No Better Ways to Generate Hard NP Instances than Picking Uniformly at Random*

Russell Impagliazzo†
Computer Science Department
University of Toronto
Toronto, Ontario, Canada

Leonid A. Levin‡
Computer Science department
Boston University
111 Cummington St. Boston

Abstract

Distributed NP (DNP) problems are ones supplied with probability distributions of instances. We can consider their hardness for typical instances rather than just for the worst case (which may be extremely rare). Reductions between such problems must approximately preserve the distributions. A number of papers show completeness of several natural DNP problems in the class of all DNP problems with P-time computable distributions. This approach has been criticized as too restrictive: hard instances may be generated with samplable but not computable in P-time distributions. There were doubts whether *natural* problems (which all have simple distributions) may be complete for the class of all NP problems with samplable distributions.

We show that every DNP problem complete for P-time computable distributions is also complete for all samplable distributions. This rather surprising observation makes the concept of average case NP completeness robust and the question of the average case complexity of complete DNP problems a natural alternative to P=?NP.

Similar techniques also yield a connection between cryptography and learning theory. Unless one-way functions exist, we can almost always estimate the probability of any next value of an unknown sequence generated in a known polynomial time. The average-case time taken to estimate this probability will be polynomially related to the average-case time to invert the hardest one-way function. Thus, the results of the recursion-theoretic model of inductive inference ([Solomonoff 64]; see also [Li Vitanyi 89]) can be achieved within the same complexity which suffice to invert one-way functions. Since it is hard to extrapolate pseudo-random functions, the converse follows from [Hastad Impagliazzo Levin Luby 90]. Thus, universal extrapolation is possible precisely when cryptography is not.

1 Introduction.

A common misinterpretation of NP completeness is that all NP-complete problems are hard, it makes no sense to seek fast algorithms. On this basis some such problems

*FOCS-1990

†Supported by NSF grant #CCR-88-13632

‡(e-mail to Lnd@cs.bu.edu) Supported by NSF grant #CCR-86-07492, MIT and Sun Microsystems

generated much hope in cryptography: "To cheat, an adversary will have to solve an NP-complete problem." [Karp 76] and others (see [Johnson 84] for a survey) noticed that this was naive. While **worst case** instances of NP-complete problems defeat our algorithms, such instances may be extremely rare. In fact, fast *on average* algorithms were found for a great many NP-complete problems. Still, some other NP problems resisted such attacks. These issues turned out to be subtle and it was not clear how a theory could distinguish intrinsically hard on average problems.

Average case complexity is very sensitive to the choice of a particular probability distribution. Restricting the class of distributions to those computable in polynomial time (P-distributions), some natural problems were shown to be *average case complete*. [Levin 86, Venkatesan Levin 88, Gurevich 87,89] consider some NP problems with uniform probability distributions: Tiling, Graph Coloration, Matrix Decomposition, etc. Their random instances are as hard as those of any P-distributed NP-problem.

Using one-way functions, however, one can generate (sample) in polynomial time instances with very weird distributions. These *samplable* (generateable in P-time) distributions belong to #P and may be neither P-time computable nor approximable by such. In general, one cannot even distinguish effectively outputs of maximal probability from those of probability 0. See [Ben-David Chor Goldreich Luby 89] for an analysis of this and other issues concerning average-case complexity. Although possibly strange, these distributions are precisely the type which can be generated by realistic algorithms.

The class of samplable NP problems has its own complete members, but until now all known such problems were artificial. If these problems have average P-time algorithms, then there is no way in time t to generate NP instances of complexity more than $t^{O(1)}$ and the P=?NP question becomes academic.

Still, one could imagine that all nice (P-distributed) NP problems are easy, while one-way functions exist (with uniformly distributed inputs but #P-distributed outputs) allowing Cryptography, Pseudo-randomness, etc.[1] Were this gap possible, the concept of completeness for P-distributed NP problems would be too weak (even meaningless). At the same time, completeness for samplable NP problems would be too strong: natural problems are P-distributed and why should any of them be complete for samplable distributions?

We rule out such a gap: both concepts are equivalent. Surprisingly, every samplable NP problem reduces to a P-distributed one, which makes the concept of average case completeness very robust.

The same techniques also yield a connection between learning theory and cryptography. We use the word "extrapolation" rather than "learning", both to distinguish our model from time-invariant models and because "learning" has distracting psychological connotations. We implement Solomonoff's recursion-theoretic notion of extrapolation in a complexity-restricted setting. We observe that the average time needed for

[1] One should not be misled by the uniform distribution of coin-flips ω used to generate instance $x = h(\omega)$ of a samplable problem $x \in ?L \in$NP. The apparently equivalent problem $\omega \in ?h^{-1}(L)$ may, in fact, be easier, since ω may contain more information than x. Paraphrasing [Ben-David Chor Goldreich Luby 89], the difference is that between generating problems that are hard for someone else and those that are hard for oneself.

(either almost optimal or any useful at all) universal extrapolation is polynomially related to the average time needed to invert the hardest one-way function.

As pointed out in [Pitt Warmuth 88], the pseudo-random functions of [Goldreich Goldwasser Micali 84] (if they exist) are unlearnable in any realistic sense. Since any one-way function yields a pseudo-random one ([Hastad Impagliazzo Levin Luby 90]), we have a natural necessary and sufficient condition for universal extrapolation to be possible.[2]

In contrast to other complexity-based learning models (e.g., [Valiant 84]), we ignore the performance of extrapolation algorithms in the worst-cases of very small probability and do not assume time-invariance of the output distribution of the unknown machine.

2 The Results.

Let $S_n = \{0,1\}^n$ and S be the set of all binary strings identified with integers when needed. Let P be the class of functions over S computable in polynomial time and preserving the input length to within a polynomial. Let distributions μ assign to every **interval** $A \subset S$ its probability $\mu(A)$ which can be extended by additivity to all other sets of integers. We will assume $\mu(A)$ to be a rational number but we can deal with real-valued distributions by approximating them (within a constant factor) with rational-valued ones. For convenience, we allow $\mu(S) < 1$ by assuming that with some probability we may get a nil $\notin S$ outcome. We will call $\lambda(\{x\}) = 2^{-|x|}/|x|(|x|+1)$ the uniform distribution. The distribution of outputs of $h : S \to S$ on μ-distributed inputs we denote $\overline{h}(\mu)$.

Without loss of generality we will consider NP-problems in the form of inverting functions $f \in$P: given $y \in S$ find $x \in f^{-1}(y)$. E.g., the Hamiltonian cycle problem may be stated as inverting the function $f(\langle G, H\rangle)$ which outputs the graph G, if H is a Hamiltonian cycle (or $0 \ldots 0$ otherwise). If $x = f(w)$, we call w a *witness* that $x \in f(S)$. A DNP problem is a pair (f, μ) of a function and a distribution of its outputs (instances of the inverting problem). A DNP problem is P-distributed if $\mu \in$P and samplable if $\mu = \overline{h}(\nu)$, where $h, \nu \in$P.

We will show a randomized reduction R, Q from any NP problem f on a samplable distribution to another, uniformly distributed, NP problem g. It will work as follows. On instance x of the first problem and random string α, R produces either an instance y of the second problem or "Nil". This y may or may not be "good"; if it is, Q maps any witness that $y \in g(S)$ to a witness that $x \in f(S)$. It need not be easy to check if y is good: given any witness to $y \in g(S)$ we can verify that it indeed generates a witness to $x \in f(S)$ and for any "yes" instance $x \in f(S)$, we require at least a $|x|^{-O(1)}$ chance that y is good. Then, repeating the reduction with a suitable polynomial number of α's, we almost surely get a good y at least once; if an algorithm for the second problem quickly finds a witness to $y \in g(S)$, then we find a witness to $x \in f(S)$ in a comparable time. To ensure that fast on-average algorithms for the second problem work quickly on average for good y, we

[2] Conceivably, the hardest one-way function may take polynomial-time to invert on one infinite set of input lengths, and exponential on another. Then the extrapolation algorithm would also be fast on some lengths, and slow on others. Thus, it might be that neither universal extrapolation nor cryptography are possible on all lengths of input. So the complementary relation between cryptography and universal extrapolation holds separately for each length range: any given level of technology is capable of either universal extrapolation or cryptography, but not both.

must show that the distribution of $R(x,\alpha)$ (for x with its samplable distribution, and random α) is at most uniform within a factor of $|y|^{O(1)}$. Thus, good $y = R(x,\alpha)$ are unlikely to fall in the small set of harder-than-most instances.

Our reductions are between *search* rather than *decision* problems. However, [Ben-David Chor Goldreich Luby 89] reduce any distributed search problem "invert f" to a decision problem of existence of $x \in f^{-1}(y)$, $Ax = 0$ with uniformly distributed rectangular matrix A.

In the formal definition of randomized reduction between DNP problems, we separate the randomized reduction into a randomizing step, followed by a deterministic reduction. Thus, we refer to one problem having a randomization which is reducible to another problem, rather than being random reducible to the problem.

We define (randomized) reductions of DNP problems by simplifying slightly [Venkatesan Levin 88]. A reduction of DNP problem (f,μ) to DNP problem (g,ν) is a pair of algorithms $R, Q \in$P with the following conditions. Let $D = \{x : \mu(\{x\}) > 0\}$. Then R, Q preserve:

1. the distribution: $\overline{R}(\mu) \leq \nu$,
2. the solvability: $R(f(S) \cap D) \subseteq g(S)$, and
3. the witnesses: if $R(x) = g(w)$, $x \in D$ then $x = f(Q(w))$.

So, typical instances are mapped into typical ones and average case complexity is preserved.[3]

[3]These requirements can be weakened: Q may have x as an extra input, R may run only in *average* polynomial time, many instances of g may be used to invert one instance of f, etc.

In many cases reductions should be randomized. To allow that we supplement the instances of the original problem with random padding (ignored by f but useful for making random decisions in the reductions). A DNP problem (f',μ') is a *randomization* of DNP problem (f,μ) if

- $f'(\langle w,\alpha\rangle) = \langle f(w),\alpha\rangle$ and
- $\mu'(\{\langle x,\alpha\rangle\})$ is either 0 or $\mu(\{x\})\lambda(\{\alpha\})$.

Here $D_x \stackrel{\text{def}}{=} \{\alpha : \mu'(\{\langle x,\alpha\rangle\}) > 0, \text{ or } \mu(x) = 0\} \subset S_{|x|^c}$ (with a constant c) must be a set of polynomial measure: $\lambda(D_x) = |x|^{-O(1)}$. (Intuitively, D_x is the set of random inputs which produce "good" y's). There is no need to require $\alpha \in D_x$ to be easy to test, since we can check directly whether the witness produced by the reduction is valid (i.e. $x' = f'(Q(w'))$) without verifying $x' = \langle x,\alpha\rangle \in D$.

Definition 1 DNP problem (f,μ) is *samplable complete* if every samplable NP problem has a randomization reducible to (f,μ).

Theorem 1 *There exists a P-distributed NP problem which is samplable complete.*

Consequently, Graph Coloration, Matrix Decomposition, Tiling and other problems complete for P-distributed NP problems are complete in the above strong sense.

3 Intuitive Idea.

Consider any samplable problem $(f,\overline{h}(\nu))$. Assume without loss of generality $|h(x)| \leq |x|$. Padding and reductions from [Levin 86] can reduce it to the case with $\nu = \lambda$.

The idea of the proof is that either the problem "given p, invert f on $h(p)$" is hard, or else p must yield some information that

$h(p)$ does not. If the first holds, we have a hard problem on the uniformly distributed p. If the second, h must be a one-way function of a kind, in that, given $x = h(p)$ it should be difficult to generate a *random* p' with $h(p') = x$. This kind of function was called a *distributionally* one-way function in [Impagliazzo Luby 89], who show how to construct a normal one-way function from a distributionally one-way function. Thus, we have either a hard average-case problem, or a one-way function.

However, a one-way function does not automatically yield a hard average-case problem. We invert h on $x = h(p)$, where p, rather than x, is chosen at random. However, if we can approximate $\|h^{-1}(h(p))\|$, then techniques of [Hastad Impagliazzo Levin Luby 89] allow to convert h into a one-way function h' which is almost always one-to-one and approximately length-preserving. We let $h'(p, g_1, g_2) = g_2(h(p) \circ g_1(p)) \circ g_1 \circ g_2$, where, for $l = \log \|h^{-1}(h(p))\|$, g_1 is a hash function mapping $|p| = n$ bit strings to $l + O(\log n)$ bit strings, and g_2 is a hash function from strings of length at most $n + l + O(\log n)$ to $n + O(\log n)$ bit strings. The idea is that g_1 provides a random "label" determining, with $h(p)$, the value of p without revealing any useful information. Thus, adding $g_1(p)$ makes h almost one-to-one. g_2 then compresses this information, making the function almost length-preserving, without losing security or making it significantly less one-to-one. Since h' is almost length-preserving and one-to-one, its range is not too far away from the uniform distribution: a random string will have a good chance of having a pre-image under h', and this pre-image will often be hard to find. This will be our hard average-case problem.

A problem remains that it may be infeasible to approximate $l = \log(\|h^{-1}(h(p))\|)$. This problem is solved simply by guessing a value at random from 1 to n; there is a good chance we are right.

The proof outline above has several steps and cases. It turns out possible to combine most of these steps and cases into a single, simple reduction. The above sketch might be helpful to keep in mind when reading the proof below, but is not followed to the letter.

A few of the differences between the proof outline above and our actual construction are as follows. First, instead of needing our one-way function to be one-to-one with high probability, it will suffice that it is one-to-one with a reasonable chance (for any fixed value of x). Therefore, we can remove the $O(\log n)$ terms in the number of bits we hash to.

Second, instead of using hash function g_1 to provide l bits of information about the p used to generate x, we will use a *length-increasing* hash function g' mapping $k = n - l$ bits to n bits so that the p generating x is in the range of g', i.e., $x = h(g'(r))$ for a k bit string r. This simplifies slightly the construction, and is basically equivalent: since a $1/2^l$ fraction of n bit strings are in the range of g', the fact that x is in the range reveals l bits worth of information about x. (To better see the equivalence, consider the special case when g_1 is a random $n \times l$ matrix, and $g'(r) = Mr + v$, where M is a random $n - l \times n$ matrix, and v is a random vector of length n. Then knowing that $g_1(p) = s$ for some fixed s or that p is in the range of g' both convey precisely the information that x is an element of a skew-subspace of dimension l.) Before, the role of g_2 was to condense an $n + l$ bit string, $h(p) \circ g_1(p)$, that is determined by the n bit string p, back into an n bit string. Now we have a string $g'(r)$ for r a $n - l$ bit string, so we will instead want g_2 to compress this n bit string

into its information content of $n - l$ bits.

Third, we will combine the problems of inverting h' and of finding a witness for x.

4 Construction/Proof.

For concreteness, we will use the following family of simple hash functions.

Let G_n be the set of pairs of n bit strings. We can interpret such strings as elements of GF(2^n) (the finite field of cardinality 2^n). Let $x_{\leq k}$ denote the k-bit prefix of x. Then, for any $k \leq n$, we can think of $r \in S_k$ as describing the first 2^k elements of the field, and G_n as a family of maps $S_k \to S_n$ by $g_{(a,b)}(r) = ar+b$. We can also think of an element (c, d) of G_n as determining a hash function $S_n \to S_k$ by $g_{(c,d)}(x) = (cx + d)_{\leq k}$. In some cases when we care only about collisions of different x, we need not b and set it to 0. In either context, G_n is a standard example of a family of pairwise independent universal hash functions ([Carter Wegman 79]). The proofs below hold equally well for any such family. However, G_n is convenient: simple, concrete, easy to compute, and has $2n$-bits elements.

We reduce DNP problem $(f, \overline{h}(\lambda))$ to a P-distributed (F, λ). The inputs to F are: $w, a, b, c \in S_n, r \in S_{k-1}$, $k \leq n$. F outputs $0 \ldots 0$ if $h(ar+b) \neq f(w)$. Otherwise it outputs $(f(w)c)_{\leq k}, a, b, c$.

Thus, to invert F one first must find r, by inverting the function $h'(r, a, b, c) = ((h(ar+b)c)_{\leq k}, a, b, c)$, and then invert f on $h(ar+b)$. The proof that the reduction is valid basically shows that h' is a frequently one-to-one function which is as hard to invert as it is to find random pre-images of h.

The randomization supplements instances x of $(f, \overline{h}(\lambda))$ with a random padding $\alpha = (k, a, b, c)$ restricted to a set D_x described below. The reduction $R(x, \alpha)$ yields $((xc)_{\leq k}, a, b, c)$, while $Q(w, r, a, b, c) = (w, \alpha)$.

We need to define D_x, the "good" random supplements for x, so that the reduction conditions are satisfied. Intuitively, D_x is the set of supplements where we guess the value of k correctly, and then pick a, b, c so that h' has a unique pre-image on the problem reduced to. Note, that there is no need for membership of D_x to be easily decidable. For each $\alpha = (k, a, b, c) \in D_x$ we require that:

1. there are "good" r, s.t. $h(ar+b) = x$ (then R preserves the solvability), and

2. for all "bad" r, $(h(ar+b)c)_{\leq k} \neq (xc)_{\leq k}$ (so, $(xc)_{\leq k}$ determines x and Q preserves the witnesses); and

3. $k = 1 - \lfloor \log \lambda(h^{-1}(x)) \rfloor$ (then R preserves the distribution, as x is unique and $\overline{h}(\lambda)(x) = O(2^{-k})$).

So, one only needs to show $\lambda(D_x) = |x|^{-O(1)}$, i.e., the reduction has a polynomial chance of being good. The clause 3 determines k and has, obviously a $1/n$ chance.

Given clause 3, the first clause holds for most (a, b). Let $X = h^{-1}(x)$. $\|X\| \geq 2^{n+1-k}$. There are $\geq 2^n$ pairs $r \in S_{k-1}, p \in X$. For each (r, p) there are 2^n pairs (a, b), s.t. $ar + b = p$. For each two distinct pairs (r, p), $(r'p')$ at most one pair (a, b) satisfies both equations. So, by inclusion-exclusion, at least $2^n 2^n - 2^n(2^n - 1)/2 > 2^{2n}/2$ pairs (a, b) satisfy $ar+b = p$ for some $r \in S_{k-1}, p \in X$.

Finally, for each a, b, clause 2 holds on most c. Indeed, $(h(ar+b)c)_{\leq k} - (xc)_{\leq k} = ((h(ar+b)-x)c)_{\leq k}$ is uniformly distributed over c, for any given $a, b, r, x \neq h(ar+b)$. Thus, at most 2^{-k} fraction of pairs r, c can violate clause 2. So, for most c, clause 2 holds with all $r \in S_{k-1}$.

5 Extrapolation.

One-Way Functions Defeat Universal Extrapolation. The above technique also helps to clarify whether the universal ability of extrapolation is computationally feasible. This depends entirely on the existence of one-way functions. If there exist a function $f \in$P which is infeasibly hard on average to invert, then one can construct a pseudo-random function $g_s(x)$ (see [Hostadt Impagliazzo Levin Luby 90, Goldreich Levin 89]). Such function cannot be distinguished from random functions $G(x)$ by an observer which can query it for any number of values of x but does not know s and does not have computational power huge enough to invert $f(s)$ for average s. Obviously then, any attempt of such observer to learn something (in any meaningful sense of the word) about g_s will be absolutely fruitless. We will see, on the other hand, that if all $f \in$P are easy on average to invert, than our power to extrapolate probabilistic functions from P is pretty strong and universal.

Recursively Enumerable Extrapolation In [Solomonoff 64] the concept of a Universal probability distribution is proposed which should provide a foundation for extrapolating unknown probabilistic functions. This Universal measure U(x) proposes an *a priori* probability that the binary sequence produced by an unknown probabilistic process has x as a prefix. The approach is based on several insights.

First, distributions, according to which x is random, are not unique and should include some defined by short programs. E.g., if distribution μ_p is complicated due to its dependence on a messy random parameter p, then x is, as well, random with respect to a simpler distribution μ, which first chooses p at random and then generates x according to μ_p. U is an upper bound, within a reasonable constant factor, for μ that have short programs.

Second, while U$(\{x\})/\mu(\{x\})$ can take arbitrarily large values c, this can happen only with the small probability $1/c$ for x generated according to μ. So, U is likely to produce both reasonable upper and lower bounds.

Finally, while the ratio between U and μ is likely to be not very large, it will not be small either. However, it can be ignored in comparison with huge (typically exponential in $|x|$) ratios of probabilities of x under different distributions.

The ideas of [Solomonoff 64] were deep but technical implementation was imperfect: sums in its formulas diverge, limits do not exist etc. A better implementation was proposed in [Zvonkin Levin 70, Levin 73]). There, universal distribution was defined as an enumerable *semi-additive* function (semimeasure) $m(x) \geq \sum_i m(x,i)$, $x = x_1, ..., x_k$ rather than an *additive* measure U$(x) = \sum_i$U(x,i). There is a Universal semimeasure M, dominating all others within a constant factor. Extrapolation of sequence x according to this distribution (if it were possible) would indeed be powerful. One would make no more mistakes than the length of the shortest program generating x. Besides, if x is generated with computable (or enumerable) distribution μ then $\mu(x) = O(M(x))$ and the chance of $M(x) > c\mu(x)$ is at most $1/c$. The problem is: M is not computable!

Space-Restricted Extrapolation. [Kolmogorov 65] proposed to impose computational restrictions on algorithms with respect to which Kolmogorov Complexity is defined (formalized as *complexity majorants* in [Zvonkin Levin 70]). This covers M also,

since $|\log M(x)|$ is a version of Kolmogorov Complexity. There is a samplable in space S version $M_S(x)$ of M. It dominates within a constant factor all distributions samplable in space $S - 1/o(1)$. If x is generated (within space so restricted) by a probabilistic algorithm with probability $\mu(x)$, then the statements of the previous paragraph hold for M_S as well as for M. M_S can be computed in probabilistic space $O(S)$, but unfortunately, this may take time 2^S.

Time-Restricted Extrapolation. Realistic extrapolations should take a bounded (polynomial) time. The previous paragraph, however, cannot be modified by simply replacing "space" with "time". Possible existence of one-way functions (and their pseudorandom offspring) would prevent the time restricted version M_T from being easily computable. Suppose, however, that one-way functions do not exist, i.e. all $f \in$P are invertible in small (say polynomial) average time. Then M_T can be estimated by a fast on average probabilistic algorithm and used for extrapolation in the above manner. Indeed, let U be the Universal Turing Machine which interprets the prefix p of its binary input $\alpha = p\alpha'$ as a description of an arbitrary Turing Machine P and simulates up to T steps of $P(\alpha')$. In the course of this computation it will write the output bits of $U_T(\alpha)$ on its write-only output tape. Let $M_T(x)$ be the probability that x is a prefix of $U_T(\alpha)$ and $k = \lfloor -\log M_T(x) \rfloor$.

Then one can apply the same reasoning as in the previous section. Let $a, b \in \mathrm{GF}(2^T)$ and the *complexity* $K_{a,b}(x)$ be the minimal $|r|$, s.t. $U_{|a|}(ar+b)$ has x as a prefix. Then $|K_{a,b}(x) + \log M_T(x)| \leq 3$ for each x and at least $2/3$ of a, b. So, one can estimate $M_T(x)$ by trying to invert the transformation $(a, b, r) \to (a, b, |r|, U_{|a|}(ar+b))$. This yields extrapolation method akin to "Occam Razor" principle: the likeliest are strings of the smallest complexity. All benefits of the above paragraphs will then be applicable to the time-restricted version M_T of the universal distribution.

This can be summarized as follows:

Definition 2 *Let $s : S \to S$ be a time-constructible, monotone function. We say a function t is* s-feasible *if $t(n) \leq s(n^{O(1)})n^{O(1)}$. A probabilistic algorithm $A(\omega, x, \varepsilon)$, running in time $T_A(|x|/\varepsilon)$ is* s-feasible *if T_A is. A function $f \in P$ is* s-one-way *if no s-feasible algorithm A can, for every length n, invert f on $y = f(x)$ (i.e., $f(A(\omega, y, \varepsilon)) = y$) with probability $1 - \varepsilon$ over random $x \in \{0,1\}^n$.*

The above definition differs slightly from standard treatment of the inverting probabilities of one-way functions, but a trivial padding argument shows their equivalence.

Lemma 1 *If there are no s-one-way functions, then, for every polynomial-time computable function F and any samplable distribution μ, there is an s-feasible algorithm which, on input $y = F(x), \varepsilon$ for x selected according to μ, with probability at least $1-\varepsilon$, computes $\mu(F^{-1}(x))$, within a factor of $1+\varepsilon$.*

Applying the lemma to a polynomial-time universal machine U as above, we get:

Proposition 1 *The following is equivalent to the non-existence of s-one-way functions: there is an s-feasible algorithm which, on input $y, 1^T$ and ε, computes $M_T(y)$, within a factor of $1+\varepsilon$, with probability $1-O(\varepsilon)$, when $y \in \{0,1\}^n$ is selected according to any unknown distribution μ, samplable in time T.*

Details of proofs and applications to extrapolation will be given in the final paper.

6 Acknowledgements.

We are grateful Mike Luby, Yuri Gurevich, and Shai Ben-David for helpful conversations.

References

[1] Shai Ben-David, Benny Chor, Oded Goldreich and Michael Luby. On the Theory of Average Case Complexity. ACM, *21st Annual Symposium on Theory of Computing*, 1989, 204-216.

[2] J. Carter and M. Wegman. Universal Classes of Hash Functions. *JCSS* 18:143-154, 1979.

[3] O.Goldreich, H.Krawcyzk, M.Luby. On the Existence of Pseudorandom Generators. *FOCS*, 1988, 12-24.

[4] O. Goldreich, L. Levin. A Hard-Core Predicate for all One-Way Functions. ACM, *Symp. on Theory of Computing*, 1989, pp. 25-32.

[5] Yuri Gurevich. Complete and Incomplete Randomized NP Problems. *FOCS*, 1987, pp. 111-117. Also to appear in *J. Comp. Sys. Sci.* as "Average Case Complexity".

[6] Yuri Gurevich. The Challenger-Solver Game. *Bull. of Europ. Assoc. for Theor. Comp. Sci.*, Oct. 1989.

[7] Yuri Gurevich. Matrix Decomposition Problem is Complete for the Average Case. These Proceedings, 1989.

[8] J. Hastad, R. Impagliazzo, L. Levin, M. Luby. Pseudo-Random Generators from Any One-way Function. To be published. Preliminary versions in STOC 1989 pp. 12-24, and 1990 pp. 395-404.

[9] D.Johnson. The NP-Completeness Column – an Ongoing Guide. *Journal of Algorithms* 5:284-299, 1984.

[10] R. Karp. The Probabilistic Analysis of some Combinatorial Search Algorithms. *Algorithms and Complexity*. J.F. Traub, ed., Academic Press, NY 1976, pp. 1-19.

[11] A.N. Kolmogorov. Three Approaches to the Concept of the Amount of Information. *Probl. Inf. Transm.*, 1(1), 1965.

[12] L. Levin. On the Notion of a Random Sequence. *DAN SSSR = Soviet Math. Dokl.* 14(5):1413-1416, 1973.

[13] L. Levin. Average Case Complete Problems. *SIAM Journal of Computing* 15: 285-286, 1986.

[14] Ming Li, Paul Vitanyi. A Theory of Learning Simple Concepts under Simple Distributions and Average Case Complexity for the Universal Distribution. *FOCS*, 1989.

[15] L. Pitt, M. K. Warmuth. Reductions among prediction problems: on the difficulty of predicting automata. *Structure in Complexity*, 1988, pp. 60-69.

[16] R.J. Solomonoff. A Formal Theory of Inductive Inference. *Information and Control* 7(1):l-22, 1964.

[17] Adi Shamir. A Polynomial Time Algorithm for Breaking the Basic Markle-Hellman Cryptosystem. *FOCS* 1982.

[18] L. G. Valiant. The theory of the learnable. *Comm., ACM*, 27(11):1134-1142, 1984.

[19] R. Venkatesan and L. Levin. Random Instances of a Graph Coloring Problem are Hard. *STOC*, 1988.

[20] A.K. Zvonkin, L. Levin. The Complexity of finite objects and the Algorithmic Concepts of Information and Randomness. *UMN = Russian Math. Surveys* 25(6):83-124, 1970.