On the Theory of Average Case Complexity [*]

Shai Ben-David[†]
Benny Chor
Oded Goldreich[‡]

Dept. of Computer Science
Technion, Haifa, Israel

Michael Luby[§]

Dept. of Computer Science
University of Toronto, Canada
and
ICSI, Berkeley, CA 94704

Draft from 1989, minor corrections 1997

Abstract

This paper takes the next step in developing the theory of average case complexity initiated by Leonid A Levin. Previous works [Levin 84, Gurevich 87, Venkatesan and Levin 88] have focused on the existence of complete problems. We widen the scope to other basic questions in computational complexity. Our results include:

- the equivalence of search and decision problems in the context of average case complexity;
- an initial analysis of the structure of distributional-NP (i.e. NP problems coupled with "simple distributions") under reductions which preserve average polynomial-time;
- a proof that if all of distributional-NP is in average polynomial-time then non-deterministic exponential-time equals deterministic exponential time (i.e., a collapse in the worst case hierarchy);
- definitions and basic theorems regarding other complexity classes such as average log-space.

An exposition of the basic definitions suggested by Levin and suggestions for some alternative definitions are provided as well.

Remark: This version seems to be a pre-publication draft from 1989. A better version has appeared in *Journal of Computer and system Sciences*, Vol. 44, No. 2, April 1992, pp. 193–219. I've corrected some errors which I found while scanning, but did not proofread this version. [O.G., 1997]

[*]An extended abstract has appeared in the *Proc. 21st ACM Symp. on Theory of Computing.* An abstract has appeared in the *Proc. 4th Conf. on Structure in Complexity Theory.*

[†]Supported by Technion V.P.R. Fund - E.N.J. Bishop Research Fund.

[‡]Partially supported by grant No. 86-00301 from the United States - Israel Binational Science Foundation (BSF), Jerusalem, Israel.

[§]Partially supported by a Natural Sciences and Engineering Research Council of Canada operating grant A8092 and by a University of Toronto grant.

1 Introduction

The average complexity of a problem is, in many cases, a more significant measure than its worst case complexity. This has motivated the development of a rich area in algorithms research – the probabilistic analysis of algorithms [Johnson 84, Karp 86]. However, this line of research has so far been applicable only to specific algorithms and with respect to specific, typically uniform, probability distributions.

The general question of average case complexity was addressed for the first time by [Levin 84]. Levin's work can be viewed as the basis for a theory of average NP-completeness, much the same way as [Cook 71] (and [Levin 73]) works are the basis for the theory of NP-completeness. Subsequently [Gurevich 87] has presented several additional complete problems and pointed out limitations of deterministic reductions in proving completeness results. [Venkatesan and Levin 88] showed the completeness, with respect to randomized reductions, of a graph coloring problem with uniform distribution. In this paper we widen the scope of investigation and consider basic computational questions in the context of average case complexity.

(A reader not familiar with the basic definitions of average-case complexity, may find it useful to read some exposition of these definitions before proceeding to the rest of this introduction. Such an exposition is provided in Section 2 (and the appendices).)

An average case complexity class consists of pairs, called distributional problems. Each such pair consists of a decision (search) problem and a probability distribution on problem instances. Most of our work deals with the class $\mathrm{DistNP} \stackrel{\mathrm{def}}{=} \langle \mathrm{NP}, \text{P-computable} \rangle$, defined by [Levin 84], which is a distributional analogue of NP. P-computable is the class of distribution functions that can be computed in polynomial-time (i.e., there exists a polynomial time algorithm that on input x computes the accumulative probability of all strings $y \leq x$). The easy distributional problems are those solvable in average polynomial-time. We denote this class by Average-P. Reductions between distributional problems are defined in a way guaranteeing that if Π_1 is reducible to Π_2 and Π_2 is in Average-P, then so is Π_1.

A basic question regarding the theory of computational complexity is the relation between search and decision problems. Unfortunately, the standard polynomial-time Turing reduction of search problems to decision problems is not applicable in the distributional context. Instead, we present a randomized reduction of DistNP search problems to DistNP decision problems. Interestingly, this reduction can be carried out in RNC, yielding a reduction of P search problems to P decision problems (cf. [Karp Upfal and Wigderson 85]). Without such a result the study of decision problems does not reflect the structure of search problems.

If DistNP is not a subset of Average-P, then the complete problems in DistNP are not in Average-P. A natural question is whether every DistNP problem is either in Average-P or complete for DistNP. We resolve this question by showing that problems which are neither easy nor complete do exist. In fact, we show that the structural results of classical complexity theory (e.g., of [Ladner 75]) can be translated to the distributional context. Furthermore, we define a notion of one distribution being "harder" than another, and demonstrate a rich structure of distributions.

It is not clear whether DistNP $\subseteq$ Average-P (even if $\mathrm{P} \neq \mathrm{NP}$). We give strong indication that DistNP is not a subset of Average-P by relating this question to a classical one in (worst case) complexity. Specifically, we prove that if DistNP $\subseteq$ Average-P then $\mathrm{NTime}(2^{O(n)}) = \mathrm{DTime}(2^{O(n)})$.

Of all the definitions made in [Levin 84] the most controversial one is the association of the class of "simple" distributions with P-computable, which may seem too restricting. We present a wider family of natural distributions, P-samplable, which consists of distributions that can be sampled by probabilistic algorithms working in time polynomial in the length of the sample generated. We define the class of distribution problems ⟨NP, P-samplable⟩ and present complete problems for this class. We show that if one-way functions exist, then there are P-samplable distributions that are "very far" from any P-computable distribution. However, it seems that the distributions in P-samplable are too complicated.

Can the theory of average case complexity be meaningfully applied to structures other than NP with polynomial-time reductions? We believe that the answer is in the affirmative and suggest definitions of distributional-P and average log-space reductions. We exhibit complete distributional problems for the class ⟨P, logspace-computable⟩, relate the distributional question "is ⟨P, logspace-computable⟩ $\subseteq$ Average-logspace?" to the worst-case question "is Dspace(n) = DTime($2^{O(n)}$)?".

1.1 Organization

Section 2 presents the basic definitions of the theory of average case complexity. Further discussion of the definition of "easy on the average" can be found in Appendix A. Section 3 deals with the question of search versus decision; Section 4 investigates the structure of DistNP; Section 5 relates questions regarding average case complexity to traditional questions of worst-case; Section 6 introduces and studies P-samplable distributions; and Section 7 presents definitions and results for average logSpace.

2 Definitions and Notations

In this section we present definitions of various concepts that are used throughout the paper. Most definitions originate from [Levin 84], but the reader is advised to look for further explanations and motivating discussions elsewhere (e.g., [Johnson 84, Gurevich and McCauley 87, Goldreich 88]).

For sake of simplicity, we consider the standard lexicographic ordering of binary strings. Any fixed efficient enumeration will do. (An *efficient enumeration* is a 1-1 and onto mapping of strings to integers which can be computed and inverted in polynomial-time.) By writing $x < y$ we mean that the string x precedes y in lexicographic order, and $y - 1$ denotes the immediate predecessor of y. Also, we associate pairs, triples etc. of binary strings with single binary strings in some standard manner (i.e. encoding).

Definition: (Probability Distribution Function:) A *distribution function* $\mu : \{0,1\}^* \to [0,1]$ is a non-decreasing function from strings to the unit interval [0,1] which converges to one (i.e., $\mu(0) \geq 0$, $\mu(x) \leq \mu(y)$ for each $x < y$, and $\lim_{x \to \infty} \mu(x) = 1$). The *density function* associated with the distribution function μ is denoted μ' and defined by $\mu'(0) = \mu(0)$ and $\mu'(x) = \mu(x) - \mu(x-1)$ for every $x > 0$. Clearly, $\mu(x) = \sum_{y \leq x} \mu'(y)$.

For notational convenience, we often describe distribution functions converging to some c$\neq$1. In all the cases where we use this convention it is easy to normalize the distribution, so that it converges to one. An important example is the *uniform* distribution function μ_0 defined as $\mu_0'(x) = \frac{1}{|x|^2} 2^{-|x|}$.

Definition: (A Distributional Problem): A *distributional* decision *problem* (resp. *distributional* search *problem*) is a pair (D, μ) (resp. (S, μ)), where $D : \{0,1\}^* \to \{0,1\}$ (resp. $S \subseteq \{0,1\}^* \times \{0,1\}^*$) and $\mu : \{0,1\}^* \to [0,1]$ is a distribution function.

In the sequel we consider mainly decision problems. Similar formulations for search problems can be easily derived.

2.1 Average-P and Distributional-NP

Simple distributions are identified with the P-computable ones. The importance of restricting attention to simple distributions is demonstrated in Sections 5 and 6.
Definition: (P-computable): A distribution μ is in the class P-computable if there is a deterministic polynomial time Turing machine that on input x outputs the binary expansion of $\mu(x)$ (the running time is polynomial in $|x|$).
It follows that the binary expansion of $\mu(x)$ has length polynomial in $|x|$.

If the distribution function μ is in P-computable then the density function, μ', is computable in time polynomial in $|x|$. The converse, however, is false, unless P = NP (see [Gurevich and McCauley 87]). In spite of this remark we usually present the density function and leave to the reader the verification that the corresponding distribution function is in P-computable.

We now present the class of distributional problems which corresponds to (the traditional) NP. Most of the results in the paper refer to this class.

Definition: (The class DistNP): A distributional problem (D, μ) belongs to the class DistNP if D is an NP-predicate and μ is in P-computable. DistNP is also denoted $\langle$NP, P-computable$\rangle$.

The following definitions, regarding average polynomial-time, may seem obscure at first glance. It is important to point out that the naive formalizations of these definitions suffer from serious problems such as not being closed under functional composition of algorithms, being model dependent, encoding dependent etc. For a more detailed discussion, see Appendix A.

Definition: (Polynomial on the Average): A function $f : \{0,1\}^* \rightarrow \mathbf{N}$ is *polynomial on the average* with respect to a distribution μ if there exists a constant $\epsilon > 0$ such that

$$\sum_{x \in \{0,1\}^*} \mu'(x) \cdot \frac{f(x)^\epsilon}{|x|} < \infty$$

The function $l(x) = f(x)^\epsilon$ is *linear on the average* w.r.t. μ.
Thus, a function is polynomial on the average if it is bounded by a polynomial in a function which is linear on the average. In fact, the basic definition is that of a function which is linear on the average; see also Definitions 2 and 5 (in Sections 5.2 and 7, respectively).

Definition: (The class Average-P): A distributional problem (D, μ) is in the class Average-P if there exists an algorithm A solving D, so that the running time of A is polynomial on the average with respect to the distribution μ.

We view the classes Average-P and DistNP as the average-case analogue of P and NP (respectively). Another candidate for an analogue to NP (denoted Average-NP) is the class of distributional problems which can be solved by a non-deterministic machine running in average polynomial time with respect to a P-computable distribution. However, we feel that DistNP better addresses the original motivation of investigating the average case complexity of NP. All known results (e.g. [Levin 84, Gurevich 87, Venkatesan and Levin 88]), as well as the ones shown in this paper, for the class DistNP hold also for Average-NP.

2.2 Reducibility between Distributional Problems

We now present definitions of (average polynomial time) reductions of one distributional problem to another. Intuitively, such a reduction should be efficiently computable, yield a valid result and "preserve" the probability distribution. The purpose of the last requirement is to ensure that the reduction does not map very likely instances of the first problem to rare instances of the second problem. Otherwise, having a polynomial time on the average algorithm for the second distributional problem does not necessarily yield such an algorithm for the first distributional problem. Following is a definition of randomized Turing reductions. Definitions of deterministic and many-to-one reductions can be easily derived as special cases.

Definition: (Randomized Turing Reductions): We say that the probabilistic oracle Turing machine M *randomly reduces* the distributional problem (D_1, μ_1) to the distributional problem (D_2, μ_2) if the following three conditions hold.

1) *Efficiency:* Machine M is polynomial time on the average taken over x with distribution μ_1 and the internal coin tosses of M with uniform probability distribution (i.e., let $t_M(x,r)$ be the running time of M on input x and internal coin tosses r, then there exists $\epsilon > 0$ such that $\sum_{x,r} \mu_1'(x)\mu_0'(r) \cdot \frac{t_M(x,r)^\epsilon}{|x|} < \infty$, where μ_0 is the uniform distribution).

2) *Validity:* For every $x \in \{0,1\}^*$,

$$Prob(M^{D_2}(x) = D_1(x)) \geq \frac{2}{3}$$

where $M^{D_2}(x)$ is the random variable (determined by M's internal coin tosses) which denotes the output of the oracle machine M on input x and access to oracle for D_2.

3) *Domination:* There exists a constant $c > 0$ such that for every $y \in \{0,1\}^*$,

$$\mu_2'(y) \geq \frac{1}{|y|^c} \cdot \sum_{x \in \{0,1\}^*} Ask_M(x,y) \cdot \mu_1'(x)$$

where $Ask_M(x,y)$ is the probability (taken over M's internal coin tosses) that "machine M asks query y on input x".

In the definition of deterministic Turing reductions $M^{D_2}(x)$ is determined by x (rather than being a random variable) and $Ask_M(x,y)$ is either 0 or 1 (rather than being any arbitrary rational in $[0,1]$).

In the rest of the paper whenever we use the term *reduction* we mean a reduction of distributional problems, as defined above. We use $\propto^{\mathrm{T}}$ ($\propto^{\mathrm{T}}_{\mathrm{R}}$) to denote deterministic (resp. randomized) Turing reduction, and $\propto$ and $\propto_{\mathrm{R}}$ to denote many-to-one reductions.

It can be proven that if (D_1, μ_1) is deterministically (resp. randomly) reducible to (D_2, μ_2) and if (D_2, μ_2) is solvable by a deterministic (resp. randomized) algorithm with running time polynomial on the average then so is (D_1, μ_1).

Reductions are transitive in the special case in which on input x they ask queries of length at least $|x|^c$, for some constant $c > 0$. All reductions we present have this property.

2.3 A Generic DistNP Complete Problem

The following distributional version of *Bounded Halting*, denoted $\Pi_{BH} = (BH, \mu_{BH})$, is known to be DistNP-complete. The proof, presented in Appendix B, is due to [Gurevich 87] (an alternative proof is implied by [Levin 84]).

$BH(M, x, 1^k) = 1$ iff there exists a computation of the non-deterministic machine M on input x which halts within k steps. The distribution μ_{BH} is defined in terms of its density function

$$\mu'_{BH}(M, x, 1^k) = (\frac{1}{|M|^2} \cdot 2^{-|M|}) \cdot (\frac{1}{|x|^2} \cdot 2^{-|x|}) \cdot \frac{1}{k^2}$$

Note that μ'_{BH} is very different from the uniform distribution on binary strings (e.g., consider relatively large k).

3 Search versus Decision Problems

In this section we show that search and decision distributional problems are equivalent with respect to randomized reductions.

Before presenting our reduction of distributional search problems to distributional decision problems we remark that the standard Turing reduction of a search problems to the corresponding decision problem does not have the domination property. On input x, the oracle machine tries to find a "solution" y by asking an oracle queries of the form "is p a prefix of a solution to x". Before adapting this reduction to our setting, we need to fix a distribution μ_2 on the (x, p) pairs. A natural choice is $\mu'_2(x, p) = \mu'_1(x) \cdot \frac{1}{|p|^2} 2^{-|p|}$, where μ_1 is the input distribution to the search problem. However, with this choice of μ_2, the above reduction violates the domination condition, since when (for example) $|p| = |y|/2$ the reduction maps an instance of the search problem to a much more rare instance of the decision problem (the ratio of probabilities of these instances is $< 2^{-|y|/2}$). It's not clear how to construct *polynomial time computable* distributions for the above decision problem so that the reduction will satisfy the domination condition.

Instead, we reduce every distributional search problem to an appropriate (distributional) decision problem in two steps. First, we randomly reduce the (distributional) search problem to a related (distributional) problem with a unique solution. This reduction follows the underlying principles of the proof of [Valiant and Vazirani 85] (see also [Sipser 83] and [Stockmeyer 83]). Next, we reduce such a search problem to a related decision. The reader may easily verify that these reductions can be performed in fast parallel time (RNC).

3.1 Reducing a Search Problem to a Search of Unique Solution

Definition 1 : Let $S \subseteq \{0,1\}^* \times \{0,1\}^*$. The *unique solution search problem* of S is to find, on input x, the unique y satisfying $(x, y) \in S$. If no such y exists or it is not unique then nothing is required.

Theorem 1 : Let $\Pi_1 = (S_1, \mu_1) \in$ DistNP. Then there exists a unique solution search problem $\Pi_2 = (S_2, \mu_2) \in$ DistNP such that $\Pi_1 \propto_{\mathrm{R}}^{\mathrm{T}} \Pi_2$.

Proof : For sake of simplicity, assume $(x, y) \in S_1$ implies $|x| = |y|$. Let $n = |x|$ and $H_{n,k}$ be a set of universal hash functions (e.g., $n \times k$ Boolean matrices) as in [Carter and Wegman 79].

Define $S_2 \subseteq \{0,1\}^* \times \{0,1\}^*$ as follows: $(x', y) \in S_2$ for $x' = (x, k, h, \alpha)$, if $(x, y) \in S_1$, $h \in H_{n,k}$ and $h(y) = \alpha$. The density function μ'_2 assigns (x, k, h, α) probability $\mu'_1(x) \cdot n^{-1} \cdot |H_{n,k}|^{-1} 2^{-k}$ if $k \in \{1, 2, \ldots, n\}$, $h \in H_{n,k}$ and $\alpha \in \{0,1\}^k$, and 0 otherwise.

The reduction, effected by a probabilistic oracle machine M proceeds as follows. On input $x \in \{0,1\}^n$, machine M tries the following for every value of $k \in \{1, 2, \ldots, n\}$. It selects at random with uniform probability distribution an h in $H_{n,k}$ and $\alpha \in \{0,1\}^k$, makes the oracle query (x,k,h,α) and tests the reply. This is repeated $O(1)$ times.

Clearly, M is efficient. To see that M is a valid reduction consider $m = |\{y : (x,y) \in S_1\}|$ and suppose $m \geq 2$ (the case $m \leq 1$ is easy). Let $k = \lceil \log_2 m \rceil$ if $m \leq \frac{4}{3} \cdot 2^{\lceil \log_2 m \rceil}$ and $k = \lfloor \log_2 m \rfloor$ otherwise. Then $|m \cdot 2^{-k} - 1| \leq \frac{1}{3}$. For a randomly selected $h \in H_{n,k}$ and $\alpha \in \{0,1\}^k$, the expected number of y's satisfying $(x,y) \in S_1$ and $h(y) = \alpha$ is between $\frac{2}{3}$ and $\frac{4}{3}$. By the properties of universal hashing (i.e., pairwise independence of the images of pairs of points) it follows that the probability that there is a unique y satisfying the above conditions is $\geq \frac{1}{2}$ (use Chebyshev's inequality). In such a case the oracle returns the correct answer (i.e. this y).

It is left to verify that the reduction satisfies the domination condition. The quadruple (x,k,h,α) appears as a query of M with probability $O(1) \cdot \mu_1'(x) \cdot |H_{n,k}|^{-1} \cdot 2^{-k}$ while $\mu_2'(x,k,h,\alpha) = \mu_1'(x) \cdot n^{-1} \cdot |H_{n,k}|^{-1} 2^{-k}$. $\square$

Remarks:

- For some problems, like Π_{BH}, the proof can be easily modified so that the (search) problem Π_{BH} can be reducible to the unique solution (search) problem of Π_{BH}. In the case of Π_{BH} this is done by incorporating k, h and α into the input to a slightly modified machine and increasing the step count. Namely, the problem of finding a m-step long computation of M which accepts x is randomly reduced to the problem of finding the unique $(m+(mk)^{O(1)})$-step computation of M' which accepts $x' = (x,k,h,\alpha)$, where M' on input x' first lets M run on x and then checks whether applying h, to an encoding of the non-deterministic moves taken by M, yields α.

- Almost the same construction will reduce any DistNP decision problem to an DistNP decision problem with unique witnesses.

3.2 Reducing Search of Unique Solution to Decision problem

Theorem 2 : Let $(S_1, \mu_1) \in$ DistNP be a distributional unique-search problem. Then there exists a distributional decision problem $(D_2, \mu_2) \in$ DistNP such that $(S_1, \mu_1) \propto^{\mathrm{T}} (D_2, \mu_2)$.

Proof : Once again, assume that $(x,y) \in S_1$ implies $|x| = |y|$, and let $n = |x|$. Define $D_2 \subseteq \{0,1\}^* \times \{0,1\}$ as follows: For all x, $D_2(x,i) = \sigma$ if $1 \leq i \leq n$ and there exists y such that $(x,y) \in S_1$ and the i^{th} bit of y equals σ. For each (x,i), we set $\mu_2'(x,i) = \mu_1'(x) \cdot \frac{1}{|x|}$ if $1 \leq i \leq n$, and 0 otherwise.

The reduction maps x to the queries $(x,1), (x,2), \ldots, (x,|x|)$. Clearly, the reduction is efficient and valid (i.e. if there exists a unique y such that $(x,y) \in S_1$ then the bits of y are reconstructed by the answers to the above queries). Query (x,i) appears with probability $\mu_1'(x)$ while $\mu_2'(x,i) = \frac{\mu_1'(x)}{|x|}$, hence domination property is satisfied as well. $\square$

For some problems, like Π_{BH}, the proof can be easily modified so that the (search) problem Π_{BH} can be reducible to the decision problem Π_{BH}. In the case of Π_{BH} this is done by incorporating i into the input to a slightly modified machine and increasing the step count. On input (x,i) the modified machine uses the first choice in its i-th step (instead of performing a non-deterministic choice).

Combining the constructions of Theorems 1 and 2, we have found for every distributional search problem Π_s a related distributional decision problem Π_d such that Π_s is randomly reducible to Π_d.

Open Problem 1 : Can every distributional search problem (in DistNP) be *deterministically* reduced to some distributional decision problem (in DistNP)?

4 On the Structure of DistNP

[Ladner 75] has demonstrated the richness of the structure of NP under polynomial reductions. The average case complexity counterpart, DistNP, is not less complex. There are several different ways to define a (average-polynomial) "reducibility order" on this class and they all enjoy structure theorems analogous to those of [Ladner 75]. We present here only some characteristic examples.

4.1 Structure of Decisions

The natural way to define a complexity ordering on the class DistNP is through the reducibility of distributional problems. Namely, $\Pi_1 \leq \Pi_2$ if $\Pi_1 \propto \Pi_2$ (and $\Pi_1 < \Pi_2$ if $\Pi_1 \leq \Pi_2$ but not vice versa). Another possibility is to fix a P-computable distribution μ and consider the relation $\leq_\mu$ defined on NP by $D_1 \leq_\mu D_2$ iff $(D_1, \mu) \leq (D_2, \mu)$. Intuitively, $D_1 \leq_\mu D_2$ means that D_1 is not harder than D_2 with respect to the instance distribution μ. Theorem 3 implies that for every $\mu \in$ P-computable having a NP problem which is not easy on the average with respect to μ, there is an infinite hierarchy on NP with respect to $\leq_\mu$. It should be noted that in general the order $\leq_\mu$ does not coincide with the order ($\leq_{\mathrm{P}}$) induced on NP by the usual polynomial-time reductions; that is, $D_1, D_2 \in$ NP and $D_1 \leq_{\mathrm{P}} D_2$ does not imply $D_1 \leq_\mu D_2$ for a specific or for any $\mu \in$ P-computable.

Theorem 3 : For every computable distribution μ and every computable language D_1 such that (D_1, μ) is not in Average-P, there exists a problem (D_2, μ) such that (D_2, μ) is also not in Average-P, (D_1, μ) is not (Turing) reducible to (D_2, μ), and (D_2, μ) is (many-to-one) reducible to (D_1, μ) (i.e., $D <_\mu D_2 <_\mu D_1$, for every $D \in$ P). Furthermore, D_2 is logSpace-reducible (i.e., worst case reducible) to D_1 (hence, if $D_1 \in$ NP then so is D_2).

Proof : Similar to the proof of Theorem 4 (below), except that we use $\bar{\sigma}$ to modify D instead of μ. $\square$

Corollary 1 : If DistNP is not contained in Average-P then there exists a problem (D, μ) in DistNP which is neither complete for DistNP nor easy on the average.

Proof : Apply Theorem 3 to any DistNP-complete problem. $\square$

4.2 Structure of Distributions

There is yet another natural order which emerges in the context of distributional problems: an ordering of distributions. We define $\mu_1 \leq_D \mu_2$ if $(D, \mu_1) \leq (D, \mu_2)$. Intuitively, $\mu_1 \leq_D \mu_2$ means that the problem D is not harder on input distribution μ_1 than on distribution μ_2.

Theorem 4 : For every computable distribution μ_1 and every computable language D such that (D, μ_1) is not in Average-P, there exists a distribution μ_2 which is *log-space reducible* to μ_1 such that (D, μ_2) is not in Average-P, (D, μ_1) is not (Turing) reducible to (D, μ_2), and (D, μ_2) is (many-to-one) reducible to (D, μ_1) (i.e. $\mu_{fin} <_D \mu_2 <_D \mu_1$, for each distribution μ_{fin} giving 0 weight to all but finitely many strings). (We say that μ_2 is *log-space reducible* to μ_1 if μ_2 can be computed by a log-space oracle machine with access to an oracle for μ_1.) Furthermore, for every computable language D', $(D, \mu_1) \not\propto^{\mathrm{T}} (D', \mu_2)$.

Proof : The basic idea behind the proof is adapted from [Ladner 75] (which, in turn, is modeled after similar results in recursion theory). One constructs the desired infinite object (a language or, in our case, a distribution) in stages. In each stage a finite segment is added, the final object is the (infinite) union of the segments defined along the construction. The construction is governed by a sequence $< S_i : i \in N >$ of demands that have to be met. The demands should be such that:

- Given any finite initial segment σ of the construction and any demand S_i, there exists a finite extension of σ that meets the demand.
- Every construction, along which all the demands are met, gives rise to an object of the desired type (e.g., a μ_2 as required for the theorem).

Here we construct an infinite sequence $\bar{\sigma}$ of 0's and 1's. We shall define μ_2 by letting $\mu_2'(x)$ be 0 if $\bar{\sigma}(|x|) = 0$ and $\mu_2'(x) = \mu_1'(x)$ otherwise. Hence,

$$\mu_2(x) = \sigma(|x|) \cdot (\mu_1(x) - \mu_1(1^{(|x|-1)})) + \sum_{i<|x|} \sigma(i) \cdot (\mu_1(1^i) - \mu_1(0^i))$$

yielding a log-space reduction of the computation of μ_2 to the computations of μ_1 and σ. Let us describe our sequence $< S_i : i \in N >$ of demands.
The demands: Essentially we have two types of demands. Demands for 'sparseness' of μ_2 which make sure that for every D' the original pair (D, μ_1) is not reducible to (D', μ_2), and 'denseness' demands to guarantee that (D, μ_2) is not easy on the average. (We get the reducibility of (D, μ_2) to (D, μ_1) for free, as any μ_2 defined as above is dominated by μ_1).

For the 'denseness' demands we enumerate all Average-P machines and make sure that none solves (D, μ_2). Namely, we list all triples $< A_i, c_i, \epsilon_i >$ such that A_i is a Turing machine, c_i is an integer and ϵ_i is a rational in the unit interval. Formally, a finite string σ *satisfies* the triple $< A_i, c_i, \epsilon_i >$ if either A_i errs on some input x of length $\leq |\sigma|$ (i.e., $A_i(x) \neq D(x)$) or A_i runs too slow on the average (i.e., $\sum_{|x| \leq |\sigma|} \mu_2'(x) \cdot \frac{t_i(x)^{\epsilon_i}}{|x|} > c_i$, where $t_i(x)$ is the running time of A_i on input x).

For the 'sparseness' demands we enumerate all Average-P oracle machines and make sure that none can reduce (D, μ_1) to (D, μ_2) (or to (D', μ_2) for any D'). Namely, we list all quadruples $< D_i, M_i, c_i, \epsilon_i >$ such that D_i is a (code of a Turing machine for a) recursive language, M_i is an oracle machine, and c_i, ϵ_i are as above. The demands $< D', M', C_i, \epsilon_i >$ should make sure that M' cannot be used to reduce (D, μ_1) to (D', μ_2). Consequently, we let a finite string σ meet such a demand if either, for some x (with $|x| < |\sigma|$), the reduction fails (i.e., $M_i^{D_i}(x) \neq D(x)$) or the reduction runs for too much time (i.e., $\sum_{|x| \leq |\sigma|} \mu_1'(x) \cdot \frac{t_i(x)^{\epsilon_i}}{|x|} > c_i$, where $t_i(x)$ is the running time of $M_i^{D_i}$ on input x) or M_i violates the domination condition by making (on input x s.t. $|x| < |\sigma|$ and $\mu_1'(x) \neq 0$) a query y such that $|y| < |\sigma|$ and $\mu_2'(y) = 0$.

Our sequence $< S_i : i \in N >$ of demands will be any recursive enumeration of all the demands described above, in which odd i's carry 'sparseness' demands and even i's constitute 'denseness' demands.

Lemma 1 : Given any finite initial segment σ of the construction and any demand S_i, there exists a finite extension of σ that meets the demand.

Proof : To handle a 'denseness' demand $S_i =< A_i, c_i, \epsilon_i >$ we make μ_2 dense enough by extending σ by a tail of 1's. If no such finite extension meets S_i, then A_i computes D correctly in time polynomial on the average with respect to a distribution μ_2 which differs from μ_1 (possibly) only on strings of length $\leq |\sigma|$. It follows that (D, μ_1) is in Average-P contradicting the hypothesis of the theorem.

Given an 'sparseness' demand $S_i =< D_i, M_i, c_i, \epsilon_i >$, we make μ_2 sparse enough by extending σ by a tail of 0's. If no such extension meets S_i, then (as $\mu_2'(y) = 0$ whenever $\bar{\sigma}(|y|) = 0$) the domination constraint guarantees that M_i does not ask its oracle queries of length $> |\sigma|$. It follows that M_i computes D in average polynomial time (with respect to μ_1) with the help of a finite oracle (i.e., the first $2^{|\sigma|+1}$ bits of D_i) therefore (D, μ_1) is in Average-P, a contradiction.$\diamond$

Clearly, the infinite sequence $\bar{\sigma}$ (constructed as above) satisfies all the demands. To complete the proof of the theorem, we need to provide an efficient procedure for constructing $\bar{\sigma}$. It suffices to show that $\sigma(k)$ can be computed in $k^{O(1)}$ steps by an oracle machine given oracle μ_1. To this end, we employ an observation of [Ladner 75]: meeting a 'denseness' (respectively, 'sparseness') demand requires extending the current σ by sufficiently many $1'$s (respectively, $0'$s), yet adding too many $1'$s ($0'$s) will not cause any harm. Consequently, when faced with such a demand, we can keep defining $\bar{\sigma}(k) = 1$ (or 0) until k is big enough to allow sufficient computing time for realizing that the demand has been met.

Lemma 2 : Let f be any unbounded, non-decreasing and time-constructible function. A sequence $\bar{\sigma}$ as in Lemma 1 can be constructed by an oracle machine which, given access to μ_1, computes $\sigma(k)$ in $\leq f(k)$ steps.

Proof : On input k, we compute $\sigma(k)$ by conducting the following procedure with time bound $f(k)$. Starting with $i = 1$ and $j = 1$, the procedure tries to see if demand S_i is satisfied by the j-bit long prefix of $\bar{\sigma}$ (in which case i is incremented). Let $P_{i,j}$ denote the procedure's computation regarding the pair (i, j). The computation of $P_{i,j}$ starts by setting $\sigma(j)$ to 0 if i is odd and $\sigma(j) = 1$ otherwise. These stored values may be used in computations of the procedure on larger j's. If ($P_{i,j}$ terminates and) it is found that S_i is indeed satisfied by the j-long prefix then both i and j are incremented. If ($P_{i,j}$ terminates and) it is found that S_i is not satisfied by the j-long prefix then only j is incremented. Eventually, the procedure (either terminates or) is stopped by the time-out mechanism (i.e. after $f(k)$ steps) and $\sigma(k)$ is set to 0 if the current i is odd and to 1 otherwise.

In its computation, the above procedure uses oracle calls to μ_1 as well as previously computed values of σ. (The values of σ are used to compute μ_2.) The procedure also computes the values of D and D_i (for some i's) on various strings, and runs various regular and oracle machines on various input strings while counting of the number of steps these machine make.

A detailed description of $P_{i,j}$ follows. If i is even then the procedure tries to satisfy the denseness demand $S_i =< A_i, c_i, \epsilon_i >$. It runs A_i on all strings of length j, computes D on all these strings, compares the values obtained, computes the sum $\sum_{|x| \leq j} \mu_2'(x) \cdot \frac{t_i(x)^{\epsilon_i}}{|x|}$ and compares it to c_i (to this end the procedure uses calls to μ_1 and the stored values of $\sigma(1), ..., \sigma(j-1)$). Thus, the procedure easily determines if S_i was satisfied by the strings of length $\leq j$. If i is odd then the procedure tries to satisfy the sparseness demand $S_i =< D_i, M_i, c_i, \epsilon_i >$. It runs the oracle machine M_i on all strings of length *less than* j answering the oracle calls of length less than j by computing the value of D_i, computes the density of μ_1 on these inputs and compares it to the density of μ_2 on the queries (to this end the procedure uses calls to μ_1 and the stored values of $\sigma(1), ..., \sigma(j-1)$). We stress that in case oracle calls were made to strings of length $\geq j$ the demand S_i is not satisfied by the strings of length $< j$. The procedure also computes D on all strings of length $< j$, compares the values to those obtained from the oracle machine, computes the sum $\sum_{|x|<j} \mu_2'(x) \cdot \frac{t_i(x)^{\epsilon_i}}{|x|}$ and compares it to c_i.

Clearly, σ is constructed in the number of steps claimed. The reader may verify that σ defined by the above procedure satisfies all demands. $\diamond$

The theorem follows by noting that having access to an oracle for μ_1 we can, on an input x, compute $\mu_2(x)$ in log-space (hint: $\mu_2(x) = \sum_{i<|x|} \sigma(i)\cdot(\mu_1(1^i) - \mu_1(0^i)) + \sigma(|x|)\cdot(\mu_1(x) - \mu_1(1^{(|x|-1)}))$). $\square$

Corollary 2 : For every distribution $(D, \mu_1) \in$DistNP$-$Average-P there exists a distribution μ_2 such that $(D, \mu_2) \in$DistNP$-$Average-P and $(D, \mu_2) < (D, \mu_1)$. Furthermore, for every $D' \in$ NP, $(D, \mu_1) \not\propto^{\mathrm{T}} (D', \mu_2)$.

5 Relations to Worst-Case Complexity

In this section we present theorems which relate questions about average-case complexity classes to questions about worst-case complexity classes.

5.1 On Average-P vs. DistNP

The first question we address is how likely is it that every NP problem has an easy on the average solution with respect to every P-computable distribution.

Theorem 5 : If NTime($2^{\mathrm{O(n)}}$) $\neq$ DTime($2^{\mathrm{O(n)}}$) then DistNP is not a subset of Average-P.

The theorem follows easily from the following

Proposition 1 : NTime($2^{\mathrm{O(n)}}$) $\neq$ DTime($2^{\mathrm{O(n)}}$) if and only if there exists a unary language $L \in$ NP such that $(L, \mu_1) \notin$Average-P, where $\mu_1'(1^n) = n^{-2}$.

Proof : Book [B] proved that NTime($2^{\mathrm{O(n)}}$) $\neq$ DTime($2^{\mathrm{O(n)}}$) iff there exists a unary language which is in NP but not in P. Membership in a unary language is decidable in polynomial time iff there is an algorithm which decides membership in L and has running time polynomial on average with respect to the distribution on unary strings $\mu'(1^n) = n^{-2}$. (Hint: $\exists \epsilon > 0$ s.t. $\sum_n \frac{1}{n^2} \cdot t(n)^\epsilon < \infty$ implies $t(n) < n^{2/\epsilon}$ for all but finitely many n's.) $\square$

5.2 On Average-P vs. NP with Arbitrary Distributions

In this subsection we show that if no restrictions are placed on the distributions then average-case complexity classes take the form of the traditional worst-case complexity classes. For example.

Theorem 6 : If P $\neq$ NP then there exist $D \in$ NP and an exponential-time computable μ such that $(D, \mu) \notin$Average-P.

Theorem 6 is derived as a special case of Proposition 2 (below), by using the fact that P $\neq$ NP implies the existence of a problem $D \in$ (NP $-$ P) $\cap$ DTime(2^n) (hint: padding).

Definition 2 : Let $f : \mathbf{N} \mapsto \mathbf{N}$. AverDTime($f(n)$) denotes the class of distributional problems (D, μ) solvable by a deterministic algorithm A with running time function $t_A(x) \leq f(l(x))$, where $l : \{0,1\}^* \mapsto \mathbf{N}$ is linear on the average with respect to μ (i.e., $\sum_x \mu'(x)\frac{l(x)}{|x|} < \infty$).

Clearly, AverDTime($n^{O(1)}$) equals Average-P.

A special case of Definition 2 is when the function $l : \{0,1\}^* \mapsto \mathbf{N}$ is linear everywhere (i.e. $l(x) = O(|x|)$). We would like to have such D be in DTime($f(n)$). To this end, we redefine DTime($f(n)$) to be the class of problems D solvable by a deterministic algorithm A with running time function $t_A(x) \leq f(O(|x|))$ (instead of $t_A(x) \leq f(|x|)$ as is standard practice).

Proposition 2 : Let $f, g : \mathbf{N} \mapsto \mathbf{N}$ be any two monotone and time-constructible functions such that for every $c > 1$ and all sufficiently large n's: $g(n) > f(c \cdot n)$. Let $D \in \text{DTime}(g(n))$. Then there exists an μ, computable in time $2^n \cdot g(O(n))$, so that $D \in \text{DTime}(f(n))$ if and only if $(D, \mu) \in \text{AverDTime}(f(n))$.

Proof : Clearly, $D \in \text{DTime}(f(n))$ implies $(D, \mu) \in \text{AverDTime}(f(n))$ for every μ. For the other direction assume that $D \in (\text{DTime}(g(n)) - \text{DTime}(f(n)))$. Consider an enumeration $M_1, M_2, \ldots$ of deterministic machines.

For every such machine, M_i, define a distribution μ_i as follows. If M_i errs on some input, then let x_1 denote such an input. Else, let $x_1, x_2, \ldots$ be an infinite sequence such that x_j is the first input for which both $f(j^3|x_j|) < g(|x_j|)$ and machine M_i makes more than $f(j^3|x_j|)$ steps on x_j. (Such an infinite sequence must exist since $f(c^3 n) < g(n)$ for every c and all sufficiently large n, and since machine M_i cannot answer correctly on all sufficiently long x's in $f(c^3|x|)$ steps for some constant c.) Let $\mu_i'(y)$ be $\frac{1}{j^2}$ if $y = x_j$ and 0 otherwise. It follows that M_i does not solve D in average time $f(n)$ with respect to the distribution μ_i. (Otherwise, we get $t_i(x_j) \leq f(l_i(x_j))$ where $l_i(x_j) \leq j^2|x_j|$ for all but finitely many j's; and it follows that $t_i(x_j) \leq f(j^2|x_j|)$ in contradiction to the definition of the x_j's.)

Let $\mu(x) = \sum_{i \leq |x|} \frac{1}{i^2} \cdot \mu_i(x)$. The distribution function μ is computed, on input x, by running each M_i ($i \leq i \leq |x|$) on all strings $y \leq x$, counting the number of steps and comparing the output of $M_i(y)$ with the value of $D(y)$. Each M_i is run with time bound $g(|x|)$, and D is computed in $g(O(|x|))$ steps. One can easily verify that $(D, \mu) \notin \text{AverDTime}(f(n))$ and that $\mu(x)$ is computable in time $O(2^{|x|} \cdot g(O(|x|)))$. $\Box$

6 Polynomially Samplable Distributions

In this section we consider a natural extension of the class of P-computable distributions – the class of P-samplable distributions. Distributions in this class arise in probabilistic polynomial time computations. We show that under "modest" cryptographic assumptions, there are P-samplable distributions that are "very far" from any P-computable distribution. We proceed to present a complete distributional problem in $\langle$NP, P-samplable$\rangle$. The proof of completeness here is very different than for the $\langle$NP, P-computable$\rangle$ case.

Definition 3 *(The class* P-samplable*)* : A distribution μ is in the class P-samplable if there exists a polynomial P and a probabilistic algorithm A that outputs the string x with probability $\mu'(x)$ within $P(|x|)$ steps.

Elements in a P-samplable distribution are generated in time polynomial in their length.

Theorem 7 : Every P-computable distribution is also P-samplable.

Proof : The sampling algorithm A picks at random with uniform distribution a truncated real ρ in $[0, 1]$ (the length of expansion depends on the following search). It then finds, via binary search, and queries to μ, the unique string $x \in \{0,1\}^*$ satisfying $\mu(x-1) < \rho \leq \mu(x)$. $\Box$

The above proof associates every P-computable distribution with a standard sampling algorithm. With respect to this standard algorithm the coin tosses used in the generation of an instance can be reconstructed from the instance itself (i.e., given x one can efficiently generate a random

ρ such that A on coin tosses ρ outputs x). It follows that solving a problem on instances generated by a P-computable distribution does not become easier if we are given the coin tosses used in the construction of the instance to the problem. Hence instances of distributional problems in DistNP$-$Average-P are hard also for the person generating them.

The converse of Theorem 7 (above) is unlikely. Namely,

Theorem 8 : If one-way functions exists, then there is a P-samplable distribution μ which is not dominated by any distribution η with a polynomial-time computable η'. (Loosely speaking, a one-way function is a polynomial-time computable function which is "hard to invert" on most instances. See, for example, [Hastad 90].)

In particular, μ itself is not P-computable and no P-computable distribution dominates it. An alternative proof of this corollary (under the same condition) is now available using [Levin 88] and [Hastad 89]. Using a weaker condition, namely P $\neq$ NP, one can prove that $\mathrm{P}-\mathrm{samplable} \neq \mathrm{P}-\mathrm{computable}$ (hint: use the construction of [Gurevich and McCauley 87] of a distribution with polynomial-time computable density function which is not P-computable itself).

Proof Sketch: Assume first that there exists a length preserving one-way 1-1 function f (i.e., $|f(x)| = |x|$ and $f(x) = f(y)$ implies $x = y$). Define the distribution μ on $\{0,1\}^*$ by

$$\mu'(y,z) = \begin{cases} \frac{1}{n^3 \cdot 2^n} & \text{if } |y| = n,\ \exists x \text{ s.t. } y = f(x), \text{ and } z \text{ is a prefix of } x \\ 0 & \text{otherwise} \end{cases}$$

It is easy to see that μ is P-samplable: first select n with probability $\frac{1}{n^2}$, next choose x uniformly in $\{0,1\}^n$, and output (y,z), where $y \leftarrow f(x)$ and z is a prefix of x with randomly selected length.

Suppose η is a P-computable distribution dominating μ (i.e., $\exists c \forall x$: $\eta'(x) > \frac{1}{|x|^c} \cdot \mu'(x)$). For most y's of length n, there could be only polynomially many w's such that $\eta'(y,w) \geq \frac{1}{n^{c+3} \cdot 2^n}$ (otherwise η would sum up to more than 1). We call such a *w heavy for y*, and we say that y is *typical* if it has only polynomially many heavy w's. Hence, most y's are typical. Given a typical y, an oracle for η' can be used to find $f^{-1}(y)$ in polynomial time, by searching the "prefixes-tree" proned at non-heavy nodes. Following is a program for the oracle machine.

On input y, the machine initiates $z \leftarrow 0$ and proceeds in stages. In every stage the machine computes $\eta'(x,z)$. If $\eta'(y,z) \geq \frac{1}{|y|^{c+3} 2^{-|y|}}$ then the machine sets $z \leftarrow z0$. Else letting $z = z'01^k$ (with $k \geq 0$), the machine sets $z \leftarrow z'1$. If $f(z) = y$ the machine outputs z and stops, else the machine proceeds to the next stage.

The reader can easily verify that on input a typical y the machine runs for polynomial-time, and that for every z prefix of $f^{-1}(y)$ domination implies that $\eta'(y,z) \geq \frac{1}{|y|^{c+3} 2^{-|y|}}$. It follows that the machine inverts f on most inputs in polynomial-time, a contradiction to the one-way hypothesis.

We now outline the ideas required to extend the above argument to the case in which "only" an arbitrary one-way function is guaranteed to exist. First, we note that the above argument can be easily extended to the case where the function is length preserving, and one-way 1-1 on $\frac{1}{2n}$ of the instances of length n. This condition is hereafter reffered as condition (*). Next, we transform any one-way function into one satisfying condition (*). The transformation is carried out in two stages. Given an arbitrary one-way function f_0, we can easily construct a length preserving one-way function f_1 (e.g., let $f_1(x'x'') = f_0(x')01^{n-1-|f_0(x'x'')|}$, where $n = |x'x''|$ and $|x'| = n^\epsilon$ where ϵ is chosen so that $f_0(z)$ is computed in $< |z|^{1/\epsilon}$ steps). In the second stage, we use ideas of [Impagliazzo, Levin and Luby 89] to construct a function f_2 which satisfies condition (*). Define $f_2(x,k,h) = f_1(x), k, h, (h(x)\downarrow_k)$, where $n = |x|$, $h \in H_{n,n}$ is a universal hashing function and $\alpha \downarrow_k$

denotes the first k bits of α. Certainly, f_2 is one-way and 1-1 on most triples (x,k,h) satisfying $k = \lceil \log_2 |f^{-1}(f(x))| \rceil$, and hence satisfies condition (*). The theorem follows. $\Box$

The above theorem motivates the definition of the following class of distribution problems.

Definition 4 (The class $\langle$NP, P-samplable$\rangle$): A problem (D,μ) belongs to the class $\langle$NP, P-samplable$\rangle$ if D is an NP-predicate and μ is in P-samplable.

Intuitively, the question of whether there exists a problem in $\langle$NP, P-samplable$\rangle$ which is not in Average-P can be interpreted as asking whether one party can efficiently find instances of an NP problem which will be hard to solve for another party. This should be contrasted with the question of whether there exists a problem in $\langle$NP, P-computable$\rangle$ which is not in Average-P, a question which can be interpreted (see discussion proceeding Theorem 7 above) as asking whether one can efficiently find instances of an NP problem which will be hard to solve, even for the person who has constructed them!

A complexity class related to $\langle$NP, P-samplable$\rangle$ has been defined and investigated (independently of our work) by [Hemachandra, Allender, Feigenbaum, Abadi and Broder 90]. They consider probabilistic polynomial-time algorithms which generate yes-instances of an NP problem together with a corresponding solution. The instance (without the corresponding solution, of course) is fed to an arbitrary polynomial-time algorithm (hereafter referred to as a *solver*) which is required to find a solution. The pair (i.e., NP together with an yes-instance generator) is called *invulnerable* if for every (polynomial-time) solver there exists a constant α and infinite set of integers S such that for every $n \in S$ the probability that the solver succeeds on length n is $< 1 - \alpha$ (the probability is taken over the internal coin tosses of both algorithms). The pair is called *almost-everywhere invulnerable* if the above holds for a set S containing all but a finite number of the integers. Reductions among pairs are defined so that the invulnerability of the original pair implies the invulnerability of the reduced pair. Strong reductions are defined so that the above holds with respect to "almost-everywhere invulnerability". Using techniques similar to those of the following proof, Hemachandra et. al. demonstrated the existence of a complete problem (with respect to regular reductions). The result has been augmented to strong reductions by [Feigenbaum, Lipton and Mahaney 89].

Theorem 9 : There exist problems which are complete in $\langle$NP, P-samplable$\rangle$.

Proof (following ideas implicit in [Levin 85]): The proof is based on the observation that it is possible to effectively "enumerate" all sampling algorithms which work in quadratic time (in length of their output). Two remarks are in place. Firstly, the reader may find the term "enumeration" misleading. We don't really enumerate all machines running in quadratic time (such an enumeration is not possible as indicated by [Gurevich, Shelah 89]). Instead, we enumerate all Turing machines and modify each of them to stop in time quadratic in the length of the output. This is obtained by augmenting the machine so that it pads its output once entering the original halting state. The padding is long enough to make the modified machine run in time quadratic in its output. It is important to note that the output of machines, which originally run in time quadratic in their output, remains unchanged. Secondly, the reader may wonder why it suffices to consider only distributions with quadratic running time (instead of arbitrary polynomial). The reason is that every problem (D,μ) in $\langle$NP, P-samplable$\rangle$ can be reduced (by padding) to a problem (D',η) (in $\langle$NP, P-samplable$\rangle$) where η is a distribution which can be sampled by an algorithm running quadratic time (in the length of its input). In particular suppose that there exists a polynomial, P, and a sampling algorithm for μ such that every x is output within $P(|x|)$ steps. Then a sampling algorithm for η pads each output x by $P(|x|) - |x|$ zeros, and D' ignores this padding (i.e. $D'(y) = D(y')$, where $y = y'y''$ and $P(|y'| - 1) < |y| \le P(|y'|)$).

Let $\eta_1, \eta_2, \ldots$ be an enumeration of the distributions generated by (modified) sampling machines running in time quadratic in their output. Define a *universal distribution*

$$\eta'_U(x) \overset{\text{def}}{=} \sum_{i=1}^{\infty} \frac{\eta'_i(x)}{i^2} \, .$$

Clearly, η_U is P-samplable (e.g., first select i with probability $\frac{1}{i^2}$ and next sample η_i). We now show that the distributional problem (BH, η_U) is complete in $\langle$NP, P-samplable$\rangle$. First, we remind the reader that any problem in $\langle$NP, P-samplable$\rangle$ can be reduced to a distribution with a quadratic time sampling algorithm. Hence it suffices to deal with these distibutions. Let (D, μ) be an arbitrary problem in $\langle$NP, P-samplable$\rangle$, where μ is computable in quadratic time, and let S be the corresponding sampling algorithm. Let f be the standard Karp reduction of D to BH (i.e., the reduction which maps an instance x of D to a triple $(M, x, 1^k)$, where M is an NP machine for D and k is a bound on the running time of M on strings of length $|x|$). Note that $|f(x)| > 2|x|$ and that f is computable in quadratic time. Thus, applying f to the output of S yields a sampling algorithm, S_j, which runs in quadratic time and hence samples one of the distributions in the enumeration $\eta_1, \eta_2 \ldots$. Namely, for some j, we have $\eta'_j(f(x)) = \mu'(x)$. Clearly, $(D, \mu) \propto (BH, \eta_j) \propto (BH, \eta_U)$. $\Box$

Remarks:

- All natural NP-complete problems have a distribution so that they are complete in $\langle$NP, P-samplable$\rangle$ with that distribution. A sufficient condition is that the Karp reduction of BH to the NP-complete problem, D, does not decrease the length by too much. Namely, let g be a Karp reduction of BH to D so that $\exists \epsilon > 0$ such that $\forall x$: $|f(x)| \ge |x|^{\epsilon}$. Let η_g be the distribution induced by sampling η_U and applying g on its output. Then, η_g is P-samplable and $(BH, \eta_U) \propto (D, \eta_g)$.

- The construction can be modified so that, provided one-way functions exist, the resulting problem is complete in $\langle$NP, P-samplable$\rangle$ but is not a member of $\langle$NP, P-computable$\rangle$.

- The proof of Theorem 9 depends heavily on the enumeration of P-samplable distributions. Such effective enumeration is not known for P-computable distributions.

7 Average logspace

The "natural" adaptation of the definition of Average-P fails for Average-logspace. We present an alternative definition of Average-logspace, which satisfies some desired properties, such as Average-logspace $\subseteq$ Average-P. We define the class $\langle$P, logspace-computable$\rangle$, and give an appropriate version of the bounded halting problem, together with a distribution in logspace-computable, which are shown to be complete in $\langle$P, logspace-computable$\rangle$ with respect to logspace reductions.

The first attempt at the definition is the following: An algorithm A is logspace on the average with respect to distribution μ if

$$\sum_{x \in \{0,1\}^*} \mu'(x) \cdot \frac{s_A(x)}{\log |x|} < \infty$$

where $s_A(x)$ denotes the work space used by algorithm A on input x. Unfortunately, this definition has some serious handicaps, the most upsetting of which is that for every $0 < \alpha < 1$, algorithms that use work space n^α on every input of length n, will be in average logspace with respect to the uniform distribution. (As a consequence, average logspace will not necessarily be contained in average-P.) Instead, we propose the following definitions, suggested independently by [Levin 88].

Definition 5 *(Logarithmic on the Average)* : A function $f : \{0,1\}^* \rightarrow \mathbf{N}$ is *logarithmic on the average* with respect to a distribution μ if there exists a constant $\epsilon > 0$ such that

$$\sum_{x \in \{0,1\}^*} \mu'(x) \cdot \frac{(2^{f(x)})^\epsilon}{|x|} < \infty$$

Thus, a function is logarithmic on the average if it is bounded by a logarithm in a function which is linear on the average.

Definition 6 *(The class* Average-logspace*)* : A distributional problem (D, μ) is in the class Average-logspace if there exists an algorithm A solving D using work space s_A, which is logarithmic on the average with respect to the distribution μ.

This revised definition overcomes the above mentioned difficulties. In addition, the notion of domination of probability distributions will still be applicable, and Average-logspace is closed under average logspace (many-to-one) reductions.

This approach can be generalized to the definition of the class Average-Uniform-NC. To do this, we use the characterization of Uniform-NC by alternating logspace and poly-log time Turing machines [Ruzzo 81]. We now require that both the exponent of the work space (i.e. $2^{s_A(x)}$) and the exponent of the time to some power $\delta > 0$ (i.e. $2^{t_A(x)^\delta}$) be polynomial on the average.

The class $\langle$P, logspace-computable$\rangle$ is defined analogously to the definition of $\langle$NP, P-computable$\rangle$. Namely, $\langle$P, logspace-computable$\rangle$ consists of distributional problems, (D, μ), with $D \in$ P and the distribution function μ is logspace computable. It should be noticed that many natural distributions, including the uniform distribution and the distribution μ_{BH} (of subsection 2.3), are logspace computable.

Deterministic Bounded Halting (*DBH*) is defined over triples $(M, x, 1^k)$, where M is a deterministic machine, x is a binary string and k is an integer (given in unary). The problem is to determine whether M accepts x within k steps. Clearly, $DBH \in$ P, and it is not hard to see that it is P-complete with respect to logSpace reductions.

Theorem 10 : The distributional problem (DBH, μ_{BH}) is complete in $\langle$P, logspace-computable$\rangle$. (μ_{BH} is as defined in subsection 2.3.)

Proof Sketch: The proof uses ideas of the $\langle$NP, P-computable$\rangle$-completeness proof of Distributional Bounded Halting. This proof is presented in Appendix B. The heart of the proof is an encoding C_μ which is 1-1 and satisfies $|C_\mu(x)| \leq O(1) - \log_2 \mu'(x)$. If $\mu'(x) \leq 2^{-|x|}$ then $C_\mu(x) = 0x$ else $C_\mu(x) = 1z$ where z is a shortest binary expansion of a real in the interval $(\mu(x-1) - \mu(x)]$. Clearly, the computation of C_μ essentially reduces to computing μ on two values. Thus, when μ is P-computable the encoding C_μ is polynomial-time computable. As here μ is logspace computable, so is the encoding function C_μ used in the reduction. The decoding algorithm C_μ^{-1} is not logspace computable. However, decoding can be done in deterministic polynomial time (by binary search), which is sufficient for our purpose. $\Box$

We remark that it is possible to define a version of the tiling problem that with a natural distribution constitute a complete distributional problem in $\langle$NP, P-computable$\rangle$. The input to a tiling problem is an alphabet Σ, a family of tiles $F \subseteq \Sigma^4$, an initial tiling $\alpha \in F^*$, and an integer k presented in unary. The question in the standard version is whether the initial tiling α can be extended to a tiling of the $k \times k$ square. Our version of tiling is restricted to "forcing families". A family of tiles is called *forcing* if, for every row of tiles, the row of tiles which can be placed on top of it is unique except for the end tiles. Note that it suffices to consider rows of length 3, and thus families can be tested for the "forcing property" in logSpace.

We derive results analogous to those appearing in Section 5 (using essentially the same proof techniques). For example

Theorem 11 : If $\mathrm{DTime}(2^{\mathrm{O(n)}}) \neq \mathrm{DSpace(n)}$ then there exists a problem in $\langle$P, logspace-computable$\rangle$ which is not in Average-logspace.

All the results in Section 4, dealing with the structure of $\langle$NP, P-computable$\rangle$, can be modified to the context of $\langle$P, logspace-computable$\rangle$.

8 Concluding Remarks

In general, a theory of average case complexity should provide

- a specification of a broad class of *interesting* distributional problems;
- a definition capturing the subclass of (distributional) problems which are *easy* on the average;
- notions of reducibility which allow to infer the easiness of one (distributional) problem from the easiness of another;
- and, of course, results...

It seems that the theory of average case complexity, initiated by Levin and further developed in [Gurevich 87,Venkatesan and Levin 88] and here, satisfies these expectations to some extent. This should not discourage the reader from trying to suggest alternative definitions, or get convinced that we should stick to the ones presented above. In particular,

- generalize or provide a better alternative for the class DistNP (especially with respect to the condition imposed on the distribution function);
- try to provide a more natural (but yet as robust) definition of problems which are "polynomial-time on the average";
- and, naturally, try to find a real natural distributional problem which is complete in DistNP (e.g., subset sum with uniform distribution).

In addition to their central role in the theory of average-case complexity, reductions which preserve uniform (or very simple) instance distribution are of general interest. Such reductions, unlike most known reductions used in the theory of NP-completeness, have a range which is a non-negligible part of the set of all possible instances of the target problem (i.e. a part which cannot be claim to be only a "pathological subcase"). It is interesting to further study the corresponding reducibility relation.

Note added in proofs

Following the presentation of this work in conferences, [Impagliazzo, Levin 90] proved that every language which is $\langle$NP, P-computable$\rangle$-complete is also $\langle$NP, P-samplable$\rangle$-complete. This important result makes the theory of average case very robust. In particular, we believe that it eliminates the motivation for providing an alternative to DistNP=$\langle$NP, P-computable$\rangle$ (a suggestion made above). For further discussion the reader is referred to [Impagliazzo, Levin 90].

A distributional version of a natural problem from computational algebra has been recently shown to be DistNP-complete by [Gurevich 90]. Thus, DistNP-complete problems are known for the following areas: computability (e.g. Bounded-Halting), combinatorics (e.g. tiling and graph colouring), formal languages and algebra (e.g. of matrix groups).

Acknowledgements

We would like to thank Shimon Even, Mauricio Karchmer, Hugo Krawczyk, Ronny Roth, and Avi Wigderson for helpful discussions. We are grateful to Leonid Levin for very interesting discussions. Finaly, we wish to thank Yuri Gurevich for pointing out a misleading statement in an early version of the proof of Theorem 9.

References

[1] Carter, J., and M. Wegman, "Universal Classes of Hash Functions", *JCSS*, 1979, Vol. 18, pp. 143–154.

[2] Cook, S.A., "The Complexity of Theorem Proving Procedures", *Proc. 3rd ACM Symp. on Theory of Computing*, pp. 151–158, 1971.

[3] Feigenbaum, J., R.J. Lipton, and S.R. Mahaney, "A Completeness Theorem for Almost-Everywhere Invulnerable Generators", AT&T Bell Labs. Technical Memo., Feb. 1989.

[4] Garey, M.R., and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman and Company, New York, 1979.

[5] Goldreich, O., "Towards a Theory of Average Case Complexity (a survey)", TR-531, Computer Science Department, Technion, Haifa, Israel, March 1988.

[6] Gurevich, Y., "Complete and Incomplete Randomized NP Problems", *Proc. of the 28th IEEE Symp. on Foundation of Computer Science*, 1987, pp. 111–117.

[7] Gurevich, Y., "Matrix Decomposition Problem is Complete for the Average Case", *Proc. of the 31st IEEE Symp. on Foundation of Computer Science*, 1990, pp. 802-811.

[8] Gurevich, Y., and D. McCauley, "Average Case Complete Problems", preprint, 1987.

[9] Gurevich, Y., and S. Shelah, "Time polynomial in Input or Output", *Jour. of Symbolic Logic*, Vol. 54, No. 3, 1989, pp. 1083–1088.

[10] Hastad, J., "Pseudo-Random Generators with Uniform Assumptions", *Proc. 22nd ACM Symp. on Theory of Computing*, pp. 395–404, 1990.

[11] Hemachandra, L., E. Allender, J. Feigenbaum, M. Abadi and A. Broder, "On Generating Solved Instances of Computational Problems", *Advances in Cryptograghy - CRYPTO'88*, Lecture Notes in CS 403, S. Goldwasser (ed.), pp 297–310, Springer Verlag, 1990.

[12] Impagliazzo, R., and L.A. Levin, "No Better Ways to Generate Hard NP Instances than Picking Uniformly at Random", *Proc. of the 31st IEEE Symp. on Foundation of Computer Science*, 1990, pp. 812–821.

[13] Impagliazzo, R., L.A. Levin and M. Luby, "Pseudorandom Number Generation from any One-Way Function", *Proc. 21st ACM Symp. on Theory of Computing*, 1989, pp. 12–24.

[14] Johnson, D.S., "The NP-Complete Column—an ongoing guide", *Jour. of Algorithms*, 1984, Vol. 4, pp. 284–299.

[15] Karp, R.M., "Reducibility among Combinatorial Problems", *Complexity of Computer Computations*, R.E. Miller and J.W. Thatcher (eds.), Plenum Press, pp. 85–103, 1972.

[16] Karp, R.M., "Probabilistic Analysis of Algorithms", manuscript, 1986.

[17] Karp, R.M., E. Upfal, and A. Wigderson, "Are Search and Decision Problems Computationally Equivalent?", *Proc. 17th ACM Symp. on Theory of Computing*, 1985, pp. 464–475.

[18] Ladner, R.E., "On the Structure of Polynomial Time Reducibility", *Jour. of the ACM*, 22, 1975, pp. 155–171.

[19] Levin, L.A., "Universal Search Problems", *Problemy Peredaci Informacii 9*, pp. 115–116, 1973. Translated in *problems of Information Transmission 9*, pp. 265–266.

[20] Levin, L.A., "Average Case Complete Problems", *SIAM Jour. of Computing*, 1986, Vol. 15, pp. 285–286. Extended abstract appeared in *Proc. 16th ACM Symp. on Theory of Computing*, 1984, p. 465.

[21] Levin, L.A., "One-Way Function and Pseudorandom Generators", *Proc. 17th ACM Symp. on Theory of Computing*, 1985, pp. 363–365.

[22] Levin, L.A., "Homogeneous Measures and Polynomial Time Invariants", *Proc. 29th IEEE Symp. on Foundations of Computer Science*, 1988, pp. 36–41.

[23] Ruzzo, W.L., "On Uniform Circuit Complexity", *JCSS*, 22, 1981, pp. 365–385.

[24] Sipser, M., "A Complexity Theoretic Approach to Randomness", *Proc. 15th ACM Symp. on Theory of Computing*, 1983, pp. 330–335.

[25] Stockmeyer, L.J., "The Complexity of Approximate Counting", *Proc. 15th ACM Symp. on Theory of Computing*, 1983, pp. 118–126.

[26] Valiant, L.G., and V.V. Vazirani, "NP is as Easy as Detecting Unique Solutions", *Proc. 17th ACM Symp. on Theory of Computing*, 1985, pp. 458–463.

[27] Venkatesan, R., and L.A. Levin, "Random Instances of a Graph Coloring Problem are Hard", *Proc. 20th ACM Symp. on Theory of Computing*, 1988, pp. 217–222.

APPENDICES

Appendix A: Failure of a naive formulation of average polynomial-time

When asked to motivate his definition of average polynomial-time, Leonid Levin replies, non-deterministically, in one of the following three ways:

- "This is *the* natural definition".
- "This definition is *not important* for the results in my paper; only the definitions of reduction and completeness matter (and also they can be modified in many ways preserving the results)".
- "Any definition which *makes sense* is either equivalent or weaker".

For further elaboration on the first argument the reader is referred to Leonid Levin. The second argument is, off course, technically correct but unsatisfactory. We will need a definition of "easy on the average" when motivating the notion of a reduction and developing useful relaxations of it. The third argument is a thesis which should be interpreted along Wittgenstein's suggestion to the teacher: "say nothing and restrict yourself to pointing out errors in the students' attempts to say something". We will follow this line here by arguing that the definition which seems natural to an average computer scientist suffers from serious problems and should be rejected.

Definition X (naive formulation of the notion of easy on the average): A distributional problem (D, μ) is polynomial-time on the average if there exists an algorithm A solving D (i.e. on input x outputs $D(x)$) such that the running time of algorithm A, denoted t_A, satisfies $\exists c > 0 \forall n$:

$$\sum_{x \in \{0,1\}^n} \mu'_n(x) \cdot t_A(x) < n^c$$

where $\mu'_n(x)$ is the conditional probability that x occurs given that an n-bit string occurs (i.e., $\mu'_n(x) = \frac{\mu'(x)}{\sum_{y \in \{0,1\}^n} \mu'(y)}$).

The problem which we consider to be most upsetting is that Definition X is not robust under functional composition of algorithms. Namely, if the distributional problem A can be solved in average polynomial-time given access to an oracle for B, and problem B can be solved in polynomial-time then it does **not** follow that the distributional problem A can be solved in average polynomial-time. For example, consider uniform probability distribution on inputs of each length and an oracle Turing machine M which given access to oracle B solves A. Suppose that M^B runs $2^{\frac{n}{2}}$ steps on $2^{\frac{n}{2}}$ of the inputs of length n, and n^2 steps on all other inputs of length n; and furthermore that M when making t steps asks a single query of length $\sqrt{t}$. (Note that machine M, given access to oracle for B, is polynomial-time on the average.) Finally, suppose that the algorithm for B has cubic running-time. The reader can now verify that although M given access to the oracle B is polynomial-time on the average, combining M with the cubic running-time algorithm for B *does not* yield an algorithm which is polynomial-time on the average according to Definition X. It is easy to see that this problem does not arise when using the definition presented in Section 2.

The source of the above problem with Definition X is the fact that the underlying definition of polynomial-on-the-average is not closed under application of polynomials. Namely, if $t : \{0,1\}^* \to \mathbf{N}$ is polynomial on the average, with respect to some distribution, it does not follow that also $t^2(\cdot)$ is polynomial on the average (with respect to the same distribution). This technical problem is also the source of the following problem, that Levin considers most upsetting: Definition X is *not* machine independent. This is the case since some of the simulations of one computational model on another square the running time (e.g., the simulation of two-tape Turing machines on a one-tape Turing machine or the simulation of a RAM (Random Access Machine) on a Turing machine).

Another two problems with Definition X have to do with the fact that it deals separately with inputs of different length. The first problem is that Definition X is very dependent on the particular encoding of the problem instance. Consider, for example, a problem on simple undirected graphs for which there exist an algorithm A with running time $t_A(G) = f(n, m)$, where n is the number of vertices in G and m is the number of edges (in G). Suppose that if $m < n^{\frac{3}{2}}$ then $f(n, m) = 2^n$ and else $f(n, m) = n^2$. Consider the distributional problem which consists of the above graph problem with the uniform probability distribution on all graphs with the same number of vertices. Now, if the graph is given by its (incident) matrix representation then Definition X implies that A solves the problem in average polynomial-time (the average is taken on all graphs with n nodes). On the other hand, if the graphs are represented by their adjacency lists then the modified algorithm A (which transforms the graphs to matrix representation and applies algorithm A) is judged by Definition X to be non-polynomial on the average (here the average is taken over all graphs of m edges). This of course will not happen when working with the definition presented in Section 2. The second problem with dealing separately with different input lengths is that it does not allow one to disregard inputs of a particular length. Consider for example a problem for which we are only interested in the running-time on inputs of odd length.

After pointing out several weaknesses of Definition X, let us also doubt its "clear intuitive advantage" over the definition presented in Section 2. Definition X is derived from the formulation of worst case polynomial-time algorithms which requires that $\exists c > 0 \forall n$:

$$\forall x \in \{0,1\}^n : t_A(x) < n^c$$

Definition X was derived by applying the expectation operator to the above inequality. But why not make a very simple algebraic manipulation of the inequality before applying the expectation operator? How about taking the c-th root of both sides and dividing by n; this yields $\exists c > 0 \forall n$:

$$\forall x \in \{0,1\}^n : \frac{t_A(x)^{\frac{1}{c}}}{n} < 1$$

Applying the expectation operator to the above inequality leads to the definition presented in Section 2... We believe that this definition demonstrates a better understanding of the nature of the expectation operator!

Robustness under functional composition as well as machine independence seems to be essential for a coherent theory. So is robustness under efficiently effected transformation of problem encoding. These are one of the primary reasons for the acceptability of P as capturing problems which can be solved efficiently. In going from worst case analysis to average case analysis we should not and would not like to lose these properties.

Appendix B: DistNP-completeness of Π_{BH}

The proof, presented here, is due to [Gurevich 87] (an alternative proof is implied by [Levin 84]).

In the traditional theory of NP-completeness, the *mere* existence of complete problems is almost immediate. For example, it is extremely simple to show that the *Bounded Halting* problem is NP-complete.

Bounded Halting (BH) is defined over triples $(M, x, 1^k)$, where M is a non-deterministic machine, x is a binary string and k is an integer (given in unary). The problem is to determine whether there exists a computation of M on input x which halts within k steps. Clearly, Bounded Halting is in NP (here its crucial that k is given in unary). Let D be an arbitrary NP problem, and let M_D be the non-deterministic machine solving it in time $P_D(n)$ on inputs of length n, where P_D is a fixed polynomial. Then the reduction of D to BH consists of the transformation $x \rightarrow (M_D, x, 1^{P_D(|x|)})$.

In the case of distributional-NP an analogous theorem is much harder to prove. The difficulty is that we have to reduce all DistNP problems (i.e. pairs consisting of decision problems and simple distributions) to one single distributional problem (i.e. Bounded Halting with a single simple distribution). Applying reductions as above we will end up with many distributional versions of Bounded Halting, and furthermore the corresponding distribution functions will be very different and will not necessarily dominate one another. Instead, one should reduce a distributional problem, (D, μ), with an arbitrary P-computable distribution to a distributional problem with a fixed (P-computable) distribution (e.g. Π_{BH}). The difficulty in doing so is that the reduction should have the domination property. Consider for example an attempt to reduce each problem in DistNP to Π_{BH} by using the standard transformation of D to BH, sketched above. This transformation fails when applied to distributional problems in which the distribution of (infinitely many) strings is much higher than the distribution assigned to them by the uniform distribution. In such cases, the standard reduction maps an instance x having probability mass $\mu'(x) \gg 2^{-|x|}$ to a triple $(M_D, x, 1^{P_D(|x|)})$ with much lighter probability mass (recall $\mu'_{BH}(M_D, x, 1^{P_D(|x|)}) < 2^{-|x|}$). This violates the domination condition, and thus an alternative reduction is required. The key to the alternative reduction is an (efficiently computable) encoding of strings taken from an arbitrary polynomial-time computable distribution by strings which have comparable probability mass under a fixed distribution. This encoding will map x into a code of length bounded above by the logarithm of $1/\mu'(x)$. Accordingly, the reduction will map x to a triple $(M_{D,\mu}, x', 1^{|x|^{O(1)}})$, where $|x'| < O(1) + \log_2 1/\mu'(x)$, and $M_{D,\mu}$ is a non-deterministic Turing machine which first retrieves x from x' and then applies the standard non-deterministic machine (i.e., M_D) of the problem D. Such a reduction will be shown to satisfy all three conditions (i.e. efficiency, validity, and domination). Thus, instead of forcing the structure of the original distribution μ on the target distribution μ_{BH}, the reduction will incorporate the structure of μ into the the reduced instance.
The following technical Lemma is the basis of the reduction.

Coding Lemma: Let μ be a polynomial-time computable distribution function. Then there exist a coding function C_μ satisfying the following three conditions.

1) *Compression:* $\forall x$

$$|C_\mu(x)| \leq 2 + \min\{|x|, \log_2 \frac{1}{\mu'(x)}\}$$

2) *Efficient Encoding:* The function C_μ is computable in polynomial-time.

3) *Unique Decoding:* The function C_μ is one-to-one (i.e. $C_\mu(x) = C_\mu(x')$ implies $x = x'$).

Proof: The function C_μ is defined as follows. If $\mu'(x) \leq 2^{-|x|}$ then $C_\mu(x) = 0x$ (i.e. in this case x serves as its own encoding). If $\mu'(x) > 2^{-|x|}$ then $C_\mu(x) = 1z$, where $0.z$ is the binary expansion of a fraction in the interval $(\mu(x-1), \mu(x)]$ which has binary expansion of minimum length. In other words, $z = z'1$ where z' is the longest common prefix of the binary expansions of $\mu(x-1)$ and $\mu(x)$ (e.g. if $\mu(1010) = 0.10000$ and $\mu(1011) = 0.10101111$ then $C_\mu(1011) = 1z$ with $z = 101$).

We now verify that C_μ so defined satisfies the conditions of the Lemma. If $\mu'(x) \leq 2^{-|x|}$ then $|C_\mu(x)| = 1 + |x| < 2 + \log_2 \frac{1}{\mu'(x)}$. If $\mu'(x) > 2^{-|x|}$ then the interval $(\mu(x-1), \mu(x)]$ must contain a fraction with binary expansion of length $\leq \log_2(\frac{\mu'(x)}{2})^{-1}$ and hence $|C_\mu(x)| \leq 1 + 1 + log_2 \frac{1}{\mu'(x)}$.

Clearly, C_μ can be computed in polynomial-time by computing $\mu(x-1)$ and $\mu(x)$. Finally, note that C_μ is one-to-one by considering the two cases, $C_\mu(x) = 0x$ and $C_\mu(x) = 1z$. $\Box$

Using the coding function presented in the above proof, we introduce a non-deterministic machine $M_{D,\mu}$ so that the distributional problem (D,μ) is reducible to $\Pi_{BH} = (BH, \mu_{BH})$ in a way that all instances (of D) are mapped to triples with first element $M_{D,\mu}$. On input $y = C_\mu(x)$, machine $M_{D,\mu}$ computes $D(x)$, by first retrieving x from $C_\mu(x)$ (e.g., guess and verify) and next running the non-deterministic polynomial-time machine (M_D) which solves D.

The reduction maps an instance x (of D) to the triple $(M_{D,\mu}, C_\mu(x), 1^{P(|x|)})$, where $P(n) \stackrel{\text{def}}{=} P_D(n) + P_C(n) + n$, $P_D(n)$ is a polynomial bounding the running time of M_D on acceptable inputs of length n, and $P_C(n)$ is a polynomial bounding the running time of an algorithm for encoding inputs (of length n).

Proposition: The above mapping constitutes a reduction of (D,μ) to (BH, μ_{BH}).

Proof:

- The transformation can be computed in polynomial-time. (Recall that C_μ is polynomial-time computable.)

- By construction of $M_{D,\mu}$ it follows that $D(x) = 1$ if and only if there exists a computation of machine $M_{D,\mu}$ that on input $C_\mu(x)$ halts outputting 1 within $P(|x|)$ steps. (On input $C_\mu(x)$, machine $M_{D,\mu}$ non-deterministically guesses x, verifies in $P_C(|x|)$ steps that x is encoded by $C_\mu(x)$, and non-deterministically "computes" $D(x)$.)

- To see that the distribution induced by the reduction is dominated by the distribution μ_{BH}, we first note that the transformation $x \to C_\mu(x)$ is one-to-one. It suffices to consider instances of BH which have a preimage under the reduction (since instances with no preimage satisfy the condition trivially). All these instances are triples with first element $M_{D,\mu}$. By the definition of μ_{BH}
$$\mu'_{BH}(M_{D,\mu}, C_\mu(x), 1^{P(|x|)}) = c \cdot \frac{1}{P(|x|)^2} \cdot \left(\frac{1}{|C_\mu(x)|^2} \cdot 2^{-|C_\mu(x)|}\right)$$
where $c = \frac{1}{|M_{D,\mu}|^2 \cdot 2^{|M_{D,\mu}|}}$ is a constant depending only on (D,μ).
By virtue of the coding Lemma
$$\mu'(x) \leq 4 \cdot 2^{-|C_\mu(x)|}$$
It thus follows that
$$\begin{aligned} \mu'_{BH}(M_{D,\mu}, C_\mu(x), 1^{P(|x|)}) &\geq c \cdot \frac{1}{P(|x|)^2} \cdot \frac{1}{|C_\mu(x)|^2} \cdot \frac{\mu'(x)}{4} \\ &> \frac{c}{4 \cdot |M_{D,\mu}, C_\mu(x), 1^{P(|x|)}|^2} \cdot \mu'(x) \end{aligned}$$

The Proposition follows. $\Box$