CHALMERS

Reinforcement learning

A comparison oflearning agents in environments with large discrete state spaces

Bachelor's thesis in Computer Science

JOHAN ANDERSSON
EMIL KRISTIANSSON
JOAKIM PERSSON
ADAM SANDBERG ERIKSSON
DANIEL TOOM
JOPPE WIDSTAM

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sweden 2014

BACHELOR'S THESIS IN COMPUTER SCIENCE

Reinforcement learning

A comparison of learning agents in environments with large discrete state spaces

JOHAN ANDERSSON
EMIL KRISTIANSSON
JOAKIM PERSSON
ADAM SANDBERG ERIKSSON
DANIEL TOOM
JOPPE WIDSTAM

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY

Göteborg, Sweden 2014

Reinforcement learning
A comparison of learning agents in environments with large discrete state spaces
JOHAN ANDERSSON
EMIL KRISTIANSSON
JOAKIM PERSSON
ADAM SANDBERG ERIKSSON
DANIEL TOOM
JOPPE WIDSTAM

Examiner: JAN SKANSHOLM

Bachelor's thesis 2014:05
ISSN 1654-4676
Department of Computer Science and Engineering
Chalmers University of Technology
SE-412 96 Göteborg
Sweden
Telephone: +46 (0)31-772 1000

Chalmers Reproservice
Göteborg, Sweden 2014

Abstract

For some real-world optimization problems where the best behavior is sought, it is infeasible to search for a solution by making a model of the problem and performing calculations on it. When this is the case, good solutions can sometimes be found by trial and error. Reinforcement learning is a way of finding optimal behavior by systematic trial and error. This thesis aims to compare different reinforcement learning techniques and evaluate them. Model-based interval estimation (MBIE) and Explicit Explore or Exploit using dynamic bayesian networks (DBN-E^3) are two algorithms that are evaluated. To evaluate the techniques, learning agents were constructed using the algorithms and then simulated in the environment Invasive Species from the Reinforcement Learning Competition. The results of the study show that an optimized version of DBN-E^3 is better than MBIE at finding an optimal or near optimal behavior policy in Invasive Species for a selection of environment parameters. Using a factored model like a DBN shows certain advantages operating in Invasive Species, which is a factored environment. For example it achieves a near optimal policy within fewer episodes than MBIE.

Keywords: Reinforcement learning, Artificial intelligence, MBIE, DBN-E^3, Invasive Species, Learning algorithms

Sammanfattning

För vissa verklighetsbaserade optimeringsproblem där det gäller att finna ett bästa beteende är det orimligt söka efter det optimala beteendet genom att skapa en modell av problemet och sedan utföra beräkningar på den. När så är fallet, kan man hitta bra lösningar genom trial and error. Reinforcement learning är ett sätt att hitta optimalt beteende genom att systematisk pröva sig fram. Detta kandidatarbete har som mål att jämföra olika inlärningstekniker och utvärdera dessa. Model-based interval estimation (MBIE) och Explicit Explore or Exploit med dynamiska bayesnät (DBN-E^3) är två självlärande algoritmer som här utvärderas. För att utvärdera de olika teknikerna implementerades agenter som använde algoritmerna, och dessa testkördes sedan i miljön Invasive Species från tävlingen Reinforcement Learning Competition. DBN-E^3 är bättre än MBIE på att hitta en optimal eller nära optimal policy i Invasive Species med valda parametrar. Att använda en faktoriserad modell, så som DBN-E^3 visar på tydliga fördelar i faktoriserade miljöer, som i till exempel Invasive Species. Ett exempel på detta är att den når en nästan optimal policy på färre episoder än MBIE.

Contents

Chapter 1

Introduction

Reinforcement learning, a subfield of artificial intelligence, is the study of algorithms that learn how to choose the best actions depending on the situation. In a reinforcement learning problem the algorithm is not told which actions give the best results, but instead it has to interact with the environment to learn when and where to take a certain action. When the agent has learned about each situation or state of the environment, it will arrive at an optimal sequence of actions (Barto and Sutton 1998).

Reinforcement learning can be exemplified by how a newborn animal learns how to stand up. There is no tutor to teach it, so instead it tries various combinations of movements, while remembering which of them lead to success and which of them lead to failure. Probably, at first, most of its attempted movement patterns will lead to it falling down — a negative result, making the animal less likely to try those patterns again. After a while, the animal finds some combination of muscle contractions that enables it to stand — a positive result, which would make it more likely to perform those actions again.

In the real world, in order to decide what actions lead to success, one almost always needs to consider the circumstances, or the state of the environment, in which the actions are taken. For instance, a person might be rewarded if they sang beautifully at a concert, but doing the same at the library would probably get them thrown out. Furthermore, for many real world domains it is common that the state space (the set of possible states of the environments in which actions can be taken) is very large (Guestrin et al. 2003). To continue our example, there are probably countless places and situations where it is possible to sing. In this case, how does one abstract and find the qualities of the environment that are important for deciding on the optimal action?

There are two practical problems connected to reinforcement learning in large state spaces. First, it is hard to store representations of the states of the environment in the main memory of a computer (Szepesvári 2010). Second, it takes too long to repeatedly visit all the states to find the best action to take in all of them (Dietterich, Taleghan, and Crowley 2013).

1.1 Purpose and problem statement

This thesis aims to further the knowledge of reinforcement learning or more specifically, algorithms applied to reinforcement learning problems with large state spaces. Reinforcement learning algorithms struggle with environments consisting of large state spaces due to practical limitations in memory usage and difficulties in estimating the value of actions and states within reasonable processing time. Therefore, we aim to evaluate techniques to circumvent or reduce the impact of these limitations. This is done by implementing algorithms that use these techniques and comparing the algorithms in an environment with a large state space.

Limitations Due to time constraints only two algorithms are tested, which apply different techniques. Furthermore, the evaluation of the selected algorithms is done in only one environment covering a certain problem with a large state space.

Simulation environment The environment used is Invasive Species from the Reinforcement Learning Competition, described in section 3.1. Depending on the parameters, the problem can have both a large state space and/or a large action space. This makes Invasive Species a good choice with the given problem statement.

Agents to be evaluated Two algorithms that are suitable to the purpose and problem statement, since they apply different techniques for working with large state spaces, are Explicit Explore or Exploit in dynamic bayesian networks (DBN-E^3) and model-based interval estimation (MBIE). They are described in detail in chapter 3.

Chapter 2

Technical background

The term artificial intelligence was coined by John McCarthy who described it as “the science and engineering of making intelligent machines” (McCarthy 2007). More specifically, it addresses creating intelligent computer machines or software that can achieve specified goals computationally. These goals can comprise anything, e.g. writing poetry, playing complex games such as chess or diagnosing diseases. Different branches of artificial intelligence include planning, reasoning, pattern recognition and the focus of this thesis: learning from experience. This chapter gives a more formal description of reinforcement learning and the main topics and concepts necessary to understand the rest of the report.

2.1 Reinforcement learning

In artificial intelligence, reinforcement learning is the problem of learning from experience. A reinforcement learning algorithm uses past experiences and domain knowledge to make intelligent decisions about future actions (Barto and Sutton 1998).

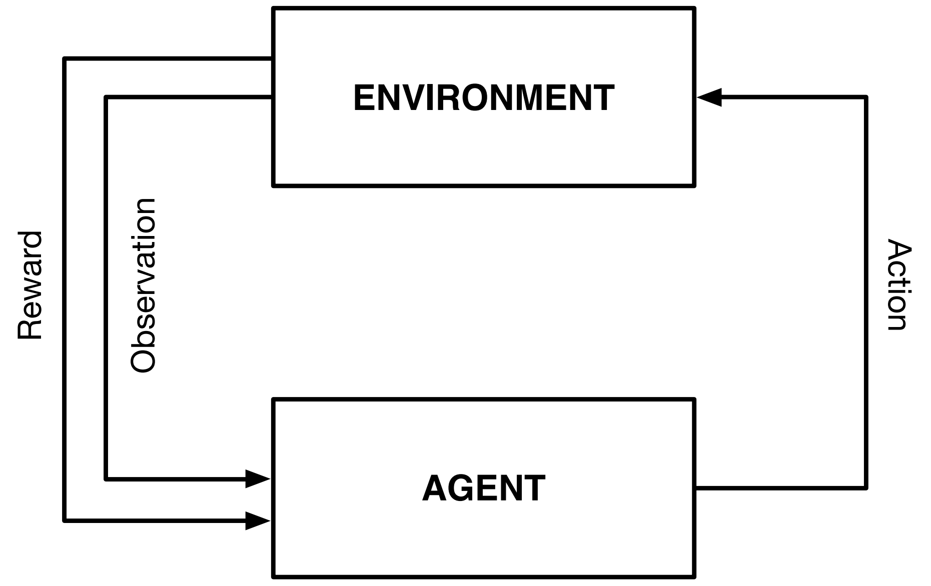


Figure 2.1: *The reinforcement learning process*

The reinforcement learning problem is modeled as a sequential decision problem, see figure 2.1 for a graphical representation of the process. A learning agent performs an action and receives a reward according to some measure of how desirable the results of the action are. After the action is taken, the state of the environment changes and the agent then receives a new observation and the process repeats. The goal of the reinforcement learning agent is to maximize the reward received over a certain time period, the horizon, finding a balance between immediate and future rewards (Barto and Sutton 1998).

There are several ways of further dividing reinforcement learning problems into other subcategories. Some examples are the dichotomies episodic/non-episodic problems, problems with continuous/discrete state spaces, problems with continuous/discrete action spaces, problems with one/multiple concurrent agents etc. In the text below, the two first of these dichotomies, which are relevant to this thesis, are discussed.

2.1.1 Episodic and non-episodic problems

One can categorize reinforcement learning problems based on whether or not the time steps are divided into episodes. If they are, the problem is called episodic and otherwise it is called non-episodic. Episodic problems are common in for example games, which end when they are won or lost. An episode here corresponds to a full game from start to finish. In a non-episodic problem, the interaction between the actor and environment goes on continuously without end (Barto and Sutton 1998). Two examples of non-episodic applications are controlling a robot arm (as well as other robotics problems) and maneuvering a helicopter (Ng et al. 2006).

2.1.2 Continuous and discrete problems

Reinforcement-learning problems can also be categorized by their state spaces. Some problems have a discrete state space whereas for some problems the state space is continuous. An important difference between the two kinds of problems is how an agent can treat similar states in the model. In a continuous problem it is much easier to classify states located around the same "position" as belonging to the same group, since the states usually more or less meld together. For example, if the reinforcement learning problem is to control a robot arm whose position constitutes the state of the environment, states corresponding to positions in the same general area are not very different. This means that these states might be treated in the same way or a similar way by an agent. In contrast, consider the discrete problem of a game of chess, wherein the placement of a pawn in one of two adjacent squares could dramatically change the evaluation of the state and the outcome of the game (Barto and Sutton 1998).

2.2 Markov decision process

Within reinforcement learning, the concept of Markov decision processes (MDP) is central. An MDP is a way to model an environment where state changes in it are dependent on both random chance and the actions of an agent. An MDP is defined by the quadruple $(S, A, P(\cdot,\cdot,\cdot), R(\cdot,\cdot))$ (Altman 2002):

Definition 2.1. MDP

S A set of states that the environment can be in.

A A set of actions that are allowed in the MDP.

$P\colon S \times A \times S \to [0, 1]$ A probability distribution over the transitions in the environment. This function describes the probability of ending up in a certain target state when a certain action is taken from a certain origin state.

$R\colon S \times A \to \mathbb{R}$ A function for the reward associated with a state transition. In some definitions of MDPs the reward function only depends on the state.

MDPs are similar to Markov chains, but there are two differences. First, there is a concept of rewards in MDPs, which is absent in Markov chains. Second, in a Markov chain, the only thing that affects the probabilities of transitioning to other states is the current state, whereas in an MPD both the current state and the action taken in that state are needed to know the probability distribution connected with the next state (Altman 2002).

If there are only a few states s' for which $P(s, a, s') > 0$, the MDP is called a sparse MDP. That is to say, when an agent performs a certain action, a, in a certain state, s, the environment can only end up in a small fraction out of the total number of states (Dietterich, Taleghan, and Crowley 2013).

2.2.1 Utility

The concept of utility is used as a measurement of how well an agent performs over time. In the simplest case, the utility is just the sum of all rewards received in an episode. See equation (2.1), where T is the length of an episode and r_k is the expected reward at time step k.

$$U_0 = r_0 + r_1 + \cdots + r_{T-1} = \sum_{k=0}^{T-1} r_k \tag{2.1}$$

However, in a non-episodic environment, the interaction between the environment and the agent can go on forever. Here the concept of a discount factor, γ, is essential. This parameter is a value between 0 and 1 which describes how fast the value of expected future rewards decays as one looks further into the future (Barto and Sutton 1998). Thus, the utility can be expressed as in equation (2.2) in the case that discounting is used.

$$U_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{2.2}$$

2.2.2 Markov property

A defining characteristic of an MDP is the Markov property—that, given the current state of the environment, one cannot gain any more information about its future behavior by also considering the previous actions and states it has been in. Equation (2.3) defines the Markov property: the probability of state s_{t+1} only depends on the previous state s_t and action a_t.

$$\Pr(s_{t+1}|s_t, a_t, \ldots, s_1, a_1) = \Pr(s_{t+1}|s_t, a_t) = P(s_t, a_t, s_{t+1}) \tag{2.3}$$

This can be compared to the state of a chess game, where the positions of the pieces at any time completely summarizes everything relevant about what has happened previously in the game. That is, no more information about previous moves or states of the board is needed to decide how to play or predict the future outcome of the game (disregarding psychological factors). A chess MDP that uses a chess board as its state representation could thus be an example of an MDP with the Markov property (Altman 2002).

2.2.3 Representations

Two ways to represent an MDP are extensional and factored representation. Depending on what problem domain is used, it can be advantageous to use a factored representation due to computational considerations(Boutilier, Dean, and Hanks 1999).

Extensional representation The most straightforward way to model an MDP is called extensional representation, where the set of states and actions are enumerated directly. It is also commonly referred to as an explicit representation and this is the definition we have used so far in the report when discussing the abstract view of an MDP (Boutilier, Dean, and Hanks 1999).

Factored representation A factored representation of the states of an MDP often results in a more compact way of describing the set of states. Certain properties or features of the states are used to factor the state space into different sets. Which properties or features are used is chosen by the algorithm designer, to fit the environment. Formally a state s is in a state space $S = S_1 \times \cdots \times S_k$ of k factors, with the state at time t being $s_t = (s_t(1), \ldots, s_t(k))$.

When the MDP is factored, it enables factored representations of states, actions and other components of the MDP as well. When using a factored action representation, an action can be taken based on specific state features instead of on the whole state. If the individual actions affect relativity few features or if the effects contain regularities then using a factored representation can result in compact representations of actions (Boutilier, Dean, and Hanks 1999).

2.3 Basic algorithms for solving MDPs

A policy π is a function from a state s to an action a that operates in the context of a Markov decision process, i.e. $\pi\colon S \to A$. A policy is thus a description of how to act in each state of an MDP. An arbitrary policy is denoted by π and the optimal policy (the policy with the maximal utility in the MDP) is denoted by π^*. The rest of this section describes some basic algorithms for solving an MDP, that is to say, finding an optimal policy for it (Barto and Sutton 1998).

2.3.1 Value functions

To solve an MDP, most algorithms use an estimation of values of states or states and actions. Two value functions are usually defined, the state-value function $V : S \to \mathbb{R}$ and the state-action-value function $Q : S \times A \to \mathbb{R}$. As the names imply, V signifies how good a state is, while Q signifies how good an action in a state is. The state-value function, $V^\pi(s)$, returns the expected value when starting in state s and following policy π thereafter. The state-action-value function, $Q^\pi(s, a)$, returns the expected value when starting in state s and taking action a and thereafter following policy π. The value functions for the optimal policy, the policy that maximizes the utility, are denoted by $V^*(s)$ and $Q^*(s, a)$ (Barto and Sutton 1998).

Equations (2.4) and (2.5) show the state-value function V and the state-action-value function Q defined in terms of utility (section 2.2.1). The future states, s_t, are stochastically distributed according to the transition probabilities of the MDP and the policy π (see section 2.2).

$$V_t^\pi(s) = \mathbb{E}\left\{ U_t \middle| \, s_t = s \right\} \tag{2.4}$$

$$Q_t^\pi(s, a) = \mathbb{E}\left\{ U_t \middle| \, s_t = s, a_t = a \right\} \tag{2.5}$$

Both V^π and Q^π can be estimated from experience. This can be done by maintaining the average of the rewards that have followed each state when following the policy π. When the number of times the state has been encountered goes to infinity, the average over these histories of rewards converges to the true values of the value function (Barto and Sutton 1998).

Using dynamic programming to find V and Q Another way to find value functions is to use dynamic programming techniques. Dynamic programming is a way of dividing a problem into subproblems that can be solved independently. If the result of a particular subproblem is needed again, it can be looked up from a table (Bellman 1957). Examples of dynamic programming algorithms are policy iteration and value iteration, which are discussed in sections 2.3.2 and 2.3.3. These algorithms are often the basis for more advanced algorithms, among them the ones described in chapter 3.

2.3.2 Policy iteration

Policy iteration is a method for solving an MPD that will converge to an optimal policy and the true value function in a finite number of iterations if the process is a finite MDP. The algorithm consists of three steps: initialization, policy evaluation and policy improvement (Barto and Sutton 1998).

Initialization

Start with an arbitrary policy π and arbitrary value function V_0.

Policy evaluation

Compute an updated value function, V, for the policy π in the MDP by using the update rule (2.6) until V converges. Convergence means that $|V_{k+1}(s) - V_k(s)| \leq \epsilon, \forall s$, where ϵ is chosen as a small value. In the update rule, $\pi(s,a)$ is the probability of taking action a in state s using policy π.

$$V_{k+1}(s) = \sum_a \pi(s,a) \sum_{s'} P(s,a,s') \left[R(s,a) + \gamma V_k(s')\right] \tag{2.6}$$

Policy improvement

Improve the policy by making it greedy with regard to V, equation (2.7)[1]. This means that the policy will describe the action in each state that maximizes the expected V-value of the following state.

$$\pi_{k+1}(s) = \arg\max_a \sum_{s'} P(s,a,s') \left[R(s,a) + \gamma V(s')\right] \tag{2.7}$$

Repeat evaluation and improvement until π is stable between two iterations.

2.3.3 Value iteration

Value iteration is a simplification of policy iteration where only one step of policy evaluation is performed in each iteration. Value iteration does not compute an actual policy until the value function has converged (Barto and Sutton 1998). Value iteration works as follows:

Initialization

Start with an arbitrary value function V_0.

Value iteration

Update the value function for each state using the update rule (2.8).

$$V_{k+1}(s) = \max_a \sum_{s'} P(s,a,s') \left[R(s,a) + \gamma V_k(s')\right] \tag{2.8}$$

[1] $\arg\max_a f(a)$ gives the a that maximizes $f(a)$.

Repeat value iteration until V converges and set $V = V_k$. As in policy iteration, by convergence is meant that $|V_{k+1}(s) - V_k(s)| \leq \epsilon, \forall s$, where ϵ is a small value.

Compute the policy using equation (2.9).

$$\pi(s) = \arg\max_a \sum_{s'} P(s, a, s') \left[R(s, a) + \gamma V(s')\right] \tag{2.9}$$

2.4 Dynamic bayesian networks

A bayesian network is a graphical model that represents random variables and their dependencies on each other (see figure 2.2). Each random variable corresponds to a node and a directed edge represents a dependency between the target and the source node (Heckerman 1998). For instance in figure 2.2, Z depends on the setting of X and Y.

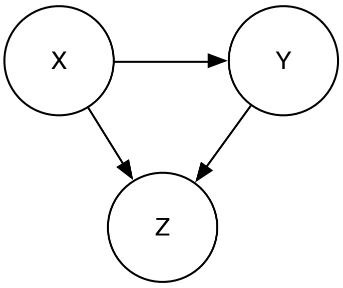


Figure 2.2: *A simple bayesian network*

A dynamic bayesian network (DBN) is a bayesian network where the random variables are allowed to depend on prior settings of the same random variables as well as each other, see figure 2.3.

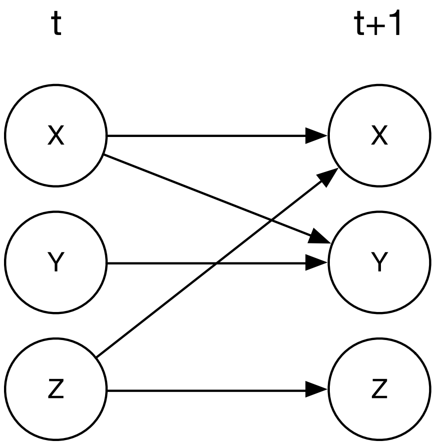


Figure 2.3: *A simple dynamic bayesian network*

In bayesian networks as well as dynamic bayesian networks one can define the parents of a random variable to be the variables that it depends on. In the bayesian network in figure 2.2 the parents of Z are $pa(Z) = \{X, Y\}$ and in figure 2.3 the parents of Y are $pa(Y_{t+1}) = \{Y_t, X_t\}$. A random variable X is not a parent of Y if they are independent, i.e. $\Pr(X, Y) = \Pr(X)\Pr(Y)$.

States in an MDP can be factored into several state variables representing different features of the state. Using this factorization transition probabilities can be represented by a set of bayesian networks, one network for each possible action.

In each network every state variable, $s_t(i)$, is a node representing the state variable at time t, as well as a node, $s_{t+1}(i)$, representing the same state variable at time $t + 1$. If the probability distribution of a certain state variable $s_{t+1}(i)$ is affected by the value of another state variable $s_t(j)$ if action a is taken then there is a directed edge from $s_t(j)$ to $s_{t+1}(i)$ in the DBN corresponding to action a (Guestrin et al. 2003).

Chapter 3

Environment and algorithms

This chapter gives a description of the environment and the algorithms that were used in the experiment described in chapter 4. The environment Invasive Species is a simulation of a river network with invading species, where to goal is to eradicate unwanted species. It is further described in section 3.1.

Two algorithms are covered in this chapter, both of which deal with the problems that arise with large state spaces; however, they differ in the methods they apply. In the following chapter the general ideas behind the algorithms, as well as specific details, are presented. The model-based interval estimation algorithm, described in section 3.2, utilizes clever estimations of confidence intervals for the state-action value functions to improve performance in sparse MDPs. Section 3.3 is on an algorithm that uses dynamic bayesian networks and factored representations to improve the E^3 algorithm to efficiently deal with factored MDPs.

3.1 Environment specification, Invasive Species

When the agents were tested, the Invasive Species environment from the 2014 edition of the Reinforcement Learning Competition was used. The environment is a simulation of an invasive species problem, in this case a river network where the goal of the agent is to eradicate unwanted species while replanting native species.

The environment's model of the river network has parameters, such as the size of the river network and the rate at which plants spread, which can be configured in order to create different variations of the environment. The size of the river network is defined by two parameters: the number of reaches and the number of habitats per reach. A habitat is the smallest unit of land that is considered in the problem. A habitat can either be invaded by the tamarix tree, which is an unwanted species, empty or occupied by native species. A reach is a collection of neighboring habitats. The structure of the river network is defined in terms of which reach is connected to which (Taleghanand, Crowley, and Dietterich 2014). In figure 3.1 a model of a river network is shown.

There are four possible actions (eradicate tamarix trees, plant native trees, eradicate tamarix trees and plant native trees and a wait-and-see action), and the agent chooses one of these actions for each reach and time step. If the agent chooses to eradicate tamarix trees or plant native trees in a reach, all habitats of that reach are targeted by this action. What actions are available to the agent depends on the state of each reach. It is always possible to choose the wait-and-see action, but there has to be one or more tamarix-invaded habitats in a reach for the eradicate or eradicate-and-plant actions to be available and there has to be at least one empty habitat in a reach for the plant-native-trees action to be available (Taleghanand, Crowley, and Dietterich 2014).

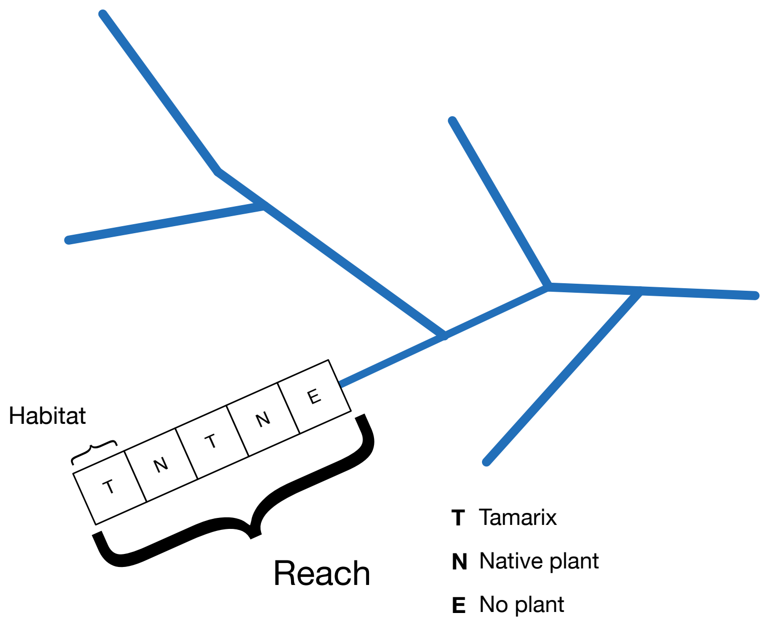


Figure 3.1: *A river network, as modeled by the Invasive Species reinforcement learning environment.*

3.2 Model-based interval estimation

Model-based interval estimation (MBIE) is a version of value iteration whose main feature is its use of confidence intervals for the state-action values. Optimistic bounds to these confidence intervals are computed, by means of finding an optimistic bound for transition probabilities. The optimistic bounds for the transition probabilities are then used in standard value iteration to compute state-action values, from which an optimal policy can be found. State values are also computed at this point.

Since the confidence intervals become less wide when the number of data points that they are based on increases, the more times a state-action pair has been observed, the less optimistic the bounds for these intervals will be (Dietterich, Taleghan, and Crowley 2013). This has the net result of promoting exploration of state-action pairs as long as the confidence intervals are wide, but as the state-action pairs have been tried more and more times, the agent behaves in a less exploratory fashion.

3.2.1 Value iteration with confidence intervals

The upper bounds of the confidence intervals for the state-action values are calculated as in equation (3.1) and then, using these results, equation (3.2) gives the state values by taking the best action for each state.

$$\begin{aligned} Q_{upper}(s,a) =& R(s,a) + \\ & \max_{\tilde{P} \in CI(P(s,a),\delta)} \gamma \sum_{s'} \tilde{P}(s'|s,a) \max_{a'} Q_{upper}(s',a') \end{aligned} \tag{3.1}$$

$$V_{upper}(s) = \max_{a} Q_{upper}(s,a) \tag{3.2}$$

Section 3.2.3 describes a method, Compute$\tilde{P}$, that efficiently calculates the maximization step in equation (3.1) by finding the probability distribution, $\tilde{P}$, that maximizes the sum in the equation.

3.2.2 Optimistic estimations of transition probabilities

The first step of the MBIE algorithm is to find optimistic estimations of transition probabilities. Equation (3.3) describes, as a set, the confidence interval (CI) used by MBIE for the probability distribution over destination states when taking action a in state s. So, the first task of the algorithm is to find the element within this set that is maximally optimistic, meaning that it gives the best value to (s, a) in the following value-iteration step of the MBIE algorithm. A description of how this is done in practice is found in section 3.2.3.

In equation (3.3), $\hat{P}$ is the observed probability distribution (treated as a vector) for destination states from (s, a), $N(s, a)$ is the number of times that the state-action pair (s, a) has been observed, δ is a confidence parameter, ω is the value given by equation (3.4) and $\|x\|_1$ denotes the L^1-norm of the vector x. The L^1-norm is the sum of the absolute value of all elements of a vector.

$$\begin{aligned} &CI\left(\hat{P} \mid N(s,a), \delta\right) \\ &\quad = \left\{\tilde{P} \mid \|\tilde{P} - \hat{P}\|_1 \leq \omega(N(s,a), \delta), \|\hat{P}\|_1 = 1, \hat{P}_i \geq 0\right\} \end{aligned} \tag{3.3}$$

In equation (3.4), ω gives a bound for how much the vector of transition probabilities can be changed from the observed values, while remaining within the confidence interval. In this equation, $|S|$ is the number of states in the MDP and the other variables have the same meaning as in equation (3.3). For the derivation of this equation, see Strehl and Littman (2008).

$$\omega(N(s,a), \delta) = \sqrt{\frac{2|\ln(2^{|S|} - 2) - \ln \delta|}{N(s,a)}} \tag{3.4}$$

3.2.3 Compute$\tilde{P}$

The method for finding the sought element within the set denoted by equation (3.3) is referred to as Compute$\tilde{P}$ in this thesis. The fundamental idea of the Compute$\tilde{P}$ method is that it starts with the observed transition probabilities $\hat{P}$ and then it moves probability mass from "bad" outcomes to "good" outcomes and finally returns the resulting probability distribution, $\tilde{P}$.

Initialization The state transition probability distribution is initialized according to equation (3.5), which corresponds to the observed probabilities.

$$\tilde{P}(s'|s,a) := \hat{P}(s'|s,a) = \frac{N(s,a,s')}{N(s,a)} \tag{3.5}$$

In equation (3.5) $N(s, a, s')$ is the number of times action a has been taken in state s and the agent ended up in state s'.

Moving probability mass The procedure of moving probability mass is done by first finding the outcome state with the best state-value and observed probability less than 1, calling it $\overline{s}$. Analogously the outcome with the worst state-value with an observed probability of greater than 0 is found, and this state is called $\underline{s}$. If a state-value has not been computed yet for a certain state, it is assumed to have the maximum possible value.

The probability values $\tilde{P}(\underline{s}|s,a)$ and $\tilde{P}(\overline{s}|s,a)$ are then increased or decreased according to equations (3.6) and (3.7).

$$\tilde{P}(\underline{s}|s,a) := \tilde{P}(\underline{s}|s,a) - \xi \tag{3.6}$$

$$\tilde{P}(\overline{s}|s,a) := \tilde{P}(\overline{s}|s,a) + \xi \tag{3.7}$$

Since the sum of the probabilities needs to equal one and no single transition probability may fall below zero or exceed one, the probability distribution can only be modified by at most ξ, as given by equation (3.8), where $\Delta\omega = \omega/2$. The variable $\Delta\omega$ denotes the total mass that can be moved, without $\tilde{P}$ having a lower chance than $1-\delta$ of being within the confidence interval for the probability distribution. If ξ is less than $\Delta\omega$, new states $\overline{s}$ and $\underline{s}$ are found, and probabilities are moved until mass equal to $\Delta\omega$ has been moved in total or the probability mass has all been moved to an optimal state. The resulting vector, $\tilde{P}$, is the one that maximizes the sum in equation (3.1).

$$\xi = \min\{1 - \tilde{P}(\overline{s}|s,a), \tilde{P}(\underline{s}|s,a), \Delta\omega\} \tag{3.8}$$

3.2.4 Optimizations based on Good-Turing estimations

One problem with the method described above is that probability mass can be moved to any destination state, without any consideration taken whether this outcome has ever been observed. Dietterich, Taleghan, and Crowley (2013) and the MBIE-algorithm in this thesis make use of an optimization that deals with this by limiting the probability mass that can be moved to outcomes that have never been observed. This leads to better approximations of sparse MDPs. The limit that is used is the approximation of the probability mass in unobserved outcomes as estimated by Good and Turing as $\hat{M}_0(s,a) = |N_1(s,a)|/N(s,a)$ (Good 1953). In this equation, $N_1(s,a)$ is a set of the states that have been observed exactly once as an outcome when taking action a in state s and $N(s,a)$ is the number of times that action a has been taken in state s in total.

3.3 E^3 in factored Markov decision processes

The second algorithm studied in this thesis is a version of the E^3 algorithm that focuses on factored problem domains by modeling them as a dynamic bayesian network. The original E^3 algorithm is described in section 3.3.1, which gives a broad overview along with the key strategies used in the algorithm. The following section, 3.3.2, considers some ways to extend the original algorithm and make use of factored representations and planning in factored domains to improve the running time of the algorithm.

3.3.1 The E^3 algorithm

E^3 (Explicit Explore or Exploit) is an algorithm that divides the state space into two parts — known states and unknown states — in order to decide whether it is better to explore unknown states or to exploit the agent's knowledge of the known states. A state is considered to be known if the E^3 agent has visited it enough times. All other states are either unknown or have never even been visited. Unknown and unvisited states are treated in the same way. In the following sections a description is given of the three phases of operation of E^3 which are called balanced wandering, exploration and exploitation (Kearns and Singh 2002).

Balanced wandering When the agent finds itself in a state that it has not visited a large enough number of times to be considered a known state, it enters a phase called balanced wandering. When in balanced wandering, the agent always takes the action performed from this state the least number of times.

Exploration When the agent from the balanced wandering phase enters a state that is known, it performs a policy computation to find a policy that maximizes the agent's chance of ending up in an unknown state.

This exploration policy calculation is performed on an MDP which contains all known states and their experienced transition probabilities. All unknown states are gathered in a super-state with transition probability 0 to all known states and 1 to itself. The rewards are set to 0 for known states whereas the reward for the super-state is set to the maximum possible reward. A policy based on this MDP definition will strive to perform actions that reach the super-state, i.e., an unknown state.

If the probability of ending up in the super-state is below a certain threshold, it can be proved that the agent knows enough about the MDP that it is probable that it will be able to calculate a policy that is close to optimal (Kearns and Singh 2002).

Exploitation Thus when the agent knows enough about the MDP it performs a policy computation to find a policy that maximizes rewards from the known part of the MDP. This exploitation policy computation is performed on an MDP comprising all known states, their observed transition probabilities and their observed rewards. A super-state representing all unknown states is also added to the MDP with reward 0 and transition probability 0 to all known states and 1 to itself. This MDP definition will result in a policy that favors staying in the known MDP and finding a policy with high return.

Leaving the exploitation and exploration phases When the agent is in either the exploration or exploitation phase, there are two events that can trigger it to exit these phases. First, if the agent enters an unknown state, it goes back to the balanced wandering phase. Second, if it has stayed in the exploration or exploitation phase for T time steps, where T is the horizon for the discounted MDP, it goes back to the behavior described in the "exploration" section above.

3.3.2 Factored additions to E^3

The E^3 algorithm does not exploit that the underlying Markov decision process may be structured in a way that allows certain optimizations. For instance, by factoring the problem as a dynamic bayesian network, the running time can be improved dramatically (Kearns and Koller 1999).

When using a factored representation some changes to the original algorithm are required to make it compatible. One issue that has to be solved is how to perform planning with the new representation. In this thesis a modified version of value iteration was used for planning and it is described later in this section. In section 6.4.2 some other methods are presented.

Dynamic bayesian network structure Assume that the states of an MDP each are divided into several variables. For instance, the Invasive Species MDP described in section 3.1 constitutes such a case, where the status of each reach can be considered a variable on its own. The number of tamarix trees, native trees and empty slots in a certain reach at time step $t + 1$ depends not on the whole state of the environment at time t, but only on the status of adjacent reaches. Those variables on which another variable depend are called its parents.

An MDP that follows the description in the previous paragraph is described as factored. With the assumption of a factored MDP, it is possible to describe its transition probabilities as a dynamic bayesian network, where one would have a small transition probability table for each of the reaches in the MDP, instead of a large table for the transitions for the whole states.

Chapter 4

Method

This chapter covers the preliminaries and preparations carried out before the execution of the experiments. It covers how the agents described in chapter 3 were implemented and the additions and modifications that were made to them. The chapter concludes with a description of the tools used for the experiment.

4.1 Algorithm implementation

We implemented two existing algorithms and we also added some extensions. The experiment utilized RL-Glue (section 4.3) to connect the agents and environment to each other. To verify the behavior of our agents we utilized smaller environments (see sections 4.1.3 and 4.1.4) where the correct operation is either easy to derive or obvious from inspection. By starting with smaller problems and using an iterative approach it was possible to identify bottlenecks in our implementations and correct possible errors early.

4.1.1 Implementation choices and extensions for MBIE

How often to perform planning It is possible to perform planning and compute a new policy once for each action taken by the agent. However, this would be unnecessarily slow to compute. The planning comprises iterating Q-value updates to convergence and then using these converged values to update V-tables, a considerable number of computations. So instead of planning after every action taken, the algorithm only performs planning and updates the policy at some given interval.

For small variants of the Invasive Species environment the policy computation is performed every time the number of observations of a state-action pair has doubled. For large variants we perform planning when the total number of actions taken has increased by 50% since the last time when planning was performed. A large variant is defined as one where the number of state variables exceeds 9. This number was determined by running some preliminary tests and choosing a value giving a reasonable run time.

Optimizing bounds Another optimization that can be performed is tweaking $\Delta\omega$ in equation (3.8) to fit the environment that the agent is used with. Equation (3.8) gives bounds for which it can be proved that the method converges to an optimal policy, given some confidence parameter. In practice, however, this value can be reduced by quite a bit in order to speed up the rate at which the agent considers state-action pairs known.

A simple linearly declining function can be used instead of equation (3.8). In the so called realistic implementation of MBIE we have used $\omega = 1 - \alpha N(s, a)$. The value of the α parameter was decided through experimentation (see section 4.2).

4.1.2 Implementation choices and extensions for DBN-E^3

Planning in dynamic bayesian networks The DBN-E^3 algorithm does not in itself define what algorithm should be used for planning when the MDP is structured as a DBN (Kearns and Koller 1999). It considers planning a black box, leaving the choice of planning algorithm to the implementers.

Value iteration can be done with a factored representation of an MDP in a fairly straightforward manner. The same equations that normal value iteration (section 2.3.3) is based on can be used when the MDP is factored too. The only difference is that in order to calculate the probability of a state transition, $P(s_t, a_t, s_{t+1})$ one has to find the product of all the partial transitions,

$$\prod_i P\left(s_{t+1}(i) \mid pa(s_t(i)), a_t\right) \tag{4.1}$$

where i ranges over all partial state indices, $s(i)$ is the partial state of s with index i and $pa(s(i))$ is the setting of the parents of the partial state $s(i)$ as described in section 2.2.3.

When an MDP has this structure, observations of partial transitions can be pooled together when the state variables are part of similar structures in the MDP. In the version of DBN-E^3 described here, all state variables that have the same number of parent variables have their observations pooled together. This means that state variables corresponding to reaches that have no other reaches upstream all share the same entry in the transition probability table. In the same way, reaches with the same number of directly adjacent reaches upstream share their entries.

One policy per state variable For some MDPs it is possible to compute a separate policy for each state variable individually. This is the case when there is a separate action taken for each state variable, which is true for the Invasive Species environment (section 3.1). In the implementation of E^3 used in this thesis, this policy computation is performed in two steps.

In the first step, a policy is calculated for state variables that have no other state variables than themselves as parents in the DBN, and these states are marked as done. This calculation is done by value iteration where the reward function is described in equation (4.2). Since there is now a decided action for each value for these state variables, the transition probabilities for these variables can be considered as pure Markov chains in the next step. In this step, a policy is found for state variables whose parents are marked as done, until all state variables are done. In this second step, the transition probabilities of the parents are thus treated as independent of the action taken.

The reward function for the partial action $a(i)$ and the partial state $s(i)$ in the Invasive Species environment can be described as follows:

$$\begin{aligned} R(s(i), a(i)) = {} & c(s(i))r_c + t(s(i))r_t \\ & + n(s(i))r_n + e(s(i))r_e \\ & + x(a(i))t(s(i))r_x + p(a(i))e(s(i))r_p \end{aligned} \tag{4.2}$$

where

$c(s(i))$ is 1 if $s(i)$ is infected, 0 otherwise.

$x(a(i))$ is 1 if action was taken to exterminate tamarix trees, 0 otherwise.

$p(a(i))$ is 1 if action was taken to plant native trees, 0 otherwise.

$t(s(i))$, $n(s(i))$, $e(s(i))$ is, in $s(i)$, the number of tamarix-infested habitats, habitats with native trees and empty habitats, respectively.

r_c, r_t, r_n, e_r, r_x and r_p are rewards given for each infected reach, tamarix-invaded habitat, native habitat, empty habitats, extermination of tamarix tree and restoration of native tree in empty slot, respectively.

Since equation (4.2) is a simple linear equation, the unknown variables (r_i, r_t, r_x and r_p) can be calculated exactly once a few data points have been collected. Once this is done, the agent can use equation (4.2) to calculate the reward for any partial state-action pair.

Planning for each state variable individually has the benefit of making the planning algorithm linear in the number of state variables, greatly reducing the time needed to calculate a policy. However, there are several downsides to using this kind of approximation, some of which are discussed in section 6.1.1.

4.1.3 GridWorld

The GridWorld environment was implemented to easily be able to verify the correctness of the MBIE algorithm. It consists of a grid of twelve squares with one blocked square, one starting square, one winning square, one losing square and eight empty squares. The agent can take five actions, north, south, west, east or exit. The exit-action is only possible from the winning or losing state. When taking an action being in one state and the action is directed to another empty state there is an 80% probability to succeed and 10% probability to fail and 10% to go sideways.

4.1.4 Network simulator

A simple computer network simulation was implemented to verify the behavior of an early version of E^3 algorithm. In this environment, the agent tries to keep a network of computers up and running. All computers start in the running state, but there is a chance that they randomly stop working. If a computer is down, it has a chance to cause other computers connected to it to also fail. In each time step, the agent chooses one computer to restart, which with 100 percent probability will be in working condition in the next time step. The agent is rewarded for each computer in the running state after each time step.

4.2 Test specification

The testing of the agents required us to choose certain sets of parameters, for the environment, the two different agents and the experiment itself.

Environment parameters The Invasive Species environment requires a number of parameters. For further explanation of the environment parameters consult the environment webpage[1]. The parameters used in testing the agents were as follows.

Table 4.1: Dynamic parameters common to both species

Parameter	Value
Eradication rate	0.85
Restoration rate	0.65
Downstream spread rate	0.5
Upstream spread rate	0.1

[1] `http://2013.rl-competition.org/domains/invasive-species`

Table 4.2: Dynamic parameters, for the two species of trees

Parameter	Native	Tamarix
Death rate	0.2	0.2
Production rate	200	200
Exogenous arrival	Yes	Yes
Exogenous arrival probability	0.1	0.1
Exogenous arrival number	150	150

Table 4.3: Cost function parameters

Parameter	Value
Cost per invaded reach	10
Cost per tree	0.1
Cost per empty slot	0.01
Eradication cost	0.5
Restoration cost	0.9

Table 4.4: Variable costs depending on number of habitats affected by action

Parameter	Value
Eradication cost	0.4
Restoration cost for empty slot	0.4
Restoration cost for invaded slot	0.8

Agent Parameters The agents evaluated required different types of parameters. Some preliminary tests were run and then the parameters giving best results were chosen. Parameters for MBIE are found in table 4.5 and table 4.6.

Table 4.5: Parameters for proper MBIE

Parameter	Value
Discount factor, γ	0.9
Confidence, δ	95%
$\Delta\omega$	$\frac{1}{2}\sqrt{\frac{2\lvert\ln(2^{\lvert S\rvert}-2)-\ln\delta\rvert}{N(s,a)}}$

Table 4.6: Parameters for realistic MBIE

Parameter	Value
Discount factor, γ	0.9
$\Delta\omega$	$\max\{0, 1-0.05N(s,a)\}$

For DBN-E^3 a higher exploration limit resulted in the agent starting to exploit earlier but with a slightly lower final return. A higher partial state known limit resulted in later exploitation but no appreciable difference in final return. The values in table 4.7 were a good middle ground.

Table 4.7: DBN-E^3 parameters

Parameter	Value
Discount factor, γ	0.9
Exploration limit	5%
Partial state known limit	5

Experiment parameters The tests performed had to be long enough to sample enough data to extract relevant results without making the running time too long. A single test consisted of a specific number of episodes with a specific length. A good combination was required to efficiently evaluate the agents. If a single episode consisted of too many samples it would be difficult to see the learning process as results are reported as total reward over an episode and that process might be hidden as its impact on the total reward is smaller with a longer episode. On the other hand, if an episode length is too short it would end before the agents could do any valuable learning.

In addition to a satisfactory episode length, a reasonable number of episodes needs to be sampled for it to be possible to draw conclusions from the results. If the number of episodes is too small, the convergence of the agents cannot be seen. However, too many episodes would lead to unnecessary data collection, since the algorithms would already have converged and no interesting changes would happen. Some preliminary experiments were run in order to tune the experiment parameters to suitable values. The episode length was set to 100 samples and there were 100 episodes per test.

To evaluate the agents in the Invasive Species environment combinations of reaches and number of habitats per reach that can be seen in table 4.8 were chosen. Combinations were chosen to have a wide range in the total number of states for the agents to deal with and to test how the agents deal with taking actions that have to take into account several state components.

As seen in table 4.8, the state count increases rapidly when habitats are added to the problem. This is obvious from the fact that the state count depends exponentially on the total number of habitats; see equation (4.3), where h is the number of habitats per reach and r is the number of reaches.

$$|S| = 3^{hr} \tag{4.3}$$

Table 4.8: Combinations of reaches and habitats used in testing.

Reaches	Habitats per reach	Total state count
5	1	243
3	2	729
3	3	19 683
10	1	59 049
4	3	531 441
5	3	14 348 907

4.3 RL-Glue

To evaluate the agents the RL-Glue framework was used, which acts as an interface for communication between the agent and the environment. The software uses the RL-Glue protocol, which specifies how a reinforcement learning problem should be divided when constructing experiments and how the different programs should communicate (Tanner and White 2009).

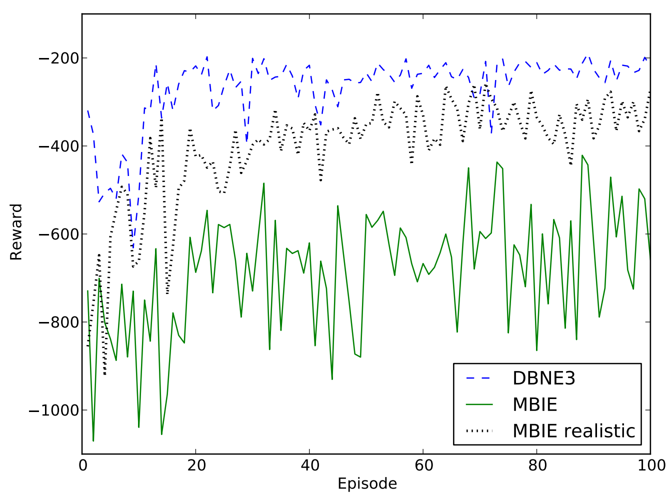
RL-Glue divides the reinforcement learning process into three separate programs: an agent, an environment and an experiment. RL-Glue provides a server software that manages the communication between these programs. The agent and the environment programs are responsible for executing the tasks as specified by RL-Glue and the experiment program acts as a bridge between the agent and environment (Tanner and White 2009).

The modular structure of RL-Glue makes it easier to construct repeatable reinforcement learning experiments. By separating the agent from the environment it is possible to reuse the environment and switch out the agent. It also makes it a lot easier to cooperate and continue working on existing environments implemented by other programmers.

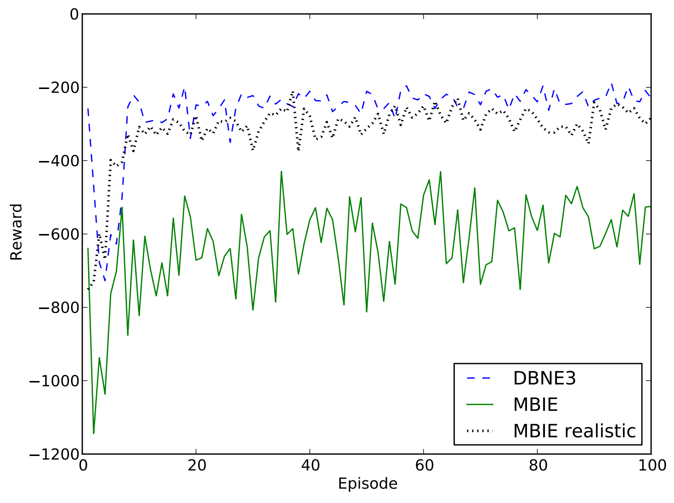
Chapter 5

Results

In figures 5.1, 5.2 and 5.3 we present test results from running our agents on the Invasive Species environment on different sizes of river networks.



(a) *3 reaches and 3 habitats per reach*



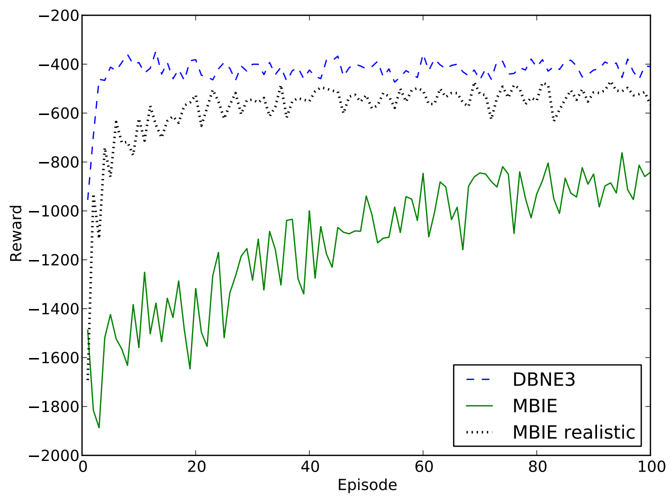
(b) *3 reaches and 2 habitats per reach*

Figure 5.1: *Test runs with different number of reaches and habitats*

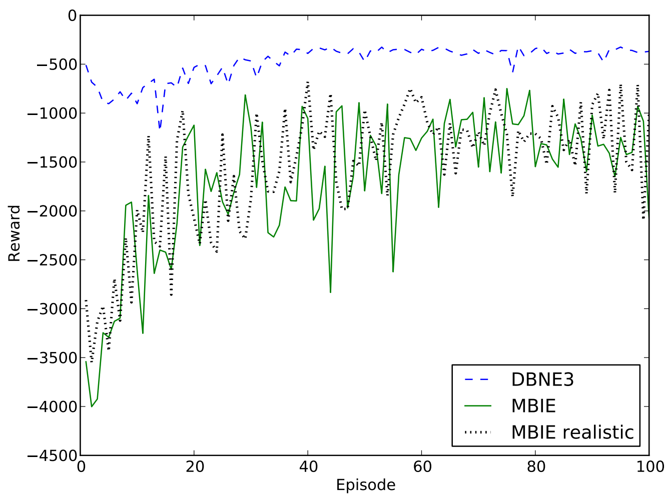
The agent learns for 100 episodes with 100 samples per episode, other parameters are specified in section 4.2, each test was run five times and these are the average results. The Invasive Species environment associates a certain cost with each state and action, thus the reward is always negative.

With smaller state spaces (see figure 5.1a, 5.1b and 5.2a) we can see that the realistic MBIE agent outperforms the original MBIE agent and comes close to the DBN-E^3 agent. In the tests where the state space is larger (see figure 5.2b, 5.3b and 5.3a) original MBIE and realistic MBIE are very close to each other in performance while the DBN-E^3 agent outperforms both.

In each of the test runs, the DBN-E^3 algorithm exhibits a period of learning that corresponds to exploration (as described in section 3.3.1) where the agent has not explored the environment enough and seeks more information. This is apparent in figure 5.1a.

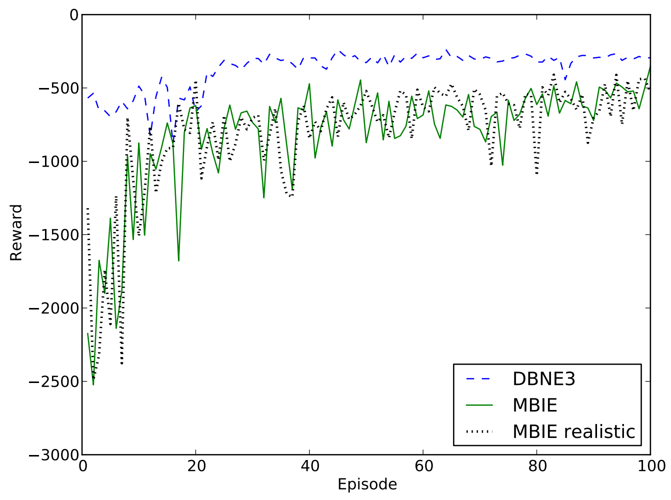


(a) *5 reaches and 1 habitats per reach*

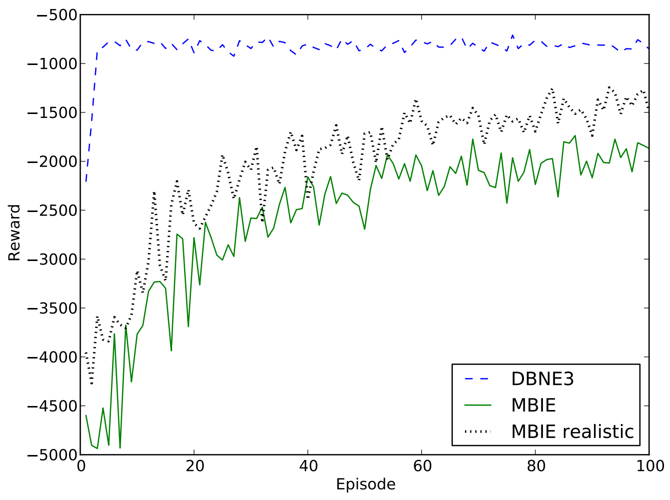


(b) *5 reaches and 3 habitats per reach*

Figure 5.2: *Test runs with different number of reaches and habitats*



(a) *4 reaches and 3 habitats per reach*



(b) *10 reaches and 1 habitats per reach*

Figure 5.3: *Test runs with different number of reaches and habitats*

Chapter 6

Discussion

This chapter is a discussion of the results and the method used in the project. The algorithms are evaluated with regard to their performance achieved in the tests. Their strengths and weaknesses are discussed as well as their suitability for problems with an environment that has a large discrete state space. There is also a discussion on ethical aspects regarding the work in this thesis.

6.1 Evaluation of the agents

As an initial comment, the fact that we have reasonable results proves that the techniques used work at least in a limited sense of the word. The tests completed within a reasonable time frame and did not run into hardware limitations. Furthermore, the results recorded show an increase in reward over time as expected of a working agent during its learning process.

6.1.1 E^3 in factored MDPs

In a comparison of the DBN-E^3 agent's behavior in the different-size environments, the similarity of the shape of the graphs is striking. At first there is a period of lower and lower performance. For the smallest environments, this period is very short, however. Next, there is a period of fairly constant high performance, which lasts until the end of the experiment. This behavior is clearly visible in figures 5.1.

The similar shapes can be explained as a consequence of the different phases of the DBN-E^3 algorithm and the structure of the studied MDP. In the beginning of the experiment, the algorithm will spend almost all of its time in the balanced wandering and exploration phases. The longer the agent has been exploring, and the more states become known, the further the agent has to explore into hard-to-reach parts of the MDP to find unexplored states. Now, in the case of the Invasive Species environment with the parameters chosen as in the experiments presented, the most easily reachable states are the ones where there is no tamarix infection. This means that the harder a state is to arrive at, the more infected reaches it will probably contain, and thus the performance of the DBN-E^3 agent in the exploration phase will fall as the experiment progresses. However, once the agent knows enough about the environment to enter the exploitation phase, the DBN-E^3 agent spends close to no time at all exploring unknown states, and it retains high performance.

Possible issues with the one-policy-per-reach optimization In section 4.1.2 an optimization is described that works very well for the particular environment and environment settings that the agent was tested for. However, this optimization makes several assumptions that may cause problems if the settings are changed. For instance, one assumption is that the state of a reach is only affected directly by its adjacent parents in river network. If the state of a reach was made to depend significantly on other reaches two or more levels up in the river network, the agent would probably not be able to converge on an optimal policy.

Another assumption that could lead to problems with other environment settings is the assumption that the maximal action cost in the environment is impossible or very hard to break. The Invasive Species environment has a maximum cost for actions. However, with the standard settings it is mathematically impossible to break this maximum. Our implementation of DBN-E^3 would achieve very poor performance if this was not the case, since a large penalty is given when the maximum action cost is breached.

DBN structure Finally, in the Invasive Species environment, the structure of the DBN underlying the MDP is known at the start of the experiment, so the agent does not need to infer it from observations. If this was not the case, all the DBN optimizations would be useless unless some kind of algorithm for inferring the DBN structure was added to the agent.

6.1.2 MBIE

In comparison to the DBN-E^3 performance graphs, the MBIE performance exhibits a much smoother transition from poor to good performance. This is due to the fact that MBIE does not have a clear distinction between exploration and exploitation in phases. Instead, MBIE in effect always gives state-action pairs that are relatively unexplored a bonus to their expected value in order to promote exploration.

In the graphs for MBIE there are several “dips” in performance as for example in figure 5.2b. These could be explained as cases when the algorithm by chance enters previously unexplored states and spends several steps exploring this and similar/adjacent states.

Realistic MBIE and original MBIE The realistic version of MBIE outperforms the original MBIE in every test. This is probably explained by the fact that the state that is the easiest to arrive at is the one that gives the greatest reward (see section 6.1.1). Since the realistic version of MBIE considers states known and thus evaluates them realistically rather than optimistically much sooner than the original version of MBIE, it spends much less time exploring unknown states, which are bound to give lower rewards than the more easily explored states.

Impact of planning-frequency A factor to be considered when viewing the results is the impact of the frequency of planning. For example when looking at large state spaces in MBIE (both versions) planning is only performed when the sample size has increased by 50%. This is quite infrequent when viewing our tests of 100 episodes of 100 samples per episode, resulting in many episodes using the same policy and making it harder to deduce improvement over time as they come in discrete intervals rather than continuously. This issue does not arise in DBN-E^3 which does planning continuously when required.

6.1.3 DBN-E^3 vs MBIE

An expectation on both agents is that they should converge to a near optimal behavior as time goes to infinity. However, it is clear from the results presented in chapter 5 that neither version of the MBIE agent reaches the same level of performance as the DBN-E^3 agent in the tests we performed. In the smaller problem as seen for example in figure 5.1b, the difference is not large between the DBN-E^3 agent and the realistic MBIE. Nevertheless, it is still clear that DBN-E^3 is far superior in finding an optimal policy.

Strehl and Littman (2004) tested the performance of the MBIE algorithm along with the E^3 algorithm in two non-factored environments, RiverSwim and SixArms. Strehl and Littman (2004) concludes that the MBIE outperforms E^3 in both of the environments. This thesis utilizes a factored environment for tests and thus the DBN-E^3 algorithm outperforms the non-factored MBIE algorithm. Due to the significant performance increase in relation to the non-factored version of the E^3 algorithm the DBN-E^3 outperforms the MBIE algorithm.

As discussed in section 6.4.4 we did not make any factored additions to the MBIE algorithm and it is uncertain whether the DBN-E^3 algorithm would retain its superior performance if this had been done.

Unfair comparisons The comparison between our implementations of MBIE and DBN-E^3 are not very fair. The DBN-E^3 implementation has been heavily optimized to work with factored MDPs and the Invasive Species environment in particular, whereas the MBIE implementation is much more generalized. The discussion regarding the generality of the agents is further continued in section 6.2.1. To make a more nuanced comparison between MBIE and DBN-E^3 one could use smaller non-factored environments as well as the larger factored Invasive Species environment.

Large state spaces When the number of states is increased it is clear that the algorithms still work. In a comparison between the 4 reaches, 3 habitats-per-reach case seen in figure 5.3a, and the 5 reaches, 3 habitats-per-reach case seen in figure 5.2b, this can be seen. The algorithms still improve over time although taking longer time to converge, which is expected as the state space increases in size.

A notable difference between the original and realistic versions of MBIE is that the realistic version has a similar level of performance as the original MBIE in large state spaces whereas in small state spaces the realistic MBIE is clearly better than the original. Compare figure 5.1b with figure 5.2b to see this difference. In small state spaces the realistic version will know the real value of more states than the original version, while in large state spaces both versions will prioritize unknown states.

The DBN-E^3 algorithm successfully deals with large state spaces. DBN-E^3 utilizes a factored approach in representing transition probabilities, which allows the algorithm to learn about the environment quicker whereas a non-factored approach would struggle to learn the transition probabilities. This can be seen in the result where the DBN-E^3 algorithm quickly knows enough to start exploiting even in cases such as figure 5.2b. Furthermore, using a factored approach to policy calculation allows quick policy calculations, making the algorithm fast to run in large state spaces.

6.2 Potential factors impacting the results

There are several factors that possibly reduce the reliability of the results. For instance, all evaluations used only one environment, the implementation of the algorithms could contain errors and the correct results for all test runs are not always known.

6.2.1 Impact of using one environment

The results presented in this report are only collected from one environment. Considering this, it is challenging to evaluate and verify the generality of the implementations. The DBN-E^3 implementation is specifically optimized for the Invasive Species environment, which reduces the credibility regarding the generality of the techniques discussed in this thesis, due to the fact that it is not possible within the scope of this thesis to evaluate the performance of the DBN-E^3 without the optimizations applied.

The generality of our MBIE implementation is more probable due to the fact it was simultaneously tested alongside Invasive Species in a simpler environment: GridWorld (section 4.1.3), thereby forcing a degree of generality during construction of the agent.

6.2.2 Implementation of algorithms

One of the biggest challenges when constructing and evaluating algorithms is to validate the actual implementation of the algorithm. When results seem erroneous it is hard to immediately realize whether it is the code that contains bugs or the math was incorrectly interpreted. The lack of similar work containing results also makes it hard to estimate of how well our algorithms perform in comparison to similar implementations.

In order to increase the credibility of the implementations of this thesis, one method is to perform unit testing of the implementation. By creating implementations which enable unit testing it becomes easier to test the individual pieces of the algorithms and thereby verify correct behavior. Comparing the process of using automated tests with manually validating the results of the complete algorithms behavior as described in section 4.1, the advantages of the former are obvious.

6.2.3 Testing methodology

Evaluating agents is hard due to the process of choosing appropriate parameters for both the algorithm and the environment. For example it is uncertain how much the parameters for the algorithms affect the outcome of the experiment and trying out all possible combination is not feasible within the scope of this thesis. However, a possible solution for this problem is to derive the optimal parameters using mathematical proofs.

The same problem appears when choosing the parameters for the Invasive Species environment. The obvious question is whether a different choice of parameter values would cause the DBN-E^3 algorithm to perform in the same way or if the optimizations discussed in section 6.1.1 would not work in another setting. Nevertheless, with the MBIE algorithm there is a greater probability that the results will be comparable to the results described here. This is the case, as it has been tailored to the particular experiments that were performed to a lower extent than DBN-E^3 has.

The last potential issue of the evaluation is rather complex. Is the solution that the agents find really an optimal solution for the problem environment? When the number of habitats and reaches increases it becomes impossible to check manually if the policy computed by the algorithm is correct. This may be a common problem when evaluating reinforcement learning algorithms; as Dietterich, Taleghan, and Crowley (2013) mention in their report, they too are uncertain regarding the fact that their implementation achieves an optimal solution.

6.3 Using models for simulating real world problems

This section focuses on the further impact of reinforcement learning using models of the real world and simulations. The environment Invasive Species is a simulation of the problem with invasive species. This domain focuses on the problem where a spreading process needs to be controlled in a river network with native and invading plant species (Taleghanand, Crowley, and Dietterich 2014).

It is commonly known how fragile ecosystems are to changes. Ecosystems are often complex and it is common that changes that help the system in the short term can damage it in the long run. By using simulations with self-learning algorithms it is possible to test more methods than time and money would allow in the real world to find good long-term policies. This presents a practical use of reinforcement learning. However, as simulations are only rough models of the real world the results from the simulation will also at best only be roughly correct.

6.4 Further work

In this section we discuss possible further work and extensions to our thesis.

6.4.1 Testing algorithms with more environments

As mentioned in section 6.2.1 only one environment has been used for testing. Due to the close ties to the Invasive Species environment during development there is a risk that the algorithms as implemented will not perform well in other environments. By using several environments in testing one can study the generality of the algorithms and maybe improve them. One possible environment to extend the study with is the Tetris domain representing the game of Tetris from the 2008 edition of the Reinforcement Learning Competition (Whiteson, Tanner, and White 2010).

6.4.2 Planning algorithm for DBN-E^3 in factored MDPs

DBN-E^3 utilizes a dynamic bayesian network in order to factor the representation of the environment. In section 3.3.2 a slightly modified version of value iteration is used for planning due to its simplicity and the scope of the project. Below are other possible planning algorithms for factored MDPs that are more sophisticated and specifically derived for use in factored MDPs.

Approximate value determination One example of a planning algorithm for factored MDPs is Approximate value determination. This utilizes a value determination algorithm to optimally approximate a value function for factored representations using dynamic bayesian networks. The algorithm uses linear programming in order to achieve as good an approximation as possible, over the factors associated with small subsets of problem features. However, as the authors of the algorithm mention, their algorithm does not take advantage of factored conditional probability tables. This leaves room for further improvements using dynamic programming steps (Koller and Parr 1999).

Approximate value functions This method represents the approximation of the value functions as a linear combination of basis functions. Each basis involves a small subset of the environment variables. A strength is that the algorithm comes in both a linear and dynamic programming version. However, it could be more complex than approximate value determination to implement. Guestrin et al. (2003) presents results for problems with 10^{40} states, which may very well result in a bigger improvement than the previous option discussed.

6.4.3 Improvements to MBIE

Dietterich, Taleghan, and Crowley (2013) suggests several improvements to a confidence interval based algorithm called DDV. In MBIE we implement some of these improvements, such as the Good-Turing optimization (see section 3.2.4), but Dietterich, Taleghan, and Crowley (2013) suggests several more, their DDV algorithm promises to “terminate[. . .] with a policy that is approximately optimal with high probability after only polynomially many calls to the simulator.”

6.4.4 Extend MBIE to utilize a factored representation

In the discussion of this thesis, the issue of unfairness when comparing the two agents has been highlighted. Due to lack of foresight we chose to not implement a factored version of MBIE. We could not simply add the factored representation to the MBIE algorithm due to diverging source trees when we realized our mistake. Therefore as an extension to this thesis the implemented MBIE algorithm could be extended to make use of a dynamic bayesian network as the underlying model. This would have paved way for more interesting test results.

References

Altman, Eitan (2002). “Applications of Markov decision processes in communication networks”. In: *Handbook of Markov decision processes*. Springer, pp. 489–536.

Barto, Andrew and Richard Sutton (1998). *Reinforcement learning: An introduction*. MIT press.

Bellman, Richard (1957). “A Markovian decision process”. In: *Journal of Mathematics and Mechanics* 6.5, pp. 679–684.

Boutilier, Craig, Thomas Dean, and Steve Hanks (1999). “Decision-theoretic planning: Structural assumptions and computational leverage”. In: *Journal Of Artificial Intelligence Research* 11, pp. 1–94.

Dietterich, Thomas, Majid Alkaee Taleghan, and Mark Crowley (2013). “PAC Optimal Planning for Invasive Species Management: Improved Exploration for Reinforcement Learning from Simulator-Defined MDPs”. In: *Twenty-Seventh AAAI Conference on Artificial Intelligence*.

Good, Irving (1953). “The population frequencies of species and the estimation of population parameters”. In: *Biometrika* 40.3-4, pp. 237–264.

Guestrin, Carlos et al. (2003). “Efficient solution algorithms for factored MDPs”. In: *J. Artif. Intell. Res.(JAIR)* 19, pp. 399–468.

Heckerman, David (1998). “A Tutorial on Learning with Bayesian Networks”. In: *Learning in Graphical Models*. Vol. 89. Springer Netherlands, pp. 301–354.

Kearns, Michael and Daphne Koller (1999). “Efficient reinforcement learning in factored MDPs”. In: *IJCAI*. Vol. 16, pp. 740–747.

Kearns, Michael and Satinder Singh (2002). “Near-optimal reinforcement learning in polynomial time”. In: *Machine Learning* 49.2-3, pp. 209–232.

Koller, Daphne and Ronald Parr (1999). “Computing factored value functions for policies in structured MDPs”. In: *IJCAI*. Vol. 99, pp. 1332–1339.

McCarthy, John (2007). *What is Artificial Intelligence?* URL: `http://www-formal.stanford.edu/jmc/whatisai/` (visited on 02/10/2014).

Ng, Andrew Y et al. (2006). “Autonomous inverted helicopter flight via reinforcement learning”. In: *Experimental Robotics IX*. Springer, pp. 363–372.

Strehl, Alexander and Michael Littman (2004). “An empirical evaluation of interval estimation for markov decision processes”. In: *Tools with Artificial Intelligence, 2004. ICTAI 2004. 16th IEEE International Conference on*. IEEE, pp. 128–135.

Strehl, Alexander and Michael Littman (2008). “An analysis of model-based interval estimation for Markov decision processes”. In: *Journal of Computer and System Sciences* 74.8, pp. 1309–1331.

Szepesvári, Csaba (2010). “Algorithms for reinforcement learning”. In: *Synthesis Lectures on Artificial Intelligence and Machine Learning* 4.1, pp. 1–103.

Taleghanand, Majid, Mark Crowley, and Thomas Dietterich (2014). *Invasive Species*. URL: `https://sites.google.com/site/rlcompetition2014/domains/invasive-species` (visited on 03/31/2014).

Tanner, Brian and Adam White (2009). “RL-Glue: Language-independent software for reinforcement-learning experiments”. In: *The Journal of Machine Learning Research* 10, pp. 2133–2136.

Whiteson, Shimon, Brian Tanner, and Adam White (2010). “The Reinforcement Learning Competitions”. In: *AI-Magazine* 31.2, pp. 81–94.