

Average-Case Computational Complexity Theory

Jie Wang¹

ABSTRACT Being NP-complete has been widely interpreted as being computationally intractable. But NP-completeness is a worst-case concept. Some NP-complete problems are “easy on average”, but some may not be. How is one to know whether an NP-complete problem is “difficult on average”? The theory of average-case computational complexity, initiated by Levin about ten years ago, is devoted to studying this problem. This paper is an attempt to provide an overview of the main ideas and results in this important new sub-area of complexity theory.

1 Introduction

Despite many years of intensive study, there has been no success in finding efficient algorithms to solve NP-complete problems, where by efficient algorithm we mean an algorithm which is fast in the worst case. Thus the search for algorithms that are “efficient” according to some more modest criterion has attracted increasing attention. One approach is to find algorithms to solve NP-complete problems which, although possibly very slow on some inputs, are fast on average with respect to the underlying probability distributions on instances. Algorithms that are fast on average have been found for several NP-complete problems, such as the vertex k -coloring problem [Wil84] and the Hamiltonian path problem [GS87] under commonly used distributions on graphs. Yet there are NP-complete problems which have resisted so far such “average-case” attacks. Are these problems difficult on average? What does it mean for a problem to be difficult on average, and how is one to know? In his seminal paper [Lev86], Levin initiated the study of answering these questions. Two fundamental and robust notions were defined along similar lines to the NP-completeness theory. Namely, the notion of average polynomial time for measuring “easiness” on average and the notion of average-case NP-completeness for measuring “hardness” on average for computational problems. Levin then showed that a tiling problem is average-case NP-complete if each parameter of an instance is

¹Department of Mathematical Sciences, University of North Carolina, Greensboro, NC 27412, USA. Supported in part by NSF under grant CCR-9424164.

randomly selected. This framework has been studied and enhanced by a number of researchers and several more average-case NP-complete problems have been found. Such average-case completeness results, as indicated by Levin [Lev86], could not only save misguided “positive” efforts such as trying to find fast-on-average algorithms for problems that may not have one, but could also be used in areas (like cryptography) where hardness on average of some problems is a frequent assumption.

This paper is intended to present systematically the current state of knowledge on average-case complexity. It is centered around three fundamental ideas: the notion of average polynomial time, the notion of reductions and completeness, and the notion of feasible distributions.

2 Average Polynomial Time

We use the binary alphabet $\Sigma = \{0, 1\}$ for encoding strings. Denote by \leq the standard lexicographical order on Σ^* . A *probability distribution* μ is a real-valued function from Σ^* to $[0, 1]$ such that $\sum_x \mu(x) = 1$. We assume that μ on the empty string is always 0. We also use *distribution*, *probability*, *weight*, or *density* to denote probability distribution. The *distribution function* of μ , denoted by μ^* , is defined by $\mu^*(x) = \sum_{y \leq x} \mu(y)$.² For a set $A = \{x: \phi(x)\}$, we use $\mu[\phi(x)]$ or $\mu[A]$ to denote $\sum_{x \in A} \mu(x)$. A conditional distribution of μ on a set A is equal to $\mu(x)/\mu[A]$ if $x \in A$ and $\mu[A] > 0$, and 0 otherwise. For a function f , we use $f^\varepsilon(x)$ to denote $(f(x))^\varepsilon$ for $\varepsilon > 0$ and use f^{-1} to denote the inverse of f . We use \mathbb{R}^+ to denote the positive reals and \mathbb{N} to denote the natural numbers.

An obvious choice to measure the average efficiency of an algorithm is to use expected polynomial time. An algorithm runs in *expected polynomial time* over a probability distribution μ if $(\exists k \geq 0)(\forall n) \sum_{x, |x|=n} t(x)\mu_n(x) \leq O(n^k)$, where $t(x)$ is the running time of the algorithm on x , and μ_n is the conditional distribution on strings of length n . However, this definition is machine dependent and so cannot be used to build a robust theory. There are several issues on defining a robust notion of average polynomial time, which are discussed below. These issues were either mentioned explicitly or hinted at by Levin [Lev86] and they have been elaborated from various aspects by Johnson [Joh84], Gurevich [Gur89, Gur91a, Gur91b], Venkatesan [Ven91], and Impagliazzo [Imp95], from which Levin’s definition of average polynomial time (given in Definition 2.1) can be derived naturally and be well justified.

²Levin [Lev86] used μ to denote a distribution function and used μ' to denote its density. Since almost all of our statements can be stated directly in terms of probability distributions, denoting a probability distribution by μ without a prime seems more convenient. The notations μ and μ^* were first used in [Gur91a].

Model Independence. Assume an algorithm runs in polynomial time on a $1 - 2^{-0.1n}$ fraction of instances of length n , and runs in $2^{0.09n}$ time on instances in the remaining $2^{-0.1n}$ fraction. Then it is easy to see that the expected running time on strings of length n is bounded above by a polynomial in n . However, the expected running time will be exponential in a machine model that is quadratically slower. If a problem is easy on average in one model of computation, then it should be easy on average on all polynomially equivalent models. Even within the same model, if t is polynomial on average, then for any $k > 0$, t^k should also be polynomial on average. Moreover, if a problem is polynomial on average under one encoding method, then it should be polynomial on average under all polynomially equivalent encodings.

Balancing. Assume an algorithm solves a problem in polynomial time on a $1 - n^{-2}$ fraction of instances of length n . Then we still have no guarantee of an algorithm that is fast on average since it may take exponential time to solve the instances in the remaining n^{-2} fraction. A balance is therefore required between the fraction of hard instances and the hardness of these instances. The fraction of the hard instances should be polynomially related to the time needed to solve them. Only a sub-polynomial fraction of instances should require super-polynomial time.

Distribution. An average could be taken on instances of fixed size or on *all* instances. One may tend to consider distributions defined on instances of a bounded size at any fixed point in time, but it is also enough to consider distributions defined on all instances by first selecting a length according to some appropriate distribution and then selecting strings of that length according to the given distribution on strings of bounded size.

Under the worst-case measurement, the running time of an algorithm is measured as a function of the length of the inputs. Under the average-case measurement, an efficient algorithm with input distribution μ is allowed to run a longer time on instances with smaller weights. Instances with smaller weights are rarer and so one may use a function $r: \Sigma^+ \rightarrow \mathbb{R}^+$ to measure the “rareness” of instances, defined in such a way that if the weight of an instance x is smaller, then the value of its rareness $r(x)$ is larger. As discussed in the balancing issue, the fraction of inputs x with high values of rareness should be small. Probably the most general way to satisfy this requirement is to have, for some positive constants k and c and for all $l \in \mathbb{R}^+$, $\mu[r(x) > l^k] < c/l$, or equivalently, $\sum_x r^\varepsilon(x)\mu(x) < \infty$ for some $\varepsilon > 0$.³ One may then measure the average-case running time of an algo-

³To see the equivalence, assume that $\mu[r(x) > l^k] < c/l$. Let $\varepsilon = \frac{1}{2k}$, then for all $l \in \mathbb{R}^+$, $\mu[r^\varepsilon(x) > l] < c/l^2$, and so $\sum_x r^\varepsilon(x)\mu(x) \leq \sum_{n=1}^{\infty} n\mu[n-1 < r^\varepsilon(x) \leq n] = \sum_{n=0}^{\infty} \mu[r^\varepsilon(x) > n] < \infty$. The other direction is straightforward by a simple application of Markov’s inequality.

rithm on input x as a function of $|x|r(x)$ rather than $|x|$. This formulates the idea that a longer time is allowed on inputs with smaller weights, and it also guarantees model independence. The running time t of an algorithm is to be called *polynomial on average* if there exists a $k > 0$ such that for all x , $t(x) \leq (|x|r(x))^k$. Hence, $t^{1/k}(x)|x|^{-1} \leq r(x)$. Let $\delta = \min\{\varepsilon, 1\}$, then $\sum_x t^{\delta/k}(x)|x|^{-1}\mu(x) \leq \sum_x (t^{1/k}|x|^{-1})^\delta \mu(x) < \sum_x r^\delta(x)\mu(x) < \infty$. This gives rise to the following definition due to Levin [Lev86].

Definition 2.1 A function $f: \Sigma^+ \rightarrow \mathbb{N}$ is *polynomial on μ -average* if there exists an $\varepsilon > 0$ such that $\sum_x f^\varepsilon(x)|x|^{-1}\mu(x) < \infty$.

Clearly, Definition 2.1 is model- and encoding-independent, and it satisfies the balancing requirement as shown in footnote 3, based on which the following lemma is straightforward.

Lemma 2.1 *A function f is polynomial on μ -average iff there are constants $c, k > 0$ such that for all $l \in \mathbb{R}^+$, $\mu[f(x) > (l|x|)^k] < c/l$.*

Further exposition on the balancing issue can be found in [Gur89, Gur91b]. Regarding the distribution issue, Impagliazzo [Imp95] observed that Definition 2.1 is equivalent to taking the average on instances up to length n since, when n is sufficiently large, $\mu[|x| \leq n]$ is greater than a fixed positive constant.

Lemma 2.2 *A function f is polynomial on μ -average iff there exists $\varepsilon > 0$ such that $(\exists c > 0)(\forall n) \sum_{x, |x| \leq n} f^\varepsilon(x)\mu_{\leq n}(x) \leq cn$, where $\mu_{\leq n}$ is the conditional distribution of μ on $\{x: |x| \leq n\}$.*

If the average is taken over strings of length exactly n , then Lemma 2.2 is still true under the assumption that, for some fixed polynomial p and for every n , either $\mu[|x| = n] = 0$ or $\mu[|x| = n] \geq 1/p(n)$ [Gur91a].

Clearly, every polynomial is polynomial on average with respect to any distribution μ . If a function f is an expected polynomial over a distribution μ , namely, for some constants $k \geq 0$ and $c > 0$ and for all n , $\sum_{|x|=n} f(x)\mu_n(x) \leq cn^k$, then $\sum_{|x|=n} f^\varepsilon(x)|x|^{-1}\mu_n(x)$ (where $\varepsilon = \min\{1, 1/k\}$) $< \sum_{|x|=n} (1 + (f^\varepsilon(x)|x|^{-1})^{1/\varepsilon})\mu_n(x) \leq 1 + c$. It follows that f is polynomial on μ -average. If f and g are polynomial on μ -average, then so are $\max(f, g)$, $f + g$, $f \cdot g$, and f^l for any positive real number l [Gur91a]. To see the case of $f \cdot g$, for instance, it suffices to note that $f \cdot g \leq f^2 + g^2$.

A problem is solvable in average polynomial time with respect to distribution μ if it can be solved by a *deterministic* algorithm whose running time is polynomial on μ -average. A problem with an associated probability distribution is called a *distributional problem*. Other names such as “random problems” [Lev86] and “randomized problems” [Gur91a] have also been used in the literature. The term “distributional problem” was first used in [BCGL92]. We adopt this terminology in this paper.

3 Average-Case Completeness

Given two distributional problems, we wish to know which one is computationally more difficult. The standard technique for such comparisons is to find a reduction from one problem to another. Levin [Lev86] provided such a notion. Gurevich [Gur91a] advanced this theory in several ways which, at the same time, made Levin’s ideas accessible.

3.1 Polynomial-Time Reductions

For simplicity, we first consider distributional decision problems. Such a problem is comprised of a set of instances which are either positive or negative for the underlying decision problem, and a probability distribution on these instances. It is customary to only specify the set of positive instances and we are only concerned with instances with positive probability. Under such a convention, we use (D, μ) to denote a distributional decision problem, where D is the set of all positive instances x with $\mu(x) > 0$. The problem is to decide, for a given instance x with $\mu(x) > 0$, whether $x \in D$. If $D \in \text{NP}$, then (D, μ) is called a distributional NP decision problem.

Denote by AP the class of all distributional decision problems (D, μ) such that D is solvable in time polynomial on μ -average.

Let f be a function with input distribution μ . It is reasonable to define the output distribution on y , denoted by $f(\mu)(y)$, to carry all the weights of those instances x with $f(x) = y$. Namely, for all $y \in \text{range}(f)$, $f(\mu)(y) = \sum_{f(x)=y} \mu(x)$. Denote by $f^*(\mu)$ the distribution function of $f(\mu)$, so $f^*(\mu)(y) = \sum_{f(x) \leq y} \mu(x)$. Reductions f from (A, μ) to (B, ν) should be closed for AP, meaning that if $(B, \nu) \in \text{AP}$ then so is (A, μ) . One way to guarantee this is to require that f efficiently reduce A to B as in the worst case, and it should not reduce “frequent” instances of A to “rare” instances of B . This means that the induced weight on the output $y = f(x)$ should be bounded above (within a polynomial factor) by the weight on y according to the distribution of B . Namely, $f(\mu)(y) \leq |y|^{O(1)} \nu(y)$.⁴ If $|y| \leq |x|^{O(1)}$, it follows that there exists a distribution μ_1 such that $\forall x: \mu(x) \leq |x|^{O(1)} \mu_1(x)$ and $\nu(f(x)) = f(\mu_1)(f(x))$,⁵ which turns out to be sufficient for defining reductions. Levin [Lev86] defined reductions based on these ideas, and their current forms, namely, Definitions 3.2 and 3.3, were given by Gurevich in [Gur91a].

Definition 3.1 Let μ and ν be distributions on the same domain. Then μ is *dominated by* ν , written as $\mu \preceq \nu$, if there exists a polynomial p such

⁴Actually, we are dealing with the conditional probability distribution $\nu[y|Y]$ with $Y = \text{range}(f)$. Since $\nu[y|Y] = \nu(y)/\nu[Y]$ and $\nu[Y]$ is a positive constant, it is equivalent to using $\nu(y)$ in the inequality.

⁵The converse is also true if $|x|$ is bounded by a polynomial in $|f(x)|$ [Gur91a].

that for all x , $\mu(x) \leq p(|x|)\nu(x)$.

Definition 3.2 Let μ and ν be distributions on the instances of the decision problems A and B , respectively, and f be a reduction from A to B . Then μ is *dominated by ν with respect to f* , written as $\mu \preceq_f \nu$, if there exists a distribution μ_1 on A such that $\mu \preceq \mu_1$ and $\nu(y) = f(\mu_1)(y)$ for all $y \in \text{range}(f)$, i.e., $\nu(y) = \sum_{f(x)=y} \mu_1(x)$.

Definition 3.3 (A, μ) is *polynomial-time reducible* to (B, ν) if there is a polynomial-time computable reduction f such that A is many-one reducible to B via f and $\mu \preceq_f \nu$.

If a reduction f is one-one, then it is straightforward to see that $\mu \preceq_f \nu$ iff $\mu \preceq \nu \circ f$. This observation is useful in proving completeness results. In the sequel, we will often use “p-time” to denote “polynomial-time”, and “ap-time” to denote “average polynomial-time.”

Lemma 3.1 (1) If (A, μ) is p-time reducible to (B, ν) and (B, ν) is in AP, then so is (A, μ) . (2) The p-time reductions are transitive.

Proof. (1) Let f be a p-time reduction from (A, μ) to (B, ν) . Then there is a distribution μ_1 and a polynomial p such that, for all x , $\mu(x) \leq p(|x|)\mu_1(x)$, and for all $y \in \text{range}(f)$, $\nu(y) = \sum_{f(x)=y} \mu_1(x)$. Since $(B, \nu) \in \text{AP}$, B is deterministically decidable in time t where, for some $\varepsilon > 0$, $\sum_y t^\varepsilon(y)|y|^{-1}\nu(y) < \infty$. For any $k > 0$, $\sum_y t^{\varepsilon/k}(y)|y|^{-1/k}\nu(y) < \sum_y (1 + t^\varepsilon(x)|y|^{-1})\nu(y) < \infty$. Since f is p-time computable, there is a $k > 0$ such that $|f(x)| < |x|^k$ for all but finitely many x . Let $h = t \circ f$. Then $\sum_x h^{\varepsilon/k}(x)|x|^{-1}\mu_1(x) \leq \sum_{f(x)=y} t^{\varepsilon/k}(y)|y|^{-1/k}\mu_1(x) \leq \sum_y t^{\varepsilon/k}(y)|y|^{-1/k}\nu(y) < \infty$. It follows that $h(x)/p^{k/\varepsilon}(|x|)$ is polynomial on μ -average, and so then is $(h(x)/p^{k/\varepsilon}(|x|))p^{k/\varepsilon}(|x|) = h(x)$. Thus, deciding whether $x \in A$ can be carried out in time $h(x)$ plus the time to compute $f(x)$ and so $(A, \mu) \in \text{AP}$. The proof of (2) is straightforward. ■

3.2 Polynomial-Time Computable Distributions

Let (D, μ) be a distributional NP problem. If every other distributional NP problem is p-time reducible to it, then (D, μ) has no ap-time algorithm unless every NP problem under any allowable distribution has one. In order for such a (D, μ) to exist, a certain condition on the computability of allowable distributions is needed. If arbitrary distributions are allowed, then no complete distributional problems can exist with respect to one-one, p-time reductions. In particular, Wang and Belanger [WB93a] showed that for any distribution ν , there is a distribution μ such that for any one-one, p-time reduction f , μ cannot be dominated by ν with respect to f . Reductions that are one-one are a reasonable assumption as all the known NP-complete problems are indeed complete via one-one, p-time reductions.

Levin [Lev86] suggested that it is reasonable to require distributions be p-time computable and it is also reasonable to restrict attention to distributions with rational values (see Lemma 3.2). A function $f: \Sigma^+ \rightarrow [0, 1]$ is p-time computable if there is a deterministic algorithm which on input x outputs $f(x)$ in time bounded by a polynomial in $|x|$. Gurevich [Gur91a] extended this notion to real-valued functions following [Ko83].

Definition 3.4 Let $f: \Sigma^+ \rightarrow [0, 1]$ be a real-valued function. Then f is p-time computable if there is a deterministic algorithm which, on every string x and every positive integer k , outputs a finite binary fraction y in time bounded by a polynomial in $|x|$ and k and such that $|f(x) - y| \leq 2^{-k}$.

Clearly, if μ^* is p-time computable then so is μ . Blass showed that the converse is not true unless $P = NP$ (see [Gur91a]). With this fact in mind, we assume that in the sequel, when we say that a distribution μ is p-time computable we mean that both μ and μ^* are p-time computable. There is strong evidence to believe, as hypothesized by Levin (see [Joh84]), that any “natural” probability distribution is either p-time computable or else is dominated by one that is. One can indeed verify that all commonly used distributions do satisfy this property.

Gurevich [Gur91a] showed that any p-time computable distribution is bounded above by a nicely-behaved (“linearly presentable”) p-time computable distributions as follows.

Lemma 3.2 *Let μ be a p-time computable distribution. Then there is a distribution ν such that for all x , $\nu(x) > 0$, $\nu(x)$ has at most $4 + 2|x|$ digits, and $\mu(x) < 4\nu(x)$. Moreover, there is a deterministic algorithm that on input x outputs $\nu^*(x)$ in time polynomial in $|x|$.*

Proof. Since μ^* is p-time computable, there is a p-time computable finite binary fraction $b(x)$ such that for all x , $|\mu^*(x) - b(x)| < d_x/2$, where $d_x = 2^{-2|x|}$. Round $b(x)$ down to $2|x| + 1$ digits. Add $d_x/2$ if the last digit is 1. This produces a binary fraction $B(x)$ with at most $2|x|$ digits, and $|\mu^*(x) - B(x)| < d_x$. Define ν by $\nu(x) = \frac{1}{4}(B(x) - B(x^-) + 3d_x)$, where x^- is the immediate predecessor of x . Then $\nu(x)$ has at most $4 + 2|x|$ digits and $\nu^*(x) = \frac{1}{4}(B(x) + 3 \sum_{y \leq x} d_y) = \frac{1}{4}(B(x) + 3 \sum_{n=1}^{|x|} 2^{-n}) \rightarrow 1$ (as $|x| \rightarrow \infty$). It follows that $4\nu(x) > (\mu^*(x) - d_x) - (\mu^*(x^-) + d_x) + 2d_x = \mu(x)$. ■

Let f be a p-time computable reduction and μ an input distribution. If the induced output distribution $f(\mu)$ is p-time computable, one might suspect that μ must also be p-time computable, which, however, was proven false in [WB93a]. To see this, first construct a set $H = \{x_1, x_2, \dots\}$ with $H \notin P$ and for all n , $|x_n| = n$, then define a distribution μ such that for almost all (*i.e.*, all but finitely many) x , $\mu(x) = 2^{-2|x|}$ if $x \in H$ and $2^{-3|x|}$ otherwise. Let $b(x)$ be a finite binary fraction with $|\mu(x) - b(x)| < 2^{-2|x|+2}$, then for almost all x , $x \in H$ iff $b(x)$ has a 1 by the $2|x| + 1$ position to the right of the binary point. Now define a distribution ν such that for almost

all x , $\nu(x) = \frac{3}{2}2^{-2|x|}$ if $x \in \{0\}^*$ and $|x|^{-2}2^{-|x|}$ otherwise. Then ν is p-time computable but μ is not. If μ were p-time computable, then there is a p-time computable $b(x)$ which implies that $H \in P$, a contradiction. Define f by $f(x) = 0^{|x|}$, then for almost all $y \in \{0\}^*$, $f(\mu)(y) = \sum_{f(x)=y} \mu(x) = \nu(y)$.

3.3 Uniform Distributions

As discussed in the distribution issue in Section 2, one can define a distribution on all binary strings by first selecting lengths and then selecting strings of that length. Although it is mathematically impossible to select strings with equal chance from an infinite sample space, strings of the same length can be selected with equal likelihood. It is also impossible to select integers from \mathbb{N} with the same probability, but we can select an integer with a probability close to be “uniform.”

Definition 3.5 A p-time computable distribution μ on Σ^+ is called *uniform* if for all x , $\mu(x) = \rho(|x|)2^{-|x|}$, where $\sum_n \rho(n) = 1$ and there is a polynomial p such that for all but finitely many n , $\rho(n) \geq 1/p(n)$.

The second requirement in the above definition guarantees that almost every length gets a “fair” amount of weight to be selected.⁶ It is *important* to note that for the purpose of proving completeness results, $\rho(n) \geq 1/p(n)$ is the only requirement needed since domination allows a polynomial factor, and so some longer strings can certainly be given more weights than shorter ones. Levin [Lev86] used n^{-2} for $\rho(n)$ for notational convenience (normalized by dividing by $\sum_n n^{-2} = \pi^2/6$), and $|x|^{-2}2^{-|x|}$ is often referred to as the default uniform distribution.

3.4 Distribution Controlling Lemma

Distributions that are p-time computable can be dominated (with respect to p-time computable functions) by uniform distributions, which is an important property for proving completeness results. Levin [Lev86] first proved this property using a “perfect rounding” technique, which is interesting in its own right (see [Gur91a] for an exposition). Gurevich [Gur91a] provided a different and easier proof, based on which Wang and Belanger [WB95] further showed that if the distribution is not too small, it will also dominate the same uniform distribution within a constant factor.

⁶We may also weaken the second requirement in Definition 3.5 to only require that, for all but finitely many n , either $\rho(n) = 0$ or $\rho(n) \geq 1/p(n)$. In this case, μ is uniform in the sense that if a string x can be picked with a positive probability, then every string of length $|x|$ will have the same chance to be selected with a “fair” probability greater than $2^{-|x|}/p(|x|)$.

Lemma 3.3 (Distribution Controlling Lemma) *Let μ be a p -time computable distribution.*

1. *There exists a total, one-one, p -time computable and invertible function $\alpha: \Sigma^+ \rightarrow \Sigma^+$ such that for all x , $\mu(x) < 4 \cdot 2^{-|\alpha(x)|}$.*
2. *Moreover, if there exists a polynomial p such that for all x , $\mu(x) > 2^{-p(|x|)}$, then there is a total, one-one, p -time computable and invertible function $\beta: \Sigma^+ \rightarrow \Sigma^+$ such that for all x , $2^{-|\beta(x)|} \leq 4 \cdot \mu(x) < 20 \cdot 2^{-|\beta(x)|}$.*

Proof Sketch. Let ν be the linear presentable distribution that bounds $\mu/4$, as guaranteed by lemma 3.2. Let $\alpha(x)$ be the shortest binary string such that $\nu^*(x^-) < 0.\alpha(x)1 \leq \nu^*(x)$. Then $\nu(x) = \nu^*(x) - \nu^*(x^-) < 2^{-|\alpha(x)|}$. Now let l be the position of the leftmost digit that is 1 in the binary fraction produced by a p -time algorithm that computes $\mu(x)/4$ on input $(x, 1^{p(|x|)+4})$. Then $2^{-(l+1)} \leq \mu(x)/4 < 5 \cdot 2^{-(l+1)}$. Let $\beta(x) = \alpha(x)10^k$, where $k = l - |\alpha(x)|$. ■

3.5 Distributional NP-Completeness

Denote by DistNP the class of all distributional NP problems (D, μ) such that $\mu \preceq \nu$ for some p -time computable distribution ν . Other names such as RNP [Gur91a], $\langle \text{NP}, \text{P-comp} \rangle$ [BCGL92], and DNP [WB93a] have also been used for DistNP. DistNP was first used in [IL90, BCGL92]. A distributional problem is average-case NP-complete (or complete for DistNP) if it is in DistNP and every distributional problem in DistNP is reducible to it. Levin [Lev86] showed that a distributional tiling problem is average-case NP-complete. Levin also implicitly showed that a distributional halting problem is average-case NP-complete. Gurevich [Gur91a] explicitly defined such a problem and provided a simple proof of its completeness.

Let M_1, M_2, \dots be a fixed enumeration of nondeterministic Turing machines in which the index i is an integer in binary form that codes up the symbols, states, and transition table of the i -th Turing machine M_i . The DISTRIBUTIONAL HALTING PROBLEM (VERSION 1) consists of binary strings i, x and 1^n as instances, where i and n are integers. The question is to decide whether M_i accepts x within n steps. Denote by K_1 the set of all positive instances. The probability distribution $\mu_{K_1}(i, x, 1^n)$ of instance $(i, x, 1^n)$ (positive or negative) is proportional to $2^{-(l+m)}l^{-2}m^{-2}n^{-2}$, where $l = |i|$ and $m = |x|$.⁷ The halting problem is then denoted by $\text{DH1} = (K_1, \mu_{K_1})$.

Theorem 3.4 *DH1 is complete for DistNP.*

⁷The index i here is selected uniformly as a binary string. How to select i , however, is not important to obtain the completeness result. One can use one's favorite distributions to select it, e.g., one may select states and transition functions according to different distributions.

Proof. Let (D, μ) be an arbitrary problem in DistNP. Then there is a nondeterministic Turing machine M which accepts D in polynomial time. By the Distribution Controlling Lemma, there is a total, one-one, p-time computable and invertible function α such that $\mu(x) \preceq 2^{-|\alpha(x)|}$. Define a Turing machine M' as follows. On input y , if $\alpha^{-1}(y)$ is defined, then M' simulates M on $\alpha^{-1}(y)$ and rejects otherwise. Clearly for all x , M accepts x iff M' accepts $\alpha(x)$. It is easy to see that M' on input $\alpha(x)$ is bounded in time $p(|x|)$ for some polynomial p . Let i be an index such that $M' = M_i$. Let $f(x) = (i, \alpha(x), 1^{p(|x|)})$. Then f is one-one and p-time computable. By construction, $x \in D$ iff $f(x) \in K_1$. Moreover, $\mu_{K_1}(f(x)) = 2^{-|\alpha(x)|}/q(|x|)$ for some polynomial q , which dominates $\mu(x)$. So $\mu \preceq_f \mu_{K_1}$. ■

The distributional tiling problem was first shown to be complete by Levin in [Lev86] and a detailed proof can be found in [Gur91a, BW93]. A tile is a square with a symbol on each side which may not be rotated or turned over. We assume that there are infinite copies of each tile. By a tiling of an $n \times n$ square we mean an arrangement of n^2 tiles covering the square in which the symbols on the common sides of adjacent tiles are the same. The DISTRIBUTIONAL TILING PROBLEM (DT) consists of binary strings T , 1^n , and S as instances, where T is a finite set of tiles, n is an integer, and $S = s_1 s_2 \dots s_k$ ($k \leq n$) is a sequence of tiles that match each other, *i.e.* the symbol on the right side of s_i is the same as that on the left side of s_{i+1} . The question is to decide whether S can be extended to tile an $n \times n$ square using tiles from T . Denote by BT the set of all positive instances $(T, 1^n, S)$. The probability distribution μ_{BT} on instance $(T, 1^n, S)$ (positive or negative) is proportional to $\mathbf{Pr}[T]n^{-2}\mathbf{Pr}[S]$, where $\mathbf{Pr}[T]$ is the probability of T (one can use one's favorite distribution to select T or simply select one uniformly at random among binary strings since T can be coded in binary) and $\mathbf{Pr}[S]$ is the probability of choosing S , where S is chosen by choosing k at random with probability $\frac{1}{n}$, choosing the first tile s_1 at random from T , and choosing the s_i ($i > 1$) sequentially and uniformly at random from those tiles in T that match s_{i-1} .

Let $(D, \mu) \in \text{DistNP}$ and let α be as in the proof of Theorem 3.4. Let $x' = \alpha(x)$. Similar to the proof of Theorem 3.4 and the standard completeness proof for bounded tiling in the worst case, one might be tempted to reduce (D, μ) to DT by reducing an instance x of D to an instance of DT in which S represents the initial instantaneous description of an appropriate Turing machine on x' followed by some “blank” tiles. The problem with this approach is that $\mathbf{Pr}[S]$, which is at most $3^{-|x'|}$, is too small to dominate $\mu(x)$. If S does not end with “blank” tiles, then $\mathbf{Pr}[S]$ could be dominated by $2^{-|x'|}$. But something needs to be done to make sure that reductions will not reduce a negative instance of D to a positive instance of DT.

Theorem 3.5 *The distributional tiling problem is complete for DistNP.*

Proof Sketch. Let $(D, \mu) \in \text{DistNP}$ and α be as in the proof of Theorem

3.4. Let M be a nondeterministic Turing machine which accepts D . Assume that the length of every computation path of M on input x is bounded by a polynomial in $|x|$. For any string y , let σ be $|y|$ written in binary. Set $y_L = 0^{|\sigma|}1\sigma y$, then $|y_L| = |y| + O(\log |y|)$ and y can be found efficiently from any string beginning with y_L . Construct a Turing machine M' as follows. It rejects any input if it does not begin with y_L for some y , or $\alpha^{-1}(y)$ is not defined. M' then simulates M on $\alpha^{-1}(y)$. Let $x' = \alpha(x)$. Then M' will either accept all or reject all inputs beginning with x'_L (depending on whether or not M accepts x), and there exists a polynomial p such that the length of every computation path of M' on x is strictly less than $p(|x|)$. Following a standard procedure (e.g., see [LP81]), we can construct tiles to represent the transition function of M' , and tiles that can duplicate tape symbols a and pairs (q, a) in the vertical direction, where q is the accepting state. We then construct tiles $((q_0, d), \$, \$, \#)$ and $(d, \$, \#, \#)$ for $d \in \{0, 1\}$, where $(\cdot, \cdot, \cdot, \cdot)$ represents the symbols on each side of a tile in the order of top, bottom, left, and right, q_0 is the initial state of M' and $\$$ and $\#$ are symbols that have not been used by other tiles. Let $T(M')$ be the set of all these tiles. Let $f(x) = (T(M'), 1^{p(|x|)}, S)$, where $S = ((q_0, x'_1), \$, \$, \#)(x'_2, \$, \#, \#) \cdots (x'_{|x'|}, \$, \#, \#)$ and x'_i represents the i th bit of x'_L . The probability of the i th tile of S for $i > 1$ is $\frac{1}{2}$ since there are only two tiles that can match the $(i-1)$ th tile. Since $|x'_L| = |x'| + O(\log |x'|)$ and $\log |x'| = O(\log |x|)$, $\Pr[S]$ is proportional to $2^{-|x'_L|}/p(|x|) = |x|^{-O(1)}2^{-|x'|}$ and so $\mu \preceq \mu_{\text{BT}}$. It is easy to see that $x \in D$ iff $f(x) \in \text{BT}$ with S occupying the bottom left-hand side of the $p(|x|) \times p(|x|)$ square since the tiling can only duplicate the computation path which can be extended to cover the entire square only when an accepting state is reached. ■

Several more natural NP-complete problems such as the Post correspondence problem [Gur91a], the word problem for Thue systems, [WB95] and the word problem for finitely presented groups [Wan95a] have also been shown to be complete for DistNP using p-time reductions in which each parameter of the instances is selected uniformly at random. Proofs of these results are more involved and are not included here due to page limitations.

3.6 Average Polynomial-Time Reductions

As indicated in [Lev86] and discussed in [Gur91a], instead of requiring a reduction f be p-time computable, one only needs to require that f be computable in ap-time. This weakens the original definition in two ways: the reduction is weaker and the domination is weaker. The distribution μ is *weakly dominated by* the distribution ν if for all x , $\mu(x) \leq g(x)\nu(x)$, for some function g which is polynomial on μ -average. The following definition was shown to be the most general in the context of deterministic many-one reductions [BG91].

Definition 3.6 (A, μ) is *ap-time reducible* to (B, ν) if there is a many-one reduction f that reduces A to B such that f is computable in time polynomial on μ -average and μ is weakly dominated by μ_1 for some μ_1 such that for all x , $\nu(f(x)) = f(\mu_1)(f(x))$.

Lemma 3.6 (1) If (A, μ) is *ap-time reducible* to (B, ν) and (B, ν) is in AP, then so is (A, μ) . (2) The *ap-time reductions* are transitive.

The proof of Lemma 3.6 is similar to that of Lemma 3.1, and can be found in [Gur91a]. It is not known whether ap-time reductions are more powerful than p-time reductions for DistNP problems although one suspects that they are. One nice feature of ap-time reductions is that they can be used to establish completeness results for the class of distributional problems which are solvable by nondeterministic algorithms in average polynomial time. It is not known whether the same can be shown using p-time reductions. Studying such problems seems both logical and natural as noted in [BCGL92, Gur91a]. The class of such problems, denoted by ANP, is a nondeterministic version of AP, which properly includes DistNP [WB93a] and is a logical average-case analog of NP. The proof of Theorem 3.4 can be easily modified to show the following result [WB93a].

Theorem 3.7 If (D, μ) is in ANP, and μ is weakly dominated by a p-time computable distribution, then (D, μ) is *ap-time reducible* to the *distributional halting problem*.

Thus, all average-case NP-complete problems are also ap-time complete for problems in ANP with p-time computable distributions. On the other hand, there are problems which are not in DistNP but are ap-time complete for problems in ANP with p-time computable distributions [WB93a].

3.7 Distributional Search Problems

A distributional search problem is specified by (R, μ) , where R is a binary predicate and μ is a distribution. Given an input x , the search problem is to find w (a witness) such that $R(x, w)$ is satisfied. (R, μ) is solvable in average polynomial time if there is a deterministic algorithm solving the search problem R in polynomial time on μ -average. Reductions on distributional search problems, which have the desired properties, can be similarly defined following the framework given in [VL88].

Definition 3.7 Let (R_i, μ_i) ($i \in \{1, 2\}$) be two distributional search problems and $Y_i = \{x : \mu_i(x) > 0 \ \& \ (\exists w)R_i(x, w)\}$. (R_1, μ_1) is *p-time reducible* to (R_2, μ_2) if there are p-time computable functions f and g satisfying the following conditions: (1) $(\forall x)[x \in Y_1 \text{ iff } f(x) \in Y_2]$; (2) $(\forall x \in Y_1)(\forall w)[R_2(f(x), w) \rightarrow R_1(x, g(x, w))]$; and (3) $\mu_1 \preceq_f \mu_2$.

The notion of ap-time reductions for search problems can be similarly defined. If R is p-time computable and there is a polynomial p such that $(\forall x, w)[R(x, w) \rightarrow |w| \leq p(|x|)]$, then R is called an NP predicate and (R, μ) is called a distributional NP search problem. A distributional NP problem (R, μ) , where μ is dominated by a p-time computable distribution, is said to be complete if any other distributional NP search problem (R', ν) with ν dominated by a p-time computable distribution is reducible to (R, μ) . The search version of the distributional halting problem is complete for distributional NP search problems.

4 Randomization

Finding more average-case NP-complete problems has been a central issue in research. The notions of p-time reductions and ap-time reductions have played an important role in this study. However, Gurevich [Gur87] observed that NP-complete problems with “flat” distributions (see below for a definition) cannot be complete for DistNP under p-time or ap-time reductions unless $\text{EXP} = \text{NEXP}$, where $\text{EXP} = \text{DTIME}(2^{\text{poly}})$ and $\text{NEXP} = \text{NTIME}(2^{\text{poly}})$. To overcome this obstacle, Venkatesan and Levin [VL88] introduced a new type of reduction using randomized algorithms. An important application of such reductions is the result of Impagliazzo and Levin [IL90] who showed that polynomial-time sampling cannot generate harder instances than picking instances uniformly at random.

4.1 Flat Distributions and Incompleteness

A distribution μ is *flat* if $(\exists \varepsilon > 0)(\forall x)[\mu(x) \leq 2^{-|x|^\varepsilon}]$. Flat distributions are commonly used in practice. For example, the most frequently used model for undirected random graphs without self-loops is $\mathcal{G}(|V|, p)$ with $0 < p < 1$. It consists of all graphs with vertex set V in which vertices are labeled and the edges are chosen independently with probability p , which is a function of $|V|$. So a graph G with vertex set V and l edges in this model occurs with probability $\mu_{\mathcal{G}(|V|, p)}(G) = p^l(1-p)^{\binom{|V|}{2}-l}$. If we assume also that p satisfies $|V|^{-2+\varepsilon} \leq p \leq 1 - |V|^{-2+\varepsilon}$ for some fixed $\varepsilon > 0$, then $\mu_{\mathcal{G}(|V|, p)}$ is flat [Gur91a] since $\mu_{\mathcal{G}(|V|, p)}(G) \leq (1 - |V|^{-2+\varepsilon})^2 \binom{|V|}{2} \approx e^{-|V|^\varepsilon}$. Similar distributions for directed graphs are also flat for the same reason [WB95]. Most of the NP-complete graph problems have other parameters as inputs, but they are bounded either by the number of vertices or by the number of edges. So their distributions are flat. Gurevich [Gur87] showed that dealing with flat distributions to get average-case completeness results using p-time or ap-time reductions is impossible unless $\text{NEXP} = \text{EXP}$.

Theorem 4.1 *No distributional NP problems with flat distributions can be complete for DistNP under ap-time reductions if $\text{EXP} \neq \text{NEXP}$.*

Proof. Suppose a problem $(D, \mu) \in \text{DistNP}$ with a flat distribution μ is complete under ap-time reductions. Then $D \in \text{EXP}$. Let C be an arbitrary decision problem in $\text{NTIME}(2^{p(n)})$ for some polynomial p . For any x , let $x' = x01^{2^{p(|x|)} - (|x|+1)}$. Then $C' = \{x' : x \in C\}$ is in NP. Define ν by $\nu(x') = c|x|^{-2}2^{-|x|}$ for the appropriate constant c and $\nu(y) = 0$ if $y \neq x'$ for any x . Then $(C', \nu) \in \text{DistNP}$, which implies that there is an ap-time reduction f from (C', ν) to (D, μ) . Hence, deciding whether $x \in C$ can be done by deciding whether $f(x') \in D$. Since f is computable in time polynomial on ν -average, computing $f(x')$ can be carried out in time $2^{O(|x|)}$ due to the particular selection of ν . Also, $\nu(x') \leq g(x')\nu_1(x')$ for some ν_1 and $\mu(f(x')) = \sum_{f(z)=f(x')} \nu_1(z)$ for all x' , where g is polynomial on ν -average and so $g(x') \leq 2^{O(|x|)}$ again due to the definition of ν . Thus, $\nu(x') \leq g(x')\mu(f(x'))$ and so $\mu(f(x')) > 2^{-q(|x|)}$ for some polynomial q . Since μ is flat, there is a k such that $|f(x')|^{1/k} \leq -\log \mu(f(x')) < q(|x|)$. This implies that $C \in \text{EXP}$. ■

If reductions f are restricted to be one–one and length-preserving within a polynomial, namely, there is a polynomial p such that for all x , $p(|f(x)|) \leq |x|$, then Wang and Belanger [WB95] showed that the above incompleteness theorem is true without any assumption. We note that all the natural NP-complete problems are indeed complete under such reductions.

Theorem 4.2 *No distributional NP problems with flat distributions can be complete for DistNP under ap-time reductions that are one–one and length-preserving within a polynomial.*

Proof. Suppose to the contrary that there were an ap-time reduction f reducing DH1 to a problem $(D, \mu) \in \text{DistNP}$ with flat distribution μ , where f is one–one and length-preserving within a polynomial. Then μ_{K_1} is (weakly) dominated by $\mu \circ f$. Since μ is flat, there is an $\epsilon > 0$ such that $\mu(x) \leq 2^{-|x|^\epsilon}$ for all but finitely many x . Since f is length-preserving within a polynomial, there is a constant $k > 0$ such that for all $(i, x, 1^n)$, $|f(i, x, 1^n)| > |(i, x, 1^n)|^{1/k}$. When n is sufficiently large and greater than $(|i| + |x|)^{4k/\epsilon}$, $|f(i, x, 1^n)| > n^{1/2k}(|i| + |x|)^{2/\epsilon}$. So $\mu(f(i, x, 1^n)) \leq 2^{-n^{O(1)}} \cdot 2^{-|i|^2} \cdot 2^{-|x|^2}$. Since f is one–one, $f(\mu_{K_1})(f(i, x, 1^n)) = \mu(f(i, x, 1^n))$, which cannot dominate the exponentially larger probability $\mu_{K_1}(i, x, 1^n) > 2^{-|i|}2^{-|x|}n^{-2}$. ■

4.2 Randomized Average Polynomial Time

Distributional problems with flat distributions may still be difficult on average. So one needs a way to identify hard-on-average problems with flat distributions. Since instances under flat distributions have small weights, searching for frequent enough instances of the target problem of a reduction

to dominate frequent instances of the source problem is crucial. Venkatesan and Levin [VL88] made the suggestion of providing reductions with a good random source to help produce instances of the target problem with large enough probability. They showed that a graph edge coloring problem with a flat distribution is complete under a randomized reduction by allowing algorithms to toss coins to determine the next moves. Blass and Gurevich [BG93] conducted a thorough study on such randomized reductions.

We assume that a randomized algorithm does not flip a coin unless the computation requires a random bit. For simplicity, coins are assumed to be unbiased. Randomized algorithms (to solve a problem) are allowed to make errors and produce incorrect outputs on some sequences of random bits. They could also run forever on some other random (infinite) sequences. Let \mathcal{A} be a randomized algorithm. If \mathcal{A} on input x halts and produces a correct output with s being the random sequence it uses, then (x, s) is called a *good* input for \mathcal{A} . Clearly, \mathcal{A} runs deterministically on (x, s) . If \mathcal{A} on input x runs deterministically, then (x, λ) is a good input, where λ denotes the empty string. Let Γ be a set of good inputs for \mathcal{A} and $\Gamma(x) = \{s : (x, s) \in \Gamma\}$. Let μ be an input distribution. If for all x with $\mu(x) > 0$, $\Gamma(x)$ is non-empty, we call Γ a *good-input domain* of \mathcal{A} (with respect to μ). Clearly, no string in $\Gamma(x)$ is a prefix of a different string in $\Gamma(x)$, for, otherwise, the longer string cannot be in $\Gamma(x)$ as the algorithm stops before it can be generated. Let $U_\Gamma(x) = 1 / \sum_{s \in \Gamma(x)} 2^{-|s|}$, which is called the *rarity* function of Γ [BG93]. The smaller the value of the rarity function, the more good random sequences there are for the algorithm. If for all x , $U_\Gamma(x) = 1$, the randomized algorithm produces a correct output with probability 1. For our purpose, we only need to require that the value of $U_\Gamma(x)$ be “reasonable” in the sense that U_Γ is polynomial on average.

Definition 4.1 Let \mathcal{A} be a randomized algorithm and μ an input distribution.

1. Let Γ be a good-input domain of \mathcal{A} . Then Γ is *nonrare* (with respect to μ) if the rarity function U_Γ is polynomial on μ -average. Define μ_Γ by, for all $(x, s) \in \Gamma$, $\mu_\Gamma(x, s) = \mu(x)2^{-|s|}U_\Gamma(x)$.
2. \mathcal{A} runs in time *polynomial on μ -average* if there exists a nonrare good-input domain Γ such that $\sum_{(x,s) \in \Gamma} t^\varepsilon(x, s)|x|^{-1}\mu_\Gamma(x, s) < \infty$ for some $\varepsilon > 0$, where $t(x, s)$ is the running time of \mathcal{A} on input x with random sequence s . \mathcal{A} runs in *polynomial time* if $t(x, s)$ is bounded by a polynomial in $|x|$ for all $(x, s) \in \Gamma$.
3. \mathcal{A} is *almost total* if for all x with $\mu(x) > 0$, $U_\Gamma(x) = 1$.

A good-input domain Γ together with μ_Γ and a zero-size function for $s \in \Gamma(x)$ (i.e., $|s| = 0$) is called a *dilation* in [Gur91a].

If the algorithm is for solving a search problem, the correctness of the output can be verified by evaluating the search predicate. For decision problems, one way to justify the correctness of the output is to make

sure that the input is good. If there are no other means available to justify the output, the good-input domain Γ (with respect to μ) is required to be *certifiable* [BG93], meaning that for every input (x, s) , whether $(x, s) \in \Gamma$ is deterministically decidable in ap-time with respect to distribution $\mu(x)|s|^{-2}2^{-|s|}$. This notion of certifiability is stronger than the one used in [BG93].

By Definition 4.1, an ap-time randomized algorithm on input x produces a correct output with probability $1/U_\Gamma(x)$, which could be small. While this may not seem satisfactory, Blass and Gurevich [BG93] showed that by iterating such an algorithm in an appropriate manner, a correct solution will be produced with probability 1 in average polynomial time. We assume that there is an efficient way to check whether an output of an iteration is correct as discussed above. Since a randomized algorithm \mathcal{A} may run a much longer time on inputs not in the good-input domain and it may not even halt on some bad inputs, a new iteration should start early without waiting for the previous one to stop. One way to carry it out is to use the “round-robin” method. Denote by \mathcal{A}^* the iteration of \mathcal{A} . At stage n of \mathcal{A}^* , it independently carries out one step in the first, second, \dots , and the n -th iterations of \mathcal{A} respectively in that order. \mathcal{A}^* stops as soon as one of the iteration stops with a correct output. So \mathcal{A}^* is a randomized algorithm whose sequence of random bits on input x is the combination of random bits of each iteration in the round-robin fashion on the same input. This is equivalent to saying that \mathcal{A}^* flips a coin only when an iteration asks for one and passes the random bit to that iteration. Several other iteration schemes have also been studied in [BG93, WB93b].

Theorem 4.3 *Let \mathcal{A} be a randomized algorithm with input distribution μ . Then \mathcal{A} runs in time polynomial on μ -average iff \mathcal{A}^* runs in time polynomial on μ -average and is almost total.*

Proof Sketch. Let \mathcal{A} be a randomized algorithm that runs in time polynomial on μ -average. Let Γ be a nonrare good-input domain. Let Γ^* denote the set of all good inputs of \mathcal{A}^* . Then for all x , $U_{\Gamma^*}(x) = 1$. To see this, run \mathcal{A}^* on x and let s_j be the random sequence used by the j -th iteration and r the random sequence used by \mathcal{A}^* . Since all these s_j are independent, $\Pr[r: s_1, s_2, \dots, s_n \notin \Gamma(x)] = (1 - U_\Gamma(x))^n \rightarrow 0$ (as $n \rightarrow \infty$), where n is the number of iterations. Let $k(x, r)$ be the smallest j with $s_j \in \Gamma(x)$ (namely, the j th iteration is a “successful” one). Let $t(x, r)$ be the time taken by \mathcal{A} on input x with random sequence $s_{k(x, r)}$ and $T(x, r)$ be the running time of \mathcal{A}^* . Then $T(x, r)$ is bounded by a quadratic polynomial of $k(x, r) + t(x, r)$. We will show that $k(x, r)$ and $t(x, r)$ are average polynomials in the next two paragraphs. It follows that \mathcal{A}^* runs in time polynomial on μ -average. (The proof of the other direction is left to the reader.)

Let $\mathbf{E}(X)$ denote the expectation of the random variable X . Let $U(x) = U_\Gamma(x)$ for simplicity. Let $p = 1/U(x)$, then $\mathbf{E}_r(k(x, r)) = \sum_{i=1}^{\infty} i(1-p)^{i-1}p = 1/p$. Since Γ is non-rare, there exists $\delta \in (0, 1)$ such that

$\mathbf{E}_x(U^\delta(x)|x|^{-1}) < \infty$. By Jensen's inequality, if f is an increasing concave function on the interval $(0, \infty)$, then for every random variable X with values in $(0, \infty)$, $\mathbf{E}(f(X)) \leq f(\mathbf{E}(X))$. Clearly, y^δ is an increasing concave function. Thus, $\mathbf{E}_{x,r}(k^\delta(x, r)|x|^{-1}) = \mathbf{E}_x(|x|^{-1}\mathbf{E}_r[k^\delta(x, r)]) \leq \mathbf{E}_x(|x|^{-1}[\mathbf{E}_r(k(x, r))^\delta]) = \mathbf{E}_x(U^\delta(x)|x|^{-1}) < \infty$.

By assumption, $\mathbf{E}_{(x,s) \in \Gamma}(t^\varepsilon(x, s)|x|^{-1}) < \infty$ for some $\varepsilon > 0$. In the following, (x, r) is always assumed to be in Γ^* , and s is always assumed to be in $\Gamma(x)$. Note that for all x , $\mathbf{E}_r(t^\varepsilon(x, s_{k(x,r)})) = \sum_{j=1}^{\infty} (\mathbf{Pr}[k(x, r) = j] \cdot \mathbf{E}_r(t^\varepsilon(x, s_j)|k(x, r) = j))$. The event $k(x, r) = j$ here is the intersection of the event $s_j \in \Gamma(x)$ and all events $s_i \notin \Gamma(x)$ with $i < j$. Since they are all independent, $\mathbf{Pr}[k(x, r) = j] = (1 - 1/U(x))^{j-1} \cdot 1/U(x)$. Since $t^\varepsilon(x, s_j)$ is independent of the events $s_i \notin \Gamma(x)$ with $i < j$, $\mathbf{E}_r(t^\varepsilon(x, s_j)|k(x, r) = j) = \mathbf{E}_r(t^\varepsilon(x, s_j)|s_j \in \Gamma(x))$. Since $\mathbf{E}_r(t^\varepsilon(x, s_j)|s_j \in \Gamma(x)) = \mathbf{E}_s t^\varepsilon(x, s) = \sum_s t^\varepsilon(x, s) 2^{-|s|} U(x)$, $\mathbf{E}_r(t^\varepsilon(x, r)) = \sum_{j=1}^{\infty} [(1 - 1/U(x))^{j-1} \sum_s t^\varepsilon(x, s) 2^{-|s|}] = U(x) \sum_s t^\varepsilon(x, s) 2^{-|s|} = \mathbf{E}_s t^\varepsilon(x, s)$. Thus, $\mathbf{E}_{x,r}(t^\varepsilon(x, r)|x|^{-1}) = \mathbf{E}_{(x,s) \in \Gamma}(t^\varepsilon(x, s)|x|^{-1}) < \infty$. \blacksquare

4.3 Randomizing Reductions and Completeness

The following definition [Gur91a, BG93, BG95] is an elaboration of a definition in [VL88].

Definition 4.2 Let (A, μ) and (B, ν) be distributional decision problems. Then (A, μ) is *ap-time randomly reducible to* (B, ν) if there is a reduction f , computable by a randomized algorithm in time polynomial on μ -average with a certifiable, nonrare good-input domain Γ , such that for all $(x, s) \in \Gamma$, $x \in A$ iff $f(x, s) \in B$, and μ_Γ is weakly dominated by ν with respect to f .

Let (R_1, μ_1) and (R_2, μ_2) be two search problems. Then (R_1, μ_1) is *ap-time randomly reducible to* (R_2, μ_2) if there is a reduction f computable by a randomized algorithm in time polynomial on μ_1 -average with a nonrare good-input domain Γ , and a function g computable in polynomial time satisfying the following conditions: (1) $(\forall(x, s) \in \Gamma)[x \in Y_1$ iff $f(x, s) \in Y_2]$, where $Y_i = \{x : \mu_i(x) > 0 \text{ and } (\exists w) R_i(x, w)\}$, $i = 1, 2$; (2) $(\forall(x, s) \in \Gamma)(\forall w)[R_2(f(x, s), w) \rightarrow R_1(x, g(x, s, w))]$; and (3) μ_Γ is weakly dominated by μ_2 with respect to f .

If the reductions f can be computed by p-time randomized algorithms in both cases, then the reductions are called *p-time randomized reductions*.

A distributional decision problem (D, μ) is solvable in *randomized ap-time* if D is decidable by a randomized algorithm in time polynomial on μ -average with a certifiable, nonrare good-input domain. Denote by RAP the class of all such problems.

A distributional search problem (R, μ) is solvable in *randomized ap-time* if there is a randomized algorithm solving R in time polynomial on μ -average.

The randomized reductions defined in Definition 4.2 are closed for randomized ap-time and they are transitive. To see this, we first note that the composition $A = A_1 \circ A_2$ of randomized algorithms uses random sequence $s = s_2 s_1$, where s_2 is the random sequence used by A_2 on input x and s_1 is the random sequence used by A_1 on input $A_2(x, s_2)$. We then note that certifiable good-input domains compose. Finally, we note that randomized algorithms are deterministic algorithms on good-input domains. So randomized reductions are deterministic reductions on good-input domains and so similar to Lemma 3.6, it is straightforward to show the following lemma for distributional decision problems. The lemma is also true for search problems.

Lemma 4.4 (1) *If (A, μ) is ap-time randomly reducible to (B, ν) and (B, ν) is in RAP, then so is (A, μ) .* (2) *The ap-time randomized reductions are transitive.*

Using ap-time randomized reductions, Venkatesan and Levin [VL88] showed that a graph edge coloring problem with a flat distribution is complete for distributional NP search problems, and Blass and Gurevich [BG95] showed that a matrix decomposition problem with a flat distribution is complete for DistNP. Some more distributional problems are shown to be average-case NP-complete under randomized reductions in [VR92]. Due to page limitations, exposition of these results will be given in a separate paper [Wan]. To demonstrate the idea of using randomized reductions, we show that a halting problem with a flat distribution is complete for DistNP under p-time randomized reductions. We then show, in the next section, that polynomial-time sampling does not generate harder instances than uniformly picking instances at random.

The DISTRIBUTIONAL HALTING PROBLEM (VERSION 2) consists of binary strings (i, x, t) as instances. The question is to decide whether M_i accept x within $|t|$ steps. Let K_2 denote the set of positive instances. The probability $\mu_{K_2}(i, x, t)$ of instance (i, x, t) (positive or negative) is proportional to $2^{-(l+m+n)} l^{-2} m^{-2} n^{-2}$, where $l = |i|$, $m = |x|$, and $n = |t|$. Clearly K_2 is NP-complete and μ_{K_2} is flat. Let $\text{DH2} = (K_2, \mu_{K_2})$.

Theorem 4.5 *DH2 is complete for DistNP under p-time randomized reductions.*

Proof. DH1 is reducible to DH2 by a p-time randomized reduction as follows. On input $(i, x, 1^n)$ it flips a coin n times to produce a random string s , and then outputs (i, x, s) . More precisely, we define a good-input domain Γ by $\Gamma = \{(y, s) : y = (i, x, 1^n) \text{ and } |s| = n\}$. Clearly, the rarity function $U_\Gamma(x) = 1$ and Γ is p-time computable and so is certifiable. Let $\sigma_i((x_1, x_2, x_3)) = x_i$ for $i = 1, 2, 3$. For all $(y, s) \in \Gamma$, let $f(y, s) = (\sigma_1(y), \sigma_2(y), s)$. Then f is one-one and p-time computable. Clearly, for all $(y, s) \in \Gamma$: $y \in K_1$ iff $f(y, s) \in K_2$. To check the domi-

nation property, we note that $\mu_\Gamma(y, s) = \mu_{K_1}(y)2^{-|s|} = \mu_{K_2}(f(y, s))$. Thus, f is a p-time randomized reduction from DH1 to DH2. ■

The randomized reduction constructed above is one–one and length-preserving within a polynomial. This indicates that, by Theorem 4.2, p-time randomized reductions are strictly more powerful than ap-time deterministic reductions.

4.4 Polynomial-Time Sampling

Instances can be sampled with certain probabilities. Ben-David, Chor, Goldreich, and Luby [BCGL92] gave the following definition.

Definition 4.3 A distribution μ is *p-samplable* if there exists a randomized algorithm S such that S takes no input (but tosses coins) and outputs x , if it converges, with probability $\mu(x)$ in time polynomial in $|x|$.

Given a distribution μ , if there is a deterministic algorithm that on input x outputs $\mu^*(x)$ in time polynomial in $|x|$, then μ is p-samplable [BCGL92] as follows. The sampling algorithm picks a truncated real $\gamma \in (0, 1)$ uniformly at random. It then finds, via binary search, the unique string x with $\mu^*(x^-) < \gamma \leq \mu^*(x)$. It outputs x if $|\gamma| \leq -\log(\mu^*(x) - \mu^*(x^-))$. Let S be the set of such γ such that no string in S is a prefix of a different string in S . Then the probability of sampling x is equal to $\sum_{\gamma \in S} 2^{-|\gamma|} = \mu(x)$. However, there are p-samplable distributions which are not p-computable if certain one-way functions exist [BCGL92]. A one-way function is, loosely speaking, a p-time computable function which is hard to invert on most instances.

Ben-David *et. al.* [BCGL92] showed that all p-samplable distributions can be effectively “enumerated” in a certain way to construct a universal p-samplable distribution ξ such that for any p-samplable distribution μ , (K_1, μ) is p-time reducible to (K_1, ξ) . They then showed that for any NP-complete problem A , if K_1 is p-time reducible to A via a reduction that is length-preserving within a polynomial (any natural NP-complete problem satisfies this requirement), then there is a p-samplable distribution ξ_A , obtained from ξ , such that (K_1, ξ) is p-time reducible to (A, ξ_A) . However, ξ depends heavily on an effective enumeration of all p-samplable distributions and so it is not clear what impact it may have on learning whether an NP-complete problem is difficult on average under a distribution encountered in practice. Moreover, using such a distribution to sample an instance is less convenient than simply picking one uniformly at random. Remarkably, Impagliazzo and Levin [IL90] showed that uniformly picking instances at random is all that is needed for generating hard instances. In particular, they showed that any NP search problem with a p-samplable distribution is p-time randomly reducible to an NP search problem with the default uniform distribution. So any complete distributional NP search problem is

also complete for NP search problems with p-samplable distributions. Blass and Gurevich [BG93] provided the following nice exposition on the main idea of the proof.

A sampling algorithm is *length preserving* if it uses exactly n coin tosses to produce a string of length n . It can be shown that every search problem with a p-samplable distribution is reducible to a search problem on instances sampled by a length-preserving algorithm [VL88, BCGL92].

There are two simple but unsuccessful ways in attempting to reduce a source problem with a p-samplable distribution μ to a target problem with a uniform distribution. Let S be a length-preserving sampling algorithm for μ . One may simply let the source problem be the target problem and use the identity function as a reduction. The problem of this approach is that some instance x with small uniform probability could have large probability $\mu(x)$, which could cause domination to fail. This happens when $\sum_{S(r)=x} 2^{-|r|}$ is too large. Another way to obtain a target problem is to use a reduction that reduces an instance x of the source problem to an instance r of the target problem such that $S(r) = x$. This can be done by using a randomized reduction which, on input x , randomly selects a string r of length $|x|$ and then checks whether $S(r) = x$. This method meets the domination requirement but it fails if $\sum_{S(r)=x} 2^{-|r|}$ is too small. The idea of the proof is to interpolate between these two methods, leaning toward the first when $\sum_{S(r)=x} 2^{-|r|}$ is small and toward to second when $\sum_{S(r)=x} 2^{-|r|}$ is large. Let $S^{-1}(x) = \{r : S(r) = x\}$. An instance s of the target problem should be approximately $n = |x|$ bits long for an instance x of the source problem, consisting of $l = \lceil \log(|S^{-1}(x)|) \rceil$ bits of information about some $r \in S^{-1}(x)$ and $n - l$ bits of information about x .

Some difficulties occur when implementing this idea. First, there is no efficient way to compute l from x . But one can randomly guess l , which lies between 0 and n . The probability of a correct hit is therefore the reciprocal of a polynomial which is sufficient to provide a nonrare good-input domain. Second, we do not know how to select the right bits of information about r and x . This can be solved again by randomization. We randomly choose two matrices L and M on the two-element field $\{0, 1\}$ of appropriate size to hash r and x to vectors Lr and Mx of lengths l and $n - l$, where r and x are also viewed as vectors. Randomly selected hash matrices have a reasonable probability. Thus, an instance of the target problem will be (l, L, Lr, M, Mx) with a witness (r, w) if w is a witness of $x = S(r)$ for the source problem.

Theorem 4.6 *Let (R_1, μ_1) be an NP distributional search problem, where μ_1 is p-samplable. Then (R_1, μ_1) is p-time randomly reducible to an NP distributional search problem (R_2, μ_2) , where μ_2 is the default uniform distribution.*

Proof. Let x be an instance of R_1 and $w_1(x) = \{w : R_1(x, w) \text{ holds}\}$. Let S be a length-preserving p-time sampling algorithm for μ_1 . Let $n = |x|$ and

$l = \lceil \log |S^{-1}(x)| \rceil$. Let r range over binary strings of length n , r' range over binary strings of length l . They are also viewed as $\{0, 1\}$ vectors. Let L and M range over $l \times n$ and $(n+1-l) \times n$ matrices, respectively, over the field of two elements $\{0, 1\}$. We assume that these matrices are written in binary by writing down the first row of L , then the second row, and so on. Denote by Lr (similarly Mx) the binary string obtained by multiplying L and r . Assume that integers are represented in the shortest binary notation.

Denote by $*$ the concatenation operation for strings. For any binary string y , if there are x , l , and r such that $y = l * L * r' * M * x'$, and L has full rank (*i.e.* rank l) with $Lr = r'$, $S(r) = x$, and $Mx = x'$, then set $w_2(y)$ to be the set of such (x, l, r, w) , where $w \in w_1(x)$; otherwise, define $w_2(y)$ to be empty. For any binary string z , define $R_2(y, z)$ to be true if $z \in w_2(y)$, and false otherwise. Since R_1 is an NP predicate and S is p-time computable, R_2 is an NP predicate as well.

We now define a randomized reduction from (R_1, μ_1) to (R_2, μ_2) . Let Γ be a set of (x, s) satisfying the following conditions:

1. $s = l * L * r' * M$;
2. $w_2(s * Mx) \neq \emptyset$; and
3. $\forall u \notin S^{-1}(x)$ with $|u| = n$, either $Lu \neq r'$ or $M(S(u)) \neq Mx$.

For all $(x, s) \in \Gamma$, define $f(x, s) = s * Mx$. Define g by $g(x, l, r, w) = w$. Then g is p-time computable and $R_2(f(x, s), (x, l, r, w)) \rightarrow R_1(x, g(x, l, r, w))$. Clearly, f is one-one (by Condition 3 above) and p-time computable on Γ . Note that $\mu_\Gamma(x, s) = \mu_1(x)2^{-|s|} = 2^{-|x|} |S^{-1}(x)| 2^{-|s|}$ which is proportional to $2^{-n+l-|s|}$, and $\mu_2 \circ f(x, s) = 2^{-|s|-(n+1-l)} / (|s| + (n+1-l))^2$. Hence, the domination condition is satisfied.

We have only left to show that Γ is nonrare. The rarity function of Γ is $U(x) = 1 / \sum_{s \in \Gamma(x)} 2^{-|s|} = 2^{|l|+ln+l+(n+1-l)n} / |\Gamma(x)|$. Let $E = \{(L, r') : L \text{ has full rank } l \text{ and } (\exists r \in S^{-1}(x)) [Lr = r']\}$. Let $\pi_1 = |E|$. For each $(L, r') \in E$, let π_2 be the number of M satisfying Condition 3. Then $|\Gamma(x)| = \pi_1 \cdot \pi_2$. We claim that (1) $\pi_1 > \frac{3}{32} 2^{ln+l}$, and (2) $\pi_2 \geq \frac{1}{2} 2^{(n+1-l)n}$. By these two claims and the fact that $2^{|l|} \leq 2l \leq 2n$, it follows that $U(x) < 43|x|$, and so Γ is nonrare.

To show the first claim, we first note that for a full-rank matrix, only the 0-row (meaning the row of 0's) cannot serve as the first row; given the first row, only the first row and the 0-row cannot serve as the second row of a full-rank matrix; given the first two rows, only four rows, namely, the four linear combinations of the first two rows, cannot serve as the third row, and so on. Thus, the number Δ of full-rank matrices L is $\prod_{i=0}^{l-1} (2^n - 2^i) = 2^{ln} \prod_{i=0}^{l-1} (1 - 2^{i-n}) > 2^{ln} \prod_{i=0}^{\infty} (1 - \frac{1}{2^i}) \geq \frac{1}{4} 2^{ln}$.⁸ Let r and u range over $S^{-1}(x)$ and $E(r) = \{(L, Lr) : L \text{ is of full rank}\}$. Then $|E(r)| = \Delta$,

⁸To see that $\prod_{i=0}^{\infty} (1 - 2^{-i}) \geq 1/4$, it suffices to note that $\prod_{i=2}^{\infty} (1 - 2^{-i}) > 1 - \sum_{i=2}^{\infty} 2^{-i} = 1/2$.

and $E = \bigcup_r E(r)$. Hence, $\pi_1 = |\bigcup_r E(r)|$. By the principle of inclusion and exclusion, $|\bigcup_r E(r)| \geq (\sum_r |E(r)| - \sum_{r \neq u} |E(r) \cap E(u)|)$. Let $m = |S^{-1}(x)|$, then $\sum_r |E(r)| = m\Delta$. Let $s(r, u) = |E(r) \cap E(u)|$. We are to show that $s(r, u) \leq \Delta 2^{-l}$ if $r \neq u$. Note that $(L, r') \in E(r) \cap E(u)$ iff $Lr = r' = Lu$, and for each full-rank L with $Lr = Lu$, there exists a unique r' such that $(L, r') \in E(r) \cap E(u)$. Thus, $s(r, u)$ is equal to the number of full-rank L with $L(r - u) = 0$. If $l = n$, then $Lr \neq Lu$ for any full-rank L and so $s(r, u) = 0$. Assume $l < n$. Note that for any nonzero binary vectors z, z' of length n , there is a nonsingular $n \times n$ matrix A such that $Az = z'$. Then $Lz' = 0$ iff $(LA)z = 0$, and LA has full rank iff L does. This means that $s(r, u)$ is equal to the number of full-rank L with $Lz = 0$ for any particular choice of nonzero z . Let $z = (0, 0, \dots, 0, 1)$. Then $s(r, u)$ is equal to the number of full-rank $l \times n$ matrices where the last column is 0. It follows that $s(r, u)$ is equal to the number of full-rank $l \times (n - 1)$ matrices, which is equal to $\prod_{i=0}^{l-1} (2^{n-1} - 2^i) \leq \Delta 2^{-l}$. So $\sum_{r \neq u} |E(r) \cap E(u)| \leq \frac{1}{2}m(m-1)\Delta 2^{-l} = m(m-1)\Delta 2^{-(l+1)}$. Thus, $\pi_1 = |\bigcup_r |E(r)| \geq m\Delta - m(m-1)2^{-(l+1)} > \Delta 2^l (t - \frac{1}{2}t^2)$, where $t = m/2^l \in [\frac{1}{2}, 1]$ since $l = \lceil \log m \rceil$ by definition. The minimum of this quadratic function on $[\frac{1}{2}, 1]$ is $\frac{3}{8}$. Thus, $\pi_1 \geq \frac{3}{8}\Delta 2^l > \frac{3}{32}2^{ln+l}$.

To show the second claim, we note that for each fixed (L, r') , the number of u satisfying $Lu = r'$ is 2^{n-l} . Let $k = n + 1 - l$. For each $u \notin S^{-1}(x)$ with $|u| = n$, there are exactly 2^{kn-k} matrices M satisfying the equation $M(S(u) - x) = 0$. This is true because if the i -th bit of $S(u) - x$ is 1 (such a bit exists since $S(u) \neq x$), then the value of the i -th bit of a row ρ can be chosen, depending on the values of the other $n - 1$ bits, to make $\rho(S(u) - x) = 0$. Thus, $\pi_2 \geq 2^{kn} - 2^{n-l}2^{kn-k} = \frac{1}{2}2^{kn}$. ■

Directly applying the above proof to decision problems is difficult since Γ does not seem to be certifiable. We will show, in the next section, that any distributional NP search problem with a p-time computable distribution is p-time randomized truth-table reducible to a distributional NP decision problem in DistNP. It follows that Theorem 4.6 is true for decision problems under p-time randomized truth-table reductions.

4.5 Randomized Turing Reductions

All of the reductions we have studied so far are variants of many-one reductions. Other types of reductions such as truth-table reductions and Turing reductions can be similarly defined for distributional problems. The reader should find no difficulty in defining them with the desired properties. It is a long-standing open problem whether Turing reductions (or truth-table reductions) are provably more powerful than many-one reductions on NP problems. Not much has been done for distributional NP problems. On the other hand, it is not known whether truth-table or Turing reductions can help identify more average-case NP-complete problems encountered in prac-

tice. For one thing, we note that all the known natural NP-complete problems are indeed many-one complete. Nevertheless, Turing reductions are the only known choice to bridge the gap between search problems and decision problems in the worst-case complexity. For distributional problems, randomized truth-table reductions (defined below) are sufficient [BCGL92].

Definition 4.4 Let (A, μ) and (B, ν) be two distributional (decision or search) problems. (A, μ) is *ap-time randomly Turing reducible to* (B, ν) if there is a probabilistic oracle Turing machine M with the following properties.

1. (Efficiency) M runs in p-time on μ -average in the following strong sense. Let $t(x, r)$ be the running time of M on input x and internal coin tosses r , then $(\exists \varepsilon > 0) \sum_{x,r} t^\varepsilon(x, r) |x|^{-1} \mu(x) 2^{-|r|} < \infty$.
2. (Validity) There exists a constant $c > 0$ such that for all x , $\Pr(M^B(x) = A(x)) \geq c$, where M^B is the random variable, determined by M 's internal coin tosses, for outputs of M on input x with access to oracle for B .
3. (Domination) There is a polynomial p such that for every y , $\sum_x \text{Ask}(x, y) \mu(x) \leq p(|y|) \nu(y)$, where $\text{Ask}(x, y)$ is the probability, taken over M 's internal coin tosses, that M asks query y on input x .

If $t(x, s)$ is bounded by a polynomial in $|x|$, then the reduction is called a *p-time randomized Turing reduction*.

By Theorem 4.3, an ap-time randomized algorithm that produces a correct output with probability at least c can be iterated to get a correct output with probability 1. We also note that Theorem 4.1 remains true if one uses deterministic Turing reductions [Gur91a].

If in a p-time (ap-time) randomized Turing reduction, all queries can be generated at the beginning independent of other queries, then the reduction is called a p-time (ap-time) *randomized truth-table* reduction. Randomized Turing reductions are closed for randomized ap-time solvable problems and are transitive in the special case where on input x only queries of length at least $|x|^c$ are asked, for a constant $c > 0$ [BCGL92].

As pointed out in [BCGL92], the standard Turing reduction of a search problem to the corresponding decision problem, which asks queries to the set of prefixes of a solution on input x , fails on domination when the query string y is too long. This is because y has distribution $\mu(x) 2^{-|y|} / |y|^2$ with input distribution μ , which will be too small to dominate $\mu(x)$ when $|y|$ is large. However, if the search problem has a unique solution, then instead of asking for prefixes of a solution, one can ask non-adaptively and deterministically for the i -th bit of the solution with probability $\mu(x) |x|^{-O(1)}$. The domination condition is therefore satisfied. Based on this idea, Ben-David *et al* [BCGL92] obtained the following result.

Theorem 4.7 *Let (R, μ) be a distributional NP search problem. Then there is a distributional NP decision problem (D, ν) such that (R, μ) is p-*

time randomly truth-table reducible to (D, ν) , and if μ is p -time computable then so is ν .

Proof Sketch. For simplicity, assume that $(x, w) \in R$ implies $|x| = |w|$. Let $H_{|x|,k}$ be a set of standard universal hash functions from strings of length $|x|$ to strings of length k (e.g., $n \times k$ matrices over the finite field $\{0, 1\}$). Define D as follows. An instance will be a binary string y and an integer i , and an instance will be in D if there exists a w such that $y = (x, k, h, \theta)$, $(x, w) \in R$, $h \in H_{n,k}$, $h(w) = \theta$, and the i -th bit of w is 1. The distribution ν is defined by $\nu(y, i) = \mu(x)|x|^{-2}|H_{|x|,k}|^{-1}2^{-k}$ if $y = (x, k, h, \theta)$ and $1 \leq i \leq |x|$; and 0 otherwise. Construct a truth-table reduction M as follows. On input x , for every value of $k \in \{1, 2, \dots, |x|\}$, M first selects uniformly at random an h in $H_{|x|,k}$ and a θ of length k . Let $y = (x, k, h, \theta)$. M then makes queries $(y, 1), (y, 2), \dots$, and $(y, |x|)$ to D and outputs $D(y, 1)D(y, 2) \cdots D(y, |x|)$. Clearly, M runs in time polynomial in $|x|$. To check the validity, let $m = |\{w : (x, w) \in R\}|$ and assume $m \geq 2$ (the case that $m \leq 1$ is straightforward). Let $k = \lfloor \log m \rfloor$ if $m \leq \frac{4}{3}2^{\lfloor \log m \rfloor}$ and $k = \lceil \log m \rceil$ otherwise. Then $\frac{2}{3} \leq m2^{-k} \leq \frac{4}{3}$. It follows that for a randomly selected $h \in H_{|x|,k}$ and θ of length k , the expected number of w satisfying $R(x, w)$ is between $\frac{2}{3}$ and $\frac{4}{3}$. By Markov's inequality and the property of universal hashing that pairs of hashing points are pairwise independent, the probability that there is a unique w such that $h(w) = \theta$ is $\geq c$ for some constant $c > 0$. In such a case M returns a correct answer w . To check the domination condition, it suffices to note that $((x, k, h, \theta), i)$ appears as a query with probability $|H_{|x|,k}|^{-1}|x|^{-1}2^{-k}\mu(x)$. ■

5 Hierarchies of Average-Case Complexity

Hartmanis and Sterns [HS65] showed that, for multi-tape Turing machines, if $t(n) \log t(n) = o(T(n))$ and both t and T are time-constructible, then $\text{DTIME}(t(n))$ is a proper subset of $\text{DTIME}(T(n))$. This is the best hierarchy known for the deterministic classes in the worst case.⁹ Average-case hierarchies have been investigated within several different frameworks. One approach is to study, for functions T and t with $t < T$, whether there is a language in $\text{DTIME}(T(n))$ such that a *very* large portion of instances x will require more than $t(x)$ to compute. This approach has a long history (e.g., see [GGH94]). Among others, the following result of Geske, Huynh, and Seiferas [GHS91] says that for every fully time-constructible function T , there is a set $L \in \text{DTIME}(O(T(n)))$ such that for every function t , if $t(x) \log t(x) = o(T(x))$, then every Turing machine that accepts L requires

⁹Note that such results are machine dependent. For example, the linear speed-up theorem that holds for multi-tape Turing machines does not hold for some other models [Jon93].

more than $t(x)$ steps for all but finitely many input x . It follows that there is an expected time hierarchy which is independent of distributions and is as tight as the Hartmanis-Sterns hierarchy for worst-case deterministic time.

In a different approach, Li and Vitányi [LV92] showed that under the universal distribution with probability $\mathbf{m}(x) = 2^{-K(x)}$, where $K(x)$ is the prefix variant of Kolmogorov complexity [Gac74], the expected time of any problem on strings of the same length is the same as the worst-case complexity on that length. So under the universal distribution, any hierarchy results for $\text{DTIME}(t(n))$, however tight, will also apply to the expected-time complexity classes of time t . This line of research was extended by Miltersen [Mil91]. However, the distributions used in these results are either not computable (e.g., \mathbf{m}), or require super-polynomial time to compute.

To study average-case hierarchies in the framework of Definition 2.1, a finer distinction between average time, namely, not just between average polynomial and super-polynomial, is needed. Recall that the notion of average polynomial time is defined on purpose not to distinguish t time from t^k time to guarantee machine and encoding independence. For a fixed computation model, however, we can distinguish t time from t^k time. Ben-David *et. al.* [BCGL92] suggested a direct generalization of Levin's definition. Levin's definition can be rephrased as saying that a function f is polynomial on average if there exists a linear on average function ℓ and a $k > 0$ such that for all x , $f(x) \leq \ell(x)^k$, where ℓ is linear on average if $\sum_x \ell(x)|x|^{-1}\mu(x) \leq \infty$. Ben-David *et. al.* defined a function t to be T on average if $t(x) = T(\ell(x))$ for some linear on average function ℓ . This can be rephrased as follows.

Definition 5.1 Let μ be a distribution, and let $t : \Sigma^+ \rightarrow \mathbb{N}^+$ and $T : \mathbb{N}^+ \rightarrow \mathbb{N}^+$ be functions such that T^{-1} is defined. Then t is T on μ -average if $\sum_x T^{-1}(t(x))|x|^{-1}\mu(x) < \infty$.

Denote by $\text{AvDTime}(T(n))$ the class of all distributional decision problems (D, μ) such that D can be decided by a deterministic algorithm whose running time is T on μ -average. Since the definition of linear on average does not make the distinction between n on average and cn on average for any constant c , $T(n)$ on average is the same as $T(cn)$ on average for any function T and any constant $c > 0$. Thus, it is not possible to get too fine of separation results. When T is polynomially bounded, however, the distinction between time $T(n)$ and time $T(cn)$ also does not exist for the worst-case deterministic time because of the linear speedup theorem. It is therefore possible to prove an average-case hierarchy result as tight as in the worst case. Cai and Selman [CS95] obtained such a result when time functions are restricted to be *logarithmico-exponential*.

The class \mathcal{L} of logarithmico-exponential (log-exp, in short) functions, first studied by Hardy [Har11], is the smallest class of functions $f : \mathbb{R} \rightarrow \mathbb{R}$ con-

taining every constant function $f(x) = c$ and the identity function $f(x) = x$ such that if $f(x)$ and $g(x)$ are in Γ , then so are $f(x) - g(x)$, $\exp(f(x))$ (i.e., $e^{f(x)}$), and $\ln f(x)$ (if $f(x)$ is eventually positive). It follows that $f(x) + g(x)$, $f(x) \cdot g(x)$, and $f(x)/g(x)$ are also in \mathcal{L} . It was shown in [Har24] that every function in \mathcal{L} is either eventually positive or eventually negative or identically zero. It follows that every function in \mathcal{L} is eventually monotonic by the fact that \mathcal{L} is closed under differentiation. Define $f^{(l+1)}(x) = f \circ f^{(l)}(x)$ ($l \geq 1$) and $f^{(1)}(x) = 1$. It was shown in [Har11] that if $t \in \mathcal{L}$ tends to infinity, then there is a constant l such that $\log^{(l)}(x) = o(f(x))$ as well as $f(x) = o(\exp^{(l)}(x))$. A function T is a log-exp time function if for some $f \in \mathcal{L}$ and for all $n \in \mathbb{N}$, $T(n) = \lfloor f(n) \rfloor$. It is easy to see that almost all commonly used time functions are log-exp [CS95]. (There are some exceptions, e.g., the time function $T(n) \log^* T(n)$ is not log-exp.)

Theorem 5.1 *Let $t, T : \mathbb{N} \rightarrow \mathbb{N}$ be log-exp time functions such that t is bounded above by a polynomial and T is fully time constructible. If $t(n) \log t(n) = o(T(n))$, then there exists a language L and a uniform distribution μ such that $(L, \mu) \in \text{AvDTime}(T(n)) - \text{AvDTime}(t(n))$.*

Proof Sketch. Assume $1/t(n) = o(1)$. Let $\alpha_n = T(n)/[t(n) \log t(n)]$ and $\beta_n = \log t(n)/\log \log t(n)$. Let $B_n = \sqrt{\alpha_n \beta_n / (\alpha_n + \beta_n)}$. Then $\min(\alpha_n, \beta_n)/2 \leq B_n^2 \leq \min(\alpha_n, \beta_n)$, and so $1/B_n = o(1)$. Also $B_n = o(\alpha_n)$ and $B_n = o(\beta_n)$. Let $S(n) = B_n \cdot t(n)$. It follows that $S(n) \log S(n) = o(T(n))$. Thus, there is a language $L \in \text{DTIME}(T(n))$ such that any Turing machine that accepts L requires more than $S(|x|)$ steps for all but finitely many input x . Clearly, $(L, \nu) \in \text{AvDTime}(T(n))$ for every distribution ν . Let k be an integer such that $t(n) = o(n^k)$. Then $t(n)/n^k$ is eventually monotonic decreasing. It follows that for all $a > 1$ and all sufficiently large n , $t(an) \leq a^k t(n)$, and so $t(B_n^{1/k} \cdot n) < B_n \cdot t(n)$. Since t is eventually monotonic increasing, $t^{-1}(S(n)) \geq B_n^{1/k} \cdot n$ for sufficiently large n . Let $b_n = B_n^{1/k}$, then b_n is log-exp. So there is a constant l such that $\log^{(l)}(n) = o(b_n)$. For this l , there exists an n_l such that when $n \geq n_l$, $1/(n \log n \log \log n \cdot (\log^{(l)} n)^2)$ is defined and we denote it by a_n . Define $a_n = 1$ for $n < n_l$. Then $s = \sum_n a_n < \infty$ but $\sum_n a_n b_n = \infty$. Let $\mu(x) = s^{-1} a_{|x|} 2^{-|x|}$. It follows that $(L, \mu) \notin \text{AvDTime}(t(n))$. ■

When T is exponential or higher, the fact that there is no distinction between $T(n)$ and $T(cn)$ in Definition 5.1 can produce counterintuitive results. For example, it is easy to see that $\text{AvDTime}(4^n) = \text{AvDTime}(2^n)$ since $\sum_x \log_4 t(x) |x|^{-1} \mu(x) = \frac{1}{2} \sum_x \log_2 t(x) |x|^{-1} \mu(x)$ [CS95], but by the hierarchy result in [GHS91], there is a language in $\text{DTIME}(4^n)$ which requires more than 2^n time to compute on all but finitely many inputs.

Although some hierarchy results can still be obtained [SY92, BW95], one would like to have a general hierarchy theorem as tight as the Hartmanis-Sterns hierarchy for worst-case deterministic time. To achieve such a result, it is necessary to distinguish among cn on average for different val-

ues of c . One way to do so is to define a function ℓ to be n on average if $\sum_x \ell(x)|x|^{-1}\mu(x) \leq 1$, and so we would be comparing ℓ to $|x|$ rather than $c|x|$ for arbitrary c . A function t could then be said to be T on average if $t(x) = T(\ell(x))$ for some n on average function ℓ , *i.e.*, if $\sum_x T^{-1}(t(x))|x|^{-1}\mu(x) \leq 1$. This would then make the distinction between a function being 2^n on average and 4^n on average. However, since an algorithm which solves a decision problem can be given look-up tables to speed up computation on any given finite number of inputs, we would again have $\text{AvDTime}(2^n) = \text{AvDTime}(4^n)$. Requiring that $(\forall n) \sum_{x, |x| \geq n} T^{-1}(t(x))|x|^{-1}\mu_{\geq n} \leq 1$ will remove dependency on any finite number of inputs, where $\mu_{\geq n}$ is the conditional distribution of μ on $\{x: |x| \geq n\}$. It follows that $(\forall n) \sum_{x, |x| \geq n} T^{-1}(t(x))|x|^{-1}\mu(x) \leq \mu[|x| \geq n]$. Cai and Selman [CS95], then, proposed the following definition.

Definition 5.2 Let μ be a distribution and $T: \mathbb{N} \rightarrow \mathbb{N}$ a function such that T^{-1} is defined. A function $t: \Sigma^+ \rightarrow \mathbb{N}$ is *T on μ -average* if for all $n \geq 1$, $\sum_{|x| \geq n} T^{-1}(t(x))|x|^{-1}\mu(x) \leq \mu[|x| \geq n]$.

By the hierarchy result of Geske *et. al.* [GHS91], we immediately get a strong hierarchy result under Definition 5.2. That is, for time-constructible and monotonic increasing functions t and T , if $t(n) \log t(n) = o(T(n))$, then there is a language L such that for any distribution μ , L is computable by a deterministic Turing machine in time T on μ -average, but no deterministic Turing machine can compute L in time t on μ -average. Note that this hierarchy result is independent of distributions.

Clearly, any distributional decision problem that is computable in average polynomial time under Definition 5.2 is in AP. While the converse is not true, Cai and Selman [CS95] showed that there is a partial converse. That is, under a fairly reasonable condition on the distribution μ , called condition W , a distributional decision problem is computable in average polynomial time under Definition 5.2 iff it is in AP. A distribution μ is said to satisfy *condition W* if $(\exists k > 0)(\forall n)[\mu[|x| \geq n] = \Omega(n^{-k})]$. Thus, the theory of reducibility is preserved for average polynomial time under Definition 5.2 when restricted to distributions that satisfy condition W . If condition W is not satisfied, one may get some unwanted results. For example, Belanger and Wang [BW] showed that p-time reductions are not closed for average polynomial time under Definition 5.2. In particular, they constructed two distributional decision problems (A, μ) and (B, ν) , with p-time computable μ and ν , such that (A, μ) is p-time reducible to (B, ν) and (B, ν) is solvable in average polynomial time under Definition 5.2, but (A, μ) is not.

There is yet another different approach to study average-case hierarchies under an entirely different average-case measurement. Reischuk and Schindelbauer [RS93] suggested measuring average-case complexity with respect to the ranking of the input distribution by decreasing weights in-

stead of the individual values. Two distributions μ and ν are said to have the same rank if for all x and y , $\mu(x) \leq \mu(y)$ iff $\nu(x) \leq \nu(y)$. The *ranking function* $\text{rank}[\mu](x)$ of a distribution μ is defined by $\text{rank}[\mu](x) = |\{z \in \Sigma^+ : \mu(z) \geq \mu(x)\}|$. Let $t : \Sigma^+ \rightarrow \mathbb{N}$ and $T : \mathbb{N} \rightarrow \mathbb{N}$ be two functions such that T^{-1} is defined. Then t is T on $\text{rank}[\mu]$ -average if for all real-valued, non-negative, monotone functions m with $\sum_x m(\mu(x)) \leq 1$, $\sum_x T^{-1}(t(x))|x|^{-1}m(\mu(x)) < \infty$. This definition depends only on $\text{rank}[\mu]$ and not on the individual values of each distribution with the same rank, and it turns out to be powerful for obtaining tight hierarchy results. In fact, it delivers the optimal separation result, namely that separation of average time complexity classes can be obtained under the assumption that $t(n) = o(T(n))$ [RS93].

However, it is not known whether there are natural NP-complete problems under commonly used distributions that are solvable in average polynomial time with respect to the rank of the distributions. Also, Belanger and Wang [BW95] showed that no NP problems averaging with respect to p-time computable ranking functions are harder than NP problems averaging under uniform distributions. In particular, they showed that there is an NP problem A such that if A is solvable in polynomial time on μ -average, where $\mu(x) = |x|^{-2}2^{-|x|}$, then every NP problem is solvable in polynomial time on $\text{rank}[\nu]$ -average by a *randomized* algorithm, where ν is any distribution and $\text{rank}[\nu]$ is p-time computable. Note that randomized algorithms are used here for the following reasons. One might attempt to prove this result by considering the function $f(x) = |\{y : \mu(y) < \mu(x)\}|$, which transforms the input distribution into a monotone distribution of the outputs. Under monotone distributions no sets have probability greater (by a super-polynomial factor) than under uniform distribution. However, while f may be p-time computable, f may not transform an NP problem into an NP problem if the outputs of f are super-polynomially shorter than the inputs. Also, while any fixed monotone distribution is bounded by the uniform distribution with a super-polynomial factor, the same factor may not work with every monotone distribution, and all of them have to be taken care of at the same time. These difficulties can be avoided using a randomization technique [BW95].

6 A Brief Survey of Other Results

Average-case complexity has been investigated in various other aspects and a brief survey of these results is presented in this section.

1. NP-Isomorphism Problem with Respect to Random Instances.

The Berman-Hartmanis NP-isomorphism conjecture [BH77] has provided much of the impetus for research in structural complexity theory during the last two decades. It conjectures that all many-one NP-complete prob-

lems are polynomially isomorphic. Following Berman and Hartmanis' isomorphism theory [BH77], Wang and Belanger [WB95] provided a general framework for studying the isomorphism problem for NP-complete sets with respect to random instances. Let (A, μ) and (B, ν) be two distributional decision problems. (A, μ) and (B, ν) are *p-isomorphic* if there is a one-one, onto, p-time computable and invertible reduction f such that (A, μ) is p-time reducible to (B, ν) via f and (B, ν) is p-time reducible to (A, μ) via f^{-1} . By Theorem 4.2 it follows that any NP-complete problem with a flat distribution cannot be p-isomorphic to DH1.

Write $\mu \approx \nu$ if $\mu \preceq \nu$ and $\nu \preceq \mu$. Let f and g be one-one, p-time computable, and invertible reductions of (A, μ) to (B, ν) and (B, ν) to (A, μ) , respectively, such that $f \circ g$ and $g \circ f$ are length-increasing. Assume also that $\mu \approx \nu \circ f$ and $\nu \approx \mu \circ g$. Then (A, μ) is p-isomorphic to (B, ν) [BW93, WB95]. Using this result and the Distribution Controlling Lemma (Lemma 3.3 (2)), Wang and Belanger [WB95] showed that all the known average-case NP-complete problems under p-time reductions are p-isomorphic.

2. Relations to Worst-Case Complexity Classes. It is not known whether DistNP is included in AP. Results in this line will be in the form “DistNP \subseteq AP iff ...”, “DistNP \subseteq AP if ...”, or something of this nature unless there is a major breakthrough. The following two results are of this type and they are related to worst-case complexity classes. It was shown in [Boo74] that $\text{DTIME}(2^{O(n)}) \neq \text{NTIME}(2^{O(n)})$ implies the existence of a tally language D in NP – P. Let $\mu(1^n) = \frac{1}{n(n+1)}$. If DistNP \subseteq AP, then $D \in$ P, a contradiction. Hence, if $\text{DTIME}(2^{O(n)}) \neq \text{NTIME}(2^{O(n)})$, then AP does not include DistNP [BCGL92]. A similar result can also be found from [Gur91a] under the assumption that EXP \neq NEXP.

The following result shows how P = NP may fail using average-case complexity. A set D is *almost in NP with respect to μ* if there is a deterministic Turing machine which accepts D in time T , and D has a subset D' in NP such that $\sum_{x \in D - D'} 2^{\delta T(x)} |x|^{-1} \mu(x) < \infty$ for some constant $\delta > 0$. A distributional decision problem (D, μ) is *hard on positive instances* if for any deterministic Turing machine that accepts D in time T and for any $\varepsilon > 0$, $\sum_{x \in D} T^\varepsilon(x) |x|^{-1} \mu(x) = \infty$. Wang and Belanger [WB93a] showed that P \neq NP iff there is $(D, \mu) \in$ ANP – AP such that D is almost in NP with respect to μ and (D, μ) is hard on positive instances, where μ is exponential-time computable.

3. Problems that Are Easy on Average under a Set of Distributions. One may wonder whether there is a set which is not in P but is in AP under every p-time computable distribution. A diagonalization construction provides an affirmative answer [Sch95a], but it is not known whether there are such natural problems. If a problem is in AP under every exponential-time computable distribution, then it has to be in P [SY92]. As

a remark, we note that no AP problems under p-time computable distributions are harder on average than simply a P problem.¹⁰ More results on structural properties of the class of sets that are in AP under every p-time computable distribution can be found in [SY92, SY95, Sch95b, KS95], and a connection to the measure theory can be found in [Sch96].

A similar approach has also been applied to studying the class of optimization problems. Under the worst-case complexity, $P = NP$ implies that all NP optimization problems are polynomial-time solvable. Schuler and Watanabe [SW95] investigated how much of this can be carried out in the average-case setting. They showed that if every NP decision problem is in AP under every P^{NP} -samplable distribution, then every NP optimization problem is solvable in average polynomial time with respect to every p-time computable distribution, and vice versa.

4. Generating Test Instances for Promise Problems. Suppose one wants to design an algorithm which outputs a satisfying assignment in polynomial time on average if the input is guaranteed to be a satisfiable Boolean formula. The correctness of the algorithm is often justified by testing the algorithm on some randomly generated satisfiable Boolean formulas. Watanabe [Wat94] provided a framework for investigating the difficulty of generating test instances for promise NP search problems and some “completeness” results were shown based on the known average-case NP-complete problems.

5. Other Complexity Measures. The notion of average polynomial time can be generalized to other complexity measures. Among others, sub-polynomial average time has attracted some attention. Log-space average-case measurement has been investigated in [BCGL92]. Another obvious extension is to study the average-case counterparts of ZPP, BPP, P/poly, etc. using the notion of average polynomial time.

6. Optimization Problems. We end this list by asking how average-case complexity may help to obtain results for actual optimization problems. Although certain NP optimization problems cannot have good approximation solutions in deterministic polynomial time unless $P = NP$ (see, e.g. [ALMSS92]), exact solutions, or good approximations could still be obtained in average polynomial time, or some average-case “completeness” results could be shown.

¹⁰To see this, let K_0 be K_1 restricted to deterministic machine M_i , and μ_{K_0} the same as μ_{K_1} . Then clearly, K_0 is complete in P. Also, (K_0, μ_{K_0}) is ap-time complete for AP with respect to p-time computable distributions [WB93a].

Acknowledgments: I thank Leonid Levin for his Complexity Theory Seminar at Boston University in 1987–88 where I first learned average-case complexity and I thank Steven Homer for his encouragement. I am grateful to Jay Belanger for many valuable discussions and a careful reading of this paper. I thank Leonid Levin, Yuri Gurevich, and Ramarathnam Venkatesan for a number of conversations in the past several years which have helped shape my view on average-case complexity. I am grateful to all the comments I received on the early drafts of this paper, and in particular, I thank Jay Belanger, Jin-Yi Cai, Yuri Gurevich, Osamu Watanabe, and an anonymous referee for their constructive comments, which have helped improve my presentation.

7 References

- [ALMSS92] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and hardness of approximation problems. *Proc. of FOCS*, IEEE, pages 14–23, 1992.
- [BW] J. Belanger and J. Wang. Reductions and convergence rates of average time. Manuscript.
- [BW93] J. Belanger and J. Wang. Isomorphisms of NP-complete problems on random instances. *Proc. 8th Structures*, IEEE, pages 65–74, 1993.
- [BW95] J. Belanger and J. Wang. Rankable distributions do not provide harder instances than uniform distributions. *Proc. 1st COCOON*, vol 959 of *Lect. Notes in Comp. Sci.*, pages 410–419, 1995.
- [BCGL92] S. Ben-David, B. Chor, O. Goldreich, and M. Luby. On the theory of average case complexity. *J. Comp. Sys. Sci.*, 44:193–219, 1992. (First appeared in *Proc. 21st STOC*, ACM, pages 204–216, 1989.)
- [BH77] L. Berman and J. Hartmanis. On isomorphisms and density of NP and other complete sets. *SIAM J. Comput.*, 6:305–321, 1977.
- [BG91] A. Blass and Y. Gurevich. On the reduction theory for average-case complexity. *Proc. 4th CSL (Workshop on Computer Science Logic)*, vol 533 of *Lect. Notes in Comp. Sci.*, pages 17–18, 1991.
- [BG93] A. Blass and Y. Gurevich. Randomizing reductions of search problems. *SIAM J. Comput.*, 22:949–975, 1993.
- [BG95] A. Blass and Y. Gurevich. Matrix transformation is complete for the average case. *SIAM J. Comput.*, 24:3–29, 1995.
- [Boo74] R. Book. Tally languages and complexity classes. *Inf. and Control*, 26:213–229, 1974.

- [CS95] J.-Y. Cai and A. Selman. Average time complexity classes. *Elect. Col. Comp. Complexity* TR95-019, 1995.
- [Gac74] P. Gács. On the symmetry of algorithmic information. *Soviet Math. Dokl.*, 15:1477–1481, 1974.
- [GHS91] J. Geske, D. Huynh and J. Seiferas. A note on almost-everywhere complex sets with application to polynomial complexity degrees. *Inf. and Comput.*, 92:97-104, 1991.
- [GGH94] M. Goldmann, P. Grape, and J. Håstad. On average time hierarchies. *Inf. Proc. Lett.*, 49:15–20, 1994.
- [Gur87] Y. Gurevich. Complete and incomplete randomized NP problems. *Proc. 28th FOCS*, IEEE, pages 111–117, 1987.
- [Gur89] Y. Gurevich. The challenger-solver game: variations on the theme of $P =? NP$. *EATCS Bulletin*, pages 112–121, 1989. It has been reprinted in *Current Trends in Theoretical Computer Science* (G. Rozenberg and A. Salomaa eds), World Scientific, pages 245–253, 1993.
- [Gur91a] Y. Gurevich. Average case completeness. *J. Comp. Sys. Sci.*, 42:346–398, 1991.
- [Gur91b] Y. Gurevich. Average case complexity. *Proc. 18th ICALP*, vol 510 of *Lect. Notes in Comp. Sci.*, pages 615–628, 1991.
- [GS87] Y. Gurevich and S. Shelah. Expected computation time for Hamiltonian path problem. *SIAM J. Comput.*, 16:486–502, 1987.
- [Har11] G. Hardy. Properties of logarithmico-exponential functions. *Proc. London Math. Soc.*, 10:54–90, 1911.
- [Har24] G. Hardy. *Orders of Infinity, The 'infinitärrechnen' of Paul du Bois-Reymond*. Vol. 12 of Cambridge Tracts in Math. & Mathematical Physics. Cambridge University Press, 1924.
- [HS65] J. Hartmanis and R. Stearns. On the computational complexity of algorithms. *Trans. Amer. Math. Soc.*, 117:285–306, 1965.
- [Imp95] R. Impagliazzo. A personal view of average-case complexity. *Proc. 10th Structures*, IEEE, pages 134–147, 1995.
- [IL90] R. Impagliazzo and L. Levin. No better ways to generate hard NP instances than picking uniformly at random. *Proc. 31st FOCS*, IEEE, pages 812–821, 1990.
- [Joh84] D. Johnson. The NP-completeness column: an ongoing guide. *J. of Algorithms*, 5:284–299, 1984.
- [Jon93] N. Jones. Constant time factors *do* matter. *Proc. 25th STOC*, ACM, pages 602–611, 1993.

- [KS95] C. Karg and R. Schuler. Structure in average case complexity. *Proc. 6th ISSAC*, 1995, to appear.
- [Ko83] K. Ko. On the definition of some complexity classes of real numbers. *Math. Systems Theory*, 16:95–109, 1983.
- [Lev86] L. Levin. Average case complete problems. *SIAM J. Comput.*, 15:285–286, 1986. (First appeared in *Proc. 16th STOC*, ACM, page 465, 1984.)
- [LP81] H. Lewis and C. Papadimitriou. *Elements of the Theory of Computation*. Prentice-Hall, 1981.
- [LV92] M. Li and P. Vitányi. Average case complexity under the universal distribution equals worst-case complexity. *Inf. Proc. Lett.*, 42:145–149, 1992.
- [Mil91] P. Miltersen. The complexity of malign ensembles. *Proc. 6th Structures*, IEEE, pages 164–171, 1991.
- [RS93] R. Reischuk and C. Schindelhauer. Precise average case complexity. *Proc. 10th STACS*, vol 665 of *Lect. Notes in Comp. Sci.*, pages 650–661, 1993.
- [Sch95a] R. Schuler. Some properties of sets tractable under every polynomial-time computable distribution. *Inf. Proc. Lett.*, 55:179–184, 1995.
- [Sch95b] R. Schuler. Average polynomial time is hard for exponential time under sn-reductions. *Proc. 15th FST&TCS*, 1995, to appear.
- [Sch96] R. Schuler. Truth-table closure and Turing closure of average polynomial time have different measures in EXP. *Proc. 11th Complexity*, IEEE, 1996, to appear.
- [SW95] R. Schuler and O. Watanabe. Towards average-case complexity analysis of NP optimization problems. *Proc. 10th Structures*, IEEE, pages 148–159, 1995.
- [SY92] R. Schuler and T. Yamakami. Structural average case complexity. *Proc. 12th FST&TCS*, vol 652 of *Lect. Notes in Comp. Sci.*, pages 128–139, 1992.
- [SY95] R. Schuler and T. Yamakami. Sets computable in polynomial time on average. *Proc. 1st COCOON*, vol 959 of *Lect. Notes in Comp. Sci.*, pages 400–409, 1995.
- [Ven91] R. Venkatesan. *Average-Case Intractability*. Ph.D. Thesis (Advisor: L. Levin), Boston University, 1991.
- [VL88] R. Venkatesan and L. Levin. Random instances of a graph coloring problem are hard. *Proc. 20th STOC*, ACM, pages 217–222, 1988.

- [VR92] R. Venkatesan and S. Rajagopalan. Average case intractability of Diophantine and matrix problems. *Proc. 24th STOC*, ACM, pages 632–642, 1992.
- [Wan] J. Wang. Average-case intractable NP problems. In preparation.
- [Wan95a] J. Wang. Average-case completeness of a word problem for groups. *Proc. 27th STOC*, ACM, pages 325–334, 1995.
- [Wan95b] J. Wang. Random instances of bounded string rewriting are hard. *J. of Computing and Information, Vol. 1, No. 1, Special Issue: Proc. 7th ICCI*, pages 11–23, 1995.
- [WB93a] J. Wang and J. Belanger. On average-P vs. average-NP. In *Complexity Theory—Current Research* (K. Ambos-Spies, S. Homer, and U. Schöninghs eds.), pages 47–67. Cambridge University Press, 1993. (First appeared in *Proc. 7th Structures*, IEEE, pages 318–326, 1992.)
- [WB93b] J. Wang and J. Belanger. Honest iteration schemes of randomizing algorithms. *Inf. Proc. Lett.*, 45:275–278, 1993.
- [WB95] J. Wang and J. Belanger. On the NP-isomorphism problem with respect to random instances. *J. Comp. Sys. Sci.*, 50:151–164, 1995.
- [Wat94] O. Watanabe. Test instance generation for promised NP search problems. *Proc. 9th Structures*, IEEE, pages 205–216, 1994.
- [Wil84] H. Wilf. An $O(1)$ expected time algorithm for the graph coloring problem. *Inf. Proc. Lett.*, 18:119–122, 1984.
- [Yam] T. Yamakami. Polynomial-time samplable distributions. Manuscript.