Optimal Agents

Nicholas James Hay

under the supervision of

Professor Cristian S. Calude

A thesis submitted in partial fulfilment of the requirements
for the degree of Bachelor of Science (Honours) in Computer Science,
The University of Auckland, 2005.

Abstract

Artificial intelligence can be seen as the study of agents which achieve what we want, an agent being an interactive system with an input/output interface. This can be mathematically modelled through decision theory, using utility functions to capture what we want, and with the expected utility of an agent measuring how well it achieves that. Optimal agents, those that maximally achieve what we want, have maximal expected utility.

We describe this theory of optimal agents. We detail how these optimal agents behave, giving an explicit formula for their actions. We discuss applications of this theory, in particular Marcus Hutter's AIXI [Hut04]. Finally, we suggest possible directions for future research extending Hutter's work on universal AI.

Contents

1 Introduction **1**

1.1 Relation to previous work 1

1.2 Acknowledgements 2

2 Decision theory **3**

2.1 Predicted effects 3

2.1.1 The effects of chess 3

2.1.2 Probability distributions 4

2.1.3 Conditional probability distributions 4

2.2 The best effects: a decision maker's preference 5

3 Agents **7**

3.1 String notation 8

3.2 Example agents 8

3.3 Agents in the real world 10

4 Choosing agents **13**

4.1 Predicted effect of an agent 13

4.2 Optimal agents have maximal expected utility 17

4.3 Expectimax agents are optimal 18

5 Applications **25**

5.1 Hutter's AIμ and AIXI 25

5.2 Learning what we want 27

6 Conclusion **29**

A Probability Theory **31**

A.1 Logic 31

A.2 Probability as uncertain logic 32

A.3 Variables and probability distributions 33

A.4 Marginalisation 34

A.5 Bayes' theorem 34

B Expectimax Agents are Optimal Agents **35**

B.1 Possible futures 35

B.2 Identities of the expected utility of an agent 36

B.3 Expectimax agents are optimal agents 38

C Odds and Ends **45**

C.1 Extended agents are equivalent to agents 45

C.2 Nondeterministic agents are deterministic 45

Chapter 1

Introduction

Our approach to artificial intelligence (AI) follows [Rus97]:

"[O]ne can define AI as the problem of designing systems that *do the right thing*."

Intelligence, from this perspective, is the property which makes human brains particularly successful at achieving good things (when it's applied towards that end!). However it's the good things that matter; intelligence, in AIs, is ultimately only a means to our ends. It's for this reason we mathematically define systems that achieve good things, rather than intelligent systems.

We focus on interactive systems called agents. Agents have a fixed input stream ("observations") and output stream ("actions") but are otherwise isolated from their environment. The prototypical example is a robot, with sensors as inputs and motor actions as outputs. We focus on these systems as they can be modelled simply: as functions mapping inputs to outputs.

We frame AI as a decision theoretical problem: how to best *decide* which agent to implement in order to achieve what we want (the right thing; good things). This framework requires a mathematical description of what we want, of the agents we choose from, and our knowledge of their effects.

We first give an introduction to decision theory, then a description of agents, finally applying decision theory to the problem of choosing an agent. Decision theory will give us the tools (i.e. expected utility) to formally define the optimal agent. We then describe the how these agents behave, giving a worked example. We end with some applications, including Marcus Hutter's AIXI and agents that learn what we want.

1.1 Relation to previous work

[RN03] describes the agent centered approach to AI. A similar definition of optimal agents to ours, as agent functions which maximise an expected utility, can be found in [Rus97]. Agents that maximise expected utility are common in the economics literature, e.g. see their use in [Han02]. [HBH88] offers a discussion of decision theoretical approaches to artificial intelligence.

The explicit form for the optimal agent can be found in [Hut04], although Hutter uses rewards [KLM96],[SB98] rather than general utility functions. Our proof of equivalence is original, although the result is not. Hutter's work inspired our reinvention of optimality by general utility functions; his AIXI model is discussed later.

1.2 Acknowledgements

Particular thanks to my supervisor Cristian Calude, for support and advice, and Marcus Hutter, who's work on universal AI [Hut04] inspired this work.

Chapter 2

Decision theory

Decisions are made by choosing the action with the best predicted effect. Consider a chess player deciding between strategies. The main effect of implementing a chess strategy is whether it results in a win, a loss, or a draw. Although there are other effects, this is the only one the chess player directly cares for (we suppose). The chess player will select the strategy with the best effect, for example the strategy most likely to win.

Decision theory offers a mathematical model for the above process: making a choice. We break this down into a method for predicting the effect of an action, and a method for judging which of a set of effects are best. The former represents the factual beliefs of the decision maker, the latter their preferences or desires. We develop a mathematical model of each in turn. We do not attempt to exactly model human decision making, but describe a common idealisation.

See [Ber93] for a more detailed introduction to the decision theory we describe here (Bayesian decision theory).

2.1 Predicted effects

2.1.1 The effects of chess

There are three outcomes of a chess game: {`loss`, `draw`, `win`}. Although a particular chess strategy will result in exactly one of these outcomes, the chess player can't generally be *certain* which one it will be. Uncertainties arise through lack of information about the other player's strategy, or the inability to fully compute the consequences of that information. As a result, we need to handle predictions involving uncertainty e.g. "with this strategy we will probably lose, but we might draw".

Probability theory offers a mathematical treatment of uncertainty (see chapter A). Each element of a set of mutually exclusive and exhaustive statements (i.e. a set such that exactly one statement is true), such as {`loss`, `draw`, `win`}, is assigned a real number in $[0, 1]$ which together sum to 1. The larger the real number ("probability"), the more the event is expected to occur.

For example, suppose we have exactly three strategies a, b, c. By some predictive mechanism, a is predicted to be equally likely to lose, draw, or win. b definitely won't draw, but could equally win or lose. c is predicted to definitely draw. This is represented by the following probabilities:

i	$p_i(\texttt{loss})$	$p_i(\texttt{draw})$	$p_i(\texttt{win})$
a	$1/3$	$1/3$	$1/3$
b	$1/2$	0	$1/2$
c	0	1	0

where $p_a(\texttt{loss})$, for instance, is the probability that strategy a will result in a `loss`.

(Caution: probability theory is widely used to model randomness, with probabilities being interpreted as relative frequencies of events. We use it, instead, to model the "subjective" plausibility of or belief in a proposition. These probabilities may or may not be derived from empirical frequencies. See [JB03], [Jay86], and chapter A.)

2.1.2 Probability distributions

In general, we have a set R of mutually exclusive and exhaustive outcomes. This may be the set of values of a variable within reality (e.g. the outcome of a chess game, the temperature in a room at a given time, or the energy a nuclear reactor generates over a particular day). These are all possible *relevant* consequences of our choice. To each outcome $r \in R$ we assign a real number $p(r) \in [0, 1]$, together summing to 1

$$\sum_r p(r) = 1$$

Thus, we represent the predicted effect of a choice by a function

$$p\colon R \to [0, 1]$$

over outcomes called a probability distribution. Let $\Delta(R)$ be the set of all such probability distributions.

In the following "outcome" will denote the actual consequence of a choice or action within the world. "Predicted effect", "effect", or "uncertain outcome" will denote a probability distribution over outcomes. The former is what actually happens in reality, the latter is our knowledge of it.

2.1.3 Conditional probability distributions

We have identified actions with the probability distributions describing their effects, but this hides nontrivial inference. In practice actions will be understood by their proximal effects, with the ultimate effect of this on the outcome $r \in R$ being derived by some predictive mechanism. For instance, chess strategies (actions) are understood by which chess pieces to move when (proximal effects), the ultimate effect being whether we win, lose, or draw. This predicted ultimate effect may be computed by simulating chess games.

We do not analyse the details of this predictive process, only the function it implements. Denoting the set of actions by A, we encapsulate this process with the notation:

$$P(r|a)$$

for the probability of outcome $r \in R$ given that action $a \in A$ is taken. For each action a, $p_a : r \mapsto P(r|a)$ is the distribution capturing its predicted effect. P is called a conditional probability distribution, for it gives the probability of each outcome *conditional* on the fact that a certain action a is taken.

In our example above, the actions are the choice of a particular chess strategy: $A = \{a, b, c\}$. The set of outcomes is $R = \{\texttt{loss}, \texttt{draw}, \texttt{win}\}$. The conditional probability distribution $P(r|a)$ is the table, with a the row, r the column, and $P(r|a)$ the probability at that location.

2.2 The best effects: a decision maker's preference

We model a decision maker's judgement of which effect is best through utility functions. Utility functions capture tradeoffs between uncertain outcomes. For example, different chess players will have different preferences between a 50/50 chance of winning or losing and a certain chance of drawing. Some would consider these possibilities equally good, others would prefer to certainly draw, still others would take the 50/50 chance. Different utility functions can capture these kinds of different preferences.

A utility function

$$U : R \to \mathbb{R}$$

is a function mapping each possible outcome $r \in R$ to a numerical value $U(r)$. The greater the value, the more the decision maker desires the outcome. In the chess example this assigns a numerical weight to each outcome: $\{\texttt{loss}, \texttt{draw}, \texttt{win}\}$.

Recall from the previous chapter the predicted effect of an action $a \in A$ is denoted by the conditional distribution $P(r|a)$. This gives the predicted probability of each outcome $r \in R$. With a utility function we can compute the expected utility of an action a:

$$E[U|a] = \sum_{r \in R} P(r|a)U(r)$$

This is the average utility, the utility of each outcome weighted by its predicted probability. The idea is the more probability assigned to valuable outcomes the larger the expected utility gets; the less probability the lower the expected utility. The best actions are those with the largest expected utility.

Recall our three chess strategies a, b, c. We show the expected utility of each of these strategies for three different utility functions. These utility functions describe chess players with different preferences. All agree that `win` is better than `draw` which is in turn better than `loss`, but disagree about how much better they are. The first, U_1 considers `draw` to be equivalent to a 50/50 chance of `win` or `loss`. U_2 considers `draw` to be worse than a 50/50 chance, U_3 considers `draw` better.

The following table shows the expected utilities:

i	$P(\texttt{loss}\|i)$	$P(\texttt{draw}\|i)$	$P(\texttt{win}\|i)$	$E[U_1\|i]$	$E[U_2\|i]$	$E[U_3\|i]$
a	$1/3$	$1/3$	$1/3$	0	$-1/6$	$1/6$
b	$1/2$	0	$1/2$	0	0	0
c	0	1	0	0	$-1/2$	$1/2$
U_1	-1	0	1			
U_2	-1	$-1/2$	1			
U_3	-1	$1/2$	1			

Each utility function orders the effects a, b, c by their expected utility, where we write $a \prec b$ if $E[U|a] < E[U|b]$, $a \approx b$ if $E[U|a] = E[U|b]$:

$$\begin{aligned} U_1 &: a \approx b \approx c \\ U_2 &: c \prec a \prec b \\ U_3 &: b \prec a \prec c \end{aligned}$$

The different values given to `draw` yield the different preferences between otherwise identical predictions.

Actions which have maximal expected utility will be termed *optimal*. With A the set of possible actions, the subset of optimal actions is denoted

$$\operatorname*{argmax}_{a \in A} E[U|a] \subseteq A$$

where $\operatorname{argmax}_{x \in X} f(x)$ is the subset of X where f attains its maximum on X

$$\operatorname*{argmax}_{x \in X} f(x) = \{x_0 \in X : f(x_0) = \max_{x \in X} f(x)\}$$

If there is a unique optimal action this will be a singleton set. Although optimal actions are entirely equivalent to the decision maker, for theoretical clarity we avoid making an arbitary choice by selecting them all.

Chapter 3

Agents

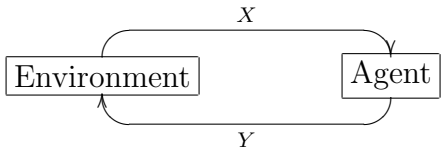
We define an *agent* to be a system isolated from its *environment* except for a fixed input/output interface. Intuitively, these systems input percepts or observations and output actions. The environment consists of everything that isn't the agent. This definition is independent of "intelligence": agents can act in stupid ways, and not all intelligent systems are agents.

(Unlike other definitions of agents (e.g. [**?**]) we add no further restrictions on the system e.g. that it acts as if it had purpose. We will select agents based on how well they achieve our goals. Any restrictions are at best unnecessary and at worst risk removing agents that perform well. "Agent" may be a misnomer for the general class of interactive systems we investigate here.)

For example, a chess strategy describes an agent in two senses:

1. A complete chess strategy can be presented as an agent which inputs the opponent's chess moves, and outputs the recommended move.

2. A human using that chess strategy describes a chess playing agent. Input, observations of the chess board. Output, chess movements.



The regulated interaction between agent and environment allows us to abstract away from the internal details of both. In particular, since agent and environment interact only through the input/output (I/O) interface, we can characterise an agent by which outputs follow which inputs. Assuming discrete time steps $T = 1, 2, \ldots$ and with X/Y denoting the fixed set of possible inputs/outputs at each time step, the complete history of interaction can be represented as a pair of sequences:

$$x_1 x_2 \ldots x_N \in X^N, \qquad y_1 y_2 \ldots y_N \in Y^N$$

Typically we view this in an interleaved fashion: the agent receives an input, then takes an action (or vice versa). Russell and Norvig [RN03] has inputs (percepts)

before outputs (actions), whereas Hutter [Hut04] has outputs before inputs. This only makes a difference at the very first and lasts time step – whether the agent's first action is taken with or without input, and whether its last is followed by an input or not. We follow Hutter's convention.

Further assuming the agent to be deterministic (see chapter C for why nondeterminism is unnecessary) and in a fixed state at $T = 1$, we can characterise its behaviour by a function mapping a finite sequence of inputs to an output. This *agent function* maps the history of inputs, past and present, to the present output. Agent functions have the form:

$$a\colon X^* \to Y$$

where X^* is the set of finite sequences of elements from X.

This function summarises the behaviour of the system, ignoring its internal structure. $a(\epsilon)$ is the first output of the system, $a(x_1)$ is the second output of the system assuming x_1 was the first input, $a(x_1x_2)$ is the third output assuming x_1 and x_2 were the first and second inputs, etc.

Interactive systems used in theories of interactive computation, such as Persistent Turing Machines [Gol04] and Chronological Turing Machines [Hut04], are subclasses of agents. These theories can be used to define computable agents. For tractable agents, see boundedly optimal agents in [Rus97].

3.1 String notation

If we have a string over X:

$$x = x_1x_2 \ldots x_n$$

where each $x_i \in X$, i.e. a finite sequence of elements from X, we use the following notation to denote substrings:

$$\begin{aligned} x_{i:j} &= x_i \ldots x_j \\ x_{<i} &= x_1 \ldots x_{i-1} \\ x_{\leq i} &= x_1 \ldots x_i \end{aligned}$$

We will use an interleaved notation to describe histories

$$\begin{aligned} y_1x_1 \ldots y_{i-1}x_{i-1}y_i &= yx_1 \ldots yx_{i-1}y_i = yx_{1:i-1}y_i = yx_{<i}y_i \\ y_1x_1 \ldots y_{i-1}x_{i-1}y_ix_i &= yx_1 \ldots yx_{i-1}yx_i = yx_{1:i} \end{aligned}$$

y_1 is the first output, x_1 the first input, y_2 the second output etc. yx_1 is the first output/input pair, yx_2 the second, etc. Histories will either end in an input from the environment as in $yx_{1:i}$, or an output from the agent as in $yx_{<i}y_i$.

3.2 Example agents

Example 3.1. One of the simplest agents, a thermostat regulates the temperature of a room. X, the agent's set of inputs, is the (finite) set of temperature readings

from the thermometer, for example $\{0.0, 0.1, 0.2, \ldots, 99.9, 100.0\}$. Y, the agent's set of outputs, is the set $\{(0,0),(0,1),(1,0),(1,1)\}$ with the first element of the pair controlling whether the cooler is on (1 for on, 0 for off), the second element controlling whether the heater is on. The agent function for this simple thermostat is

$$a(x_1 \ldots x_i) = \begin{cases} (0,1) \text{ if } x_i < T - \epsilon \\ (1,0) \text{ if } x_i > T + \epsilon \\ (0,0) \text{ otherwise} \end{cases}$$

$x_1 \ldots x_i$ is the sequence of temperature readings, where x_1 is the first and x_i is the most recent. T is the target temperature, and ϵ is a small tolerance to avoid oscillation and damage to the hardware, and because we really have a target temperature *range* $[T - \epsilon, T + \epsilon]$. Note we are treating the inputs x_i as numbers, leaving implicit the decoding function $d\colon X \to \mathbb{R}$.

The agent function checks to see if the most recent temperature reading, x_i, is too hot or too cold. If so, it activates the cooler or heater respectively. If it's within the target temperature range it turns both heater and cooler off.

More sophisticated agents could predict temperature changes, finely control the air conditioning hardware, or achieve more complex temperature goals. This sophistication entails larger input and outputs sets X and Y, and more complex agent functions a. □

Example 3.2. Robotic vacuum cleaners (after [RN03]) move around a room sucking up dirt. We define the agent to be the *software* controlling the robot (more precisely, the internal computer running the software). The robot body itself is a (known) part of the environment the agent is within, the agent's interface with the environment connects to internal parts of this robot body.

The agent's input comes from a variety of instruments: cameras for obstacle detection, internal sensors (battery charge, temperature, damage indicators, free storage space, etc), user controls, traction and acceleration sensors, dirt detectors, etc. As a result, its input is a tuple of k subinputs from different sensors, for example $\langle i, b, \ldots, d\rangle$ where i is a frame from the camera, b is the battery charge left, and d is a measure of how dirty the carpet is. So $X = X_1 \times X_2 \times \ldots \times X_k$ where X_1 is the set of video frames, X_2 is the set of battery charges, etc.

Its output is likewise broken into components, making Y also a cartesian product of sets Y_i. These suboutputs would include power levels for the driving and fan motors, sound output for user interaction, and so forth. The description of the interface is necessarily low level as that is how the agent actually interacts with its environment (but see below). Achievment of higher level goals and action such as moving from A to B or, indeed, cleaning the room, require coordinated sequences of outputs. The output can thus be thought of as *atomic* or low-level actions.

Finally, its agent function describes the behaviour of the cleaner's software. It describes what the software would do given any possible sequence of inputs. This software will guide the robot around the room (presumably) cleaning it. □

Example 3.3. The Human brain can be seen as an agent. Imagine a surface surrounding the brain isolating it from the rest of the body. This cuts across the cranial nerves and spinal cord, the brain's major communication pathway with the

outside world. The agent's interface is the state of these nerves along the surrounding surface, with X the state of all the incoming neurons in this slice at a given time instant, and Y the state of the outgoing neurons. The agent function describes exactly how the brain reacts to any possible sequence of neural inputs, the mapping between incoming and outgoing neurons.

This is necessarily a rough view of things, since there are interactions with the environment outside neural signals (e.g. the blood stream, blunt trauma). (The same can be said for any real world agent, to a greater or lesser extent.) □

3.3 Agents in the real world

As the above examples show, we can model real world systems as agents. This modelling process has a number of complications:

1. *Interpreting a system as agents in different ways.* In the robotic vacuum cleaner example we have a central computer connected to a number of sensors and peripheral motor controllers themselves small computers. We can define the agent and its interface in different ways by including more of less of the vacuum cleaner: are the peripheral controllers part of the agent or part of the environment? Further choices involve using different abstractions of the world in which to set the division of agent and environment. Perhaps we could even interpret the entire robot as an agent, with dirt and light as input...

2. *Interactions outside the declared interface.* In the vacuum cleaner, the agent function is implemented on a computer. Since these are electrical devices, their internal state leaks out on electromagnetic radiation, however faint. This means different programs implementing the same agent function may act differently: different code will leak different information. Conversely, the computer can be be manipulated by external radiation, potentially making the output not simply a function of the input (i.e. the output is different, given fixed input, if the external radiation is different). More drastic undeclared interactions include the environment directly damaging the robot's computer. These interactions make agent functions an approximation of reality.

3. *Incomplete control over agent function.* We assume the ability to arbitarily select the agent function. For our purposes this is one of the key differences between agent and environment. (Another is that we don't care directly about the state of the agent, only how it affects the state of the environment.) In reality we have additional constraints such as computability and tractability. In general, we are constrained by the need for the agent to run in real time and be implemented by present-day humans.

4. *Intuitive understanding of inputs and outputs.* In the above, we describe agent/environment interaction on an intuitive level. For instance, we describe inputs as video frames from cameras, outputs as setting the power of motors. Even setting aside the difficulties of defining a precise and stable interface, these descriptions are approximate. That part of the input is produced by a

video camera is an assumption about the environment that need not hold. In particular, agent actions could invalidate this assumption e.g. by disconnecting the video feed.

Aside from defining the size of the set of inputs and outputs, any description of what an input or output 'means' makes assumptions about how the environment is structured. Where and at what level these descriptions are made is a matter of human psychology, but they're not often at the very edge of the causal chain. In the robotic vacuum cleaner example, we think of the input as a video frame (ignoring the other inputs) which we think of as being caused by the video camera. But further causes extend both ways: closer to the agent it's generated by electrical cabling connected to the camera, closer to the environment it's caused by photons propagating through space.

These complications highlight the difference between mathematically well defined problems, however complex, and the real world. By defining all elements involved we avoid complications caused by the partial control and knowledge we otherwise have. However, these complications need to be addressed when applying theory to systems we can actually build.

Chapter 4

Choosing agents

We apply the decision theory described in chapter 2 to the problem of choosing an agent from chapter 3. The optimal agent is the one with maximal expected utility. Successful artificial intelligences are candidates for optimal agents, where they use their intelligence to achieve the utility function we specify.

Which agent functions are optimal will depend both on the prior knowledge of the environment, used to predict each agent function's effect, and on the utility function, used to specify which effects we prefer. These are free variables of the theory described here: different choices for them will lead to different optimal agents. If we know more about the environment we can use this knowledge to design specialised agents. If we know less we must design more general agents. Similarly, with different utility functions different agents are optimal. If we want to minimise pollution, one variety of agent is optimal; if we want to maximise pollution a different variety of agent is optimal.

For theoretical convenience we assume agent and environment interact for exactly N steps. Given this, we redefine the agent function from chapter 3:

Definition 4.1. An *agent function* is a map:

$$a\colon X^{<N} \to Y$$

where

$$X^{<N} = \bigcup_{i=0}^{N-1} X^i$$

is the set of all sequences of inputs X of length less than N, and X^i is the standard cartesian product $\underbrace{X \times \ldots \times X}_{i}$. □

Instead of X^*, all finite sequences, as the domain of an agent function we have $X^{<N}$, all sequences of length less than N.

4.1 Predicted effect of an agent

Consider again the problem of choosing between chess strategies. Following chapter 2 we will measure the performance of a chess strategy via its expected utility. This will

be computed from a probability distribution over $\{\texttt{loss}, \texttt{draw}, \texttt{win}\}$ (its predicted effect), and a utility function over outcomes. Following chapter 3 we will model chess strategies by agent (functions).

This chapter will describe how to predict the effect of an agent. This amounts to describing how

$$P(r|a)$$

(the probability that outcome r occurs given that we have implemented agent a) depends on the agent function a. In the chess example, this is how the probability for a loss/draw/win depends on the chess strategy.

We will discover that the knowledge of outcome of an agent

$$P(r|a)$$

reduces to the knowledge of the outcome of a complete interaction history

$$P(r|yx_{1:N})$$

and knowledge of how the environment reacts

$$P(x_i|yx_{1:i-1}y_i)$$

These two distributions, unlike the first, can be freely choosen for different problems; they parameterise the knowledge about an agents' effect on the outcome.

We use two assumptions:

1. The agent a can only affect the environment by way of the input/output interface. This has two consequences.

   First, given the complete history the outcome r is independent of which agent a is implemented. Once we know the complete history of interaction $yx_{1:N}$, knowing the exact agent function a tells us nothing more about which outcome r will occur. We express this probabilistically by the equation:

   $$P(r|yx_{1:N}, a) = P(r|yx_{1:N})$$

   The probability of outcome r given that we know history $yx_{1:N}$ occurs and agent a is implemented, is the same as the probability of outcome r given that we just know history $yx_{1:N}$ occurs. The agent function doesn't tell us anything more.

   Second, once we know the immediate past $yx_{1:i-1}y_i$, the agent function a tells us nothing more about how the environment will react:

   $$P(x_i|yx_{1:i-1}y_i, a) = P(x_i|yx_{1:i-1}y_i)$$

   This is because the environment receives no agent outputs after y_i but before x_i.

2. The agent is perfectly implemented. That is, once we know the past inputs $x_{1:i-1}$ we know what the next action will be:

   $$y_i = a(x_{1:i-1})$$

   This means

   $$P(yx_{1:N}|a)$$

   can be recursively defined just in terms of knowledge of how the environment reacts, since a completely describes how the agent reacts.

We expand $P(r|a)$ to include the complete history $yx_{1:N}$:

$$\begin{aligned} P(r|a) &= \sum_{yx_{1:N}} P(r, yx_{1:N}|a) \\ &= \sum_{yx_{1:N}} P(r|yx_{1:N}, a)P(yx_{1:N}|a) \end{aligned}$$

Both steps are instances of probability theory theorems: marginalisation, then the product rule. (See chapter A for a brief review of these theorems.) This says the probability of outcome r given that agent a is implemented, is the probability that for some complete history $yx_{1:N}$:

1. Complete history $yx_{1:N}$ occurs, given that we know agent a is implemented.
2. Outcome r occurs, given that we know complete history $yx_{1:N}$ occurs and agent a is implemented.

Recall that our assumption that the agent interacts with the environment only through the input/output streams implies

$$P(r|yx_{1:N}, a) = P(r|yx_{1:N})$$

If we know the history of interaction between agent and environment $yx_{1:N}$, additionally knowing the exact agent a tells us no more about the outcome r. So

$$P(r|a) = \sum_{yx_{1:N}} P(r|yx_{1:N})P(yx_{1:N}|a) \tag{4.1}$$

The right hand side is the probability that for some complete history $yx_{1:N}$, the complete history occurs given that agent a is implemented, and outcome r occurs given that the complete history $yx_{1:N}$ occurs.

We recursively expand $P(yx_{1:i}|a)$ for $0 \leq i \leq N$, where $yx_{1:i}$ is the statement that the history starts with outputs/inputs $yx_{1:i}$:

$$\begin{aligned} P(\epsilon|a) &= 1 \\ P(yx_{1:i}|a) &= P(yx_{1:i-1}, y_i, x_i|a) \\ &= P(yx_{1:i-1}|a)P(y_i|yx_{1:i-1}, a)P(x_i|yx_{1:i-1}y_i, a) \\ &= P(yx_{1:i-1}|a)[y_i = a(x_{1:i-1})]P(x_i|yx_{1:i-1}y_i) \end{aligned} \tag{4.2}$$

where

$$[X] = \begin{cases} 0 \text{ if } X \text{ is false} \\ 1 \text{ if } X \text{ is true} \end{cases}$$

The first equality is trivial: any complete history starts with the empty string ϵ. In the second we apply the product rule twice. For the third:

1. Once we know the immediate past $yx_{1:i-1}$, the next output y_i is completely determined by the agent function a:

   $$P(y_i|yx_{1:i-1}, a) = [y_i = a(x_{1:i-1})]$$

   i.e. $P(y_i|yx_{1:i-1}, a)$ is 1 exactly when y_i has the correct value $a(x_{1:i-1})$, 0 otherwise.

   As an aside, this suggests two directions we might generalise agents. First we might allow nondeterministic agents π which have a probability distribution over possible future actions where

   $$\pi(y_i|x_{1:i-1}) = P(y_i|yx_{1:i-1}, \pi)$$

   is the probability that action y_i is (randomly) chosen given input $x_{1:i-1}$. This reduces to the deterministic case when:

   $$\pi(y_i|x_{1:i-1}) = [y_i = a(x_{1:i-1})]$$

   for some agent function a.

   Second, we could allow agent functions (i.e. extended agent functions; see definition 4.7 in chapter 4.3) that depend on the past actions as well as past inputs:

   $$P(y_i|yx_{1:i-1}, a) = [y_i = a(yx_{1:i-1})]$$

   Neither case, nor a combination of both, results in agents with higher expected utilities (see chapter C) so we stick with regular agent functions (although we use extended agent functions in chapter B).

2. Once we know the immediate past $yx_{1:i-1}y_i$, the agent function a tells us nothing more about how the environment will react:

   $$P(x_i|yx_{1:i-1}y_i, a) = P(x_i|yx_{1:i-1}y_i)$$

We will later use the above identity to give a explicit formula for the actions optimal agents take (in theorem B.5).

As a result of the above, the probability distribution $P(r|a)$ is specified by:

1. $P(r|yx_{1:N})$. This infers the outcome r from the complete history.

2. For all $1 \leq i \leq N$, $P(x_i|yx_{1:i-1}y_i)$. This infers the next input x_i from the past history $yx_{1:i-1}y_i$.

These two distributions can be chosen arbitarily, to represent different knowledge about how the agent influences the outcome. Once they are chosen $P(r|a)$ is uniquely defined. (See equations 4.1 and 4.2 we derived above.)

Example 4.2. Consider again our chess playing agent. There are different ways to interface this agent with the chess game. We could use a robot, with input X video frames and output Y motor commands to a robot arm. Or, we could directly encode the pieces of the chess board in a grid as input X, with descriptions of chess moves (e.g. `Nf3`, `fxe5`) as output Y. Or, we could have move descriptions as both input X and output Y. Each method yields different input and output sets X and Y, along with different probability distributions. We'll assume the second option.

We will suppose a single game is being played, with a time limit of N steps. If the game isn't completed in this time limit, it's a draw. If it's completed before, we pad out the inputs e.g. keep on inputing the final chess board setup until N steps have occurred. Although one can extend the theory to handle variable time limits, we do not do so here.

In chapter 2.2 we discussed different utility functions U over

$$R = \{\texttt{loss}, \texttt{draw}, \texttt{win}\}$$

We use the same utility functions and variable (i.e. set of outcomes R) here. Now

$$P(r|yx_{1:N})$$

infers whether we've won or lost from the entire sequence of chess board inputs and move outputs. Given our representation we can be sure what the outcome of the game is: look to see if the game's finished within time, and if so who's in check mate.

Finally,

$$P(x_i|yx_{1:i-1}y_i)$$

predicts the next move x_i of the opponent player (more properly, the chess board state after their next move). $yx_{1:i-1}y_i$ tells us everything that's happened so far, up to our last move. This probability distribution will depend on what knowledge we have of the other player. We might simulate the opponent's strategy or brain to determine what move they are likely to next make. □

4.2 Optimal agents have maximal expected utility

Finally, combining all previous chapters, we have the expected utility of an agent:

$$E[U|a] = \sum_{r \in R} U(r)P(r|a)$$

which we can use to define optimal agents. This measures how well agent a is expected to perform, given what we know about the environment and what we want the agent to achieve.

Definition 4.3. An *optimal agent* a^* is one with maximal expected utility:

$$E[U|a^*] = \max_a E[U|a]$$

equivalently

$$a^* \in \operatorname*{argmax}_a E[U|a]$$

where $\operatorname{argmax}_a E[U|a]$, the set of arguments a that maximise $E[U|a]$, denotes the set of all optimal agents. □

In the previous chapter we described the probability distributions needed to specify $P(r|a)$. With the addition of the utility function $U(r)$, we have all the ingredients necessary to define optimal agents i.e. everything necessary to evaluate $E[U|a]$:

1. X, Y, N. The set of inputs, the set of outputs, and the total number of time steps agent and environment interact.

2. R, $U(r)$, $P(r|yx_{1:N})$. The set of outcomes, the utility function describing how we desire outcomes, and the distribution capturing what we can predict about the outcome from the complete interaction history.

3. $P(x_i|yx_{1:i-1}y_i)$. What we know about the agent's next input (generated by the environment) given the past.

The informal use of "we" in "...how we desire outcomes" and "what we know about..." above can be interpreted in two senses:

1. The knowledge we, as humans, really have of the environment; what we truly desire of the outcome. The major problem here is our knowledge and desires are extremely complex and not explicitly known to us. Furthermore, we want agents with better knowledge of the environment, and perhaps "better" desires, than us.

2. The knowledge and desires we want, or presume, the agent to have. One must be careful to realise "knowledge" and "desire" are short for probability distributions and utility functions, respectively. When reasoning about agents one should use the mathematics of the latter, rather than our intuitions of the former; agents need act nothing like humans.

See also examples 4.2, 4.9, and chapter 5.

4.3 Expectimax agents are optimal

We give an explicit characterisation of the optimal agent functions implicitly defined in the previous chapter. This will make use of the structure within $P(r|a)$ we previously extracted (i.e. equations 4.1 and 4.2).

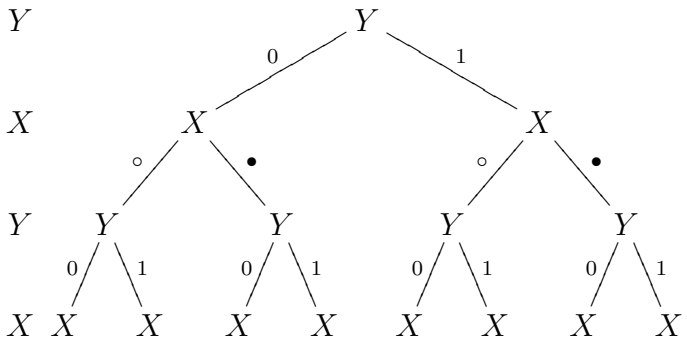
We sketch a brute-force algorithm to compute an optimal agent's action. At each time step i the algorithm computes the expected utility of each available *action*

$y_i \in Y$ by evaluating a function over the tree of all possible futures. It then selects the action with maximal expected utility. This algorithm is similar to the minimax algorithm of game theory [RN03], and is exactly the algorithm for computing AIXI generalised to arbitary utility functions and probability distributions [Hut04].

Suppose we have an input set $X = \{\circ, \bullet\}$ and an output set $Y = \{0, 1\}$. At any point in time i, our past is a string

$$yx_{<i} = y_1 x_1 \dots y_{i-1} x_{i-1}$$

The set of futures extendeding this past form a tree:



which can be extended downwards until we reach the leaves. The root is labelled with $yx_{<i}$, the next level has the futures after one output $yx_{<i}0$ and $yx_{<i}1$, the second after another input $yx_{<i}0\circ$, etc. Nodes alternate being being output nodes Y and input nodes X. The Y nodes represent choices for the agent, the X nodes possible inputs from the environment. The leaves of these trees are labelled by strings of the form $yx_{1:N}$: complete interaction histories.

The immediate choice faced by the agent is which child of the root Y node to pick. As each edge leading to a child is labelled by a possible output (either 0 or 1), this is equivalent to deciding the next output.

We define the valuation function over this tree of possible histories used by the agent. This is recursively defined from leaf to root. This function assigns a number to each node, which will turn out to the maximum expected utility any agent could achieve from that point in time (see chapter B). The agent will pick the Y node with the largest value (i.e. largest expected utility) to be the next output.

Definition 4.4. Recursively define the following valuation function $V(h)$ over histories:

$$\begin{aligned} V(yx_{1:N}) &= \sum_{r \in R} U(r) P(r|yx_{1:N}) \\ V(yx_{<i}y_i) &= \sum_{x_i \in X} P(x_i|yx_{<i}y_i) V(yx_{<i}yx_i) \\ V(yx_{<i}) &= \max_{y_i \in Y} V(yx_{<i}y_i) \end{aligned}$$

□

$V(yx_{1:N})$ is the value of a complete history of interaction. A complete history tells us as much as we'll ever know about the outcome r, so this value is simply the

expected utility given that history

$$V(yx_{1:N}) = \sum_r U(r)P(r|yx_{1:N}) = E[U|yx_{1:N}]$$

$V(yx_{<i}y_i)$ is the value of a X-node: it is the expected value of the X nodes below it, weighted by the probability that that X node will be selected by the environment i.e. the environment will send that particular input to the agent.

$V(yx_{<i})$, the value of an Y node, assumes that in the future the agent picks a y_i node with maximal value (expected utility) $V(yx_{<i}y_i)$.

Thus, an expectimax agent:

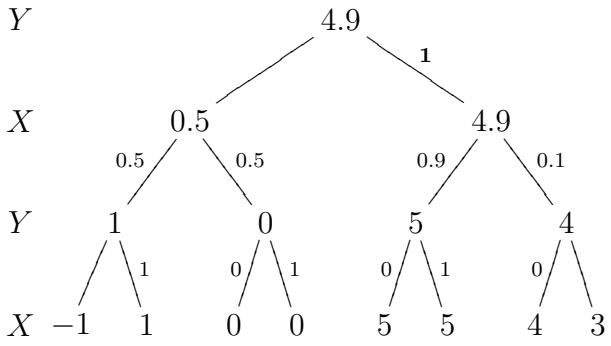
Definition 4.5. (Expectimax agent) An expectimax agent a_0 is one that for all histories $h \in YX^{<N}$ takes an action maximising the valuation $V(h)$ (definition 4.4):

$$\forall h \in YX^{<N}\ a_0(h) \in \operatorname*{argmax}_y V(hy)$$

□

An expectimax agent, given any history $h = yx_{1:i} \in YX^{<N}$, chooses the extension hy which has the largest value. If we label the root of the previous tree h, this corresponds to a choice of a child X node that has maximal value $V(hy)$. This doesn't uniquely define an agent: if there are multiple actions with the maximal value any one can be choosen. In the chess example, if two moves have the same expected utility in a given situation, there will be an expectimax agent that picks the first one and another that picks the second.

Example 4.6. The following is the previous tree with the valuation function displayed on it. X-node edges are labelled with probabilities $P(x_i|yx_{<i}y_i)$. Y-nodes edges are labelled if the child node has the value of the parent (i.e. has a maximal value within all its siblings). This is just the root of a full tree, it extends at least one more level.



Notice that Y node values are the maximum of the values of their X node children; X node values are the weighted sum of their Y node children, with the weightings on the edges. □

Note that the agent function takes a parameter of the form

$$yx_{1:i}$$

rather than

$$x_{1:i}$$

i.e. we include the agent's past outputs. We call these extended agent functions:

Definition 4.7. An *extended agent function* is a map:

$$a\colon YX^{<N} \to Y$$

where

$$YX^{<N} = \bigcup_{i=0}^{N-1} (Y \times X)^i$$

is the set of all sequences of paired outputs and inputs $Y \times X$ of length less than N. □

Extended agents are technically useful as we can define the action y_i at step i without having to compute $y_{<i}$. This allows us to recursively define a function from longer to shorter histories. They aren't any more general: every extended agent function has an equivalent agent function (see chapter C).

Finally, expectimax agents are optimal agents.

Theorem 4.8. (Expectimax agents are optimal) If a_0 is an expectimax agent

$$\forall h \in YX^{<N}\ a_0(h) \in \operatorname*{argmax}_y V(hy)$$

then a_0 is optimal

$$a_0 \in \operatorname*{argmax}_a E[U|a]$$

Proof. See corollary B.8 in chapter B. □

In fact a stronger result (almost) holds: the set of expectimax agents is exactly the set of optimal agents. This, however, requires we ignore how the agents behave on histories which are expected impossible i.e. any history $yx_{1:i}$ where for some $1 \leq k \leq i$ either

1. y_k is not an action the agent would take

$$y_k \neq a_0(yx_{<k})$$

or,

2. x_k is not an observation the agent would expect

$$P(x_k|yx_{<k}y_k) = 0$$

Expectimax agents must behave optimally even under these impossible situations, where as arbitary optimal agents need not. In a sense, expectimax agents are "more optimal" than arbitary optimal agents. As such, it suffices to consider only expectimax agents.

Example 4.9. Suppose we wish to design an agent to supervise quality control for a production line. The production line produces one widget per time step, and the agent has two options: either accept the widget and allow it to pass, or reject it. Each widget is either good or bad, and the aim is to accept all and only good widgets. All items that are rejected are destructively tested, so the agent knows whether these were good or not. The agent otherwise cannot tell the quality of a widget. The widgets are produced by a machine which has a fixed, but unknown error rate – a fixed probability of producing bad rather than good widgets.

We wish to model this situation using the above formalism of maximal expected utility, then solve for the optimal behaviour using the explicit form described above.

First, our sets:

1. $Y = \{0, 1\}$. 0 means accept the widget, 1 means reject it.

2. $X = \{0, 1\}$. If the last action y_i was 1 (reject), the next input x_i is 0 if the widget was good or 1 if it was bad. Otherwise the input is 0.

3. $R = R_1 \times R_2 = [0, N] \times [0, N]$. Our variable of interest is the number of good and bad widgets accepted. Note that for $r = \langle r_1, r_2 \rangle \in R$:

$$r_1 + r_2 \leq N$$

   They will sum to exactly the number of widgets accepted i.e. the number of y_i's that equal 0.

We give a utility of $+\alpha$ for every good widget accepted and $-\beta$ for every bad widget accepted, where $\alpha, \beta \geq 0$:

$$U(r) = U(\langle r_1, r_2 \rangle) = \alpha r_1 - \beta r_2$$

If we receive $\$\alpha$ for every good widget and get penalised $\$\beta$ for every bad widget this utility is the total number of dollars we will earn.

We define the probability distributions we need

$$P(r|yx_{1:N}), P(x_i|yx_{<i}y_i)$$

with the help of an auxillary variable: ρ, the widget producing machine's fixed probability of error. We then have,

$$\begin{aligned} P(r|yx_{1:N}) &= \int P(r|\rho, yx_{1:N}) P(\rho|yx_{1:N}) d\rho \\ P(x_i|yx_{<i}y_i) &= \int P(x_i|\rho, yx_{<i}y_i) P(\rho|yx_{<i}y_i) d\rho \end{aligned}$$

by a continuous generalisation of marginalisation and the product rule. This first infers the error probability ρ from the past, then the outcome or next input. Our prior information about the problem suggests the following:

1. If $\sum_{i=1}^{N}(1-y_i)$, the number of widgets accepted, is equal to $r_1 + r_2$ then

$$\begin{pmatrix} P(r|\rho, yx_{1:N}) = P(\langle r_1, r_2\rangle|\rho, yx_{1:N}) = \\ r_1 + r_2 r_1 \rho^{r_1}(1-\rho)^{r_2} \end{pmatrix}$$

   otherwise the probability is zero. There are $\binom{r_1+r_2}{r_1}$ ways to have a sequence of r_1 bad and r_2 good widgets, each with probability $\rho^{r_1}(1-\rho)^{r_2}$.

2. Once we know the probability that the machine will make a bad widget, ρ, we know all we can about whether the next block is bad and thus the next input x_i:

$$P(x_i|\rho, yx_{<i}y_i) = \begin{cases} [x_i = 0] & \text{if } y_i = 0 \\ 1-\rho & \text{if } y_i = 1, x_i = 0 \\ \rho & \text{if } y_i = 1, x_i = 1 \end{cases}$$

   If $y_i = 0$ (accept) the next input x_i is always 0. If $y_i = 1$ (reject), our expectation is a ρ probability for being bad, $1-\rho$ for it being good.

3. The last step infers the probability of error ρ from the past history $yx_{1:i}$. Let b be the number of bad widgets rejected, g the number of good

$$\begin{aligned} b &= \Sigma_{j=1}^{i} x_j y_j \\ g &= \Sigma_{j=1}^{i}(1-x_j)y_j \end{aligned}$$

   then

$$\begin{aligned} P(\rho|yx_{1:i}) &= \frac{P(x_{1:i}|\rho, y_{1:i})P(\rho|y_{1:i})}{\int P(x_{1:i}|\rho, y_{1:i})P(\rho|y_{1:i})d\rho} \\ &= \frac{\rho^b(1-\rho)^g}{\int \rho^b(1-\rho)^g d\rho} \\ &= \frac{1}{B(b,g)}\rho^b(1-\rho)^g \end{aligned}$$

   where B is the Beta function. This is a Beta distribution (see [Ber93] p650). This formula assumes a prior expectation that the error probability ρ could equally take on any value between 0 and 1. Note also that

$$P(\rho|yx_{1:i}y_i) = P(\rho|yx_{1:i})$$

   since a trailing action tells us nothing more about the widget generating machine.

Next, we use theorem 4.8 to find the optimal agent. We evaluate the valuation over histories $V(h)$.

$$\begin{aligned} V(yx_{1:N}) &= \sum_r U(r)P(r|yx_{1:N}) \\ &= \sum_r (\alpha r_1 - \beta r_2)P(\langle r_1, r_2\rangle|yx_{1:N}) \\ &= \alpha \sum_{r_1} r_1 P(r_1|yx_{1:N}) - \beta \sum_{r_2} r_2 P(r_2|yx_{1:N}) \end{aligned}$$

One can show that the mean of this Beta distribution is

$$\int \rho P(\rho|yx_{1:N})d\rho = \frac{\Sigma_{j=0}^{i} x_j y_j + 1}{\Sigma_{j=0}^{i} y_j + 2}$$

and as a result that

$$\begin{aligned}\sum_{r_1} r_1 P(r_1|yx_{1:i}) &= \Sigma_{j=0}^{N}(1-y_j)\frac{\Sigma_{j=0}^{N}(1-x_j)y_j + 1}{\Sigma_{j=0}^{N} y_j + 2} \\ \sum_{r_2} r_2 P(r_2|yx_{1:i}) &= \Sigma_{j=0}^{N}(1-y_j)\frac{\Sigma_{j=0}^{N} x_j y_j + 1}{\Sigma_{j=0}^{N} y_j + 2}\end{aligned}$$

and that

$$V(yx_{1:N}) = \frac{\Sigma_{j=0}^{N}(1-y_j)}{\Sigma_{j=0}^{N} y_j + 2}\left(\sum_{j=0}^{N}(\alpha - (\alpha+\beta)x_j)y_j + (\alpha-\beta)\right)$$

(The details of this calculation aren't important.) The right hand sum is the utility of the rejected widgets, $+\alpha$ for each good one $-\beta$ for each bad one (with an extra good and bad widget thrown in). The left hand fraction rescales this utility from the number of rejected (plus 2) widgets to the number of accepted widgets. The intuition behind this is we expect the same proportion of bad widgets in both the accepted and rejected set, so we simply rescale the latter to estimate the former.

Through further calculation, we find

$$V(yx_{<N}y_N) = \frac{\Sigma_{j=0}^{N-1}(1-y_j) - y_N}{\Sigma_{j=0}^{N-1} y_j + 2 + y_N}\left(\sum_{j=0}^{N-1}(\alpha - (\alpha+\beta)x_j)y_j + (\alpha-\beta)\right)$$

Similarily, we find:

$$V(yx_{<i}y_i) = \frac{\Sigma_{j=0}^{i-1}(1-y_j) - y_i - K}{\Sigma_{j=0}^{N-1} y_j + 2 + y_i + K}\left(\sum_{j=0}^{i-1}(\alpha - (\alpha+\beta)x_j)y_j + (\alpha-\beta)\right)$$

where $K = (N-i)$ if the right hand sum is negative, $K = 0$ otherwise. In both cases the next action (y_N or y_i) influences the scaling fraction: if we reject the next widget there will be less to accept later on. The optimal action, for any i, is thus:

$$\operatorname*{argmax}_y V(yx_{<i}y) = \begin{cases} \{0\} & \text{if } \sum_{j=0}^{i-1}(\alpha - (\alpha+\beta)x_j)y_j + (\alpha-\beta) > 0 \\ \{0,1\} & \text{if } \sum_{j=0}^{i-1}(\alpha - (\alpha+\beta)x_j)y_j + (\alpha-\beta) = 0 \\ \{1\} & \text{if } \sum_{j=0}^{i-1}(\alpha - (\alpha+\beta)x_j)y_j + (\alpha-\beta) < 0 \end{cases}$$

The optimal action for the agent is accept (0) if the utility of the rejected widgets (plus $\alpha - \beta$) is greater than zero, reject (1) if its less than zero, or both if its zero. The intuition behind this is the average utility of the rejected fraction (if we had accepted it) is the expected gain if we accept the next widget. If this is negative, we stand to lose: reject. If this is positive, we stand to gain: accept. If it is zero, nothing should change: either is ok.

The benefit of theorem 4.8 is a concrete grasp of how optimal agents act, along with a tool to analyse their behaviour. □

Chapter 5

Applications

This definition of optimality, i.e. of $E[U|a]$, depends on two key parameters:

1. R, $U(r)$, $P(r|yx_{1:N})$. What we want (R, $U(r)$) and how to infer whether we've got it ($P(r|yx_{1:N})$).

2. $P(x_i|yx_{1:i-1}y_i)$. What we know about which inputs the environment supplies given past history of interaction.

We discuss applications of the above theory. First we describe (implicit) applications used by Marcus Hutter in his work on universal AI [Hut04]: AIμ and AIXI. AIμ is intended to be the best AI for any fixed environment, although it requires complete knowledge of this environment to define. AIXI is intended to be the best environmentally independent AI, performing "almost" as well as the correct AIμ in any environment.

We next suggest an area for future work. Although AIμ has complete knowledge of the environment (in a focused sense; see below) it cannot be guaranteed to achieve what we want. This is simply because the utility function it uses is fixed and not our own: it seeks to maximise the reward it receives. With humans in the environment controlling the rewards this may indirectly achieve what we want, but this is not necessarily so.

One possibility is to generalise AIμ to an agent which transparently achieves what we want by adding an extra assumption: its utility function expresses what we want. Designing an AIXI analog for this class of agents is not trivial, especially since "what we want" isn't well defined. This is meant to capture and generalise the idea of communicating what we want to the AI, of the AI extracting what the right thing is from the environment, that is implicit in AIμ. One way to formalise this is by encoding the desired utility function into the environment and requiring the agent to extract it.

5.1 Hutter's AIμ and AIXI

Marcus Hutter's AIμ and AIXI [Hut04] was the original impetus for this thesis. AIμ is a model of (unattainably) optimal AI, with perfect knowledge of the environment. This is a special case of this model with two features:

1. The input X is divided into two parts

$$X = R \times O$$

a reward R (not to be confused with our set of outcomes R) and an ordinary input O. Rewards encode real numbers, and the goal of the agent is the maximise the expected total reward received:

$$\sum_r U(r)P(r|xy_{1:N}) = \sum_i R(x_i)$$

where $R(x_i)$ extracts the reward component from the input. Similar reward-based goals are found in the field of reinforcement learning [SB98].

2. The probability distribution

$$P(x_i|yx_{1:i-1}y_i)$$

predicts the environment as well as possible. Assuming deterministic environments, this always predicts the next input correctly. This corresponds to complete knowledge of how the universe reacts. This isn't complete knowledge of the entire universe, but only the aspect that matters to the AI (which inputs follow which outputs). The AI only "cares" about the reward input it receives.

(Actually, this describes AIμ with a fixed lifetime and full lookahead; see [Hut04].)

AIμ is intended as the goal: the agent which has learnt everything necessary about the environment, and performs the task we wish it to perform. Although the utility function is not something humans directly want (i.e. maximising a numerical input to the agent), the idea is that humans can control how the agent receives rewards and thus control the agent itself. (This seems to require us humans have *perfect* control of the reward system and AI: any loophole, no matter how tiny, will allow AIμ to bypass humans and directly maximise rewards.)

AIXI relaxes the assumption of perfect knowledge. In fact, the only thing it assumes about the environment is that it's computable; more precisely, that it can be accurately modelled as a computable function p mapping agent actions to inputs:

$$p\colon Y^{<N} \to X$$

Under a formalisation of Occam's razor, simpler functions (and thus simpler environments) are seen as initially more plausible. We measure the simplicity of a computable function by the length of the program needed to encode it, $|p|$. We give each possible function p a probability of

$$2^{-|p|}$$

(this requires the set of programs is a prefix free binary language; see [Cal02]). If we were to randomly flip a coin to create a sequence of bits until it described a valid program, e.g. 0, 01, 011, 0111, $2^{-|p|}$ would the the probability that program p was produced.

As as result (applying similar logic as in example 4.9 but with p replacing ρ) we have:

$$\begin{aligned}
P(x_i|yx_{1:i-1}y_i) &= \sum_p P(x_i, p|yx_{1:i-1}y_i) \\
&= \sum_p P(p|yx_{1:i-1}y_i)P(x_i|yx_{1:i-1}y_i, p) \\
&= \sum_p \left(\frac{P(x_{1:i-1}|p, y_{1:i})P(p|y_{1:i})}{\sum_{p'} P(x_{1:i-1}|p', y_{1:i})P(p'|y_{1:i})} \right) [p(y_{1:i}) = x_i] \\
&= \sum_p \left(\frac{[p(y_{1:i-1}) = x_{1:i-1}]2^{-|p|}}{\sum_{p'} [p'(y_{1:i-1}) = x_{1:i-1}]2^{-|p'|}} \right) [p(y_{1:i}) = x_i] \\
&= \frac{\sum_p [p(y_{1:i}) = x_{1:i}]2^{-|p|}}{\sum_p [p(y_{1:i-1}) = x_{1:i-1}]2^{-|p'|}}
\end{aligned}$$

where

$$p(y_{1:i}) = x_{1:i}$$

is short for

$$p(y_1) = x_1, \quad p(y_{1:2}) = x_2, \quad p(y_{1:3}) = x_3, \quad \ldots$$

The probability that input x_i follows $yx_{1:i-1}y_i$ is proportional to the probability that some environment p produces inputs $x_{1:i}$ when the agent takes actions $y_{1:i}$. The denominator is a normalisation constant to ensure this is a probability distribution.

Hutter proves a number of theorems which demonstrate that this model performs almost as well as the AIμ model and as good as any other model independent of the environment. The intuition behind this is if the environment is encoded by a program p, AIXI can learn this program at a rate roughly proportional to the length of this program, after which it performs as well as AIμ.

5.2 Learning what we want

Although Hutter demonstrates AIXI converges toward AIμ, left informally analysed is the claim that AIμ achieves (e.g. by learning) what we want. Regardless of when and whether it does, there is need for a theoretical framework in which to analyse such questions.

One possibility is to generalise AIμ: have an agent with perfect knowledge of the environment and perfect knowledge of what we want (e.g. the human utility function). Call this AI*. One can then study whether and under what circumstances AIXI, AIμ, or any other universal model converges to AI*. This rephrases the problem as extracting a utility function from the environment, or of communicating a utility function to an agent. Alternatively, we can think of this as defining agents that both learn how the environment works and learn what we want.

We can state this problem a little more precisely. Suppose we want can be encoded in a utility function

$$U\colon R \to \mathbb{R}$$

over some set of outcomes R, where

$$P(r|yx_{1:N})$$

defines how to extract knowledge of the outcome r from knowledge of the complete interaction $yx_{1:N}$. This probability distribution effectively *defines* what the outcome $r \in R$ is; it is otherwise simply a meaningless set. The utility function describes which outcome we want to occcur, and how we want tradeoffs under uncertainty to be taken. (See chapters 2.1 and 2.2 if this doesn't make sense.)

We will suppose the agent knows the distribution (i.e. knows how to infer the outcome R) but not the utility U, although the following can easily be generalised so that both are unknown. We define a class of environments E which encode these utility functions; each environment $e \in E$ encodes utility function

$$U_e$$

(there may be multiple environments encoding the same utility function). This may be an environment filled with agents (e.g. humans) which use this utility function, or a tablet describing it, or some more abstract specification. Defining how to encode the utility function is part of the problem. We want to design an agent which learns this utility function, and then tries to maximise it. That is, we wish to design an agent a^* such that for every $e \in E$:

$$E[U_e|e, a^*]$$

is near to

$$\max_a E[U_e|e, a]$$

where

$$E[U_e|e, a]$$

is the utility agent a achieves in environment e. That is, for every environment e in the class E the agent a^* performs "almost as well", according to the utility function U_e encoded in the environment e, as the optimal agent for the environment/utility function can.

The aim is to model simplifications of our situation: humans, existing within the agent's environment, want various things. The agent should try to determine and achieve what us humans want it to achieve. This would be considerably simplified if we knew exactly what we wanted, e.g. if all humans wanted was to calculuate as many digits of π as possible: then we could directly code the utility function into the agent. As we don't understand ourselves well enough to completely describe what we want, we need to design agents that can learn or otherwise extract what we want themselves.

Chapter 6

Conclusion

We've described a decision theoretical approach to AI through optimal agents. We explained how probability distributions represent uncertain knowledge of outcomes, and how utility functions represent preferences over outcomes under uncertainty. The best choice results in an uncertain outcome with maximal expected utility.

The agent approach to AI, that of intelligent interactive systems, was formalised through agent functions. The best agent function, that of the optimal agent, has maximal expected utility. We described an explicit algorithm for computing such agents, the expectimax algorithm, which evaluates a numerical function over the tree of possible futures to determine the best next action.

We briefly discussed how Marcus Hutter's [Hut04] model of AI, AIμ and AIXI, is a particular instance of the optimal agent defined here. We sketched an approach to defining when an agent can be said to learn what we want. Future work would apply this, or a similar framework, to analyse when and whether AIXI can be shown to achieve what we want.

Appendix A

Probability Theory

We introduce probability theory as a logic of uncertainty (see [JB03],[Jay86] for a more detailed presentation; [Har02] for a different derivation). We first describe propositions, then how to assign truth values and probabilities to them. We end with a description of probability distributions, and a several theorems for manipulating them.

A.1 Logic

Propositions are statements that can be either true or false. For example, "the agent receives input 0 at time step 12". When modelling an agent, or a decision problem in general, we implicitly use a background set of propositions. This background set will vary from problem to problem. With our agent model we have propositions of the form:

"The agent receives input x at time step i"	for all $x \in X$, $i \in [1, N]$
"The agent sends output y at time step i"	for all $y \in Y$, $i \in [1, N]$
"Agent function a is implemented"	for all agent functions a
"Outcome r occurs"	for all outcomes $r \in R$

These schemata describe families of propositions, one for each instantiation of the variables.

Propositions can form other propositions. Negation constructs "the agent does not receive input 0 at time step 12" from the first example sentence. In general, from propositions "X" and "Y" we can construct "not X", "X and Y", "X or Y" each with their obvious meanings. These are operations on the background set of propostions. We assume this background set is closed under these operations e.g. if "X" and "Y" are in the set, "X and Y" is too.

Logic deals with the truth of propositions. Each proposition is assigned a truth value, either true or false, in a consistent way. The consistency requirement is if one proposition logically implies another then whenever the first proposition is true, the second is true e.g. if the proposition "X" is true then the proposition "X or Y" must be true, if "X and Y" is true then "X" must be true.

A.2 Probability as uncertain logic

Assigning true and false to propositions is sufficient to handle both the actual state of the world and certain knowledge of it. Uncertain knowledge requires degrees of belief lying between the extremes of certainly true and certainly false. To achieve this we assign propositions real numbers between 0 and 1, called probabilities. 0 means certainly false, 1 means certainly true, anything else is more or less uncertain. We denote proposition X's probability as

$$P(X)$$

For propositions X and Y, $X \wedge Y$ or XY will denote the proposition "X and Y", $X \vee Y$ the proposition "X or Y", $\neg X$ the proposition "not X".

An important feature of uncertain life is we can learn new things. In particular, if we discover a certain proposition is true this changes our probabilities that other propositions are true. If we discover "it is raining", we assign a higher probability to "the ground is wet". If E is the proposition describing everything we know to be true (our *e*vidence), then

$$P(X|E)$$

is the probability of X given that we know E is true. This is called the probability of X conditional on E. It turns out that that

$$P(X|E) = \frac{P(XE)}{P(E)}$$

If we know multiple things, $E_1, \ldots, E_n$, we can combine them into a single proposition

$$P(X|E_1 \vee \ldots \vee E_n) = P(X|E)$$

Our unconditional probability from before is a special case where E is always known to be true i.e. where $P(E) = 1$. This holds for tautologies $\top$ such as $E \wedge \neg E$ (e.g. "to be or not to be" – necessarily true since at least one of them holds):

$$P(X) = P(X|\top)$$

Just as with truth values, there are consistency requirements in assigning probabilities. Firstly, we require logically equivalent statements to have the same probability. Secondly,

$$P(\neg X|E) = 1 - P(X|E) \tag{A.1}$$
$$P(XY|E) = P(X|E)P(Y|XE) \tag{A.2}$$

where X, Y are arbitary propositions, and E is the background evidence proposition we condition on. The first rule means that belief in X is disbelief in $\neg X$, and conversely. The second rule, also called the product rule, means we can judge the probability of XY in two steps, all conditional on E: the probability of X then the probability of Y given X. The probablity of "its raining and the ground is wet", is the probabilty of "its raining" times the probability of "the ground is wet" given that "its raining".

The above elementary rules of probability can be used to derive further results, and can be derived from more basic principles [JB03]. For example,

$$P(X \vee Y|E) = P(X|E) + P(Y|E) - P(XY|E)$$

where $X \vee Y$ is "X or Y". We also have Bayes' theorem:

$$P(X|YE) = \frac{P(X|E)P(Y|XE)}{P(Y|E)}$$

A.3 Variables and probability distributions

We don't deal with propositions directly, but through variables. A variable V has an set of possible values $V = \{v_1, \ldots, v_n\}$, although it is not generally known which value it has. For each value $v \in V$ the proposition

$$V = v$$

says that V has the value v. Exactly one such proposition will be true i.e.

$$P(V = v_1 \vee \ldots \vee V = v_n) = 1$$

and for any $i \neq j$

$$P(V = v_i, V = v_j) = 0$$

Variables are useful when we have a set of alternatives, exactly one of which must be true. For example, the input the agent received at time step i. Any proposition P corresponds to the variable V_P which has values $\{0, 1\}$ depending on whether P is true or false. More generally, any set of propositions $S = \{P_1, \ldots, P_n\}$ that are mutually exclusive and exhaustive i.e. such that

$$P(P_1 \vee \ldots \vee P_n) = 1$$

and for any $i \neq j$

$$P(P_i, P_j) = 0$$

forms a variable in an obvious way.

It can be shown, via induction on n and eq (A.2), that

$$\sum_{v \in V} P(V = v) = 1$$

This leads to probability distributions, functions which assign probabilities to values of variables rather than general propositions:

$$P\colon V \to [0, 1]$$

where $P(v) = P(V = v)$. These are simpler to deal with as the consistency requirement is simply that all the values sum to 1

$$\sum_{v} P(v) = 1$$

When dealing with multiple variables, say X and Y and Z,

$$P(x, y|z)$$

denotes the conditional probability that variable X has value x and Y value y, given that we know Z has value z. Implicitly, $x \in X$, $y \in Y$, $z \in Z$. We use labels to denote the variables involved. A more explicit version of the above would be

$$P(X = x, Y = y|Z = z)$$

It is generally clear from context which variable X a label x refers to.

A.4 Marginalisation

Suppose we have two variables X and Y. $P(y)$ denotes the probability that Y has value y, $P(x, y)$ that X has value x and Y value y. Whatever value Y has, e.g. y, X must have exactly one value. As a result,

$$P(y) = \sum_{x \in X} P(x, y)$$

holds. This marginalisation theorem is often used to introduce an additional variable.

A.5 Bayes' theorem

Suppose we have two variablse X and E. Typically X is something we are uncertain of, and E is something we are certain of e.g. evidence. We have

$$P(x|e) = \frac{P(e|x)P(x)}{\sum_{x \in X} P(e|x)P(x)}$$

where

$$P(x)$$

is the prior probability that X has value x, before knowing evidence e, and

$$P(e|x)$$

is the probability we would observe evidence e if x happened to be true.

Appendix B

Expectimax Agents are Optimal Agents

This chapter supplies a proof that expectimax agents are optimal agents. See chapter 4 for why this is interesting.

Recall definition 4.5 of the expectimax agent. We have a valuation function $V(h)$ over histories:

$$\begin{aligned} V(yx_{1:N}) &= \sum_r U(r)P(r|yx_{1:N}) \\ V(yx_{<i}y_i) &= \sum_{x_i} P(x_i|yx_{<i}y_i)V(yx_{<i}yx_i) \\ V(yx_{<i}) &= \max_{y_i} V(yx_{<i}y_i) \end{aligned}$$

An expectimax agent a_0 is one that for all histories $h \in YX^{<N}$ takes an action maximising this valuation:

$$\forall h \in YX^{<N}\ a_0(h) \in \operatorname*{argmax}_y V(hy)$$

B.1 Possible futures

Given a history of interaction h, we define the set of possible futures extending h, relative to an agent a. This is the set of all futures where the actions are consistent with the agent function a and no "impossible" inputs occur. It will turn out that optimal agent functions need only be optimal on this subset of histories, and can take any action on "impossible" futures.

Definition B.1. $H(\dot{yx}_{\leq i}, a)$ is the set of all possible futures (also called valid futures) extending $\dot{yx}_{\leq i}$ by agent a:

$$\begin{aligned} H(\dot{yx}_{\leq i}, a) = \{&\dot{yx}_{\leq i} yx_{i+1:j-1} : \\ &j \in [i+1, N],\ yx_{i+1:j-1} \in (YX)^{j-i-1}, \\ &(\forall k > i)\ y_k = a(\dot{yx}_{\leq i} yx_{i+1:k-1})\ \wedge\ P(x_k|\dot{yx}_{\leq i} yx_{i+1:k-1}y_k) \neq 0\} \end{aligned}$$

□

This definition says a possible future is made up of:

$$\dot{yx}_{\leq i} y x_{i+1:j-1}$$

the actual past $\dot{yx}_{\leq i}$ and a future $yx_{i+1:j-1}$ of variable length. Every future action y_k for $i < k \leq j-1$ is the action agent a would take:

$$y_k = a(\dot{yx}_{\leq i} y x_{i+1:k-1})$$

and each input x_k for $i < k \leq j-1$ is seen as possible given the past:

$$P(x_k | \dot{yx}_{\leq i} y x_{i+1:k-1} y_k) \neq 0$$

We use the following notation to conserve space:

Definition B.2. The utility of a complete history $yx_{1:N}$ is defined to be:

$$U(yx_{1:N}) = \sum_{r \in R} U(r) P(r | yx_{1:N})$$

□

The final definition we require is:

Definition B.3. The expected utility of an agent a, given that it starts acting in step i with past history $\dot{yx}_{<i}$:

$$E[U | \dot{yx}_{<i}, a] \quad = \quad \sum_{r \in R} U(r) P(r | \dot{yx}_{<i}, a)$$

Note that this reduces to the original definition $E[U|a]$ when $i = 1$ i.e. when agent starts on first step. □

It will turn out that

$$V(\dot{yx}_{<i}) = \max_a E[U | \dot{yx}_{<i}, a]$$

i.e. the valuation of history $\dot{yx}_{<i}$ is the expected utility of the optimal agent, given that it starts acting at step i with history $\dot{yx}_{<i}$. This is the crucial intermediate result in showing the expectimax agent, defined through V, yields optimal agents.

B.2 Identities of the expected utility of an agent

We will use the following identities to prove our later result.

The following lemma rephrases definition B.3:

Lemma B.4.

$$E[U | \dot{yx}_{<i}, a] \quad = \quad \sum_{y x_{i:N}} U(\dot{yx}_{<i} y x_{i:N}) P(y x_{i:N} | \dot{yx}_{<i}, a)$$

$$E[U | \dot{yx}_{<i} y_i, a] \quad = \quad \sum_{y x_{i:N}} U(\dot{yx}_{<i} y x_{i:N}) P(x_i y x_{i+1:N} | \dot{yx}_{<i} y_i, a)$$

Proof. The following invokes probability theory (in particular, marginalisation $P(y|z) = \sum_x P(x, y|z)$ and the product rule $P(x, y|z) = P(x|z)P(y|x, z)$; see chapter A) and definition B.2:

$$\begin{aligned}
E[U|\dot{y}x_{<i}, a] &= \sum_{r\in R} U(r)P(r|\dot{y}x_{<i}, a) \\
&= \sum_{r\in R} U(r) \sum_{yx_{i:N}} P(r, yx_{i:N}|\dot{y}x_{<i}yx_{i:N}, a) \\
&= \sum_{r\in R} U(r) \sum_{yx_{i:N}} P(yx_{i:N}|\dot{y}x_{<i}, a)P(r|\dot{y}x_{<i}yx_{i:N}, a) \\
&= \sum_{yx_{i:N}} \left(\sum_{r\in R} U(r)P(r|\dot{y}x_{<i}yx_{i:N}, a) \right) P(yx_{i:N}|\dot{y}x_{<i}, a) \\
&= \sum_{yx_{i:N}} U(\dot{y}x_{<i}yx_{i:N})P(yx_{i:N}|\dot{y}x_{<i}, a) \\
E[U|\dot{y}x_{<i}y_i, a] &= \sum_{r\in R} U(r)P(r|\dot{y}x_{<i}y_i, a) \\
&= \sum_{r\in R} U(r) \sum_{x_iyx_{i+1:N}} P(x_iyx_{i+1:N}|\dot{y}x_{<i}y_i, a)P(r|\dot{y}x_{<i}yx_{i:N}, a) \\
&= \sum_{yx_{i:N}} \left(\sum_{r\in R} U(r)P(r|\dot{y}x_{<i}yx_{i:N}, a) \right) P(yx_{i:N}|\dot{y}x_{<i}, a) \\
&= \sum_{yx_{i:N}} U(\dot{y}x_{<i}yx_{i:N})P(x_iyx_{i+1:N}|\dot{y}x_{<i}y_i, a)
\end{aligned}$$

□

The following theory states the major recursive properties of the expected utility of an agent given a fixed history. The expected utility for shorter histories can be defined in terms of longer histories, the value on the longest history being indepedent of the agent a (there are no actions for it to take).

Theorem B.5. The following identities hold:

$$\begin{aligned}
E[U|\dot{y}x_{1:N}, a] &= U(\dot{y}x_{1:N}) \\
E[U|\dot{y}x_{<i}, a] &= E[U|\dot{y}x_{<i}a(\dot{y}x_{<i}), a] \\
E[U|\dot{y}x_{<i}\dot{y}_i, a] &= \sum_{x_i} P(x_i|\dot{y}x_{<i}\dot{y}_i)E[U|\dot{y}x_{<i}\dot{y}_ix_i, a]
\end{aligned}$$

Proof. 1. Invoking definition B.2:

$$\begin{aligned}
E[U|\dot{y}x_{1:N}, a] &= \sum_{r\in R} U(r)P(r|\dot{y}x_{1:N}) \\
&= U(\dot{y}x_{1:N})
\end{aligned}$$

2. Invoking lemma B.4, the product rule, and the fact that $P(y_i|\dot{y}x_{<i}, a) = [y_i =$

$a(\dot{y}x_{<i})$]:

$$\begin{aligned}
& E[U|\dot{y}x_{<i}, a] \\
= & \sum_{yx_{i:N}} U(\dot{y}x_{<i}yx_{i:N})P(yx_{i:N}|\dot{y}x_{<i}, a) \\
= & \sum_{yx_{i:N}} U(\dot{y}x_{<i}yx_{i:N})P(y_i|\dot{y}x_{<i}, a)P(x_iyx_{i+1:N}|\dot{y}x_{<i}y_i, a) \\
= & \sum_{x_iyx_{i+1:N}} U(\dot{y}x_{<i}a(\dot{y}x_{<i})x_iyx_{i+1:N})P(x_iyx_{i+1:N}|\dot{y}x_{<i}a(\dot{y}x_{<i}), a) \\
= & E[U|\dot{y}x_{<i}a(\dot{y}x_{<i}), a]
\end{aligned}$$

3. Invoking lemma B.4, the product rule, and the fact that $P(x_i|\dot{y}x_{<i}y_i, a) = P(x_i|\dot{y}x_{<i}y_i)$:

$$\begin{aligned}
& E[U|\dot{y}x_{<i}\dot{y}_i, a] \\
= & \sum_{x_iyx_{i+1:N}} U(\dot{y}x_{<i}yx_{i:N})P(x_iyx_{i+1:N}|\dot{y}x_{<i}y_i, a) \\
= & \sum_{x_iyx_{i+1:N}} U(\dot{y}x_{<i}yx_{i:N})P(x_i|\dot{y}x_{<i}y_i, a)P(yx_{i+1:N}|\dot{y}x_{<i}\dot{y}_ix_i, a) \\
= & \sum_{x_i} P(x_i|\dot{y}x_{<i}y_i, a) \sum_{yx_{i+1:N}} U(\dot{y}x_{<i}yx_{i:N})P(yx_{i+1:N}|\dot{y}x_{<i}\dot{y}_ix_i, a) \\
= & \sum_{x_i} P(x_i|\dot{y}x_{<i}y_i)E[U|\dot{y}x_{<i}\dot{y}_ix_i, a]
\end{aligned}$$

□

B.3 Expectimax agents are optimal agents

We prove this through two theorems. Theorem B.6 shows optimal agent functions can be described component-wise. Theorem B.7 shows these components correspond to those in the expectimax agent definition. Corollary B.8 shows that expectimax agents are optimal agents, the final result. That the entire set of optimal agents are expectimax agents on the set of possible futures (see definition B.1 above) is left implicit. (See also the notes after theorem 4.8.)

Theorem B.6. For any fixed history $\dot{y}x_{\leq i}$ where $0 \leq i \leq N$:

$$a_0 \in \operatorname*{argmax}_a E[U|\dot{y}x_{\leq i}, a]$$

iff

$$\forall h \in H(\dot{y}x_{\leq i}, a_0)\ a_0(h) \in \operatorname*{argmax}_y \max_a E[U|hy, a]$$

This theorem shows optimal agents can be defined componentwise. The equation

$$a_0 \in \operatorname*{argmax}_a E[U|y\dot{x}_{\leq i}, a]$$

means a_0 is an optimal agent, given fixed history $y\dot{x}_{\leq i}$. That is, if it only acts on steps $i+1 \dots N$, and given a fixed history $y\dot{x}_{\leq i}$ on steps $1 \dots i$, it has maximal expected utility. For $i = 0$ this is the standard definition of optimality $E[U|\epsilon, a] = E[U|a]$: the agent acts on all N steps. For $i = N$ this is a trivial statement: there are no actions after the Nth steps, so the agent a has no effect on the expectation $E[U|yx_{\leq N}, a]$.

The other statement:

$$\forall h \in H(y\dot{x}_{\leq i}, a_0)\ a_0(h) \in \operatorname*{argmax}_y \max_a E[U|hy, a]$$

says that for all valid futures h extending $y\dot{x}_{\leq i}$ agent a_0 takes the best action y, assuming that future actions are taken optimally (this is the meaning of $\max_a E[U|hy, a]$). This defines individual values of a_0.

Proof. We prove this by induction over i in the theorem.

For $i = N$ the equivalence is vacuously true:

1. $a_0 \in \operatorname{argmax}_a E[U|y\dot{x}_{\leq N}, a]$ holds for any a_0, since the agent acts for only N steps.

2. $H(y\dot{x}_{\leq N}, a_0)$ is the empty set

Supposing the inductive hypothesis holds for $i+1$ where $1 \leq i+1 \leq N$, we show it holds for i. For any history $y\dot{x}_{\leq i}$ we have:

$$\begin{aligned}
& a_0 \in \operatorname*{argmax}_a E[U|y\dot{x}_{\leq i}, a] \\
\overset{(1)}{\iff}\quad & a_0 \in \operatorname*{argmax}_a E[U|y\dot{x}_{\leq i} a(y\dot{x}_{\leq i}), a] \\
\overset{(2)}{\iff}\quad & a_0(y\dot{x}_{\leq i}) = y^*_{i+1} \in \operatorname*{argmax}_{y_{i+1}} \max_a E[U|y\dot{x}_{\leq i} y_{i+1}, a] \\
\wedge\quad & a_0 \in \operatorname*{argmax}_a E[U|y\dot{x}_{\leq i} y^*_{i+1}, a] \\
\overset{(3)}{\iff}\quad & a_0(y\dot{x}_{\leq i}) = y^*_{i+1} \in \operatorname*{argmax}_{y_{i+1}} \max_a E[U|y\dot{x}_{\leq i} y_{i+1}, a] \\
\wedge\quad & \forall x_{i+1} : P(x_{i+1}|y\dot{x}_{\leq i} y^*_{i+1}) \neq 0, \quad a_0 \in \operatorname*{argmax}_a E[U|y\dot{x}_{\leq i} y^*_{i+1} x_{i+1}, a] \\
\overset{(4)}{\iff}\quad & \forall h \in H(y\dot{x}_{\leq i}, a_0)\ a_0(h) \in \operatorname*{argmax}_y \max_a E[U|hy, a]
\end{aligned}$$

Justification for each step:

1. Lemma (B.5).

2. First note the equivalence:

$$(x,y) \in \operatorname*{argmax}_{x,y} f(x,y) \iff \begin{array}{rcl} x & \in & \operatorname{argmax}_x \max_y f(x,y) \\ \wedge \quad y & \in & \operatorname{argmax}_y f(x,y) \end{array} \tag{B.1}$$

holds i.e. when maximising a function of two variables one can choose the first coordinate then the second.

For any history $y\dot{x}_{\leq i}$ we can split the agent function a into two parts: $\langle a(y\dot{x}_{\leq i}), a^\dagger \rangle$ where $a^\dagger$ is value of a on all partial histories *except* $y\dot{x}_{\leq i}$. This separates the present decision ($a(yx_{\leq i})$) from the rest of the agent function ($a^\dagger$). With this decomposition:

$$\begin{aligned} & a_0 \in \operatorname*{argmax}_a E[U|y\dot{x}_{\leq i} a(y\dot{x}_{\leq i}), a] \\ \iff \quad & \langle a_0(y\dot{x}_{\leq i}), a_0^\dagger \rangle \in \operatorname*{argmax}_{y_{i+1}, a^\dagger} E[U|y\dot{x}_{\leq i} y_{i+1}, a^\dagger] \end{aligned}$$

as $E[U|y\dot{x}_{\leq i} y_{i+1}, a]$ depends only on the $a^\dagger$ component i.e.

$$E[U|y\dot{x}_{\leq i} y_{i+1}, a] = E[U|y\dot{x}_{\leq i} y_{i+1}, a^\dagger] \tag{B.2}$$

(The present decision, $a(y\dot{x}_{\leq i})$, has already been taken so it cannot affect the expected utility.) Applying equivalence (B.1), we get:

$$\begin{aligned} \iff \quad & a_0(y\dot{x}_{\leq i}) = y^*_{i+1} \in \operatorname*{argmax}_{y_{i+1}} \max_{a^\dagger} E[U|y\dot{x}_{\leq i} y_{i+1}, a^\dagger] \\ \wedge \quad & a_0^\dagger \in \operatorname*{argmax}_{a^\dagger} E[U|y\dot{x}_{\leq i} y^*_{i+1}, a^\dagger] \end{aligned}$$

Finally, applying (B.2) again we get:

$$\begin{aligned} \iff \quad & a_0(y\dot{x}_{\leq i}) = y^*_{i+1} \in \operatorname*{argmax}_{y_{i+1}} \max_{a} E[U|y\dot{x}_{\leq i} y_{i+1}, a] \\ \wedge \quad & a_0 \in \operatorname*{argmax}_{a} E[U|y\dot{x}_{\leq i} y^*_{i+1}, a] \end{aligned}$$

3. By lemma (B.5):

$$\begin{aligned} & a_0 \in \operatorname*{argmax}_a E[U|y\dot{x}_{\leq i} y^*_{i+1}, a] \\ \iff \quad & a_0 \in \operatorname*{argmax}_a \sum_{x_{i+1}} P(x_{i+1}|y\dot{x}_{\leq i} y^*_{i+1}) E[U|y\dot{x}_{\leq i} y^*_{i+1} x_{i+1}, a] \end{aligned}$$

Now,

$$\begin{aligned} & \forall x_{i+1} : P(x_{i+1}|y\dot{x}_{\leq i} y^*_{i+1}) \neq 0, \quad a_0 \in \operatorname*{argmax}_a E[U|y\dot{x}_{\leq i} y^*_{i+1} x_{i+1}, a] \\ \implies \quad & a_0 \in \operatorname*{argmax}_a \sum_{x_{i+1}} P(x_{i+1}|y\dot{x}_{\leq i} y^*_{i+1}) E[U|y\dot{x}_{\leq i} y^*_{i+1} x_{i+1}, a] \end{aligned}$$

holds because if a value x_0 is a maximum of each of $f_j(x)$ with nonzero weight c_j then it is a maximum of $\sum_j c_j f_j(x)$ (weights are nonnegative).

The converse holds for the following reason. If $x_0 = \langle x_0^1, \ldots, x_0^n \rangle$ maximises the function $f(x) = f(x^1, \ldots, x^n) = \sum_j c_j f_j(x^j)$ then x_0^j maximises each function f_j with nonzero weight c_j. Since $E[U|yx_{\leq i}y^*_{i+1}x_{i+1}, a]$ depends only on the value of a at arguments extending $yx_{\leq i}y^*_{i+1}x_{i+1}$ (as the agent takes actions depending on the entire past history), we can split a into separate parts $a^{x_{i+1}}$ for each value of x_{i+1} – one for each term in the below sum. So,

$$\begin{aligned}
& a_0 \in \operatorname*{argmax}_a \sum_{x_{i+1}} P(x_{i+1}|yx_{\leq i}y^*_{i+1}) E[U|yx_{\leq i}y^*_{i+1}x_{i+1}, a] \\
\iff \quad & a_0 \in \operatorname*{argmax}_a \sum_{x_{i+1}} P(x_{i+1}|yx_{\leq i}y^*_{i+1}) E[U|yx_{\leq i}y^*_{i+1}x_{i+1}, a^{x_{i+1}}] \\
\implies \quad & \forall x_{i+1} : P(x_{i+1}|yx_{\leq i}y^*_{i+1}) \neq 0, \\
& a_0^{x_{i+1}} \in \operatorname*{argmax}_{a^{x_{i+1}}} E[U|yx_{\leq i}y^*_{i+1}x_{i+1}, a^{x_{i+1}}] \\
\iff \quad & \forall x_{i+1} : P(x_{i+1}|yx_{\leq i}y^*_{i+1}) \neq 0, \\
& a_0 \in \operatorname*{argmax}_a E[U|yx_{\leq i}y^*_{i+1}x_{i+1}, a]
\end{aligned}$$

In summary,

$$\begin{aligned}
& a_0 \in \operatorname*{argmax}_a E[U|yx_{\leq i}y^*_{i+1}, a] \\
\iff \quad & a_0 \in \operatorname*{argmax}_a \sum_{x_{i+1}} P(x_{i+1}|yx_{\leq i}y^*_{i+1}) E[U|yx_{\leq i}y^*_{i+1}x_{i+1}, a] \\
\iff \quad & \forall x_{i+1} : P(x_{i+1}|yx_{\leq i}y^*_{i+1}) \neq 0, \quad a_0 \in \operatorname*{argmax}_a E[U|yx_{\leq i}y^*_{i+1}x_{i+1}, a]
\end{aligned}$$

4. With the shorthand

$$Q = \{x_{i+1} : P(x_{i+1}|yx_{\leq i}y^*_{i+1}) \neq 0\}$$

by the inductive hypothesis we have

$$\begin{aligned}
& \forall x_{i+1} \in Q, \quad a_0 \in \operatorname*{argmax}_a E[U|yx_{\leq i}y^*_{i+1}x_{i+1}, a] \\
\iff \quad & \forall x_{i+1} \in Q, \quad \forall h \in H(yx_{\leq i}y^*_{i+1}x_{i+1}, a_0) \\
& a_0(h) \in \operatorname*{argmax}_y \max_a E[U|hy, a]
\end{aligned}$$

Recall that $H(yx_{\leq i}, a_0)$ is the set of all possible futures extending $yx_{\leq i}$ by agent a_0. That is, each future action y_j is the action a_0 would take:

$$y_{i+1} = a_0(yx_{\leq i}yx_{i:j})$$

and each input x_j is seen as possible given the past:

$$P(x_j|yx_{\leq i}yx_{i:j}) \neq 0$$

Since every valid extension of $yx_{\leq i}$ is either $yx_{\leq i}$ itself or a valid extension of $yx_{\leq i}y^*_{i+1}x_{i+1}$ for some $x_{i+1} \in Q$ (since $y^*_{i+1} = a_0(yx_{\leq i})$) we have:

$$H(yx_{\leq i}, a_0) = \{yx_{\leq i}\} \cup \bigcup_{x_{i+1} \in Q} H(yx_{\leq i}a_0(yx_{\leq i})x_{i+1}, a_0)$$

As a result,

$$\begin{aligned} & a_0(\dot{y}x_{\leq i}) = y^*_{i+1} \in \operatorname*{argmax}_{y_{i+1}} \max_a E[U|\dot{y}x_{\leq i}y_{i+1}, a] \\ \wedge \quad & \forall x_{i+1} \in Q, \quad \forall h \in H(\dot{y}x_{\leq i}y^*_{i+1}x_{i+1}, a_0) \\ & a_0(h) \in \operatorname*{argmax}_y \max_a E[U|hy, a] \\ \iff \quad & \forall h \in H(\dot{y}x_{\leq i}, a_0)\ a_0(h) \in \operatorname*{argmax}_y \max_a E[U|hy, a] \end{aligned}$$

□

Theorem B.7. The valuation $V(h)$ is exactly the expected utility of the optimal agent given a fixed history:

$$V(h) = \max_a E[U|h, a]$$

for any history $h = yx_{<i}$ or $h = yx_{<i}y_i$

This shows at any point in time the expectimax agent selects the action which yields the largest expected utility, assuming that future actions are taken optimally. The valuation $V(h)$ computed over the tree of possible futures is exactly $\max_a E[U|h, a]$.

Proof. Recall definitions (4.4) and (B.2):

$$\begin{aligned} V(yx_{1:N}) &= U(yx_{1:N}) \\ V(yx_{<i}y_i) &= \sum_{x_i} P(x_i|yx_{<i}y_i)V(yx_{<i}yx_i) \\ V(yx_{<i}) &= \max_{y_i} V(yx_{<i}y_i) \end{aligned}$$

Equality between V and $\max_a E[U|\cdot, a]$ follows from the definition of V by induction over histories.

1. By theorem (B.5), $U(yx_{1:N}) = E[U|yx_{1:N}, a]$. Since the left hand side is independent of a it equals $\max_a E[U|yx_{1:N}, a]$. Thus

$$V(yx_{1:N}) = U(yx_{1:N}) = \max_a E[U|yx_{1:N}, a]$$

2. We can divide a into segments a^{x_i} for each $x_i \in X$ such that:

$$E[U|yx_{<i}yx_i, a] = E[U|yx_{<i}yx_i, a^{x_i}] \tag{B.3}$$

This is possible because the value on the left hand side depends on a evaluated only on histories extending $yx_{<i}yx_i$: for different x_i, a can be independently defined. Importantly, this means we can construct agents maximising

$E[U|yx_{<i}yx_i, a]$ for each x_i and combine them into one that maximises them all.

$$\begin{aligned}
V(yx_{<i}y_i) &\overset{(1)}{=} \sum_{x_i} P(x_i|yx_{<i}y_i)V(yx_{<i}yx_i) \\
&\overset{(2)}{=} \sum_{x_i} P(x_i|yx_{<i}y_i) \max_a E[U|yx_{<i}yx_i, a] \\
&\overset{(3)}{=} \sum_{x_i} \max_a P(x_i|yx_{<i}y_i)E[U|yx_{<i}yx_i, a] \\
&\overset{(4)}{=} \sum_{x_i} \max_{a^{x_i}} P(x_i|yx_{<i}y_i)E[U|yx_{<i}yx_i, a^{x_i}] \\
&\overset{(5)}{=} \max_a \sum_{x_i} P(x_i|yx_{<i}y_i)E[U|yx_{<i}yx_i, a^{x_i}] \\
&\overset{(6)}{=} \max_a \sum_{x_i} P(x_i|yx_{<i}y_i)E[U|yx_{<i}yx_i, a] \\
&\overset{(7)}{=} \max_a E[U|yx_{<i}y_i, a]
\end{aligned}$$

Steps 1 follows from the definition of V, step 2 by the inductive hypothesis. Step 3 holds as $P(x_i|yx_{<i}y_i)$ is a positive constant independent of a. Step 4 is the application of equation (B.3): we decompose a into $\langle a^{x_i} : x_i \in X\rangle$.

Step 5 follows form the result mentioned before: maxima for a^{x_i} separately can be combined into a single maxima a. In addition, we can maximise a sum $\sum_j f_j(c_j)$ by maximising each term $f_j(c_j)$ separately (see theorem (B.6)). Finally, step 6 and 7 follows from equation (B.3) and theorem (B.5) respectively.

3.

$$\begin{aligned}
V(yx_{<i}) &\overset{(1)}{=} \max_{y_i} V(yx_{<i}y_i) \\
&\overset{(2)}{=} \max_{y_i} \max_a E[U|yx_{<i}y_i, a] \\
&\overset{(3)}{=} \max_a \max_{y_i} E[U|yx_{<i}y_i, a] \\
&\overset{(4)}{=} \max_a E[U|yx_{<i}, a]
\end{aligned}$$

Step 1 follows from the definition of V, step 2 from the inductive hypothesis, and step 4 from theorem (B.5). Finally step 3 holds as

$$\max_x \max_y f(x, y) = \max_{x,y} f(x, y) = \max_y \max_x f(x, y)$$

selecting the maximal pair x, y can be done in any order.

□

Corollary B.8. (Expectimax agent is optimal) If

$$\forall h \in YX^{<N}\ a_0(h) \in \operatorname*{argmax}_y V(hy)$$

then

$$a_0 \in \operatorname*{argmax}_a E[U|a]$$

Proof. In the following, the first step holds as $YX^{<N} \subseteq H(\epsilon, a_0)$ whilst the final steps are instances of theorems B.7 and B.6, respectively with $i = 0$. Recall also that $E[U|\epsilon, a] = E[U|a]$.

$$\begin{aligned}
& \forall h \in YX^{<N}\ a_0(h) \in \operatorname*{argmax}_y V(hy) \\
\implies\ & \forall h \in H(\epsilon, a_0)\ a_0(h) \in \operatorname*{argmax}_y V(hy) \\
\iff\ & \forall h \in H(\epsilon, a_0)\ a_0(h) \in \operatorname*{argmax}_y \max_a E[U|hy, a] \\
\iff\ & a_0 \in \operatorname*{argmax}_a E[U|a]
\end{aligned}$$

□

Appendix C

Odds and Ends

C.1 Extended agents are equivalent to agents

Extended agents and agents are equivalent in the following sense:

1. Any extended agent a_e uniquely defines an agent a by:

$$a(x_{1:i}) = a_e(\bar{a}_e(x_{1:i}))$$

where $\bar{a}_e(x_{1:i})$ maps $x_{1:i}$ to $yx_{i:1}$ determining the past actions from the past inputs:

$$\begin{aligned} \bar{a}_e(\epsilon) &= \epsilon \\ \bar{a}_e(x_{1:i}) &= \bar{a}_e(x_{<i})a_e(\bar{a}_e(x_{<i}))x_i \end{aligned}$$

We often refer to both unextended and extended agents as "agents"; which kind of agent will be clear from the context.

2. Any agent a defines a family of extended agents equivalent to it (i.e. set of a_e such that $a(x_{<i}) = a_e(\bar{a}_e(x_{<i}))$) by recursively defining for all $x_{1:i}$:

$$a_e(\bar{a}_e(x_{1:i})) = a(x_{1:i})$$

(this definition is non-circular as $\bar{a}_e$ depends only on values of a_e shorter than $x_{1:i}$) and setting a_e arbitarily on other arguments.

C.2 Nondeterministic and deterministic agents equivalent

A nondeterministic agent π defines the probability of randomly choosing a particular output y_i given an input $x_{<i}$ where

$$\pi(y_i|x_{<i})$$

denotes this probability. When,

$$\pi(y_i|x_{<i}) = [y_i = a(x_{<i})]$$

for some agent function a we call the agent π deterministic. The following theorem shows there are deterministic optimal agents.

Theorem C.1. For every nondeterministic agent π there exists a deterministic agent a_π with equal or greater expected utility:

$$E[U|\pi] \leq E[U|a_\pi]$$

Proof. We prove a stronger result by induction on i: for any history $\dot{y}x_{<i}$ and nondeterministic agent π we can find a deterministic agent a_π such that

$$E[U|\dot{y}x_{<i}, \pi] \leq E[U|\dot{y}x_{<i}, a_\pi]$$

For a complete history $\dot{y}x_{<N+1} = \dot{y}x_{1:N}$ this holds trivially as

$$E[U|\dot{y}x_{1:N}, \pi] = U(\dot{y}x_{1:N}) = E[U|\dot{y}x_{1:N}, a_\pi]$$

for any agent a_π by theorem B.5.

Suppose we are given a fixed history $\dot{y}x_{<i}$ where $i \leq N$. By the inductive hypothesis we can construct an a such that $E[U|\dot{y}x_{<i}yx_i, \pi] \leq E[U|\dot{y}x_{<i}yx_i, a]$ holds for any given history $\dot{y}x_{<i}yx_i$. We can combine these function into a single agent a_π, since $E[U|\dot{y}x_{<i}yx_i, a]$ depends only on a evaluated at extensions of $\dot{y}x_{<i}yx_i$. Note that

$$E[U|\dot{y}x_{<i}y_i, \pi] \leq E[U|\dot{y}x_{<i}y_i, a_\pi]$$

additionally holds since:

$$\begin{aligned} \sum_{x_i} P(x_i|\dot{y}x_{<i}y_i)E[U|\dot{y}x_{<i}yx_i, a_\pi] &\leq \sum_{x_i} P(x_i|\dot{y}x_{<i}y_i)E[U|\dot{y}x_{<i}yx_i, \pi] \\ E[U|\dot{y}x_{<i}y_i, a_\pi] &\leq E[U|\dot{y}x_{<i}y_i, \pi] \end{aligned}$$

The first statement holds since $E[U|\dot{y}x_{<i}yx_i, a_\pi] \leq E[U|\dot{y}x_{<i}yx_i, \pi]$ for any history $\dot{y}x_{<i}yx_i$, the next by theorem B.5.

Consider:

$$\begin{aligned} E[U|\dot{y}x_{<i}, \pi] &\overset{(1)}{=} \sum_{y_i} \pi(y_i|\dot{y}x_{<i})E[U|\dot{y}x_{<i}y_i, \pi] \\ &\overset{(2)}{\leq} \sum_{y_i} \pi(y_i|\dot{y}x_{<i})E[U|\dot{y}x_{<i}y_i, a_\pi] \\ &\overset{(3)}{\leq} \max_{y_i} E[U|\dot{y}x_{<i}y_i, a_\pi] \\ &\overset{(4)}{=} E[U|\dot{y}x_{<i}, a_\pi] \end{aligned}$$

These steps follow by:

1. Theorem B.5 and the definition of nondeterministic agents π.

2. By construction of a_π.

3. Since $\max_x f(x)$ is larger than any convex combination $\sum_x w_x f(x)$:

$$\sum_x w_x f(x) \leq \sum_x w_x \left(\max_x f(x)\right) = \max_x f(x)$$

   the same holds for $f(y) = E[U|y\dot{x}_{<i}y, a_\pi]$. This will be equality iff π gives nonzero probability only to maximal actions; if any less than optimal action is given nonzero probability this inequality will be strict.

4. We add an additional requirement on a_π that

$$a_\pi(y\dot{x}_{<i}) \in \operatorname*{argmax}_{y_i} E[U|y\dot{x}_{<i}y_i, a_\pi]$$

   We can do this since we've only defined it on histories longer than $y\dot{x}_{<i}$ so far: it will still be the case that:

$$E[U|y\dot{x}_{<i}yx_i, \pi] \leq E[U|y\dot{x}_{<i}yx_i, a_\pi]$$

   since the value on the RHS depends only on a evaluated at histories longer than $y\dot{x}_{<i}$. Then, by theorem B.5 we get the equality.

Thus we can construct an agent a_π such that:

$$E[U|y\dot{x}_{<i}, \pi] \leq E[U|y\dot{x}_{<i}, a_\pi]$$

□

Bibliography

[Ber93] James O. Berger. *Statistical Decision Theory and Bayesian Analysis (Springer Series in Statistics)*. Springer, 1993.

[Cal02] Cristian S. Calude. *Information and Randomness: An Algorithmic Perspective*. Berlin, 2 edition, 2002.

[Gol04] Dina Goldin. Turing machines, transition systems, and interaction. *Information and Computation Journal*, 194:101–128, 2004.

[Han02] Robin Hanson. Disagreement is unpredictable. *Economics Letters*, 77(3):365–369, 2002.

[Har02] Michael Hardy. Scaled boolean algebras. *Advances in Applied Mathematics*, 2002. to Appear.

[HBH88] Eric J. Horvitz, John S. Breese, and Max Henrion. Decision theory in expert systems and artificial intelligence. *Int. J. Approx. Reasoning*, 2(3):247–302, 1988.

[Hut04] Marcus Hutter. *Universal Artificial Intelligence: Sequential Decisions based on Algorithmic Probability*. Springer, Berlin, 2004.

[Jay86] E. T. Jaynes. Bayesian methods: General background. In J. H. Justice, editor, *Maximum-Entropy and Bayesian Methods in Applied Statistics*. Cambridge Univ. Press, 1986.

[JB03] E. T. Jaynes and G. Larry Bretthorst. *Probability Theory : The Logic of Science*. Cambridge University Press, 2003.

[KLM96] L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[RN03] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, Englewood Cliffs, New Jersey, 2 edition, 2003.

[Rus97] Stuart J. Russell. Rationality and intelligence. *Artif. Intell.*, 94(1-2):57–77, 1997.

[SB98] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning)*. The MIT Press, 1998.