THE DISCOVERY OF ALGORITHMIC PROBABILITY

Ray J. Solomonoff *

Royal Holloway, University of London
Mailing Address: P.O.B. 400404, Cambridge, Ma. 02140, U.S.A.
rjsolo@ieee.org

1 Introduction

This paper will describe a voyage of discovery — the discovery of Algorithmic Probability. But before I describe that voyage —a few words about motivation.

Motivation in science is roughly of two kinds: In one, the motivation is discovery itself — the joy of "going where no one has gone before" — the excitement of creating new universes and exploring them.

Another kind of motivation is the achievement of a previously defined larger goal, and there may be many subsidiary discoveries on the path to this goal.

In my own case both kinds of motivation were very strong. I first experienced the pure joy of mathematical discovery when I was very young — learning algebra. I didn't really see why one of the axioms was "obvious", so I tried reversing it and exploring the resultant system. For a few hours I was in this wonderful never-never land that I myself had created! The joy of exploring it only lasted until it became clear that my new axiom wouldn't work, but the motivation of the joy of discovery continued for the rest of my life.

The motivation for the discovery of Algorithmic Probability was somewhat different. It was the result of "*goal* motivated discovery" — like the discovery of the double helix in biology, but with fewer people involved and relatively little political skullduggery.

The goal I set grew out of my early interest in science and mathematics. I found that while the discoveries of the past were interesting to me, I was

*This work was supported in part by the United States Air Force Office of Scientific Research under contracts AF-19(628)5975, AF-49(638)-376, and Grant AF AFOSR 62-377; in part by the Advanced Research Projects Agency of the Department of Defence under Office of Naval Research Contracts N00014-70- A-0362-0003 and N00014-70-A-0362-0005 and in part by the Public Health Service under NIH Grant GM 11021-01.

even more interested in how people discovered things. Was there a general technique to solve all mathematical problems? Was there a general method by which scientists could discover all scientific truths? The problems seemed closely related to induction and so around 1942 I first defined a general induction problem. It had two aspects:

The first I called MTM (Mathematical Thinking Machine). The problem is to do induction when the model generating the data is known to be deterministic (non-probabilistic).

The second I called NMTM (Non-MTM). The problem is to do induction when the model may be probabilistic.

At the time, I felt that the second problem was more difficult; that it was what scientists did when they invented theories to account for data. For both problems, I wanted real, practically realizable solutions, though as "study problems" I considered cases in which computation might be impractical [1].

I was well aware that the problems were very difficult, and I expected to spend the rest of my life working on them.

As is normal under such circumstances, the definitions of the problems changed as I moved toward solutions.

When embarking on a very difficult task, it is well to prepare for it by collecting tools that are likely to be useful. In this case the tools were kinds of knowledge. Some tools that seemed relevant:

1. A good knowledge of science and how scientists make discoveries.

2. A good understanding of mathematics and how to apply it to all kinds of problems.

3. Understanding of probability and statistics.

4. A general knowledge of human activities — how people solve problems, how they think they solve problems, how they make predictions.

Of the four items, knowledge of science and math has been most useful. Studying probability revealed important deficiencies in our understanding of it. I found surprisingly little in conventional statistics to be relevant to my goals.

Understanding humans is certainly important when living in a community of them, but applying this understanding to my goals has become relevant only after I had discovered Algorithmic Probability. At this point the politics and rhetoric of science become dominant factors influencing the reception and perception of my discovery by the scientific community.

My university education began in 1946: I chose the University of Chicago because it had very good mathematics and physics departments. It was also very good in the humanities — a part of my education that I had hitherto neglected. I decided to major in physics because it was the most general of the sciences, and through it, I could learn to apply mathematics to problems of all kinds.

[1] Years later I found the solutions to the two problems to be identical.

In addition to physics and math, I studied the logical basis of probability with Carnap, and mathematical biology with Rapoport and Rashevsky. I left the University in 1951 with an MS in physics, and I began half time work in industry as a mathematician-physicist. The rest of my time was spent on induction research.

The next section gives some important influences on my thought up to the time I left the University.

2 Through The University

2.1 Bridgman

My earliest contact with modern scientific philosophy may have been P.W. Bridgman's concept of "operational definition"(Bri 27). An operational definition of anything is a precise sequence of physical operations that enable one to either construct it or identify it with certainty. When one can't make an operational definition of something, this is usually an indication of poor understanding of it. Attempts to operationalize definitions can be guides to discovery. I've found this idea to be an invaluable tool in telling me whether or not I really understand something.

Though individual concepts in a theory need not be operational, the theory as a whole must have operational meaning. Rapoport (Rap 53) discusses the application of operational concepts in daily life as well in science.

2.2 Korzybski

Alfred Korzybski has been variously described as charlatan, dilettante and genius. He summarized what he felt were the most important general principles in the history and philosophy of science, and tried to apply those principles to everyday life as well as to scientific investigation. I found his major work, "Science and Sanity" (Kor 58) unreadable, but I got several important heuristic principles from authors that interpreted and/or popularized his work (Rap 53, Joh 46, Hay 41).

A few principles:

(a) The map is not the same as the territory it purportedly describes. My interpretation of this idea has expanded over the years. Korzybski's emphasis was on features of the territory that the map did not have. Here, we mean "territories" and "maps" in a very general sense — anything in the real world and something that is supposed to describe the thing in the real world. Since then I have learned to appreciate the importance of the rich heuristic associations of maps — features that the territories don't necessarily have, but nevertheless make it much easier for us to understand (and often misunderstand!)the territories.

(b) Two valued logic is usually inappropriate for dealing with events in the real world. Part of this takes the form of a gray scale for both data and predictions. Zadeh's "Fuzzy Sets" (Zad 65) may be regarded as one way to develop this idea. Probability theory is another.

(c) When working on a difficult problem, usually people break it down into subproblems, and try to work the sub-problems –but often the set of subproblems is not solvable and/or is not really equivalent to the main problem. Either try to break the problem down in a different way or solve the main problem directly. This last would be the "holistic approach".

(d) Often people do not realize that the ill-defined or apparently unsolvable problem they are working on is really a *sub-problem* and that (c) is relevant.

2.3 Freud/Poincaré

From Freud I got the idea of the unconscious mind: that there were things going on in one's brain that one didn't have direct access to. Poincaré, made it clear that much of his serious problem solving occurred in his subconscious, and I felt this was very common in problem solving of all kinds in the sciences and the arts.

This view was one important reason for my later rejection of "Expert Systems" as a significant step toward Artificial Intelligence. Expert Systems were (at best) expressions of peoples' *conscious* thought, which was, I felt, a very small fraction of human problem solving activity.

Other implications: Memory is what you invent to explain the things that you find in your head. Over the years, the "facts" in this paper will be gradually revised as I reread my research notes.

Explanations that people give for their own behavior are not to be taken too seriously—including discussions in this paper.

2.4 Bayes, Probability, and Human Learning

Probability theory would seem to offer an immediate solution to the NMTM problem. This turned out to be false. Probability theory tells how to derive new probability distributions from old probability distributions. It tells us how to make decisions from known or derived probability distributions and known utility functions. It does *not* tell how to get a probability distribution from data in the real world — which is what I wanted.

There is a definition of probability in terms of frequency that is sometimes usable. It tells us that a good estimate of the probability of an event is the frequency with which it has occurred in the past. This simple definition is fine in many situations, but breaks down when we need it most, i.e.: its precision decreases markedly as the number of events in the past (The "sample size") decreases. For sample sizes of 1 or 2 or none, the method is essentially useless.

Another common difficulty: It gives no suggestion as to how to deal with the "data fusion" problem : Suppose left-handed men have a probability (in the frequency sense) of .01 of dying at age 60. Black-haired men have a probability of .001 of dying at age 60. What is the probability of a left handed black haired man dying at age 60?

If, as is often the case in such situations, we have not collected data on black-haired, left-handed men — or if we only have two or three cases, then the frequency concept of probability tells us little or nothing.

Bayes' Theorem seemed like a very attractive approach to the general problem. If you had an a priori distribution over all possible universes, Bayes' Theorem would give an exact probability distribution for the continuation of any possible sequence of data. The difficulty was in obtaining the a priori probability distribution.

There had been some discussion of "personal probability" — in which each person developed an a priori probability distribution that was an appropriate summary of his life experience. This explained why different people made different predictions based on, apparently, the same data. Still, the origin of this personal a priori distribution was unclear — not good enough to use for real prediction.

My general conclusion was that Bayes' theorem was likely to be the key. That a person was born with a reasonably good built-in a priori probability distribution. The person would then make predictions and decisions based on this distribution. The distribution was then modified by their life experience. The initial "Built-in" distribution was obtained by organic evolution. There was a strong selection in favor of organisms that made decisions on the basis of "good" a priori probability distributions. The organisms making poor decisions would tend to have fewer descendants.

This is a Chomsky-like way of explaining where our personal probabilities come from. Still, it doesn't tell what that distribution is, and it doesn't tell enough about it to be useful in making probability estimates.

The biological origin of the a priori distribution suggests that one might learn much about it by studying living creatures –that this might be a very good organizing principal for the science of psychology.

The correspondences between probability evaluation and human learning are very close:

(1) Both involve prediction of the future based on data of the past.

(2) In both of them, prediction alone is of little value. The prediction must have an associated quantitative precision before it can be used to make decisions – as in statistical decision theory.

(3) In both cases the precision of prediction is critically dependent upon the quality and quantity of data in the past.

(4) In both cases, the precision of the prediction is critically dependent on the quality and quantity of the computational resources available. Human decisions

improve considerably if people have much time to organize data and try various theories in attempts to understand it.

That probability has to be **defined** in terms of the computational resources necessary to calculate it, is a relatively recent development [2].

2.5 Shannon

Shannon's papers (Sha 48) and subsequent developments in information theory have had a profound effect on my ideas about induction. Most important was the idea that information was something that could be quantified, and that the quantity of information was closely related to its probability. It suggested to me what I called at that time "The Information Packing Problem"— How much data could one pack into a fixed number of bits, or conversely how could one store a certain body of data using the least number of bits?

The idea was that the amount of data one could pack into a certain number of bits was related to the redundancy or information content of the data. Since information content was related to probability, inverting a solution to the information packing problem could give one probabilities.

Unfortunately, it was always possible to pack an arbitrary data string into 1 bit, using a suitable definition — a clearly inappropriate solution. I wasn't familiar enough with Universal Turing Machines to escape from that dilemma. Nevertheless — when I finally *did* discover Algorithmic Probability — the fact that it solved the "information packing problem" was one of the clues that lead me to believe it was correct.

2.6 Carnap

Carnap was one of the last of the philosophers of science called "Logical Positivists". He felt that most of the problems of philosophy could be solved through the analysis of language.

Although he was a professor in the philosophy department at the University of Chicago, most of his students were physicists or mathematicians. When I first met him, he was working on a general theory of probability — much as I was.

Some important ideas that I got from him:

That there were several definitions of the word "probability" — the best known was the frequency concept of probability, which he called P_1. However, there was another kind of probability: it was the degree of confidence one had in a hypothesis with respect to a certain body of data. He called this $P_2(H, D)$.

Carnap's model of probability started with a long sequence of symbols that was a description of the entire universe. Through his own formal linguistic analysis, he was able to assign

[2] See section 5.2, on Practical Induction, for further discussion of this point

a priori probabilities to any possible string of symbols that might represent the universe. He derived his $P_2(H, D)$ from this a priori distribution using Bayes' Theorem.

I liked his function that went directly from data to probability distribution without explicitly considering various theories or " explanations" of the data. It was a Korzybski-like way of avoiding the difficulties inherent in the verification of probabilistic theories. I also liked his idea of $P_2(H, D)$ and the idea of representing the universe by a digital string, but his method of computing the a priori probability distribution seemed unreasonable to me. The distribution depended very much on just what language was used to describe the universe. Furthermore, as one made the describing language larger and more complete, predictions were less and less contingent on the data. Carnap admitted these difficulties, but he felt that his theory nonetheless had redeeming qualities and that we would eventually find a way out of these difficulties.

Algorithmic Probability is close to Carnap's model, and it does overcome the difficulties described.

3 The Discovery of Algorithmic Probability

3.1 Huffman

Huffman coding (Huf 52) was an approximate solution to a special case of the "information packing problem". It was usually used if one had a finite number of symbol types of known frequencies. It was a good code if the data being coded was well approximated by a Bernoulli sequence, but was inappropriate if the data had more complex structure.

Nevertheless, when I did discover Algorithmic Probability, I realized that it was the inverse of Huffman's problem. He obtained a short code from knowledge of probabilities. I obtained probabilities from knowledge of short codes.

In the years before I had actually proved the correctness of Algorithmic Probability, it's relation to Huffman's work was strong evidence that I was on the right track.

3.2 Minsky and McCarthy

I first met Marvin Minsky and John McCarthy around 1952, soon after leaving the University. At that time, Minsky was mainly interested in human learning and problem solving. He wanted to design machines to simulate this aspect of human behavior and was beginning to get McCarthy interested. My own goals were more grandiose. I was interested in prediction and problem solving in general and persuaded Minsky that our machines would eventually go well beyond human capabilities.

Through our similarity of interest, Minsky and I soon became close friends. Although I lived in New York, I would often visit Boston, where Minsky lived. Though we were working on essentially the same problems, our backgrounds were different and we had much to teach each other.

From both Minsky and McCarthy, I learned to understand and appreciate Turing machines - both universal and non- universal. Up until that time, I had only a poor understanding of formal logic and the limits imposed by Gödel's Theorems. The translation of formal logic and recursive function theory into theorems about Turing machines was a real revelation for me. It gave me a quick intuitive grasp of many ideas that I had before found incomprehensible. It is not unusual for the translation of a problem into a new language to have this wonderful effect.

In 1956, McCarthy and Shannon organized the "Summer Study Group in Artificial Intelligence" at Dartmouth — a gathering of most of the world's researchers in A.I. and related fields. At that time, most of us had had a fair amount of experience with neural nets. Some notable exceptions: McCarthy, Newell and Simon. Shannon had done pioneering work on Boolean networks –which were close to neural nets.

One day McCarthy gave a talk on "Well defined mathematical problems". His thesis was that all mathematical problems could be formulated as problems of inverting Turing machines.

Specifically, given a machine M whose inputs were strings of symbols. Given a desired output string, s, we are required to find a string p such that $M(p) = s$.

McCarthy gave many examples to show how problems could be put into this form.

I asked him about the induction problem: "Suppose you are given a long sequence of symbols describing events in the real world. How can you extrapolate that sequence"?

Next day he said, "Suppose we were wandering about in an old house, and we suddenly opened a door to a room and in that room was a computer that was printing out your sequence. Eventually it came to the end of the sequence and was about to print the next symbol. Wouldn't you bet that it would be correct?"

There were several difficulties in implementing this idea as a prediction scheme:

First: What machine should be used?

Second: There may be a large number of inputs to the machine that give the same initial output but extrapolate differently. Which should we use? Third: Suppose the machine emits no symbols after the sequence to be extrapolated?

Though these difficulties all seemed quite serious, I remembered the idea because it seemed intuitively reasonable.

3.3 An Inductive Inference Machine

After the Dartmouth conference, I incorporated much of my thinking on prediction into the report and paper "An Inductive Inference Machine"(Sol 56, Sol 57).

How the system operates: It is initially shown a training set of a number of two dimensional patterns that represent correctly worked arithmetic problems, e.g.:

$$\begin{array}{cccc} = & 0 & 1 & 1 \\ & 0 & 1 & 1 \end{array} \quad , \quad \begin{array}{cccc} = & 1 & 0 & 1 \\ & 1 & 0 & 1 \end{array}$$

Then it is given a problem set of 2 dimensional patterns in which one or more of the positions has a question mark in it, e.g.:

$$\begin{array}{cccc} = & 1 & 0 & 0 \\ & 1 & ? & 0 \end{array} \quad , \quad \begin{array}{cccc} = & 0 & 0 & ? \\ & 0 & 0 & 1 \end{array}$$

The problem is to find what symbol the question mark represents.

To solve the problem, the system has a small number of primitive abstractions and transformations for combining and modifying abstractions. These primitive and transformed abstractions are used to create small two dimensional patterns that can be used for prediction.

Initially, it tries all of the primitive abstractions on the training set to see if any lead to correct predictions. If any do, and they are applicable to problems in the problem set, they are used for prediction.

If no suitable predictive abstractions are found in the primitive set, then members of the primitive set are transformed and combined by the set of transformations to produce new abstractions. These are tested on the training set and the successful ones are used for prediction on the problem set.

We continue generating more and more abstractions until we find at least one that works on the training set and is

applicable to the problem set.

After each round of training, various of the abstractions are given utility scores, depending on how successful they were in the prediction process.

In the next round of examples and problems, abstractions with high utilities are used preferentially in creating new trial abstractions.

We continue with a sequence of problems of increasing difficulty.

Four important aspects of the system:

(1) The nature of the problem sequence.

(2) The set of primitive abstractions and transformations.

(3) How the utility function is evaluated and how it governs the generation of new abstractions.

(4) The technique used for searching for new abstractions that are both consistent with the training examples and applicable to the problem examples.

I spent much time looking for an effective utility function without finding a really good solution.

Many years later, Algorithmic Probability proved to be the ideal tool for the design of utility functions (Sol 86, Sol 89) and Levin's search algorithm (Lev 73a, Sol 84) turned out to be the best way to search for good new abstractions.

3.4 The Discovery of Algorithmic Probability

On reading Chomsky's "Three Models for the Description of Language" (Cho 56), I found his rules for generating sentences to be very similar to the techniques I had been using in the 1957 paper to create new abstractions from old, but his grammars were organized better, easier to understand, and easier to generalize. It was immediately clear that his formal languages were ideal for induction. Furthermore, they would give a kind of induction that was considerably different from techniques used in statistics up to that time. The kinds of regularities it could recognize would be entirely new.

At the time of Chomsky's paper, I was trying to find a satisfactory utility evaluation function for my own system. I continued working on this with no great success until 1958, when I decided to look at Chomsky's paper more closely. It was easy for me to understand and build upon. In a short time, I devised a fast left to right parser for context free languages and an extremely fast matrix parser for context sensitive languages. It took advantage of special 32 bit parallel processing instructions that most computers have.

My main interest, however, was learning. I was trying to find an algorithm for the discovery of the "best" grammar for a given set of acceptable sentences. One of the things sought for: Given a set of positive cases of acceptable sentences and several grammars, any of which is able to generate all of the sentences — what goodness of fit criterion should be used? It is clear that the "Ad-hoc grammar", that lists all of the sentences in the corpus, fits perfectly. The "promiscuous grammar" that accepts any conceivable sentence, also fits perfectly. The first grammar has a long description, the second has a short description. It seemed that some grammar half way between these, was "correct" — but what criterion should be used?

There are other modes of learning in which the "goodness of fit" criterion is clearer.

One such learning environment involves a "teacher", who is able to tell the "learner" if a proposed sentence is within the language or not. Another training environment gives negative as well as positive examples of sentences.

Neither of these training environments are easy to obtain in the real world. The "positive cases only, with a few errors" environment is, by far, most widely available.

The real breakthrough came with my invention of probabilistic languages and their associated grammars. In a deterministic (non-probabilistic) language, a string is either an acceptable sentence or it is not an acceptable sentence. Taking a clue from Korzybski — we note that in the real world, we usually don't know for sure whether anything is true or false —but we can assign probabilities. Thus a probabilistic language assigns a probability value to every possible string. In a "normalized" language, the total probability of all strings is one.

It is easy to give examples of probabilistic grammars: any context free or context sensitive generative grammar can be written as a set of rewrite rules with two or more choices for each rewrite. If we assign probabilities to each of the choices, we have a probabilistic grammar.

The way probabilistic grammars define a solution to the "positive examples only" induction problem:

Each possible non-probabilistic grammar is assigned an a priori probability, by using a simple probabilistic grammar to generate non-probabilistic grammars.

Each non-probabilistic grammar that could have created the data set can be changed to a probabilistic grammar by giving it probabilities for each of its choices. For the particular data set of interest, we adjust these probabilities so the probability that the grammar will create that data set is maximum.

Given the data d_i ,the probability that a particular grammar G_j, created d_i is the a priori probability of G_j multiplied by the probability that d_i would be generated if we knew G_j to be the generating grammar (Bayes' Theorem). We then chose the grammar for which this product is largest.

The promiscuous grammar has high a priori probability, but assigns low probability to the data. The ad-hoc grammar has very low a priori probability, but assigns probability 1 to the data. These are two extreme grammar types: the best choice is usually somewhere between them.

For more detailed discussion see Hor 71, Sol 59 and Sol 64b, pp. 240-251.

Upon inventing a new kind of object, one wants to investigate its properties. One way to do this is to generalize the object. However, before generalizing *probabilistic* grammars, let us first generalize the more familiar *deterministic* grammars.

What is the most general deterministic grammar? A grammar can take the form of an *acceptance* rule or the form of a *generation* rule.

The most general deterministic acceptance rule: Suppose $M()$ is a universal Turing machine. If we feed it the finite string x, its output (when and if it ever stops) will be denoted by $M(x)$. Let a be a finite string that describes an acceptance grammar. Let s be a finite string that is a candidate sentence. Then s is an acceptable sentence in the language described by a if and only if $M(as)$ prints 1 and stops[3]. It is clear that there are languages and candidate strings such that the question of acceptance is undecidable. The generalization to probabilistic languages is immediate. If b is the description of the probabilistic language and s is a candidate string, then $M(bs)$ prints out a binary string that represents the probability that s is in the language described by b.

A useful restriction on M is that its output tape be unidirectional — once it

[3] Here as indicates the concatenation of strings a and s

has printed an output symbol it cannot erase it. In this case, even if the machine does not stop, we can sometimes obtain an approximation to the probability.

The most general deterministic *generative* grammar works similarly to the deterministic *acceptance* grammar. If q is the description of a generative grammar, and n is any positive integer, and $M(qn)$ prints out the string s and stops, then s is the nth sentence of the language.

The generalization to *probabilistic* generative grammars is also immediate. If h is the description of the probabilistic language and r is an infinite random string, then the probability that $M(hr)$ will print out a string s and stop is the probability with which s occurs in the language.

Since r is an infinite string, we will need to modify M to accommodate it. We do this by giving M three tapes: A unidirectional output tape, a unidirectional input tape, and an infinite bi-directional work tape. On the input tape, M first reads h, then it reads as many bits of r as it needs to generate the output string. The unidirectional output tape enables us to have at least partial knowledge of the output if the machine never stops.

How is the *probabilistic* language described by $M(hr)$ related to the *deterministic* language described by $M(qn)$?

In $M(qn)$, the integer n is used to determine what choices are to be made in string generation by the grammar described by q.

In $M(hr)$, the random number, r, is used to make the probabilistic choices in string generation by the grammar described by h.

The expression $M(hr)$ is extremely interesting. The argument hr consists of a random part preceded by a non-random description of a language. What would happen if the argument of M was purely random: $M(r)$? If L is the length of string h, in bits, then r has the probability of just 2^{-L} of starting with the string h and generating the language defined by h. The result is that the probability distribution implied by $M(r)$ is the same as the weighted average of all of the distributions, $M(h_i r)$.

Here h_i is the ith of the set of all descriptions of all grammars.

The weight associated with h_i is 2^{-L_i}, where L_i is the length of the description, h_i, in bits.

It is not unreasonable to assign 2^{-L_i} as the a priori probability of the grammar h_i. This would be the value given if we had a very simple probabilistic grammar generating the set of grammars, $[h_i]$.

If we do this, our weighted average amounts to the a priori probability distribution implied by all grammars.

This means that if we consider all possible descriptions of all possible grammars, then $M(r)$ is an a priori probability distribution on all finite strings— a truly amazing result, since there is no mention of grammars in the expression footnote More carefully defined versions of this apriori distribution were investigated by Levin (Lev 73b, 74), Gács (Gác 74), and Chaitin (Cha 75). Li and Vitányi call it the Universal Discrete Distribution (Li 93,section 4.3.4).!

Suppose we define the shortest description of s as the shortest string, d_s, such that $M(d_s) = s$. Then the probability assigned to s is approximately 2^{-L_s} since this is the probability that the first L_s bits of r will be d_s. Here L_s is the length of d_s in bits.

I then used the 2^{-L_s} idea to devise a priori probability distributions for sequential extrapolation (Sol 60a). Using the distribution, 2^{-L_s} as a first approximation, I obtained 2 different models for sequential prediction [4]:

The first was based on probabilities of finite strings.

The second model (which I will refer to as *Algorithmic Probability*) defined the a priori probability as the probability distribution on the output strings induced by a universal Turing machine with random input. It was similar to the $M(r)$ distribution for finite strings, but it allowed infinite output strings.

In the next two years, I refined the second model (which is very probably our best model for sequential prediction) and I added three more models. These 5 models were published in 1964 (Sol 64a,64b).

At the time, I felt it likely that all of the models would give about the same probability distributions.

I recently reviewed the 1964 paper with the benefit of 30 years of hindsight. Two of the five models described are certainly correct. While one of the models is, strictly speaking, meaningless, it has been of great practical and heuristic value. It is the closest approximation to Algorithmic Probability I know of that has actually been implemented in a prediction scheme (Ris 89).

For a more detailed discussion of the 1964 paper see the Appendix.

4 After the Discovery

At first, most of my evidence for the validity of Algorithmic Probability was very informal:

It corresponded to (and defined more exactly) the idea of Occam's razor — that "simple" hypotheses are more likely to be correct.

It was similar to Carnap's model of induction, but it seemed to overcome its deficiencies.

Both Huffman coding and the "Information Packing Problem" used probabilities to compress information. Algorithmic Probability inverted this process and obtained probabilities from compression.

That Algorithmic Probability was relatively independent of the choice of which universal machine was used seemed likely but was not altogether cer-

[4] These two models were first described in a talk given at the Conference on "Cerebral Systems and Computers" at the California Institute of Technology Feb 8–11,1960 — then in a Zator Co. report (Sol 60a),and again in a more widely circulated AFOSR report (Sol 60b).

Minsky briefly described these ideas in 1961 (Min 61),then in more detail in 1962, including a discussion of "the invariance theorem"(Min 62). The 1961 paper was then reprinted in "Computers and Thought"(Fei 63),a very widely read book that first introduced the world to Artificial Intelligence.

tain. It was clear that probabilities assigned by different machines would differ by only a constant factor (independent of length of data described), but this constant factor could be *quite large*. Changing reference machines corresponds to changing from one computer language to another. Although BASIC and FORTRAN are almost identical, translating from one to the other requires a program of certainly more than 1000 bytes. This corresponds to a "constant factor" of greater than 10^{300} — a truly enormous number.

Fortunately, in just about all applications of probability, we are not interested in absolute probabilities, but only in probability *ratios*. I had some heuristic arguments to the effect that the probability ratio for alternative possible continuations of a sequence should be relatively machine independent if the sequence were very long.

The second half of the 1964 paper was devoted to examples of how this probability evaluation method could be applied. That it seemed to give reasonable answers was some of the strongest evidence that I had for its correctness.

There was one property that I felt the general induction system should possess: for long sequences the general system should give at least as good results as any other induction system. There was a heuristic argument to show this would be so.

In 1968 I devised a simpler minimal criterion for correctness of an induction system: Suppose we have an infinite string of symbols that was generated by a probabilistic generator that had a finite description. Then for a sufficiently long sequence of data, the method should give predictions with probability values very close to those given by the generator.

Later that year an event occurred that enabled me to prove that Algorithmic Probability satisfied this criterion.

4.1 Willis

In 1968 I was asked to review "Computational Complexity and Probability Constructions", a paper by David Willis. It was the first substantive response I'd seen to my 1964 paper giving the four models. I found his paper to be excellent. Willis avoided the "halting problem" by defining computationally limited Turing machines that had no halting problems. From these machines, he was able to define probabilities in an exact manner. By raising the computational limits on his machines, he approached the behavior of universal machines.

In addition to its value as a rigorous treatment of a subject that I had treated relatively informally, I was able to use Willis' results to prove that these induction methods satisfied my "correctness" criterion. The methods converged surprisingly rapidly to the correct probability values.

Unfortunately it took me about 6 months to read Willis' paper with sufficient care. By that time, the other two reviewers and the journal editor had rejected it. One of the reviewers felt that it didn't add much to what I had said in my 1964 paper.

I wrote Willis, telling him what a great paper it was and urged him to send it to a different journal. It was finally published in JACM in 1970 (Wil 70).

It was not until 1975 that I first published the theorem on the convergence of Algorithmic Probability to the correct values (Sol 75a, Sol 75b) and not until 1978 that the proof itself was published (Sol 78).

Meanwhile, Cover (Cov 74; Sol 78, p.425) had shown that if one used "Extension Probability" as the basis of a gambling scheme, the yield would be about the maximum obtainable. The proof was applicable to all complexity based probability measures.

These two demonstrations suggested very strongly that complexity based induction would be a very good basis for practical prediction.

4.2 Levin

The first paper I read by Levin was one he had written with Zvonkin in 1970 (Zvo 70) reviewing work in the Soviet Union on Algorithmic Probability and complexity that had been inspired by Kolmogorov, Levin's thesis advisor. Since then I had gotten the impression that Levin was in some sort of political difficulties and I wondered whether I could engineer some world academic pressure to get him out of prison (if it gotten to that point!).

In 1978 I was much relieved to receive a phone call from Levin: he was safe and sound at MIT with lots of amazing stories to tell about his adventures. He had, indeed, indulged in "political incorrectness", and it was only through the influence of Kolmogorov that he was able to get out of the country unscathed!

Soon after I met him, he told me that he had a near optimum solution to the general inversion problem (Lev 73a).

Inversion problems are the P and NP problems of computational complexity theory: i.e. Given a machine, M, that maps finite strings onto finite strings. Given the finite string, x. How can we find in minimal time, a string, p, such that $M(p) = x$?

Suppose there exists an algorithm, A, that can examine M and x, then print out p within time T. Levin had a search procedure that, without knowing A, could do the same thing that A did, but in no more time than $CT2^L$. Here, L is the length of the shortest description of A, using a suitable reference machine, and C is a measure of how much slower the reference machine is than a machine that implements A directly. An alternative form of this cost of the search is CT/P. Here $P = 2^{-L}$ is approximately the a priori probability of the algorithm, A.

The parameter T/P plays a critical role in searches of all kinds. In designing sequences of problems to train an induction machine the T/P value of a particular problem at a particular point in the training of the machine gives an upper bound on how long it will take the machine to solve the problem. In analogy with human problem solving, I christened T/P "Conceptual Jump Size".

Before I met Levin, I had been using T/P as a rough estimate of the cost of a search, but I had no proof that it was achievable.

Sometime later, I showed that Levin's search procedure (which I will henceforth denote by "Lsearch") is very close to optimum if the search is a "Blind Search" [5]. A blind search is one in which the searcher has only the probabilities of each of the things to try, and is unable to learn anything (i.e. modify the probabilities) from any trial.

In Artificial Intelligence research,"Blind search" is usually impractical, since the size of the search space increases exponentially with the problem size. This is called "The exponential explosion".

Heuristic search deals with this explosion by sharply reducing the size of the search space. This process of cutting out parts of the space is implemented through domain-specific knowledge of the search space, or by more general methods that work in many domains.

The probabilistic analog of a heuristic is a function that assigns low probabilities to areas of the search space that would be cut out by the heuristic. In machine learning theory this is called "bias".

While many problems in science and mathematics can be formulated as inversion problems, there is another large set of problems that cannot. These are the time (or resource) limited optimization problems.

Suppose we have a machine, M, whose inputs are finite strings, and whose outputs are numbers. We are given a fixed time T. The problem is to find within time T an input string, s, such that $M(s)$ is as large as possible.

If no time limit is given, the problem can be generalized so that after a certain minimum time, one should always have the latest best solution to the problem available. If asked for a solution at a time T (previously unknown to the problem solver), the solution presented should be not much different from the one obtainable if the limit T were known in advance footnote Dean, Thomas and Boddy (Dea 88) have coined the term "anytime algorithm" for a solution to this type of optimization problem. Sigart Bulletin of the ACM Vol 7, No.2 April 1996, has references and recent work in this area..

An example of an optimization problem of the first kind: design an automobile in 6 months having certain specifications and having minimum cost. Many problems in science and engineering are of this type.

Sometime later, Levin and I independently generalized Lsearch to include time limited optimization problems.

[5] Though Lsearch has been widely described (Lev 73; Sol 84,86,89; Li 93 pp. 410-413) there has been little application of it to real problem solving. Paul and Solomonoff (Pau 94) discuss its application to several problems and calculate T/P (Conceptual Jump Size) for solutions, but Schmidhuber (Schm 94) was perhaps the first to actually run a computer program that used Lsearch to solve problems. While it only solved simple problems in neural net design the technique used is very general and of much interest. The probabilistic version of Lsearch used in the program had a serious error in it, but it has been replaced with a more conventional non-probabilistic Lsearch that seems to work fine.

In Lsearch, I had what seemed to be a powerful approach to problem solving. The most general kind of induction can be formulated as the problem of finding short descriptions of data – - which is a time limited optimization problem, and therefore amenable to Lsearch.

4.3 Incremental Learning

The next step was to embed these problem solving techniques into a system for machine learning.

Since Lsearch wasn't much good without the probabilistic equivalent of heuristics, it would be necessary to have the machine modify the probability distribution over the search space as a result of its own experience in problem solving.

As originally envisioned, the new system was very similar to my old "Inductive Inference Machine" of 1956. The machine starts out, like a human infant, with a set of primitive concepts [6].

We then give it a simple problem, which it solves using these primitive concepts. In solving the first problem, it acquires new concepts that help it solve the more difficult second problem, and so on. A suitable training sequence of problems of increasing difficulty brings the machine to a high level of problem solving skill.

The system overcomes many limitations of other learning systems. Most of them are limited in the types of concepts they can discover, even with infinite search time. This can be because the system has an "incomplete" set of concepts (not "universal" in the sense of "Universal Turing Machine"), and/or because the search algorithm is inadequate. We give our machine a complete set of concepts early in its training. The use of Lsearch then guarantees that any describable concept will eventually be discovered by the system.

The scope of problems the system can handle are inversion problems and time limited optimization problems. These cover a very large fraction of the problems encountered in science and engineering.

There are two aspects of the efficiency of any system for machine learning . First — What is the computational complexity of its solutions to problems? How much time and memory are required? Second — What is the informational efficiency? How much training is needed for the system to learn? How many problems and/or examples are required?

In the particular training environment I had in mind, I felt that Lsearch would be extremely efficient in both ways — I had heuristic arguments suggesting that it was within a factor of 4 of being optimum.

[6] The term "concept" in machine learning theory has a special meaning – the generalization of the set of positive training instances that we want the machine to learn. Here, it is being used in a more colloquial manner – I mean it to be any kind of intellectual abstraction. In the present context, any "concept" can be represented by string of computer instructions – a "macro". They are combined by concatenation.

An additional attractive feature of the system: the improvement of the general operational efficiency of the system can be formulated as a time limited optimization problem — so after the system has had enough experience in problems of improving programs, we have it spend half of its time on self-improvement.

To get some feedback on the effectiveness of these ideas I needed a suitable training sequence of problems of increasing difficulty. Designing such a sequence proved to be very difficult. I was eventually able to design three training sequences in varying degrees of completeness.

The first learned the rules of arithmetic after being shown examples of correctly worked problems. This was done in some detail and T/P values (Conceptual Jump Size) were computed for each stage of the learning (Pau 94).

The second induced the laws of algebra after being given a sequence of simple linear equations to solve.

The third learned to solve general linear equations, then general quadratics, then general cubic equations.

Though the system seemed to work (on paper, at least) for these training sequences, I felt that as a whole, they didn't do what I wanted them to do. I didn't see how I could fit them together to solve more difficult problems — and ultimately, to solve problems they were not designed to solve. Another thing that disturbed me was that the system never used *parts* of the solutions to previous problems to help solve new problems. It only used the completed solutions of previous problems. This deficiency was not in the system itself, but only in the inadequacy of the training sequences.

One promising approach generalizes the idea of training sequences. Most training in the real world does not consist of problems alone. Wholly or partially solved problems, statements of fact and hints of various kinds are only a few of the components of common training environments. It would seem to be much easier to implement learning with this larger universe of techniques. I have examined some training environments of other systems for machine learning.

Genetic Algorithms have been a rich source of ideas on possible training environments. These algorithms use either a complete set of concepts or can be easily modified to do so. They work by combining concepts that have been useful in the past to generate promising new trial solutions, much as my own system does.

Consider a Genetic Algorithm System designed to find values of s that maximize the function $M(s)$. As before, M is a machine with strings, s as input and real numbers between zero and ten as output. We start out with $[s_i]_0, i = 1, 2, ..., 100$, a population of 100 random strings, the zeroth generation. The mean value of $M(s_i)$ over that population is, say, 2. We make the following training sequence, for the sets, $[s_i]_j$. The first problem is to get the average value of $M(s_i)$ to be 3. The second problem is to get an average of 4. The jth problem is to get an average of $j + 1$ etc., up to $j = 7$ or $j = 8$. Each problem builds on the information obtained in the solution to the previous problem.

It is easy to train infant systems in this way, using a sequence of inversion problems derived from an optimization problem —but mature systems with more experience do not solve optimization problems this way. They use Lsearch for a special direct solution technique that is close to optimum (Sol 84).

The training sequences that I designed for my own system were constructed very carefully. At each step in the training sequence, I knew an upper bound on how long the Lsearch would take, because I knew at least one solution and at least one code for that solution. In using training sequences such as the one described for Genetic Algorithms, one usually has no a priori idea as to how long it will take to solve the problem, and often one doesn't know if it is at all solvable by the system.

Nevertheless, I did try a training sequence for a problem solved by one of J. Koza's Genetic programs (Koz 90, pp. 16-24). It was the problem of learning an 11 input Boolean multiplexer function. The general k address bit multiplexer has $k + 2^k$ inputs and 1 output. k of the inputs designate an address of one of the 2^k other inputs. The output is the same as the input at the designated address.

The problem is to construct a Boolean function that simulates the multiplexer for all 2^{11} of its input configurations.

Using genetic programming, Koza started with the Boolean functions AND, OR, NOT,and IF [7] and obtained a solution in 9 generations using more than seventy million trials. Using Lsearch, and only the IF function, I obtained a solution in 3 generations using about ten million trials. In general, a very simple Lsearch obtains a solution for the k address bit multiplexer in just k generations. It is unclear, however, as to how much longer it would take to do an Lsearch over the space containing all 4 Boolean functions.

While Koza's solutions had many extraneous functions in them, the Lsearch solutions were all of minimal complexity. This makes it much easier to understand the solutions. It also makes it possible to search for common subtrees in the trees that represent more successful functions. These subtrees could then be used in constructing new trial functions. I have not yet tried this refinement, however.

It should be noted that these results, while suggestive, are only theoretical pen and paper calculations and not directly comparable with Koza's computer simulations.

Artificial Life and Organic Evolution are also sources of useful training environments. Here we have competition between species. As each species improves, it presents a somewhat larger challenge to its competitor. As the competing species evolve together, they present training sequences for each other.

D. Hillis (Hil 90) has used a simulation of this kind of competition to evolve a solution to a fairly difficult problem in computer design.

My present research continues to be the exploration of training environments for incremental learning systems. I'm also trying to get a better understanding

[7] IF is a 3 input, 1 output Boolean function corresponding to a multiplexer with $k = 1$

of how probabilistic heuristics are discovered and applied to Lsearch.

5 Relation of this Work to That of Others

5.1 Theoretical Analysis

The five years following the 1964 publications was a time of much activity in this field, but except for the work of Willis and Levin, little of it seems to have been inspired by my own work.

Kolmogorov (Kol 65, 68a, 68b, 69) was interested in randomness and complexity of one string with respect to another, as well as the development of information theory and probability based on lengths of codes. Strangely enough, he did not appear to be interested in inductive inference. [8]

He defined the Algorithmic Complexity of a string to be the length of the shortest code needed to describe it. A random string was one whose complexity was about as large as its length. These definitions motivated a group of brilliant associates to explore their properties.

Martin-Löf (Mar 66) developed a definition of randomness somewhat different from Kolmogorov's, using "randomness tests".

Kolmogorov had read my 1964 paper by the time that Levin joined his group. From Kolmogorov's description of this paper, Levin put my a priori probability into a more exact form. His work in this area was similar to that of Willis, but while Willis avoided the incomputability problems of universal machines by using finite approximations to them, Levin faced these incomputability problems directly. He defined "semi-computability"; a kind of weakened computability that enabled him to analyze the behavior of many otherwise intractable functions. While Willis' work seemed closer to practical realization, Levin's was a model of mathematical elegance.

Apparently independently of my own work and that of Kolmogorov, Chaitin published two papers (Cha 66, 69) defining randomness in terms of program length. In the first of these papers he informally suggested that the shortness of a program that describes a sequence might be an index as to how good a theory that program represents. He did not, however, investigate this idea at any length.

In the period from 1970 to 1975, Levin (Lev 73b, 74) , Gács (Gác 74) and Chaitin (Cha 75) defined universal a priori probability distributions on finite strings. These were closely related to the probability distribution on all finite strings induced by $M(r)$ (section 3.4) in my original generalization of probabilistic languages.

Cover (Cov 74, Sol 78, p.428) used Chaitin's distribution on finite strings to define "Extension Complexity". This in turn was used to define a probability

[8] Levin (Lev 95) attributes this to induction being an ill-defined mathematical problem. — I do not, however, find this convincing.

measure, which was similar to the fourth model I had described in 1964 (Sol 64a) [9].

In an attempt to devise a better definition of randomness than is afforded by minimal description length, Schnorr (Schn 73) defined what he called "process complexity". This turned out to be the negative logarithm of what I had defined to be Algorithmic Probability.

For a history of discoveries in this field, as well as detailed treatment of the discoveries themselves, the book of Li and Vitányi (Li 93) is unexcelled.

5.2 Practical Induction

5.2.1 Optimal Approximations

The convergence theorem (Sol 78, Theorem 3) makes Algorithmic Probability look very attractive as a means of induction. It is the only induction system we know of that is "complete". By this we mean that if there is any describable regularity in a body of data, Algorithmic Probability is guaranteed to discover it using a relatively small sample of the data. It is the only probability evaluation method known to be complete. As a necessary consequence of its completeness, this kind of probability must be incomputable. Conversely, any computable probability measure cannot be complete.

We are using the term "incomputable" in a special way. Algorithmic Probability is as "computable" as the value of π – but with one important difference; when we make successive approximations to the value of π, we know how large the error in each approximation can be. In the case of Algorithmic Probability, we have a procedure for successive approximations that is guaranteed to converge to the right value. However, at no point in the calculation can we make a useful estimate of the error in the current approximation.

This might be regarded as a serious criticism of the use of Algorithmic Probability or approximations to it, to solve practical problems, but it is not. It is a difficulty shared by all probability evaluation methods. If they are complete, then they are incomputable. If they are computable, (either as independent probability measures or as approximations to Algorithmic Probability) then they *must* be incomplete. This incompleteness implies that there have to be regularities that are invisible to them. When used with data having regularities of these kinds, computable methods will have errors of unknown size. It is likely that all quantitative methods of dealing with uncertainty have this uncertainty of error size.

It has only been through the analysis of Algorithmic Probability that these very general limitations of knowledge of error size have become known. I will

[9] Recognizing that models are identical is often non-trivial. I read Cover's 1974 paper very carefully and wrote an extensive analysis of it in 1978 (Sol 78); but it was only recently, on rereading my 1964 paper, that I realized that one of its models was close to Cover's "Extension Complexity". See Appendix for further discussion.

show, however, that in spite of it's incomputability, Algorithmic Probability can serve as a kind of "Gold Standard" for induction systems – that while it is never possible to tell how close a particular computable measure is to this standard, it is often possible to know how much closer one computable measure is to the standard than another computable measure is. I believe that this "partial ordering" may be as close as we can ever get to a standard for practical induction. I will outline a general procedure that tells us how to spend our time most efficiently in finding computable measures that are as close as possible to this standard. This is the very best that we can ever hope to do.

To better understand the foregoing, let us consider the following definition of Algorithmic Probability:

$$P(x) = \sum 2^{-l_i} \tag{1}$$

$P(x)$ is the Algorithmic Probability of finite string x.
l_i is the length of the i^{th} description of string x.
The sum is over all such descriptions. (See Sol 78, p. 423 for more details).

That the sum is incomputable, is associated with the fact that it is often impossible to verify in finite time, whether a particular string is a description of x or not.

Over the years there has been a general impression in the scientific community that this incomputability would make it impossible to use Algorithmic Probability as a tool for statistical prediction (see for example, Ris 95 p.197).

From the beginning, however, this difficulty was recognized and methods for dealing with it were proposed (Sol 64a section 3.1.2.1). Willis (Wil 70) formalized one of these methods in what we will call "Resource Bounded Algorithmic Probability".

The most efficient way to implement Resource Bounded Algorithmic Probability is to approximate equation 1 by the largest lower bound on $P(x)$ that can be demonstrated in time, T. This is usually done by finding many short codes for x that give terms summing to that bound. This kind of approximation to Algorithmic Probability is an example of a "Time limited optimization problem" and is directly solvable by Lsearch [10].

By getting as large a value of the sum as we can, we get as close as possible to Algorithmic Probability in the allotted time. This seems like the best possible way to spend our time computing probabilities, since Algorithmic Probability is at present the best theoretical induction system we have.

Resource Bounded Algorithmic Probability is then an ideal way to *define* "Practical Probability". It captures a vital feature of any possible practical application of probability: While the value of Algorithmic Probability is clearly defined in a mathematical sense, it is incomputable in any practical sense. Any

[10] Although the technique has been described in detail (Sol 84), I know of no attempt to solve a real time limited optimization problem using Lsearch.

value we obtain as an approximation will depend critically on just what computational resources we used to get it. This leads to a definition of Practical Probability that is a function of three arguments:

1. The data.
2. The a priori information — which is uniquely characterized by our choice of universal reference machine.
3. The resources available for computation – time and memory.

Any failure to specify each of these arguments exactly can lead to gross ambiguities in the value of the probability.

5.2.2 Suboptimal Approximations

There have been several *suboptimal* approximations to Algorithmic Probability. One of the earliest is that of Van Heerden (Van 63). [11] He considered the prediction of binary sequences by Boolean functions. His criterion for the best function to use: Add the length of the description of each function to the number of errors it made in prediction. Least is best.

We may regard this technique as approximating the sum in equation 1 by the single largest term we can find.

Before I read his report,I had considered this kind of minimum description length as an approximation to the universal a priori distribution (Sol 60a equation 1). The particular form that I used was very general but it always assigned probabilities that were integral powers of two. While this method clearly did not give correct probabilities, I was at that time uncertain as to how large the error was. In my research notes I subsequently called this prediction technique "The VH method".

There are theoretical reasons for believing that the error may be small if the shortest code one has found thus far were indeed the shortest code for the data. However, for induction problems in which one is uncertain as to the class of stochastic functions that generated the data (as in psychology, economics, geology), one cannot know one has the shortest code and the amount of accuracy lost must remain unknown.

Wallace and Boulton (Wal 68) used what they called MML (Minimum Message Length) to obtain a continuum of probability values. They considered various functional forms to assign probabilities to a sequence of data symbols. The function selected was the one for which the length of description of the function minus the logarithm of the probability assigned to the data by the function, was minimal. This technique is the same as including in equation 1 certain terms in addition to that corresponding to the shortest code found. Since the sum is larger than Van Heerden's, it is a uniformly better approximation to Algorithmic Probability.

[11] Peter Van Heerden is best known for his discovery of a method to store information optically in a three dimensional crystal.

Rissanen started out (Ris 78) with a technique similar to MML which he called Minimum Description Length (MDL) but he slowly modified it over the years. The latest version, called "Stochastic Complexity" (Ris 89) is not far from the negative logarithm of Algorithmic Probability.

Rissanen avoids the incomputability problem by not using all possible functions. He considers limited classes of predictive primitive recursive functions, then takes a weighted sum of their predictions. The weights are assigned to the functions using classical ideas on a priori probability distributions, modified by considerations of code length. Since he sums over more terms of equation 1 than Wallace and Boulton do, he gets a better approximation to Algorithmic Probability.

We can, however, get even closer to Algorithmic Probability than Stochastic Probability does. Rissanen's "primitive recursive functions" are the functions that are commonly used in the sciences. They are well behaved and the values of their arguments for which the functions are defined are known or computable.

A larger, less well behaved class of functions is the set of partial recursive functions. They are definer for certain values of their arguments, but not for others. However, if a function is *not* defined for a certain argument, it is often impossible to be certain of this fact. These arguments correspond to certain intractable descriptions of x in equation 1.

We can deal very nicely with functions of this kind, using Resource Limited Algorithmic Probability. The Lsearch algorithm tells just how much time to spend trying to verify a potential code that doesn't seem to be defined.

How often do partial recursive functions occur in probabilistic calculations involving real world events? It appears that there are many areas of science in which functional forms are either partial recursive or *practically* partial recursive - i.e. the time needed to compute the functions for certain of their arguments is greater than what we have available, and we cannot tell in advance for which values of the arguments this is true. Long branching causal chains giving rise to functions of this sort commonly occur in geology, biology, sociology and economics.

In predicting earthquakes or the motion of financial markets, we cannot afford the limitations of suboptimal approximations. We need the full power of Resource Limited Algorithmic Probability.

A Appendix

A.1 Description of the Five Models

The first 1964 paper (Sol 64a) described 5 models for induction. Four of these models were based on two kinds of Universal Turing Machines.

The first kind of Machine, M_1, was a very general Universal Turing Machine, with no constraints on its input or output.

The second kind of machine, M_2, was a special 3 tape machine: It had unidirectional input and output tapes, as well as a bidirectional work tape. The unidirectional output meant that once an output symbol was written, it could not be erased.

In M_1, once a symbol was written as output, the machine could erase it and write another symbol if the program told it to do so. M_2 also had what I called in my research notes "The Sequential Property": if $M_2(x) = a$ (x and a being finite strings), then $M_2(xy) = ab$ (Sol 64a, p.16). This means that if we concatenate anything onto the end of the code for a, then the machine's output string must start with the string a [12].

The first model for induction was based on M_1. An a priori probability was assigned to a sequence on the basis of a weighted sum of all possible programs for that sequence with all possible finite continuations of it. The weight assigned to a program of length N was 2^{-N}.

The second model (which I now call *Algorithmic Probability*) was based on M_2. Random bits were fed into the machine as input. The probability assigned to a particular string, s, was then the probability that the machine would have as output a string that had s as prefix — i.e. its output would be s, possibly followed by a finite or infinite number of symbols.

The third model used Machine M_1. To obtain the a priori probability of the string, s, we first select a large number, N. Then $C(s, N)$ is the number of input strings of length N, that cause the machine to print an output that has s as prefix, before it stops. $C(N)$ is the number of inputs of length N that cause the machine to eventually stop, regardless of what its output is. Then the a priori probability assigned to string s is the limit, as N approaches infinity, of $C(s, N)/C(N)$.

The fourth model was the same as the third, except that M_2 was used instead of M_1.

The fifth model makes probability evaluations by using a weighted mean of the evaluations given by all possible probability evaluation methods. The weight given to any particular evaluation method is the product of two factors. The first factor is the probability assigned to the known data by that probability evaluation method. The second factor is the a priori probability of that method. If the smallest number of bits needed for a program to generate a particular evaluation method is N, then this factor is approximately 2^{-N} for that method.

A.2 Evaluation of the Five Models

The second model is certainly correct. The description of the machine, M_2, with unidirectional input and output is correct in all details. The final expression

[12] A partial recursive function having the "sequential property" was later called a "process" by Levin (Zvo 70, definition 3.1) and by Schnorr (Sch 73, p.378). Li and Vitányi (Li 93, P. 238, Definition 4.13)called it a "monotonic machine."

(ibid, eq.(7)), gives the conditional probability that the sequence T will be followed by sequence a.

The conditional probability given in this equation is that of a *semi-measure* rather than a normalized *measure* (Li 93, p. 215) . In general, it is smaller than a normalized conditional probability. However, Peter Gács has shown (Li 93, Theorem 5.1, p. 285) that the error in this kind of conditional probability converges rapidly to zero — just as it does for a normalized conditional probability measure (Sol 76, Theorem 3, p. 426).

The fourth model, also based on M_2, is also correct. It certainly converges, and because it only considers finite continuations of the known sequence, it is similar to Cover's "Extension Probability" (Cov 74). While it is better than Extension Probability in both betting yield and size of error in probability estimate, I do not know whether it is *significantly* better.

The first and third models are based on M_1, the unrestricted Universal Turing Machine. The 1964 paper does not describe the machine very exactly. As a result, while it is possible to define its operation so that the expressions for probability given by both models converge, it is also possible to define the operations in ways such that I have been unable to tell whether the expressions will converge or not. This puts models one and three in a kind of limbo.

The fifth model considers "all possible probability methods." Unfortunately, it is not possible to effectively enumerate all such methods, so the recipe, if taken literally, is meaningless. On the other hand, in practical prediction it is often quite possible to take a weighted sum of a large number of methods that *are* effectively enumerable. At the present time, the best approximations to Algorithmic Probability that have been programmed, have taken this form (Ris 89).

Acknowledgement

The treatment of the effects of partial recursive functions on Algorithmic Probability was inspired by a long discussion with Arun Sharma.

1948	Shannon				
1949					
1950	Carnap				
1951					
1952	Huffman				
1953					
1954					
1955					
1956	Chomsky				
1957					
1958					
1959					
1960	Solomonoff				
1961	Minsky				
1962	Minsky				
1963	Minsky			Van Heerden	63
1964	Solomonoff				
1965			Kolmogorov		65
1966		Chaitin	Martin Löf		66
1967					
1968	Willis		Kolmogorov	Wallace, Boulton	68
1969		Chaitin	Kolmogorov		69
1970	Willis		Zvonkin, Levin		70
1971					
1972					
1973					
1974	Cover				
1975	Solomonoff	Chaitin			75
1976					
1977					
1978	Solomonoff			Rissanen	78

Timeline of Papers Referenced

The times are all publication dates, except for "Willis 1968," which gives the year that I first read the paper published in 1970.

References

[1] (Bri 27) Bridgman, P.W, *The Logic of Modern Physics*, The Macmillan Company, New York, 1927.

[2] (Cha 66) Chaitin, G.W, "On the Length of Programs for Computing Finite Binary Sequences," *Journal of the Assoc. of Comp. Mach.*, 13, pp. 547–569, 1966.

[3] (Cha 69) Chaitin, G.W, "On the Length of Programs for Computing Finite Binary Sequences: Statistical Considerations," *Journal of the Assoc. of Comp. Mach.*, 16, pp. 145–159, 1969.

[4] (Cha 75) Chaitin, G.W, "A Theory of Program Size Formally Identical to Information Theory," *Journal of the Assoc. of Comp. Mach.*, vol. 22, no. 3, pp. 329–340, July, 1975.

[5] (Cho 56) Chomsky, A.N. "Three Models for the Description of Language," *IRE Transactions on Information Theory*, Vol. IT–2, No. 3, Sept. 1956, pp. 113–124.

[6] (Cov 74) Cover, T.M. "Universal Gambling Schemes and the Complexity Measures of Kolmogorov and Chaitin," Rep. 12, Statistics Dept., Stanford Univ., Stanford, Ca, 1974.

[7] (Dea 88) Dean, Thomas and Boddy, "An Analysis of Time-Dependent Planning," *Proceedings of the Seventh National Conference on Artificial Intelligence*, pp 49-54, Minneapolis Minnesota, 1988.

[8] (Fei 63) Feigenbaum, E.A. and Feldman, J. *Computers and Thought*, McGraw-Hill Book Co., New York, 1963.

[9] (Gác 74) Gács. P. "On the Symmetry of Algorithmic Information," *Soviet Math. Dokl.*, 15:1477–1480, 1974. Correction, Ibid, 15:1480, 1974.

[10] (Hay 41) Hayakawa, S.I. *Language in Action*, Harcourt, Brace and Co., New York, 1941.

[11] (Hil 92) Hillis, D. "Co-evolving Parasites Improves Simulated Evolution as an Optimization Procedure," *Artificial Life* II, Langdon, C., Taylor, C., Farmer, J., and Rasmussen, S. (Eds), pp 313–324.

[12] (Hor 71) Horning, J. "A Procedure for Grammatical Inference," *Proceedings of the IFIP Congress 71*, Amsterdam, North Holland, pp. 519–523, 1971.

[13] (Huf 52) Huffman, D.A. "A Method for Construction of Minimum-Redundancy Codes," *Proceedings IRE*, 40:1098–1101, 1952.

[14] (Joh 46) Johnson, W. *People in Quandaries; the Semantics of Personal Adjustment*, Harper and Row, New York, 1946.

[15] (Kol 65) Kolmogorov, A.N. "Three Approaches to the Quantitative Definition of Information." *Problems Inform. Transmission*, 1(1): 1–7, 1965

[16] (Kol 68a) Kolmogorov, A.N. "Logical Basis for Information Theory and Probability Theory." *IEEE Trans. on Information Theory*, IT–14(5):662–664, 1968.

[17] (Kol 68b) Kolmogorov, A.N. "Some Theorems on Algorithmic Entropy and the Algorithmic Quantity of Information." *Uspekhi Mat. Nauk*, 23(2):201, 1968.

[18] (Kol 69) Kolmogorov, A.N. "On the Logical Foundations of Information Theory and Probability Theory." *Problems Inform. Transmission*, 5:1–4, 1969.

[19] (Kor 58) Korzybski, A. *Science and Sanity*, Internat. Non-Aristotelian Library Pub. Co., Lakeville, Conn., 1958.

[20] (Koz 90) Koza, J. "Genetic Programming: A Paradigm for Genetically Breeding Populations of Computer Programs to Solve Problems," Rep. No. STAN-CS-90-1314, Dept. of Computer Science, Stanford Univ., Stanford, Ca, 1990, pp. 16–24.

[21] (Lev 73a) Levin, L.A. "Universal Search Problems," *Problemy Peredaci Informacii 9*, pp. 115–116, 1973. Translated in *Problems of Information Transmission 9*, pp. 265–266.

[22] (Lev 73b) Levin, L.A. "On the Notion of a Random Sequence," *Soviet Math. Dokl*, 14:1413–1416, 1973.

[23] (Lev 74) Levin, L.A. "Laws of Information Conservation (Non-Growth) and Aspects of the Foundation of Probability Theory," *Problems of Information Transmission*, 10:206–210. 1974.

[24] (Lev 95) Levin, L.A. Personal communication.

[25] (Li 93) Li, M. and Vitányi, P. *An Introduction to Kolmogorov Complexity and Its Applications*, Springer-Verlag, N.Y., 1993.

[26] (Mar 66) Martin-Löf, P. "The Definition of Random Sequences," *Information and Control*, 9:602–619, 1966.

[27] (Min 61) Minsky, M.L.. "Steps Toward Artificial Intelligence," *Proceedings of the Institute of Radio Engineers*, 49:8–30, 1961.

[28] (Min 62) Minsky, M.L.. "Problems of Formulation for Artificial Intelligence," *Mathematical Problems in the Biological Sciences*, Proceedings of Symposia in Applied Mathematics XIV, R.E. Bellman (ed.), pp. 35–46, American Mathematical Society, R.I., 1962.

[29] (Pau 94) Paul, Wolfgang J., and Solomonoff, R. "Autonomous Theory Building Systems," Computer Science Department, University of Sarbrücken, Germany. To appear in *Annals of Operations Research* ,1994

[30] (Rap 53) Rapoport, A. *Operational Philosophy*, New York, J. Wiley, 1965.

[31] (Ris 78) Rissanen, J. "Modeling by the Shortest Data Description," *Automatica*, 14:465–471, 1978.

[32] (Ris 89) Rissanen, J. *Stochastical Complexity and Statistical Inquiry*, World Scientific Publishing Company, 1989.

[33] (Ris 95) Rissanen, J. "Stochastic Complexity in Learning," in Vitányi, Paul (ed.), *Computational Learning Theory (Second European Conference, EuroCOLT '95)*, Springer-Verlag, Berlin, 1995.

[34] (Sha 48) Shannon, C.E. "The Mathematical Theory of Communication," *Bell System Technical Journal*, 27:379–423, 623–656, 1948.

[35] (Schm 94) Schmidhuber, J.H. "Discovering Problem Solutions with Low Kolmogorov Complexity and High Generalization Capability," Technical Report FKI–194–94, Fakultät für Informatik, Technische Universität München, 1994.

[36] (Schn 73) Schnorr, C.P. "Process Complexity and Effective Random Tests," *J. Comput. System Sci.*, 7:376–388, 1973.

[37] (Sol 56) Solomonoff, R.J. "An Inductive Inference Machine," A privately circulated report, August 1956.

[38] (Sol 57) Solomonoff, R.J. "An Inductive Inference Machine," *IRE Convention Record, Section on Information Theory*, 1957.

[39] (Sol 59) Solomonoff, R.J. "A Progress Report on Machines to Learn to Translate Languages and Retrieve Information," *Advances in Documentation and Library Science*, Vol. III, pt. 2, pp. 941–953. (Proceedings of a conference in September 1959).

[40] (Sol 60a) Solomonoff, R.J. "A Preliminary Report on a General Theory of Inductive Inference," Report V– 131, Zator Co., Cambridge, Mass., Feb. 4, 1960.

[41] (Sol 60b) Solomonoff, R.J. "A Preliminary Report on a General Theory of Inductive Inference," (Revision of Report V–131), Contract AF 49(639)–376, Report ZTB–138, Zator Co., Cambridge, Mass., Nov, 1960.

[42] (Sol 64a) Solomonoff, R.J. "A Formal Theory of Inductive Inference," *Information and Control*, Part I: Vol 7, No. 1, pp. 1–22, March 1964.

[43] (Sol 64b) Solomonoff, R.J. "A Formal Theory of Inductive Inference," *Information and Control*, Part II: Vol 7, No. 2, pp. 224–254, June 1964.

[44] (Sol 75a) Solomonoff, R.J. "Inductive Inference Theory - a Unified Approach to Problems in Pattern Recognition and Artificial Intelligence," *Proceedings of the 4th International Conference on Artificial Intelligence*, pp 274- -280, Tbilisi, Georgia, USSR, September 1975.

[45] (Sol 75b) Solomonoff, R.J. "The Adequacy of Complexity Models of Induction," *International Congress of Logic, Methodology and Philosophy of Science*, Section VI, pp 19- -20, London, Ontario, Canada, September 1975.

[46] (Sol 78) Solomonoff, R.J. "Complexity-Based Induction Systems: Comparisons and Convergence Theorems," *IEEE Trans. on Information Theory*, Vol IT–24, No. 4, pp. 422- -432, July 1978.

[47] (Sol 84) Solomonoff, R.J. "Optimum Sequential Search," Oxbridge Research, P.O. Box 391887, Cambridge, Mass. 02139, June 1984.

[48] (Sol 86) Solomonoff, R.J. "The Application of Algorithmic Probability to Problems in Artificial Intelligence," in *Uncertainty in Artificial Intelligence*, Kanal, L.N. and Lemmer, J.F. (Eds), Elsieevr Science Publishers B.V., PP. 473–491, 1986.

[49] (Sol 89) Solomonoff, R.J. "A System for Incremental Learning Based on Algorithmic Probability," *Proceedings of the Sixth Israeli Conference on Artificial Intelligence, Computer Vision and Pattern Recognition*, Dec. 1989, pp. 515–527.

[50] (Van 63) van Heerden, P.J. "A General Theory of Prediction," Technical Report, Polaroid Corp., Cambridge, Mass., 1963.

[51] (Wal 68) Wallace, C.S and Botouln, D.M. "An Information Measure for Classification," *Computing Journal*, 11:185–195, 1968.

[52] (Wil 70) Willis, D.G. "Computational Complexity and Probability Constructions," *Journal of the Assoc. of Comp. Mach.*, pp. 241–259, April 1970.

[53] (Zad 65) Zadeh, L. "Fuzzy Sets," *Information and Control*, pp. 338–353, 8(3), June 1965.

[54] (Zvo 70) Zvonkin, A.K.,and Levin, L.A., “The Complexity of Finite Objects and the Development of the Concepts of Information and Randomness by Means of the Theory of Algorithms,” *Russ. Math. Survs*, Vol. 25, No. 6, pp. 83–124, 1970.