A computational approximation to the AIXI model

Sergey PANKOV
National High Magnetic Field Laboratory
Florida State University
Tallahassee, FL, 32306, USA

Abstract.
Universal induction solves in principle the problem of choosing a prior to achieve optimal inductive inference. The AIXI theory, which combines control theory and universal induction, solves in principle the problem of optimal behavior of an intelligent agent. A practically most important and very challenging problem is to find a computationally efficient (if not optimal) approximation for the optimal but incomputable AIXI theory. We propose such an approximation based on a Monte Carlo algorithm that samples programs according to their algorithmic probability. The approach is specifically designed to deal with real world problems (the agent processes observed data and makes plans over range of divergent time scales) under limited computational resources.

Keywords. AIXI, Solomonoff induction, Kolmogorov complexity, Monte Carlo, algorithmic probability, control theory, intelligent agent

Introduction

There exists a universally optimal theory of artificial intelligence (AI), the AIXI theory [1,2], which combines universal induction [3] with control theory [4]. The AIXI deals with optimal performance (in terms of maximizing its reward) of an agent within an environment in the context of unlimited computational resources. Computationally universally optimal approaches have also been discussed, such as AIXI*tl* [1,2] (which is an approximation to AIXI) or the self-referential Goedel machine [5]. Both approaches are asymptotically optimal, meaning that no other method can possibly outperform (in terms of required computational resources) them in the long run, up to a non-universal additive constant, which may be huge in practice. Both approaches use search for a proof that would justify a particular choice of an agent's strategy or a modification thereof. In the AIXI*tl* the search algorithm is hard-coded, while the Goedel machine permits rewriting of its search algorithm itself, if a provably better algorithm is found. This ability to self-improve (rewrite any part of its code) may significantly reduce the above mentioned non-universal constant.

One could think [5] of using the AIXI*tl* model to initialize the Goedel machine (that is to use it as an initial problem solver of the Goedel machine), which will

eventually self-improve far beyond the initial AIXItl. However, a few legitimate practical questions arise (if one to have in mind complex real world environments), such as: how long might one wait before a significant self-improvement happens? Will not the hidden constants render that time impractically long? Or, should not the initial solver, for example AIXItl, be (at least) "comparably intelligent" to its creator to begin with, if a significant self-improvement were to be expected in the foreseeable future?

Therefore, giving up computational universality would be justifiable at the current stage, provided one gains in low computational complexity on a certain class of problems, deemed (subjectively) important to the field of AI, or artificial general intelligence (AGI). The real value of the AIXI theory is that it provides a prescription for optimal (fastest in the number of agent's observations and actions) way of learning and exploiting the environment. This is analogous to how Solomonoff induction (which, like AIXI, is incomputable), gives a prescription for optimal (fastest in the number of observations) inductive inference. We, therefore, believe that any reasonable computational model of intelligence must recover the AIXI model in the limit of infinite computational resources.

In this paper we present a particular Monte Carlo (MC) algorithm for computational implementation of Solomonoff induction. The algorithm samples candidate programs, that produce a desirable output, according to their algorithmic probability. The idea behind the MC algorithm is to make a Turing machine run "in reverse", from its output to the input program. This guarantees finding some candidate programs, even with very limited computational resources. The agent then uses the discovered programs to predict its possible future observations and corresponding rewards (with predictions possibly reaching far ahead in time). These predicted outcomes are used by the agent to choose its immediate actions, employing a generalized expectation maximization procedure. We argue that our approach reduces to the AIXI model in the limit of infinite computational resources. Finally we discuss what representation of a Turing machine should be most suitable for the approach to be applicable to the real world problems.

The paper is organized as follows. In section 1, we introduce notations and basic concepts related to the universal induction and the AIXI. The main idea of our Monte Carlo algorithm is explained in section 2 in a simple setting of a conventional Turing machine (TM). In section 3, we generalize expectation maximization procedure to long term policies sampled probabilistically under limited computational resources. The general assumptions we make about possible environments relevant to AGI are discussed in section 4. In section 5, we outline a new TM representation, which is meant to accommodate the assumptions made in the previous section.

1. Basics of Solomonoff induction and AIXI model

In this section we introduce notations and briefly overview basic theoretical concepts that are used in this paper.

1.1. Notations

We use strings of symbols as an abstraction for various information transfer processes. Individual symbols in a string drawn from the alphabet $\mathcal{X}$ are denoted x_i, with the subscript indicating their placement in the string. A string of length $l+1$ may be written in one of the following ways: $x_m x_{m+1}...x_{m+l}$, $x_{m:m+l}$ or implicitly $\mathbf{x}$, if the subscript is not needed or its range is understood implicitly. The letter $p \equiv \mathbf{p}$ is reserved for a program string. In some cases we enumerate pairs of symbols (for example $x_1 y_1 x_2 y_2 ...$), then a shorthand notation may be used $xy_i \equiv x_i y_i$, and correspondingly $xy_{i:j} = xy_i xy_{i+1}...xy_j = x_i y_i x_{i+1} y_{i+1}...x_j y_j$. One should not confuse $\mathbf{xy}$ and $\mathbf{xy}$: $x_{1:m} y_{m+1:n} = \mathbf{xy}$ while $xy_{1:m} = \mathbf{xy}$. The symbol $*$ represents any substring. The subscript in $\mathbf{x}_i$ indicates string enumeration, not enumeration of individual symbols. The length of a binary representation of a string $\mathbf{x}$ is denoted $l(\mathbf{x})$, so for $\mathcal{X} = \{0,1\}$ one has $l(x_{1:m}) = m$. Probability of a string $x_{1:m}$ to be sampled from a probability distribution μ is denoted $\mu(x_{1:m})$. Conditional probability for a string $x_{1:n}$ to occur given its first $m < n$ symbols $x_{1:m}$ is defined as $\mu(x_{1:n}|x_{1:m}) \equiv \mu(x_{1:n})/\mu(x_{1:m})$.

1.2. Solomonoff induction

Consider the problem of predicting the symbol x_i given the knowledge of previous $i-1$ symbols $x_{1:i-1}$ in a string sampled from some distribution. The Bayesian inference solves this problem, assuming that the prior probabilities of possible distributions generating the string are known. The problem of choosing the prior probabilities in an optimal way is solved [3] by the universal prior, also called Solomonoff prior. Solomonoff's universal prior $M(\mathbf{x})$ is defined as:

$$M(\mathbf{x}) = \sum_{p:U(p)=\mathbf{x}*} 2^{-l(p)} \tag{1}$$

The summation is over all possible programs p, which output strings starting with $\mathbf{x}$ when run on a universal Turing machine U (see section 2). The inference with the universal prior is called Solomonoff induction (or universal induction). One of the central properties of universal induction is that the (expected) number of incorrectly predicted bits is bounded by the Kolmogorov complexity $K(\rho)$ of the generating distribution function ρ [6,7], (the Kolmogorov complexity [8] is defined, loosely speaking, as the shortest possible description of an object one can produce using a universal Turing machine - mathematical formalization of a general purpose computer). The universal prior can be equivalently presented [2] as a sum over all (semi-)computable (semi-)measures (distribution functions), weighted by their algorithmic probability ($\sim 2^{-K}$):

$$M(\mathbf{x}) \sim \xi(\mathbf{x}) = \sum_{\rho} 2^{-K(\rho)} \rho(\mathbf{x}) \tag{2}$$

1.3. AIXI

All or nearly all formalizable problems in the AI field can be cast in a form of an agent interacting with an environment and seeking to maximize its reward. The agent interacts with its environment in cycles: in ith cycle the agent receives a signal x_i from the environment, processes it, then sends a signal y_i (also called action) to the environment. The subscript i is also referred to as time. With every signal x_i comes a reward $v(x_i)$. Assume for now that the agent is in the beginning of the ith cycle and wants to maximize its expected reward for the next k cycles ahead in the environment described by a known measure μ [4]. To compute the maximized expected reward V^*_{ik} (counting the reward from cycle i to $i+k$), we introduce an auxiliary function V^+_{ik} defined recursively as:

$$V^+_{ik}(xy_{i:j-1}) = \sum_{x_j} \mu(xy_{1:j-1}x_j|xy_{1:j-1}) \max_{y_j} V^+_{ik}(xy_{i:j}) \tag{3}$$

where $i < j \leq i+k$ and $V^+_{ik}(xy_{i:i+k}) = \sum_{j=i}^{j=i+k} v(x_j)$. Then $V^*_{ik}(x_i) = \max_{y_i} V^+_{ik}(xy_i)$. If the measure μ is unknown, the AIXI's prescription is to substitute it with ξ defined in Eq.(2).

2. Monte Carlo algorithm for a "conventional" Turing machine

By a conventional Turing machine we imply a machine described below. This should be contrasted with an effective representation of a Turing machine, discussed in section 5.

A Turing machine [9] is a computational device consisting of a read/write head and an infinite tape (or possibly several tapes). The head assumes some state $s \in S$ and is capable of simple actions, like (reading)writing a symbol $x \in \mathcal{X}$ (from)onto the tape, moving left or right, halting. The number of states $|S|$ and the size of the alphabet $\mathcal{X}$ are finite. The machine's computation is governed by some fixed rules $F : \mathcal{X} \times S \to S \times A$, which defines a step of TM computation as follows. In a state s the machine reads a symbol x, maps the pair (x, s) to (s', a) according to F, assumes new state s' and performs the action $a \in A$. A machine capable of simulating any other machine is called universal and is denoted U.

In Solomonoff/AIXI theory one needs to sample programs p, producing a desired output $\mathbf{x}* = U(p)$, according to the probability $2^{-l(p)}$. In the following we devise a Monte Carlo algorithm for computing time-bound universal prior $M_\tau(\mathbf{x})$, which is constructed analogous to $M(\mathbf{x})$ (Eq.1) but from programs whose runtime does not exceed τ steps of a Turing machine computation.

Note that Solomonoff prior in Eq.(1) is defined via a monotone machine U (see section 3). In a simplified example considered in this section the TM will not be monotone, and only halting programs will be considered. Therefore, the sampled probability will not quite correspond to Eq.(1), and we will consider a single tape machine for illustrative purpose only. However, in the next section we will use monotone machines and will include non-halting computations in consideration, thus restoring exact correspondence to Solomonoff prior.

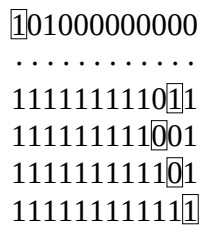

Figure 1. Figure represents a computation of a hypothetical single tape Turing machine, outputting a string of 1s. Each line shows content of the tape as computation progresses (from top to bottom). The transparent box represents the TM's head. The program length $l(p) = 3$. Only the first and a few concluding steps are shown. The machine halts in the last line.

For simplicity we consider a single tape machine with $\mathcal{X} = \{0, 1\}$. The string $\mathbf{x}^\tau$ represents the content of the tape in the τth step of machine's computation. An action $a = (x_t, \Delta t)$ consists of writing x_t in the head's position t, then moving to a position $t' = t + \Delta t$, where $\Delta t = \pm 1$, (then reading a new symbol x'_t, that is understood implicitly). Fig.(1) shows a run of a hypothetical program on such a machine, outputting a string of 1s. The head is initially at $t = 1$ and in the last step τ_n it is at the end of the outputted string.

The triplet $C = (s, \mathbf{x}^\tau, t)$ completely describes the Turing machine in the τth step and is called its τth configuration, (we may also indicate the step index explicitly, as C_τ). We consider a version of a Markov chain Monte Carlo algorithm, known as Metropolis-Hastings algorithm [10,11]. Given a positive function $P(C)$, the algorithm will sample TM configurations with relative probability $P(C)/P(C')$, if run sufficiently long. Choosing

$$P(C = (s, \mathbf{x}, t)) = 2^{-l(\mathbf{x})}, \tag{4}$$

will then achieve our goal of sampling Cs with their algorithmic probabilities, that is $2^{-l(\mathbf{x})}$. In our hypothetical example in Fig.(1) the leftmost and rightmost $1s$ can be thought of as the string $\mathbf{x}$ delimiters, indicating the string's beginning and end.

The Monte Carlo procedure works as follows. Given a configuration C, a move to a configuration C' is proposed according to the distribution function (called proposal density) $Q(C' = (s', \mathbf{x}^{\tau'}, t')|C = (s, \mathbf{x}^\tau, t))$ defined as:

$$Q(C'|C) = \frac{\tilde{Q}(C'|C)}{\sum\limits_{C'} \tilde{Q}(C'|C)},$$

$$\tilde{Q}(C'|C) =$$

$$\begin{cases} 1, \text{ for } \begin{cases} \tau_{max} > \tau' = \tau + 1 > 1, \ F(x_t^\tau, s) = (s', a' = (x_t^{\tau'}, t' - t)), \\ \tau_{max} > \tau = \tau' + 1 > 1, \ F(x_{t'}^{\tau'}, s') = (s, a = (x_{t'}^\tau, t - t')), \end{cases} \\ 0, \text{ otherwise,} \end{cases} \tag{5}$$

where τ_{max} is the limit on runtime of a program. The proposed move is then accepted with probability:

$$\min\left\{\frac{P(C')}{P(C)}\frac{Q(C|C')}{Q(C'|C)},1\right\}. \tag{6}$$

Proposing a MC move and then accepting or rejecting it constitutes a MC step. One can show that the prescribed MC procedure will sample C correctly provided: 1) $P(C)$ is invariant under a MC step (follows directly from Eq.(6)), 2) the procedure is ergodic (in practice it means that any C and C' can be connected by a series of MC moves).

The algorithm starts with $C = (s_h, \mathbf{x}^{\tau_{max}} = \mathbf{x}, t = l(\mathbf{x}))$, where s_h is the halting state and $\mathbf{x}$ is program's desired output. It is assumed that the program starts at $t = 1$ and halts with the head over the last symbol of $\mathbf{x}$. The rationale of the construction of $Q(C'|C)$ as shown in Eq.(5) is that for any current C_τ the sequence of $C_{\tau'}$ for $\tau \leq \tau' \leq \tau_{max}$ represents (as in Fig.(1)) a computation of the Turing machine outputting $\mathbf{x}$. Thus for any C_τ with $t = 1$, corresponding $\mathbf{x}^\tau$ is a program p outputting $\mathbf{x}$. These programs are sampled with frequencies proportional to program's algorithmic probabilities. Notice that the time-bound universal prior can then be evaluated as $M_{\tau_{max}} = 1/\left\langle 2^{l(p)}\right\rangle_{MC}$, where $\langle ...\rangle_{MC}$ is the averaging over the Monte Carlo samples.

We would like to note that one can slightly redefine how a program is supplied to the TM, so that $\mathbf{x}^\tau$ in any configuration C_τ could be regarded as a program (and not only when $t = 1$). Such redefinition will affect the sampling probability by at most a factor of $l(\mathbf{x})$. What one gains is that the MC simulation is guaranteed to produce a candidate program after any number of MC steps. This viewpoint of the algorithm is adopted in section 5.

Our method should be contrasted with a conceptually very different approach of Schmidhuber [12,13], in which one also samples candidate programs according to their algorithmic probability for the purpose of inductive inference. In that approach one uses the speed prior - a time bounded version of the universal prior. The speed prior takes into consideration computational complexity of a program, by allocating computational resources (time) in proportion to its algorithmic probability. So a program p of length $l(p)$ taking time $t(p)$ to run is sampled with frequency $\propto 2^{-l(p)}/t(p)$. This implies that the speed prior sampling (closely related to Levin's universal search [14]) will take time $\mathcal{O}(2^{l(p)}t(p))$ to sample a program p. And although the approach is optimal for finding the simplest program (here, simplest = minimal $l(p) + \log_2 t(p)$), the search time becomes prohibitively large if the simplest program producing a desired output is not short.

Similarly to the speed prior sampling, our MC method will also sample short programs more frequently. The difference is that the correct algorithmic probability is only guaranteed in the long run, because the output string $\mathbf{x}$ may be much longer than the program p outputting it. In such a case the initial configurations C will be highly unlikely in the long run of the MC. This initial stage of sampling unlikely configurations can be viewed as a search for short programs (or at least sufficiently short; here and throughout the paper "sufficiently short" = sufficiently short for discovering interesting regularities), dominating the sampling in the long run. The MC search, which is not universally optimal, in certain cases will quickly find a sufficiently short program. In which cases this is possible will depend on ruggedness of the search landscape, that is on how much $l(\mathbf{x}_\tau)$ varies

along the MC chains (chain = sequence of MC steps). The crucial idea is that the landscape depends on the choice of the Turing machine. We will hypothesize that there is a particular machine choice that makes the MC search very efficient on an important class of problems.

3. Policy computation

We will not discuss computational efficiency of our approach in this paper, we only notice that it is expected to work fast in certain environments described in section 4. For now we simply assume that the MC succeeds in finding sufficiently short programs outputting the data observed by the agent.

A ubiquitous problem faced by the agent is the exponentially large space of states. Only an exponentially small fraction of the states can be sampled in practice when computing the agent's policy. And because in a typical environment of interest (see section 4) an exponentially small fraction of the most probable states have a combined probability close to 1, there should be a strong bias toward those states. It is easy to check that to achieve the quickest convergence in averaging some function $\phi(x)$ over the states x distributed according to $P(x)$, one should sample the states with probability $\propto (|\phi(x) - \bar{\phi}|P(x))^{2/3}$, where $\bar{\phi}$ is the expectation value of $\phi(x)$. In our problem we only know how to efficiently sample $\propto P(x)$ (as will be shown below), but this still gives (typically exponentially) faster convergence than with the uniform sampling. Generally, the strategy of favoring samples that potentially contribute the most to a quantity of interest, is called importance sampling.

The AIXI agent essentially uses Solomonoff induction to predict future and computes its expected reward from that prediction. The universal prior (Eq.(1)) is defined in terms of a (universal) monotone Turing machine [8]. A monotone machine has a unidirectional (unidirectional = head moves in one direction only, to the right) input and output tapes, which are used for reading a program and writing the output of a computation, respectively. The machine also uses internal tapes for computations. The time bounded universal prior $M_\tau(\mathbf{x})$ is now defined in terms of minimal programs (minimal program = input string to the left of the input head when the last symbol of the output string is being printed) that output $\mathbf{x}$ within the time bound τ.

Consider the problem of sampling extensions $\mathbf{y}$ of length l of a given string $\mathbf{x}$ with probability $M_\tau(\mathbf{xy}|\mathbf{x})$. The solution is obvious: first sample a minimal program p outputting $\mathbf{x}$ according to its algorithmic probability, then use fair coin flips to extend the program as far as needed to output l symbols more (within the time bound). How can this be realized within our MC procedure? First, one generalizes the MC to monotone machines. This is straightforward, we just need to consider more complex configurations $C = (s, \mathbf{x}_1 \times ... \times \mathbf{x}_n, t_1 \times ... \times t_n)$ for an n-tape machine, and choose rules F so that the monotone machine is represented appropriately. Second, the MC should start with a non-halting state, so the discovered programs would not halt upon printing $\mathbf{x}$.

When computing agent's policy via V_k^+ (see Eq.(3)), one again faces a problem of exponential growth (here, the growth of the number of possible policies

with k), which seems to render planning impossible for any but very small k. To deal with this problem we propose to use an importance sampling procedure. In particular, the agent should be learning its own actions the way it learns the environment through universal induction. To compute its policy the agent now samples possible policies that are most likely. It then performs the reward expectation maximization on these sampled policies. This procedure is explained below in detail. Notice then, that in the limit of infinite resources the exact expectimax result (as prescribed by Eq.(3)) is recovered.

As we have said, the agent's input and output signals x_i and y_i are now learned on equal footing. The Solomonoff prior is thus defined on the strings $\mathbf{xy}$, (and not just on strings of $\mathbf{x}$, conditioned on $\mathbf{y}$ in chronological manner, see [2]). We shift the agent's cycle index so that it is non-negative integer on future symbols, and negative on its history (the history = observed symbols, is denoted $\mathbf{xy}_{<0}$). We first use the MC procedure to sample the strings from $M_\tau(\mathbf{xy}|\mathbf{xy}_{<0})$ as described above. The ith sampled string is denoted $\mathbf{xy}^{(i)}$, the total reward along the string (starting from the current time = 0) is denoted $V^{(i)}$ and the number of times the string was sampled (multiplicity of the string) is denoted $m^{(i)}$. In the AIXI theory the next action (output signal y_0 in our case) is chosen by computing $V^+_{0k}(xy_{0:j})$ in the space of all possible strings (see Eq.(3)). We want to compute V^+_{0k} over a restricted space of sampled strings only. We want to use frequency of sampling of a particular string as an approximation to the probability measure (as in Eq.(3)). A minor complication relative to Eq.(3) is that sampling now depends on probabilities of actions y_i. An optimal action at (cycle) time n is defined as $y^+_n \equiv \mathrm{argmax}_{y_n} \tilde{V}^+_{0k}(xy_{0:n})$, where $\tilde{V}^+_{0k}$ is the analog of V^+_{0k} defined with approximate probability measure (see Eq.(7)). The idea is to drop the strings in which there are suboptimal choices of y_i. The procedure can be formulated recursively as

$$m^{(i)}_{n-1} = \begin{cases} m^{(i)}_n, & \text{if } y^{(i)}_n = y^+_n \\ 0, & \text{if } y^{(i)}_n \neq y^+_n \end{cases}$$

$$\tilde{V}^+_{0k}(xy_{0:n-1}) = \frac{\sum_{i:xy^{(i)}_{0:n-1}=xy_{0:n-1}} \tilde{V}^+_{0k}(xy^{(i)}_{0:n}) m^{(i)}_{n-1}}{\sum_{i:xy^{(i)}_{0:n-1}=xy_{0:n-1}} m^{(i)}_{n-1}} \tag{7}$$

starting from the horizon $n = k$ and setting there $m^{(i)}_k = m^{(i)}$ and $\tilde{V}^+_{0k}(xy^{(i)}_{0:k}) = V^{(i)}$. The summation in Eq.(7) is over the sampled strings which start with $xy_{0:n-1}$. It is not difficult to check (using Eq.(2)) that in the limit of infinite computational resources (that is $m^{(i)} \to +\infty$ for all possible trajectories) the exact expectation maximization procedure Eq.(3) is recovered.

4. Assumptions about the environment

The approximation to the AIXI that we consider is (most likely) not computationally universal. But we will give a couple reasons which, in our mind, justify such

an approach. First, the known universal schemes [14,15,16] are only asymptotically optimal, that is for a generic problem there likely to be a prohibitively large initial computation, needed to find an optimal algorithm. It is far from obvious how this entry cost can be reduced (for a wide class of interesting problems) without jeopardizing universality. Second, one could wonder about the implication of the anthropic principle for utility of the algorithmic prior, even if the universes are sampled from it [17]. A simple description does not by itself make a universe life-friendly (for example, a program computing digits of π has no room for life). For evolution, and hence for an intelligent life (as we now it), to succeed, a certain degree of space-time repetitiveness in the environment is desired. A bias toward such environments may potentially be very helpful, especially for achieving the goals of artificial general intelligence (AGI).

Motivated by the above ideas of life-friendliness as well as by general observation of the class of problems facing the AGI community, we make certain assumptions about the environment, as explained below. We consider computable environments - environments that can be described by a code, and we have in mind the shortest such code (given a particular environment). Think about the code as organized as a collection of subroutines, calling other subroutines and so on. If removing a particular subroutine call causes only a localized change in the output of a computation, such subroutine is said to represent a feature or detail. The time and extent of the output change define the time and (time) scale of the feature. A hierarchy of subroutine-calls sets the hierarchy of features (details).

A typical environment that we are interested in possesses features on all time scales, and a typical larger scale detail is built on top of smaller (yet comparable) scale details. Description of a typical detail is simple (here, simple = low Kolmogorov complexity), given description of the smaller scale details it is built on. We assume that a typical environment structure is hierarchical and somewhat self-similar, that is the scale of features changes exponentially with hierarchy levels. The details, separated by the time much larger than their scale, can typically be viewed as i.i.d. (independent identically distributed).

The assumptions that we made are not very specific because we do not rigorously investigate computational complexity of our approach. The intention is simply to give a feeling for what kind of problems this approach may be suitable.

5. Redefinition of the Turing machine

In this section we discuss how the idea of using a Metropolis Hastings MC to sample programs from the universal prior should be further expanded and transformed in view of the assumptions about the agent's environment that were made in section 4. Because of the paper length restrictions, we will not go into details here, but rather outline general principles that one should follow. The details will be published elsewhere.

We expect the real world agent to process information and choose its actions in online manner. This means, on one hand, that the agent should be able to reuse fragments of its code, which it has constructed (to explain the data) over its history. On the other hand, most of the processing should be dedicated to

the recent data in a small, relative to the agent's lifespan, time window. In other words, the code explaining observations should be constructed incrementally, yet a typical new piece of code will be strongly connected to the already existing code. For the MC procedure formulated for a conventional Turing machine, which executes its program in a linear fashion, this should be a serious challenge, because of the problem of matching beginning of the new piece to the ending of the old code. (Remember that the MC procedure reconstructs a TM computation in reverse, hence the issue of matching in incremental learning).

Therefore, we need an effective representation of the code, which will allow efficient MC simulation. We list several properties that this representation must have.

- The new representation should conveniently enable reuse of its code. From the point of view of the original Turing machine, one modifies the MC procedure to include nontrivial moves, performing many original moves at once. Quite naturally, the new representation should contain the subroutines of section 4 as building blocks, and the new moves should operate on these blocks (subroutines).
- One consequence of our assumptions is that a typical cause-effect chain that we need to consider is short (it has small number of intermediate links, though time scope can be large), while the number of chains in the time window, where the data is actively processed, can be large. Hence, because the MC algorithm reconstructs a program in reverse (from effect to cause), execution of a program in the new representation should follow causal logical structure of the program, as opposed to a linear execution.
- The new representation should incorporate the hierarchical organization of details, with their scale changes exponentially with hierarchy levels. Because the MC affects the details that have causal effect on current observations, the time window of MC processing changes with the hierarchy level also exponentially. The same can be said about the range of the horizon (how far ahead the agents looks to predict): the smaller scale details have proportionally shorter horizon, than larger scale details.
- Solomonoff induction prescribes to try all possible programs, but assign lower prior probability to longer programs. In practice, due to resource limitations, one can only keep track of most probable programs. This also implies that in online inference one cannot (practically) sample candidate programs independently (a la Levin search), but should instead modify incrementally programs with high posterior probabilities. Some of the modifications that increase the posterior should be taken care automatically. For example, many subroutines will have free parameters. Those parameters should be encoded efficiently, with some kind of Shannon-Fano [18] or (adaptive) Huffman [19] coding, generally following concepts of minimum description length (MDL) [20,21],(or, more broadly, Solomonff induction). Creation of new subroutines is another type of modifications. The probability of creating a particular subroutine should be determined through its algorithmic probability, as a function of its description length. Part of its description will encode calls of certain subprograms which create the new subroutine. The length of encodings of these subprograms is again deter-

mined from the same MDL principle: the larger the number of useful subroutines they create, the shorter their description will be, the more often they will be used. Then there will be higher level subprograms responsible for creating lower level subprograms and so on. These hierarchy of subprograms for creating modifications of the code essentially represent hierarchy of meta-learning levels.

6. Conclusion

We have presented very basics of ideas underlying the computational approximation to the AIXI model that we propose. We simultaneously pursued two goals: 1) to construct an approximation that reduces to the AIXI model in the limit of infinite computational resources, 2) to design an approach that does not sacrifice, when computational resources are limited, important real world capabilities, such as the ability to plan over the range of divergent time scales, from minimal (single cycle) to very large (exponential number of cycles). The details of the approach implementation relevant to the real world environments will be published elsewhere. The main immediate challenge we would like to address is to verify whether our approach is indeed suitable for tackling nontrivial real world problems. One of the steps that we consider in this direction is to test the proposed Monte Carlo algorithm on image recognition.

References

[1] M. Hutter. Lecture Notes in Artificial Intelligence (LNAI 2167), *Proc. 12th Eurpean Conf. on Machine Learning, ECML*, 2001, 226-238

[2] M. Hutter. *Universal Artificial Intelligence: Sequential Decisions Based On Algorithmic Probability.* Springer, 2004.

[3] R. J. Solomonoff. A formal theory of inductive inference: parts 1 and 2. *Information and Control*, 7:1-22, 224-254, 1964.

[4] R. E. Bellman. *Dynamic Programming.* Princeton University Press, Princeton, NJ, 1957.

[5] J. Schmidhuber. Goedel machines: self-referential universal problem solvers making provably optimal self-improvements. In B. Goertzel and C. Pennachin, editors, *Artificial General Intelligence*, 119-226, 2006.

[6] R. J. Solomonoff. Complexity-based induction systems: comparisons and convergence theorems. *IEEE Trans. on Information Theory*, IT-24:422-432, 1978.

[7] M. Hutter. New error bounds for Solomonoff prediction. *J. Computer and System Science* 62(4):653-667, 2001.

[8] M. Li and P. M. B. Vitanyi. *An introduction to Kolmogorov complexity and its applications.* Springer, 2nd edition, 1997.

[9] A. M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proc. London Math. Soc.*, 2(42):230-265, 1936.

[10] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equation of state calculationsby fast computing machines. *Journal of Chemical Physics*, 21:10871092, 1953

[11] W. K. Hastings. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 57(1):97109, 1970

[12] J. Schmidhuber. Discovering neural nets with low Kolmogorov complexity and high generalization capability. *Neural Networks*, 10(5):857-873, (1997)

[13] J. Schmidhuber. The Speed Prior: A New Simplicity Measure Yielding Near-Optimal Computable Predictions. In J. Kivinen and R. H. Sloan, editors, *Proceedings of the 15th Annual Conference on Computational Learning Theory (COLT 2002)*, Sydney, Australia, Lecture Notes in Artificial Intelligence, pages 216–228. Springer, 2002.

[14] L. A. Levin. Universal sequential search problems. *Problems of Information Transmission*, 9(3):265–266, 1973.

[15] J. Schmidhuber. Optimal ordered problem solver. *Machine Learning*, 54:211-254, 2004.

[16] M. Hutter. The fastest and shortest algorithm for all well-defined problems. *International Journal of Foundations of Computer Science*, 13(3):431-443, 2002.

[17] J. Schmidhuber. A Computer Scientist's View of Life, the Universe, and Everything. In C. Freksa, ed., *Foundations of Computer Science: Potential - Theory - Cognition, Lecture Notes in Computer Science*, pp. 201-208, Springer, 1997.

[18] C. E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379-423, 623-656, 1948.

[19] Huffman, D.A. A Method for the Construction of Minimum-Redundancy Codes. *Proceedings of the I.R.E.*, 1952, pp 1098-1102

[20] J. J. Rissanen. *Stochastic complexity in Statistical inquiry.* World Scentific, 1989.

[21] C. S. Wallace. *Statistical and Inductive Inference by Minimum Message Length.* Springer, 2005.