point is no longer representative for the new situation. A lot of new data has to be collected for it to compensate for the old data and to represent the new situation. This takes a lot of rounds, at least as many rounds as were run before the switch as seen in figures 5.4a and 5.5a. When more switches take place, or the switch happens later, it takes longer and longer for these algorithms to recover, since they have more and more old data stored.
Standard Q-learning can recover more quickly. This algorithm only needs to update the Q-values. This can be done relatively quick.
A way as to speed up the recovery of replay and model-based Q-learning could be to put a limit on the amount of data that is stored. If there is a limit on the amount of data stored, there is less "old" information that has to be compensated for. Theoretically, the algorithms can then recover faster since less new data is needed to represent the new situation. Furthermore, the speed of recovery should not be negatively impacted when there are more switches or when the switch happens later, since there is not more data stored than before. This hypothesis could be the focus of subsequent research.

In this thesis we focused on comparing the algorithms from a proof-of-concept perspective. Because of this no hyper-parameter tuning was carried out. Performing extensive hyper-parameter tuning is another possibility for subsequent research. As mentioned in chapter 4 we have four parameters, ϵ, α, γ and the sample-size for replay and model-based Q-learning. Different values for these parameters could change the performance of the algorithms.
For example, with a different discount factor, γ, one could make future rewards more or less important. This could influence the agent's actions. With a smaller discount factor, the agent will prefer immediate rewards, with a larger discount factor the agent will care more about the high reward, even though it is a couple of steps away. Changing the discount factor could thus lead to a different score. Furthermore, standard Q-learning could theoretically recover even faster with a larger adaptation factor (α), since the Q-values will be updated more quickly.
These parameters thus have a substantial influence on the performance of the algorithms. With extensive hyper-parameter tuning one could see exactly what the influence of each parameter has on each algorithm, furthermore, one could find the optimal values for each algorithm. Once these values are found, the performance of the algorithms will be optimal. The performance of the algorithms can then be compared to see if the hyper-parameter tuning leads to a different conclusion. It is not very likely that the performance of the algorithms will change drastically, since we did not choose the parameters at random, but did think about their values. However, there might be some interesting changes in the performance.

Chapter 7

Conclusion

In this thesis we compared model-based with model-free methods for the domain of argument-based inquiry to answer the following research question:

"Does model-based q-learning perform similar to model-free approaches for argument-based inquiry?"

To answer this question, we first answered the subquestions.

To answer the first subquestion, we looked at what the literature said about the circumstances one approach would be preferred over the other. In section 3.3 we found that model-based methods should work best when you have a problem with a discrete domain and a limited amount of available actions, an idea about what a realistic model for the problem looks like, and enough data to generate the model. However, model-based methods could be disastrous when the model starts generating false transitions.
For subquestion two we compared this conclusion to the characteristics of our MDP in section 3.3.1. From this we concluded that for our problem it is the case that we have a discrete domain with a limited amount of available actions, an idea for a perfect model of the environment, and, if we train the algorithm for enough rounds, we have enough transitions to generate the model. This knowledge, together with conclusion of section 3.3, led to the hypothesis that a model-based approach should theoretically work best for our argument-based inquiry domain.

In chapter 5 we tested this hypothesis by running simulations of inquiries, thereby answering subquestion 3. From these results we can conclude that model-based Q-learning has a similar performance as standard Q-learning, it learns the optimal policy just like standard Q-learning and even does it in less rounds. This answers our research question. Model-based Q-learning does indeed perform similar to model-free approaches for argument-based inquiry.
However, there are some trade-offs with model-based Q-learning. Model-based Q-learning recovers far less quickly than standard Q-learning, and it makes the system more complex.
Taking all factors into account, the transparency of the algorithms, the baseline performance and the recovery of the algorithms, which approach should be preferred in the domain of online fraud complaints?
The performance and transparency is very important for the police. Recoverability is also important, since probabilities change in the real world. However, it is more important that the algorithm recovers, than that it recovers quickly. The probabilities do not considerably change in a week. Therefore, it is acceptable if the training after a change in weights takes a bit longer if it comes with improved transparency. Furthermore, it is likely that the recoverability of model-based Q-learning can be

improved by simple measures as explained in the discussion. Standard Q-learning probably will still recover more quickly, but the difference will be smaller.
Thus, given the importance of performance and transparency at the police, I believe that slow recovery is acceptable and that in the domain of online fraud complaints, model-based Q-learning is the preferred method.

References

[1] Floris Bex, Joeri Peters, and Bas Testerink. “AI for online criminal complaints: From natural dialogues to structured scenarios”. In: *Artificial Intelligence for Justice Workshop (ECAI 2016)*. 2016, p. 22.

[2] Phan Minh Dung. “On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games”. In: *Artificial intelligence* 77.2 (1995), pp. 321–357.

[3] Hado P van Hasselt, Matteo Hessel, and John Aslanides. “When to use parametric models in reinforcement learning?” In: *Advances in Neural Information Processing Systems*. 2019, pp. 14322–14333.

[4] Henry Prakken. “An abstract framework for argumentation with structured arguments”. In: *Argument and Computation* 1.2 (2010), pp. 93–124.

[5] D Silver et al. “Mastering the game of Go without human knowledge”. In: *nature* 550 (2017), pp. 354–359.

[6] Richard S Sutton and Andrew G Barto. “Reinforcement learning: An introduction”. In: (2011).

[7] Bas Testerink, Daphne Odekerken, and Floris Bex. “A Method for Efficient Argument-Based Inquiry”. In: *International Conference on Flexible Query Answering Systems*. Springer. 2019, pp. 114–125.

[8] Bas Testerink, Daphne Odekerken, and Floris Bex. “AI-assisted message processing for the Netherlands National Police”. In: (2019).

[9] NM Uijlen. “Applying reinforcement learning to argument-based inquiry”. MA thesis. 2019.

[10] Marco Wiering and Martijn Van Otterlo. “Reinforcement learning”. In: *Adaptation, learning, and optimization* 12 (2012), p. 3.