The New AI:
General & Sound & Relevant for Physics

Technical Report IDSIA-04-03, January 2003

To appear in B. Goertzel and C. Pennachin, eds.:

Artificial General Intelligence (2003, submitted 2002)

Jürgen Schmidhuber

juergen@idsia.ch – http://www.idsia.ch/∼juergen

IDSIA, Galleria 2, 6928 Manno (Lugano), Switzerland

Abstract. Most traditional artificial intelligence (AI) systems of the past 50 years are either very limited, or based on heuristics, or both. The new millennium, however, has brought substantial progress in the field of theoretically optimal and practically feasible algorithms for prediction, search, inductive inference based on Occam's razor, problem solving, decision making, and reinforcement learning in environments of a very general type. Since inductive inference is at the heart of all inductive sciences, some of the results are relevant not only for AI and computer science but also for physics, provoking nontraditional predictions based on Zuse's thesis of the computer-generated universe.

1 Introduction

Remarkably, there is a theoretically *optimal* way of making predictions based on observations, rooted in the early work of Solomonoff and Kolmogorov [59,28]. The approach reflects basic principles of Occam's razor: simple explanations of data are preferable to complex ones.

The theory of inductive inference quantifies what simplicity really means. Given certain very broad computability assumptions, it provides techniques for making optimally reliable statements about future events, given the past.

Once there is an optimal, formally describable way of predicting the future, we should be able to construct a machine that continually computes and executes action sequences that maximize expected or predicted reward, thus solving an ancient goal of AI research.

For many decades, however, AI researchers have not paid a lot of attention to the theory of inductive inference. Why not? There is another reason besides the fact that most of them have traditionally ignored theoretical computer science: the theory has been perceived as being associated with excessive computational costs. In fact, its most general statements refer to methods that are optimal (in a certain asymptotic sense) but incomputable. So researchers in machine learning and artificial intelligence have often resorted to alternative methods

that lack a strong theoretical foundation but at least seem feasible in certain limited contexts. For example, since the early attempts at building a "General Problem Solver" [36,43] much work has been done to develop mostly heuristic machine learning algorithms that solve new problems based on experience with previous problems. Many pointers to *learning by chunking, learning by macros, hierarchical learning, learning by analogy,* etc. can be found in Mitchell's book [34] and Kaelbling's survey [27].

Recent years, however, have brought substantial progress in the field of *computable* and *feasible* variants of optimal algorithms for prediction, search, inductive inference, problem solving, decision making, and reinforcement learning in very general environments. In what follows, I will focus on results predominantly from my own lab at IDSIA.

Sections 3, 4, 7 relate Occam's razor and the notion of simplicity to the shortest algorithms for computing computable objects, and will concentrate on recent *asymptotic* optimality results for universal learning machines, essentially ignoring issues of practical feasibility—compare my postdoc Hutter's contribution [25] in this volume.

Section 5, however, will focus on our recent non-traditional simplicity measure which is *not* based on the shortest but on the *fastest* way of describing objects, and Section 6 will use this measure to derive non-traditional predictions concerning the future of our universe.

Sections 8, 9, 10 will finally address quite pragmatic issues and "true" time-optimality: given a problem and only so much limited computation time, what is the best way of spending it on evaluating solution candidates? In particular, Section 9 will present an optimally fast way of incrementally solving each task in a problem sequence, given a probability distribution (the *bias*) on programs computing solution candidates. Bias shifts are computed by program prefixes that modify the distribution on their suffixes by reusing successful code for previous tasks (stored in non-modifiable memory). No tested program gets more runtime than its probability times the total search time. In illustrative experiments, ours becomes the first general system to *learn* a universal solver for arbitrary n disk *Towers of Hanoi* tasks (minimal solution size $2^n - 1$). It demonstrates the advantages of incremental learning by profiting from previously solved, simpler tasks involving samples of a simple context free language. Section 10 discusses how to use this approach for building universal reinforcement learners.

2 More Formally

What is the optimal way of predicting the future, given the past? Which is the best way to act such as to maximize one's future expected reward? Which is the best way of searching for the solution to a novel problem, making optimal use of solutions to earlier problems?

Most previous work on these old and fundamental questions has focused on very limited settings, such as Markovian environments where the optimal next action, given past inputs, depends on the current input only [27].

We will concentrate on a much weaker and therefore much more general assumption, namely, that the environment's responses are sampled from a computable probability distribution. If even this weak assumption were not true then we could not even formally specify the environment, leave alone writing reasonable scientific papers about it.

Let us first introduce some notation. B^* denotes the set of finite sequences over the binary alphabet $B = \{0, 1\}$, B^∞ the set of infinite sequences over B, λ the empty string, $B^\sharp = B^* \cup B^\infty$. x, y, z, z^1, z^2 stand for strings in $B^\sharp$. If $x \in B^*$ then xy is the concatenation of x and y (e.g., if $x = 10000$ and $y = 1111$ then $xy = 100001111$). For $x \in B^*$, $l(x)$ denotes the number of bits in x, where $l(x) = \infty$ for $x \in B^\infty$; $l(\lambda) = 0$. x_n is the prefix of x consisting of the first n bits, if $l(x) \geq n$, and x otherwise ($x_0 := \lambda$). log denotes the logarithm with basis 2, f, g denote functions mapping integers to integers. We write $f(n) = O(g(n))$ if there exist positive constants c, n_0 such that $f(n) \leq cg(n)$ for all $n > n_0$. For simplicity let us consider universal Turing Machines [64] (TMs) with input alphabet B and trinary output alphabet including the symbols "0", "1", and " " (blank). For efficiency reasons, the TMs should have several work tapes to avoid potential quadratic slowdowns associated with 1-tape TMs. The remainder of this paper assumes a fixed universal reference TM.

Now suppose bitstring x represents the data observed so far. What is its most likely continuation $y \in B^\sharp$? Bayes' theorem yields

$$P(xy \mid x) = \frac{P(x \mid xy)P(xy)}{P(x)} \propto P(xy) \tag{1}$$

where $P(z^2 \mid z^1)$ is the probability of z^2, given knowledge of z^1, and $P(x) = \int_{z \in B^\sharp} P(xz)dz$ is just a normalizing factor. So the most likely continuation y is determined by $P(xy)$, the *prior probability* of xy. But which prior measure P is plausible? Occam's razor suggests that the "simplest" y should be more probable. But which exactly is the "correct" definition of simplicity? Sections 3 and 4 will measure the simplicity of a description by its length. Section 5 will measure the simplicity of a description by the time required to compute the described object.

3 Prediction Using a Universal Algorithmic Prior Based on the Shortest Way of Describing Objects

Roughly fourty years ago Solomonoff started the theory of universal optimal induction based on the apparently harmless simplicity assumption that P is computable [59]. While Equation (1) makes predictions of the entire future, given the past, Solomonoff [60] focuses just on the next bit in a sequence. Although this provokes surprisingly nontrivial problems associated with translating the bitwise approach to alphabets other than the binary one — this was achieved only recently [20] — it is sufficient for obtaining essential insights. Given an observed bitstring x, Solomonoff assumes the data are drawn according to a

recursive measure μ; that is, there is a program for a universal Turing machine that reads $x \in B^*$ and computes $\mu(x)$ and halts. He estimates the probability of the next bit (assuming there will be one), using the remarkable, well-studied, enumerable prior M [59,75,60,16,31]

$$M(x) = \sum_{\substack{program \ prefix \ p \ computes \\ output \ starting \ with \ x}} 2^{-l(p)}. \tag{2}$$

M is *universal*, dominating the less general recursive measures as follows: For all $x \in B^*$,

$$M(x) \geq c_\mu \mu(x) \tag{3}$$

where c_μ is a constant depending on μ but not on x. Solomonoff observed that the conditional M-probability of a particular continuation, given previous observations, converges towards the unknown conditional μ as the observation size goes to infinity [60], and that the sum over all observation sizes of the corresponding μ-expected deviations is actually bounded by a constant. Hutter (on the author's SNF research grant ""Unification of Universal Induction and Sequential Decision Theory") recently showed that the number of prediction errors made by universal Solomonoff prediction is essentially bounded by the number of errors made by any other predictor, including the optimal scheme based on the true μ [20].

Recent Loss Bounds for Universal Prediction. A more general recent result is this. Assume we do know that p is in some set P of distributions. Choose a fixed weight w_q for each q in P such that the w_q add up to 1 (for simplicity, let P be countable). Then construct the Bayesmix $M(x) = \sum_q w_q q(x)$, and predict using M instead of the optimal but unknown p. How wrong is it to do that? The recent work of Hutter provides general and sharp (!) loss bounds [21]:

Let $LM(n)$ and $Lp(n)$ be the total expected unit losses of the M-predictor and the p-predictor, respectively, for the first n events. Then $LM(n) - Lp(n)$ is at most of the order of $\sqrt{Lp(n)}$. That is, M is not much worse than p. And in general, no other predictor can do better than that! In particular, if p is deterministic, then the M-predictor soon won't make any errors any more.

If P contains *all* recursively computable distributions, then M becomes the celebrated enumerable universal prior. That is, after decades of somewhat stagnating research we now have sharp loss bounds for Solomonoff's universal induction scheme (compare recent work of Merhav and Feder [33]).

Solomonoff's approach, however, is uncomputable. To obtain a feasible approach, reduce M to what you get if you, say, just add up weighted estimated future finance data probabilities generated by 1000 commercial stock-market prediction software packages. If only one of the probability distributions happens to be close to the true one (but you do not know which) you still should get rich.

Note that the approach is much more general than what is normally done in traditional statistical learning theory, e.g., [66], where the often quite unrealistic assumption is that the observations are statistically independent.

4 Super Omegas and Generalizations of Kolmogorov Complexity & Algorithmic Probability

Our recent research generalized Solomonoff's approach to the case of less restrictive nonenumerable universal priors that are still computable in the limit [50,53].

An object X is formally describable if a finite amount of information completely describes X and only X. More to the point, X should be representable by a possibly infinite bitstring x such that there is a finite, possibly never halting program p that computes x and nothing but x in a way that modifies each output bit at most finitely many times; that is, each finite beginning of x eventually *converges* and ceases to change. This constructive notion of formal describability is less restrictive than the traditional notion of computability [64], mainly because we do not insist on the existence of a halting program that computes an upper bound of the convergence time of p's n-th output bit. Formal describability thus pushes constructivism [5,1] to the extreme, barely avoiding the nonconstructivism embodied by even less restrictive concepts of describability (compare computability *in the limit* [17,40,15] and Δ_n^0-describability [42][31, p. 46-47]).

The traditional theory of inductive inference focuses on Turing machines with one-way write-only output tape. This leads to the universal enumerable Solomonoff-Levin (semi) measure. We introduced more general, nonenumerable, but still limit-computable measures and a natural hierarchy of generalizations of algorithmic probability and Kolmogorov complexity [50,53], suggesting that the "true" information content of some (possibly infinite) bitstring x actually is the size of the shortest nonhalting program that converges to x and nothing but x on a Turing machine that can edit its previous outputs. In fact, this "true" content is often smaller than the traditional Kolmogorov complexity. We showed that there are *Super Omegas* computable in the limit yet more random than Chaitin's "number of wisdom" *Omega* [10] (which is maximally random in a weaker traditional sense), and that any approximable measure of x is small for any x lacking a short description.

We also showed that there is a universal cumulatively enumerable measure of x based on the measure of all enumerable y lexicographically greater than x. It is more dominant yet just as limit-computable as Solomonoff's [53]. That is, if we are interested in limit-computable universal measures, we should prefer the novel universal cumulatively enumerable measure over the traditional enumerable one. If we include in our Bayesmix such limit-computable distributions we obtain again sharp loss bounds for prediction based on the mix [50,53].

Our approach highlights differences between countable and uncountable sets. Which are the potential consequences for physics? We argue that things such as *un*countable time and space and *in*computable probabilities actually should not play a role in explaining the world, for lack of evidence that they are really necessary [50]. Some may feel tempted to counter this line of reasoning by pointing out that for centuries physicists have calculated with continua of real numbers, most of them incomputable. Even quantum physicists who are ready

to give up the assumption of a continuous universe usually do take for granted the existence of continuous probability distributions on their discrete universes, and Stephen Hawking explicitly said: *"Although there have been suggestions that space-time may have a discrete structure I see no reason to abandon the continuum theories that have been so successful."* Note, however, that all physicists in fact have only manipulated discrete symbols, thus generating finite, describable proofs of their results derived from enumerable axioms. That real numbers really *exist* in a way transcending the finite symbol strings used by everybody may be a figment of imagination — compare Brouwer's constructive mathematics [5,1] and the Löwenheim-Skolem Theorem [32,58] which implies that any first order theory with an uncountable model such as the real numbers also has a countable model. As Kronecker put it: *"Die ganze Zahl schuf der liebe Gott, alles Übrige ist Menschenwerk"* ("God created the integers, all else is the work of man" [6]). Kronecker greeted with scepticism Cantor's celebrated insight [7] about real numbers, mathematical objects Kronecker believed did not even exist.

Assuming our future lies among the few (countably many) describable futures, we can ignore uncountably many nondescribable ones, in particular, the random ones. Adding the relatively mild assumption that the probability distribution from which our universe is drawn is cumulatively enumerable provides a theoretical justification of the prediction that the most likely continuations of our universes are computable through short enumeration procedures. In this sense Occam's razor is just a natural by-product of a computability assumption! But what about falsifiability? The pseudorandomness of our universe might be effectively undetectable in principle, because some approximable and enumerable patterns cannot be proven to be nonrandom in recursively bounded time.

The next sections, however, will introduce additional plausible assumptions that do lead to *computable* optimal prediction procedures.

5   Computable Predictions through the Speed Prior Based on the Fastest Way of Describing Objects

Unfortunately, while M and the more general priors of Section 4 are computable in the limit, they are not recursive, and thus practically infeasible. This drawback inspired less general yet practically more feasible principles of minimum description length (MDL) [68,41] as well as priors derived from time-bounded restrictions [31] of Kolmogorov complexity [28,59,8]. No particular instance of these approaches, however, is universally accepted or has a general convincing motivation that carries beyond rather specialized application scenarios. For instance, typical efficient MDL approaches require the specification of a class of computable models of the data, say, certain types of neural networks, plus some computable loss function expressing the coding costs of the data relative to the model. This provokes numerous *ad-hoc* choices.

Our recent work [55], however, offers an alternative to the celebrated but noncomputable algorithmic simplicity measure or Solomonoff-Levin measure discussed above [59,75,60]. We introduced a new measure (a prior on the computable

objects) which is not based on the **shortest** but on the **fastest** way of describing objects.

Let us assume that the observed data sequence is generated by a computational process, and that any possible sequence of observations is therefore computable in the limit [50]. This assumption is stronger and more radical than the traditional one: Solomonoff just insists that the probability of any sequence prefix is recursively computable, but the (infinite) sequence itself may still be generated probabilistically.

Given our starting assumption that data are deterministically generated by a machine, it seems plausible that the machine suffers from a computational resource problem. Since some things are much harder to compute than others, the resource-oriented point of view suggests the following postulate.

Postulate 1 *The cumulative prior probability measure of all x incomputable within time t by any method is at most inversely proportional to t.*

This postulate leads to the Speed Prior $S(x)$, the probability that the output of the following probabilistic algorithm starts with x [55]:

1. Toss an unbiased coin until heads is up; let i denote the number of required trials; set $t := 2^i$.
2. If the number of steps executed so far exceeds t then exit. Execute one step; if this leads to a request for a new input bit (of the growing self-delimiting program, e.g., [30,31]), toss the coin to determine the bit, and set $t := t/2$.
3. Go to **2.**

Algorithm **GUESS** is very similar to a probabilistic search algorithm used in previous work on applied inductive inference [47,49]. On several toy problems it generalized extremely well in a way unmatchable by traditional neural network learning algorithms.

With S comes a computable method **AS** for predicting optimally within ϵ accuracy [55]. Consider a finite but unknown program p computing $y \in B^\infty$. What if Postulate 1 holds but p is not optimally efficient, and/or computed on a computer that differs from our reference machine? Then we effectively do not sample beginnings y_k from S but from an alternative semimeasure S'. Can we still predict well? Yes, because the Speed Prior S dominates S'. This dominance is all we need to apply the recent loss bounds [21]. The loss that we are expected to receive by predicting according to **AS** instead of using the true but unknown S' does not exceed the optimal loss by much [55].

6   Speed Prior-Based Predictions for Our Universe

> *"In the beginning was the code."*
> FIRST SENTENCE OF THE GREAT PROGRAMMER'S BIBLE

Physicists and other inductive scientists make predictions based on observations. Astonishingly, however, few physicists are aware of the theory of *optimal* inductive inference [59,28]. In fact, when talking about the very nature of their inductive business, many physicists cite rather vague concepts such as Popper's falsifiability [39], instead of referring to quantitative results.

All widely accepted physical theories, however, are accepted not because they are falsifiable—they are not—or because they match the data—many alternative theories also match the data—but because they are simple in a certain sense. For example, the theory of gravitation is induced from locally observable training examples such as falling apples and movements of distant light sources, presumably stars. The theory predicts that apples on distant planets in other galaxies will fall as well. Currently nobody is able to verify or falsify this. But everybody believes in it because this generalization step makes the theory simpler than alternative theories with separate laws for apples on other planets. The same holds for superstring theory [18] or Everett's many world theory [13], which presently also are neither verifiable nor falsifiable, yet offer comparatively simple explanations of numerous observations. In particular, most of Everett's postulated many worlds will remain unobservable forever, but the assumption of their existence simplifies the theory, thus making it more beautiful and acceptable.

In Sections 3 and 4 we have made the assumption that the probabilities of next events, given previous events, are (limit-)computable. Here we make a stronger assumption by adopting **Zuse's thesis** [73,74], namely, that the very universe is actually being computed deterministically, e.g., on a cellular automaton (CA) [65,67]. Quantum physics, quantum computation [3,11,38], Heisenberg's uncertainty principle and Bell's inequality [2] do **not** imply any physical evidence against this possibility, e.g., [63].

But then which is our universe's precise algorithm? The following method [48] does compute it:

> Systematically create and execute all programs for a universal computer, such as a Turing machine or a CA; the first program is run for one instruction every second step on average, the next for one instruction every second of the remaining steps on average, and so on.

This method in a certain sense implements the simplest theory of everything: *all* computable universes, including ours and ourselves as observers, are computed by the very short program that generates and executes *all* possible programs [48]. In nested fashion, some of these programs will execute processes that again compute all possible universes, etc. [48]. Of course, observers in "higher-level" universes may be completely unaware of observers or universes computed by

nested processes, and vice versa. For example, it seems hard to track and interpret the computations performed by a cup of tea.

The simple method above is more efficient than it may seem at first glance. A bit of thought shows that it even has the optimal order of complexity. For example, it outputs our universe history as quickly as this history's fastest program, save for a (possibly huge) constant slowdown factor that does not depend on output size.

Nevertheless, some universes are fundamentally harder to compute than others. This is reflected by the Speed Prior S discussed above (Section 5). So let us assume that our universe's history is sampled from S or a less dominant prior reflecting suboptimal computation of the history. Now we can immediately predict:

1. Our universe will not get many times older than it is now [50] — essentially, the probability that it will last 2^n times longer than it has lasted so far is at most 2^{-n}.

2. Any apparent randomness in any physical observation must be due to some yet unknown but *fast* pseudo-random generator PRG [50] which we should try to discover. **2a.** A re-examination of beta decay patterns may reveal that a very simple, fast, but maybe not quite trivial PRG is responsible for the apparently random decays of neutrons into protons, electrons and antineutrinos. **2b.** Whenever there are several possible continuations of our universe corresponding to different Schrödinger wave function collapses — compare Everett's widely accepted many worlds hypothesis [13] — we should be more likely to end up in one computable by a short *and* fast algorithm. A re-examination of split experiment data involving entangled states such as the observations of spins of initially close but soon distant particles with correlated spins might reveal unexpected, nonobvious, nonlocal algorithmic regularity due to a fast PRG.

3. Large scale quantum computation [3] will not work well, essentially because it would require too many exponentially growing computational resources in interfering "parallel universes" [13].

4. Any probabilistic algorithm depending on truly random inputs from the environment will not scale well in practice.

Prediction **2** is verifiable but not necessarily falsifiable within a fixed time interval given in advance. Still, perhaps the main reason for the current absence of empirical evidence in this vein is that few [12] have looked for it.

In recent decades several well-known physicists have started writing about topics of computer science, e.g., [38,11], sometimes suggesting that real world physics might allow for computing things that are not computable traditionally. Unimpressed by this trend, computer scientists have argued in favor of the opposite: since there is no evidence that we need more than traditional computability to explain the world, we should try to make do without this assumption, e.g., [73,74,14,48].

7 Optimal Rational Decision Makers

So far we have talked about passive prediction, given the observations. Note, however, that agents interacting with an environment can also use predictions of the future to compute action sequences that maximize expected future reward. Hutter's recent *AIXI model* [22] (author's SNF grant 61847) does exactly this, by combining Solomonoff's M-based universal prediction scheme with an *expectimax* computation.

In cycle t action y_t results in perception x_t and reward r_t, where all quantities may depend on the complete history. The perception x'_t and reward r_t are sampled from the (reactive) environmental probability distribution μ. Sequential decision theory shows how to maximize the total expected reward, called value, if μ is known. Reinforcement learning [27] is used if μ is unknown. AIXI defines a mixture distribution ξ as a weighted sum of distributions $\nu \in M$, where M is any class of distributions including the true environment μ.

It can be shown that the conditional M probability of environmental inputs to an AIXI agent, given the agent's earlier inputs and actions, converges with increasing length of interaction against the true, unknown probability [22], as long as the latter is recursively computable, analogously to the passive prediction case.

Recent work [24] also demonstrated AIXI's optimality in the following sense. The Bayes-optimal policy p^ξ based on the mixture ξ is self-optimizing in the sense that the average value converges asymptotically for all $\mu \in \mathcal{M}$ to the optimal value achieved by the (infeasible) Bayes-optimal policy p^μ which knows μ in advance. The necessary condition that $\mathcal{M}$ admits self-optimizing policies is also sufficient. No other structural assumptions are made on $\mathcal{M}$. Furthermore, p^ξ is Pareto-optimal in the sense that there is no other policy yielding higher or equal value in *all* environments $\nu \in \mathcal{M}$ and a strictly higher value in at least one [24].

We can modify the AIXI model such that its predictions are based on the ϵ-approximable Speed Prior S instead of the incomputable $\mathcal{M}$. Thus we obtain the so-called *AIS model*. Using Hutter's approach [22] we can now show that the conditional S probability of environmental inputs to an AIS agent, given the earlier inputs and actions, converges against the true but unknown probability, as long as the latter is dominated by S, such as the S' above.

8 Optimal Universal Search Algorithms

In a sense, searching is less general than reinforcement learning because it does not necessarily involve predictions of unseen data. Still, search is a central aspect of computer science (and any reinforcement learner needs a searcher as a submodule—see Section 10). Surprisingly, however, many books on search algorithms do not even mention the following, very simple asymptotically optimal, "universal" algorithm for a broad class of search problems.

Define a probability distribution P on a finite or infinite set of programs for a given computer. P represents the searcher's initial bias (e.g., P could be based on program length, or on a probabilistic syntax diagram).

> Method LSEARCH: Set current time limit T=1. WHILE problem not solved DO:
>
>> Test all programs q such that $t(q)$, the maximal time spent on creating and running and testing q, satisfies $t(q) < P(q)\ T$. Set $T := 2T$.

LSEARCH (for *Levin Search*) may be the algorithm Levin was referring to in his 2 page paper [29] which states that there is an asymptotically optimal universal search method for problems with easily verifiable solutions, that is, solutions whose validity can be quickly tested. Given some problem class, if some unknown optimal program p requires $f(k)$ steps to solve a problem instance of size k, then LSEARCH will need at most $O(P(p)f(k)) = O(f(k))$ steps — the constant factor $P(p)$ may be huge but does not depend on k. Compare [31, p. 502-505] and [23] and the fastest way of computing all computable universes in Section 6.

Recently Hutter developed a more complex asymptotically optimal search algorithm for *all* well-defined problems, not just those with with easily verifiable solutions [23]. HSEARCH cleverly allocates part of the total search time for searching the space of proofs to find provably correct candidate programs with provable upper runtime bounds, and at any given time focuses resources on those programs with the currently best proven time bounds. Unexpectedly, HSEARCH manages to reduce the unknown constant slowdown factor of LSEARCH to a value of $1 + \epsilon$, where ϵ is an arbitrary positive constant.

Unfortunately, however, the search in proof space introduces an unknown *additive* problem class-specific constant slowdown, which again may be huge. While additive constants generally are preferrable over multiplicative ones, both types may make universal search methods practically infeasible.

HSEARCH and LSEARCH are nonincremental in the sense that they do not attempt to minimize their constants by exploiting experience collected in previous searches. Our method *Adaptive* LSEARCH or ALS tries to overcome this [57] — compare Solomonoff's related ideas [61,62]. Essentially it works as follows: whenever LSEARCH finds a program q that computes a solution for the current problem, q's probability $P(q)$ is substantially increased using a "learning rate," while probabilities of alternative programs decrease appropriately. Subsequent LSEARCHes for new problems then use the adjusted P, etc. A nonuniversal variant of this approach was able to solve reinforcement learning (RL) tasks [27] in partially observable environments unsolvable by traditional RL algorithms [71,57].

Each LSEARCH invoked by ALS is optimal with respect to the most recent adjustment of P. On the other hand, the modifications of P themselves are not necessarily optimal. Recent work discussed in the next section overcomes this drawback in a principled way.

9 Optimal Ordered Problem Solver (OOPS)

Our recent OOPS [54,52] is a simple, general, theoretically sound, time-optimal way of searching for a universal behavior or program that solves each problem in a sequence of computational problems, continually organizing and managing and reusing earlier acquired knowledge. For example, the n-th problem may be to compute the n-th event from previous events (prediction), or to find a faster way through a maze than the one found during the search for a solution to the $n-1$-th problem (optimization).

Let us first introduce the important concept of bias-optimality, which is a pragmatic definition of time-optimality, as opposed to the asymptotic optimality of both LSEARCH and HSEARCH, which may be viewed as academic exercises demonstrating that the $O()$ notation can sometimes be practically irrelevant despite its wide use in theoretical computer science. Unlike asymptotic optimality, bias-optimality does not ignore huge constant slowdowns:

Definition 1 (BIAS-OPTIMAL SEARCHERS). *Given is a problem class $\mathcal{R}$, a search space $\mathcal{C}$ of solution candidates (where any problem $r \in \mathcal{R}$ should have a solution in $\mathcal{C}$), a task dependent bias in form of conditional probability distributions $P(q \mid r)$ on the candidates $q \in \mathcal{C}$, and a predefined procedure that creates and tests any given q on any $r \in \mathcal{R}$ within time $t(q, r)$ (typically unknown in advance). A searcher is n-bias-optimal ($n \geq 1$) if for any maximal total search time $T_{max} > 0$ it is guaranteed to solve any problem $r \in \mathcal{R}$ if it has a solution $p \in \mathcal{C}$ satisfying $t(p, r) \leq P(p \mid r) \, T_{max}/n$. It is bias-optimal if $n = 1$.*

This definition makes intuitive sense: the most probable candidates should get the lion's share of the total search time, in a way that precisely reflects the initial bias. Now we are ready to provide a general overview of the basic ingredients of OOPS [54,52]:

Primitives. We start with an initial set of user-defined primitive behaviors. Primitives may be assembler-like instructions or time-consuming software, such as, say, theorem provers, or matrix operators for neural network-like parallel architectures, or trajectory generators for robot simulations, or state update procedures for multiagent systems, etc. Each primitive is represented by a token. It is essential that those primitives whose runtimes are not known in advance can be interrupted at any time.

Task-specific prefix codes. Complex behaviors are represented by token sequences or programs. To solve a given task represented by task-specific program inputs, OOPS tries to sequentially compose an appropriate complex behavior from primitive ones, always obeying the rules of a given user-defined initial programming language. Programs are grown incrementally, token by token; their beginnings or *prefixes* are immediately executed while being created; this may modify some task-specific internal state or memory, and may transfer control back to previously selected tokens (e.g., loops). To add a new token to some program prefix, we first have to wait until the execution of the prefix so far *explicitly requests* such a prolongation, by setting an appropriate signal in the internal state. Prefixes that cease to request any further tokens are called *self-delimiting* programs

or simply programs (programs are their own prefixes). *Binary* self-delimiting programs were studied by [30] and [9] in the context of Turing machines [64] and the theory of Kolmogorov complexity and algorithmic probability [59,28]. OOPS, however, uses a more practical, not necessarily binary framework.

The program construction procedure above yields *task-specific prefix codes* on program space: with any given task, programs that halt because they have found a solution or encountered some error cannot request any more tokens. Given the current task-specific inputs, no program can be the prefix of another one. On a different task, however, the same program may continue to request additional tokens. This is important for our novel approach—incrementally growing self-delimiting programs are unnecessary for the asymptotic optimality properties of LSEARCH and HSEARCH, but essential for OOPS.

Access to previous solutions. Let p^n denote a found prefix solving the first n tasks. The search for p^{n+1} may greatly profit from the information conveyed by (or the knowledge embodied by) $p^1, p^2, \ldots, p^n$ which are stored or *frozen* in special *non*modifiable memory shared by all tasks, such that they are accessible to p^{n+1} (this is another difference to *non*incremental LSEARCH and HSEARCH). For example, p^{n+1} might execute a token sequence that calls p^{n-3} as a subprogram, or that copies p^{n-17} into some internal *modifiable* task-specific memory, then modifies the copy a bit, then applies the slightly edited copy to the current task. In fact, since the number of frozen programs may grow to a large value, much of the knowledge embodied by p^j may be about how to access and edit and use older p^i $(i < j)$.

Bias. The searcher's initial bias is embodied by initial, user-defined, task dependent probability distributions on the finite or infinite search space of possible program prefixes. In the simplest case we start with a maximum entropy distribution on the tokens, and define prefix probabilities as the products of the probabilities of their tokens. But prefix continuation probabilities may also depend on previous tokens in context sensitive fashion.

Self-computed suffix probabilities. In fact, we permit that any executed prefix assigns a task-dependent, self-computed probability distribution to its own possible continuations. This distribution is encoded and manipulated in task-specific internal memory. So unlike with ALS [57] we do not use a prewired learning scheme to update the probability distribution. Instead we leave such updates to prefixes whose online execution modifies the probabilities of their suffixes. By, say, invoking previously frozen code that redefines the probability distribution on future prefix continuations, the currently tested prefix may completely reshape the most likely paths through the search space of its own continuations, based on experience ignored by *non*incremental LSEARCH and HSEARCH. This may introduce significant problem class-specific knowledge derived from solutions to earlier tasks.

Two searches. Essentially, OOPS provides equal resources for two near-*bias-optimal* searches (Def. 1) that run in parallel until p^{n+1} is discovered and stored in non-modifiable memory. The first is exhaustive; it systematically tests all possible prefixes on all tasks up to $n + 1$. Alternative prefixes are tested on all

current tasks in parallel while still growing; once a task is solved, we remove it from the current set; prefixes that fail on a single task are discarded. The second search is much more focused; it only searches for prefixes that start with p^n, and only tests them on task $n+1$, which is safe, because we already know that such prefixes solve all tasks up to n.

Bias-optimal backtracking. HSEARCH and LSEARCH assume potentially infinite storage. Hence they may largely ignore questions of storage management. In any practical system, however, we have to efficiently reuse limited storage. Therefore, in both searches of OOPS, alternative prefix continuations are evaluated by a novel, practical, token-oriented backtracking procedure that can deal with several tasks in parallel, given some *code bias* in the form of previously found code. The procedure always ensures near-*bias-optimality* (Def. 1): no candidate behavior gets more time than it deserves, given the probabilistic bias. Essentially we conduct a depth-first search in program space, where the branches of the search tree are program prefixes, and backtracking (partial resets of partially solved task sets and modifications of internal states and continuation probabilities) is triggered once the sum of the runtimes of the current prefix on all current tasks exceeds the prefix probability multiplied by the total search time so far.

In case of unknown, infinite task sequences we can typically never know whether we already have found an optimal solver for all tasks in the sequence. But once we unwittingly do find one, at most half of the total future run time will be wasted on searching for alternatives. Given the initial bias and subsequent bias shifts due to $p^1, p^2, \ldots$, no other bias-optimal searcher can expect to solve the $n+1$-th task set substantially faster than OOPS. A by-product of this optimality property is that it gives us a natural and precise measure of bias and bias shifts, conceptually related to Solomonoff's *conceptual jump size* of [61,62].

Since there is no fundamental difference between domain-specific problem-solving programs and programs that manipulate probability distributions and thus essentially rewrite the search procedure itself, we collapse both learning and metalearning in the same time-optimal framework.

An example initial language. For an illustrative application, we wrote an interpreter for a stack-based universal programming language inspired by FORTH [35], with initial primitives for defining and calling recursive functions, iterative loops, arithmetic operations, and domain-specific behavior. Optimal metasearching for better search algorithms is enabled through the inclusion of bias-shifting instructions that can modify the conditional probabilities of future search options in currently running program prefixes.

Experiments. Using the assembler-like language mentioned above, we first teach OOPS something about recursion, by training it to construct samples of the simple context free language $\{1^k 2^k\}$ (k 1's followed by k 2's), for k up to 30 (in fact, the system discovers a universal solver for all k). This takes roughly 0.3 days on a standard personal computer (PC). Thereafter, within a few additional days, OOPS demonstrates incremental knowledge transfer: it exploits aspects of its previously discovered universal $1^k 2^k$-solver, by rewriting its search procedure such that it more readily discovers a universal solver for all k disk *Towers of Hanoi*

problems—in the experiments it solves all instances up to $k = 30$ (solution size $2^k - 1$), but it would also work for $k > 30$. Previous, less general reinforcement learners and *non*learning AI planners tend to fail for much smaller instances.

Future research may focus on devising particularly compact, particularly reasonable sets of initial codes with particularly broad practical applicability. It may turn out that the most useful initial languages are not traditional programming languages similar to the FORTH-like one, but instead based on a handful of primitive instructions for massively parallel cellular automata [65,67,74,72], or on a few nonlinear operations on matrix-like data structures such as those used in recurrent neural network research [69,44,4]. For example, we could use the principles of OOPS to create a non-gradient-based, near-bias-optimal variant of Hochreiter's successful recurrent network metalearner [19]. It should also be of interest to study probabilistic *Speed Prior*-based OOPS variants [55] and to devise applications of OOPS-like methods as components of universal reinforcement learners (see below). In ongoing work, we are applying OOPS to the problem of optimal trajectory planning for robotics in a realistic physics simulation. This involves the interesting trade-off between comparatively fast program-composing primitives or *"thinking primitives"* and time-consuming *"action primitives"*, such as *stretch-arm-until-touch-sensor-input*.

10 OOPS-Based Reinforcement Learning

At any given time, a reinforcement learner [27] will try to find a *policy* (a strategy for future decision making) that maximizes its expected future reward. In many traditional reinforcement learning (RL) applications, the policy that works best in a given set of training trials will also be optimal in future test trials [51]. Sometimes, however, it won't. To see the difference between searching (the topic of the previous sections) and reinforcement learning (RL), consider an agent and two boxes. In the n-th trial the agent may open and collect the content of exactly one box. The left box will contain $100n$ Swiss Francs, the right box 2^n Swiss Francs, but the agent does not know this in advance. During the first 9 trials the optimal policy is *"open left box."* This is what a good searcher should find, given the outcomes of the first 9 trials. But this policy will be suboptimal in trial 10. A good reinforcement learner, however, should extract the underlying regularity in the reward generation process and predict the future reward, picking the right box in trial 10, without having seen it yet.

The first general, asymptotically optimal reinforcement learner is the recent AIXI model [22,24] (Section 7). It is valid for a very broad class of environments whose reactions to action sequences (control signals) are sampled from arbitrary computable probability distributions. This means that AIXI is far more general than traditional RL approaches. However, while AIXI clarifies the theoretical limits of RL, it is not practically feasible, just like HSEARCH is not. From a pragmatic point of view, what we are really interested in is a reinforcement learner that makes optimal use of given, limited computational resources. In

what follows, we will outline how to use OOPS-like bias-optimal methods as components of universal yet feasible reinforcement learners.

We need two OOPS modules. The first is called the predictor or world model. The second is an action searcher using the world model. The life of the entire system should consist of a sequence of *cycles* 1, 2, ... At each cycle, a limited amount of computation time will be available to each module. For simplicity we assume that during each cyle the system may take exactly one action. Generalizations to actions consuming several cycles are straight-forward though. At any given cycle, the system executes the following procedure:

1. For a time interval fixed in advance, the predictor is first trained in bias-optimal fashion to find a better world model, that is, a program that predicts the inputs from the environment (including the rewards, if there are any), given a history of previous observations and actions. So the n-th task ($n = 1, 2, \ldots$) of the first OOPS module is to find (if possible) a better predictor than the best found so far.

2. Once the current cycle's time for predictor improvement is used up, the current world model (prediction program) found by the first OOPS module will be used by the second module, again in bias-optimal fashion, to search for a future action sequence that maximizes the predicted cumulative reward (up to some time limit). That is, the n-th task ($n = 1, 2, \ldots$) of the second OOPS module will be to find a control program that computes a control sequence of actions, to be fed into the program representing the current world model (whose input predictions are successively fed back to itself in the obvious manner), such that this control sequence leads to higher predicted reward than the one generated by the best control program found so far.

3. Once the current cycle's time for control program search is used up, we will execute the current action of the best control program found in step 2. Now we are ready for the next cycle.

The approach is reminiscent of an earlier, heuristic, non-bias-optimal RL approach based on two adaptive recurrent neural networks, one representing the world model, the other one a controller that uses the world model to extract a policy for maximizing expected reward [46]. The method was inspired by previous combinations of *non*recurrent, *reactive* world models and controllers [70,37,26].

At any given time, until which temporal horizon should the predictor try to predict? In the AIXI case, the proper way of treating the temporal horizon is not to discount it exponentially, as done in most traditional work on reinforcement learning, but to let the future horizon grow in proportion to the learner's lifetime so far [24]. It remains to be seen whether this insight carries over to OOPS-based RL. In particular, is it possible to prove that certain OOPS-RL variants are a near-bias-optimal way of spending a given amount of computation time on RL problems? Or should we instead combine OOPS and Hutter's time-bounded AIXI(t, l) model? We observe that certain important problems are still open.

11   Conclusion

Recent theoretical and practical advances are currently driving a renaissance in the fields of universal learners and optimal search [56]. A new kind of AI is emerging. Does it really deserve the attribute *"new,"* given that its roots date back to the 1960s, just two decades after Zuse built the first general purpose computer in 1941? An affirmative answer seems justified, since it is the recent results on practically feasible computable variants of the old incomputable methods that are currently reinvigorating the long dormant field. The "new" AI is new in the sense that it abandons the mostly heuristic or non-general approaches of the past decades, offering methods that are both general and theoretically sound, and provably optimal in a sense that *does* make sense in the real world.

We are led to claim that the future will belong to universal or near-universal learners that are more general than traditional reinforcement learners / decision makers depending on strong Markovian assumptions, or than learners based on traditional statistical learning theory, which often require unrealistic i.i.d. or Gaussian assumptions. Due to ongoing hardware advances the time has come for optimal search in algorithm space, as opposed to the limited space of reactive mappings embodied by traditional methods such as artificial feedforward neural networks.

It seems safe to bet that not only computer scientists but also physicists and other inductive scientists will start to pay more attention to the fields of universal induction and optimal search, since their basic concepts are irresistibly powerful and general and simple. How long will it take for these ideas to unfold their full impact? A very naive and speculative guess driven by wishful thinking might be based on identifying the *"greatest moments in computing history"* and extrapolating from there. Which are those "greatest moments"? Obvious candidates are:

1. *1640:* first mechanical calculator (Pascal, France).
2. *Two centuries later:* concept of a programmable computer (Babbage, UK).
3. *One century later:* first working programmable computer (Zuse, Berlin), plus fundamental theoretical work on universal integer-based programming languages and the limits of proof and computation (Gödel, Austria, reformulated by Turing, UK). (The next 50 years saw many theoretical advances as well as faster and faster switches—relays were replaced by tubes, tubes by transistors, single transistors by numerous transistors etched on chips—but arguably this was rather predictable, incremental progress without radical shake-up events.)
4. *Half a century later:* the World Wide Web (UK's Berners-Lee, Switzerland).

This list seems to suggest that each major breakthrough tends to come twice as fast as the previous one. Extrapolating the trend, optimists should expect another radical change by 2015, which happens to coincide with the date when the fastest computers will match brains in terms of raw computing power, according to frequent estimates based on Moore's law. The author is confident that the coming 2015 upheaval (if any) will involve universal learning algorithms and

optimal incremental search in algorithm space—possibly laying a foundation for the remaining series of faster and faster additional revolutions culminating in an "Omega point" expected around 2040.

12   Acknowledgments

References

1. M. Beeson. *Foundations of Constructive Mathematics.* Springer-Verlag, Heidelberg, 1985.
2. J. S. Bell. On the problem of hidden variables in quantum mechanics. *Rev. Mod. Phys.*, 38:447–452, 1966.
3. C. H. Bennett and D. P. DiVicenzo. Quantum information and computation. *Nature*, 404(6775):256–259, 2000.
4. C. M. Bishop. *Neural networks for pattern recognition.* Oxford University Press, 1995.
5. L. E. J. Brouwer. Over de Grondslagen der Wiskunde. Dissertation, Doctoral Thesis, University of Amsterdam, 1907.
6. F. Cajori. *History of mathematics (2nd edition).* Macmillan, New York, 1919.
7. G. Cantor. Über eine Eigenschaft des Inbegriffes aller reellen algebraischen Zahlen. *Crelle's Journal für Mathematik*, 77:258–263, 1874.
8. G.J. Chaitin. On the length of programs for computing finite binary sequences: statistical considerations. *Journal of the ACM*, 16:145–159, 1969.
9. G.J. Chaitin. A theory of program size formally identical to information theory. *Journal of the ACM*, 22:329–340, 1975.
10. G.J. Chaitin. *Algorithmic Information Theory.* Cambridge University Press, Cambridge, 1987.
11. D. Deutsch. *The Fabric of Reality.* Allen Lane, New York, NY, 1997.
12. T. Erber and S. Putterman. Randomness in quantum mechanics – nature's ultimate cryptogram? *Nature*, 318(7):41–43, 1985.
13. H. Everett III. 'Relative State' formulation of quantum mechanics. *Reviews of Modern Physics*, 29:454–462, 1957.
14. E. F. Fredkin and T. Toffoli. Conservative logic. *International Journal of Theoretical Physics*, 21(3/4):219–253, 1982.
15. R. V. Freyvald. Functions and functionals computable in the limit. *Transactions of Latvijas Vlasts Univ. Zinatn. Raksti*, 210:6–19, 1977.
16. P. Gács. On the relation between descriptional complexity and algorithmic probability. *Theoretical Computer Science*, 22:71–93, 1983.
17. E. M. Gold. Limiting recursion. *Journal of Symbolic Logic*, 30(1):28–46, 1965.
18. M.B. Green, J.H. Schwarz, and E. Witten. *Superstring Theory.* Cambridge University Press, 1987.

19. S. Hochreiter, A. S. Younger, and P. R. Conwell. Learning to learn using gradient descent. In *Lecture Notes on Comp. Sci. 2130, Proc. Intl. Conf. on Artificial Neural Networks (ICANN-2001)*, pages 87–94. Springer: Berlin, Heidelberg, 2001.

20. M. Hutter. Convergence and error bounds of universal prediction for general alphabet. *Proceedings of the 12th European Conference on Machine Learning (ECML-2001)*, (TR IDSIA-07-01, cs.AI/0103015), 2001.

21. M. Hutter. General loss bounds for universal sequence prediction. In C. E. Brodley and A. P. Danyluk, editors, *Proceedings of the 18th International Conference on Machine Learning (ICML-2001)*, pages 210–217. Morgan Kaufmann, 2001. TR IDSIA-03-01, IDSIA, Switzerland, Jan 2001, cs.AI/0101019.

22. M. Hutter. Towards a universal theory of artificial intelligence based on algorithmic probability and sequential decisions. *Proceedings of the 12th European Conference on Machine Learning (ECML-2001)*, (TR IDSIA-14-00, cs.AI/0012011):226–238, 2001.

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

24. M. Hutter. Self-optimizing and Pareto-optimal policies in general environments based on Bayes-mixtures. In J. Kivinen and R. H. Sloan, editors, *Proceedings of the 15th Annual Conference on Computational Learning Theory (COLT 2002)*, Lecture Notes in Artificial Intelligence, pages 364–379, Sydney, Australia, 2002. Springer.

25. M. Hutter. A gentle introduction to the universal algorithmic agent AIXI. In B. Goertzel and C. Pennachin, editors, *Real AI: New Approaches to Artificial General Intelligence*. Plenum Press, New York, 2003. To appear.

26. M. I. Jordan and D. E. Rumelhart. Supervised learning with a distal teacher. Technical Report Occasional Paper #40, Center for Cog. Sci., Massachusetts Institute of Technology, 1990.

27. L.P. Kaelbling, M.L. Littman, and A.W. Moore. Reinforcement learning: a survey. *Journal of AI research*, 4:237–285, 1996.

28. A.N. Kolmogorov. Three approaches to the quantitative definition of information. *Problems of Information Transmission*, 1:1–11, 1965.

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

30. L. A. Levin. Laws of information (nongrowth) and aspects of the foundation of probability theory. *Problems of Information Transmission*, 10(3):206–210, 1974.

31. M. Li and P. M. B. Vitányi. *An Introduction to Kolmogorov Complexity and its Applications (2nd edition)*. Springer, 1997.

32. L. Löwenheim. Über Möglichkeiten im Relativkalkül. *Mathematische Annalen*, 76:447–470, 1915.

33. N. Merhav and M. Feder. Universal prediction. *IEEE Transactions on Information Theory*, 44(6):2124–2147, 1998.

34. T. Mitchell. *Machine Learning*. McGraw Hill, 1997.

35. C. H. Moore and G. C. Leach. FORTH - a language for interactive computing, 1970. http://www.ultratechnology.com.

36. A. Newell and H. Simon. GPS, a program that simulates human thought. In E. Feigenbaum and J. Feldman, editors, *Computers and Thought*, pages 279–293. McGraw-Hill, New York, 1963.

37. Nguyen and B. Widrow. The truck backer-upper: An example of self learning in neural networks. In *Proceedings of the International Joint Conference on Neural Networks*, pages 357–363. IEEE Press, 1989.

38. R. Penrose. *The Emperor's New Mind.* Oxford University Press, 1989.

39. K. R. Popper. *The Logic of Scientific Discovery.* Hutchinson, London, 1934.

40. H. Putnam. Trial and error predicates and the solution to a problem of Mostowski. *Journal of Symbolic Logic*, 30(1):49–57, 1965.

41. J. Rissanen. Stochastic complexity and modeling. *The Annals of Statistics*, 14(3):1080–1100, 1986.

42. H. Rogers, Jr. *Theory of Recursive Functions and Effective Computability.* McGraw-Hill, New York, 1967.

43. P. S. Rosenbloom, J. E. Laird, and A. Newell. *The SOAR Papers.* MIT Press, 1993.

44. D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing*, volume 1, pages 318–362. MIT Press, 1986.

45. C. Schmidhuber. Strings from logic. Technical Report CERN-TH/2000-316, CERN, Theory Division, 2000. http://xxx.lanl.gov/abs/hep-th/0011065.

46. J. Schmidhuber. Reinforcement learning in Markovian and non-Markovian environments. In D. S. Lippman, J. E. Moody, and D. S. Touretzky, editors, *Advances in Neural Information Processing Systems 3*, pages 500–506. Morgan Kaufmann, 1991.

47. J. Schmidhuber. Discovering solutions with low Kolmogorov complexity and high generalization capability. In A. Prieditis and S. Russell, editors, *Machine Learning: Proceedings of the Twelfth International Conference*, pages 488–496. Morgan Kaufmann Publishers, San Francisco, CA, 1995.

48. J. Schmidhuber. A computer scientist's view of life, the universe, and everything. In C. Freksa, M. Jantzen, and R. Valk, editors, *Foundations of Computer Science: Potential - Theory - Cognition*, volume 1337, pages 201–208. Lecture Notes in Computer Science, Springer, Berlin, 1997. Submitted 1996.

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

50. J. Schmidhuber. Algorithmic theories of everything. Technical Report IDSIA-20-00, quant-ph/0011122, IDSIA, Manno (Lugano), Switzerland, 2000.

51. J. Schmidhuber. Sequential decision making based on direct search. In R. Sun and C. L. Giles, editors, *Sequence Learning: Paradigms, Algorithms, and Applications.* Springer, 2001. Lecture Notes on AI 1828.

52. J. Schmidhuber. Bias-optimal incremental problem solving. In *Advances in Neural Information Processing Systems 15*. MIT Press, Cambridge, MA, 2002. To appear.

53. J. Schmidhuber. Hierarchies of generalized Kolmogorov complexities and nonenumerable universal measures computable in the limit. *International Journal of Foundations of Computer Science*, 13(4):587–612, 2002.

54. J. Schmidhuber. Optimal ordered problem solver. Technical Report IDSIA-12-02, arXiv:cs.AI/0207097 v1, IDSIA, Manno-Lugano, Switzerland, July 2002.

55. 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)*, Lecture Notes in Artificial Intelligence, pages 216–228. Springer, Sydney, Australia, 2002.

56. J. Schmidhuber and M. Hutter. NIPS 2002 workshop on universal learning algorithms and optimal search. Additional speakers: R. Solomonoff, P. M. B. Vitányi, N. Cesa-Bianchi, I. Nemenmann. Whistler, CA, 2002.

57. J. Schmidhuber, J. Zhao, and M. Wiering. Shifting inductive bias with success-story algorithm, adaptive Levin search, and incremental self-improvement. *Machine Learning*, 28:105–130, 1997.

58. T. Skolem. Logisch-kombinatorische Untersuchungen über Erfüllbarkeit oder Beweisbarkeit mathematischer Sätze nebst einem Theorem über dichte Mengen. *Skrifter utgit av Videnskapsselskapet in Kristiania, I, Mat.-Nat. Kl.*, N4:1–36, 1919.

59. R.J. Solomonoff. A formal theory of inductive inference. Part I. *Information and Control*, 7:1–22, 1964.

60. R.J. Solomonoff. Complexity-based induction systems. *IEEE Transactions on Information Theory*, IT-24(5):422–432, 1978.

61. R.J. Solomonoff. An application of algorithmic probability to problems in artificial intelligence. In L. N. Kanal and J. F. Lemmer, editors, *Uncertainty in Artificial Intelligence*, pages 473–491. Elsevier Science Publishers, 1986.

62. R.J. Solomonoff. A system for incremental learning based on algorithmic probability. In *Proceedings of the Sixth Israeli Conference on Artificial Intelligence, Computer Vision and Pattern Recognition*, pages 515–527. Tel Aviv, Israel, 1989.

63. G. 't Hooft. Quantum gravity as a dissipative deterministic system. Technical Report SPIN-1999/07/gr-gc/9903084, http://xxx.lanl.gov/abs/gr-qc/9903084, Institute for Theoretical Physics, Univ. of Utrecht, and Spinoza Institute, Netherlands, 1999. Also published in *Classical and Quantum Gravity 16*, 3263.

64. A. M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society, Series 2*, 41:230–267, 1936.

65. S. Ulam. Random processes and transformations. In *Proceedings of the International Congress on Mathematics*, volume 2, pages 264–275, 1950.

66. V. Vapnik. *The Nature of Statistical Learning Theory*. Springer, New York, 1995.

67. J. von Neumann. *Theory of Self-Reproducing Automata*. University of Illionois Press, Champain, IL, 1966.

68. C. S. Wallace and D. M. Boulton. An information theoretic measure for classification. *Computer Journal*, 11(2):185–194, 1968.

69. P. J. Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. PhD thesis, Harvard University, 1974.

70. P. J. Werbos. Learning how the world works: Specifications for predictive networks in robots and brains. In *Proceedings of IEEE International Conference on Systems, Man and Cybernetics, N.Y.*, 1987.

71. M.A. Wiering and J. Schmidhuber. Solving POMDPs with Levin search and EIRA. In L. Saitta, editor, *Machine Learning: Proceedings of the Thirteenth International Conference*, pages 534–542. Morgan Kaufmann Publishers, San Francisco, CA, 1996.

72. S. Wolfram. Universality and complexity in cellular automata. *Physica D*, 10:1–35, 1984.

73. K. Zuse. Rechnender Raum. *Elektronische Datenverarbeitung*, 8:336–344, 1967.

74. K. Zuse. *Rechnender Raum*. Friedrich Vieweg & Sohn, Braunschweig, 1969.

75. A. K. Zvonkin and L. A. Levin. The complexity of finite objects and the algorithmic concepts of information and randomness. *Russian Math. Surveys*, 25(6):83–124, 1970.