Agents Using Speed Priors

Daniel Filan

A subthesis submitted in partial fulfillment of the degree of
Bachelor of Philosophy (Honours) at
The Research School of Computer Science
Australian National University

October 2015

Typeset in Palatino by TEX and LATEX 2ε.

Except where otherwise indicated, this thesis is my own original work.

Daniel Filan
22 October 2015

Acknowledgements

This thesis could not have been completed without external help, at least without a significant reduction in quality. As such, I would like to thank the following people, organisations, and entities:

- Marcus Hutter, my supervisor. I would not be doing my honours program in computer science had he not agreed to let me do a reading course on AIXI without any prior computer science background, or more directly, had he not agreed to supervise me as an honours student. He also deserves credit for taking a chance and letting me study the speed prior, a speculative topic that carried the risk that I would not come up with anything. Finally, throughout this year he has given me advice on how to proceed with my research, both in the form of suggestions of questions to investigate, and help with proving theorems.
- Jan Leike, Mayank Daswani, and Tom Everitt, Marcus Hutter's PhD students. My discussions with them regularly helped me to clarify ideas, and often suggested things to prove and ways to prove them (for instance, Jan suggested a simpler way of proving Theorem 4.3 than what I had originally). Tom Everitt and Jan Leike were also kind enough to read over a draft of this thesis.
- The Machine Intelligence Research Institute. They funded the MIRIx workshop, run by Jan Leike, where I first came across the speed prior, and where the variant S_{Kt} which I explore in this thesis and the idea that it could predict polynomial-time computable sequences was introduced to me.
- Tor Lattimore and Laurent Orseau. An email conversation with them helped clarify an important proof to me, and unfortunately made me realise that I could not adapt it for use in this thesis.
- The anonymous reviewers of the Algorithmic Learning Theory 2015 conference. I submitted a paper comprising most of Chapters 3, 4, and 5 to that conference, and although my paper was rejected, the advice of the anonymous reviewers helped to develop this thesis.
- The Australian National University. They have generously given me the National Merit Scholarship. This helped support my stay at the ANU, and without it I would have had much less time to work on my studies and research.
- Last but not least, my parents, Susan Filan and Andy Filan. It barely needs to be said, but my entire scholastic career (not to speak of the rest of my life) is due to them.

Abstract

The speed prior S_{Fast} is a probability distribution over infinite binary sequences, favouring those sequences that are efficiently computable with short programs. This prior was introduced and shown to be computable in Schmidhuber [2002]. However, questions remain as to whether it succeeds at prediction or is efficiently computable. In this thesis, we show that S_{Fast} is efficiently computable when predicting an efficiently computable sequence, and bound its worst-case time complexity. We also introduce a new prior S_{Kt} and show that it makes few errors when predicting efficiently computable sequences, but that it is not efficiently computable itself. Finally, we investigate a generalisation of S_{Kt} in the reinforcement learning setting, and show that in certain cases it does very well.

Contents

Chapter 1

Introduction

In their famous textbook on the topic, Russell and Norvig [2009] define artificial intelligence (AI) as the field that "attempts not just to understand but also to *build* intelligent entities". However, this definition does not tell us what constitutes intelligence. Russell and Norvig give us four possible definitions:

- Thinking and reasoning like a human.
- Thinking and reasoning optimally, given what one knows.
- Behaving like a human.
- Behaving optimally, given what one knows.

We, like Russell and Norvig, concentrate on the fourth of these. A modification of this definition has been put forward as the common factor in an attempted comprehensive list of definitions of intelligence:

Definition 1.1 (Legg-Hutter intelligence (informal))**.** *Intelligence* measures an agent's ability to achieve goals in a wide range of environments [Legg and Hutter 2007].

As a formal framework to study intelligence, we consider two possible tasks: sequence prediction and reinforcement learning. In the sequence prediction task, our agent must predict a sequence that is stochastically generated independently of the agent's predictions. This essentially measures the agent's knowledge of its environment. However, our true interest is in reinforcement learning (RL), where an agent interacts with the environment and receives rewards depending on what has happened in those interactions. We encode the goals that we wish the agent to achieve in the rewards that we give it, and ask that the agent receive as high a reward sum as possible [Sutton and Barto 1998]. Although our main interest is in RL, seeing as it more obviously tests intelligence as defined in Definition 1.1, we study sequence prediction since it is simpler, and since we feel that learning one's environment is a good sign that one can act well in it.

Our general approach is a Bayesian one. The agent has some sort of probability distribution P over history sequences that it might encounter, and after history h, the

probability that the continued history will be h' is given by Bayes' rule:

$$P(h'|h) = \frac{P(h')}{P(h)} \tag{1.1}$$

One justification of this was given by Cox [1946], who showed that if our beliefs in propositions conditional on other propositions can be summarised in a function from propositions to real numbers, and if this function satisfies certain plausible conditions, then it is isomorphic to a probability function[1].

Another justification is based on Dutch Book arguments, which suppose that one values a bet that pays \$1 if proposition S is true and nothing otherwise at $\$P(S)$, where $P(S)$ is one's degree of belief in S. These arguments show that if one's degrees of belief violate the axioms of probability, then one would be willing to accept each of a collection of bets which, when taken together, guarantee the agent a sure loss (see Ramsey [1960, Chapter VII, Truth and Probability] and Teller [1973]). This is clearly problematic[2], and gives us extra motivation to be Bayesian.

As is made plain by (1.1), the beliefs of the agent of what may happen next are entirely determined by its prior probabilities $P(h)$. We are therefore driven to wonder how to set this prior. Occam's razor gives us a guide: we should give high probability to simple histories. However, we then need some definition of simplicity, or conversely, complexity.

Kolmogorov complexity formalises our intuitive understanding of complexity. Relative to some programming language (or universal Turing machine), the Kolmogorov complexity of string x is the length of the shortest program that prints x [Kolmogorov 1965; Li and Vitányi 2008]. Simple strings have Kolmogorov complexity that is shorter than their length: for instance, any programmer worth their salt should be able to write a short program that prints a string containing one million zeros. Structured objects also have low Kolmogorov complexity: for instance, one short program can print out arbitrarily many digits of π, which means that the digits of π have very low Kolmogorov complexity. Conversely, for a long random string with no interesting structure, the shortest program which prints the string likely needs to have the string hard-coded in, and therefore the Kolmogorov complexity is slightly greater than the length of the string.

In 1964, Ray Solomonoff came up with the idea of using a prior based on Kolmogorov complexity for sequence prediction [Solomonoff 1964a; Solomonoff 1964b]. The prior is defined thusly:

$$P(h) = \sum_{\text{programs } p \text{ that compute } h} 2^{-\text{length}(p)}$$

[1] It is worth pointing out that Cox's original proof was not entirely rigorous, but this has later been fixed by additionally assuming a technical axiom that does not change the flavour of the result [Van Horn 2003; Paris 2006, Chapter 3].

[2] The precise reason *why* it is problematic is something of a philosophical question. One could either think that a Dutch-Bookable agent would in fact accept a Dutch Book, which would lose them money, or that Dutch-Bookable beliefs motivate one to act in a knowably sub-optimal way, or that a Dutch-Bookable agent evaluates identical bets differently depending on how they are described [Talbott 2015].

or, to use a more compact notation that we adopt for the rest of this thesis,

$$P(h) = \sum_{p \to h} 2^{-|p|} \tag{1.2}$$

Since 2^{-x} decays quickly with x, this sum is essentially dominated by the shortest program, so $P(h) \approx 2^{-K(h)}$, or in other words, P is the probability distribution with respect to which the optimal codelength for history h is the Kolmogorov complexity of h [MacKay 2003, Section 5.3]. Another way of thinking about this result is that it gives the probability of obtaining h if we write a program by writing down i.i.d. coinflips (assuming that our programming language is in binary) [3].

It turns out that a predictor using the Solomonoff prior does very well. If the sequence is being produced by a deterministic program, then such a predictor only makes a finite number of errors, bounded by the length of the program (multiplied by a constant factor). If instead the sequence is stochastically generated according to a computable probability distribution, $E_{\text{Sol}}(t)$ is the expected number of errors the Solomonoff inductor would make up to time t, and $E_{\text{optimal}}(t)$ is the expected number of errors that the optimal predictor would make up to time t, then $E_{\text{Sol}}(t) - E_{\text{optimal}}(t) = O(\sqrt{E_{\text{optimal}}(t)})$—in other words, Solomonoff induction does not make many more errors than the optimal predictor. Furthermore, for any predictor, if $E(t)$ is the number of errors made by that predictor up to time t, then $E_{\text{Sol}}(t) - E(t) = O(\sqrt{E_{\text{Sol}}(t)})$—in other words, no other prediction scheme beats Solomonoff induction by much [Hutter 2005, Section 3.4].

One can also consider a reinforcement agent that uses the Solomonoff prior. This agent, typically called AIXI[4], is defined and investigated in Hutter [2005]. Although the sequence prediction results suggest that AIXI should perform well in any computable environment, it was recently discovered that all of the known optimality properties of AIXI are actually trivial or relative to the UTM used [Leike and Hutter 2015a]. Furthermore, we might wish that the agent's value-to-go (that is, the weighted sum of its expected future rewards) converges to the optimal agent's value-to-go given our agent's history in every environment. This is the idea behind the condition of weak asymptotic optimality. Unfortunately, it has been proven that AIXI fails to be weakly asymptotically optimal, even though weakly asymptotically optimal agents exist [Orseau 2013; Lattimore and Hutter 2011]. Therefore, although AIXI's optimality properties are trivial, its suboptimality properties are non-trivial. It does, however, have one desirable property: it is self-optimising in environment classes where any agent is self-optimising, if it is told that it is in that environment class. That is to say, if it is told that it is in one of a certain class of environments, and there is some agent such that that agent's value-to-go converges to the optimal value-to-go in each environment

[3] One small problem with Equation (1.2) is that it is not actually a probability distribution—the sum of the probabilities of continuations of h is actually less than the probability of h. Instead, we are dealing with a more general object called a semimeasure, from which we derive almost all of the utility of probabilities.

[4] The Solomonoff prior is often written as ξ. Therefore, an agent using the Solomonoff prior is an AI that uses ξ, or AIξ, or AIXI.

in the class, then AIXI's value-to-go also converges to the optimal value-to-go in each environment in the class, if it initially conditions on being in that class. In other words, the only way that AIXI can fail to have its value-to-go converge to optimum is for this to be impossible. Furthermore, this is a non-trivial optimality condition: environment classes where AIXI would be self-optimising exist, and include the interesting cases of ergodic Markov decision processes (that is, those where the agent knows what state of the environment it is in, and the probabilities for the next environmental state and reward depend only on the previous state and the agent's previous action) [Hutter 2005, Theorem 5.34].

Unfortunately, the Solomonoff prior is incomputable [Hutter 2005, Theorem 2.23]. This is essentially because to compute $P(h)$, we must know which programs print h, which requires a solution to the halting problem. We therefore cannot use Solomonoff induction to predict the weather or election winners.

In Schmidhuber [2002], a variant of the Solomonoff prior is introduced, called the speed prior. Instead of being based on Kolmogorov complexity, it is inspired by the Kt complexity measure, defined by

$$Kt(x) := \min\{|p| + \log(t) \mid p \text{ is a program which prints } x, \text{ and } t \text{ the time it takes for } p \text{ to print } x\}$$

Therefore, we penalise programs not just for being long, but for their time inefficiency. This has the desirable property of being computable: intuitively, the reason that we could not compute the Solomonoff prior was because there were short programs which take a long time to print the object whose prior we are computing, and therefore we cannot tell whether to include them in our sum (1.2) by simply waiting for some known length of time. However, these programs are essentially irrelevant to the speed prior, precisely because they take so long to compute.

Schmidhuber [2002] claims that this prior makes optimal predictions if our universe is a simulation run on some universal Turing machine, and derives some intriguing conclusions for physics from this assumption. However, we were unable to find in the literature any theorems about the performance of the speed prior in general environments, or about its computational complexity. In this thesis, we remedy this gap.

1.1 Thesis outline

In Chapter 2, we give a technical background to the fields of algorithmic information theory, Solomonoff induction, and reinforcement learning, providing context for the work of our thesis, before giving the definition of the speed prior in Schmidhuber [2002], which we call S_{Fast} (for reasons which become clear in that chapter).

These preliminaries being out of the way, we introduce our own speed prior S_{Kt} in Chapter 3, show that it also has the properties that make S_{Fast} a speed prior, and rewrite both priors in ways that demonstrate their similarities and differences.

After defining the speed priors, in Chapter 4 we prove error bounds for S_{Kt}-based

sequence prediction. Although we prove results for general stochastic sequences, the 'headline' result is that when predicting polynomial-time computable sequences, the S_{Kt}-based predictor only makes logarithmically many errors.

We next tackle the issue of computational complexity in Chapter 5, where we show that no approximation to $S_{Kt}(x)$ is computable in polynomial time, although approximations are computable in doubly-exponential time—that is, $2^{2^{O(|x|)}}$. S_{Fast}, however, does exponentially better, being computable in exponential time. We also show that these upper bounds reduce by an exponential factor when we are predicting a polynomial-time computable infinite string: that is, for prefixes of this string, S_{Kt} is computable in exponential time, and S_{Fast} is computable in polynomial time.

Moving away from the realm of sequence prediction, in Chapter 6 we investigate a Bayesian reinforcement learner with a prior based on S_{Kt}, and show that in certain environment classes it is self-optimising.

Finally, in Chapter 7, we conclude, summarising the thesis and indicating possible avenues of future research. Appendix A gives a list of the meanings of notation used throughout the thesis.

Chapter 2

Technical Background

2.1 Turing machines

Turing machines provide an idealised version of a computer. The basic idea is that the computer has some number of input tapes, some number of working tapes, and one output tape. The one-dimensional tapes are infinitely long, and are divided up into squares. Above each tape is a read/write head, which can read what is on the square it is over (unless it is over the output tape), write a possibly different symbol on the square (unless it is over one of the input tapes), and move from square to square. The machine also has an internal state, and a transition function, which takes the current state and symbols under the current heads, and returns an instruction to (possibly) write a symbol on the output tape, move the each head left or right, and change state. There is also a halting state, from which the transition function always does nothing and returns to the halting state, representing the ending of a computation. For a formal definition, see [Hopcroft et al. 2001, Section 8.2].

There are three types of Turing machine that we are concerned with. The first is a monotone machine: this has one input tape, and along both the input tape and the output tape, the head may only move in one direction (without loss of generality, we specify that it moves left to right). We say that a monotone machine T computes string x on input string p, and write $p \xrightarrow{T} x$, if T prints all of x when the input begins with p, and if when the last symbol of x is output, the input head has read all of p but no more. In particular, we do not require that the machine halt after printing x. This is the type of machine that we are typically concerned with.

The second type of machine, which as far as we know is unique to this thesis, we call a mixed-input machine. This machine has two input tapes, and similarly to monotone machines, along the input and output tapes, the head may only move right to left. We say that a mixed-input machine T computes string x on input tuple of strings (p, q), and write $(p, q) \xrightarrow{T} x$, if T prints all of x when the input on the first tape begins with p and the input on the second tape begins with q, and if when the last symbol of x is output, the input head on the first tape has read all of p but no more, and the input tape on the second head has read some prefix of q. Intuitively, the first input tape acts as it would in a prefix machine, and the second tape provides side information, which the machine does not have to use all of. We only use this type of machine in the context

of providing a formal definition of reinforcement learning environments.

The third type of machine is a prefix machine, which operates much like a monotone machine, except that we say that a prefix machine T computes string x on input p if T prints all of x and then halts having read all of p but no more. This type of machine, although typically used in the literature, will not be used much in this thesis.

Because we can encode all the details of a monotone or mixed-input Turing machine[1] in a finite binary string, there are only countably many, and we can therefore effectively enumerate them, writing each Turing machine as T_i for some $i \in \mathbb{N}$. This means that there is a universal Turing machine (UTM).

Theorem 2.1. There is a prefix-free coding $\langle i \rangle$ of the natural numbers and a universal monotone Turing machine U_{mon} such that $p \xrightarrow{T_i} x$ if and only if $\langle i \rangle p \xrightarrow{U_{\text{mon}}} x$, and U_{mon} halts after printing x on input $\langle i \rangle p$ if and only if T_i halts after printing x on input p. U_{mon} prints nothing given input that does not start with $\langle i \rangle$ for some i. The time it takes U_{mon} to output x on input $\langle i \rangle p$ is only polynomially larger than the time it takes T_i to print x on input p.

Also, there is a universal mixed-input machine U_{mix} such that $(p, q) \xrightarrow{T_i} x$ if and only if $(\langle i \rangle p, q) \xrightarrow{U_{\text{mix}}} x$, and U_{mix} halts after printing x on input $(\langle i \rangle p, q)$ if and only if T_i halts after printing x on input (p, q). U_{mix} prints nothing given input whose first tape does not start with $\langle i \rangle$ for some i. The time it takes U_{mon} to output x on input $(\langle i \rangle p, q)$ is only polynomially larger than the time it takes T_i to print x on input (p, q).

For proof, see Hopcroft, Motwani, and Ullman [2001, Section 9.2]. Throughout this thesis, we refer to the universal machine of each type as U, and trust the reader to infer which it is from context. We also typically write $p \to x$ for $p \xrightarrow{U} x$, since we are usually interested in the behaviour of the universal machine.

We write the set of binary characters as $\mathbb{B}$, and for any alphabet $\mathcal{X}$, we write the set of strings of length n in that alphabet as $\mathcal{X}^n$, the set of finite strings in that alphabet as $\mathcal{X}^*$, and the set of infinite strings in that alphabet as $\mathcal{X}^\infty$. A finite string of arbitrary length is written as x, the n^{th} symbol of x is written as x_n, the concatenation of finite string x with the possibly infinite string y is written xy, $x_{j:k}$ denotes $x_j x_{j+1} \cdots x_k$, $x_{<k}$ denotes $x_{1:k-1}$, and an infinite string is written $x_{1:\infty}$. If $k > t$, $x_{k:t} := \epsilon$, the empty string. All of our monotone Turing machines only read and write binary characters. Our mixed-input Turing machines read inputs $(p, q) \in \mathbb{B}^* \times \mathcal{A}^*$, where $\mathcal{A}$ is some arbitrary finite alphabet, and output symbols in $\mathcal{E}$, another arbitrary finite alphabet.

Also, throughout this thesis, if we have functions $f, g : \mathbb{N} \to \mathbb{R}$, we write $f \overset{\times}{\leq} g$ if $f(n) = O(g(n))$, and $f \overset{\times}{=} g$ if $f \overset{\times}{\leq} g$ and $g \overset{\times}{\leq} f$. For functions $f, g : \mathbb{B}^* \to \mathbb{R}$, we write $f \overset{\times}{\leq} g$ if there exists some constant $c > 0$ such that for all $x \in \mathbb{B}^*$, $f(x) < cg(x)$, and $f \overset{\times}{=} g$ if $f \overset{\times}{\leq} g$ and $g \overset{\times}{\leq} f$.

[1] That is, its number of tapes, number of states, number of symbols, transition function, and initial state.

2.2 Algorithmic information theory

Algorithmic information theory formalises the concept of 'simplicity' and 'complexity' of strings. The intuition is that simple strings should have a short description, or more specifically, have a short program that prints them. This is formalised in the definition of prefix Kolmogorov complexity, relative to Turing machine T:

Definition 2.2 (Prefix Kolmogorov complexity)**.**

$$K_T(x) := \min_p\{|p| : p \xrightarrow{T} x, \text{ and } T \text{ halts immediately after printing } x\}$$

That is, it is the shortest program that prints x on prefix machine T.

We can also consider prefix Kolmogorov complexity relative to the universal machine U. Since $p \xrightarrow{T_i} x$ with T_i then immediately halting if and only if $\langle i\rangle p \to x$ with U then immediately halting, $K_U(x) \leq K_{T_i}(x) + |\langle i\rangle|$. Therefore, complexities are almost lowest when measured on the universal Turing machine, but for an irrelevant additive constant. Furthermore, if there are two universal Turing machines U_1 and U_2, using two different codings of the natural numbers, then $|K_{U_1}(x) - K_{U_2}(x)| \leq c$ [Kolmogorov 1965].

Two variants of Kolmogorov complexity are be relevant in this thesis. The first variant is monotone Kolmogorov complexity, which refers to computation on monotone Turing machines [Levin 1973]:

Definition 2.3 (Monotone Kolmogorov complexity)**.**

$$Km_T(x) := \min_p\{|p| : p \xrightarrow{T} x\}$$

The second is Kt complexity, and is more significantly different. The intuition behind it is that if it is difficult to compute an object—that is, if requires much computation time—then the object is not simple, even if it has a short program. This can be formalised as [Li and Vitányi 2008, Definition 7.5.1]

Definition 2.4 (Kt complexity)**.**

$$Kt_T(x) := \min_p\{|p| + \log t(T,p,x) : p \xrightarrow{T} x, \text{ and } T \text{ takes } t(T,p,x) \text{ steps to print } x \text{ on input } p\}$$

log refers to the base 2 logarithm. We write ln for the natural logarithm.

With all complexities, for the rest of the thesis we take them with respect to the universal machine U, and write $K = K_U$, $Km = Km_U$, and $Kt = Kt_U$. We also write $t(p,x) := t(U,p,x)$.

Kt complexity is distinct from the others not only because it takes time into account, but also in that it is computable (a proof of which is deferred until Section 2.6), while K and Km are not [Hutter 2005, Theorem 2.13].

2.3 Solomonoff induction

Solomonoff induction is a method of sequence prediction that uses the Solomonoff prior, a probability distribution over sequences over a finite alphabet [Solomonoff 1964a]. The Solomonoff prior weights sequences with short programs more highly, and Solomonoff induction predicts the most likely continuation of the sequence seen so far with respect to the prior.

A probability distribution over sequences is a function P that takes a finite string, and returns the probability that the finite string is a prefix of the whole sequence. Intuitively, this function should satisfy $P(\epsilon) = 1$, where ϵ is the empty string, and $P(x) = \sum_{b \in \mathbb{B}} P(xb)$ for any finite binary string x. If P does satisfy these conditions, we call it a probability measure. However, we have to deal with objects that are slightly more general than measures:

Definition 2.5. A *semimeasure* is a function $P : \mathbb{B}^* \to \mathbb{R}$ which satisfies

$$P(\epsilon) = 1 \text{ and } P(x) \geq \sum_{b \in \mathbb{B}} P(xb)$$

A semimeasure can be thought of as a defective probability measure, which loses probability mass as the sequence gets longer and longer. Alternatively, it can be thought of a normal probability measure that puts positive probability on the sequence ending. Apart from 'named' semimeasures like the Solomonoff prior, we usually write semimeasures with Greek letters such as μ (which we reserve for the 'true' semimeasure of the environment), ν, and ρ.

Like probability measures, we can conditionalise semimeasures on strings, and write

$$\nu(x|y) = \frac{\nu(yx)}{\nu(y)}$$

Note that this conditional is not defined if $\nu(y) = 0$.

With this out of the way, we may define the Solomonoff prior.

Definition 2.6 (Solomonoff prior)**.**

$$M(x) := \sum_{p : p \to x} 2^{-|p|}$$

We claim that M is a semimeasure. Firstly, $\epsilon \to \epsilon$, so $M(\epsilon) = 2^0 = 1$. Secondly, for all p such that $p \to xb$, where $x \in \mathbb{B}^*$ and $b \in \mathbb{B}$, there is some prefix $q \sqsubseteq p$ (from here on, we write $q \sqsubseteq p$ to mean that q is a prefix of p, and $q \sqsubset p$ to mean that q is a proper prefix of p) such that $q \to x$. Furthermore, if $p \to xb$, then for no p' such that $p \sqsubseteq p'$ is it the case that $p' \to xb'$ for some $b' \in \mathbb{B}$. Therefore, if $q \to x$, $\{p : q \sqsubseteq p \text{ and } \exists b \in \mathbb{B} : p \to xb\}$ is prefix-free, and this remains true if we take the set

of $p_{(|q|+1):|p|}$, where we delete the prefix q from each program. This lets us compute

$$\begin{aligned}
\sum_{b\in\mathbb{B}} M(xb) &= \sum_{b\in\mathbb{B}} \sum_{p:p\to xb} 2^{-|p|} \\
&= \sum_{\substack{q:q\to x,\\ \exists b\in\mathbb{B},p\in\mathbb{B}^*:p\to xb}} \sum_{b\in\mathbb{B}} \sum_{p:q\sqsubset p,p\to xb} 2^{-|p|} \\
&= \sum_{\substack{q:q\to x,\\ \exists b\in\mathbb{B},p\in\mathbb{B}^*:p\to xb}} 2^{-|q|} \sum_{p:q\sqsubset p,\exists b\in\mathbb{B}:p\to xb} 2^{-|p_{(|q|+1):|p|}|} \\
&\leq \sum_{\substack{q:q\to x,\\ \exists b\in\mathbb{B},p\in\mathbb{B}^*:p\to xb}} 2^{-|q|} \qquad (2.1) \\
&\leq \sum_{q:q\to x} 2^{-|q|} \\
&= M(x)
\end{aligned}$$

where we use the Kraft inequality for (2.1).

It is worth noting that, since the function 2^{-k} decays so quickly, $M(x) \approx 2^{-Km(x)}$, relating monotone Kolmogorov complexity and the Solomonoff prior[2].

There is also a completely different definition of the Solomonoff prior. To give it, though, we must first go on a slight tangent.

Definition 2.7. A function $f : \mathbb{B}^* \times \mathbb{N} \to \mathbb{R}$ is *finitely computable* if there is some prefix Turing machine T that accepts a prefix coding of $x \in \mathbb{B}^*$ followed by a prefix coding of $k \in \mathbb{N}$ as input, and outputs a prefix coding of $n \in \mathbb{Z}$ followed by a prefix coding of $d \in \mathbb{N}$ such that $n/d = f(x,k)$. In cases like this, we simply say that T computes $f(x,k)$.

A function $f : \mathbb{B}^* \to \mathbb{R}$ is *lower semicomputable* if there is some finitely computable function $\phi : \mathbb{B}^* \times \mathbb{N} \to \mathbb{R}$ such that for all $x \in \mathbb{B}^*$, $\lim_{k\to\infty} \phi(x,k) = f(x)$, and $\phi(x,k) \leq \phi(x,k+1)$.

Intuitively, if f is a lower semicomputable function, then we can get an increasingly good approximation of $f(x)$ from below, but we do not necessarily know how good our approximation is.

We next state a useful theorem:

Theorem 2.8. ν is a lower semicomputable semimeasure if and only if there exists some monotone Turing machine T such that

$$\nu(x) = \sum_{p:p\xrightarrow{T}x} 2^{-|p|}$$

We say that ν is generated by T.

[2]However, this should not be taken literally, since although $2^{-Km(x)} \overset{\times}{\leq} M(x)$, $M(x) \neq O(2^{-Km(x)})$ [Gács 1983].

For proof, see Li and Vitányi [2008, Theorem 4.5.2]. This in hand, and writing $\mathcal{M}_{\text{LSS}}$ for the set of lower semicomputable semimeasures, we now show a rather different way of writing M:

Theorem 2.9.

$$M(x) = \sum_{\nu \in \mathcal{M}_{\text{LSS}}} w_\nu \nu(x)$$

where

$$w_\nu = \sum_{i:\, \nu \text{ is generated by } T_i} 2^{-|\langle i \rangle|}$$

Proof.

$$\begin{aligned}
M(x) &= \sum_{p: p \to x} 2^{-|p|} \\
&= \sum_{i \in \mathbb{N}} \sum_{q: \langle i \rangle q \to x} 2^{-|\langle i \rangle q|} \\
&= \sum_{i \in \mathbb{N}} 2^{-|\langle i \rangle|} \sum_{q: q \xrightarrow{T_i} x} 2^{-|q|} \\
&= \sum_{\nu \in \mathcal{M}_{\text{LSS}}} \sum_{i:\, \nu \text{ is generated by } T_i} 2^{-|\langle i \rangle|} \nu(x) \\
&= \sum_{\nu \in \mathcal{M}_{\text{LSS}}} w_\nu \nu(x)
\end{aligned}$$

□

This property of the Solomonoff prior, that it is a mixture over all lower semicomputable semimeasures (which, in practice, contains all probability distributions that we might care about learning), is actually what is used to show its optimality properties.

Suppose that we have some loss function $\ell(x_t, y_t)$ which represents the loss that our predictor incurs when predicting y_t if the actual next symbol is x_t. We define the Λ_ν predictor as the predictor which, at each timestep, minimises ν-expected loss, predicting

$$y_t^{\Lambda_\nu} := \operatorname*{argmin}_{y_t \in \mathbb{B}} \sum_{x_t \in \mathbb{B}} \nu(x_t | x_{<t}) \ell(x_t, y_t)$$

where the symbols previously seen are $x_{<t} := x_{1:t-1}$. If $\nu(x_{<t}) = 0$, we let Λ_ν predict an arbitrary symbol. We would like our predictor to have low μ-expected total loss, where μ is the true distribution of outputs. Up to time n, we write this total loss as

$$L_{n\mu}^{\Lambda_\nu} := \mathbb{E}_\mu \left[\sum_{t=1}^{n} \ell\left(x_t, y_t^{\Lambda_\nu}\right) \right]$$

We may now state our first optimality theorem for Solomonoff induction:

Theorem 2.10.

$$L_{n\mu}^{\Lambda_M} - L_{n\mu}^{\Lambda_\mu} \leq O\left(\sqrt{L_{n\mu}^{\Lambda_\mu}}\right)$$

and for any predictor Λ,

$$L_{n\mu}^{\Lambda_M} - L_{n\mu}^{\Lambda} \leq O\left(\sqrt{L_{n\mu}^{\Lambda_M}}\right)$$

In particular, if μ is deterministic, Λ_M only incurs finite loss.

For a proof, see Hutter [2005, Theorem 3.48]. This theorem essentially shows that Solomonoff induction does not do much worse than ideal prediction, and that no predictor does much better. Note that if we let $\ell(x_t, y_t) = 1 - \delta_{x_t,y_t}$, then the loss incurred by the predictor is simply the number of errors it makes, so the loss bound has an error bound as a special case.

We also have a Pareto optimality result for Solomonoff induction:

Definition 2.11. Let $\mathcal{F}(\mu, \nu)$ be a performance measure of priors ν in environments μ that we wish to be as small as possible. We call prior ν *Pareto optimal* in environment class $\mathcal{M}$ if there is no prior ρ such that for all $\mu \in \mathcal{M}$, $\mathcal{F}(\mu, \rho) \leq \mathcal{F}(\mu, \nu)$, and for at least one $\mu \in \mathcal{M}$, $\mathcal{F}(\mu, \rho) < \mathcal{F}(\nu, \nu)$. In other words, there must be no prior that is at least as good as ν in all environments in $\mathcal{M}$ and better in one.

Theorem 2.12 (Pareto optimality of Solomonoff induction)**.** M is Pareto optimal in the class of lower semicomputable semimeasures other than M with respect to the following performance measures:

$$\begin{aligned}
s_t(\mu, \nu) &= \sum_{x_t} \left(\mu(x_t|x_{<t}) - \nu(x_t|x_{<t})\right)^2 \\
S_n(\mu, \nu) &= \mathbb{E}_\mu \left[\sum_{t=1}^{n} s_t(\mu, \nu)\right] \\
d_t(\mu, \nu) &= \sum_{x_t} \mu(x_t|x_{<t}) \ln \frac{\mu(x_t|x_{<t})}{\nu(x_t|x_{<t})} \\
D_n(\mu, \nu) &= \mathbb{E}_\mu \left[\sum_{t=1}^{n} d_t(\mu, \nu)\right] \\
l_t(\mu, \nu) &= \mathbb{E}_\mu \left[\ell(x_t, y_t^{\Lambda_\nu}) \mid x_{<t}\right] \\
L_n(\mu, \nu) &= \mathbb{E}_\mu \left[\sum_{t=1}^{n} l_t(\mu, \nu)\right]
\end{aligned}$$

For proof, see Hutter [2005, Theorem 3.66]. Note that the theorem as stated there refers to priors $\xi_{\mathcal{M}} = \sum_{\nu \in \mathcal{M}} w_\nu \nu$ being Pareto optimal in $\mathcal{M}$. It would therefore apply to $\mathcal{M}_{\text{LSS}}$, in our case, which is trivial, since $M \in \mathcal{M}_{\text{LSS}}$, and presumably the predictor

that uses M performs the best in environment M. However, we note that

$$\begin{aligned} M(x) &= \sum_{\nu \in \mathcal{M}_{\text{LSS}}} w_\nu \nu(x) \\ &= w_M M(x) + \sum_{\nu \in \mathcal{M}_{\text{LSS}} \setminus \{M\}} w_\nu \nu(x) \\ &= \sum_{\nu \in \mathcal{M}_{\text{LSS}} \setminus \{M\}} \frac{w_\nu}{1 - w_M} \nu(x) \end{aligned}$$

Therefore, we can write M as a mixture of measures in $\mathcal{M}_{\text{LSS}} \setminus \{M\}$, which lets us apply the theorem.

2.4 Reinforcement learning

In reinforcement learning, an agent interacts with the environment in cycles: the agent takes some action, and the environment gives the agent some percept and reward. The aim of the agent is to maximise the amount of reward that it receives, and to have some strategy to do this with no prior knowledge of the environment.

More precisely, our setup is the agent model, as defined in [Sutton and Barto 1998, Chapter 3]. We have an agent situated in an environment, interacting in cycles. In cycle k, the agent outputs some action $a_k \in \mathcal{A}$, which the agent can choose however it likes, and the environment outputs some percept $e_k \in \mathcal{E}$, which can be decomposed into an observation $o_k \in \mathcal{O}$ and reward $r_k \in \mathcal{R} \subset [0, 1]$.

The agent is defined by a policy $\pi : (\mathcal{A} \times \mathcal{E})^* \to \mathcal{A}$ that determines the action that the agent takes given the prior history.

To explain how we define the environment, a new definition is required.

Definition 2.13. A *conditional semimeasure* is a function $\nu : \mathcal{E}^* \times \mathcal{A}^\infty \to \mathbb{R}$ such that for all action sequences $a_{1:\infty}$,

$$\nu(\epsilon || a_{1:\infty}) = 1 \text{ and } \nu(e_{1:t} || a_{1:\infty}) \geq \sum_{e_{t+1} \in \mathcal{E}} \nu(e_{1:t} e_{t+1} || a_{1:\infty})$$

A conditional semimeasure is *chronological* if $\nu(e_{1:t} || a_{1:\infty}) = \nu(e_{1:t} || a'_{1:\infty})$ whenever $a_{1:t} = a'_{1:t}$. Therefore, if ν is a chronological conditional semimeasure, we may write $\nu(e_{1:t} || a_{1:\infty})$ simply as $\nu(e_{1:t} || a_{1:k})$ for any $k \geq t$. We conditionalise on chronological conditional semimeasures, writing

$$\nu(e_{k:t} | e_{<k} || a_{1:\infty}) = \frac{\nu(e_{1:t} || a_{1:\infty})}{\nu(e_{<k} || a_{1:\infty})}$$

The environment is defined by a chronological conditional semimeasure μ. This semimeasure gives the probabilities of the environment's responses given some action history, and the chronological condition guarantees that the environmental response at time t does not depend on actions performed after t. Also, for convenience, we write an element of $(\mathcal{A} \times \mathcal{E})^*$ as $æ_{1:k} := a_1 e_1 a_2 e_2 \cdots a_k e_k$.

The interaction could either last for a finite time, or for an infinite time. In the finite case, we wish to judge the agent by its expected total reward. There are two ways of doing this: one takes expected total reward over all histories of length m, where m is the lifetime of the agent. This would give

$$V_{1m}^{\pi\mu} = \sum_{e_{1:m}} (r_1 + \cdots + r_m)\mu(e_{1:m}||a_{1:m})$$

where $a_k = \pi(æ_{<k})$.

In the infinite lifetime case, we let the agent look m_k timesteps ahead at time k. So, at each timestep k, it has some policy π_k which would have value

$$V_{km_k}^{\pi_k\mu} = \sum_{e_{k:m_k}} (r_k + \cdots + r_{m_k})\mu(e_{k:m_k}|e_{<k}||a_{1:m_k})$$

where for $t \geq k$, $a_t = \pi_k(æ_{<t})$ (note that $V_{km_k}^{\pi_k\mu}$ actually depends on the interaction history $æ_{<k}$, a dependence which we omit for succinctness). This agent acts according to π_k at time k, and plans to do so in the future, but at time $k+1$ it will act according to π_{k+1} which may be different to π_k—in particular, if the agent chooses the policy which maximises $V_{km_k}^{\pi_k\nu}$ for some chronological conditional semimeasure ν, this may be different for each timestep k if some actions receive high enough reward between m_k and m_{k+1}. Therefore, $V_{km_k}^{\pi_k\mu}$ represents the value the agent would receive if it stuck to its plan, rather than the value the agent actually does receive[3].

A different approach to working with agents with infinite lifetime is to discount reward received at timestep k by some factor γ_k, so that $\sum_{k=1}^{\infty} \gamma_k < \infty$. This gives a value function

$$V_{k\gamma}^{\pi\mu} = \frac{1}{\Gamma_k} \lim_{m\to\infty} \sum_{æ_{k:m}} (\gamma_k r_k + \cdots + \gamma_m r_m)\mu(e_{k:m}|e_{<k}||a_{1:m})$$

where $a_t = \pi(æ_{<t})$ and $\Gamma_k = \sum_{t=k}^{\infty} \gamma_k$. This scheme, with $\gamma_k = \gamma^k$ for some $\gamma \in (0,1)$, is most typically used in reinforcement learning, since it looks at the entire performance of the agent and does not incentivise agents to switch policy [Sutton and Barto 1998], but we do not use it much in this thesis.

2.5 AIXI

AIXI is a reinforcement learner that is based on an analogue of the Solomonoff prior for chronological conditional semimeasures:

Definition 2.14 (Solomonoff conditional semimeasure)**.**

$$M(e_{1:t}||a_{1:\infty}) := \sum_{p:(p,a_{1:t})\to e_{1:t}} 2^{-|p|}$$

[3]Note that this requires a broadening of the notion of an agent to a family of policies, one at each timestep.

Note that we use the same symbol for the conditional semimeasure $M(e_{1:t}||a_{1:\infty})$ as for the semimeasure $M(x)$. This should not be too confusing, since we typically include the arguments to each function, and they are used in different contexts ($M(x)$ in the context of sequence prediction, and $M(e_{1:t}||a_{1:\infty})$ in the context of reinforcement learning).

We claim that M is a chronological conditional semimeasure. Firstly, it is clear that $M(e_{1:t}||a_{1:\infty})$ only depends on $a_{1:t}$. Secondly, since $(\epsilon, \epsilon) \to \epsilon$, $M(\epsilon||a_{1:\infty}) = 1$. Thirdly, if $(p, a_{1:t}) \to e_{1:t}$, then by our definition of computation on mixed-tape machines, $(p, a_{1:t+1}) \to e_{1:t}$, so the rest of the proof goes identically to that of proving that the Solomonoff prior is a semimeasure.

Next, we give two theorems that closely relate our theory of chronological conditional semimeasures to that of plain semimeasures:

Theorem 2.15. The function ν is a lower semicomputable chronological conditional semimeasure (LSCCS) if and only if there exists some mixed-input Turing machine T such that

$$\nu(e_{1:t}||a_{1:\infty}) = \sum_{p:(p,a_{1:t}) \xrightarrow{T} e_{1:t}} 2^{-|p|}$$

We say that ν is generated by T.

Note that for this theorem to be rigorous, we must think of the input to ν as actually being $æ_{1:t}$ and extend our notion of lower semicomputability to functions $(\mathcal{A} \times \mathcal{E})^* \to \mathbb{R}$. The proof is essentially identical to that of Theorem 2.8.

Theorem 2.16. Let $\mathcal{M}_{\text{LSCCS}}$ be the class of lower semicomputable chronological conditional semimeasures. Then,

$$M(e_{1:t}||a_{1:\infty}) = \sum_{\nu \in \mathcal{M}_{\text{LSCCS}}} w_\nu \nu(e_{1:t}||a_{1:\infty})$$

where

$$w_\nu = \sum_{i:\ \nu \text{ is generated by } T_i} 2^{-|\langle i \rangle|}$$

Again, the proof is essentially identical to that of Theorem 2.9.

We define AIXI as the policy which maximises $V_{1m}^{\pi M}$ or $V_{km_k}^{\pi_k M}$. In the finite lifetime case we write it as π^M, and in the infinite lifetime case we write it as π_k^M at timestep k. Given the success of Solomonoff induction, we might suspect that AIXI performs well in all environments in $\mathcal{M}_{\text{LSCCS}}$. Indeed, we can use the Solomonoff prediction results to show that AIXI capably predicts the next percepts along its action history. However, this does not suffice: for good performance, the agent needs to predict what will happen given actions that the agent may not take.

AIXI is known to be Pareto optimal: that is, there is no policy that does at least as good as AIXI in all environments in $\mathcal{M}_{\text{LSCCS}} \setminus \{M\}$ and strictly better in one, as measured by total value gained [Hutter 2005, Theorem 5.23]. However, it was recently

proved that any policy is Pareto optimal in $\mathcal{M}_{\text{LSCCS}} \setminus \{M\}$, rendering Pareto optimality trivial[4] [Leike and Hutter 2015a].

Furthermore, AIXI is non-trivially sub-optimal. We might wish that for every environment μ where the agent has discounted infinite lifetime, the future value of agent π converges to that of the optimal policy π^μ given the interaction history of π, meaning that as time goes on, the agent makes fewer and fewer mistakes:

$$\forall \mu \in \mathcal{M}_{\text{LSCCS}} \ \lim_{k\to\infty} \left(V_{k\gamma}^{\pi^\mu \mu} - V_{k\gamma}^{\pi\mu} \right) = 0 \text{ with } \mu\text{-probability 1} \tag{2.2}$$

We might settle for this convergence 'on average':

$$\forall \mu \in \mathcal{M}_{\text{LSCCS}} \ \lim_{k\to\infty} \frac{1}{k} \sum_{t=1}^{k} \left(V_{t\gamma}^{\pi^\mu \mu} - V_{t\gamma}^{\pi\mu} \right) = 0 \text{ with } \mu\text{-probability 1} \tag{2.3}$$

Note that the probability quantifiers refer to the environmental sequence, which affects the agent's actions. In Lattimore and Hutter [2011], it is shown that no agent satisfies (2.2), but that some agents satisfy (2.3). However, AIXI does not satisfy (2.3) when the discounting is computable: in some bandit-like environments, it stops exploring, causing it to be non-trivially sub-optimal by this measure [Orseau 2013].

However, a generalisation of AIXI does have non-trivial optimality properties. For any class of environments $\mathcal{M}$, we can define the Bayesian mixture

$$\xi_{\mathcal{M}} = \sum_{\nu\in\mathcal{M}} w_\nu \nu \text{ where } w_\nu > 0, \ \sum_{\nu\in\mathcal{M}} w_\nu \leq 1$$

This generalisation proves to have an interesting optimality property in some environment classes.

Definition 2.17. In the finite lifetime setting, a sequence of policies π_m is *self-optimising* in environment class $\mathcal{M}$ if for all $\nu \in \mathcal{M}$,

$$\lim_{m\to\infty} \left(\frac{1}{m} V_{1m}^{\pi_m \nu} - \frac{1}{m} V_{1m}^{\pi^\nu \nu} \right) = 0$$

That is to say, the average reward of π_m approaches the average reward of π^ν as the lifetime approaches infinity.

In the discounted infinite lifetime setting, a policy π is *self-optimising* in environment class $\mathcal{M}$ if for all $\nu \in \mathcal{M}$,

$$\lim_{k\to\infty} \left(V_{k\gamma}^{\pi\nu} - V_{k\gamma}^{\pi^\nu \nu} \right) = 0 \text{ with } \nu\text{-probability 1}$$

That is to say, the value-to-go of π almost surely approaches the value-to-go of π^ν (given the action history of π) as time goes to infinity. Note that these two definitions are analogous, since $V_{1m}^{\pi_m \nu} \in [0, m]$, while $V_{k\gamma}^{\pi\nu} \in [0, 1]$. In other words, in the finite

[4] In fact, the theorem showed that Pareto optimality was trivial in any environment containing all POMDPs. POMDPs, or Partially Observable Markov Decision Processes, are defined in Murphy [2000].

lifetime setting, we average over past reward, while in the discounted infinite lifetime setting, we have already averaged over future reward in the definition of $V_{k\gamma}^{\pi\nu}$.

The sequence of policies $\pi_m^{\xi,\mathcal{M}}$ that maximise $V_{1m}^{\pi\xi,\mathcal{M}}$ is self-optimising whenever any sequence of policies is self-optimising in $\mathcal{M}$ in the finite lifetime setting, and the policy $\pi^{\xi,\mathcal{M}}$ is self-optimising whenever any policy is self-optimising in $\mathcal{M}$ in the discounted infinite lifetime setting. Therefore, $\xi_{\mathcal{M}}$ is 'as self-optimising as possible' [Hutter 2005, Theorems 5.29, 5.34]. Furthermore, this is a non-trivial optimality condition: the class of ergodic MDPs admits self-optimising policies (both for finite lifetime and for discounted infinite lifetime whenever $\lim_{k\to\infty} \gamma_{k+1}/\gamma_k = 1$).

2.6 Schmidhuber's speed prior

Loosely speaking, the theory of Solomonoff induction starts with an incomputable complexity measure, formalises Occam's razor with respect to that complexity measure, and results in a prior that is incomputable but good at prediction. We may therefore wonder if we could start with the computable Kt complexity measure, formulate a prior that rewards simplicity with respect to this measure, and get good results from that. This line of reasoning is the inspiration for the definition of the speed prior.

To define the speed prior as it appears in Schmidhuber [2002], we must first define the FAST algorithm with respect to our reference monotone UTM U. For each $i \in \mathbb{N}$, FAST performs PHASE i, whereby $2^{i-|p|}$ instructions of all programs satisfying $|p| \leq i$ are executed as they would be on U, and the outputs are sequentially printed on adjacent sections of the output tape, separated by blanks. If string x is computed by program p in PHASE i, then we write $p \to_i x$. Note that $p \to_i x$ iff $|p| + \log t(p, x) \leq i$, showing the relation to Kt complexity[5]. With this out of the way, we can now give the definition.

Definition 2.18 (Schmidhuber's speed prior)**.**

$$S_{\text{Fast}}(x) := \sum_{i=1}^{\infty} 2^{-i} \sum_{p \to_i x} 2^{-|p|}$$

Note that S_{Fast} is a semimeasure.

Due to the properties of FAST, this prior penalises strings with high Kt complexity. Two properties of S_{Fast} make this clear. The first is that the prior probability with respect to S_{Fast} of all strings incomputable within time t is at most $1/t$. The second is more complex:

Theorem 2.19. Let $x_{1:\infty} \in \mathbb{B}^\infty$. Suppose that $p^x \in \mathbb{B}^*$ outputs $x_{1:n}$ within at most $f(n)$

[5] In fact, this provides a way of calculating the Kt complexity of a string x to the nearest integer: simply run FAST until the PHASE where x is first printed, and estimate $Kt(x)$ by the number of that PHASE. There is an upper bound on how long this takes, since for any x there is the 'print x' program which has length $|x| + O(1)$ and takes time $O(|x|)$ to print, meaning that it outputs x in PHASE $|x| + \log |x| + O(1)$.

steps for all n, and $\lim_{n\to\infty} f(n)/g(n) = 0$. Then,

$$\lim_{n\to\infty} \frac{\sum_{i=1}^{\infty} 2^{-i} \sum_{p \xrightarrow[i,\geq g(n)]{} x_{1:n}} 2^{-|p|}}{\sum_{i=1}^{\infty} 2^{-i} \sum_{p \xrightarrow[i,\leq f(n)]{} x_{1:n}} 2^{-|p|}} = 0$$

where $p \xrightarrow[i,\leq f(n)]{} x_{1:n}$ (respectively, $p \xrightarrow[i,\geq g(n)]{} x_{1:n}$) means that p computes $x_{1:n}$ in PHASE i within at most $f(n)$ (respectively, at least $g(n)$) timesteps.

In other words, if some program computes $x_{1:n}$ in time $f(n)$, then the contribution to S_{Fast} by programs that take much longer than $f(n)$ goes to 0. In this thesis, we will call any prior which satisfies these two properties a speed prior.

Furthermore, S_{Fast} is computable: if we wish to compute $S_{\text{Fast}}(x)$ to within absolute accuracy of 2^{-i} (that is, to get a result that is no more than 2^{-i} less than the actual value of $S_{\text{Fast}}(x)$), all we need do is perform the first i PHASEs of FAST, and add up all of the contributions to S_{Fast} found in those PHASEs.

Now that S_{Fast} has been defined, this thesis investigates it and a different prior with similar properties, aiming to determine whether they are efficiently computable, whether they are successful at sequence prediction, and whether they can be used for reinforcement learning.

Chapter 3

The Other Speed Prior

3.1 Definition of S_{Kt}

In this section, we introduce a prior that penalises strings of high Kt complexity more directly. In analogy with the Solomonoff prior $M(x) = \sum_{p \to x} 2^{-|p|} = \sum_{p \to x} 2^{-Km\text{-cost}(p,x)}$, where

$$Km\text{-cost}(p,x) := \begin{cases} |p| & \text{if } p \to x \\ \infty & \text{otherwise} \end{cases}$$

is the minimand of monotone Kolmogorov complexity, we define the Kt-cost of a computation of a string x by program p as the minimand of Kt, that is,

$$Kt\text{-cost}(p,x) := |p| + \log t(p,x)$$

(where if $p \not\to x$, then $t(p,x) := \infty$). We then define our speed prior as

Definition 3.1 (Our speed prior)**.**

$$S_{Kt}(x) := \sum_{p \to x} 2^{-Kt\text{-cost}(p,x)} = \sum_{p \to x} \frac{2^{-|p|}}{t(p,x)}$$

This is also a semimeasure.

3.2 Comparable definitions

Definitions 2.18 of S_{Fast} and 3.1 of S_{Kt} have been given in different forms—the first in terms of PHASEs of FAST, and the second in terms of Kt-cost. In this section, we show that each can be rewritten in a form similar to the other's definition, which should shed light on the differences and similarities between the two. Our rewriting of S_{Fast} is used later to bound its computation time.

Proposition 3.2.

$$S_{\text{Fast}}(x) \stackrel{\times}{=} \sum_{p \to x} \frac{2^{-2|p|}}{t(p,x)}$$

The idea of the proof, and also of the proof of Proposition 3.3, is that if $p \to_i x$ then $i \approx |p| + \log t(p,x)$, so $2^{-i}2^{-|p|} \approx 2^{-2|p|}/t(p,x)$.

Proof. First, we note that for each program p and string x, if $p \to_i x$, then for all $j \geq i$, $p \to_j x$. Now,

$$\sum_{j=i}^{\infty} 2^{-j} \times 2^{-|p|} = 2 \times 2^{-i} \times 2^{-|p|}$$

$$\Rightarrow \sum_{i=1}^{\infty} 2^{-i} \sum_{p \to_i x} 2^{-|p|} \stackrel{\times}{=} \sum_{i=1}^{\infty} 2^{-i} \sum_{\substack{p \to_i x \\ p \not\to_{i-1} x}} 2^{-|p|} \tag{3.1}$$

since all of the contributions to $S_{\text{Fast}}(x)$ from program p in phases $j \geq i$ add up to twice the contribution from p in PHASE i alone.

Next, suppose $p \to_i x$. Then, by the definition of FAST,

$$t(p,x) \leq 2^{i-|p|} \quad \Leftrightarrow \quad \log t(p,x) \leq i - |p| \quad \Leftrightarrow \quad |p| + \log t(p,x) \leq i$$

Also, if $p \not\to_{i-1} x$, then either $|p| > i-1$, implying $|p| + \log t(p,x) > i-1$, or $t(p,x) > 2^{i-1-|p|}$, also implying $|p| + \log t(p,x) > i-1$. Therefore, if $p \to_i x$ and $p \not\to_{i-1} x$, then

$$i - 1 < |p| + \log t(p,x) \leq i$$

implying

$$-|p| - \log t(p,x) - 1 < -i \leq -|p| - \log t(p,x) \tag{3.2}$$

Subtracting $|p|$ and exponentiating yields

$$\frac{1}{2t(p,x)} 2^{-2|p|} \leq 2^{-i-|p|} \leq \frac{2^{-2|p|}}{t(p,x)}$$

giving

$$2^{-i-|p|} \stackrel{\times}{=} \frac{2^{-2|p|}}{t(p,x)}$$

Therefore,

$$\sum_{i=1}^{\infty} 2^{-i} \sum_{\substack{p \to_i x \\ p \not\to_{i-1} x}} 2^{-|p|} \stackrel{\times}{=} \sum_{p \to x} \frac{2^{-2|p|}}{t(p,x)}$$

which, together with equation (3.1), proves the proposition. □

Proposition 3.3.

$$S_{Kt}(x) \stackrel{\times}{=} \sum_{i=1}^{\infty} 2^{-i} \left(\#\{p \in \mathbb{B}^* : p \to_i x \text{ and } p \not\to_{i-1} x\}\right)$$

Proof. Using equation (3.2), we have that if $p \to_i x$ and $p \not\to_{i-1} x$, then

$$\frac{2^{-|p|-1}}{t(p,x)} \leq 2^{-i} \leq \frac{2^{-|p|}}{t(p,x)}$$

so

$$2^{-i} \stackrel{\times}{=} \frac{2^{-|p|}}{t(p,x)}$$

Summing over all programs p such that $p \to_i x$ and $p \not\to_{i-1} x$, we have

$$2^{-i} \left(\#\{p \in \mathbb{B}^* : p \to_i x \text{ and } p \not\to_{i-1} x\}\right) \stackrel{\times}{=} \sum_{\substack{p \to_i x, \\ p \not\to_{i-1} x}} \frac{2^{-|p|}}{t(p,x)}$$

Then, summing over all phases i, we have

$$\sum_{i=1}^{\infty} 2^{-i} \left(\#\{p \in \mathbb{B}^* : p \to_i x \text{ and } p \not\to_{i-1} x\}\right) \stackrel{\times}{=} \sum_{p \to x} \frac{2^{-|p|}}{t(p,x)}$$

□

3.3 S_{Kt} is a speed prior

Although we have defined S_{Kt}, we have not shown any results that indicate it deserves to be called a speed prior. As explained in Section 2.6, two key properties of S_{Fast} justify its description as a speed prior: firstly, that the cumulative prior probability measure of all x incomputable in time t is at most inversely proportional to t, and secondly, that if $x_{1:\infty} \in \mathbb{B}^\infty$ and program $p^x \in \mathbb{B}^*$ computes $x_{1:n}$ within at most $f(n)$ steps, then the contribution to $S_{\text{Fast}}(x_{1:n})$ by programs that take time much longer than $f(n)$ vanishes as $n \to \infty$. In this subsection, we prove that both of these properties also hold for S_{Kt}.

Proposition 3.4. Let $\mathcal{C}_t$ be the set of strings x that are incomputable in time t that additionally satisfy the following property: for any proper prefix $y \sqsubset x$, y is computable in time t. By definition, all strings that are incomputable in time t have as a prefix an element of $\mathcal{C}_t$, and $\mathcal{C}_t$ is a prefix-free set (by construction). Furthermore, the probability of all strings incomputable in time t is simply the sum of the probabilities of all elements of $\mathcal{C}_t$. Given this definition,

$$\sum_{x \in \mathcal{C}_t} S_{Kt}(x) \leq \frac{1}{t}$$

Proof. Using the definition of S_{Kt} and that $t(p,x) \geq t$ for all $x \in \mathcal{C}_t$, we calculate

$$\sum_{x \in \mathcal{C}_t} S_{Kt}(x) = \sum_{x \in \mathcal{C}_t} \sum_{p \to x} \frac{2^{-|p|}}{t(p,x)} \leq \frac{1}{t} \sum_{x \in \mathcal{C}_t} \sum_{p \to x} 2^{-|p|} \leq \frac{1}{t}$$

by the Kraft inequality, since the fact that $\mathcal{C}_t$ is a prefix-free set guarantees that the set of

programs that compute elements of $\mathcal{C}_t$ is also prefix-free, due to our use of monotone machines. □

Theorem 3.5. Let $x_{1:\infty} \in \mathbb{B}^\infty$ be such that that there exists a program $p^x \in \mathbb{B}^*$ which outputs $x_{1:n}$ in $f(n)$ steps for all $n \in \mathbb{N}$. Let $\lim_{n\to\infty} f(n)/g(n) = 0$. Then,

$$\lim_{n\to\infty} \frac{\sum_{p \xrightarrow[\geq g(n)]{} x_{1:n}} 2^{-|p|}/t(p, x_{1:n})}{\sum_{p \xrightarrow[\leq f(n)]{} x_{1:n}} 2^{-|p|}/t(p, x_{1:n})} = 0$$

where $p \underset{\leq t}{\longrightarrow} x$ (respectively, $p \underset{\geq t'}{\longrightarrow} x$) means that program p computes string x in no more than t (respectively, no less than t') steps.

Proof.

$$\begin{aligned}
&\lim_{n\to\infty} \frac{\sum_{p \xrightarrow[\geq g(n)]{} x_{1:n}} 2^{-|p|}/t(p, x_{1:n})}{\sum_{p \xrightarrow[\leq f(n)]{} x_{1:n}} 2^{-|p|}/t(p, x_{1:n})} \\
&\leq \lim_{n\to\infty} \frac{\sum_{p \xrightarrow[\geq g(n)]{} x_{1:n}} 2^{-|p|}/g(n)}{2^{-|p^x|}/f(n)} && (3.3) \\
&\leq \lim_{n\to\infty} \frac{f(n)}{g(n)} \frac{\sum_{p \to x_{1:n}} 2^{-|p|}}{2^{-|p^x|}} && (3.4) \\
&\leq \lim_{n\to\infty} \frac{f(n)}{g(n)} \frac{1}{2^{-|p^x|}} && (3.5) \\
&= 0
\end{aligned}$$

Equation (3.3) comes from increasing $1/t(p, x_{1:n})$ to $1/g(n)$ in the numerator, and decreasing the denominator by throwing out all terms of the sum except that of p^x, which takes $f(n)$ time to compute $x_{1:n}$. Equation (3.4) takes $f(n)/g(n)$ out of the fraction, and increases the numerator by adding contributions from all programs that compute $x_{1:n}$. Equation (3.5) uses the Kraft inequality to bound $\sum_{p \to x_{1:n}} 2^{-|p|}$ from above by 1. Finally, we use the fact that $\lim_{n\to\infty} f(n)/g(n) = 0$. □

Chapter 4

Prediction

4.1 Polynomial time estimable measures

In this section, we prove a performance bound on S_{Kt}-based sequence prediction, when predicting a sequence drawn from a measure that is estimable in polynomial time. Since we were unable to prove a similar bound for S_{Fast}, this provides some weak evidence that S_{Kt} is better at prediction than S_{Fast}.

For the purpose of this section, we write S_{Kt} somewhat more explicitly as

$$S_{Kt}(x) = \sum_{p \xrightarrow{U} x} \frac{2^{-|p|}}{t(U, p, x)}$$

and give some auxiliary definitions. Let $\langle\cdot\rangle_{\mathbb{B}^*}$ be a prefix-free coding of the strings of finite length and $\langle\cdot\rangle_{\mathbb{N}}$ be a prefix-free coding of the integers, where both of these prefix-free codings are computable and decodable in polynomial time.

Definition 4.1. A function $f : \mathbb{B}^* \to \mathbb{R}$ is *finitely computable* if there is some prefix Turing machine T_f that when given input $\langle x\rangle_{\mathbb{B}^*}$ outputs $\langle m\rangle_{\mathbb{N}}\langle n\rangle_{\mathbb{N}}$, where $f(x) = m/n$. The function f is *finitely computable in polynomial time* if it takes T_f at most $p(|x|)$ timesteps to halt on input x, where p is a polynomial.

Definition 4.2. Let $f, g : \mathbb{B}^* \to \mathbb{R}$. The function g is *estimable in polynomial time by* f if f is finitely computable in polynomial time and $f(x) \stackrel{\times}{=} g(x)$. The function g is *estimable in polynomial time* if it is estimable in polynomial time by some function f.

Note that if f is finitely computable in polynomial time, it is estimable in polynomial time by itself. For a measure μ, estimability in polynomial time captures our intuitive notion of efficient computability: we only need to know μ up to a constant factor for all practical purposes, and we can find this out in polynomial time.

An example of a measure that is estimable in polynomial time is μ_π, which gives probability $2/3$ that the i^{th} bit will be the i^{th} digit of the binary expansion of the circle constant π, and $1/3$ otherwise. For instance, since $\pi = 11.0010\cdots$ in binary, $\mu_\pi(11001) = (2/3)^5$, while $\mu_\pi(100110) = (2/3)^4 \times (1/3)^2$. This is estimable in polynomial time because we can calculate the i^{th} digit of π in polynomial time [Bailey et al. 1997], and it only takes polynomial time to multiply numbers.

Key to our results, both in this chapter and in Chapter 5, will be that for measures μ estimable in polynomial time by semimeasures,

$$S_{Kt}(x) \overset{\times}{\geq} \frac{\mu(x)}{(g(|x|) - O(1)\log \mu(x))^{O(1)}}$$

and

$$S_{\text{Fast}}(x) \overset{\times}{\geq} \frac{\mu(x)^2}{(g(|x|) - O(1)\log \mu(x))^{O(1)}}$$

Compare this to the simpler dominance relation for the Solomonoff prior, $M(x) \overset{\times}{\geq} \mu(x)$.

Theorem 4.3. If μ is a measure that is estimable in polynomial time by some semimeasure ν, and x is a sequence sampled from μ, then the expected loss incurred by the $\Lambda_{S_{Kt}}$ predictor is bounded by

$$L_{n\mu}^{\Lambda_{S_{Kt}}} - L_{n\mu}^{\Lambda_\mu} \leq 2D_n + 2\sqrt{L_{n\mu}^{\Lambda_\mu} D_n}$$

where $D_n = O(\log n)$.[1]

Since Λ_μ can incur at most $O(n)$ loss in timesteps 1 to n, this means that $\Lambda_{S_{Kt}}$ only incurs at most $O(\sqrt{n \log n})$ extra loss in expectation, although this bound is be much tighter in more structured environments where Λ_μ makes few errors.

In order to prove this theorem, we use the following lemma:

Lemma 4.4. Let ν be a semimeasure that is finitely computable in polynomial time. There exists a Turing machine T_ν such that for all $x \in \mathbb{B}^*$

$$\nu(x) = \sum_{p \xrightarrow{T_\nu} x} 2^{-|p|} \tag{4.1}$$

and

$$2^{-Km_{T_\nu}(x)} \geq \nu(x)/4 \tag{4.2}$$

where $Km_{T_\nu}(x)$ is the length of the shortest program for x on T_ν.[2]

Note that there is a proof in Li and Vitányi that there is some machine T_ν such that (4.1) holds [Li and Vitányi 2008, Theorem 4.5.2], but they do not prove (4.2), and we wish to have insight into the operation of the machine in order to prove Theorem 4.3.

Proof of Lemma 4.4. The machine T_ν is essentially a decoder of an algorithmic coding scheme with respect to ν. It uses the natural correspondence between $\mathbb{B}^\infty$ and $[0, 1]$,

[1] A similar bound that can be proved the same way is $\sqrt{L_{n\mu}^{\Lambda_{S_{Kt}}}} - \sqrt{L_{n\mu}^{\Lambda_\mu}} \leq \sqrt{2D_n}$ for the same D_n [Hutter 2007, Equations 8, 5].

[2] Note that this lemma would be false if we were to let ν be an arbitrary lower-semicomputable semimeasure, since if $\nu = M$, this would imply that $2^{-Km(x)} \stackrel{\times}{=} M(x)$, which was disproved in Gács [1983].

associating a binary string $x_1x_2x_3\cdots$ with the real number $0.x_1x_2x_3\cdots$. It determines the location of the input sequence on this line, and then assigns certain intervals for output strings, such that the width of the intervals for output string x is equal to $\nu(x)$. Then, if input string p lies inside the interval for the output string x, it outputs x.

T_ν first calculates $\nu(0)$ and $\nu(1)$, and sets $[0, \nu(0))$ as the output interval for 0 and $[\nu(0), \nu(0)+\nu(1))$ as the output interval for 1. It then reads the input, bit by bit. After reading input $p_{1:n}$, it constructs the input interval $[0.p_1p_2\cdots p_n, 0.p_1p_2\cdots p_n111111\cdots)$, which represents the interval that $0.p_1p_2\cdots p_np_{n+1}\cdots$ could lie in. It then checks if this input interval is contained in one of the output intervals. If it is, then it prints output appropriate for the interval, and if not, then it reads one more bit and repeats the process.

Suppose the first output bit is a 1. Then, T_ν calculates $\nu(10)$ and $\nu(11)$, and forms the new output intervals: $[\nu(0), \nu(0)+\nu(10))$ for outputting 0, and $[\nu(0)+\nu(10), \nu(0)+\nu(10)+\nu(11))$ for outputting 1. It then reads more input bits until the input interval lies within one of these new output intervals, and then outputs the appropriate bit. The computation proceeds in this fashion.

Equation (4.1) is satisfied, because $\sum_{p \xrightarrow{T_\nu} x} 2^{-|p|}$ is just the total length of all possible input intervals that fit inside the output interval for x, which by construction is $\nu(x)$.

To show that (4.2) is satisfied, note that $2^{-Km_{T_\nu}(x)}$ is the length of the largest input interval for x. Now, input intervals are binary intervals (that is, their start points and end points have a finite binary expansion), and for every interval I, there is some binary interval contained in I with length $\geq 1/4$ that of I. Therefore, the output interval for x contains some input interval with length at least $1/4$ that of the length of the output interval. Since the length of the output interval for x is just $\nu(x)$, we can conclude that $2^{-Km_{T_\nu}(x)} \geq \nu(x)/4$. □

Proof of Theorem 4.3. Using Lemma 4.4, we show a bound on S_{Kt} that bounds its KL divergence with μ. We then apply Hutter's unit loss bound [Hutter 2005, Theorem 3.48] (originally shown for the Solomonoff prior, but valid for any prior) to show the desired result.

First, we reason about the running time of the shortest program that prints x on T_ν. Since we would only calculate $\nu(y0)$ and $\nu(y1)$ for $y \sqsubseteq x$, this amounts to $2|x|$ calculations. Each calculation need only take polynomial time in the length of its argument, because T_ν could just simulate the machine that takes input x and returns the numerator and denominator of x, prefix-free coded, and it only takes polynomial time to undo this prefix-free coding. Therefore, the calculations take at most $2|x|f(|x|) =: g(|x|)$, where f is a polynomial. We also, however, need to read all the bits of the input, construct the input intervals, and compare them to the output intervals. This takes time linear in the number of bits read, and for the shortest program that prints x, this number of bits is (by definition) $Km_{T_\nu}(x)$. Since $2^{-Km_{T_\nu(x)}} \stackrel{\times}{=} \nu(x)$, $Km_{T_\nu}(x) \leq -\log\nu(x) + O(1)$, and since $\nu(x) \stackrel{\times}{=} \mu(x)$, $-\log\nu(x) \leq -\log\mu(x) + O(1)$. Therefore, the total time taken is bounded above by $g(|x|) - O(1)\log\mu(x)$, where we absorb the additive constants into $g(|x|)$.

This out of the way, we can calculate

$$\begin{aligned} S_{Kt}(x) &= \sum_{p \xrightarrow{U} x} \frac{2^{-|p|}}{t(U,p,x)} \\ &= \sum_{i \in \mathbb{N}} 2^{-|\langle i \rangle|} \sum_{q \xrightarrow{T_i} x} \frac{2^{-|q|}}{t(T_i,q,x)^{O(1)}} \\ &\overset{\times}{\geq} \sum_{p \xrightarrow{T_\nu} x} \frac{2^{-|p|}}{t(T_\nu,p,x)^{O(1)}} \\ &\geq \frac{2^{-Km_{T_\nu}(x)}}{(g(|x|) - O(1)\log\mu(x))^{O(1)}} \\ &\overset{\times}{\geq} \frac{\mu(x)}{(g(|x|) - O(1)\log\mu(x))^{O(1)}} \end{aligned} \tag{4.3}$$

Now, Hutter's unit loss bound tells us that

$$L_{n\mu}^{\Lambda_{S_{Kt}}} - L_{n\mu}^{\Lambda_\mu} \leq 2D_n(\mu||S_{Kt}) + 2\sqrt{L_{n\mu}^{\Lambda_\mu} D_n(\mu||S_{Kt})} \tag{4.4}$$

where $D_n(\mu||S_{Kt}) := \mathbb{E}_\mu\left[\ln \mu(x_{1:n})/S_{Kt}(x_{1:n})\right]$ is the relative entropy. We can calculate $D_n(\mu||S_{Kt})$ using equation (4.3):

$$\begin{aligned} D_n(\mu||S_{Kt}) &= \mathbb{E}_\mu\left[\ln \frac{\mu(x_{1:n})}{S_{Kt}(x_{1:n})}\right] \\ &\overset{\times}{\leq} \mathbb{E}_\mu\left[\ln\left((g(n) - O(1)\log\mu(x_{1:n}))^{O(1)}\right)\right] \\ &\overset{\times}{\leq} \mathbb{E}_\mu\left[\ln(g(n) - O(1)\log\mu(x_{1:n}))\right] \\ &\leq \ln \mathbb{E}_\mu\left[g(n) - O(1)\log\mu(x_{1:n})\right] \\ &= \ln\left(g(n) + O(1)H_\mu(x_{1:n})\right) \end{aligned} \tag{4.5}$$

where $H_\mu(x_{1:n})$ denotes the binary entropy of the random variable $x_{1:n}$ with respect to μ

$$\begin{aligned} &\leq \ln\left(g(n) + O(n)\right) \\ &\overset{\times}{=} \log n \end{aligned} \tag{4.6}$$

where (4.5) comes from Jensen's inequality. Equations (4.4) and (4.6) together prove the theorem. □

We therefore have a loss bound on the S_{Kt}-based sequence predictor in environments that are estimable in polynomial time by a semimeasure. Furthermore:

Corollary 4.5.

$$L_{n\mu}^{\Lambda_{S_{Kt}}} \leq 2D_n(\mu||S_{Kt}) \overset{\times}{=} \log n$$

for deterministic measures[3] μ computable in polynomial time, if correct predictions incur no loss.

Compare to M-based prediction, which only incurs constant loss in this situation.

We should note that this method fails to prove similar bounds for S_{Fast}, since we instead get

$$\begin{aligned} S_{\text{Fast}}(x) &\stackrel{\times}{=} \sum_{p \xrightarrow{U} x} \frac{2^{-2|p|}}{t(U,p,x)} \\ &\stackrel{\times}{\geq} \frac{\mu(x)^2}{(g(|x|) - O(1)\log \mu(x))^{O(1)}} \end{aligned} \tag{4.7}$$

which gives us

$$\begin{aligned} D_n(\mu||S_{\text{Fast}}) &= \mathbb{E}_\mu \left[\ln \frac{\mu(x_{1:n})}{S_{\text{Fast}}(x_{1:n})} \right] \\ &\leq O(\log n) + H_\mu(x_{1:n}) \end{aligned}$$

Since $H_\mu(x_{1:n})$ can grow linearly in n (for example, take μ to be $\lambda(x) = 2^{-|x|}$, the uniform measure), this can only prove a trivial linear loss bound.

One important application of Theorem 4.3 is to the 0-1 loss function. Then, it states that a predictor that outputs the most likely successor bit according to S_{Kt} only makes logarithmically many errors in a deterministic environment computable in polynomial time. In other words, S_{Kt} quickly learns the sequence it is predicting, making very few errors.

4.2 Arbitrary measures

In this section, we generalise the prediction result above to measures of arbitrary time complexity.

Definition 4.6. Let $f, g : \mathbb{B}^* \to \mathbb{R}$. The function g is *estimable in $h(n)$ time by f* if f is finitely computable in at most $h(n)$ time and $f(x) \stackrel{\times}{=} g(x)$.

Theorem 4.7. If μ is a measure that is estimable in $f(n)$ time by some semimeasure ν, and x is a sequence sampled from μ, then the expected loss incurred by the $\Lambda_{S_{Kt}}$ predictor is bounded by

$$L_{n\mu}^{\Lambda_{S_{Kt}}} - L_{n\mu}^{\Lambda_\mu} \leq 2D_n + 2\sqrt{L_{n\mu}^{\Lambda_\mu} D_n}$$

where $D_n \stackrel{\times}{\leq} \log n + \log f(n)$.

Proof. Reviewing the operation of the machine T_ν as defined in Lemma 4.4, we see that in order to output x, we must calculate the value of ν a total of $2|x|$ times, and

[3] That is, measures that give probability 1 to prefixes of one particular infinite sequence.

that each time we calculate we need time $|x|^{O(1)} f(|x|)$, not just to calculate, but also to prepare the output intervals for comparison. We still must read input of length $-\log \mu(x) + O(1)$. Therefore,

$$S_{Kt}(x) \stackrel{\times}{\geq} \frac{\mu(x)}{(|x|^{O(1)} f(|x|) - O(1) \log \mu(x))^{O(1)}}$$

so

$$\begin{aligned} D_n(\mu||S_{Kt}) &\stackrel{\times}{\leq} \ln \mathbb{E}_\mu \left[n^{O(1)} f(n) - O(1) \log \mu(x_{1:n}) \right] \\ &= \ln \left(n^{O(1)} f(n) + O(1) H_\mu(x_{1:n}) \right) \\ &\leq \ln \left(n^{O(1)} f(n) + O(n) \right) \\ &\stackrel{\times}{\leq} \log f(n) + \log n \end{aligned}$$

We then apply Hutter's unit loss bound. □

Note that this bound is trivial if $f(n)$ is exponential, but if $f(n)$ is $2^{o(n)}$, we get a bound that is $o(n)$ (assuming Λ_μ makes $o(n)$ expected errors).

Chapter 5

Time Complexity

We have proved that S_{Fast} is computable in Section 2.6. However, no bounds are given for the running time of the proposed algorithm. Given that the major advantage of S_{Fast}-based prediction over M-based prediction is its computability, it is of interest to determine the time required to compute S_{Fast}, and whether such a computation is feasible or not. The same questions apply to S_{Kt}, to a greater extent because we have not yet shown that S_{Kt} is computable.

In this chapter, we show that an arbitrarily good approximation to $S_{\text{Fast}}(x)$ is computable in time exponential in $|x|$, and an arbitrarily good approximation to $S_{Kt}(x)$ is computable in time doubly-exponential in $|x|$. We do this by explicitly constructing algorithms that perform PHASEs of FAST until enough contributions to S_{Fast} or S_{Kt} are found to constitute a sufficient proportion of the total.

We also show that no such approximation of S_{Kt} can be computed in polynomial time. We do this by contradiction: showing that if it were possible to do so, we would be able to construct an 'adversarial' sequence that was computable in polynomial time, yet could not be predicted by our approximation of S_{Kt}, a contradiction.

5.1 Upper bounds

5.1.1 S_{Fast}

Theorem 5.1. For any $\varepsilon > 0$, there exists an approximation $S^{\varepsilon}_{\text{Fast}}$ of S_{Fast} such that $|S^{\varepsilon}_{\text{Fast}}/S_{\text{Fast}} - 1| \leq \varepsilon$ and $S^{\varepsilon}_{\text{Fast}}(x)$ is computable in time exponential in $|x|$.

Proof. First, we note that in PHASE i of FAST, we try out $2^1 + \cdots + 2^i = 2^{i+1}$ program prefixes p, and each prefix p gets $2^{i-|p|}$ steps. Therefore, the total number of steps in PHASE i is $2^1 \times 2^{i-1} + 2^2 \times 2^{i-2} + \cdots + 2^i \times 2^{i-i} = i2^i$, and the total number of steps in the first k PHASEs is

$$\#\text{ steps} = \sum_{i=1}^{k} i2^i = 2^{k+1}(k-1) + 2 \tag{5.1}$$

Now, suppose we want to compute a sufficient approximation $S^{\varepsilon}_{\text{Fast}}(x)$. If we compute k phases of FAST and then add up all the contributions to $S_{\text{Fast}}(x)$ found in

those phases, the remaining contributions must add up to $\leq \sum_{i=k+1}^{\infty} 2^{-i} = 2^{-k}$. In order for the contributions we have added up to contribute $\geq 1 - \varepsilon$ of the total, it suffices to use k such that

$$k = \left\lfloor \log \left(\frac{1}{\varepsilon S_{\text{Fast}}(x)} \right) + 1 \right\rfloor \tag{5.2}$$

Now, we know that since the uniform measure $\lambda(x) = 2^{-|x|}$ is computable in polynomial time, we can substitute λ into equation (4.7) to obtain

$$S_{\text{Fast}}(x) \overset{\times}{\geq} \frac{2^{-2|x|}}{(|x|^{O(1)} + \log(2^{|x|}))^{O(1)}} = \frac{1}{|x|^{O(1)} 2^{2|x|}} \tag{5.3}$$

Substituting equation (5.3) into equation (5.2), we get

$$k \leq \log \left(\frac{O(2^{2|x|}|x|^{O(1)})}{\varepsilon} \right) + 1 = -\log \varepsilon + 2|x| + O(\log |x|) \tag{5.4}$$

So, substituting equation (5.4) into equation (5.1),

$$\begin{aligned} \#\text{ steps} &\leq 2^{-\log \varepsilon + 2|x| + O(\log |x|) + 1}(-\log \varepsilon + 2|x| + O(\log |x|) - 1) + 2 \\ &= \frac{1}{\varepsilon} 2^{2|x|} |x|^{O(1)} (-\log \varepsilon + 2|x| + O(\log |x|)) + O(1) \\ &\leq 2^{O(|x|)} \end{aligned}$$

Therefore, $S^{\varepsilon}_{\text{Fast}}$ is computable in exponential time. □

5.1.2 S_{Kt}

Theorem 5.2. For any $\varepsilon > 0$, there exists an approximation S^{ε}_{Kt} of S_{Kt} such that $|S^{\varepsilon}_{Kt}/S_{Kt} - 1| \leq \varepsilon$ and $S^{\varepsilon}_{Kt}(x)$ is computable in time doubly-exponential in $|x|$.

Proof. We again use the general strategy of computing k PHASEs of FAST, and adding up all the contributions to $S_{Kt}(x)$ we find. Once we have done this, the other contributions come from computations with Kt-cost $> k$. Therefore, the programs making these contributions either have a program of length $> k$, or take time $> 2^k$ (or both).

First, we bound the contribution $C_{t>2^k}$ to $S_{Kt}(x)$ by computations of time $> 2^k$, writing $p \underset{>i}{\longrightarrow} x$ to mean p computes x in time $> i$:

$$C_{t>2^k} \leq \sum_{p \underset{>2^k}{\longrightarrow} x} \frac{2^{-|p|}}{t(p,x)} < \frac{1}{2^k} \sum_{p \to x} 2^{-|p|} \leq \frac{1}{2^k}$$

Next, we bound the contribution by computations with programs of length $|p| > k$. We note that since we are dealing with the monotone UTM, the worst case is that all programs have length $k+1$, and the time taken is only $k+1$ (since, by the definition

of monotone machines, we need at least enough time to read the input). Then, the contribution from these programs is $2^{k+1} \times (1/(k+1)) \times 2^{-k-1} = 1/(k+1)$, meaning that the total remaining contribution after k PHASEs is no more than $2^{-k} + 1/(k+1) \leq 2/(k+1)$.

So, in order for our contributions to add up to $\geq 1 - \varepsilon$ of the total, it suffices to use k such that

$$k = \left\lfloor \frac{2}{\varepsilon S_{Kt}(x)} \right\rfloor \tag{5.5}$$

Now, since λ is estimable in polynomial time, we substitute it into equation (4.3) to obtain

$$S_{Kt}(x) \stackrel{\times}{\geq} \frac{1}{|x|^{O(1)} 2^{|x|}} \tag{5.6}$$

Substituting equation (5.6) into equation (5.5), we get

$$k \leq \frac{O(|x|^{O(1)} 2^{|x|})}{\varepsilon} \tag{5.7}$$

So, substituting equation (5.7) into equation (5.1), we finally obtain

$$\#\text{ steps} \leq 2^{O(|x|^{O(1)} 2^{|x|})/\varepsilon} \left(\frac{O(|x|^{O(1)} 2^{|x|})}{\varepsilon} \right) + 2 \leq 2^{2^{O(|x|)}}$$

Therefore, S^{ε}_{Kt} is computable in doubly-exponential time.

□

5.2 S_{Kt} is not efficiently computable

Theorem 5.3. For no $\varepsilon > 0$ does there exist an approximation S^{ε}_{Kt} of S_{Kt} such that $|S^{\varepsilon}_{Kt}/S_{Kt} - 1| \leq \varepsilon$ and $S^{\varepsilon}_{Kt}(x)$ is computable in time polynomial in $|x|$.

The proof of this theorem relies on the following lemma:

Lemma 5.4. If S^{ε}_{Kt} is an approximation of S_{Kt} as given in Theorem 5.3, then the bound of Theorem 4.3 applies to S^{ε}_{Kt}. That is,

$$L_{n\mu}^{\Lambda_{S^{\varepsilon}_{Kt}}} - L_{n\mu}^{\Lambda_{\mu}} \leq 2D_n + 2\sqrt{L_{n\mu}^{\Lambda_{\mu}} D_n}$$

where $D_n = O(\log n)$.

Proof of Lemma 5.4. From the definition of S^{ε}_{Kt}, we have that $S^{\varepsilon}_{Kt} \geq (1-\varepsilon) S_{Kt}$. Then,

$$D_n(\mu || S^{\varepsilon}_{Kt}) := \mathbb{E}_{\mu} \left[\ln \frac{\mu(x_{1:n})}{S^{\varepsilon}_{Kt}(x_{1:n})} \right] \leq \mathbb{E}_{\mu} \left[\ln \frac{\mu(x_{1:n})}{S_{Kt}(x_{1:n})} \right] - \ln(1-\varepsilon) \stackrel{\times}{=} \log n$$

for μ estimable in polynomial time by a semimeasure, where we use equation (4.6) for the final 'equality'. Therefore, the bound of Theorem 4.3 applies. □

Proof of Theorem 5.3. Suppose by way of contradiction that S^{ε}_{Kt} were computable in polynomial time. Then, the sequence $x_{1:\infty}$ would also be computable in polynomial time, where

$$x_n = \begin{cases} 1 & \text{if } S^{\varepsilon}_{Kt}(0|x_{1:n-1}) \geq S^{\varepsilon}_{Kt}(1|x_{1:n-1}) \\ 0 & \text{if } S^{\varepsilon}_{Kt}(0|x_{1:n-1}) < S^{\varepsilon}_{Kt}(1|x_{1:n-1}) \end{cases}$$

$x_{1:\infty}$ is therefore an adversarial sequence against S^{ε}_{Kt}: it predicts whichever symbol S^{ε}_{Kt} thinks less likely, and breaks ties with 1.

Now, consider an S^{ε}_{Kt}-based predictor $\Lambda_{S^{\varepsilon}_{Kt}}$ that minimises 0-1 loss—that is, one that predicts the more likely continuation according to S^{ε}_{Kt}. Further, suppose this predictor breaks ties with 0. Since the loss bound of Theorem 4.3 applies independently of tie-breaking method, Lemma 5.4 tells us that $\Lambda_{S^{\varepsilon}_{Kt}}$ must make only logarithmically many errors when predicting $x_{1:\infty}$. However, by design, $\Lambda_{S^{\varepsilon}_{Kt}}$ errs every time when predicting $x_{1:\infty}$. This is a contradiction, showing that S^{ε}_{Kt} cannot be computable in polynomial time. □

Note that the same proof shows that S^{ε}_{Kt} cannot be computable in time $2^{o(x)}$, using the result on loss bounds for general measures proved in Section 4.2. Unfortunately, we cannot use this proof technique to show lower bounds on the computation time of $S^{\varepsilon}_{\text{Fast}}$, since we have no non-trivial predictive results for S_{Fast}. However, it does show that if S_{Fast} is computable in polynomial time, then it cannot predict all sequences computable in polynomial time.

5.3 Predicting $f(n)$-computable sequences

In this section, we bound the time taken to compute S_{Kt} and S_{Fast} along a sequence computable in $f(n)$ time: that is, if the prefixes $x_{1:n}$ of $x_{1:\infty} \in \mathbb{B}^{\infty}$ are computable in time $f(n)$ for all n, we bound the time of computing $S(x_{1:n}0)$ and $S(x_{1:n}1)$ for $S \in \{S^{\varepsilon}_{Kt}, S^{\varepsilon}_{\text{Fast}}\}$.

Theorem 5.5. If $x_{1:\infty}$ is computable in time $f(n)$, then $S^{\varepsilon}_{\text{Fast}}(x_{1:n}0)$ and $S^{\varepsilon}_{\text{Fast}}(x_{1:n}1)$ are computable in time $O(n^4 f(n))$, where $S^{\varepsilon}_{\text{Fast}}$ is the approximation of S_{Fast} of Theorem 5.1.

Proof. Suppose some program p^x prints $x_{1:\infty}$ in time $f(n)$. Then,

$$S_{\text{Fast}}(x_{1:n}) \geq \frac{2^{-2|p^x|}}{f(n)}$$

Substituting this into equation (5.2), we learn that we can compute $S^{\varepsilon}_{\text{Fast}}(x_{1:n})$, by computing FAST for k PHASES where

$$k \leq \left\lfloor \log(2^{2|p^x|} f(n)/\varepsilon) \right\rfloor$$

Substituting this into equation (5.1) gives

$$\begin{aligned}\#\text{ steps} &\leq 2^{\log(2^{2|p^x|}f(n)/\varepsilon)}(\log(2^{2|p^x|}f(n)/\varepsilon)-1)+2\\ &= O(f(n))\end{aligned}$$

Therefore, we only require $O(f(n))$ many steps of the FAST algorithm to compute $S^{\varepsilon}_{\text{Fast}}(x_{1:n})$. To prove that it only takes $O(n^4 f(n))$ steps to compute $S^{\varepsilon}_{\text{Fast}}(x_{1:n}b)$ for any $b \in \mathbb{B}$ requires some more careful analysis.

Let $\langle n\rangle$ be a prefix-free coding of the natural numbers in $2\log n$ bits. Then, if $b \in \mathbb{B}$, then there is some program prefix p^b such that $p^b\langle n\rangle q$ runs program q until it prints n symbols on the output tape, after which it stops running q, prints b, and then halts. In addition to running q (possibly slowed down by a constant factor), it must run some sort of timer to count down to n. This involves reading and writing the integers 1 to n, which takes $O(n\log n)$ time. Therefore, $p^b\langle n\rangle p^x$ prints $x_{1:n}b$ in time $O(f(n))+O(n\log n)$, so

$$\begin{aligned}S_{\text{Fast}}(x_{1:n}b) &\geq \frac{2^{-2|p^b\langle n\rangle p^x|}}{O(f(n))+O(n\log n)}\\ &= \frac{1}{O(f(n))+O(n\log n)}\frac{1}{n^4 2^{2|p^b|+2|p^x|}}\\ &= \frac{1}{O(n^4 f(n))}\end{aligned}$$

since, because $f(n)\geq n$, $n^4 f(n)\geq n\log n$. Using equations (5.2) and (5.1) therefore gives that we only need $O(n^4 f(n))$ timesteps to compute $S^{\varepsilon}_{\text{Fast}}(x_{1:n}b)$. □

Theorem 5.6. If $x_{1:\infty}$ is computable in time $f(n)$, then $S^{\varepsilon}_{Kt}(x_{1:n}0)$ and $S^{\varepsilon}_{Kt}(x_{1:n}1)$ are computable in time $2^{O(n^2 f(n))}$, where S^{ε}_{Kt} is the approximation of S_{Kt} of Theorem 5.2.

Proof. The proof is almost identical to the proof of Theorem 5.5: supposing that p^x prints $x_{1:n}$ in time $f(n)$, we have

$$S_{Kt}(x_{1:n}) \geq \frac{2^{-|p^x|}}{f(n)}$$

The difference is that we substitute this into equation (5.5), getting

$$k \leq \left\lfloor 2^{|p^x|+1} f(n)/\varepsilon \right\rfloor$$

and substitution into equation (5.1) now gives

$$\begin{aligned}\#\text{ steps} &\leq 2^{2^{|p^x|+1}f(n)/\varepsilon}\left(2^{|p^x|+1}f(n)/\varepsilon - 1\right)+2\\ &= 2^{O(f(n))}\end{aligned}$$

The other difference is that because we only penalise the length of the program, instead

of twice the length of the program,

$$\begin{aligned} S_{Kt}(x_{1:n}b) &\geq \frac{2^{-|p^b\langle n\rangle p^x|}}{O(f(n)) + O(n\log n)} \\ &= \frac{1}{O(f(n)) + O(n\log n)} \frac{1}{n^2 2^{|p^b|+2|p^x|}} \\ &= \frac{1}{O(n^2 f(n))} \end{aligned}$$

Therefore, we can compute $S^{\varepsilon}_{Kt}(x_{1:n}0)$ and $S^{\varepsilon}_{Kt}(x_{1:n}1)$ in time $2^{O(n^2 f(n))}$. □

Chapter 6

Reinforcement Learning

We can adapt S_{Kt} (the only speed prior that we were able to prove good predictive results for) into a chronological conditional semimeasure for use in reinforcement learning, writing

$$S(e_{1:t}||a_{1:\infty}) := \sum_{p:\,(p,a_{1:t})\to e_{1:t}} \frac{2^{-|p|}}{t((p,a_{1:t}),e_{1:t})}$$

where $t((p,a_{1:t}),e_{1:t})$ denotes the number of timesteps it takes for the mixed-input UTM U to print $e_{1:t}$ given input $(p,a_{1:t})$.

However, we focus on a generalisation of the speed environment. We note that all the predictive results of S_{Kt} came from the relation

$$S_{Kt}(x) \geq \frac{w_\mu}{g_\mu(|x|)}\mu(x) \tag{6.1}$$

where μ was an environment of interest, and g_μ was a monotonically increasing, superlinear, and subexponential function[1]. Also, measures like this are known to converge to the true distribution if $g_\mu(|x|) \overset{\times}{\leq} \exp(\sqrt{n}/\log n)$ [Ryabko and Hutter 2007], in the sense that the conditional probabilities converge to the true conditional probabilities 'on average'. We therefore consider the speed mixture over a class $\mathcal{M}$ of chronological conditional measures:

Definition 6.1 (Speed mixture)**.**

$$S_{\mathcal{M}}(e_{1:t}||a_{1:\infty}) := \sum_{\nu\in\mathcal{M}} \frac{w_\nu}{g_\nu(t)}\nu(e_{1:t}||a_{1:\infty})$$

where $\sum_{\nu\in\mathcal{M}} w_\nu \leq 1$.

[1]Note that this is not strictly true: we instead had that $S_{Kt}(x) \geq w_\mu\mu(x)/(g_\mu(|x|) - O(1)\log\mu(x))^{O(1)}$ as shown in (4.3). However, the $-O(1)\log\mu(x)$ term was less than $O(1)|x|$ in expectation, so (6.1) would have sufficed to show these results. Additionally, if μ is estimable in polynomial time by ν, then the denominator of the fraction ν must be able to be printed in polynomial time. Since the number of the digits of the denominator is bounded by $-2\log\nu(x)$, which must be bounded by a polynomial in $|x|$, and since $\log\mu(x) \leq \log\nu(x) + O(1)$, $-O(1)\log\mu(x)$ can be deterministically bounded by a polynomial in $|x|$, showing that (4.3) can be rewritten as (6.1).

We also consider the speed mixture agent $\pi^{S_\mathcal{M}}$ that maximises expected reward sum, where the expectation is with respect to $S_\mathcal{M}$.

6.1 Self-optimisingness

We might wish that the generalisation of AIXI, our agent $\pi^{S_\mathcal{M}}$ is self-optimising. In this section, we show that under certain conditions, the policy $\pi^{S_\mathcal{M}}$ (which we call the speed mixture agent) is self-optimising in every environment in $\mathcal{M}$.

6.1.1 Finite lifetime case

We consider the sequence of policies $\pi_m^{S_\mathcal{M}}$ for $m = 1, 2, 3, \ldots$ that maximise $V_{1m}^{\pi S_\mathcal{M}}$ respectively. Our main result is that this sequence is self-optimising in $\mathcal{M}$ if there is any sequence of policies that is exponentially quickly self-optimising in $\mathcal{M}$, and if $\mathcal{M}$ is sufficiently nice (in a sense that will be explained in the theorem). Recall that as in Definition 2.17, we will be considering the agent's behaviour as $m \to \infty$. In other words, we will not be looking at the behaviour of a single agent as time goes to infinity, but rather at the behaviour of a family of agents as their lifetime goes to infinity. Before proving this, we first state and prove some useful lemmas. These lemmas and their proofs are actually identical to those in [Hutter 2005, Chapter 5.4] with the replacement $w_\nu \mapsto w_\nu / g_\nu(m)$, but we give their proofs here for clarity and completeness.

Our first lemma shows that the value obtained by policy π in the speed mixture environment is a linear combination of the value it obtains in all environments in $\mathcal{M}$.

Lemma 6.2 (Linearity of value (finite lifetime))**.**

$$V_{1m}^{\pi S_\mathcal{M}} = \sum_{\nu \in \mathcal{M}} \frac{w_\nu}{g_\nu(m)} V_{1m}^{\pi \nu}$$

Proof.

$$\begin{aligned}
V_{1m}^{\pi S_\mathcal{M}} &= \sum_{e_{1:m}} (r_1 + \cdots + r_m) S_\mathcal{M}(e_{1:m} || a_{1:m}) \\
&= \sum_{e_{1:m}} (r_1 + \cdots + r_m) \sum_{\nu \in \mathcal{M}} \frac{w_\nu}{g_\nu(m)} \nu(e_{1:m} || a_{1:m}) \\
&= \sum_{\nu \in \mathcal{M}} \frac{w_\nu}{g_\nu(m)} \sum_{e_{1:m}} (r_1 + \cdots + r_m) \nu(e_{1:m} || a_{1:m}) \\
&= \sum_{\nu \in \mathcal{M}} \frac{w_\nu}{g_\nu(m)} V_{1m}^{\pi \nu}
\end{aligned}$$

where the action sequence a is generated by the policy π. □

Next, we show a technical lemma that lets us bound the difference in value of the speed mixture agent and the optimal agent in some environment in terms of the difference of values of some other agent and the optimal agent in each environment of $\mathcal{M}$.

Lemma 6.3 (Value difference relation (finite lifetime))**.** For any policy π, let

$$V_{1m}^{\pi_m^\nu \nu} - V_{1m}^{\pi\nu} =: \Delta_\nu(\pi)$$

Then,

$$V_{1m}^{\pi_m^\nu \nu} - V_{1m}^{\pi_m^{S_\mathcal{M}} \nu} \leq \frac{g_\nu(m)}{w_\nu} \sum_{\rho \in \mathcal{M}} \frac{w_\rho}{g_\rho(m)} \Delta_\rho(\pi)$$

Proof.

$$\begin{aligned}
\frac{w_\nu}{g_\nu(m)} \left(V_{1m}^{\pi_m^\nu \nu} - V_{1m}^{\pi_m^{S_\mathcal{M}} \nu} \right) &\leq \sum_{\rho \in \mathcal{M}} \frac{w_\rho}{g_\rho(m)} \left(V_{1m}^{\pi^\rho \rho} - V_{1m}^{\pi_m^{S_\mathcal{M}} \rho} \right) \\
&= \left(\sum_{\rho \in \mathcal{M}} \frac{w_\rho}{g_\rho(m)} V_{1m}^{\pi_m^\rho \rho} \right) - \left(\sum_{\rho \in \mathcal{M}} \frac{w_\rho}{g_\rho(m)} V_{1m}^{\pi_m^{S_\mathcal{M}} \rho} \right) \\
&= \left(\sum_{\rho \in \mathcal{M}} \frac{w_\rho}{g_\rho(m)} V_{1m}^{\pi_m^\rho \rho} \right) - V_{1m}^{\pi_m^{S_\mathcal{M}} S_\mathcal{M}} \qquad (6.2) \\
&\leq \left(\sum_{\rho \in \mathcal{M}} \frac{w_\rho}{g_\rho(m)} V_{1m}^{\pi_m^\rho \rho} \right) - V_{1m}^{\pi S_\mathcal{M}} \qquad (6.3) \\
&= \sum_{\rho \in \mathcal{M}} \frac{w_\rho}{g_\rho(m)} \left(V_{1m}^{\pi_m^\rho \rho} - V_{1m}^{\pi\rho} \right) \qquad (6.4) \\
&= \sum_{\rho \in \mathcal{M}} \frac{w_\rho}{g_\rho(m)} \Delta_\rho(\pi)
\end{aligned}$$

where (6.2) and (6.4) use Lemma 6.2, and (6.3) uses the optimality of $\pi_m^{S_\mathcal{M}}$ in $S_\mathcal{M}$. □

Theorem 6.4 (Self-optimising theorem for finite lifetime)**.** Let π_m be a sequence of policies, and define

$$\frac{1}{m} \left(V_{1m}^{\pi_m^\nu \nu} - V_{1m}^{\pi_m \nu} \right) =: \delta_\nu(m)$$

Suppose that for every $\mu, \nu \in \mathcal{M}$, $g_\mu(m)\delta_\nu(m)$ is bounded by some constant that depends on μ but not on ν or m, and that $g_\mu(m)\delta_\nu(m) \to 0$ (which implies that for all $\nu \in \mathcal{M}$, $\delta_\nu(m) \to 0$). Then, in every environment $\mu \in \mathcal{M}$,

$$\lim_{m \to \infty} \frac{1}{m} \left(V_{1m}^{\pi_m^\mu \mu} - V_{1m}^{\pi_m^{S_\mathcal{M}} \mu} \right) = 0$$

Note that the condition that $g_\mu(m)\delta_\nu(m)$ is bounded by a constant independent of ν and m is rather restrictive in infinite environment classes. If the functions g_μ are polynomial, it requires that the functions δ_ν are all exponentially (or superpolynomially) decaying with bounded multiplicative constants. This condition would be violated if, for example, (indexing our environments with natural numbers) $\delta_{\nu_n}(m) = n \exp(-m)$ or $\delta_{\nu_n}(m) = \exp(-m) + n \exp(-2m)$.

Proof. Using Lemma 6.3, where $\Delta_\nu(\pi_m) = m\delta_\nu(m)$, we have

$$\frac{1}{m}\left(V_{1m}^{\pi_m^\mu \mu} - V_{1m}^{\pi_m^{S_\mathcal{M}} \mu}\right) \leq \frac{g_\mu(m)}{w_\mu} \sum_{\nu \in \mathcal{M}} \frac{w_\nu}{g_\nu(m)} \delta_\nu(m)$$
$$\leq \frac{1}{w_\mu} \sum_{\nu \in \mathcal{M}} w_\nu g_\mu(m) \delta_\nu(m)$$

Now, $g_\mu(m)$ is subexponential, meaning that for all ν, $g_\mu(m)\delta_\nu(m) \to 0$ as $m \to \infty$. Since $g_\mu(m)\delta_\nu(m)$ is bounded by a constant independent of ν or m, we may use a result on the convergence of averages to show that $\sum_{\nu \in \mathcal{M}} w_\nu g_\mu(m)\delta_\nu(m) \to 0$ [Hutter 2005, Lemma 5.28 (ii)], since $\sum_{\nu \in \mathcal{M}} w_\nu \leq 1$. This proves the theorem. □

6.1.2 Infinite lifetime case

We next consider the undiscounted infinite lifetime case, where at timestep k agents only consider rewards received up to timestep m_k. Therefore, we define the policy π_k^ν as that which maximises $V_{km_k}^{\pi\nu}$, the ν-expected reward sum between timesteps k and m_k.

In the infinite lifetime case, the desirable property of being self-optimising is that

$$\frac{1}{m_k - k + 1} V_{km_k}^{\pi_k^{S_\mathcal{M}} \mu} \xrightarrow{k \to \infty} \frac{1}{m_k - k + 1} V_{km_k}^{\pi_k^\mu \mu}$$

Although this is similar to the definition for finite lifetimes, there are crucial differences: firstly, we are genuinely talking about the behaviour of a single agent as it goes through time, rather than that of a family of agents as one of their parameters increases, and secondly, we are not talking about the average value that the agent actually receives, but rather the average value that it plans to receive (despite the fact that the agent may very well change its plan later in order to receive large rewards at time $m_k + 1$, for example).

We will show that our agent is self-optimising in environment classes that admit self-optimising policies, but first, we show some preliminary lemmas of the same flavour as in Subsection 6.1.1.

Lemma 6.5 (Linearity of value (infinite lifetime))**.** For all policies π,

$$V_{km_k}^{\pi S_\mathcal{M}} = \sum_{\nu \in \mathcal{M}} \frac{w_\nu^k}{g_\nu(m_k)} V_{km_k}^{\pi\nu}$$

where

$$w_\nu^k = w_\nu \frac{\nu(e_{<k}||a_{<k})}{S_\mathcal{M}(e_{<k}||a_{<k})}$$

Proof.

$$V_{km_k}^{\pi S_\mathcal{M}} = \sum_{e_{k:m_k}} R_{k:m_k} S_\mathcal{M}(e_{k:m_k}|e_{<k}||a_{1:m_k})$$

$$
\begin{aligned}
&= \frac{1}{S_{\mathcal{M}}(e_{<k}||a_{<k})} \sum_{e_{k:m_k}} R_{k:m_k} S_{\mathcal{M}}(e_{1:m_k}||a_{1:m_k}) \\
&= \frac{1}{S_{\mathcal{M}}(e_{<k}||a_{<k})} \sum_{e_{k:m_k}} R_{k:m_k} \sum_{\nu\in\mathcal{M}} \frac{w_\nu}{g_\nu(m_k)} \nu(e_{1:m_k}||a_{1:m_k}) \\
&= \frac{1}{S_{\mathcal{M}}(e_{<k}||a_{<k})} \sum_{\nu\in\mathcal{M}} \frac{w_\nu}{g_\nu(m_k)} \sum_{e_{k:m_k}} R_{k:m_k} \nu(e_{k:m_k}|e_{<k}||a_{1:m_k}) \nu(e_{<k}||a_{<k}) \\
&= \sum_{\nu\in\mathcal{M}} \frac{w_\nu}{g_\nu(m_k)} \frac{\nu(e_{<k}||a_{<k})}{S_{\mathcal{M}}(e_{<k}||a_{<k})} \sum_{e_{k:m_k}} R_{k:m_k} \nu(e_{k:m_k}|e_{<k}||a_{1:m_k}) \\
&= \sum_{\nu\in\mathcal{M}} \frac{w_\nu^k}{g_\nu(m_k)} V_{km_k}^{\pi\nu}
\end{aligned}
$$

where all actions are generated by π, and $R_{k:m_k}$ is an abbreviation for $r_k + \cdots + r_{m_k}$. □

Lemma 6.6 (Value difference relation (infinite lifetime))**.** For any sequence of policies π_k, let

$$V_{km_k}^{\pi_k^\nu \nu} - V_{km_k}^{\pi_k \nu} =: \Delta_\nu(\pi_k)$$

Then,

$$V_{km_k}^{\pi_k^\nu \nu} - V_{km_k}^{\pi_k^{S_\mathcal{M}} \nu} \leq \frac{g_\nu(m_k)}{w_\nu^k} \sum_{\rho\in\mathcal{M}} \frac{w_\rho^k}{g_\rho(m_k)} \Delta_\rho(\pi_k)$$

Proof.

$$
\begin{aligned}
\frac{w_\nu^k}{g_\nu(m_k)} (V_{km_k}^{\pi_k^\nu \nu} - V_{km_k}^{\pi_k^{S_\mathcal{M}} \nu}) &\leq \sum_{\rho\in\mathcal{M}} \frac{w_\rho^k}{g_\rho(m_k)} \left(V_{km_k}^{\pi_k^\rho \rho} - V_{km_k}^{\pi_k^{S_\mathcal{M}} \rho} \right) \\
&= \left(\sum_{\rho\in\mathcal{M}} \frac{w_\rho^k}{g_\rho(m_k)} V_{km_k}^{\pi_k^\rho \rho} \right) - V_{km_k}^{\pi_k^{S_\mathcal{M}} S_\mathcal{M}} && (6.5) \\
&\leq \left(\sum_{\rho\in\mathcal{M}} \frac{w_\rho^k}{g_\rho(m_k)} V_{km_k}^{\pi_k^\rho \rho} \right) - V_{km_k}^{\pi_k S_\mathcal{M}} && (6.6) \\
&= \sum_{\rho\in\mathcal{M}} \frac{w_\rho^k}{g_\rho(m_k)} \left(V_{km_k}^{\pi_k^\rho \rho} - V_{km_k}^{\pi_k \rho} \right) && (6.7) \\
&= \sum_{\rho\in\mathcal{M}} \frac{w_\rho^k}{g_\rho(m_k)} \Delta_\rho(\pi_k)
\end{aligned}
$$

where (6.5) and (6.7) use Lemma 6.5, and (6.6) uses the optimality of $\pi_k^{S_\mathcal{M}}$ in $S_\mathcal{M}$. □

Theorem 6.7 (Self-optimising theorem for infinite lifetime)**.** Suppose that there exists a sequence of policies π_k such that for all $\nu \in \mathcal{M}$,

$$\frac{1}{m_k - k + 1} \left(V_{km_k}^{\pi_k^\nu \nu} - V_{km_k}^{\pi_k \nu} \right) = \delta_\nu(k)$$

where for all $\nu, \rho \in \mathcal{M}$, $\delta_\nu(k) \to 0$ with ν-probability 1, $g_\rho(k)\delta_\nu(k)^{1/2} \leq c_\nu$, and $g_\rho(m_k)\delta_\nu(k)^{1/4} \leq c'_\nu$, where c_ν and c'_ν are constants depending only on ν. Then for all $\mu \in \mathcal{M}$,

$$\lim_{k\to\infty} \frac{1}{m_k - k + 1}\left(V_{km_k}^{\pi_k^\mu \mu} - V_{km_k}^{\pi_k^{S_\mathcal{M}} \mu}\right) = 0 \text{ with } \mu\text{-probability 1}$$

The conditions of this theorem are similar to that of Theorem 6.4. One important difference is that there, we required that $g_\rho \delta_\nu$ was bounded by a constant only depending on ρ, while in this theorem, we replace δ_ν by $\delta_\nu^{1/2}$ and demand that the product be bounded by a constant only depending on ν. This rules out (for instance) δ_ν being exponentially decaying and the functions g_ρ being polynomials of unbounded degree, or even of bounded degree but unbounded constants. We also have a condition on m_k, which would be satisfied if (for instance) the functions δ_ν are exponentially decaying, the functions g_ν are polynomials, and m_k is polynomial in k. This is not such a severe restriction: $m_k - k + 1$ is analogous to the effective horizon in the context of discounting[2], and discounting schemes typically have effective horizons that only grow polynomially: in fact, the most commonly used discounting scheme, geometric discounting[3], has a constant effective horizon. Table 6.1 has a summary of discount schemes and their effective horizons.

Discount scheme	Discount factor	Effective horizon
Geometric	γ^k	$\lceil -\log 2/\log\gamma \rceil$
Power	$k^{-1-\beta}$, $\beta > 0$	$O(k)$
Near-harmonic	$k^{-1}(\ln k)^{-1-\beta}$, $\beta > 0$	$k^{2^{1/\beta}}$

Table 6.1: Discount schemes and their effective horizons

Proof. Similarly to our proof of Theorem 6.4, we use Lemma 6.6 where $\Delta_\nu(\pi_k) = \delta_\nu(k)/(m_k - k + 1)$ to write

$$\begin{aligned}\frac{1}{m_k - k + 1}\left(V_{km_k}^{\pi_k^\mu \mu} - V_{km_k}^{\pi_k^{S_\mathcal{M}} \mu}\right) &\leq \frac{g_\mu(m_k)}{w_\mu^k} \sum_{\nu\in\mathcal{M}} \frac{w_\nu^k}{g_\nu(m_k)}\delta_\nu(k) \\ &\leq \frac{g_\mu(m_k)}{w_\mu^k} \sum_{\nu\in\mathcal{M}} w_\nu^k \delta_\nu(k) \qquad (6.8)\end{aligned}$$

Next, we convert this to an equivalent expression for the Bayesian mixture over our class $\xi_\mathcal{M}(e_{<k}||a_{<k}) = \sum_{\nu\in\mathcal{M}} w_\nu \nu(e_{<k}||a_{<k})$, at which point we will be able to use a slightly different proof of self-optimisingness of these mixture agents to our advantage. Now,

[2]The effective horizon h_k is the first timestep such that $\Gamma_{k+h_k} \leq \Gamma_k/2$: that is, it measures the time after which half the possible discount-weighted reward available after time k is no longer available, and roughly tracks how farsighted the agent is.

[3]This is the only type mentioned in the classic reference on reinforcement learning, Sutton and Barto [1998], or the standard introduction to artificial intelligence, Russell and Norvig [2009].

$$\begin{aligned} w_\nu^k \delta_\nu(k) &= w_\nu \nu(e_{<k}||a_{<k}) \left(\sum_{\rho \in \mathcal{M}} \frac{w_\rho}{g_\rho(k-1)} \delta_\nu(k)^{-1} \rho(e_{<k}||a_{<k}) \right)^{-1} \\ &\leq w_\nu \nu(e_{<k}||a_{<k}) \left(\sum_{\rho \in \mathcal{M}} w_\rho \delta_\nu(k)^{-1/2} c_\nu^{-1} \rho(e_{<k}||a_{<k}) \right)^{-1} \\ &= c_\nu z_\nu^k \delta_\nu(k)^{1/2} \end{aligned}$$

where $z_\nu^k = w_\nu \nu(e_{<k}||a_{<k}) / \xi_{\mathcal{M}}(e_{<k}||a_{<k})$.

Next, we note that by assumption

$$g_\mu(m_k) \delta_\nu(k)^{1/2} \leq c'_\mu \delta_\nu(k)^{1/4}$$

and that

$$\begin{aligned} \frac{1}{w_\mu^k} &= \sum_{\nu \in \mathcal{M}} \frac{w_\nu}{g_\nu(k-1)} \frac{\nu(e_{<k}||a_{<k})}{\mu(e_{<k}||a_{<k})} \\ &< \sum_{\nu \in \mathcal{M}} w_\nu \frac{\nu(e_{<k}||a_{<k})}{\mu(e_{<k}||a_{<k})} \\ &= \frac{1}{z_\mu^k} \end{aligned}$$

Therefore,

$$\frac{g_\mu(m_k)}{w_\mu^k} \sum_{\nu \in \mathcal{M}} w_\nu^k \delta_\nu(k) < \frac{c'_\mu}{z_\mu^k} \sum_{\nu \in \mathcal{M}} c_\nu z_\nu^k \delta_\nu(k)^{1/4} \tag{6.9}$$

Using the self-optimising result for Bayesian mixtures shown in Hutter [2005, Theorem 5.34], we have that the right hand side of (6.9) goes to 0, since $\delta_\nu(k)$ is bounded above by 1, the maximum reward. Therefore, (6.8) and (6.9) together give us our desired result. □

Chapter 7

Conclusion

7.1 Summary of thesis

In this thesis, we have defined the speed prior S_{Kt}, and showed that it is able to predict efficiently computable sequences with few errors. Unfortunately, we were only able to bound the computation time by $2^{2^{O(n)}}$ (although this bound was reduced to $2^{n^{O(1)}}$ for sequences that are computable in polynomial time), and proved that S_{Kt} is not computable in sub-exponential time. Although we were not able to prove any prediction results for Schmidhuber's speed prior S_{Fast}, we showed time complexity bounds for it that were exponentially better than those for S_{Kt}. We also showed that in certain cases, a variant of S_{Kt} was self-optimising in the reinforcement learning setting.

7.2 Outlook

Unsurprisingly, there are many avenues of future research left to explore. First of all, although we were unable to show any positive prediction results about S_{Fast}, we were also unable to show that it fails at prediction where S_{Kt} succeeds. It would therefore be of interest to show any prediction results about S_{Fast}, positive or negative.

Secondly, there is an exponential gap between our time complexity lower bounds and upper bounds for S_{Kt}. It would therefore be of interest to know whether doubly-exponential time is required to compute S_{Kt} in the worst case, or whether it could be computed in (comparatively) merely exponential time. It would similarly be of interest to know whether S_{Fast} is computable in polynomial time (although an argument similar to the proof of Theorem 5.3 establishes that this cannot be true if S_{Fast} successfully predicts all polynomial-time computable sequences).

Another area of further research is in the RL domain. The self-optimising results we have shown here have had strong conditions on the environment class and the rate of self-optimisingness that we require for some other policy in the class, limiting their application. There is therefore room to weaken these conditions and see if the results still hold. It would also be worth knowing if there are in fact any environment classes that fit the conditions of Theorems 6.4 and 6.7, or whether these theorems are in fact vacuous.

It would also be desirable to prove results about the discounted infinite lifetime case,

since this is the case normally considered in reinforcement learning. Unfortunately, since the functions g_ν increase to infinity and are bounded below by 1,

$$\begin{aligned}
V_{k\gamma}^{\pi S_\mathcal{M}} &= \frac{1}{\Gamma_k} \lim_{m\to\infty} \sum_{e_{k:m}} (\gamma_k r_k + \cdots + \gamma_m r_m) S_\mathcal{M}(e_{k:m}|e_{<k}||a_{1:m}) \\
&= \frac{1}{\Gamma_k} \lim_{m\to\infty} \sum_{\nu\in\mathcal{M}} \frac{w_\nu}{g_\nu(m)} \sum_{e_{k:m}} (\gamma_k r_k + \cdots + \gamma_m r_m) \nu(e_{k:m}|e_{<k}||a_{1:m}) \\
&\leq \frac{1}{\Gamma_k} \lim_{m\to\infty} \sum_{\nu\in\mathcal{M}} \frac{w_\nu}{g_\nu(m)} \\
&= 0, \text{ independently of } \pi.
\end{aligned}$$

We must therefore use a different value function in this case, in order for it to tell the difference between policies.

Indeed, the definitions of value that we have been using are unsatisfactory for semimeasures, because semimeasures take into account the possibility of the 'end of the world': $\mu(e_{1:k}||a_{1:m}) - \sum_{e_{k+1}} \mu(e_{1:k}e_{k+1}||a_{1:m})$ can be thought of as the probability of the environment ending after outputting $e_{1:k}$. The environment might have positive probability of percept sequences that give high reward before simply ending without continuation, and the probability of these sequences is not accounted for in the definitions we have been using.

A more satisfactory recursive definition of the upcoming value at time k is given by [Leike and Hutter 2015b]:

$$W_{km}^{\pi\mu} = \sum_{t=k}^{m} \sum_{e_{k:t}} r_t \mu(e_{k:t}|e_{<k}||a_{1:t}) \tag{7.1}$$

where for all $t > k$, $a_t = \pi(æ_{<t})$. This expression takes into account reward that comes from environments that 'end early', counting the probability of rewards as they come. Note that this can be easily extended to the discounted case, and that when μ is a measure, $W_{km}^{\pi\mu} = V_{km}^{\pi\mu}$. Although we were unable to prove any results using this definition, we feel that results that did use it would be more satisfactory.

Finally, our prior is closely related to the prior $\mu_{\alpha,\gamma}$ defined in Vovk [1989], which predicts so-called '(α,γ)-computable' sequences. If α and γ are functions $\mathbb{N}\to\mathbb{N}$, then a measure ν is said to be (α,γ)-simple if there exists some 'program' $\pi^\nu \in \mathbb{B}^\infty$ such that the UTM with input x outputs $\nu(x)$ in time $\leq \gamma(|x|)$ by reading only $\alpha(|x|)$ bits of π^ν. Vovk proves that if α is logarithmic and γ is polynomial, and if both α and γ are computable in polynomial time, then there exists a measure $\mu_{\alpha,\gamma}$ which is computable in polynomial time that predicts sequences drawn from (α,γ)-simple measures.

S_{Kt} and $\mu_{\alpha,\gamma}$ are similar in spirit: both attempt to predict sequences sampled from an efficiently computable measure. It would therefore be interesting to determine whether S_{Kt} or S_{Fast} is able to predict (α,γ)-simple measures, and to compare other properties of $\mu_{(\alpha,\gamma)}$ and the speed priors.

Appendix A

List of Notation

Symbol	Meaning
$:=$	defined to be equal to
$\approx$	approximately equal to (informal)
$\#A$	cardinality of set A
$\lceil x \rceil$	ceiling of $x \in \mathbb{R}$: smallest integer $\geq x$
$\lfloor x \rfloor$	floor of $x \in \mathbb{R}$: largest integer $\leq x$
i, k, n, t, m	natural numbers
$\mathbb{N}$	set of natural numbers excluding 0
$\mathbb{Z}$	set of integers
$\mathbb{R}$	set of real numbers
$\mathbb{B}$	binary alphabet $\{0, 1\}$
$\mathcal{A}, \mathcal{E}$	finite alphabets
$\mathcal{X}$	generic finite alphabet
$\mathcal{X}^n$	set of strings in $\mathcal{X}$ of length n
$\mathcal{X}^*$	set of finite strings in $\mathcal{X}$
$\mathcal{X}^\infty$	set of infinite strings in $\mathcal{X}$
x, y, p, q	strings of finite length
ϵ	empty string
$\lvert x \rvert$	length of string x
xy	concatenation of strings x and y
x_n	n^{th} symbol of string x
$x_{m:n}$	$:= x_m x_{m+1} \cdots x_n$ if $m \leq n$, ϵ otherwise
$x_{<n}$	$:= x_{1:n-1}$
$x_{1:\infty}$	infinite string
$x \sqsubseteq y$	string x is a prefix of string y, including the case $x = y$
$x \sqsubset y$	string x is a proper prefix of string y, where $x \neq y$
$\langle \cdot \rangle$	prefix-free coding
f, g	functions
$\operatorname{argmax}_x f(x, y)$	some x such that $f(x, y)$ is maximised
$\operatorname{argmin}_x f(x, y)$	some x such that $f(x, y)$ is minimised

(continued on next page)

(continued from previous page)

Symbol	Meaning
$f(n) = O(g(n))$	for some constant $c > 0$ and $n_0 \in \mathbb{N}$, for all $n > n_0$, $f(n) \leq cg(n)$
$a = f(O(g_1(n)), O(g_2(n)), \dots)$	there exists $g'_1(n) = O(g_1(n))$, $g'_2(n) = O(g_2(n))$, $\dots$, such that $a \leq f(g'_1(n), g'_2(n), \dots)$
$f(n) = o(g(n))$	$\lim_{n\to\infty} f(n)/g(n) = 0$
$f(n) \stackrel{\times}{\leq} g(n)$	$f(n) = O(g(n))$ when $f, g : \mathbb{N} \to \mathbb{R}$
$f(x) \stackrel{\times}{\leq} g(x)$	$f(x) \leq cg(x)$ for some constant $c > 0$ when $f, g : \mathbb{B}^* \to \mathbb{R}$
$f \stackrel{\times}{=} g$	$f \stackrel{\times}{\leq} g$ and $g \stackrel{\times}{\leq} f$
log	logarithm base 2
ln	logarithm base e
ε	small positive real number
T	Turing machine (either monotone or mixed-input)
U	universal Turing machine
$p \xrightarrow[\leq t]{T} x$	program p computes x on monotone machine T in $\leq t$ steps, where omitting the $\leq t$ removes that restriction, and omitting T means that $T = U$
$(p, q) \xrightarrow{T} x$	program pair (p, q) computes x on mixed-input machine T
$p \to_i x$	program p computes x in phase i of FAST
$t(T, p, x)$	number of steps taken for program p to compute x on machine T, where omitting T means that $T = U$
$K_T(x)$	length of shortest program that prints x on prefix machine T
$Km_T(x)$	length of shortest program that prints x on monotone machine T
$Kt_T(x)$	$\min_p\{\lvert p\rvert + \log t(T, p, x) : p \xrightarrow{T} x$, and T takes $t(T, p, x)$ steps to print x on input $p\}$
$K(x)$	$:= K_U(x)$
$Km(x)$	$:= Km_U(x)$
$Kt(x)$	$:= Kt_U(x)$
Km-cost(p, x)	$:= \lvert p\rvert$ if $p \to x$, otherwise ∞—minimand of Km
Kt-cost(p, x)	$:= \lvert p\rvert + \log t(p, x)$—minimand of Kt
M	Solomonoff's universal semimeasure
S_{Fast}	the speed prior, as originally defined by Schmidhuber
S_{Kt}	our speed prior
P	some semimeasure, or sometimes measure
P^ε	an estimate of predictive measure P with relative accuracy ε, $\lvert P^\varepsilon/P - 1\rvert \leq \varepsilon$

(continued on next page)

(continued from previous page)

Symbol	Meaning
ν, ρ	some semimeasure, or sometimes measure
μ	the true environmental measure
λ	the uniform measure $\lambda(x) := 2^{-\|x\|}$
$\mathcal{M}$	countable set of semimeasures
$\xi_{\mathcal{M}}$	Bayesian mixture over environmental class $\mathcal{M}$
$\mathbb{E}_\rho$	expectation value w.r.t. the distribution ρ
$H_\mu(x_{1:n})$	binary entropy of the random variable $x_{1:n}$ with respect to μ: $H_\mu(x_{1:n}) = \mathbb{E}_\mu[\log \mu(x_{1:n})]$
$D_n(\mu\|\|\rho)$	relative entropy of ρ w.r.t. the true distribution μ over the first n symbols: $D_n(\mu\|\|\rho) := \mathbb{E}_\mu\left[\ln(\mu(x_{1:n})/\rho(x_{1:n}))\right]$
$\ell(x_t, y_t)$	loss incurred when predicting y_t and the next symbol is x_t
Λ	some way of predicting sequences
$L_{n\nu}^{\Lambda}$	ν-expected cumulative loss in steps 1 through n of predictor Λ
Λ_ρ	predictor with minimal ρ-expected loss
$æ_{k:t}$	$:= a_k e_k a_{k+1} e_{k+1} \cdots a_t e_t$
π	a policy, $\pi : (\mathcal{A} \times \mathcal{E})^* \to \mathcal{A}$
π_m^ν	optimal policy in environment ν with lifetime m
π_k^ν	optimal policy in environment ν at time k
$V_{km}^{\pi\nu}$	expected total reward received between timesteps k and m, $V_{km}^{\pi\nu} = \sum_{e_{k:m}}(r_k + \cdots + r_m)\nu(e_{k:m}\|e_{<k}\|\|a_{1:k})$ where $a_t = \pi(æ_{<t})$ for $t \geq k$

Bibliography

BAILEY, D., BORWEIN, P., AND PLOUFFE, S. 1997. On the rapid computation of various polylogarithmic constants. *Mathematics of Computation of the American Mathematical Society 66*, 218, 903–913. (p. 25)

COX, R. T. 1946. Probability, frequency and reasonable expectation. *American Journal of Physics 14*, 1, 1–13. (p. 2)

GÁCS, P. 1983. On the relation between descriptional complexity and algorithmic probability. *Theoretical Computer Science 22*, 12, 71 – 93. (pp. 11, 26)

HOPCROFT, J. E., MOTWANI, R., AND ULLMAN, J. D. 2001. *Introduction to Automata Theory, Languages, and Computation* (2 ed.). Addison Wesley. (pp. 7, 8)

HUTTER, M. 2005. *Universal Artificial Intelligence: Sequential Decisions Based on Algorithmic Probability*. Springer Science & Business Media. (pp. 3, 4, 9, 13, 16, 18, 27, 38, 40, 43)

HUTTER, M. 2007. On universal prediction and Bayesian confirmation. *Theoretical Computer Science 384*, 1, 33–48. (p. 26)

KOLMOGOROV, A. N. 1965. Three approaches to the quantitative definition of information. *Problems of Information Transmission 1*, 1, 1–7. (pp. 2, 9)

LATTIMORE, T. AND HUTTER, M. 2011. Asymptotically optimal agents. In *Algorithmic Learning Theory* (2011), pp. 368–382. Springer. (p. 17)

LEGG, S. AND HUTTER, M. 2007. A collection of definitions of intelligence. *Frontiers in Artificial Intelligence and applications 157*, 17. (p. 1)

LEIKE, J. AND HUTTER, M. 2015a. Bad universal priors and notions of optimality. In *Conference on Learning Theory* (2015). (pp. 3, 17)

LEIKE, J. AND HUTTER, M. 2015b. On the computability of AIXI. In *Uncertainty in Artificial Intelligence* (2015). (p. 46)

LEVIN, L. A. 1973. On the notion of a random sequence. *Doklady Akademii Nauk SSSR 212*, 3, 548–550. (p. 9)

LI, M. AND VITÁNYI, P. M. 2008. *An Introduction to Kolmogorov Complexity and Its Applications*. Springer Science & Business Media. (pp. 9, 12, 26)

MACKAY, D. J. C. 2003. *Information Theory, Inference and Learning Algorithms*. Cambridge University Press. (p. 3)

MURPHY, K. P. 2000. A survey of POMDP solution techniques. Technical report, University of British Columbia. Accessed 4/10/2015 at `http://www.cs.ubc.ca/~murphyk/Papers/pomdp.pdf`. (p. 17)

ORSEAU, L. 2013. Asymptotic non-learnability of universal agents with computable horizon functions. *Theoretical Computer Science 473*, 149–156. (pp. 3, 17)

PARIS, J. B. 2006. *The Uncertain Reasoner's Companion: A Mathematical Perspective*, Volume 39. Cambridge University Press.

RAMSEY, F. P. 1960. *The Foundations of Mathematics and other Logical Essays*. Littlefield, Adams & Co. Edited by R. B. Braithwaite. (p. 2)

RUSSELL, S. AND NORVIG, P. 2009. *Artificial Intelligence: A Modern Approach* (3 ed.). Pearson. (pp. 1, 42)

RYABKO, D. AND HUTTER, M. 2007. On sequence prediction for arbitrary measures. In *Information Theory, 2007. ISIT 2007. IEEE International Symposium on* (2007), pp. 2346–2350. IEEE. (p. 37)

SCHMIDHUBER, J. 2002. The speed prior: a new simplicity measure yielding near-optimal computable predictions. In *Computational Learning Theory* (2002), pp. 216–228. Springer. (pp. vii, 4, 18)

SOLOMONOFF, R. J. 1964a. A formal theory of inductive inference. Part I. *Information and Control 7*, 1, 1–22. (pp. 2, 10)

SOLOMONOFF, R. J. 1964b. A formal theory of inductive inference. Part II. *Information and Control 7*, 2, 224–254.

SUTTON, R. S. AND BARTO, A. G. 1998. *Reinforcement Learning: An Introduction*. MIT press Cambridge. (pp. 1, 14, 15, 42)

TALBOTT, W. 2015. Bayesian epistemology. In E. N. ZALTA Ed., *The Stanford Encyclopedia of Philosophy* (Summer 2015 ed.). The Metaphysics Research Lab, Center for the Study of Language and Information, Stanford University. (p. 2)

TELLER, P. 1973. Conditionalization and observation. *Synthese 26*, 2, 218–258. (p. 2)

VAN HORN, K. S. 2003. Constructing a logic of plausible inference: a guide to Cox's theorem. *International Journal of Approximate Reasoning 34*, 1, 3–24. (p. 2)

VOVK, V. G. 1989. Prediction of stochastic sequences. *Problemy Peredachi Informatsii 25*, 4, 35–49. (p. 46)