Markov Decision Networks

BY

H.H.L.M. Donkers

A thesis submitted to the examination board in partial fulfilment of the requirements for the Degree

Doctorandus Kennistechnologie (M.Sc. Knowledge Engineering)

Supervisor: Prof.dr. H.J. van den Herik
Coach: Dr.ir. J.W.H.M. Uiterwijk

Faculty of General Sciences
Universiteit Maastricht
Version July 2, 1997
Preface

This report on Markov decision networks is the product of the second part of my student internship for Knowledge Engineering at the Department of Computer Science at the Universiteit Maastricht. The product of the first part of this student internship was a report in Dutch on decision making under uncertainty (Beslissen bij onzekerheid), in which I presented an introductory overview for readers who are not acquainted with concepts like decision trees, Bayes belief networks and influence diagrams. I recommend strongly to read the first report before continuing with this second one.

The style and character of this report is much more technical than its predecessor because the main goal of this report is to give a formal description of decision trees, Markov decision trees and Markov decision networks. Decision trees are well known and broadly used, but a satisfactory formal description of these decision trees has not been given yet. Markov decision trees and Markov decision networks are new concepts that emanate from ordinary decision trees as alternatives for the well-known concept of influence diagrams. This report contains the first formal descriptions of those new concepts. An informal description of Markov decision networks has already been published in [5].

Next to the formal descriptions, this report gives a number of algorithms and computation methods that deal with (Markov) decision trees and Markov decision networks. These algorithms and computations methods are not optimal. Further research is needed to improve the algorithms and to develop more computation methods.

No discussion is given yet on the practical usability and feasibility of Markov decision trees and Markov decision networks, because this will be one of the main subjects of the Ph.D. research project that will start immediately after this student internship.

The formal descriptions and algorithms that are presented here use techniques from several divergent fields in mathematics and computer theory. These fields include Set theory, Formal languages, Matrix calculus, Linear programming, Probability theory and Markov decision processes. It is the combination of techniques that also characterizes the special Knowledge Engineering program at the Universiteit Maastricht in which mathematics and artificial intelligence come together naturally. Readers that feel uncomfortable with mathematical proofs, large expressions and algorithms in pseudocode can safely skip these parts of the text. To facilitate the conceptualization of the complex theoretical constructs, one and the same example is used as a fixed point throughout the whole report. This example is the well known “Tweetie” of decision making: the oil wildcatter’s case.

I finally wish to express my gratitude to Jos Uiterwijk for his coaching during the internship and especially to my supervisor Jaap van den Herik who kindly created the exceptional opportunity to resume and finish my studies at the Universiteit Maastricht.

Jeroen Donkers, Maastricht, June 1997
# Table of Contents

1 Introduction .................................................................................................................. 6  
1.1 What are decision trees? ............................................................................................. 6  
1.2 An example: The Oil Wildcatter .................................................................................. 7  
1.3 Solving a decision tree ................................................................................................ 9  
1.4 Limitations of decision trees ...................................................................................... 9  
1.5 Contents of this report ................................................................................................. 10  

2 Decision Trees .............................................................................................................. 12  
2.1 Findings on decision trees .......................................................................................... 12  
2.2 A framework for decision trees .................................................................................. 13  
2.3 Episodes ..................................................................................................................... 14  
2.4 Decision trees as sets of episodes .............................................................................. 15  
2.5 Histories .................................................................................................................... 18  
2.6 Strategies .................................................................................................................. 19  
2.7 Probabilities .............................................................................................................. 19  
2.8 Pay-off function ....................................................................................................... 23  
2.9 The complete framework ............................................................................................ 25  
2.10 Determining an optimal strategy .............................................................................. 26  
2.11 Conclusions ............................................................................................................. 26  

3 Markov Decision Trees ................................................................................................. 27  
3.1 A framework for Markov decision trees .................................................................... 28  
3.2 The Markov property of probabilities ....................................................................... 28  
3.2.1 State of the world ................................................................................................. 30  
3.2.2 The probability functions .................................................................................... 31  
3.2.3 The probability of states, given episodes ............................................................. 32  
3.3 The additive property of the pay-off function ........................................................... 34  
3.4 Legal Markov decision trees and strategies ............................................................... 35  
3.5 An alternative matrix notation ................................................................................... 35  
3.6 Constructing an optimal strategy ................................................................................ 38  

4 Markov Decision Networks ............................................................................................ 40  
4.1 Dividing the states of the world .................................................................................. 40  
4.2 A framework for Markov decision networks ............................................................. 42  
4.2.1 The sets ................................................................................................................ 42  
4.2.2 The structure ....................................................................................................... 43  
4.2.3 The probabilities .................................................................................................. 44  
4.2.4 Costs and utilities ................................................................................................. 44  
4.3 Episodes and strategies .............................................................................................. 45  
4.3.1 A grammar for episodes ...................................................................................... 45  
4.3.2 Evaluation of episodes ........................................................................................ 46  
4.3.3 A grammar for strategies .................................................................................... 47  
4.3.4 Deriving episodes from strategies ....................................................................... 48  
4.3.5 Evaluating strategies ........................................................................................... 49  
4.4 Computing an optimal strategy ................................................................................. 52  
4.4.1 Special case: a restricted set of episodes ............................................................. 52  
4.4.2 Another special case: Markov decision processes .............................................. 53  
4.4.3 An approximation method: evolutionary programming ..................................... 53  
4.5 A critical revisit to the oil wildcatter ......................................................................... 53  
4.5.1 Going to nine prior states .................................................................................... 54
Chapter 1

Introduction

In everyday life, we are often confronted with situations in which we have to choose among a number of possible ways to act, while we are not at all certain of the effects that our acts will have. In an attempt to predict the effects through observations, we rely on knowledge and beliefs on how and when observed facts usually agree with effects of our acting. Often some kind of model incorporating this knowledge and these beliefs is used. Such a model tells how an observation could occur and how an action would work out. The actual decision for a possible way to act, depends on how we value the effects that we expect from it.

This report deals with those cases in which we can use a probabilistic model for predicting the effects of our actions and in which we can define a measurable valuation function (e.g. costs or profits) so that we can compare the alternative ways to act and finally choose the way of acting that produces the highest (or lowest) value for this function. A sound and well-known tool for this kind of decision problems is the decision tree.

1.1 What are decision trees?

Decision trees were first described in 1944 by the Hungarian scientist John von Neumann [21]. Informally, a decision tree is a tree-structured graph that pictures the subsequent observations and decisions that a decision maker possibly encounters during the course of time. A decision tree is usually drawn sideward, with the root of the tree at the left hand and the leaves at the right hand. Time is flowing from left to right, starting with the root. There are (in the normal cases that we discuss here) three types of nodes in a decision tree: decision nodes, chance nodes and terminal nodes.

Decision nodes represent moments at which the decision maker has to choose between a number of alternative actions, denoted by the branches leaving the decision node to the right. A decision maker must choose exactly one action; mixed strategies, as are known from Game Theory, are not common in this field of decision making.

Chance nodes denote points in time at which uncertain events take place, or, more precisely, the point at which these events become observable for the decision maker or at which these events have their effects. The set of events that may occur at a specific node are represented by the branches that leave that chance node. Continuous and infinite event spaces are allowed in decision trees, but we will restrict ourselves to discrete and infinite event and decision spaces.

1 For a detailed description of decision trees see, for instance, [26] and [30]
To each branch that leaves a chance node, a number between 0 and 1 is assigned. This number is the probability that this event will occur, on the condition that all events on the path from the root of the tree to the chance node have occurred and that all actions on this path were taken. The probabilities on the branches of one and the same chance node must sum up to 1.

The last type of nodes that appear in each decision tree are the terminal nodes or leaves of the tree. Each terminal node represents a possible state of the world, after a completed episode of actions and events. The decision maker attaches a certain number to each of the terminal nodes, denoting the net pay-off that will be achieved when this state of the world is reached. If the pay-offs are negative, they are called costs, but more generally the numbers at the terminal nodes are called utilities.

1.2 An example: The Oil Wildcatter

Before we continue with the description of how decision trees are solved, we will first introduce an example that will clarify the definitions and terms above. The example that we use here is a very common toy problem in the field of decision making and can as such be compared with the well-known “Tweetie” example of nonstandard logic. The oil wildcatter’s example was first used by Raiña in 1976. We use here the slightly adapted formulation from Shenoy [30].

“An oil wildcatter must decide either to drill or not to drill. He is uncertain whether the hole is dry, wet or soaking. His monetary pay-offs and subjective probabilities are:

<table>
<thead>
<tr>
<th>State</th>
<th>Act</th>
<th>Probability of state</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dry</td>
<td>Drill</td>
<td>70;000</td>
</tr>
<tr>
<td>Wet</td>
<td>50;000</td>
<td>0</td>
</tr>
<tr>
<td>Soaking</td>
<td>200;000</td>
<td>0</td>
</tr>
</tbody>
</table>

“The cost of drilling is $70,000. The net return associated with drilling a wet hole is $50,000 that is interpreted as a return of $120,000 less the $70,000 cost of drilling. Similarly the $200,000 associated with drilling a soaking hole is a net return (a return of $270,000 less the $70,000 cost of drilling).

“At a cost of $10,000, the wildcatter could take seismic soundings that will help determine the geological structure at the site. The soundings will disclose whether the terrain below has no structure — that’s bad, or open structure — that’s so-so, or closed structure — that’s really hopeful. The experts have provided us with the following table that shows the probabilities of seismic test results conditional on the amount of oil.
Table 1.2: The conditional probabilities.

<table>
<thead>
<tr>
<th>Amount of oil</th>
<th>Seismic Test Results</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>No structure</td>
</tr>
<tr>
<td>Dry</td>
<td>0.6</td>
</tr>
<tr>
<td>Wet</td>
<td>0.3</td>
</tr>
<tr>
<td>Soaking</td>
<td>0.1</td>
</tr>
</tbody>
</table>

The complete decision tree for the oil wildcatter can only be drawn if these conditional probabilities are preprocessed, because we need the conditional probabilities on the amount of oil, given the seismic test results and the probabilities on the test results. These probabilities can easily be computed using Bayes' rule.

Table 1.3: The conditional probabilities after preprocessing.

<table>
<thead>
<tr>
<th>Seismic Test Results</th>
<th>Amount of oil</th>
<th>Probability on Test Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>No structure</td>
<td>Dry 0.732</td>
<td>0.410</td>
</tr>
<tr>
<td></td>
<td>Wet 0.428</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Soaking 0.208</td>
<td></td>
</tr>
<tr>
<td>Open structure</td>
<td>Dry 0.219</td>
<td>0.350</td>
</tr>
<tr>
<td></td>
<td>Wet 0.343</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Soaking 0.375</td>
<td></td>
</tr>
<tr>
<td>Closed structure</td>
<td>Dry 0.049</td>
<td>0.240</td>
</tr>
<tr>
<td></td>
<td>Wet 0.228</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Soaking 0.417</td>
<td></td>
</tr>
</tbody>
</table>

Now we can produce the decision tree for the oil wildcatter:

![Decision Tree](image)

Figure 1. The decision tree for the oil wildcatter.
1.3 Solving a decision tree

Solving a decision tree is very straightforward. The objective is to select one branch at each decision node in the tree, such that the total expected utility is maximal. A selection of one branch at each decision node is called a strategy, and so the task can be reformulated as to compute the optimal strategy or that strategy that maximizes the total expected utility. The total expected utility is defined recursively:

1. the expected utility at a terminal node is simply the utility attached to it;
2. the expected utility at a decision node is equal to the expected utility at the branch that is selected in the strategy;
3. the expected utility at a chance node is the sum of the expected utilities at all branches of the chance node, weighted with the probabilities attached to these branches;
4. the total expected utility is the expected utility at the root of the decision tree.

This optimal strategy is easily computed using the dynamic programming principle [34]: work backwards in time – from the leaves to the root – and select in the strategy at each decision node its branch with the highest expected utility. This process is called folding back or rolling back the tree. When we fold back the decision tree for the oil wildcatter, we obtain the optimal strategy which tells to perform the seismic test and to drill if the outcome is an open or closed structure. The expected utility or net pay-off for this strategy is $22,500.

1.4 Limitations of decision trees

This basic concept of decision trees can be extended in several ways. Modern decision-tree software supports the user with a set of extra node facilities to ease the definition of the tree. Users can define variables and use formulae for computing probabilities and utilities; subtrees can be cloned and used at several spots in a tree; chance nodes can be defined on base of a parametrized continuous probability distribution to support Monte Carlo simulations.

In spite of these extensions, decision trees suffer from intractability when applied to large or complex decision problems: the tree becomes too large to draw and the preprocessing of probabilities becomes too complex.

For at least ten years now, alternative techniques are being developed, such as the well-known influence diagrams by Howard and Matheson (see [9], [23] and [29]). The influence diagram has proven to be a useful tool and is being applied with success, but certain classes of decision problems are not easily modelled with this technique.

We will introduce another alternative for decision trees that might turn out to be useful for that class of decision problems where possible episodes of actions and events are not specified explicitly as in decision
trees and influence diagrams. We call this alternative Markov decision networks. Of course, we will lay requirements on decision problems, as do decision trees and influence diagrams. Practical research has to point out whether practical situations exist in which our demands hold.

1.5 Contents of this report

After an informal introduction of decision trees, we will present a formal description of decision trees and the folding back algorithm in chapter 2. Our formal framework differs from common descriptions of decision trees because it is based on sets instead of the usual graphical representation.

In chapter 3 the formal framework of decision trees is adapted by adding important constraints to the probability model. This results in the new concept of Markov decision trees. In this chapter we will also introduce a matrix-notation that will help to simplify the formulae in the chapters to follow. Chapter 4 is dedicated to the formal description of Markov decision networks. We also give a .rst algorithm to compute an optimal strategy for special instances of Markov decision networks.

Chapter 5 treats speciﬁc Markov decision networks as special cases of partial observed Markov decision processes (POMDPs). We introduce a rough algorithm to solve this type of Markov decision networks.
Chapter 6 contains an explorative study of infinite-length strategies and episodes in Markov decision networks. By working out three examples some clues are uncovered that may lead to a general computation method for infinite strategies. Chapters 5 and 6 are not essential for understanding the concept of Markov decision networks.
Chapter 2

Decision Trees

Although the informal description of decision trees in the previous chapter is sufficient for practical working with decision trees, we feel that a more formal description is needed to specify exactly what kind of decision problem is modelled by such a tree. Furthermore, we want to use this formal description to expand it in a natural way to a formal description of Markov decision networks.

We will therefore introduce a framework in which general decision trees can be expressed in a formal way. This framework differs from commonly used descriptions mainly because it is based on sets instead of the usual trees. But there are more relevant differences. These differences are based on two findings on decision trees that we will discuss rst.

2.1 Findings on decision trees

For a start, we observe that the chance nodes in a decision tree are not uniform. The nodes can be divided into two types: chance nodes of which the events are observed by the decision maker, and unobserved chance nodes. The last type of chance nodes can only occur at the right hand of the decision tree because they must have a terminal node attached directly to each branch\(^2\). In Figure 3 the unobserved chance nodes and their terminal nodes in the wildcatter’s decision tree are grayed.

\[\text{Figure 3. The oil wildcatter’s decision tree with unobserved chance nodes and their terminal nodes grayed.}\]

\(^2\) Strictly spoken, an unobserved chance node can have another unobserved chance node at its right, but it is always possible to transform such decision tree in an equivalent decision tree in which these extra chance nodes are replaced by a set of terminal nodes.
Observed chance nodes are always preceded by a decision node. Whether or not the decision maker has the intention to observe the outcome when he decides to take an action that leads to an observation, making such decision can be looked upon as performing an experiment that will unveil some (uncertain) information on the state of the world. Decisions that do not lead directly to an observed chance node are obviously not intended to gain information, but to alter the state of the world.

We can therefore divide decisions into decisions to perform an experiment and decisions to perform actions. The events that are observed directly after an experiment will be called the outcomes of that experiment.

The second finding is that very often final states of the world appear indistinguishably at several places in the same decision tree. These indistinguishable terminal nodes have led to the concept of coalescence in decision trees as described in [23] and were a strong motive for the development of influence diagrams as well as of Markov decision networks. We will introduce a set of mutually exclusive final states that can appear once or more at the terminal nodes of the decision tree.

Actions and experiments can also occur simultaneously at several spots in a decision tree. Therefore we will introduce a set of actions and a set of experiments whose elements will be used to construct a decision tree.

### 2.2 A framework for decision trees

Now, we can begin to formulate the framework for decision trees and start with the declaration of the sets that contain the building blocks of decision trees. A preliminary framework for decision trees is given by a tuple of sets: \( \mathbb{A}; \mathbb{E}; \mathbb{R}; \mathbb{W}; \mathbb{H} \) in which:
A is the set \( f a_1; a_2; \ldots; g \) of available actions;

\( E \) is the set \( f e_1; e_2; \ldots; g \) of available experiments;

\( R \) is the union of sets \( R_e \).

Each set \( R_e \) contains the outcomes (or results) \( f r_{e;1}; r_{e;2}; \ldots; g \) of experiment \( e \).

\( W \) is the set \( f w_1; w_2; \ldots; g \) of final states of the world;

\( H \) is the set \( f h_1; h_2; \ldots; g \) of available episodes connected to histories of events (the episode will be explained in the next section).

Example

[In our example of the oil wildcatter, the first four sets \( A, E, R \) and \( W \) would look like this:

\( A = f a_d; a_n; g \), where \( a_d \) means: drill and \( a_n \) means: do not drill.

\( E = f e_t; g \), where \( e_t \) means: seismic test.

\( R = R_e = f r_{e;1}; r_{e;2}; r_{e;3}; g \), where \( r_{e;1} \) means no structure, \( r_{e;2} \) means open structure, and \( r_{e;3} \) means closed structure.

\( W = f w_n; w_d; w_w; w_s; g \), where \( w_n \) means not drilled, \( w_d \) means dry, \( w_w \) means wet, and \( w_s \) means soaking.]

2.3 Episodes

Normally, a decision tree is represented by a graphical tree, and the possible courses of time are represented by the paths in the tree that run from the root of the tree to the leaves ([21], [23], [26]). We introduce an alternative formulation of the decision tree and its possible courses of time: the episode. Our formulation is maybe not as clear as the graphical representation in respect to practical use, but our formulation is only intended for theoretical use.

Definition 2.1 An episode \( h \) is an ordered string of elements from \( A \cup R \), that is, a string of actions and outcomes of experiments.

Experiments do not appear explicitly in an episode, but only implicitly inside sufixes of outcomes (such as \( e_t \) in \( r_{e;3} \)). We will call actions and outcomes together events. The order of the elements in the episode denotes the flow of time from left to right. An episode is delimited by brackets \( h \) and \( i \); the elements are separated by commas. A special episode is the empty episode \( h_i \) that contains no events.

Example

[Episode \( h = f r_{e;3}; a_d; i \) indicates that the seismic test \( e_t \) was executed first and outcome \( r_{e;3} \) (closed structure) was observed. After the observation, action \( a_d \) (drill) was taken.]
Definition 2.2  The number of events in an episode $h$ is called the length of the episode and is denoted by $j_h$.

Because an action can happen more than once during an episode and also an outcome can occur more than once, the position or moment of an event in the episode is needed to identify an event uniquely. We will use the notation $h[t]$ to denote the event at time $t$ in episode $h$, where $t$ is a number from 1 to $j_h$.

Example
[If $h = hr_{e_i,c}; ad_i$ then $h[2]$ will be $ad_i$. ]

Sometimes we want to refer to the experiment to which an outcome at an event belongs if the event is an outcome at all. We will use the notation AE (for Action-or-Experiment) for this.

Example
[AE($re_{i,c}$) = $ei$ and AE($ad_i$) = $ad_i$. If $h = hr_{e_i,c}; ad_i$ then $AE(h[1]) = ei$. ]

The notation $h + x$ will be used to expand an episode.

Example
[If $h = hr_{e_i,c}; ad_i$, then $h + an_d$ will become: $hr_{e_i,c}; ad_i an_d$. ]

2.4 Decision trees as sets of episodes

We introduced the concept of episodes, because using this representation has the advantage that proofs on decision trees can be formulated in terms of sets instead of graphical elements like nodes and branches. In this section we will proof that the main structure of a decision tree is equivalent to a set of episodes. This means that we can use sets of episodes instead of graphs in our further proofs.

Before we can construct this proof, we have to temporarily strip the terminal nodes from decision trees, because the terminal nodes of a decision tree are not represented in episodes:

Definition 2.3  An observed decision tree is a decision tree with all terminal nodes and unobserved chance nodes removed.

Figure 5 shows the observed decision tree that is derived from the oil wildcatter's decision tree. Observe that there is only one chance node left in the tree. Our proof will be performed in three stages. First we show that every observed decision tree represents a set of episodes. Then we show that every set of episodes can be transformed into a decision tree. Finally we show that these relations are reversible.

Lemma 2.1  Every observed decision tree represents a set of episodes.
We will prove this by simple construction: take a full path in the observed decision tree and create an episode by starting with the empty episode and expanding it with the actions and outcomes that are encountered subsequently when travelling along the path from root to leaf. This way, each path will result in an episode and all episodes produced will be different.

Furthermore, two observed decision trees that produce the same set of episodes must be equal except possibly for the ordering of their branches, for if a path in one tree does not occur in the other tree, then the produced sets of episodes must be different.

Example
The complete set of episodes in the oil wildcatter’s observed decision tree is:

\[
H = \{ \text{fail; fail; fail; } \}
\]

Lemma 2.2 Every set of episodes represents an observed decision tree.

This proof is more elaborate, but we will use construction again. Let \( H^0 \) be a (finite) set of episodes.

1. Create a partition \(^3\) \([H^0_{a_1}; H^0_{a_2}; \ldots; H^0_{a_n}]\) from \( H^0 \) such that all episodes in every subset have the same action or an outcome of the same experiment at the first position. So, put all episodes that start with \( a_i \) in subset \( H^0_{a_i} \) and all episodes that start with \( r_{e_j;k} \) (for any \( k \)) in subset \( H^0_{r_{e_j;k}} \).
2. Start constructing the decision tree with a decision node and one branch for every subset in the partition. Mark the branches with the name of the action or the experiment.
3. For each subset \( H^0_{a_i} \) create a sub-partition \([H^0_{a_i,r_{e_j}}; H^0_{a_i,r_{e_j}}; \ldots; H^0_{a_i,r_{e_j}}]\) such that all episodes in every sub-subset.

\(^3\) A partition of a set \( A \) is a division into subsets \( A_1; A_2; \ldots; \) such that \( A = \sqcup_i A_i \) and \( A_j \cap A_k = \emptyset \) if \( i \neq j \) (see [16]). We will use notation \([A_1; A_2; \ldots;]\) for partitions.
have the same outcome at the first position.

(4) Attach a chance node at the branch for \( H_{0e}^j \), add one branch for every subset \( H_{0e;ik}^j \) and mark the branch with the name of the outcome.

(5) Now strip the first element from all episodes and remove the episodes that become empty. Repeat step 1 to 4 for every subset \( H_{0ai}^j \) and \( H_{0e;ik}^j \) (taking \( H^1 = H_{0a}^i \), etc.) and recur until there are no episodes left. Attach the decision node from step 2 to the branch that was added to the subset concerned in the previous cycle.

This construction procedure will clearly produce a valid observed decision tree for every arbitrary set of episodes. Moreover, every set will result in a different tree, because step 1 or step 3 will always diverge somewhere for two different sets of episodes.

Example

The set

\[
H = fh_{nd};i;hr_{e;i};a_{nd};i;hr_{e;oi};a_{nd};i;hr_{e;ci};a_{di}g
\]

would result in the observed decision tree shown in Figure 6 when this construction algorithm is used. The first partition of step 1 would be:

\[
H^0 = [H_{0a}^j]H_{0e}^j = [fh_{nd}gifhr_{e;i};a_{nd};i;hr_{e;oi};a_{nd};i;hr_{e;ci};a_{di}g]
\]

and the partition in step 3:

\[
H^0_{0e} = [H_{0e;ni}]H_{0e;oi}H_{0e;ci} = [fh_{e;ni};a_{nd}gifhr_{e;oi};a_{nd}gifhr_{e;ci};a_{di}g]
\]

In the second recursion cycle, the set \( H^1 \) for outcome \( r_{e;in} \) would be just \( fh_{nd}ig \), so just one (trivial) subset \( H_{0nd}^1 = fh_{nd}ig \) is constructed in step 1. In spite of the fact that there is only one action, the algorithm will insert a decision node, having only one branch. The same holds for the other two outcomes. After the second cycle the recursion stops.

![Figure 6. The decision tree that results from the set of episodes H.](image)

To be able to state that there is a one-to-one relation between sets of episodes and observed decision trees, it suffices almost to combine the two facts above: every set of episodes represents a different observed decision tree and every different observed decision tree (save the ordering of the branches) produces a
different set of episodes. The only extra demand we need to ensure a one-to-one relation is that the decision tree produced from a set of episodes produces again the original set of episodes.

This last fact is easily shown: every episode that is used to construct an observed decision tree ends up as a path in this tree. When the tree is converted back into a set of episodes, then this particular path must produce exactly the same episode again.

Although there is a one-to-one relation between observed decision trees and sets of episodes, still many different (complete) decision trees may share the same observed decision tree. We solve this inconvenience by stating that all possible final states of the world are reachable after each episode albeit in many cases with probability zero. The structures of all decision trees that share the same observed decision tree now become the same. Without any loss of generality this adaptation allows us to conclude:

Theorem 2.3: There is a one-to-one relation between decision trees and sets of episodes: every decision tree induces a set of episodes and every set of episodes induces a decision tree.

We will treat sets of evidences and decisions trees as the same from now on. The reader should however be aware that thus far we only described the qualitative structure of decision trees: the theorem only holds for this part of the whole decision tree concept. Summarizing: in the set-tuple \( \{A; E; R; W; Hi\} \), the set of episodes \( H \) uniquely defines the structure of the decision tree.

2.5 Histories

In the previous section, we spoke about left hand parts of episode-strings and a path in the tree to a certain node. This concept is important for future definitions and needs to be formalized:

Definition 2.4: The history \( h(t) \) upon time \( t \) is defined as the episode that is formed by the substring from \( h \) in front of position \( t \). Per definition \( h(t) = h \) for \( t > |h| \).

Example

[If \( h = \text{hret;ci;ad} \) then \( h(3) = \text{hret;ci;ad}, h(1) = hi, \) the empty episode, and \( h(4) = h \).]

A history is equivalent to a path from the root to a decision node or terminal node in a decision tree.

Definition 2.5: Let \( h \) be an episode, not necessarily a member of \( H \), then the set \( H(h) \mu H \) is defined as the set of all episodes that share history \( h \):

\[
H(h) = \{ h \mid h(t) = h; t = |h| + 1 \}
\]

Example

[In the oil wildcatters decision tree, the episodes that share history \( \text{hret;ci} \) are:

\[
H(\text{hret;ci}) = \{ \text{hret;ci;ad}; \text{hret;ci;ad;ig} \}
\]

A set of episodes that share a history is equivalent to a subtree at a decision node or a terminal node in the decision tree. At a decision node in a decision tree, the history \( h \) represents the past events whereas
the set of episodes $H(M)$ that share this history represents all possible future events from this decision node on.

A special history is the empty history $h_i$ that is shared by all episodes in $H$, so $H(h_i) = H$ by definition.

### 2.6 Strategies

A strategy is a special selection of episodes from $H$ that represents a decision rule: if you observe this than do that...

Example

[In our oil wildcatter’s example, a strategy would be: perform the seismic test and only drill if a closed structure is observed. Another strategy would be drill always.]

A strategy must prescribe precisely what the decision maker has to do in every possible case. In case of an experiment, the strategy must prescribe one decision for each possible outcome. This means that all episodes in a strategy that share the same history, should have the same action as the next element or outcomes of the same experiment (unambiguity). Furthermore, every outcome of the same experiment that occurs after a history in some episode in $H$ should also occur in some episode in the strategy that shares the history (completeness):

**Definition 2.6**

A strategy $S \in H$ is a set of episodes that satisfies:

- **unambiguity:**
  \[
  8_{h;g} 8_t : h(t) = g(t) \implies AE(h(t)) = AE(g(t))
  \]

- **completeness:**
  \[
  \frac{1}{2} 8_{e \in E} : 8_j : H(h(t) + r_{e;j}) \in S \implies \exists \bar{h} : \bar{g} : h(t) + r_{e;j} \in S \implies h(t) + r_{e;j} \in S
  \]

We will use $S_H \in 2^H$ to denote the set of all strategies contained within $H$. Solving a decision tree is the same as selecting a strategy $S^* \in S_H$ that maximizes some given evaluation function. Such a strategy $S^*$ is called an optimal strategy.

Example

[An optimal strategy in the oil wildcatter’s example is: perform the seismic test and only drill if an open or closed structure is observed.]

### 2.7 Probabilities

In the informal description of decision trees, we used conditional probabilities to express the uncertainty of outcomes. With the aid of the definitions above, we can define a more formalized probability function $P : H \times [0;1]$ on episodes with the aid of the following single demand:
Demand 2.1 The sum of the probabilities $P(h)$ assigned to episodes $h \in H$ within every strategy $S \in S_H$ must exactly be 1:

$$\sum_{h \in S} P(h) = 1$$

This demand is reasonable if it is considered that only one episode can actually occur if the decision maker sticks to a strategy. Normally, however, probabilities are not assessed for episodes, but only for observed events in the decision tree, conditioned on the path to the event. These conditional probabilities can however be used to build a function $P(h)$ and we will show that this function satisfies our demand.

A conditional probability at an observable chance node in a decision tree can be expressed as the probability $P(h[t]|h(t))$ on the outcome $h[t] = r_{e_i}$ given its history $h(t)$. Because this probability can not depend on events that happen after moment $t$, the probability must be the same for all episodes that share history $h(t) + h[t]$. The conditional probability function can therefore be looked upon as a function on subsets of $H$ into $[0,1]$ and we can reformulate it as

$$P(h[t]|h(t)) = P(H(h(t) + h[t]))$$ (2.2)

The conditional probabilities at an observable chance node in a decision tree must sum up to 1, because once one is arrived at this node, only one branch will actually occur. When the oil wildcatter starts the seismic test, he will surely experience exactly one of the three outcomes. This demand can be expressed as:

$$\sum_{h \in E: h \in H} P(H(h(t) + r_{e_i})) = 1; \text{ if } (h[t]) = e$$ (2.3)

The probability after an action is by definition equal to 1:

$$\sum_{h \in A: h \in H} P(H(h(t) + a)) = 1; \text{ if } (h[t]) = a$$ (2.4)

We can now start to build function $P(h)$, the probability of an episode $h$. The probability of an episode is equal to the joined probability of all events in the episode:

$$P(h) = P(h[1]; h[2]; \ldots; h[jh])$$

But this joined probability can be expressed as a product of conditional probabilities, using the chain rule of probability theory:

$$P(h) = P(h[1]); P(h[2] | h[1]); \ldots; P(h[jh] | h[jh - 1]; h[jh - 2]; \ldots; h[1])$$

which is reduced by applying formula 2.2 into:

$$P(h) = P(H(h[1])); P(H(h[2] + h[2])); \ldots; P(H(h[jh] + h[jh]))$$
We have to prove however, that this function $P(h)$ satisfies Demand 2.1; in other words, formula:

$$\sum_{h \in S} X \cdot Y \cdot t \cdot j \cdot P(H(h(t + 1))) = 1$$  \hfill (2.5) 

must hold for every strategy $S$.

For a start we isolate the first elements of the episodes in a strategy $S$:

$$\sum_{h \in S} X \cdot Y \cdot t \cdot j \cdot P(H(h[1])) = \sum_{h \in S} X \cdot Y \cdot t \cdot j \cdot P(H(h(t + 1)))$$

split apart for actions and results:

$$\sum_{h \in S \setminus H(h(1))} X \cdot Y \cdot t \cdot j \cdot P(H(h(t + 1))) = \sum_{h \in S \setminus H(h(1))} X \cdot Y \cdot t \cdot j \cdot P(H(h(t + 1)))$$

From the definition of strategies in expression 2.1 follows however that $P(h)$ must be 1. The proof reduces therefore to:

$$\sum_{h \in S \setminus H(h(2))} X \cdot Y \cdot t \cdot j \cdot P(H(h(t + 1))) = 1$$

which is the same formula as we started with, except for the fact that we lost the first elements. This is a clue that induction is the correct way to construct a proof. We will use backward induction:

**Proof**

From the last two expressions we can formulate the induction step.

If we assume for step $i + 1$ that:

$$\sum_{h \in S \setminus H(h(i + 1))} X \cdot Y \cdot t \cdot j \cdot P(H(h(t + 1))) = 1$$  \hfill (2.6) 

then it follows that for step $i$:

$$\sum_{h \in S \setminus H(h(i))} X \cdot Y \cdot t \cdot j \cdot P(H(h(t + 1))) = \sum_{h \in S \setminus H(h(i))} X \cdot Y \cdot t \cdot j \cdot P(H(h(t + 1)))$$
We conclude the induction proof by the observation that for all steps where \( i = jh + 1 \), the number of elements in \( S \setminus H(h(i)) \) is one: only \( h \) itself is member of this set. In this case the induction assumption becomes trivial and true:

\[
X \quad Y \quad P(H(h(t + 1))) = \begin{cases} 
1 & \text{if } h[i + 1] \not\in A \\
0 & \text{otherwise} 
\end{cases}
\]

With this conclusion our proof is completed. \( \Box \)

The function \( P(\cdot) \) does not yet include the final states of the world. We define \( P(w_jh) \) as the conditional probability on the final states \( w_i \), given episode \( h \) where \( P_i P(w_jh) = 1 \) must hold for each \( h \not\in H \), because the final states are mutually exclusive and at least one of the final states must occur.

Example
[The probabilities for the oil wildcatter on the final states given episode \( h_{ai} \) are:

\[
\begin{array}{|c|}
\hline
P(w_jh_{ai}) & 0 \\
P(w_{ai}h_{ai}) & 0.5 \\
P(w_{wi}h_{ai}) & 0.3 \\
P(w_{wi}h_{ai}) & 0.2 \\
\hline
\end{array}
\]

Both probability functions \( P(h) \) and \( P(w_jh) \) can be expressed together in one joined probability function \( P(h;w) \) where \( P(h;w) = P(h)P(w_jh) \) is the probability of the occurrence of episode \( h \) and after that, final state \( w \). The relation between this joined probability function and the two marginal probability functions \( P(h) \) and \( P(w_j) \) is expressed by the following relations that follow from standard probability calculus:

\[
\begin{align*}
P(h) &= \sum_w P(h;w) \\
P(w_j) &= \sum_h P(h;w) \\
P(h;w) &= P(h)P(w_jh) \\
\end{align*}
\]
2.8 Pay-\( \alpha \) function

Choosing an optimal strategy within a decision tree is based on the expected value for each episode of some kind of pay-\( \alpha \) function. In the most general case this will be a pay-\( \alpha \) function \( V : H \not\rightarrow W \not\rightarrow R \) that assigns a real number or value to each pair of episodes in \( H \) and \( W \).

\textbf{Example}

[In the oil wildcatter's example, the values of the following episodes and \( W \)nal states are:

\[
\begin{align*}
V(h_{a\text{d}}; w_{x}) &= 70;000 \\
V(h_{r\text{c}}; a_{\text{d}}; w_{x}) &= 190;000 \\
V(h_{r\text{c}}; a_{\text{d}}; w_{s}) &= 10;000
\end{align*}
\]

The expected pay-\( \alpha \) of an episode is defined as:

\[
V(h) = \sum_{i} P(w_{i}|h):V(h; w_{i}) \quad \text{(2.7)}
\]

\textbf{Example}

[The expected pay-\( \alpha \) for \( V(h_{a\text{d}}) \) is:

\[
V(h_{a\text{d}}) = P(w_{x}|h_{a\text{d}}):V(h_{a\text{d}}; w_{x}) + P(w_{s}|h_{a\text{d}}):V(h_{a\text{d}}; w_{s}) + 0.5:70;000 + 0.3:50;000 + 0.2:200;000 = 20;000
\]

In many cases however, the pay-\( \alpha \) function is a composition of a cost function and a benefit function:

\[
V(h; w) = f(K(h; w); U(h; w)) \quad \text{(2.8)}
\]

A cost function is a function \( K : H \not\rightarrow W \not\rightarrow R \) that determines the costs of an episode for each of the \( W \)nal states. The value \( K(h; w) \) is the cost of reaching state \( w \) after episode \( h \).

The expected costs \( K(h) \) of an episode is defined as the expected value of costs for reaching each \( W \)nal state after episode \( h ):

\[
K(h) = \sum_{w_{i}} P(w_{i}|h):K(h; w_{i}) \quad \text{(2.9)}
\]

Often, the costs will not depend on the \( W \)nal state but only on the episode, so \( K(h; w_{i}) \) will be equal for all \( w_{i} \) and we can simply write \( K(h) \) for the costs of an episode instead of for the expected costs.
Example
[In the wildcatter's example, the costs only depend on the episodes:

<table>
<thead>
<tr>
<th><em>episode</em></th>
<th>cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>hand</td>
<td>70,000</td>
</tr>
<tr>
<td>here; and</td>
<td>80,000</td>
</tr>
<tr>
<td>here; and</td>
<td>10,000</td>
</tr>
</tbody>
</table>

]

A similar situation holds for the benefit functions. A benefit or utility function is a function $U : \mathcal{H} \times \mathcal{W} \to \mathbb{R}$ that assigns a utility to each of the final states, given one of the episodes.

The expected utility $U(h)$ of an episode is defined as:

$$U(h) = \sum_i P(w_i|h)U(h; w_i)$$

(2.10)

But the utility is often assigned only to the final states, indifferently on how the final states are reached. In those cases the utility function degenerates to $U : \mathcal{W} \to \mathbb{R}$ and the expected utility of an episode becomes:

$$U(h) = \sum_i P(w_i|h)U(w_i)$$

(2.11)

Example
[The utilities in the wildcatter's example are equal to the selling price of the oil that was found. Clearly, these utilities only depend on the final state. The utilities for $w_x$ and $w_d$ are zero, $U(w_x) = 120,000$, and $U(w_d) = 270,000$. The expected utility for $U(h_{ad})$ is:

$$U(h_{ad}) = P(w_x|h_{ad})U(w_x) + P(w_d|h_{ad})U(w_d) + P(w_w|h_{ad})U(w_w) + P(w_s|h_{ad})U(w_s) = 90,000$$

]

The utility and costs functions can be used in several ways to produce the pay-off function $V(h; w) = f(K(h; w); U(h; w))$, for instance:

2. Benefit-cost analysis: $V(h; w) = \Upsilon U(w) - K(h)$

2. Cost-effectiveness: $V(h; w) = U(w) - K(h)$

The expected pay-offs $V(h)$ for an episode become in the two cases above:

2. Benefit-cost analysis: $V(h) = \Upsilon U(h) - K(h)$

2. Cost-effectiveness: $V(h) = U(h) - K(h)$

---

4 We use utility here in the same meaning as in the medical domain; in economics, utility is roughly equal to pay-off.
Example
[The oil wildcatter uses bene..t analysis with $\triangle = \bar{\triangle} = 1$. So the value of episode $h_{ad}$ is simply

$$V(h_{ad}) = U(h_{ad}) \cdot K(h_{ad})$$

$$= 90,000 \cdot 70,000$$

$$= 20,000,000$$]

Because the episodes in each strategy constitutes a complete set in respect to the fact that $P_{h2s}P(h) = 1$,
we can also define expected values for strategies:

1. The expected costs $K(S)$ of a strategy: $K(S) = \prod_{h2s}P(h) \cdot K(h)$;
2. The expected utility $U(S)$ of a strategy: $U(S) = \prod_{h2s}P(h) \cdot U(h)$;
3. The expected pay-off $V(S)$ of a strategy: $V(S) = \prod_{h2s}P(h) \cdot V(h)$.

Definition 2.7 We call a pay-off function linear if the function $f$ is linear in its arguments.

From these two examples, only the bene..t analysis is linear.

Here, we observe that there is a fundamental difference between linear pay-off functions and non-linear pay-off functions, because only for linear pay-off functions it holds that the expected pay-off for a strategy is a function of the expected costs and utility of that strategy: $V(S) = f(K(S); U(S))$. In case of the bene..t analysis: $V(S) = \@U(S) \cdot \bar{K}(S)$, but in case of cost-effectiveness: $V(S) \neq U(S)$.

In fact, the same linear pay-off function $f$ that is used to compute the value of an episode $V(h) = f(K(h); U(h))$ is also used to compute the value of a strategy: $V(S) = f(K(S); U(S))$

2.9 The complete framework

Now that the probabilities and pay-off function are defined, we can complete our framework for decision trees. A decision tree $DT$ can be represented by the tuple $DT = \{A; E; R; W; H; V; P\}$ in which:

1. $A$ is the set $\{a_1; a_2; \ldots\}$ of available actions;
2. $E$ is the set $\{e_1; e_2; \ldots\}$ of available experiments;
3. $R$ is the union of sets $R_e$. Each set $R_e$ contains the outcomes $r_{e1}; r_{e2}; \ldots$ of experiment $e$;
4. $W$ is the set $\{w_1; w_2; \ldots\}$ of ..nal states of the world;
5. $H$ is the set $\{h_1; h_2; \ldots\}$ of available episodes.
6. $V: H \times W$ is a function that determines the pay-offs of each ..nal state after each episode.
7. $P: H \times W$ is a joined probability function over episodes and ..nal states.

We will use the calligraphic notation $DT$ to express the set of all decision trees $DT$ that can be expressed in this framework.
2.10 Determining an optimal strategy

Finding an optimal strategy in a decision tree can now be formulated as: find a $S^*$ such that

$$S^* = \max_{S \in S_H} \mathbb{V}(S)$$  \hspace{1cm} (2.12)

We will construct an algorithm that will compute $S^*$ for any set of episodes $H$: The algorithm acts the same as the folding-back algorithm for the decision tree: select an optimal decision, given that you have already selected optimal decisions at a deeper level. Finding the optimal strategy this way can be expressed as an optimality equation of Dynamic Programming (see [14], [34]):

$$\mathbb{V}(H(h)) = \max_{x \in A} \begin{cases} \mathbb{V}(H(h+x)) & \text{if } x \in A \land \mathbb{P}(H(h+x)) > \mathbb{V}(H(h)) \land \mathbb{V}(H(h)) \land \mathbb{V}(H(h)) \\
\mathbb{P}(H(h+r_x)) \mathbb{V}(H(h+r_x)) & \text{if } x \in E \land \mathbb{P}(H(h+r_x)) \land \mathbb{V}(H(h+r_x)) \land \mathbb{V}(H(h+r_x)) \end{cases}$$ \hspace{1cm} (2.13)

This formula can be applied because the conditional probabilities $\mathbb{P}(H(h+r)) = \mathbb{P}(r_j|h)$ do not depend on any event that happens after history $h+r$. This means that the basic demand for a dynamic program is satisfied. Solving this equation can be done most easily by recursion:

1. The algorithm for computing $S^*$ starts with the terminal equations. These equations hold for those places at which $H(h)$ contains exactly one element $h$ and where by definition $\mathbb{V}(H(h)) = \mathbb{V}(h)$. All episodes are placed in the selection.

2. Now the equation for $\mathbb{V}(H) = \mathbb{V}(H(h))$ must be solved recursively and only those episodes must remain in the selection that belong to $H(h')$ or $H(h' + r_{x'})$ for an optimal decision $x^*$.

3. When the last equation for $\mathbb{V}(H)$ is solved, an optimal strategy is found as our selection of episodes with $\mathbb{V}(S^*) = \mathbb{V}(H)$.

2.11 Conclusions

We defined a framework for decision trees based on sets of episodes and showed that every set of episodes is equivalent to a decision tree. This framework allowed us to express the concepts of pay-off functions, strategies and optimality in a closed form and also allowed us to express the optimality equation that leads to an optimal strategy. In the next chapter we will expand this framework to a framework for Markov decision trees.

---

The notation $\max_{S \in S_H}$ is used to abbreviate “select $S$ in $S_H$ that maximizes $\mathbb{V}(S)$”
Chapter 3

Markov Decision Trees

Before one can solve a decision problem by rolling back an ordinary decision tree, the probabilities of all episodes and of all final states, given each episode, must be assessed. If the number of episodes and their lengths grow large, assessing all probabilities is often not realizable.

The paradigm of influence diagrams was partly developed to handle this problem of probability assessment. To compute the probabilities in a decision tree, Howard and Matheson [9] introduced hidden variables that explained the outcomes of experiments and the occurrence of final states. The explanations were modelled by a Bayes belief network inside the influence diagram: the network laid down a joined probability distribution on all (hidden as well as observable) stochastic variables in the decision tree.

In the beginning of the development of influence diagrams, the probabilities inside the influence diagram where converted to probabilities in an ordinary decision tree. This tree was then solved by rolling back the tree in the normal way. In his Ph.D. thesis [23], Olmsted combines the conversion and the rolling back in one very efficient algorithm that made influence diagrams usable in practice. A disadvantage of influence diagrams was however (and still is) that not every ordinary decision tree can be expressed in an influence diagram: only a restricted class of decision trees, called symmetric decision trees, can be modelled by influence diagrams.

There is a number of extensions to the influence-diagram paradigm that can solve the symmetry problem, but they convert (part of) the diagram back into decision trees, thereby loosing somehow the gained efficiency of influence diagrams. Nevertheless, influence diagrams have proven to be very useful in many practical situations.

We will propose another method for computing probabilities (and pay-offs) in decision trees: the Markov decision network. We realize that we restrict ourselves to a distinct class of decision trees, but the decision problems of this class are typically not easy to express in influence diagrams. Our method can therefore be viewed as a supplement to the paradigm of influence diagrams. We start introducing our method in this chapter with an intermediate paradigm: the Markov decision tree.

In the previous chapter we introduced a framework for ordinary decision trees and showed that a decision tree (DT) can be represented by a tuple \( \text{DT} = \{A; E; R; W; H; V; P\} \) of sets and two functions. We showed what strategies are and how an optimal strategy in a decision tree can be computed. In the sequel

---

6 A symmetric decision tree has the property that all branches at every chance node have exactly the same structure.
we will focus on a special class of decision trees. In this class of Markov decision trees the probability function and the pay-off function possess a special property.

3.1 A framework for Markov decision trees

To start with we give the complete framework for Markov decision trees. It will hopefully look somewhat familiar to the reader because the framework for ordinary decision trees was designed with the purpose of extending it to this framework. A Markov decision tree (MDT) can be expressed by a tuple $MDT = hA; E; R; W; H; P_A; P_E; P_0; K; U_i$, where:

2. $A$ is the set of available actions;
2. $E$ is the set of available experiments;
2. $R$ is the union of sets $R_e$.

Each set $R_e$ contains the outcomes (or results) $f r_{e_1}; r_{e_2}; ...$ of experiment $e$;
2. $W$ is the set of possible states of the world;
2. $H$ is the set of available episodes;
2. $P_A: A \subseteq W \rightarrow [0; 1]$ is a transition probability function;
2. $P_E: E \subseteq W \subseteq R \rightarrow [0; 1]$ is a outcome probability function;
2. $P_0: W \rightarrow [0; 1]$ is an a priori probability function;
2. $K: A \times E \rightarrow R$ is a function that determines the costs of actions and experiments;
2. $U: W \rightarrow R$ is a function that determines the utility of states of the world.

The definitions of actions, experiments, outcomes, and episodes in this framework are exactly the same as in the framework for ordinary decision trees. The set of final states of the world in ordinary decision trees is enlarged to all states of the world in Markov decision trees, in order to include the prior states of the world and intermediate states of the world. So, the qualitative part of a Markov decision tree (the sets $A; E; R; W;$ and $H$) is not very different from that of an ordinary decision tree. The notable differences lie in the functional, quantitative part of the framework.

3.2 The Markov property of probabilities

The special property that we want to impose on the probability function $P$ is a so-called Markov property. This property is similar to the property that restricts ordinary multistage decision processes to Markov decision processes [34]. In general a Markov property can be viewed as a one-step memory situation: the new situation can be understood completely if only the latest situation is known. We introduce the Markov property by a refinement of the concepts of actions and experiments.
In the discussion on decision trees in section 2.1, we separated the concepts of actions and experiments by stating that the first type of activity was intended to change the state of the world and that the second type of activity was intended to gain information on the state of the world. Often the effect of an action on the state of the world depends solely on the actual state of the world and not on any previous events. Only if concerned processes or agents possess some kind of memory that influences their behavior (as for instance human beings or shape memory alloys), the same action could have different effects when the world is in the same state on two moments. In many decision problems, this mnemonic behavior is not present and the effects of actions and experiments depend solely on the actual state of the world, which is assumed to be unobservable for the decision maker.

Although the effects of actions only depend on the actual state of the world in our framework, we do interpret those effects as uncertain. This means that even if we know the actual state of the world, we still cannot tell the exact state of the world after the action is taken. But we can attach a fixed probability to each of the possible states. Every time we perform the same action in a world being in the same state, we expect those probabilities to be the same. We therefore define the effect of an action as the probability distribution over the states of the world after that action has been taken. There is however some ambiguity in this fixed uncertainty. Uncertainty is often caused by unknown processes like noise or observation errors. Before we can state that uncertain effects are fixed, given the state of the world, we have to be certain that the effects that cause the uncertainty do not depend on past events themselves. In the sequel we will simply assume that this is the case: the uncertainty is Markovian.

With respect to the experiments in the decision tree we will assume the same: the outcomes of experiments are uncertain, but the probabilities of the outcomes are fixed for each state of the world. We have to restrict the concept of experiments in Markov decision trees however somewhat more: experiments must not alter the state of the world. This means that we deny the decision maker access to the world. The occurrence of an outcome only affects the decision maker, but nothing inside the world.

To summarize, we expect

1. all uncertainty to be Markovian,
2. actions to have fixed but uncertain effects on the world, given the actual state of the world,
3. experiments to have fixed but uncertain outcomes, given the actual state of the world,
4. experiments to have no effect on the state of the world,
5. decision makers to be not able to observe the state of the world.

Example
In the example of the oil wildcatter all of these demands are satisfied:
(ad 1) The uncertainty is in this example only caused by the soil. We don't expect the soil to have any mnemonic properties, so the uncertainty will be Markovian.
(ad 2) Drilling at the same spot would always produce the same amount of oil. Of course, it is only possible to drill once at the same spot, but what is meant is that, in the simplified example of the oil wildcatter, no other factors determine the effects of drilling. The effects of drilling are even not uncertain in this example. The action of “not drilling” always yields the same result: no oil found.
(ad 3) The outcome of the seismic test in this example depends only on the structure of the soil.
(ad 4) The seismic test does not alter the structure of the soil.
(ad 5) The oil wildcatter cannot observe the amount of oil that is present in the soil.

3.2.1 State of the world

The discussion in the past section was heavily built on the concept of the state of the world and we used this concept also in the framework for ordinary decision trees, without explicitly stating its meaning. In Artificial Intelligence, describing the world that an intelligent agent has to interact with, is a large and important field of research [37]. Many types of complex Logics have for instance been developed to handle incomplete and changing knowledge about the surrounding world [1].

We use a very simplistic view on the world. We divide the enormous space of possible worlds that could occur in any imaginable situation into a partition of subsets of possible worlds that share some property and that we wish to treat as indistinguishable. Each of these subsets is simply given a name and is called a state of the world. As a result, all the possible states of the world are mutually exclusive.

The number of subsets that we wish to define is given by the requirements put on actions and experiments but also by practical arguments. If too few states are used, exact effects of actions and experiments cannot be guaranteed; if too many states are used, not all needed probabilities can be assessed. The main observations are that the state of the world is described by a single variable, the name of the state, and that the world can only be in exactly one state at a time.

In contrast to the ordinary decision-tree framework, here we also take interest in states of the world before the decision process. Only with (uncertain) knowledge about the prior state of the world we will be able to compute the probabilities of episodes and final states.

Example

[In the example of the oil wildcatter, we will extend the set of final states in the previous chapter to:

\[
W = \{ w_0, w_w, w_s, w_d, w_w0, w_w0, w_s0, w_x \}
\]

The first three states \( w_0, w_w0, \) and \( w_s0 \) express the possible states of the world, before anything is done by the oil wildcatter (hence the suffix \( x 0 \)), the other four states express the possible final states of the]
We could also have included the structure of the soil in separate states (for example: \(w_{0,0,0}\) for dry soil with closed structure) but we decided to treat the structure of the soil only within the seismic test. In this case, there are no intermediate states needed to model the problem.

### 3.2.2 The probability functions

We will now express the probability functions more formally in our framework and start with the states of the world. The set \(W\) is used to denote all possible states of the world that we wish to distinguish. In this framework we assume \(W\) to be finite. Knowledge about the prior state of the world is laid down in the probability function \(P_0\):

**Definition 3.1**: The a priori probability function \(P_0 : W \rightarrow [0; 1]\) gives the probability \(P_0(w)\) that state \(w\) actually occurs at the start of the decision process. Of course, \(\sum_w P_0(w) = 1\) must hold.

**Example**

(In the oil wildcatter’s tree, the a priori probabilities are:

\[
\begin{align*}
P_0(w_{00}) &= 0.5 & P_0(w_x) &= 0 \\
P_0(w_{0w}) &= 0.3 & P_0(w_d) &= 0 \\
P_0(w_{0w}) &= 0.2 & P_0(w_w) &= 0 \\
& & P_0(w_s) &= 0
\end{align*}
\]
)

The effect of the actions is expressed by the function \(P_A\):

**Definition 3.2**: The transition probability function \(P_A : A \times W \rightarrow [0; 1]\) gives the probability \(P_A(a; w; w')\) that state \(w'\) occurs if action \(a\) is taken when the world was in state \(w\). For this function, \(\sum_{a'} P_A(a'; w; w) = 1\) must hold for every action \(a'\) and state \(w\).

This function is called a transition function because the actions seem to “move” the world from one state to another.

**Example**

(In the oil wildcatter’s case, both actions \(a_d\) and \(a_{nd}\) have trivial transitions: if the oil wildcatter doesn’t drill, the world will always reach state \(w_x\) and if he does drill, the outcome of the drill is fixed given the state of the soil.

Any action is meaningless when the world is in states \(w_x, w_d, w_w\) or \(w_s\), so both actions cannot alter these states. This means that \(P_A(a; w; w')\) is always zero except for the following 14 combinations:

\[
\begin{align*}
P_A(a_{nd}; w_{00}; w_x) &= 1 & P_A(a_d; w_{00}; w_d) &= 1 \\
P_A(a_{nd}; w_{0w}; w_x) &= 1 & P_A(a_d; w_{0w}; w_d) &= 1 \\
P_A(a_{nd}; w_{0w}; w_w) &= 1 & P_A(a_d; w_{0w}; w_w) &= 1 \\
P_A(a_d; w_x; w_s) &= 1 & P_A(a_d; w_w; w_s) &= 1 \\
P_A(a_{nd}; w_w; w_d) &= 1 & P_A(a_d; w_w; w_d) &= 1 \\
P_A(a_{nd}; w_x; w_d) &= 1 & P_A(a_d; w_x; w_d) &= 1 \\
P_A(a_{nd}; w_x; w_w) &= 1 & P_A(a_d; w_x; w_w) &= 1 \\
P_A(a_{nd}; w_w; w_s) &= 1 & P_A(a_d; w_w; w_s) &= 1 \\
P_A(a_{nd}; w_s; w_d) &= 1 & P_A(a_d; w_s; w_d) &= 1 \\
P_A(a_{nd}; w_s; w_w) &= 1 & P_A(a_d; w_s; w_w) &= 1 \\
P_A(a_{nd}; w_s; w_s) &= 1 & P_A(a_d; w_s; w_s) &= 1
\end{align*}
\]
The results of experiments are finally expressed by the function $P_E$:

**Definition 3.3** The outcome probability function $P_E : E \times W \times R \rightarrow [0; 1]$ gives the probability $P_E(e; w; r)$ that outcome $r \in R_e$ occurs if experiment $e$ is taken when the world is in state $w$. For this function, $\sum_{e \in E} P_E(e; w; r) = 1$ must hold for every action $e \in E$ and state $w \in W$.

**Example** [Because the seismic test of the oil wildcatter is only performed when the world is in state $w_d$, $w_w$, or $w_s$, the values of $P_E(e; w; r_i)$ for other states is not important and is just set to $\frac{1}{3}$ everywhere. For the outcome probabilities of the seismic test we can use the table from chapter 1:]

- $P_E(e_t; w_d; r_n) = 0$
- $P_E(e_t; w_w; r_n) = 0$
- $P_E(e_t; w_s; r_n) = 0$
- $P_E(e_t; w_d; r_o) = 0$
- $P_E(e_t; w_w; r_o) = 0$
- $P_E(e_t; w_s; r_o) = 0$
- $P_E(e_t; w_d; r_c) = 0$
- $P_E(e_t; w_w; r_c) = 0$
- $P_E(e_t; w_s; r_c) = 0$

### 3.2.3 The probability of states, given episodes

These three probability functions define the original probability function $P : H \times W \rightarrow [0; 1]$ from the ordinary decision-tree framework completely. We start with computing the conditional probabilities $P[w|h]$ of states of the world $w$, given episode $h$. We will compute these probabilities by recursion. First we observe that the probabilities on the states given the empty episode are equal to the a priori probabilities:

$$P[w|h_i] = P_0(w) \quad (3.14)$$

Now we define the probability on the states after an action:

$$P[w|h + a] = \sum_{i} P[w; h] P_A(a; w; w) \quad (3.15)$$

This is a simple implementation of the rule of distributed probabilities from standard probability theory ([10], [20], [26]):

$$P[B|jC] = \sum_{i} P[A_i|jC] P[B|jA_i;C]$$

where $\{A_1; A_2; \ldots; A_n\}$ is a partition of the event space.

**Example** [If we want to compute the probability of a dry hole when we drill without a test, this would look like:]

$$P[w_d|h_{ad}] = \sum_{i} P[w; h] P_A(a_d; w; w_d)$$

$$= P_0(w_d) P_A(a_d; w; w_d)$$

$$= P_0(w_d) \times 0 = 0.5 \times 0 = 0.5$$

(The only states $w_i$ for which $P_A(a_d; w; w_d)$ is not zero are $w_d$ and $w_{ad}$, but $P_0(w_d) = 0$.)
The probabilities on the states after observing an outcome at an experiment are defined by:

\[ P[w_j h + r_{e;j}] = \sum_i P[w_i h] P_E(e; w_i; r_{e;j}) \]

(3.16)

which is also a simple implementation of another basic rule of probability theory: the well-known Bayes' rule ([10] [20] [26]).

\[ P[A_j j B; C] = \sum_i P[A_i j C] P[B j A_i j; C] \]

Again, \( A = A_1 | A_2 | \ldots | A_n \) is a partition of the event space.

Example

[The probability that the soil is dry after observing an open structure is:

\[ P[w_{d0} j h] = \sum_i P[w_i j h] P_E(e; w_i; r_{e;j}) \]

(3.17)

\[
\begin{align*}
&= \frac{P_0(w_{d0}) P_E(e; w_{d0}; r_0)}{P_0(w_{d0}) P_E(e; w_{d0}; r_0) + P_0(w_{w0}) P_E(e; w_{w0}; r_0) + P_0(w_{s0}) P_E(e; w_{s0}; r_0)} \\
&= \frac{0.3 \times 0.04 + 0.3 \times 0.4 + 0.2 \times 0.4}{0.3 \times 0.04 + 0.3 \times 0.4 + 0.2 \times 0.4} = 0.34286
\end{align*}
\]

With the same definitions we can easily compute the conditional probabilities \( P(h[t] j h(t)) \) that we used in chapter 2 to define the probabilities on the observable chance nodes in the decision tree:

\[
\begin{align*}
P(a j h(t)) &= 1 \\
P(r_{e;j} j h(t)) &= P[r_{e;j} j h(t)] = \sum_i P[w_i j h(t)] P_E(e; w_i; r_{e;j}) \\
P(h[t] j h(t)) &= \frac{P_0(w_{d0}) P_E(e; w_{d0}; r_0)}{P_0(w_{d0}) P_E(e; w_{d0}; r_0) + P_0(w_{w0}) P_E(e; w_{w0}; r_0) + P_0(w_{s0}) P_E(e; w_{s0}; r_0)} \\
&= \frac{0.3 \times 0.04 + 0.3 \times 0.4 + 0.2 \times 0.4}{0.3 \times 0.04 + 0.3 \times 0.4 + 0.2 \times 0.4} = 0.34286
\end{align*}
\]

(3.17)

By using this conversion, we proved implicitly that our three probability functions \( P_0, P_A, \) and \( P_E \) together satisfy Demand 2.1 and therefore constitute a legal probability distribution on the episodes in \( H \).

We conclude this section with the observation that we constructed a probability distribution on the episodes in \( H \) by using probability functions that were only defined locally on actions and experiments, together with an a priori probability distribution. The advantages of these probability functions with respect to the probability assessment are evident: we only have to assess these three probability functions separately in order to be able to compute the probability distribution in any decision tree that can be constructed from experiments and actions in \( A \mid E \).
3.3 The additive property of the pay-off function

The second deviation from the ordinary decision-tree framework that we make is the restriction on the pay-off function $V$. In chapter 2 we already introduced a costs function $K$ and a utility function $U$. In the framework for Markov decision trees we want to extend the locality of actions and experiments from the probability functions to the pay-off function. We expect that costs that come with actions and experiments do not depend on the state of the world or past events and are therefore constant in time. Obviously this is not always the case, but in many situations this restriction is not violated. The costs of an episode in this framework are simply equal to the sum of the costs that all actions and experiments in the episode bring along.

A similar restriction is put on the utility function: the utility of a (final) state of the world is not dependent on how that state was reached. This demand is less constraining than the demand on costs and in many cases this restriction will not give any difficulties. The wish to distinguish between states of the world can sometimes be motivated by the number of utility values that one wishes to use. The more precise one wants to estimate the pay-offs of a decision, the more (final) states have to be added to the set $W$.

To summarize,

1. The costs of actions and experiments should be independent from the state of the world and from past events. The costs are given by a function $K : A[ E \mapsto R$ where $K(a)$ is the cost of action $a$, and $K(e)$ the cost of experiment $e$. The costs $K(h)$ of an episode are defined as:

$$K(h) = \sum_{t \in h} K(h[t]);$$

$$K(h[t]) = K(a); \quad \text{if } h[t] = a \in A,$n$1

$$K(h[t]) = K(e); \quad \text{if } h[t] = e \in E;$$

2. The utility of states should be independent from the way the state is reached. The utility is simply a function $U : W \mapsto R$, where $U(w)$ denotes the utility of reaching state $w$. The expected utility $U(h)$ of an episode is defined as:

$$U(h) = \sum_{w_i \in h} P[w_i | h]U(w_i);$$

3. The pay-off function $V : H \mapsto W \mapsto R$ should be a function of the costs $K(h)$ of an episode $h$ and the utility $U(w)$ of a state $w:

$$V(h; w) = f(K(h); U(w))$$

An additional property that one could request from the pay-off function connects the function $V$ to be linear in $w$, meaning that the expected pay-off can be computed from the costs of an episode $K(h)$ and
the expected utility $U(h)$.

$$
V(h) = \sum_i P[w_i|h]V(h;w_i) = \sum_i P[w_i|h]f(K(h);U(w_i)) = f(K(h);U(h))
$$

### 3.4 Legal Markov decision trees and strategies

In the framework for Markov decision trees, not every arbitrary set of episodes $H$ is equivalent to a legal Markov decision tree. Because the probability on the outcomes of the experiments is only influenced by the state of the world, there must be a branch for each outcome at every chance node in the decision tree where an experiment appears. Expressed in our framework, this demand looks like:

$$
8_{e \in E} : 8_i : H(h + r_{ei}) \notin S ; \quad 8_i : H(h + r_{ei}) \in S
$$

The demands for legal strategies that we expressed in formula 2.1 become a little easier now, because the restriction on the set of episodes automatically holds also for all strategies. This leads to a new formulation of the definition of strategies:

**Definition 3.4** A strategy $S \mu H$ is a set of episodes that satisfies:

- **unambiguity**: $8_{h, g \in S} : 8_i : h(t) = g(t) \quad i ! \quad AE(h(t)) = AE(g(t))$

- **completeness**: $8_{e \in E} : 8_i : H(h + r_{ei}) \notin S \in S ; \quad 8_i : H(h + r_{ei}) \in S \in S$

### 3.5 An alternative matrix notation

This is a good moment to introduce an alternative notation that we will use often from now on. This notation is based on matrix and vector operations and allows us to simplify complex formulae in order to make them more easy to understand. The matrix notation is not suited for ordinary decision trees and has therefore not been introduced until this moment.

For each action $a \in A$ we introduce a $jWj \times jWj$ transition matrix $A_a$ such that $A_a(i;j) = P_A(a;w_j;w_i)$. The columns of $A_a$ must sum up to 1. Column $j$ contains the effect of action $a$ when the world is in state $w_j$. 

-34-
Example

The transition matrices for the oil wildcatter are:

\[
A_{\text{ad}} = \begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 \\
\end{bmatrix}
\]

\[
A_{\text{ae}} = \begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 \\
\end{bmatrix}
\]

when the states in \( W \) are ordered as:

\[
W = \{ w_0; w_0; w_0; w_s; w_d; w_d; w_w; w_w; w_s; w_g \}
\]

\[\text{Example} \]

The sensitivity matrix for the seismic test is:

\[
E_{\text{te}} = \begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 \\
\end{bmatrix}
\]

where the order of the outcomes is \( R_{\text{et}} = \{ r_f; r_o; r_c; g \} \).

\[\text{Example} \]

The sensitivity matrix for the seismic test is:

\[
E_{\text{th}} = \begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 \\
\end{bmatrix}
\]

where the order of the outcomes is \( R_{\text{eh}} = \{ r_f; r_o; r_c; g \} \).

For each experiment \( e \in E \) we introduce a \( j_{\text{R_e}j \in [0;1]}^{[W]} \) sensitivity matrix \( E_e \) such that \( E_e(i; j) = P_{E \in E}(e; w_j; r_i) \). The columns of \( E_e \) must also sum up to 1. Column \( j \) contains the outcomes of experiment \( e \) when the world is in state \( w_j \).

\[\text{Example} \]

The a priori probability vector for the amount of oil is:

\[
x_0 = \begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 \\
\end{bmatrix}
\]

where \( x_0 \) is called the a priori probability vector and, by definition, \( x_0(i) = P_0(w_i) \).

\[\text{Example} \]

The outcome probability vector for the amount of oil is:

\[
x_0 = \begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 \\
\end{bmatrix}
\]

where \( x_0 \) is the outcome probability vector and \( t_{e;h} = E_e \cdot x_h \).
Example
[The probability on the outcomes of the seismic test are equal to:
\[ t_{e_t;h_i} = E_{e_t} : x_0 \]
\[
\begin{pmatrix}
0 & 0:6 & 0:3 & 0:1 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 1 \\
0:3 & 0:4 & 0:4 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 1 & 0 \\
0:1 & 0:3 & 0:5 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 1 & 0 \\
\end{pmatrix}
\]
\[
= @ 0:35 A
\]
\[
= @ 0:35 A
\]
\[
= @ 0:35 A
\]
and the probability on the states of the world after a drill are:
\[ x_{r+w,h} = A_{a_d} : x_0 \]
\[
\begin{pmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 10 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{pmatrix}
\]
\[
= @ 0:35 A
\]
\[
= @ 0:35 A
\]
\[
= @ 0:35 A
\]
2 We use superscripts to denote the rows of a matrix and we use the notation \( d(M^1) \) to denote the diagonal matrix whose trace is equal to row \( M^1 \).

Example
[The...rst row of the sensitivity matrix of the seismic test leads to:
\[ d(E_{e_1}^1) = \begin{pmatrix}
0 & 0:3 & 0:1 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 1 \\
\end{pmatrix}
\]
\[
\begin{pmatrix}
@ & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{pmatrix}
\]
The advantage of this last notation will become evident in later chapters, but as an illustration, compare these two notational styles:
\[ P[w \ j \ h + r_{e_i}] = \frac{P[w \ j \ h]P_e(\epsilon_i \ w; r_{e_j})}{P[w \ j \ h]P_e(\epsilon_i \ w; r_{e_j})} \]
\[ x_{h+r_{e_j}} = \frac{d(E_{e_i}^1):x_h}{t_{e_t;h_i}(1)} \]

Example
[The...probabilities on the states of the world after a seismic test that pointed out no structure in the soil would be computed as:
\[ x_{r+w} = \frac{d(E_{e_i}^1):x_0}{t_{e_t;h_i}(1)} \]
Finally we define the utility vector $u \in \mathbb{R}^{|W|}$ as a row vector with $u(j) = U(w_j)$. The expected utility of an episode is simply: $U(h) = u : x_h$.

Example

The utility vector for the oil wildcatter is:

$$u = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 120;000 & 270;000 \end{bmatrix}$$

so, the expected utility after drilling will be:

$$U(h_{drill}) = u : x_{h_{drill}}$$

$$= \begin{bmatrix} 0 & 1 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0.5 & 0.3 \\ 0.2 & 0 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0.5 & 0.3 \\ 0.2 & 0 \end{bmatrix}$$

$$= 90;000$$

### 3.6 Constructing an optimal strategy

The special structure of the probability functions and pay-off functions allows us to create an algorithm that performs the computation of probabilities and pay-offs in a decision tree at the same time as rolling back the tree. This algorithm is in fact a recursion that in pre-order time computes the probabilities and in post-order time selects the optimal branches in the decision tree. A minor advantage of this algorithm is that it allows for implicitly defined Markov decision trees. This means that the set of episodes $H$ can be defined in terms of some set of rules, instead of being fully laid down in a graphical decision tree. The only requirement is that it must be decidable whether a given history $h$ is shared by some episodes in $H$.

The input parameters at each recursion step are a history $h$, the costs $K(h)$ of this history and a probability vector $x_h$ that describes the state of the world after this history. The first input parameters will be $h_1$, 0 and $x_0$ respectively. The output of each recursion step is the expected value $v$ of the pay-off $w$.

---

For a definition of pre- and post-order time see: [36]
for that part $S$ of the strategy that is collected by the deeper recursions. This value is used by the calling
recursion step at one level higher, to select the best decision.

At each recursion, the algorithm enumerates all actions and experiments in $A \cup E$. For every alternative
$x$ that has episodes in $H$, that means where $H(h + x)$ is not empty, the algorithm computes a new
history, costs and probability vector (in the case of an experiment, one for each outcome). With those
new values a recursion is started. The return values of these recursions are collected and in case of an
experiment, weighted with the probabilities on the outcomes. The alternative that receives the highest
value is selected by the algorithm and its value is returned to the higher level.

Below, the algorithm is presented in pseudo-code, based on the matrix notation:

```plaintext
sub opt($h; c; x; v; S$);
begin
  if $H(h) = fhg$ then return with $v \land f(c; x; u), \ S \land fg$ \ (1)
  else set $v \land 1; \ S \land fg$
  for all $a \in A$:
    if $H(h + a) \neq f$ then
      call opt($h + a, \ c + K(a), \ A_a; x, \ v', \ S'$);
      if $v' > v$ then set $v \land v'; \ S \land S'$
    end;
  for all $e \in E$:
    if $H(h + re) \neq f$ (for some $j$) then
      compute $t_{e;f} = E_e; x$;
      for all $i \in R_e$:
        call opt($h + re, \ c + K(e), \ \frac{d(E_e)}{t_{e;f(i)}}, \ v_i; \ S^i$);
        set $v_i \land P_{e;f(i)}v_i; \ S_i \land S^i$ \ (2)
      if $v > v$ then set $v \land v'; \ S \land S'$
    end;
end.
```

Ad (1): We expect the pay-off function $V(h; w) = f(K(h); U(w))$ to be linear in $w$, so: $V(h) = f(K(h); U(h)) = f(c; x; u)$. If this is not the case, we have to return $\sum_{i \in h} x(i)f(c; u(i))$ instead.

Ad (2): The expected pay-off of a strategy $S$ is defined as: $V(S) = \sum_{h \in S} P(h)\cdot V(h)$. This line computes with $t_{e;h(i)}$ in fact the probabilities $P(re; jh)$ and therefore implicitly also all $P(h)$'s.
Chapter 4

Markov Decision Networks

In this chapter we will introduce the framework of Markov decision networks. It is an extension of the framework of Markov decision trees in two senses. Firstly, the set \( W \) of possible states of the world will be divided into a number of groups that each contain a set of states of the world that have nonzero probability simultaneously. These state groups will reflect the concepts of actions and experiments. Secondly, we will use this division into state groups to introduce a graphical representation of Markov decision networks.

4.1 Dividing the states of the world

When looking at the state probability vectors of Markov decision trees more closely, one will be able to observe that in many cases the probabilities of some states of the world are always zero after an episode. In our oil wildcatter example, the probabilities on states \( w_{d0}; w_{w0}; \) and \( w_{s0} \) are always zero when an action (\( a_d \) or \( a_{nd} \)) has been taken. This is no surprise because we designed the states of the world such that these three states become meaningless after both actions.

The actions (\( a_d \) or \( a_{nd} \)) transfer the world, informally spoken, from one of the states: \( w_{d0}; w_{w0}; \) and \( w_{s0} \) to one of the states: \( w_{x}; w_{d}; w_{w}; \) and \( w_{s} \). Because of this, the transition matrices of the oil wildcatter’s Markov decision tree appear somewhat artificial. Large quantities of the matrices are filled with trivial values (indicated here with \( \varnothing \)'s):

\[
A_{a_d} = \begin{bmatrix}
0 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 \\
\end{bmatrix}, \quad A_{a_{nd}} = \begin{bmatrix}
0 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 \\
\end{bmatrix}
\]

It would make more sense to restrict both matrices to the non-trivial areas:

\[
A_{a_d} = \begin{bmatrix}
0 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 \\
\end{bmatrix}, \quad A_{a_{nd}} = \begin{bmatrix}
0 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 \\
\end{bmatrix}
\]

In these matrices, the columns stand for the states before the action \( f_w_{d0}; w_{w0}; w_{s0} \) and the rows for the states after the action \( f_w_{x}; w_{d}; w_{w}; \) and \( w_{s} \). We could say that the actions “transfer the world” from state group \( G_0 \) \( f_w_{d0}; w_{w0}; w_{s0} \) to state group \( G_u \) \( f_w_{x}; w_{d}; w_{w}; \) and \( w_{s} \).
The seismic test of the oil wildcatter is only meaningful if it is applied before any action has been taken, in other words, when the world is in any of the states $w_d^0; w_w^0; w_s^0$. We expressed this by filling the sensitivity matrix $E_{\alpha\iota}$ with $\frac{1}{3}$'s for the other states. It would make more sense if we removed those entries, just like we did with the transition matrices above:

$$E_{\alpha\iota} = \begin{pmatrix}
0 & 0.6 & 0.3 & 0.1 & 1 \\
0 & 0.3 & 0.4 & 0.4 & A \\
0 & 0.1 & 0.3 & 0.5 & 0
\end{pmatrix}$$

The columns in this matrix represent the states $w_d^0; w_w^0; w_s^0$. Now, we could say that the seismic test is only meaningful if "the world is in state group" $G_0 = f w_d^0; w_w^0; w_s^0$.

The two state groups $G_0 = f w_d^0; w_w^0; w_s^0$ and $G_u = f w_d; w_w; w_s$ also represent the prior and final states of the oil wildcatter's Markov decision tree. The a priori probability vector $x_0$ only has nonzero probabilities for states in $G_0$ (therefore we used the suffix "0") and the utilities are only meaningful for states in $G_u$ (hence the suffix "u"). We could restrict the a priori probability vector and the utility vector to these groups:

$$x_0 = \begin{pmatrix} 0 & 0.5 & 1 \end{pmatrix} ; u = \begin{pmatrix} 0 & 0 & 1 \end{pmatrix} 120;000 \quad 270;000 \quad ^{c}$$

By deleting parts of these matrices and vectors, their dimensions changed. But fortunately, these changes do not affect the computations we want to perform with them. When we stick to the legal episodes of the oil wildcatter's example, all matrices and vectors will fit together. We can, for example, not perform the seismic test after drilling, because the state probability vector after drilling has four elements:

$$x_{h|a|i} = A_{a_i}:x_0$$

$$= \begin{pmatrix} 1 & 0 & 0 & 1 \end{pmatrix} 1 \quad 0 \quad ^{c}$$

The product $E_{\alpha\iota}:x_{h|a|i}$ is meaningless because the matrix does not fit the vector. We could however use vector $x_{h|a|i}$ to compute the expected utility after drilling:

$$U(h_{a|i}) = u:x_{h|a|i}$$

$$= i \quad 0 \quad 0 \quad 120;000 \quad 270;000 \quad ^{c}$$

When we want to compute the state probabilities $x_{r|h}$ after the outcome no structure from the seismic test, we need $d(E_{\alpha|\iota})$. Because the sensitivity matrix is restricted to three columns, also $d(E_{\alpha|\iota})$ will be

-40-
restricted to three columns (and three rows):

\[
d(E_1^1) = \begin{bmatrix}
0 & 0.6 & 0 & 1 \\
0 & 0 & 0.3 & 0 \\
0 & 0 & 0 & 0.1
\end{bmatrix}
\]

The computations for \(x_{r_n}\) will become now:

\[
x_{r_n} = \frac{d(E_1^1) x_0}{x_0 \cdot A \cdot e_1(1)}
\]

\[
= \begin{bmatrix}
0.6 & 0 & 0 & 1 \\
0 & 0.3 & 0 & 0.5 \\
0 & 0 & 0.1 & 0.2
\end{bmatrix} \begin{bmatrix}
1 \\
1 \\
0.41
\end{bmatrix} = \begin{bmatrix}
0.07317 \\
0.21951 \\
0.04878
\end{bmatrix}
\]

The examples above showed that dividing the states of the world into certain groups makes the matrices and vectors more meaningful. But it also lays explicit restrictions on the applicability of actions and experiments. These facts play an important role in the construction of the framework for Markov decision networks.

4.2 A framework for Markov decision networks

After this informal introduction in which we transformed the Markov decision tree of the oil wildcatter into a Markov decision network, it is now time to give a more formal description of this framework. We will use the same symbols here as in the former frameworks of ordinary decision trees and Markov decision trees. First, we define the sets that constitute the building blocks of a Markov decision network.

4.2.1 The sets

A Markov decision network (MDN) consists of sets \(A; E; R; W; G_i\) where:

1. \(A\) is the set of available actions;
2. \(E\) is the set of available experiments;
3. \(R\) is the union of sets \(R_e\).
   Each set \(R_e\) contains the outcomes (or results) \(f_r; e_1, e_2, \ldots\) of experiment \(e_i\);
4. \(W\) is the set of mutually exclusive possible states of the world;
5. \(G\) is the set of state groups where \(G_i \subseteq W\) and \(\bigcup_i G_i = W\).
   (The state groups may overlap and do not have to form a strict partition of \(W\)).

We left out a set \(H\) of episodes. These will be treated separately because episodes are not needed in the definition of a Markov decision network.
4.2.2 The structure

Now we can define the functions $G^{in}, G^{out}, G^{test}$ that lay down the structure of a Markov decision network.

² Function $G^{in} : A \rightarrow G$, where $G^{in}(a)$ yields the group $G_i$ of states of the world for which action $a$ is meaningful.
² Function $G^{out} : A \rightarrow G$, where $G^{out}(a)$ yields the group $G_i$ of states of the world that can be reached when action $a$ is performed.
² Function $G^{test} : E \rightarrow G$, where $G^{test}(e)$ yields the group $G_i$ of states of the world for which experiment $e$ is meaningful.

These three functions can also be inverted:

² Function $A^{in} : G \rightarrow 2^A$, where $A^{in}(G_i)$ yields the set of actions that are meaningful when the world is in one of the states of $G_i$.
² Function $A^{out} : G \rightarrow 2^A$, where $A^{out}(G_i)$ yields the set of actions that can lead to states in $G_i$.
² Function $E^{test} : G \rightarrow 2^E$, where $E^{test}(G_i)$ yields the set of experiments that are meaningful when the world is in one of the states of $G_i$.

Among the state groups there is one special group, the start group $G_0$ that contains the prior states of the world. This is the only state group to which no action may lead, in other words, this is the only state group for which function $A^{out}$ may yield the empty set:

$$8_i : A^{out}(G_i) = ; ! \quad G_i = G_0$$

Among the state groups there is also another special group, the final group $G_u$ that contains the final states of the world. This group is the only group at which no action might be meaningful, that is, for which the function $A^{in}$ may yield the empty set:

$$8_i : A^{in}(G_i) = ; ! \quad G_i = G_u$$

In some situations these two special groups can be one and the same group ($G_0 = G_u$).

The structure of a Markov decision network can also be represented in a special graph. Each state group in $G$ is represented in the graph by a node (drawn as an ellipse). The actions in $A$ are represented by labeled arcs between the nodes. The arc that represents an action visualizes the transition of the world
that results from taking the action. Experiments in \( E \) are attached as triangular labels to the nodes in the graph. This indicates the state group at which an experiment is meaningful.

The graph for the oil wildcatter’s Markov decision network would look like this:

![Graph](image)

### 4.2.3 The probabilities

The probabilities inside a Markov decision network are expressed by a set of transition probability matrices, a set of sensitivity matrices and a prior probability vector:

For each action \( a \) in \( A \), we define a transition probability matrix \( A_a \), where \( A_a(i;j) \in [0;1] \) is the probability that state \( i \) in group \( G^{out}(a) \) will be reached if the world was in state \( j \) of group \( G^{in}(a) \) just before the action was taken. In terms of the Markov decision tree formalism:

\[
A_a(i;j) = P_A(a; w_j \in G^{in}(a); w_i \in G^{out}(a))
\]

Similarly, for each experiment \( e \) in \( E \), we define a sensitivity matrix \( E_e \), where \( E_e(i;j) \in [0;1] \) is the probability that outcome \( i \) of \( R_e \) is observed when the world is in state \( j \) of group \( G^{test}(e) \). Expressed in the Markov decision tree formalism:

\[
E_e(i;j) = P_E(e; w_j \in G^{test}(e); r_i \in R_e)
\]

The prior probability vector \( x_0 \) is a \( jG_0 \) sized vector in which \( x_0(i) \in [0;1] \) denotes the prior probability of the world being in state \( i \) of group \( G_0 \):

\[
x_0(i) = P_0(w_i \in G_0)
\]

### 4.2.4 Costs and utilities

The costs in a Markov decision network can be given by a function \( K : A \rightarrow R \) where \( K(a) \) is the cost of doing action \( a \), and \( K(e) \) the cost of performing experiment \( e \).
Utilities are expressed in a $j\mathbf{u}j$ sized utility vector $\mathbf{u}$ in which $u(i) \in \mathbb{R}$ denotes the utility of the world reaching final state $i$.

With the costs and utilities, describing the essential part of a Markov decision network is finished. As can be observes, we did not mention episodes nor strategies. In contrast with the definition of the frameworks of ordinary and Markov decision trees, episodes and strategies are not considered as an essential part of the framework itself, but as emerging from a Markov decision network. In the next section we will describe how episodes and strategies can be derived from a Markov decision network.

4.3 Episodes and strategies

In chapter 2 we introduced episodes as strings of which the elements denote events and the place of the elements in the string denote the temporal order of the events. In ordinary decision trees, the set $H$ of episodes was an essential part of the framework because this set expressed all possible courses of time, similar to the graphical representation of the decision tree. This set of possible courses of time can be viewed even as the starting point for the elicitation of the decision problem. The same situation holds for Markov decision trees.

When Markov decision networks are used to model a decision problem, one starts with describing the set of actions and experiments and the possible states of the world. The set of possible courses of time will then emerge from these descriptions. With the descriptions of actions and experiments one automatically decides when the actions and experiments are meaningful. An action or experiment cannot occur if the world is not in one of the states of the proper state group.

This means that the structure of a Markov decision network determines which episodes are valid and which not. In fact, the structure of a Markov network defines a grammar for episodes [27].

4.3.1 A grammar for episodes

Definition 4.1 The set of episodes $H(M)$ that are valid in a Markov decision network $M$ are strings that can be produced by the episode grammar $\mathcal{G}(M) = \mathcal{H}; \mathcal{T}; \mathcal{P}; \mathcal{S}$, where

1. $\mathcal{H}$ is the set of non terminals $f \mathbf{G}_1; \mathbf{G}_2; \ldots ; \mathbf{G}_n$ that correspond with the state groups in $M$.
2. $\mathcal{T}$ is the set of terminals $f a_1; a_2; \ldots ; r_e;1; r_e;2; \ldots ; g$ that correspond with actions and outcomes of experiments in $M$.
3. $\mathcal{P}$ is the set of the following production rules:
   - for every action $a$: $G^{\text{in}}(a) \rightarrow h_i; G^{\text{out}}(a) i$
   - for every outcome $r_e$: $G^{\text{test}}(e) \rightarrow h_i; G^{\text{test}}(e) i$
   - for the final state group: $G_u \rightarrow i$
4. $\mathcal{S}$ is the start symbol and is equal to the prior group $G_0$. 

-44-
Example

The episode grammar for the oil wildcatter is:

\[ N = f G_0; G_u g \]

\[ T = f a_n; a_{n+d}; r_n; r_0; r_c g \]

\[ P = G_0 ! r_n; G_0 ! r_o; G_0 ! r_c; G_0 ! r_o; \ldots ; r_o; a_{n+d} g \]

\[ S = G_0 \]

Some episodes that can be produced from this grammar are:

\[ h_{a d}, h_{a d}; h_{r c}, h_{r c}; a_{r o} \ldots ; a_{r o} \]

Not every episode that can be produced by this grammar makes sense: the seismic test will only be performed once in reality. In fact, performing the seismic test more than once in this model would produce erroneous results. We will return to this flaw in our example at the end of this chapter.

4.3.2 Evaluation of episodes

The structure of a Markov decision network allows us to define the probabilities, costs and expected utilities of valid episodes as recursive expressions. Before that, we have to introduce some more definitions.

Definition 4.2

A valid history \( h \) for a Markov decision network \( M \) is any string that is a left-hand part of a valid episode \( h \in H(M) \).

Example

[Some valid histories in the oil wildcatter’s example are: \( h_{a d}, h_{a d}; h_{r c}, h_{r c}; a_{r o} \ldots ; a_{r o} \).]

Definition 4.3

The state group \( G(h) \) is the group that is reached after a valid history \( h \). By definition \( G(h) = G_u \) for every valid episode \( h \in H(M) \).

Example

[\( G(h_{r o}) = G_0, G(h_{a d}) = G_u \).]

Definition 4.4

The state probability vector \( x_h \) is a \( |G(h)| \) sized vector in which \( x_h(i) = 0 \) if \( i \notin G(h) \); otherwise \( x_h(i) = x_0 \).

The computation of state probability vectors can be expressed by the following three rules:

1. \( x_{h_{a d}} = x_0 \)
2. \( x_{h_{r o} + a} = A_a x_{h_{r o}} \) if \( a \in A_{out}(G(h)) \)
3. \( x_{h_{r o} + r_c} = \frac{d(E_{e_{h_{r o}}})}{t_e_{h_{r o}}(i)} x_{h_{r o}} \) if \( e \in E_{test}(G(h)) \), where \( t_e_{h_{r o}} = E_e x_{h_{r o}} \)

The probability of a valid history \( P(h) \) can be expressed in similar rules:
(1) \( P(h_i) = 1 \)
(2) \( P(h + a) = P(h), \) if \( a \in A^{out}(G(h)) \)
(3) \( P(h + r_{e_i}) = t_{e_i}(h):P(h), \) if \( e \in E^{test}(G(h)) \); where \( t_{e_i} = E\cdot x_{hi} \)

The probability of valid episodes \( P(h) \) is computed the same way:

The computation of the total costs \( K(h) \) of a valid episode \( h \in H(M) \) can be expressed with the use of function \( AE \) that we introduced in chapter 3:

\[
K(h) = \sum_i K(AE(h[i]))
\]

Finally, the expected utility \( U(h) \) of a valid episode \( h \in H(M) \) is simply:

\[
U(h) = u\cdot x_h
\]

Not only valid episodes can be derived from the structure of a Markov decision network, also valid strategies can be produced by a grammar.

### 4.3.3 A grammar for strategies

Before we introduce the grammar for strategies, we first have to define a string notation for strategies. In chapters 2 and 3 we defined a strategy in ordinary and Markov decision trees as a set of episodes with some special properties. One of the properties that sets of episodes must have to be a strategy in Markov decision networks was the fact that if some outcome of an experiment occurred after a history \( h \), all other outcomes of the same experiment also had to appear after the same history. This means that if in a strategy \( S \) some episodes share \( h + r_{e_i} \) (so \( H(h + r_{e_i}) \mid S \) is not empty) than there must be a similar subset of episodes for all other outcomes \( r_{e_j} \). Suppose we have one episode for each outcome:

\[
\begin{align*}
&h + r_{e_1} + \text{rest}_1 \\
&h + r_{e_2} + \text{rest}_2 \\
&\ddots \\
&h + r_{e_n} + \text{rest}_n
\end{align*}
\]

where \( n = |R_{e_j}| \), and \( \text{rest}_i \) is the right-hand part of the episodes. We can pack these episodes in one string together by taking history \( h \), symbol \( e \) for the experiment and all the \( \text{rest}_i \)'s, each followed by a terminating symbol \( 3 \). The outcomes of the experiment are not represented explicitly in this notation.

\[
\begin{align*}
&h + r_{e_1} + \text{rest}_1 \\
&h + r_{e_2} + \text{rest}_2 \\
&\ddots \\
&h + r_{e_n} + \text{rest}_n
\end{align*}
\]
The same way of packing episodes in one string can be repeated until all episodes in the strategy are represented in the same string.

Example
[Using this notation, some strategies for the oil wildcatter would be:
  drill always: \texttt{h;ad;3}i
  don't drill: \texttt{ha;nd;3}i
  drill only if the seismic test detected an open or closed structure: \texttt{he;at;3}a;3;ad;3;au;3}i]

We can now define a strategy grammar that produces these strings:

\begin{definition}
  The set of episodes \( S(M) \) that are valid in a Markov decision network \( M \) are strings that can be produced by the strategy grammar \( S(M) = \mathcal{N} \cup \mathcal{T} \cup \mathcal{P} \cup \mathcal{S} \), where
  \begin{itemize}
    \item \( \mathcal{N} \) is the set of non terminals \( f \mathcal{G}_1; \mathcal{G}_2; \cdots; \mathcal{g} \) that correspond with the state groups in \( M \).
    \item \( \mathcal{T} \) is the set of terminals \( f \mathcal{a}_1; \mathcal{a}_2; \cdots; \mathcal{e}_1; \mathcal{e}_2; \cdots; \mathcal{3} \mathcal{g} \) that correspond with actions and experiments in \( M \).
    \item \( \mathcal{P} \) is the set of the following production rules
      \begin{align*}
        \text{for every action } \mathcal{a}: & \quad \mathcal{G}_i^{\mathcal{n}}(\mathcal{a}) \to \mathcal{h}_a; \mathcal{G}_i^{\mathcal{out}}(\mathcal{a})i \\
        \text{for every experiment } \mathcal{e}: & \quad \mathcal{G}_i^{\mathcal{test}}(\mathcal{e}) \to \mathcal{h}_e; \mathcal{G}_i^{\mathcal{test}}(\mathcal{e})i
      \end{align*}
      
    \text{for the final state group: } \mathcal{G}_u \to \mathcal{h}_3 i
  \end{itemize}
  \( \mathcal{S} \) is the start symbol and is equal to the prior group \( \mathcal{G}_0 \).
\end{definition}

Example
[The strategy grammar for the oil wildcatter is:
  \begin{itemize}
    \item \( \mathcal{N} = f \mathcal{G}_0; \mathcal{G}_u g \)
    \item \( \mathcal{T} = f \mathcal{a}_0; \mathcal{a}_n d; \mathcal{a}_u; \mathcal{e}_1; \mathcal{e}_2; \cdots; \mathcal{3} \mathcal{g} \)
    \item \( \mathcal{P} = \quad \mathcal{G}_0 \to \mathcal{h}_a; \mathcal{G}_u i; \mathcal{h}_n d; \mathcal{G}_u i; \mathcal{G}_0 \to \mathcal{h}_a; \mathcal{G}_0; \mathcal{G}_0 ; \mathcal{G}_0 ; \mathcal{G}_u i \to \mathcal{h}_3 i \)
    \item \( \mathcal{S} = \mathcal{G}_0 \)
  \end{itemize}
]

Observe that both the episode grammar and the strategy grammar are context free grammars. (The episode grammar is even regular [27]). This property of context insensibility is of course in strong accordance with the Markov property of the probabilities that we imposed.

4.3.4 Deriving episodes from strategies

A strategy defined on a Markov decision network is implicitly equivalent to a set of episodes, like strategies on (Markov) decision trees. This set of episodes can be constructed from a strategy string by a simple recursive procedure:
To construct the set $H$ of episodes in a strategy $S$, initialize $s \leftarrow S$ and $H \leftarrow \{g\}$, then call episodes($s; h; H$).

Example

The procedure episodes will produce the following set of episodes from the strategy $he; ahd; 3; ad; 3; i$: $fr_n; ahd; hr_o; ad; hr_c; adig$.

4.3.5 Evaluating strategies

The construction algorithm for episodes from strategies can function as a skeleton for computations on strategies. First we treat the computation of the expected costs $K(S)$ of a valid strategy. The expected costs are defined as:

$$K(S) = \sum_{h \in H(S)} P(h)K(h)$$

These expected costs are computed by the procedure costs. In the process of constructing all episodes, the procedure computes the probabilities of the episodes and collects all costs of events in the episodes:

The expected costs $K(S)$ of a strategy $S$ are then computed by initializing $s \leftarrow S$; $k \leftarrow 0$ and calling costs($s; x_0; k$).
Similarly the expected utility $U(S)$ of a valid strategy $S$ can be computed. The expected utility is defined by:

$$U(S) = \sum_{h \in H(S)} P(h):U(h)$$

If we take the expected utility of a valid episode as $U(h) = u:x_h$ then the procedure $\text{util}$ will compute the expected utility of a strategy:

```plaintext
sub util(var s; var x; var u)
    if $s = 3 + s^0$ then return with $s \leftarrow s^0$, $u \leftarrow u:x$;
    if $s = a + s^0$ then
        call $\text{util}(s^0; A_a; x; u^0)$;
        return with $u \leftarrow u^0$, and $s \leftarrow s^0$;
    end;
    if $s = e + s^0$ then
        compute $t_e = E:e$;
        for $1 \leq i \leq |R|$ call $\text{util}(s^0, \frac{d(E_i):x}{t_{e}(i)}; u^0)$
        return with $u \leftarrow u^0$, $t_e(i):u_i^0$, $s \leftarrow s^0$;
    end;
end.
```

The expected utility $U(S)$ of a strategy $S$ is then computed by initializing $s \leftarrow S$ and calling $\text{util}(s; x_0; u)$. The variable $u$ does not need to be initialized.

When we combine these procedures, we can compute the expected payoff value $V(S)$ of a valid (finite length) strategy. This procedure is based on the assumption that the value of a strategy equals the expected value of episodes in the strategy:

$$V(S) = \sum_{h \in H(S)} P(h):V(h)$$

and on the assumption that the value of an episode is given by:

$$V(h) = V(U(h); K(h))$$
The expected value $V$ of the strategy $S$ is then computed by initializing $s \leftarrow S$ and calling $value(s; x_0; k; u)$. When the pay-off value $V(h) = V(U(h); K(h))$ is linear, then $V(S)$ can also be computed based on $U(S)$ and $K(S)$. From a computational point of view, this would be not so wise because the computation for $U(S)$ and $K(S)$ (procedures $util$ and $costs$) contain both the enumeration of episodes and calculation of probabilities. In this case it is better to use a slightly altered version of the procedure $value$ that calculates the expected costs and utility in one pass:

```
sub value-1(var s;x;k;var u)
  if s = 3 + s_0 then return with s \leftarrow s_0, u \leftarrow u_0; k \leftarrow 0;
  if s = a + s_0 then
    call value-1(s;A_a;x;k+K(a);u);
    return with u \leftarrow u_0, k \leftarrow K(a) + k_0, and s \leftarrow s_0;
  end;
  if s = e + s_0 then
    compute t_e = E_e:x;
    for 1 \cdot i \cdot jR_{ej} : call value-1(s_0, d(E_i|x); k_E; u_i)
    return with u \leftarrow u_0, k \leftarrow K(e) + k_E, and s \leftarrow s_0;
  end;
end.
```

The computation of $V(S)$ is then performed by initializing $s \leftarrow S, k \leftarrow 0$, calling $value-1(s; x_0; k; u)$ and computing $V(S) \leftarrow V(k; u)$.

All recursive procedures in this section only work for finite-length strategies. In practice, these are often the only relevant strategies. The strategy grammar is however capable of producing infinite-length strategies. Such strategy in the oil wildcatter’s example could be: Keep repeating the seismic test until an open structure is observed and then drill. In a later chapter we will show how certain infinite-length strategies can be evaluated.
4.4 Computing an optimal strategy

In general, the computation of an optimal strategy for a Markov decision network is extremely difficult. Even if we use a linear evaluation function, this task is in general not feasible. In contrast to Markov decision trees, there is no finite set of episodes available in the framework for Markov decision networks from which an optimal strategy can be simply constructed. The set of episodes that can be derived from a Markov decision network is in almost all cases infinitely large. Only if there are no experiments in $M$ and the network graph contains no cycles then $H(M)$ is finite.

4.4.1 Special case: a restricted set of episodes

When we restrict the number of valid episodes on a Markov decision network to a finite set, then we can use a procedure similar to the opt procedure of chapter 3 for constructing an optimal strategy. This procedure assumes that there is some kind of rule that decides whether some history $\tilde{h}$ is part of any valid episode and furthermore, it assumes that this rule only accepts a finite number of finite valid episodes. In the procedure, this decision rule is called: allowed.

```
sub opt(\tilde{h};k;x;var v;var s);
begin
  set v 1; s h;
  if G(\tilde{h}) = Gv then
    set v' $\tilde{V}(k;x;u)$
    if $v' > v$ then set v $v'$; s $\tilde{h}$
end;
for all a $\in A$: if allowed(\tilde{h};a) then
  call opt(\tilde{h}+a, k+K(a), A_a;x, v', s);
  if $v' > v$ then set v $v'$; s $\tilde{h}$
end;
for all e $\in E$: if allowed(\tilde{h};e) then
  compute $t_{e,n} = E_e;x;$
  for 1 · i · |R_e|: call opt(\tilde{h}+r_ei, k+K(e), \frac{d(E_e):x}{t_{e,n}(i)};v';s);
  set v' $\tilde{A}_n p_{t_{e,n}(i)};v'; s' $s_i + s' + \ldots$
  if $v' > v$ then set v $v'$; s $\tilde{h}$
end;
end.
```

The optimal strategy is then computed by calling $\text{opt}(hi;0;x_0;V^n;S^n)$ after which $S^n$ will contain the string of the optimal strategy and $V^n$ its expected pay-off value. Note that this procedure can also be used to compute the optimal strategy given some history $\tilde{h}$. In that case the procedure should be called with: $\text{opt}(\tilde{h};K(\tilde{h});x_\tilde{h};V^n;S^n)$. -51-
There are numerous ways in which the decision rule allowed can be implemented. One of the easiest ways is to create a Markov decision tree with the elements of the network and check whether the set of episodes \( H(\mathbf{h}) \) in this tree sharing history \( \mathbf{h} \) is non-empty.

Another way would be to formulate a set of rules like:

1. never repeat any action;
2. never repeat the same experiment in between two actions;
3. do not allow histories longer than \( m \) events;
4. et cetera

and check whether an episode violates one of these rules.

### 4.4.2 Another special case: Markov decision processes

When we impose a special demand on the evaluation function, namely that the value of a strategy is the expected value of the episodes and that the value of an episode is a linear function of the expected utility and the costs \( V(h) = \mathcal{K}(h) + \mathcal{U}(h) \), then a Markov decision network is equivalent to a partially observed Markov decision process (POMDP). In this case, the algorithms to solve POMDPs can be used to construct an optimal strategy for the Markov decision network. We will treat this method in chapter 5.

### 4.4.3 An approximation method: evolutionary programming

In the most general case, we cannot assume that the evaluation of a strategy can be based on the expected value of the episodes pay-offs. Finding the optimal strategy in this case implies that all possible strategies must be enumerated. This is however often highly infeasible.

Sometimes an approximation of an optimal method would suffice for the decision maker. In such situations, evolutionary programming could be used. This is a derivation of the well-known genetic algorithm by John Holland [8]. To implement evolutionary programming, some kind of cross-over and mutation operators have to be defined on strategies. We will however not treat evolutionary programming in this report, although some experiments have already been performed with satisfactory results.

### 4.5 A critical revisit to the oil wildcatter

For the sake of clarity and simplicity, we constructed the following example Markov decision network for the oil wildcatter:

1. Actions, experiments and outcomes: \( A = \{\text{a}, \text{d}, \text{g}\}; E = \{\text{e}, \text{g}\}; R = \{\text{r}_0, \text{r}_1, \text{r}_2, \text{r}_3, \text{g}\} \)
2. State groups: \( G_0 = \{\text{w}_0, \text{w}_1, \text{w}_2, \text{w}_3, \text{g}\}; G_u = \{\text{w}_0, \text{w}_1, \text{w}_2, \text{w}_3, \text{g}\} \)
Relations: \( G^{in}(a_d) = G^{in}(a_{nd}) = G_0 \), \( G^{out}(a_d) = G^{out}(a_{nd}) = G_u \), \( G^{test}(e) = G_0 \);

Transition and sensitivity matrices:
\[
A_{a_d} = \begin{bmatrix}
0 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0
\end{bmatrix},
A_{a_{nd}} = \begin{bmatrix}
0 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0
\end{bmatrix},
E_{e_t} = \begin{bmatrix}
0.6 & 0.3 & 0.1 & 1 \\
0.3 & 0.4 & 0.4 & 1 \\
0.1 & 0.3 & 0.5 & 1
\end{bmatrix}.
\]

Prior probability and utility vector:
\[
x_0 = \begin{bmatrix}
0.5 \\
1 \\
0
\end{bmatrix},
\]

Costs: \( K(a_d) = 70,000 \), \( K(a_{nd}) = 0 \), and \( K(e_t) = 10,000 \);

Pay-offs: \( V(h) = U(h) - K(h) \).

Beside the trivial appearance of the transition matrices \( A_{a_d} \) and \( A_{a_{nd}} \), containing only zeros and ones, the real flaw in this model is the implementation of the seismic test, since the outcome of the seismic test is in fact not uncertain. The uncertainty is not caused by the test itself but by the structure of the soil being unknown. Once this structure is known, the outcome of the test is fixed. But the outcome of the test is an exact prediction of this structure. The test is deterministic. This means that once the seismic test has been performed, a repetition of the test would always produce the same outcome as the first test. Therefore the probabilities on the outcomes after the first test must be trivial:

\[
t_{e_t;r_n,1} = (1; 0; 0); \quad t_{e_t;r_n,2} = (0; 1; 0); \quad t_{e_t;r_n,3} = (0; 0; 1)
\]

The way, however, in which we modelled the seismic test would yield different outcomes after each repetition:

\[
t_{e_t;r_n,1} = E_{e_t}; x_{r_n,1} = E_{e_t}; \begin{bmatrix} E_{e_t}; x_{r_n,1} \\ x_0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0.6 & 0.3 & 0.1 & 1 \\
0.73171 & 0 & 0 & 0 & 0 & 0.50976 \\
0.21951 & 0 & 0 & 0.32683 & 0.16341
\end{bmatrix}
\]

The probability on the first outcome \( r_n \), is only 0.50976 and should be 1.

4.5.1 Going to nine prior states

The error (or false simplification) that we made is ignoring the importance of the number of states of the world. We should have modelled the structure of the soil in the states of the world, not within the seismic test. To do this, we define nine separate states in the prior state group \( G_0 \), one for each combination of structure and amount of oil:

<table>
<thead>
<tr>
<th>none</th>
<th>open</th>
<th>closed</th>
</tr>
</thead>
<tbody>
<tr>
<td>dry</td>
<td>wet</td>
<td>soaking</td>
</tr>
<tr>
<td>( W_{0:n} )</td>
<td>( W_{0:o} )</td>
<td>( W_{0:c} )</td>
</tr>
<tr>
<td>( W_{0:n} )</td>
<td>( W_{0:o} )</td>
<td>( W_{0:c} )</td>
</tr>
<tr>
<td>( W_{0:n} )</td>
<td>( W_{0:o} )</td>
<td>( W_{0:c} )</td>
</tr>
</tbody>
</table>
The sensitivity matrix becomes (ordering the states like $w_0$, $w_0$, $w_0$, $w_0$, $w_0$, $w_0$, $w_0$, $w_0$, $w_0$):

$$E_a = \begin{pmatrix} 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix}$$

The new transition matrices become:

$$A_{a_d} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \end{pmatrix}$$

$$A_{a_n} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

The sensitivity matrix of the seismic test becomes trivial: for all states $w_y$ the probability on outcome $r_y$ is 1. This means that the sensitivity matrix of the seismic test only depends on the second index of the states. Clearly, both transition matrices only depend on the first index of the states.

This observation leads us to the fact that the states of the world inside one state group can be modelled by the values of a set of stochastic variables, in this case the amount of oil and the structure of the soil. These two variables are represented here by the indices only, but there is a much better formalism to represent a set of stochastic variables and their joined probabilities: the Bayes belief network. (See for instance: [6], [7], [11], and [22])

4.5.2 Using Bayes belief networks to model state groups

The prior states of the world can be modelled by the following simple Bayes belief network:

```
  oil
  ▸
structure
```

Mark that the symbols of a Bayes belief network are delusively similar to that of Markov decision networks. The ellipses here denote the stochastic variables rather than state groups. The arrow denotes conditional dependency and not a transition.

The stochastic variable oil has three values: dry, wet and soaking and the variable structure also: none, open and closed. The conditional probability table that is connected to the arrow is very similar to our old sensitivity matrix:

$$P(\text{structure} | \text{oil}) = \begin{pmatrix}
\text{dry} & \text{wet} & \text{soaking} \\
\text{none} & 0.6 & 0.3 & 0.1 \\
\text{open} & 0.3 & 0.4 & 0.4 \\
\text{closed} & 0.1 & 0.3 & 0.5 \\
\end{pmatrix}$$

And the probability table for oil is equal to our old prior probability vector:

$$P(\text{oil}) = \begin{pmatrix}
\text{dry} & \text{wet} & \text{soaking} \\
0.5 & 0.3 & 0.2 \\
\end{pmatrix}$$
The new prior probability vector is in fact the joined probability distribution over this Bayes belief network: \( P(\text{oil} \mid \text{structure}) = P(\text{oil})P(\text{structure} \mid \text{oil}) \). This new vector now looks like:

\[
x_0 = \begin{bmatrix}
0.03 & 0.09 & 0.02 & 0.12 & 0.08 & 0.05 & 0.09 & 0.10
\end{bmatrix}^T
\]

All computations can now be based on the new matrices and vector. But when the behavior of an experiment or action only depends on a subset of the stochastic variables in a state group, not only the contents of several columns in the matrix will be identical, but performing a full matrix multiplication needed during the computations would include simple but superfluous calculations of marginal probabilities.

The probabilities on the outcome of the seismic test can be calculated as follows:

\[
t_e = E_e x_0
\]

In fact, the marginal probabilities \( P(\text{oil}) \) are calculated implicitly.

So, computations with Markov decision networks can be enhanced if Bayes belief networks are used to model the probabilities in the state groups and when the marginal probabilities that are stored in the Bayes belief network are used to perform calculations.

4.5.3 Expanding the graphical representation

The relations between actions, experiments and variables inside state groups can be presented graphically by drawing the Bayes belief networks inside the state groups and enlarging the arrows to denote the variables that are involved with the actions and experiments.
There is no need to represent the second state group $G_u$ by a similar set of variables.
Chapter 5

Markov decision networks as POMDP

When the evaluation function of a Markov decision network is linear, that means, if the pay-off value $V(S)$ for a strategy can be written as: $V(S) = \alpha U(S) + \beta K(S)$, then this Markov decision network can be treated as a partially observed Markov decision process (POMDP).

This fact has both implications for Markov decision networks and for POMDPs: on one side it means that algorithms for solving POMDPs in general could thus be used to solve this type of Markov decision networks. On the other side, these Markov decision networks form a specific type of POMDPs, which could lead to some specialized algorithms.

In this chapter we will first introduce normal Markov decision processes and partially observed Markov decision processes and show how a Markov decision network can be transformed into a POMDP. Then we will derive an algorithm for this specific type of POMDPs.

5.1 Markov decision processes

A Markov decision process is a formalism that can be used to model certain types of decision problems: a decision maker must decide which action to take on which moment in order to maximize the expected pay-offs. The effects and the pay-offs of the actions only depend on the actual state of the world and on actions that change the state of the world. For a handbook on Markov decision networks, see [35].

Example

[The toymaker’s example: The business of a toymaker is in one of two states at the beginning of each year: successful toy or unsuccessful toy. He has one of two things he can do in each of the states. If he has a successful toy he can decide to advertise or not to advertise and if he has an unsuccessful toy, he can decide to research or not to research. If the toymaker decides to advertise, then the state of his business in the next year will be successful toy with probability 0.9; if he decides not to advertise, this probability of staying successful is only 0.5. When the toymaker, in case of an unsuccessful toy, decides to do some research, the probability to become successful is 0.7 against 0.1 if he does not perform research. The rewards for the toymaker depend on the costs of advertisement and research and on the successfulness of the toy he produces.]

Formally, a (stationary, finite-state, discrete-time) Markov decision process is characterized by the following framework:

A Markov decision process is a tuple $(W; D; P; V)$ in which

$W$ is the set of states of the world;
D = D_1 \cup D_2 \cup \cdots \cup D_i \text{ is a set of decisions } d, \text{ where only the set } D_i \text{ of decisions is available if the world is in state } w_i;

function P: D \times W^2 \rightarrow [0; 1] \text{ describes the probability } P(d; w_i; w_j) \text{ that decision } d \text{ transfers the world from state } w_i \text{ into state } w_j;

function V: D \times W^2 \rightarrow R \text{ describes the value of the pay-off } V(d; w_i; w_j) \text{ when decision } d \text{ is taken when the world is in state } w_i \text{ before the decision and in } w_j \text{ afterwards.}

There are two types of objectives involved with Markov decision processes: determine a strategy that maximizes the expected pay-offs when the horizon is finite (that means that only a finite number of decisions are taken) or determine a strategy that maximizes the average pay-offs when the horizon is infinite (that means in practice that the number of decisions is unknown).

### 5.2 Partially observed Markov decision processes

Note that decisions in a Markov decision process are taken after observation of the actual state of the world. In many decision problems, like our Markov decision networks, the state of the world cannot be observed directly. The decision maker can only observe some phenomena that give no more than uncertain knowledge on the state of the world. When this type of uncertainty is added to Markov decision processes, the partially observed Markov decision processes or POMDPs come into existence (see, for instance, [2], [3], [4], and [12]).

A POMDP develops from a Markov decision process if we just add a set of outcomes (or results) R to the framework. The occurrences of the outcomes are uncertain, so we need a second probability function Q that determines the probability of an outcome in each situation. Because the actual state of the world is not observable, the set of decisions D is taken equal for all states. The complete framework for POMDPs is:

A partially observed Markov decision process is a tuple \( (W; D; R; P; Q; V) \) in which

\(^2\) W is the set \( f w_1; w_2; \cdots g \) of states of the world;

\(^2\) D is a set \( f d_1; d_2; \cdots g \) of available decisions or things to do;

\(^2\) R is a set \( f r_1; r_2; \cdots g \) of outcomes;

\(^2\) function P: D \times W \times W \rightarrow [0; 1] \text{ describes the probability } P(d; w_i; w_j) \text{ that taking decision } d \text{ transfers the world from state } w_i \text{ into state } w_j;

\(^2\) function Q: D \times W \times R \rightarrow [0; 1] \text{ describes the probability } Q(d; w_j; r_k) \text{ that outcome } r_k \text{ occurs if decision } d \text{ is taken and the world is put in state } w_j \text{ afterwards;}

\(^2\) function V: D \times W \times R \rightarrow R \text{ describes the value of the pay-off } V(d; w_i; w_j; r_k) \text{ when decision } d \text{ transfers the world from state } w_i \text{ to } w_j \text{ and outcome } r_k \text{ occurs. The notation } V_i^0 \text{ is used to denote the expected pay-off after stopping when the world is in state } w_i.\)
In general, the optimal solution to a POMDP in case of a finite horizon, can be computed with the following dynamic program (see [32]), which is an expression for the expected pay-offs at step \( n \) for a state probability vector \( x \), based on the expected pay-offs at step \( n - 1 \):

\[
V_n(x) = \max_{d \in D} \sum_i x_i \sum_{j,k} p_{ij} \sum_k q_{jk}^i \sum_l p_{il} \sum_k q_{lk}^i \left[ V_{n-1}(T(x; d; k)) \right] 
\]

(5.20)

where \( x_i = x(i) \) is the probability of the world being in state \( w_i \) and \( T(x; d; k) \) is the transformation of the state probability vector \( x \) caused by decision \( d \) and observation \( r_k \): The elements of this vector are defined by:

\[
T(x; d; k)(j) = \sum_i p_{ij} q_{jk}^i \frac{x_i}{\sum_l p_{il} q_{lk}^i}
\]

(5.21)

The subscript \( n \) in \( V_n(x) \) indicates the number of steps before the horizon is reached. This subscript thus counts backward in time.

The expected pay-off at the horizon \( V_0(x) \) is computed on base of the utility \( v^0 i \) of reaching the states of the world \( w_i \):

\[
V_0(x) = \sum_i x_i v^0 i
\]

(5.22)

Unfortunately, these formulae do not tell how to compute the solution in general. At this moment there exist a number of exact and approximate algorithms to solve this equation. In case of a Markov decision network, the formulae will become much easier to handle and this will lead to an algorithm for finding an optimal strategy.

### 5.3 Transforming a Markov decision network into a POMDP

Suppose we have a Markov decision network \( M = \{A; E; R; W; G\} \) with a pay-off function \( V(S) = U(S) - K(S) \). Then the following steps will produce an equivalent POMDP:

2 step 1: States. The states in a Markov decision network are equivalent to the states in a Markov decision process, so there is no need for conversion here:

\( W = W \)

2 step 2: Actions and Experiments. In a POMDP, every decision is followed by the observation of an outcome. In our Markov decision network, however, we separated experiments from actions: experiments have no effect on the world and actions do not lead to observations. To convert the network into a POMDP we have to collect actions and experiments in one set:

\( D = A \cup E \)

2 step 3: Outcomes. Because actions do not lead to observations in Markov decision networks, but

\[ ^{9} \text{We will use just one scalar } \cdot \text{, here instead of } \oplus \text{ and } \cdot , \text{ without loss of generality.} \]
will have to lead to at least one observation in a POMDP, we add a dummy outcome $r_0$ that is observed when any action is taken. The outcomes of the experiments are equivalent to the outcomes in the POMDP:

$$R = R \setminus \{r_0\}$$

2 step 4: transition probabilities. The probability $p_{ij}^d$ of reaching state $w_j$ from state $w_i$ after decision $d$ is for all experiments $e$ trivial: $p_{ij}^d = 1_{i=j}$ because experiments do not change the state of the world\(^\text{10}\). For actions $a$, the probability $p_{ij}^a$ is only meaningful if $w_i \in G^{in}(a)$ and $w_j \in G^{out}(a)$. Therefore we put all other probabilities to trivial values:

\[
\begin{align*}
p_{ij}^d &= 1_{i=j} \\
p_{ij}^a &= A_{a}(j;i) \quad \text{if } w_i \in G^{in}(a) \text{ and } w_j \in G^{out}(a) \\
p_{ij}^e &= 1_{i=j} \quad \text{if } w_i \in G^{in}(a) \\
p_{ij}^e &= 0 \quad \text{if } w_i \in G^{in}(a) \text{ but } w_j \not\in G^{out}(a)
\end{align*}
\]

2 step 5: outcome probabilities. For actions $a$ the probability on the outcomes is trivial again: $q_{ik}^a = 1_{k=0}$. In case of an experiment $e$, the probability $q_{ik}^e$ is only meaningful if $w_i \in G^{test}(e)$ and $r_k \in R_e$, the other probabilities are also trivial:

\[
\begin{align*}
q_{ik}^a &= 1_{k=0} \\
q_{ik}^e &= E_{e}(k;i) \quad \text{if } w_i \in G^{test}(e) \text{ and } r_k \in R_e \\
q_{ik}^e &= 1_{k=0} \quad \text{if } w_i \in G^{test}(e) \\
q_{ik}^e &= 0 \quad \text{if } w_i \in G^{test}(e) \text{ but } r_k \not\in R_e
\end{align*}
\]

2 step 6: pay-offs. The pay-offs in Markov decision networks are divided into costs of actions and experiments and utilities of final states. The special form of the pay-off function $V(S) = U(S) + K(S)$ causes that costs can be looked upon as negative pay-offs in intermediate stages and utilities as positive pay-offs at the final stage. The costs are not dependent on states of the world or on outcomes, so $v_{i,k}^a = K(d)$. The utilities of non-final states are not defined in Markov decision networks; therefore, we set $v_{i,k}^a = 1$ for non-final states:

\[
\begin{align*}
v_{i,k}^a &= i \quad \text{if } i \leq a \\
v_{i,k}^e &= i \quad \text{if } i \leq e \\
v_{i,k}^0 &= U(i) \quad \text{if } w_i \in G_u \\
v_{i,k}^0 &= 1 \quad \text{if } w_i \not\in G_u
\end{align*}
\]

With this final step the transformation of an arbitrary Markov decision network into an equivalent partially observed Markov decision process is completed. We will now derive an algorithm to solve these POMDPs.

\textsuperscript{10} We use notation $\delta_{i=j}$ to denote a delta-function that returns 1 if $i = j$ and 0 otherwise.
5.4 Transformation of state probability vectors

The transformation \( \mathbf{x}_0 = T(\mathbf{x}; d; k) \) produces the state probability vector that exists after taking decision \( d \) and then observing outcome \( k \) when the state probabilities were \( \mathbf{x} \). In the general POMDP this transformation is given by formula 5.21:

\[
T(\mathbf{x}; d; k)(j) = \frac{p_{i,j} x_i p_{k} : q_{k}}{p_{i,j} x_i p_{k} : q_{k}}
\]

This formula is easily derived because it is in fact just an application of Bayes' rule:

\[
T(\mathbf{x}; d; k)(j) = \frac{P(w_j x; d; r_k)}{P(w_j x; d)} = \frac{P(r_k | w_j) P(w_j; d)}{P(r_k; d)} = \frac{P(q_{k} : x_i p_{k} : q_{k}}{P(q_{k} : x_i p_{k} : q_{k}}
\]

For Markov decision networks, this transformation formula can be simplified a lot. The transformation in case of an action \( a \) is not dependent on the outcome, which is always \( r_0 \), so \( T(\mathbf{x}; a; k) = T(\mathbf{x}; a; 0) = T(\mathbf{x}; a) \) and the formula for \( T \) reduces to:

\[
T(\mathbf{x}; a)(j) = T(\mathbf{x}; a; 0)(j) = \frac{p_{i,j} x_i p_{a} : q_{a}}{p_{i,j} x_i p_{a} : q_{a}}
\]

or, using our matrix notation:

\[
T(\mathbf{x}; a) = A_a : \mathbf{x}
\]

Of course, the result of \( A_a : \mathbf{x} \) is formally not defined in this context because the matrix \( A_a \) does not ..t the vector \( \mathbf{x} \), but its meaning will be clear to the reader. We will come back to this in section 5.6.1.

In case of an experiment, the transformation reduces differently:

\[
T(\mathbf{x}; e; k)(j) = \frac{p_{i,j} x_i p_{e} : q_{e}}{p_{i,j} x_i p_{e} : q_{e}} = \frac{p_{i,j} x_i p_{e} : q_{e}}{p_{i,j} x_i p_{e} : q_{e}}
\]

-61-
\[ \mathbb{E}_i \mathbb{P}_j \mathbb{P}_k = \mathbb{P}_j \mathbb{P}_k \mathbb{P}_i \]

and again, in our matrix notation:

\[ T(x; e; k) = \frac{d(E_k)^{x}}{(E_e x)(k)} \] (5.24)

5.5 Reducing the optimality equation

As stated earlier in formula 5.20, in general the solution of a POMDP is given by the optimality function:

\[ V_n(x) = \max_{d \in D} \left( \sum_{i,j,k} x_i \mathbb{P}^d_{ij} \mathbb{P}^d_{ik} V_{n+1} \left( T(x; d; k) \right)^d \right) \]

The special structure of Markov decision networks allows us to simplify this expression. The first simplification is introducing the expected pay-offs \( V^d \) of decision \( d \):

\[ V^d = \max_{i,j,k} \left( \sum_{i,j,k} x_i \mathbb{P}^d_{ij} \mathbb{P}^d_{ik} \right) \]

\( V_n(x) \) can be written as:

\[ V_n(x) = \max_{d \in D} \left( \sum_{i,j,k} x_i \mathbb{P}^d_{ij} \mathbb{P}^d_{ik} V_{n+1} \left( T(x; d; k) \right)^d \right) \]

and the pay-offs \( V^d \) only depend on the decision \( d \):

\[ V^d = \max_{i,j,k} \left( \sum_{i,j,k} x_i \mathbb{P}^d_{ij} \mathbb{P}^d_{ik} \right) \]

The first reduced formula for \( V_n(x) \) therefore becomes:

\[ V_n(x) = \max_{d \in D} \left( \sum_{i,j,k} x_i \mathbb{P}^d_{ij} \mathbb{P}^d_{ik} V_{n+1} \left( T(x; d; k) \right)^d \right) \]

In the previous chapter we simplified the formulae for the transformations \( T(x; d; k) \). If we replace those formulae, we can reduce \( V_n(x) \) again:

\[ V_n(x) = \max_{d \in D} \left( \sum_{i,j,k} x_i \mathbb{P}^d_{ij} \mathbb{P}^d_{ik} \right) \]

Due to the special properties of actions and experiments, this formula can be reduced to:

\[ V_n(x) = \max_{d \in D} \left( \sum_{i,j,k} x_i \mathbb{P}^d_{ij} \mathbb{P}^d_{ik} \right) \]

-62-
the summation for experiments over indices \(i\) and \(k\) can be swapped safely and the term \(V_{n-1}(A,d;x)\) can be placed outside the summation for actions:

\[
V_n(x) = \max_{d \in D} \sum_{i,j} p_{ij} p_i^T i K(d) \text{ if } d \in A
\]
\[
= \max_{d \in D} \sum_{i,j} p_{ij} i q_k^T i K(d) \text{ if } d \in E
\]

Because \(p_{ij} = 1\) for all \(i\) and \(d \in A\), but also \(p_i x_i = 1\), the formula for \(V_n(x)\) for Markov decision networks reduces to:

\[
V_n(x) = \max_{d \in D} \sum_{i,j} p_{ij} i q_k^T i K(d) \text{ if } d \in E
\]

For \(d \in E\) the expression \(p_i x_i^k\) is replaced by the \(k\)th element of vector \(E_d:x\).

The expected pay-offs \(V_0(x)\) at the horizon are simply:

\[
V_0(x) = \sum_i u_i x_i \quad (5.26)
\]

if \(x_i\) is zero for all non-final states. Otherwise

\[
V_0(x) = 1
\]

5.6 Solving the optimality equation

Smallwood and Sondik [32] prove that the optimality equation has a simple, concave and piecewise continuous solution:

**Theorem 5.1** There always exist a set of vectors \(\theta_n\); such that

\[
V_n(x) = \max_i \theta_n x_i \quad (5.27)
\]

The proof of this theorem will simultaneously produce a recursive algorithm to find these so-called \(\theta\)-vectors.

**Proof**

We will use induction to \(n\). Formula 5.26 shows that the \(\theta\)-vectors exist for \(n = 0\):

\[
\theta_0 = \sum_i u_i
\]

Now suppose that formula 5.27 holds for \(V_{n-1}\):

\[
V_{n-1}(T(x;d;k)) = \max_i \theta_{n-1} T(x;d;k)
\]

in which \(T(x;d;k)\) is the state probability vector with \(n-1\) steps to go (i.e., after decision \(d\)). Then we
Before we proceed, we introduce two index functions. Function \( I(x; a) \) returns index \( i \) of the optimal vector \( @_{i} \) for action \( a \) and function \( I(x; e; k) \) returns index \( i \) of the optimal vector \( @_{i} \) for experiment \( e \) with outcome \( k \). These index functions will later be used to produce the optimal strategy. The formula above transforms into:

\[
\begin{align*}
V_{n+1} (A_{d}:x) &= \max_{@_{i}} A_{d}:x & \text{if } d \ 2 \ A \\
V_{n+1} (d) | x: (E_{d}:x)(k) &= \max_{@_{i}} d: (E_{d}:x)(k) & \text{if } d \ 2 \ E
\end{align*}
\]

These expressions can now be inserted in the optimality equation, formula 5.25:

\[
V_{n} (x) = \max_{d \in \mathbb{D}} : p_{h_{1}} (x; d) \cdot A_{d}:x \ \text{if } \ d \ 2 \ A \\
K (d) \ (E_{d}:x)(k) \ \text{if } \ d \ 2 \ E
\]

By placing \( x \) outside the brackets we get the final formula for \( V_{n} (x) \):

\[
V_{n} (x) = \max_{d \in \mathbb{D}} : \mu_{p} @_{i} (x; d; k) d: (E_{d}^{k}:x) \ i: K (d; 1) \ x : \text{if } d \ 2 \ E
\]

Symbol 1 denotes a row vector that contains only 1’s. This formula is clearly of the requested form \( V_{n} (x) = \max_{i} @_{i}:x \) where:

\[
@_{i} = \begin{cases}
\max_{d \in \mathbb{D}} : p_{h_{1}} (x; d) \cdot A_{d}:x : K (d; 1) \ x : \text{if } d \ 2 \ A \\
\max_{d \in \mathbb{D}} : @_{i} (x; d; k) d: (E_{d}^{k}:x) \ K (d; 1) \ x : \text{if } d \ 2 \ E
\end{cases}
\]

With this conclusion our proof is completed.

5.6.1 Using state groups and matrix notation

During the previous proof we used our matrix notation that was formally only defined in relation to state groups of Markov decision networks. There are two ways to overcome this discrepancy. The first way is to expand the definition of the matrices and vectors so that they cover all states of the world, but that would diminish the advantages of Markov decision networks. The other way is to keep the original definition of these matrices and vectors, but just adapt the vectors to the state groups and restrict the decisions \( d \) to meaningful actions and experiments.

-64-
If we define \( G(x) \) to be the state group \( G \) that belongs to the state probability vector \( x \), then we can say for instance that \( T(x; a) \) is only defined if \( G(x) = G^{in}(a) \) and \( T(x; e; k) \) is only defined if \( G(x) = G^{test}(e) \). If this is the case then \( G(T(x; a)) = G^{out}(a) \). With this restriction, formulae 5.23 and 5.24 are well defined.

The expression for the expected utility on terminal states that we used in formula 5.25 can be adjusted to this situation if we demand that \( V_0(x) \) is only defined if \( G(x) = G_u \). This means that at the horizon, in any case state group \( G_u \) must have been reached. This is however the only meaningful way of ending any episode in a Markov decision network.

The next adaptation is that we expect the \( \sigma \)-vectors to have the same number of elements as the state probability vectors. This however follows naturally from the proof above. Finally, we restrict the enumeration of \( d \) in all formulae in the proof to the sets \( A^{in}(G(x)) \) and \( E^{test}(G(x)) \) of meaningful decisions. We will use notation \( D(x) \) to denote set of the decisions that are meaningful for \( x \). This changes the conclusive formula 5.28 of the proof to:

\[
V_n(x) = \max_{d \in D(x)} \mu \left[ \sum_{k} \sum_{l} [\sigma_{ik}^{(x; d)} \cdot A_{l}^{i} \cdot K(d):1 : x] \right] \quad \text{if } d \in A^{in}(G(x))
\]

(5.29)

5.6.2 Premature ending episodes

In a Markov decision process, not all valid episodes \( h \) will have exactly the same length: not every episode reaches the horizon, or in fact most episodes will never reach the horizon. Every time that the terminal state group \( G_u \) is reached, an episode may end. We can model this behavior in the formula for \( V_n(x) \) without violating theorem 5.1:

\[
V_n(x) = \max_{d \in D(x)} \mu \left[ \sum_{k} \sum_{l} [\sigma_{ik}^{(x; d)} \cdot A_{l}^{i} \cdot K(d):1 : x] \right] \quad \text{if } d \in A^{in}(G(x))
\]

(5.30)

To denote the decision to end an episode prematurely, we added a new decision \( d_3 \) to \( D \). The resulting \( \sigma \)-vectors are:

\[
\sigma_{ik} = \begin{cases} 
\sum_{l} [\sigma_{ik}^{(x; d)} \cdot A_{l}^{i} \cdot K(d):1 : x] & \text{if } d \in A^{in}(G(x)) \\
\sum_{l} [\sigma_{ik}^{(x; d; k)} \cdot d(E_{d}^{k}) \cdot i : K(d):1 : x] & \text{if } d \in E^{test}(G(x)) \\
\sum_{l} [\sigma_{ik}^{(x; d; k)} \cdot d(E_{d}^{k}) \cdot i : K(d):1 : x] & \text{if } d = d_3 \text{ and } G(x) = G_u
\end{cases}
\]

This terminal expression for \( V_n(x) \) is the base on which we create an algorithm that constructs the optimal strategy for Markov decision networks.
5.7 An algorithm to solve Markov decision networks

The algorithm that we will present in this section is based on the exact (but unfortunately erroneous, see [18]) algorithm of Smallwood and Sondik [32]. Our algorithm is, like the original algorithm, based on the construction of so-called \( \Phi \)-areas. These \( \Phi \)-areas are sets of probability vectors \( x \) for which the same \( \Phi \)-vector is optimal in the sense of formula 5.30. This means that for all vectors \( x \) inside one \( \Phi \)-area, the same decision is optimal. From theorem 5.1 it follows that these \( \Phi \)-areas must exist and that every probability vector \( x \) must belong to one \( \Phi \)-area.

If we know these \( \Phi \)-areas for every step \( n \), then we can easily determine which decision is optimal for a certain probability vector \( x \): we have only to check in which \( \Phi \)-area this vector is located and look up which decisions gave rise to the optimal \( \Phi \)-vector of this \( \Phi \)-area.

The algorithm does not solve the Markov decision network for a single specific prior probability vector \( x_0 \), but for all possible probability vectors simultaneously. Furthermore, the algorithm does not search the whole space of possible episodes, but constructs only those episodes that belong to an optimal strategy for some probability vector. We will start with an informal description of the algorithm.

5.7.1 Informal description

Our algorithm starts with the horizon where there is only one \( \Phi \)-vector: \( \Phi : u \) and therefore also only one \( \Phi \)-area, which is the whole set of probability vectors defined on states in group \( G_u \).

We will use sets of probability vectors more often, so we introduce a formal definition here:

Definition 5.1. The set \( X_i \) is defined as the set of all possible probability vectors \( x \) on states of the world in group \( G_i \):

\[
X_i = \{ x \in [0,1]^{G_i}: \sum x(j) = 1 \}
\]

This set of probability vectors will be called the probability space on group \( G_i \).

We will use the notation \( X(\Phi) \mu X \) to denote an \( \Phi \)-area. A formal definition of an \( \Phi \)-area would be:

\[
X(\Phi) = \{ x \in [0,1]^{\Phi} : \sum x(j) = 1 \}
\]

The \( \Phi \)-area at the start of the algorithm is equal to \( X_u \), the whole probability space on group \( G_u \).

At the second step, the algorithm moves one step back in time and collects all state groups \( G \) for which a decision can lead to group \( G_u \). One of these groups is \( G_u \) itself because all experiments defined on \( G_u \) will leave the world in this state group and of course, \( d_3 \), is also a legal decision in this step.

All collected groups \( G_i \) will now be treated separately. For state group \( G_i \), the algorithm computes an \( \Phi \)-vector for all relevant decisions using formula 5.30 and the single \( \Phi \)-vector of the \( \Phi \)-rst step. The most
important and difficult step is to determine which of these \( \alpha \)-vectors are optimal for some \( x \in X_i \). Each of these relevant \( \alpha \)-vectors will give rise to an \( \alpha \)-area.

After finishing all relevant state groups, the algorithm moves another step back in time and collects all state groups that lead to any of the relevant state groups of step 2. Now the algorithm must create \( \alpha \)-vectors for all relevant decisions using formula 5.30 and for all relevant \( \alpha \)-vectors of a state group of step 2. Then again the algorithm determines the relevant new \( \alpha \)-vectors and moves to the next step.

The algorithm stops if the prior group \( G_0 \) is reached and no decision can improve the expected pay-off for any prior probability vector.

Example
We use the oil wildcatter’s example to demonstrate \( \alpha \)-vectors and \( \alpha \)-areas. The first \( \alpha \)-vector that the algorithm will find is simply the utility vector:

\[
\alpha_0 = u = (0; 0; 120; 000; 270; 000)
\]

The \( \alpha \)-area for this vector is the whole probability space for \( G_u \):

\[
X(\alpha_0) = X_u
\]

In the second step, the algorithm will collect group \( G_0 \) because both actions \( a_d \) and \( a_{nd} \) lead from state group \( G_0 \) into \( G_u \). Each one of these two actions will get an \( \alpha \)-vector:

\[
a_d : \quad \alpha_d = \alpha_0 A_{a_d} \quad 1:K(a_d) = (i \ 70; 000; 50; 000; 200; 000)
\]

\[
a_{nd} : \quad \alpha_{nd} = \alpha_0 A_{a_{nd}} \quad 1:K(a_{nd}) = (0; 0; 0)
\]

The \( \alpha \)-areas for these \( \alpha \)-vectors are subsets of \( X_0 \). In this simple case, we can describe these areas easily:

\[
X(\alpha_d) = \{ x \in X_0 \mid 70; 000 < x(w_i) < 50; 000 + 200; 000 \}
\]

and the other \( \alpha \)-area takes the rest of \( X_0 \):

\[
X(\alpha_{nd}) = \{ x \in X_0 \mid 50; 000 + 200; 000 \}
\]

Both \( \alpha \)-vectors are clearly relevant in this case.

This informal presentation of the algorithm gives rise to a number of questions: how can we determine relevant \( \alpha \)-vectors and how do we know that no decision can improve the expected pay-offs? We will first answer these questions and then present the exact formulation of the algorithm.

### 5.7.2 Determining relevant alpha-vectors

The problem of determining relevant \( \alpha \)-vectors among a set \( A \) of given \( \alpha \)-vectors is a problem of analytic geometry. It is equivalent to the detection of those hyperplanes from a given set that are the facets of a polyhedron that is determined by the total set of hyperplanes. We will use the method of Padberg [24] to solve this problem. This method is based on solving one linear program for each of the \( \alpha \)-vectors. We will explain this method with a simple example:
Example

Suppose we have a 2-dimensional probability space $X$ and a set of three $\mathbb{R}$-vectors: $\mathbb{R}$ = (1; 0), $\mathbb{R}$ = (0; 1), and $\mathbb{R}$ = (0.4; 0.4). The $\mathbb{R}$-vector $\mathbb{R}$ is relevant if there is some $x \in X$ for which $\mathbb{R}:x$, $\mathbb{R}:x$, and $\mathbb{R}:x$, $\mathbb{R}:x$. This is the same as stating that a positive real number $v$ exists such that $\mathbb{R}:x$, $\mathbb{R}:x$, $\mathbb{R}:x$, and $\mathbb{R}:x$, $\mathbb{R}:x$, or:

$\mathbb{R}:x$, $v$, 0
$\mathbb{R}:x$, $v$, 0
$\mathbb{R}:x$, $v$, 0

We have to find an $x$ and $v$ for which these inequalities hold. If such $x$ and $v$ exist, then we can just select that specific combination for which $\mathbb{R}:x$, $v$ is maximal.

This however leads to a standard linear program:

Maximize: $\mathbb{R}:x$, $v$
subject to: $\mathbb{R}:x$, $v$, 0
$\mathbb{R}:x$, $v$, 0
and: $x(i)$, $v$, 0

If the solution of this LP-problem has a negative objective value (that is, $\mathbb{R}:x$, $v$, < 0), then $\mathbb{R}$ is not a relevant $\mathbb{R}$-vector.

When we fill in the numbers information, our LP-problem becomes:

Maximize: $x(1)$, $v$
subject to: $x(2)$, $v$, 0
$0.4:x(1) + 0.4:x(2)$, $v$, 0
$x(1) + x(2)$ = 1
and: $x(i)$, $v$, 0

and simplified by elimination of $x(2)$:

Maximize: $x$, $v$
subject to: $x + v$, 1
$v$, 0.4
$x \cdot 1$
and: $x$, $v$, 0

which has solution $x^v$ = 1 (so the original $x^v$ = (1; 0)) and $v^v$ = 0.4 and a positive objective value 0.6.

The LP-problem for $\mathbb{R}$ is equivalent and has solution $x^v$ = (0; 1) and also a positive objective value 0.6.

The last LP-problem (for $\mathbb{R}$) is, however, different:

Maximize: $0.4:x(1) + 0.4:x(2)$, $v$
subject to: $x(1)$, $v$, 0
$x(2)$, $v$, 0
$x(1) + x(2)$ = 1
and: $x(i)$, $v$, 0

or by elimination of $x(2)$:

Maximize: $0.4$, $v$
subject to: $x$, $v$, 0
$x + v$, 1
$x$, 1
and: $x$, $v$, 0

This has solution $x^v$ = (1; 1; 1; 1), $v^v$ = 1 and the objective equal to -0.1.

Only $\mathbb{R}$ and $\mathbb{R}$ are relevant $\mathbb{R}$-vectors and the solutions to the LP-problems produced one example vector for each $\mathbb{R}$-area.

-68-
This example shows that we can find relevant vectors in a given set \( A \) by solving the following LP-program for every vector \( @ \in 2A \):

Maximize: \( \langle @; x \rangle \)

subject to: \( \langle @^k; x \rangle \leq 0 \), \( k \neq j \)

and: \( x(i) \leq 1 \)
\( x(i); v \geq 0 \)

If the objective of the solution is positive then vector \( @ \) is relevant and the solution \( x^* \) is situated in the area \( X(\@) \).

The complexity of detecting relevant vectors can grow enormously if the number of vectors is large. With \( |A| = n \) we have to solve \( n \) LP-problems with \( n \) inequalities each. Two possible ways to reduce the complexity are 1) to reduce the number of LP-problems and 2) to reduce the number of inequalities per LP-problem.

The number of LP-problems can be reduced by prechecking the set of vectors on trivial irrelevancy. If all elements of a given vector are smaller than the elements of another vector then \( @ \) can not be relevant. In practical situations, this type of vectors are generated very often.

The second complexity reduction is based on the following lemmas (which we will not proof):

Lemma 5.2 If vector \( @ \) is irrelevant in a given set \( A \) then \( @ \) will stay irrelevant in it when other vectors are added to \( A \).

Lemma 5.3 If vector \( @ \) is relevant in a given set \( A \) then \( @ \) will stay relevant in it when irrelevant vectors are removed from \( A \).

It is therefore allowed to check the vector in a given set \( A \) incrementally, as in this procedure:

```plaintext
sub checkvectors(A)
    set A^0 = \{ @ \}
    for all @ \in 2A:
        set A^0 \leftarrow A^0 \{ f @ g
        for all @ \in 2A^0
            check_relevancy(\@; A^0);
            if @ is irrelevant then set A^0 \leftarrow A^0 \{ f @ g
        end;
    end;
    return A^0;
end.
```

If the number of relevant vectors is small in comparison to the size of \( A \), then this incremental check-up will give a large reduction. If \( n \) is the size of \( A \) and \( m \) the number of relevant vectors in it, then this procedure will solve at most \( n \cdot m \) LP-problems with at most \( m \) inequalities per LP-problem.

-69-
The following table will make clear what the effects of this reduction are:

<table>
<thead>
<tr>
<th>n</th>
<th>m</th>
<th>standard #LP's</th>
<th>standard #LP's ≤ #ineq.</th>
<th>incremental #LP's</th>
<th>incremental #LP's ≤ #ineq.</th>
</tr>
</thead>
<tbody>
<tr>
<td>n = 10; m = 5</td>
<td></td>
<td>10</td>
<td>100</td>
<td>50</td>
<td>250</td>
</tr>
<tr>
<td>n = 100; m = 5</td>
<td></td>
<td>100</td>
<td>10,000</td>
<td>500</td>
<td>2,500</td>
</tr>
<tr>
<td>n = 1,000; m = 5</td>
<td></td>
<td>1,000,000</td>
<td>5,000,000</td>
<td>5,000</td>
<td>25,000</td>
</tr>
</tbody>
</table>

In practical situations, the number of relevant \(@\)-vectors is very small \((m \leq \binom{p}{m})\) so using this incremental procedure is often advantageous.

There are still other ways to reduce the complexity of selecting the relevant \(@\)-vectors, but these will be subject of future research.

### 5.7.3 When to stop?

The second question, that stemmed from the informal description of our algorithm, was how to detect that no decision can improve a given situation. A natural and simple way to solve this is to add all relevant \(@\)-vectors of the last time that the algorithm visited the state group to the set of newly generated \(@\)-vectors. If no new \(@\)-vector becomes relevant, then there is no improvement possible.

By adding the old relevant \(@\)-vectors we in fact introduced a so called nil-action, that has no effect on the state of the world and has no costs. If the algorithm finds in a certain step that all relevant \(@\)-vectors in the prior state \(G_0\) are connected to this nil-action, then the algorithm will stop. We will use symbol \(d_0\) to denote the nil-action.

We need a second stop-criterium in case the optimal strategy for some prior probability vector contains infinitely-large episodes. Therefore, the algorithm will also stop if the prior state \(G_0\) is reached after some predefined horizon limit \(t_{\text{max}}\).

### 5.7.4 The algorithm

The algorithm uses a data structure in which it stores information on the relevant \(@\)-vectors. For each relevant \(@^@_{i,j}^t\) vector that is generated in step \(t\) for state group \(G_j\), we define a tuple \(\eta_{i,j}^t = (\eta_{d_{i,j}^t}^t, l_{i,j}^t)\) in which

1. \(\eta_{d_{i,j}^t}^t\) is the \(@\)-vector itself,
2. \(\eta_{d_{i,j}^t}^t\) is a list of indices to \(@\)-areas in a previous step. The first two entries in this list point to the number and the relevant state group in this previous step.

If \(\eta_{d_{i,j}^t}^t = e \in E\) then \(l_{i,j}^t\) will contain one additional index for each outcome of \(e\).
\( A_i^t = \mathbf{f} \mathbf{g} ; \mathbf{d} ; \ldots ; \mathbf{g} \) is the set of all \( \mathbf{g} \)-areas on step \( t \) for state group \( G_i \). Now, at last, we can present the algorithm itself:

procedure optimize
  set \( A^0_i = \mathbf{f} \mathbf{g} ; \mathbf{d}; [i; 1:0]; \mathbf{g} \); set \( A^0_i = \mathbf{f} \mathbf{g} \) for all other state groups;
  set \( t = 0 \);
  repeat
    set \( t = t + 1 \);
    for all state groups \( G_i \), starting with \( G_0 \):
      set \( A^0_i = \mathbf{f} \mathbf{g} \);
      if \( G_i = G_u \) then call add-area(\( A^0_i ; \mathbf{h}; \mathbf{d}; [i; t; 0]; \mathbf{g} \));
      determine the largest 0 \( \cdot s < t \) such that \( A^s_i \) is not empty.
      if such \( s \) exists then
        for all areas \( j \) in \( \mathbf{f} \mathbf{g} \):
        call add-area(\( A^s_i ; \mathbf{h}; \mathbf{d}; [s; i; j]; \mathbf{g} \));
        for all experiments \( e \in \mathbf{E}^{\mathbf{test}}(G_i) \):
          for all combinations \( [j_1; j_2; \ldots] \) of \( j \) areas in \( A^s_i \):
          compute \( \mathbf{g} = \mathbf{g}^j \times (\mathbf{d} \mathbf{E}^e ; K(e) ; 1) \);
          call add-area(\( A^s_i ; \mathbf{h}; e; [s; i; j]; 1; \ldots]; \mathbf{g} \));
        end;
      end;
    end;
  for all actions \( a \) in \( G_i \):
    determine the largest 0 \( \cdot s < t \) such that \( A^s_i \) is not empty.
    if such \( s \) exists then for all areas \( j \) in \( A^s_i \):
      compute \( \mathbf{g} = \mathbf{g}^j \times (A^s_i \mathbf{h}; \mathbf{d}; [s; i; j]; \mathbf{g} \));
      call add-area(\( A^s_i ; \mathbf{h}; a; [s; i; j]; \mathbf{g} \));
    end;
    if \( G_i = G_0 \) and \( d^{\mathbf{d}; i} = d_3 \) for all areas \( j \) then stop.
  end;
  until \( t \leq t_{\text{max}} \) and \( A^t_i \) is not empty
end.

The add-area procedure handles the incremental selection of relevant \( \mathbf{g} \)-vectors:

procedure add-area(\( A; (\mathbf{g}; d; l) \))
  if redundant(\( A; \mathbf{g} \)) then exit;
  set \( A = A \mathbf{f} (\mathbf{g}; d; l) \mathbf{g} \)
  for all areas \( i \) in \( A \):
    if redundant(\( A; \mathbf{g} \)) then set \( A = A \mathbf{g} \mathbf{g} \);
  end.

And finally, the procedure redundant solves the LP problem:

function redundant(\( A; \mathbf{g} \) boolean;
  check for all areas \( i \) in \( A \):
    if \( \mathbf{g}(j) < \mathbf{g}(j) \) for all \( j \) then return true;
  solve the LP problem
  maximize \( \mathbf{g} \times \mathbf{v} \)
  subject to: \( \mathbf{g} \times \mathbf{x} = 0 \), for all areas \( i \) in \( A \)
  where:
  \( \mathbf{x}(j) = 1 \), \( \mathbf{x}(j) \mathbf{v} \geq 0 \)
  if the \( \mathbf{g} \times \mathbf{x} \geq \mathbf{v} \) then return true
else return false;
5.7.5 Complexity aspects of the algorithm

The special properties of a POMDP that is derived from a Markov decision network (decisions are divided into actions and experiments and the total probability space is divided into separate probability spaces for each state group) are utilized in this algorithm. In a similar algorithm for standard POMDPs the number of @vectors and their dimensions would have been much larger. This divide-and-conquer policy keeps both parameters as low as possible.

Due to the following line in the optimize procedure, the complexity of this algorithm can, however, still grow dramatically:

\[
\text{for all combinations } [j_1; j_2; \ldots ] \text{ of } jR_j \text{ areas in } A_i:
\]

If the number of areas for a certain state group becomes large or experiments have multiple outcomes, then the number of combinations will grow exponentially: the number of combinations for an experiment is equal to \(jA_jj^{R_j}\). Each one of these combinations produces a new @vector that has to be checked on relevancy. Further research is needed to reduce the number of combinations that have to be searched.

Another problem with the algorithm is the fact that the LP-problems become difficult to solve when the @vectors get too close to each other. This difficulty is caused by numerical properties of LP solving algorithms in general. If the number of steps in the algorithm becomes large, new @vectors generated tend to be more similar to already existing @vectors. Our algorithm should therefore also be provided with a mechanism to detect when new @vectors are too similar to existing ones to be recognized as relevant.

5.7.6 Constructing an optimal strategy

Although the optimize procedure does not produce an optimal strategy directly, the information that is stored in the areas in each \(A_i\) enables us to reconstruct an optimal strategy for any given prior probability vector \(x_0\). The first thing that has to be done is to determine to which @area this vector belongs in the set of areas \(A_0^p\) at the final step \(T\) for the prior state group \(G_0\). A straightforward way to do this is to compute:

\[
y = \max_i \arg \max \langle i^*: x_0 \rangle
\]

The value of \(\langle i^*: x_0 \rangle\) is the expected pay-off of this strategy. The following recursive procedure can be used to build the appropriate strategy string:
procedure buildstrategy(s; t; i; j)
  if d_{i,j} = d_3 then return with s = hBi;
  if d_{i,j} = d_0 then
    call buildstrategy(s^0; l_{i,j}^1(1); l_{i,j}^2(2); l_{i,j}^3(3));
    return with s = s^0;
  end;
  if d_{i,j} = a \ A then
    call buildstrategy(s^0; l_{i,j}^1(1); l_{i,j}^2(2); l_{i,j}^3(3));
    return with s = hai + s^0;
  end;
  if d_{i,j} = e \ E then
    for all outcomes k of \ e call buildstrategy(s^0_k; l_{i,j}^1(1); l_{i,j}^2(2); l_{i,j}^3(k + 2));
    return with s = hei + s^0 + s_k + \cdots;
  end;
end;

To create the correct strategy S just call buildstrategy(S; T; 0; y).

5.8 Conclusion

We showed that certain Markov decision networks with a linear pay-off function can be transformed into equivalent partially observed Markov decision processes (POMDPs). The special structure of the Markov decision networks lead to a special type of POMDP that allowed a divide-and-conquer policy. This policy was translated into an algorithm for solving Markov decision networks. The complexity of this algorithm is not bounded in all situations and also some numerical properties can give some problems in practice. More research is needed to solve the problems mentioned in this chapter.
Chapter 6

Infinite episodes and strategies

In previous chapters we concentrated solely on finite strategies. Infinite strategies, however, have some specific interest because optimal strategies for a Markov decision network are sometimes infinite. Infinite strategies are of course not executable in practice but the expected value of an infinite optimal strategy produces an upper bound for finite suboptimal strategies. The knowledge of this upper bound can help the decision maker to decide whether a suboptimal strategy is close enough to the optimum to be accepted as an practically executable alternative.

Before we treat infinite episodes and strategies, we first define what is meant by finite and infinite in this context.

Definition 6.1 A finite episode for a Markov decision network is an episode whose describing string is of finite length, which means that the number of events in the episode is finite.

Definition 6.2 A finite strategy for a Markov decision network is a strategy whose describing string is of finite length, which means that the number of episodes in the strategy is finite and that all episodes in the strategy are finite.

Infinite episodes and strategies are simply those strategies and episodes that are excluded by these definitions. But not all infinite strategies are equally complex. A strategy in the oil wildcatter’s example like perform the seismic test until the outcome denotes open structure, then drill is infinite in the sense of the definitions above, but is described here by only a small number of words. The same holds for episodes in this small example: all episodes are described by a sequence of seismic test outcomes, other than open structure, followed by an open structure outcome and the action: drill. Both strategies and episodes are expressed by grammars (see chapter 4). We will therefore use the formal language theory [27] to derive some properties of infinite strategies and episodes.
6.1 In..nite episodes

We start with the concept of in..nite episodes. We will show that all in..nite episodes that can occur in a Markov decision network can be described in a single ..nite regular expression. Then we will derive a special class of in..nite episodes: the iterative in..nite episode.

6.1.1 Regular expressions for episodes

From the theory of formal languages it follows that all ..nite and in..nite episodes in a Markov decision network can be described by one ..nite regular expression:

**Theorem 6.1 (Kleene's Theorem)** Every regular language in \( T^* \) can be described by a regular expression on \( T \).

(The notation \( T^* \) is used to denote all strings that can be formed from elements in \( T \).) The grammar for episodes \( \mathcal{H}(M) = \{N \mid \exists f G_1; G_2; \ldots, G \in \mathcal{T} \} \) is regular and produces a regular language of episode strings. Kleene's theorem states that there must be a unique regular expression that describes all episodes that can be produced by a Markov decision network.

**Example**

The regular expression for all episodes in the oil wildcatter example is:

\[ (r_n + r_c + r_o)^* (a_d + a_n) \]

which means: repeat an arbitrary number of outcomes \( r_n, r_c, r_o \), from the seismic test and conclude with \( a_d \) or \( a_n \).

(The '+' sign is used to denote choice and the '*' sign is used to denote zero or more repetitions of the preceding part.)

The regular expression for the episodes can be derived from the structure of the Markov decision network. To produce this expression, we must construct a so-called deterministic ..nite state automaton (DFSA) from the Markov decision network that produces the episodes. This is very straightforward. Let \( M = \{A; E; R; W\} \) be a Markov decision network. To construct a DFSA \( (M) = \{K; T; t; k_1; F\} \) from \( M \), set:

1. **Set of states** \( K = G_0 \)
2. **Input alphabet** \( T = A \cup R \)
3. **Start state** \( k_1 = G_0 \)
4. **Final states** \( F = f G_u g \)
5. **Transition function** \( t : K \times T \rightarrow K \times T \) with
   
   \[ t(G^{in}(a); a) = G^{out}(a) \quad \text{for actions } a \]
   
   \[ t(G^{test}(e); r_e) = G^{test}(e) \quad \text{for outcomes } r_e \]

**Example**

The DFSA for the episodes from the oil wildcatter's example looks like this:
The construction of the regular expression is then defined by recursion on the states of this automaton. We will translate this construction back to the Markov decision network. The initial step is to define expressions $T_{0}^{i;j}$ for strings of length 1:

For all $1 \leq i, j \leq |G|$: set $T_{0}^{i;j} = fg$.

For all actions $a$: $T_{0}^{i;j} = T_{0}^{i;j} + a_{i}^{G_{0}}$,

where $i = G_{0}^{in}(a)$; $j = G_{0}^{out}(a)$

For all experiments $e$: $T_{0}^{i;i} = T_{0}^{i;i} + h_{i}^{G_{0}} + r_{e,1}^{G_{0}} + r_{e,2}^{G_{0}} + \cdots + r_{e,j}^{G_{0}}$

where $i = G_{0}^{test}(e)$

The regular expressions in the subsequent levels are then defined by:

$$T_{1}^{i;j} = T_{1}^{1;i} + (T_{1}^{1;i} T_{1}^{1;j})^* T_{1}^{1;j}$$

in which the copy part is the expression of the previous step, pre.x is a way to get to state group $l$, loop is all ways to stay in state group $l$, and post.x is all ways to get from $l$ to $j$ on the previous level.

The total regular expression for episodes in the Markov decision networks is given by: $T_{1;n}^{0}$, where $n$ is the number of state groups: $|G|$.

**Example**

In the oil wildcatter’s example, with $G_1 = G_0$ and $G_2 = G_u$, the first step will produce:

$$T_{1;1}^{0} = hi + r_n + r_c + r_0$$

$$T_{1;2}^{0} = a_{hd} + a_d$$

$$T_{2;1}^{0} = T_{2;2}^{0} = fg$$

The second step produces:

$$T_{1;1}^{1} = T_{1;1}^{0} + (T_{1;1}^{0} (T_{1;1}^{0})^* T_{1;1}^{0}) = (T_{1;1}^{0})^* T_{1;1}^{0}$$

$$= (r_n + r_c + r_0)^*(hi + r_n + r_c + r_0)$$

$$T_{1;2}^{1} = T_{1;2}^{0} + (T_{1;1}^{0} (T_{1;2}^{0})^* T_{1;2}^{0}) = (T_{1;2}^{0})^* T_{1;2}^{0}$$

$$= (r_n + r_c + r_0)^*(a_{hd} + a_d)$$

$$T_{2;1}^{1} = T_{2;1}^{0} + (T_{2;2}^{0} (T_{2;2}^{0})^* T_{2;2}^{0}) = fg$$

-76-
So the complete expression is:

\[
T_{2;2}^2 = T_{2;2}^0 + (T_{2;2}^0 T_{1;2}^0)^n = f g
\]

6.1.2 Iterative episodes

In the previous section we showed that the set of episodes that can be constructed from a Markov decision network can be described by one regular expression. We also showed how this regular expression can be retrieved from the structure of the Markov decision network.

Based on this regular expression we can define a sublanguage for episodes on which a finite computation scheme can be given. We call the episodes that can produced by this sublanguage iterative episodes.

Definition 6.3 An iterative episode is an episode that can be described by a finite regular expression that contains no choices. The regular expression itself is called an iterative expression.

Example [The following expressions describe iterative episodes for the oil wildcatter’s example: \(a_{nd}\), \((r_n)^n a_{nd}\), \((r_n r_o)^n r_c r_o a_d\).]

Every iterative episode can be created from an iterative expression by selecting the proper number of iterations at each '*' sign in the iterative expression. We can abbreviate every iterative expression by subscribing the '*' signs with the actual number of iterations. This abbreviation will be called an instance of the iterative expression.

Example [Instances of the iterative expressions above are: \((r_n)^3 a_{nd}\) = \(r_n r_n r_n a_{nd}\), and \((r_n r_o)^n r_c r_o a_d\) which is an infinite instance.]

6.1.3 Computations on iterative episodes

An arbitrary instance of an iterative expression \(z\) can be written as:

\[
z = z_{pre} (z_{loop})^{n_h} z_{post}
\]

where \(z_{pre}\), \(z_{loop}\), and \(z_{post}\) are empty strings or instances of iterative expressions themselves. The computation can be split into three phases: compute the consequences of \(z_{pre}\), then \(z_{loop}\) and then \(z_{post}\).
The costs $K(z)$ of an episode expressed by $z$ are easy to compute:

$$K(z) = K(z_{\text{pre}}) + k \cdot K(z_{\text{loop}}) + K(z_{\text{post}})$$

where $K(hi) = 0$, of course. For infinite instances where $k = 1$, the costs are only defined if $K(z_{\text{loop}}) = 0$.

The state probability vector $x_z$ after an episode expressed by $z$ is only then computable for infinite instances if $z$ contains just actions. In that case, the state probability vector $x_z$ is expressible as a matrix product $A(z):x_0$ where matrix $A(z)$ only depends on episode $z$ and not on the prior probability $x_0$:

$$x_z = A(z):x_0$$

$$A(z) = A(z_{\text{post}}):A^k(z_{\text{loop}}):A(z_{\text{pre}})$$

$A(z)$ is the product of all transition matrices of actions in $z$ and we define $A(hi) = 1$. If $k = 1$ then the middle term is only well defined if $\lim_{k \to 1} A^k(z_{\text{loop}})$ exists.

If $z$ contains outcomes of any experiment, then the expression for $x_z$ can not be separated in this way and we can therefore not give a simple recursive expression for iterative episodes in general.

### 6.2 Infinite strategies

Because the grammar for strategies in a Markov decision network is not regular, it is in general not possible to give regular expressions for (infinite) strategies as we did for episodes. Only strategies that contain just actions can be described by a regular expression, but such strategies allow only one single episode and the description of these strategies is equal to the description of this episode.

In this section we will treat a special class of infinite strategies and present some computations on strategies in this class. The infinite strategies in the class that we want to focus on can all be described by a simple extension to the existing grammar. This grammar is very similar to a programming language because it contains a repeat-command. We call the infinite strategies in this class repetitive strategies.

We can now define a repetitive strategy grammar that produces these strings:

**Definition 6.4** The set of repetitive strategies $S(M)$ that are valid in a Markov decision network $M$ are strings that can be produced by the repetitive strategy grammar $R(M)$ = $\{N; T; P; S\}$, where

1. $N$ is the set of non terminals $fG_1; G_2; \cdot\cdot\cdot ; 4 g$ that correspond (except 4) with the state groups in $M$.
2. $T$ is the set of terminals $fa_1; a_2; \cdot\cdot\cdot ; e_1; e_2; \cdot\cdot\cdot ; r_{e_1}; r_{e_2}; \cdot\cdot\cdot ; 3; \text{repeat;until} g$ that correspond (except repeat, until) with actions, experiments and outcomes in $M$.
3. $P$ is the set of the following production rules:

\[ R(M) = \{N; T; P; S\} \]
for every action $a$: 
\[ G^{in}(a)i! h_a; G^{out}(a)i \]

for every experiment $e$: 
\[ G^{test}(e)i! h_e; G^{test}(e); \cdots; G^{test}(e)i \]

and for every outcome: 
\[ G^{test}(e)i! h\text{repeat}; G^{test}(e); 4; e; \text{until}; r_e; G^{test}(e)i \]

for the final state group: 
\[ G_u i! h\text{if} \]

2 $S$ is the start symbol and is equal to the prior group $G_0$.

Example

[A repetitive strategy for the oil wildcatter would be:
\[ h\text{repeat}; e; \text{until}; r_c; a_i \]

which means “Repeat the seismic test until a closed structure is encountered, then drill”. Another legal repetitive strategy would be:
\[ h\text{repeat}; \text{repeat}; e; \text{until}; r_o; e; \text{until}; r_c; a_i \]

“Repeat testing until an open structure outcome is followed directly by a closed structure outcome, then drill”.]

The repetitive strategy grammar is sensitive to context because the state group at the start of the repeat-until loop should be the same as the state group at the end of the loop. This means that once the repeat is started, the grammar should remember at which state group the until clause can be used. One way to solve this is by using an auxiliary nonterminal ($4$) that only can be removed when it is preceded by the correct nonterminal for ending the repeat-until loop.

6.3 Example computations on repetitive strategies

In this section we will compute the expected costs and utility of three repetitive-strategy examples. The examples are not fully specified, so in fact they represent classes of repetitive strategies. The objective of these example computations is to obtain some clues that could lead to a general computation rule and to efficient algorithms.

6.3.1 Example 1: Repeating Test Strategy

Suppose we have a Markov decision network with two state groups $G_0$ and $G_u$, one experiment $e$ with two outcomes $r_+$ and $r_-$ and one action $a$. What would be the expected utility, costs, and value of repetitive strategy $S = h\text{repeat}; e; \text{until}; r_+; a_i$ (repeat the experiment until a positive outcome is observed, then perform the action $a$) for the prior state probability $x_0$?

The expected utility for an arbitrary state probability vector $x$ can be expressed in an implicit formula:

\[ U(x) = P(r_+jx):U(x^+) + P(r_-jx):U(x^-) \quad (6.31) \]

where $x^+$ is the state probability vector in case of a positive outcome and $x^-$ the vector in case of a negative outcome. In case of a positive outcome, the strategy prescribes that the action $a$ should be
performed; this means that the utility can be computed directly for that branch:

\[ U(x^+) = u:A \cdot x^+ \]

The conditional probabilities on the outcomes, \( P(r_+ | x) \) and \( P(r_i | x) \) and the conditional state probability vectors \( x^+ \) and \( x^i \) can be computed from the sensitivity matrix \( E \) and the probability vector \( x \):

\[ P(r_+ | x) = (E: x)^+ \]
\[ P(r_i | x) = (E: x)^i \]
\[ x^+ = \frac{d(E^+): x}{(E: x)^+} \]
\[ x^i = \frac{d(E^i): x}{(E: x)^i} \]

(We use \((E: x)^+\) to denote the element of vector \(E: x\) that belongs to outcome \(r_+\)) When we enter these expressions in formula 6.31 we get:

\[ U(x) = (E: x)^+ \cdot u:A \cdot \frac{d(E^+): x}{(E: x)^+} + (E: x)^i \cdot u(A \cdot \frac{d(E^i): x}{(E: x)^i}) \]

\[ = u:A \cdot d(E^+): x + (E: x)^i \cdot u(A \cdot \frac{d(E^i): x}{(E: x)^i}) \quad (6.32) \]

Now we work out the right-hand call of \( U(\cdot) \) in this expression:

\[ U_1(x) = u:A \cdot d(E^+): x \]
\[ + (E: x)^i : u:A \cdot d(E^+): \frac{d(E^i): x}{(E: x)^i} + (E: x)^i \cdot u(A \cdot \frac{d(E^i): x}{(E: x)^i}) \]
\[ = u:A \cdot (d(E^+): d(E^i)) : x + (E: d(E^i)): x \cdot u(A \cdot \frac{d(E^i): x}{(E: d(E^i)): x}) \]

We work out the call of \( U(\cdot) \) by using 6.32 again and obtain:

\[ U_2(x) = u:A \cdot (d(E^+): d(E^i) + d(E^+): d(E^i): d(E^i)) : x \]
\[ + (E: d(E^i)): d(E^i): x \cdot u(A \cdot \frac{d(E^i): d(E^i): x}{(E: d(E^i)): d(E^i): x}) \]

We can already recognize some structure in this expression:

\[ U_2(x) = u:A \cdot (d(E^+)^i : (d(E^i)^i): x + (E: d(E^i)^2: x)^i) \cdot u(A \cdot \frac{d(E^i)^3: x}{(E: d(E^i)^2: x)}) \]

Without proof we state that:

\[ U_n(x) = u:A \cdot (d(E^+)^i : (d(E^i)^i): x + (E: d(E^i)^n: x)^i) \cdot u(A \cdot \frac{d(E^i)^{n+1}: x}{(E: d(E^i)^n: x)}) \]

This can be proven easily with induction to \( n \). To compute the expected utility \( U(x_0) \) we can take the limit for \( n \) ! 1:

\[ U(x_0) = \lim_{n \to \infty} U_n(x_0) \]
The right-hand limit will go to zero because \( U() \) is bounded and the elements of \( d(E) \) will go to zero if \( n \) reaches infinity. (Remember that \( d(E) \) is a diagonal matrix and that all elements on the diagonal are probabilities.) This means that

\[
U(x_0) = \lim_{n \to \infty} u(A) \cdot (d(E^+) \cdot \sum_{i=0}^{\infty} d(E^{-}) \cdot x_0)
\]

From elementary calculus we know however that:

\[
\lim_{n \to \infty} \sum_{i=0}^{\infty} \frac{1}{i!} p^i = \frac{1}{1 - p}; \quad \text{if } p \in [0; 1)
\]

Because the summation, power and inverse of diagonal matrices are in exact element-wise operations, the same limit can be used for \( d(E^+) \) as well, under the restriction that the probability of outcome \( r_i \) is not equal to 1 for any state:

\[
\lim_{n \to \infty} \sum_{i=0}^{\infty} \frac{1}{\prod_{j=1}^{i} d(E^{-})} = \frac{1}{\prod_{j=1}^{i} d(E^{-})}; \quad \text{if } d(E^{-}) \prod_{j=1}^{i} d(E^{-}) \in [0; 1)
\]

This reduces the formula for \( U(x_0) \) to:

\[
U(x_0) = u(A) \cdot \sum_{i=0}^{\infty} \frac{d(E^+)}{\prod_{j=1}^{i} d(E^{-})} \cdot x_0
\]

But our experiment has only two outcomes: \( r_+ \) and \( r_1 \), so the diagonal matrices \( d(E^+) \) and \( d(E^{-}) \) must be complementary: \( d(E^+) = I - d(E^{-}) \). Our expression for \( U(S) \) finally becomes trivial:

\[
U(S) = U(x_0) = u(A) \cdot x_0
\]

It seems as if the experiment has never been performed! This however makes sense because the utility of an episode does not depend on the episode itself but only on the final state probability vector. The probability of eventually receiving a positive outcome after some number of experiments is 1 for every prior state in \( G_0 \), so the action is performed with certainty. The experiment itself does not cause any change to the state of the world, even if it is performed infinitely many times. Together this means that the experiment can be viewed as being not performed at all, in respect to the expected utility.

This is surely not the case for the expected costs. The expected costs for the strategy are computed in a similar way as the expected utility. We start with the implicit formula:

\[
K(x) = K(e) + P(r_+jx):K(a) + P(r_1,jx):K(x^i)
\]

We know from the previous discussion that the action \( a \) is always performed once. So we can leave out the costs \( K(a) \) of the action here and add it to the expected costs afterwards:
\[
K(x) = K(e) + P(r_j|x):K(x') \\
= K(e) + (E:x')^i :K(\frac{d(E^{i}):x}{(E:x)^i})
\] (6.34)

This expression can be worked out in the same way as above:

\[
K_1(x) = K(e) + (E:x)^i :K(e) + (E:d(E^{i}):x)^i :K(\frac{d(E^{i}):d(E^{i}):x}{(E:d(E^{i}):x)^i})
\]

\[
= (1 + (E:x)^i )K(e) + (E:d(E^{i}):x)^i :K(\frac{d(E^{i}):d(E^{i}):x}{(E:d(E^{i}):x)^i})
\]

Then by applying 6.34 again:

\[
K_2(x) = (1 + (E:x)^i + (E:d(E^{i}):x)^i )K(e) + (E:d(E^{i}):x)^i :K(\frac{d(E^{i}):d(E^{i}):d(E^{i}):x}{(E:d(E^{i}):d(E^{i}):x)^i})
\]

and clearly:

\[
K_n(x) = (1 + (E^{X0}) d(E^{i})^i :x)^i )K(e) + (E:d(E^{i})^n:x)^i :K(\frac{d(E^{i})^{n+1}:x}{(E:d(E^{i})^n:x)^i})
\]

Taking the limit for \( n \) \( \to 1 \):

\[
K(x_0) = \lim_{n \to 1} K_n(x_0)
\]

\[
= \lim_{n \to 1} (1 + (E^{X0}) d(E^{i})^i :x)^i )K(e) + \lim_{n \to 1} (E:d(E^{i})^n:x_0)^i :K(\frac{d(E^{i})^{n+1}:x_0}{(E:d(E^{i})^n:x_0)^i})
\]

The right-hand limit goes to zero, just as above:

\[
K(x_0) = (1 + (E \lim_{i=0}^{X0} d(E^{i})^i :x_0)^i )K(e)
\]

\[
= (1 + (\frac{E}{E_+} :x_0)^i )K(e)
\]

This expression can be simplified to (where we introduce the costs for the action again):

\[
K(S) = K(x_0) = (1 + \frac{E_i}{E_+} :x_0)K(e) + K(a)
\] (6.35)

The term \( \frac{E_i}{E_+} \) is the vector that results from a elements-wise division of \( E_i \) and \( E_+ \).

This result can also be achieved in another way. This specific strategy in this Markov decision network is equivalent to a series of separate Markov chains [28]. Suppose that the prior state of the world is \( x_i \) with certainty (that is, \( x_0 = (\cdots; 0; 1; 0; \cdots) \)). The state of the world will not change as long as only the experiment is being performed and the probability on the outcomes \( r_+ \) and \( r_i \) will therefore stay the same.

This means that we have in fact a Markov chain \( C_i \) for every state of the world \( x_i \) in \( G_0 \). Each Markov chain has two states: \( c_+ \) for an observed positive outcome and \( c_i \) for an observed negative outcome.
From these two states, \( c_i^+ \) is an absorbing state: \( P(c_i^{t+1}|c_i^+) = 0 \). The costs at each transition are constant and equal to \( K(e) \) .

The number of experiments performed is equal to the number of times we have to wait before reaching state \( c_i^+ \). This number is equal to

\[
\lim_{n \to +\infty} \frac{1}{n} \sum_{k=0}^{n-1} P(c_i^{t+1}|c_i^+) = \frac{1}{1 - P(c_i^+|c_i)}
\]

when the starting state is \( c_i^- \). The total costs for this Markov chain are then:

\[
K(x_i) = K(e) + \frac{P(c_i^+|c_i)}{1 - P(c_i^-|c_i)} \cdot K(e) + \frac{E_i(i)}{E_i^-} \cdot K(e) + 1 \cdot K(e)
\]

When all Markov chains are combined and \( x_0 \) is used to weigh the costs, then formula 6.35 is obtained (without the costs of the action):

\[
K(x_0) = \sum_{i} x_0(i) \cdot K(x_i) = (1 + \frac{E_i^-}{E_i^+} \cdot x_0) \cdot K(e)
\]

If the value of our strategy \( V(S) \) is linear (\( V(S) = q \cdot U(S) + K(S) \)) then we can finally combine formulae 6.33 and 6.35 into:

\[
V(S) = (q \cdot u : A \cdot \frac{E_i^-}{E_i^+} : x_0) \cdot (K(a) + K(e))
\] (6.36)

6.3.2 Example 2: Repeating Action Strategy

In this second example we take a Markov decision net with one state group \( G_0 = G_0 \), one action \( a \) and one experiment \( e \) that has two outcomes \( r_+ \) and \( r_- \). What would be the expected utility, costs, and value of the strategy \( S = \text{repeat}; a; e; \text{until}; r_+ i \)?

The implicit expression for the utility is:

\[
U(x) = u(d(E^-): A \cdot A \cdot x + (E: A : x) \cdot U(d(E^+): A \cdot x))
\]
After working out this expression we obtain for step $n$:

$$U_n(x) = u : d(E^+) : A \sum_{i=0}^{n} (d(E^i) : A)^i : x +$$

$$(E : A : (d(E^i) : A)^n : x)^i : U(\frac{(d(E^i) : A)^{n+1}}{(E : A : (d(E^i) : A)^n : x)^i})$$

When taking the limit of this expression for $n \to 1$ the right-hand part will go to zero and we obtain:

$$U(x_0) = \lim_{n \to 1} U_n(x_0) = u : d(E^+) : A \sum_{i=0}^{\infty} (d(E^i) : A)^i : x_0$$

In the previous example we showed that the limit $\lim_{n \to 1} n \sum_{i=0}^{n} b_i = (I \cdot B)^i$ existed because $d(E^i)$ was a diagonal matrix whose diagonal elements were positive and smaller than 1. But this is not the case for matrix $d(E^i) : A$. Fortunately, we can derive a expression for our requested limit using the following lemma:

**Lemma 6.2** If the inverse of $(I \cdot B)$ exists then $\lim_{n \to 1} n \sum_{i=0}^{n} B_i = (I \cdot B)^i$.

**Proof**

Changing the notation slightly we get:

$$\sum_{i=0}^{\infty} B_i = (I \cdot B)^i$$

but,

$$(I \cdot B) \sum_{i=0}^{\infty} B_i = \sum_{i=0}^{\infty} B_i \cdot (B^i) = \sum_{i=0}^{\infty} B_i = I$$

which proofs the lemma.

But the proof also shows that if $\sum_{i=0}^{\infty} B_i$ converges, then $(I \cdot B)^i$ must exist. We therefore only have to investigate whether $\sum_{i=0}^{\infty} B_i$ converges. From linear algebra we know that the convergence of $\lim_{n \to 1} n \sum_{i=0}^{n} b_i$ of any square matrix $B$ is guaranteed if the spectral norm $\|B\|$ is smaller than 1 (see for instance [15]); but the spectral norm $\|B\|$ is equal to the absolute value of the largest eigenvalue of $B$ and all eigenvalues are bounded by any matrix norm $k_B k_p$:

$$\|B\| = j \cdot \max_j k_B k_p$$

One specific matrix norm is the column sum norm for $p = 1$:

$$k_B k_1 = \max_{i \in I} \sum_{j} \left| b_{ij} \right|$$
The column sum norm of matrix $d(E^i):A$ is in fact smaller than 1 because all columns of $A$ sum up to 1 and the left multiplication with $d(E^i)$ diminishes every column with a factor smaller than 1, at least if matrix $E$ does not contain any ones or zeros.

We can now safely state that
\[
\lim_{n! 1} X^i (d(E^i):A)^i = (I_i \ d(E^i):A)^i 1
\]
and therefore that
\[
U(S) = U(x_0) = u:d(E^+):A:(I_i \ d(E^i):A)^i 1:x_0
\]  
\tag{6.37}

The expected costs of strategy $S$ are computed similarly:

\[
K(x) = K(e) + K(a) + (E:A:x)^i K \left( \frac{d(E^i):A:x}{E:A:x} \right)
\]
working out into:

\[
K_1(x) = (1 + (E:A:x)^i) (K(e) + K(a)) + (E:A:d(E^i):A)^i K \left( \frac{d(E^i):A:d(E^i):A:x}{E:A:d(E^i):A:x} \right)
\]
and for step $n$:

\[
K_n(x) = (1 + \sum_{i=0}^{\infty} (E:A:d(E^i):A)^i x^i) (K(e) + K(a)) + (E:A:(d(E^i):A)^n x^i) K \left( \frac{(d(E^i):A)^{n+1} x}{E:A:(d(E^i):A)^n x} \right)
\]
now taking the limit for $n! 1$

\[
K(x_0) = \lim_{n! 1} K_n(x_0) = (1 + E:A \lim_{n! 1} \sum_{i=0}^{\infty} (d(E^i):A)^i x_0^i) (K(e) + K(a)) + \lim_{n! 1} (E:A:(d(E^i):A)^n x_0) K \left( \frac{(d(E^i):A)^{n+1} x_0}{E:A:(d(E^i):A)^n x_0} \right)
\]
which finally yields:

\[
K(S) = (1 + E:A:(I_i \ d(E^i):A)^i 1_{d^i} x_0) (K(e) + K(a))
\]  
\tag{6.38}

The formulae for the expected utility (6.37) and expected costs (6.38) distinctly resemble the equivalent formulae for the repeating test strategy above, especially if the expressions $d(E^+):A:(I_i \ d(E^i):A)^i 1$ and $E:A:(I_i \ d(E^i):A)^i 1$ are considered as matrices. This is no coincidence because it is possible to create a Markov decision network with a repeating test strategy that has the same expected costs and utility as this repeating action strategy.
When we rewrite 6.37 and 6.38 into

\[ U(S) = u:d(E^+):Q:x_0 \]
\[ K(S) = (1 + (E:Q)^i) x_0 (K(e) + K(a)) \]
\[ Q = A:(1 + d(E^+):A)^i \]

and define a new action \( b \) with probability matrix \( B = d(E^+):Q \) and a new experiment \( f \) with two outcomes and sensitivity matrix \( F \) such that \( F^i = (E:Q)^i \) then the repeating test strategy \( S^0 = \text{repeat}; f; \text{until}; r_f +; b \) will have the following expected utility and costs:

\[ U(S^0) = u:B:x_0 = u:d(E^+):Q:x_0 = U(S) \]
\[ K(S^0) = (1 + F^i x_0) K(f) + K(b) = (1 + (E:Q)^i x_0) (K(e) + K(a)) + 0 = K(S) \]

where we take the costs of \( b \) zero and the costs of \( f \) equal to the costs of \( a \) and \( e \) together.

So the effects of \( S = \text{repeat}; a; e; \text{until}; r_{ae}; i \) and \( S^0 = \text{repeat}; f; \text{until}; r_{f+}; b \) are exactly the same. When we view the combination \([a; e]\) as one single impure experiment that has some side-effect on the state of the world, this result shows that this impure experiment can be converted into a pure experiment \( f \) without side-effects and that all side-effects of the experiment can be collected in one final action \( b \).

6.3.3 Example 3: Action-Test Chain

We introduce a last example to illustrate how more complex infinite strategies lead to similar formulae for expected costs and utility. Suppose we have the following Markov decision network:

```
and every test \( e_i \) has two outcomes: \( r_{i+} \) and \( r_{i-} \). What would be the expected utility and expected costs of the repetitive strategy \( S = \text{repeat}, a_0, e_0, \text{until}, r_{0+}, b_0, \text{repeat}, a_1, e_1, \text{until}, r_{1+}, b_1, ..., \text{repeat}, a_m, e_m, \text{until}, r_{m+}, b_m, i? ```
We give the solution without proof. The expected utility is:

\[ U(S) = u \left( \sum_{i=0}^{\infty} B_{m_i \cdot i} \cdot d(E_{m_i}) \cdot Q_{m_i \cdot i} \right) x_0 \]  

(6.39)

and the expected costs are:

\[ K(S) = \sum_{i=0}^{\infty} (K(a_i) + K(b_i) + K(c_i)) + \frac{1}{E_i : Q_i} \sum_{j=0}^{\infty} B_{ij \cdot j} \cdot d(E_{i \cdot j}) \cdot Q_{ij \cdot i} \cdot A : x_0 \]

where

\[ Q_i = A_i \cdot (E \cdot d(E) \cdot A_i)^i \]

One can observe that the global appearance of these formulae again resembles the equivalent formulae of the repeating test strategy. When we take a transition matrix

\[ D = \sum_{i=0}^{\infty} B_{m_i \cdot i} \cdot d(E_{m_i}) \cdot Q_{m_i \cdot i} \]

and a sensitivity matrix G such that

\[ \frac{G^i}{G^{i+1}} = \sum_{i=0}^{\infty} (K(a_i) + K(e_i)) (E_i : Q_i) \sum_{j=0}^{\infty} B_{ij \cdot j} \cdot d(E_{ij \cdot j}) \cdot Q_{ij \cdot i} \]

then we can again create a repeating test strategy with the same expected costs and utility as this action-test-chain strategy.

### 6.4 Conclusion

In this chapter we developed the concept of iterative infinite episodes and repetitive infinite strategies in Markov decision networks. These concepts were formalized with the use of two grammars.

The example computations on iterative episodes showed that only under strict limitations the expected costs and utility of such episodes can be computed. The example computations for repetitive strategies however showed that expected costs and utilities of these infinite strategies are computable. During the example computations we derived some results that could give some clues on how computations on repetitive strategies in general should be performed. Further research is needed to derive such general computation rules in order to build an algorithm that accepts a repetitive strategy and produces the expected costs and utility.

If such algorithm can be constructed, then the next step would be to develop an algorithm that constructs or finds a repetitive strategy which maximizes the expected pay-off value.
Chapter 7

References


