Towards Average-Case Complexity Analysis of NP Optimization Problems*

Rainer SCHULER

Abteilung Theoretische Informatik
Universität Ulm
Oberer Eselsberg
D 89069 Ulm, Germany
schuler@informatik.uni-ulm.de

Osamu WATANABE

Department of Computer Science
Tokyo Institute of Technology
Meguro-ku Ookayama 1-12-1
Tokyo 152, JAPAN
watanabe@cs.titech.ac.jp

ABSTRACT

For the worst-case complexity measure, if P = NP, then P = OptP, i.e., all NP optimization problems are polynomial-time solvable. On the other hand, it is not clear whether a similar relation holds when considering average-case complexity. We investigate the relationship between the complexity of NP decision problems and that of NP optimization problems under polynomial-time computable distributions, and study what makes them (seemingly) different. It is shown that the difference between P_{tt}^{NP}-samplable and P^{NP}-samplable distributions is crucial.

1. Introduction

Recently, "average-case complexity" has received considerable attention by researchers in several fields of computer science. Even a problem is not (or may not be) solvable efficiently in the worst-case, it may be solvable efficiently on average. Indeed, several results have been obtained that show even simple algorithms work well on average (see, e.g., [Joh84]). On the other hand, most of those results are about concrete problems, and not so much has been done for more general study of average-case complexity, though there are many interesting open questions in this area. In this paper, we consider one of such open questions, and improve our knowledge towards this question.

We consider the following question: Suppose every NP problem is polynomial time

*The part of this work has been done while the second author was visiting Universität Ulm and supported in part by the guest scientific program of Universität Ulm. The second author is supported in part by Grant in Aid for Scientific Research of the Ministry of Education, Science and Culture of Japan under Grant-in-Aid for Research (C) 06680308 (1994).

solvable on average. Does this mean that every NP optimization problem is also polynomial time solvable on average? Here "NP problem" is a decision problem for an NP set. On the other hand, "NP optimization problem" is a problem of finding optimal solutions for a problem with a polynomial-time computable cost function. Krentel [Kre88] defined the class OptP for the class of NP optimization problems. Thus, the question is whether P = NP on average implies P = OptP on average. (Since OptP is the class of functions, "P = OptP" should be written as "PF = OptP". In this paper, however, we will use P to denote both language and function classes.)

For discussing average-case complexity, one should be careful about input distributions and distribution classes. It may not be so realistic to discuss polynomial-time computability considering any input distribution. Levin [Lev86], who established a framework for average-case complexity theory, proposed to consider only "polynomial-time computable distribution (in short, P-computable distribution)" as input distributions. Later more generalized notion, i.e., "polynomial-time samplable distribution (in short, P-samplable distribution)", has been proposed [BCGL92]. We essentially follow Levin's framework, and regard P-computable distributions (or P-samplable distributions) as realistic input distributions. Thus, by "P = NP on average" we mean that for every NP problem and every P-computable distribution, the problem is solvable in polynomial-time on average when an input instance is given under the distribution. (In this introduction, we will use, e.g., "P $=_{\rm ave}$ NP (under P-comp. dist.)" to mean "P = NP on average for any P-computable distribution.")

For the worst-case complexity measure, we have P = NP $\Longrightarrow$ P = OptP. This is from the following reason: Every NP optimization problem A is polynomial-time solvable by some algorithm Q by using some NP set X as an oracle. But since P = NP, we can replace oracle X with some polynomial-time machine M for X. Thus, Q^M solves X in polynomial-time. This simple argument does not work, however, in the average-case complexity. Even if X is solvable by M in polynomial-time on average under any P-computable distribution, this does not mean that Q^M runs in polynomial-time on average under every P-computable distribution. This is because queries to X may occur under a very strange distribution, for which no algorithm solves X in polynomial-time on average. Thus, it is not clear that the relation P $=_{\rm ave}$ NP (under P-comp. dist.) $\Longrightarrow$ P $=_{\rm ave}$ OptP (under P-comp. dist.) holds; or, it may not hold at all. In this paper, we study what makes this relation difficult.

We consider two approaches. First, we investigate how much we need to enlarge a distribution class $\mathcal{D}_1$ so that the following implication holds: P $=_{\rm ave}$ NP (under $\mathcal{D}_1$ dist.) $\Longrightarrow$ P $=_{\rm ave}$ OptP (under P-comp. dist.). Secondly, we consider for which class $\mathcal{D}_2$

can we prove the following implication: $\mathrm{P} =_{\mathrm{ave}} \mathrm{NP}$ (under P-comp. dist.) $\Longrightarrow \mathrm{P} =_{\mathrm{ave}}$ NP (under $\mathcal{D}_2$ dist.). Obviously, if $\mathcal{D}_1 \subseteq \mathcal{D}_2$, then we have an affirmative answer to our question. While we have been unable to achieve this, we can prove the following results.

(1) If $\mathrm{P} =_{\mathrm{ave}} \mathrm{NP}$ under every P^{NP}-samplable distribution, then $\mathrm{P} =_{\mathrm{ave}} \mathrm{OptP}$ under every P-computable distribution. Furthermore, the converse relation holds. That is, the assumption is indeed necessary for showing $\mathrm{P} =_{\mathrm{ave}} \mathrm{OptP}$ (under P-comp. dist.).

(2) If $\mathrm{P} =_{\mathrm{ave}} \mathrm{NP}$ under every P-computable distribution, then $\mathrm{P} =_{\mathrm{ave}} \mathrm{NP}$ under every $\mathrm{P}_{\mathrm{tt}}^{\mathrm{NP}}$-samplable distribution.

Thus, we now know that the difference between $\mathrm{P}_{\mathrm{tt}}^{\mathrm{NP}}$-samplable and P^{NP}-samplable distributions is crucial for our question. Motivated by this, we also study how strong $\mathrm{P}_{\mathrm{tt}}^{\mathrm{NP}}$-samplable distributions are, and obtain the following result.

(3) Every #P-computable distribution can be approximated within constant factor by some $\mathrm{P}_{\mathrm{tt}}^{\mathrm{NP}}$-samplable distribution.

Thus, from this and the above result (2), we can show that #P-computable distributions are not stronger than P-computable distributions for discussing the average-case polynomial-time computability of NP.

Impagliazzo and Levin [IL90] made an important observation on different classes of distributions. They showed that if $\mathrm{P} =_{\mathrm{ave}} \mathrm{NP}$ (under P-comp. dist.), then it indeed holds that $\mathrm{P} =_{\mathrm{ave}} \mathrm{NP}$ (under P-samplable dist.). For obtaining the above results (1) and (2), we extend their technique and prove that if $\mathrm{P} =_{\mathrm{ave}} \mathrm{NP}$ (under P-comp. dist.), then $\mathrm{P} =_{\mathrm{ave}} \mathrm{NP}$ even for any "average" P-samplable distribution. For showing (3), we use another property of hash functions.

2. Preliminaries

In this paper, we follow the standard definitions and notations in computational complexity theory (see, e.g., [BDG88, BDG91]).

Throughout this paper, we fix our alphabet to $\Sigma = \{0, 1\}$, and by a *string* we mean an element of Σ^*. For any string x, let $|x|$ denote the length of x. For any $n \geq 0$ and any set L of strings, let $L^{\leq n}$ and $L^{=n}$ be the set of strings in L of length $\leq n$ and of length n respectively. We use $\| L \|$ to denote the cardinality of L. Let $\mathbf{N}$ denote the set of nonnegative integers. Usually, we assume the binary encoding of $\mathbf{N}$ on Σ^*, but sometimes numbers are encoded in a tally form, i.e., as a string in 0^*. For any $n \in \mathbf{N}$, let $\overline{n}$ denote 0^n.

We use a standard one-to-one pairing function from $\Sigma^* \times \Sigma^*$ to Σ^* that is computable and invertible in polynomial-time. For inputs x and y, we denote the output of the pairing function by $\langle x, y \rangle$; this notation is extended to denote any n tuple. We also use a polynomial-time computable pairing function, say, $\langle n, x, y \rangle_n$ such that for every $n \geq 0$ and for all (x, y) in some finite set D_n, $\langle n, x, y \rangle_n$ is of the same length, which is uniquely determined by n. (We assume that $\langle n, x, y \rangle_n$ is undefined for $(x, y) \notin D_n$.) It is not so hard to define such pairing functions by using standard padding technique. We often omit specifying D_n when it is clear from the context.

For any random event α, let $\Pr_\alpha\{\, \Phi(\alpha) \,\}$ be the probability $\Phi(\alpha)$ holds. For example, for any randomized machine M, $\Pr_M\{\, \Phi \,\}$ is the probability that Φ holds when M executes following its internal coin tosses; $\Pr_{x \in U}\{\, \Phi(x) \,\}$ is the probability that Φ holds when x is chosen from U randomly. The latter one is also written as $\Pr\{\, \Phi(x) \,|\, x \in U \,\}$.

For our computation model, we use *randomized* (oracle) Turing machines. We say that a machine M *accepts* a language L if for all $x \in \Sigma^*$,

$$\begin{aligned} x \in L &\implies \Pr_M\{\, M \text{ accepts } x \,\} \geq 2/3, \text{ and} \\ x \notin L &\implies \Pr_M\{\, M \text{ accepts } x \,\} \leq 1/3. \end{aligned}$$

Similarly, for any (multi-valued) function f, we say that f is *computed* by M if for every x, the probability that $M(x)$ computes $f(x)$ is greater than $2/3$. Note that for any decision problem and any single-valued function evaluation problem, we can easily reduce the error probability by executing a machine several times and then taking their majority. Also even for evaluating a multi-valued function, if it is easy to verify the correctness of a given answer, then we can easily reduce the error probability.

Throughout this paper, we use this randomized polynomial-time computability instead of the deterministic one. Thus, by "f is polynomial-time computable", we precisely mean that f is polynomial-time computable by some randomized machine. Note that for a randomized machine M and any input x, the running time of M on x may differ depending on the random sequence that M uses. The running time of M on x (written as $\mathrm{time}_M(x)$) is formally defined as an expected running time of M on x over all random sequences of M.

Optimization problems we consider are specified by polynomial-time computable functions. For a polynomial-time computable function *cost*: $\Sigma^* \times \Sigma^* \to \mathbf{N}$, and a polynomial p, the *NP optimization problem* specified by *cost* and p is to compute the following function *opt-val*:

$$\textit{opt-val}(x) \;=\; y \in \Sigma^{\leq p(|x|)} \text{ such that } \textit{cost}(x, y) = \textit{opt}(x),$$

where $opt(x) = \max\{cost(x, y') \mid y' \in \Sigma^{\leq p(|x|)}\}$. Notice that *opt-val* is multi-valued in general. OptP is the class of functions like *opt-val*. (In order to keep similarity with "NP search problem", we modified the original definition of OptP [Kre88], where OptP is defined as the class of functions like *opt*. It is, however, easy to show that the above class OptP is a generalization of the original class, and they are essentially the same for discussing polynomial-time computability.)

For a given NP set L, an NP search problem for L is to search, for a given instance x in L, *some* witness for $x \in L$. More formally, for a polynomial-time computable predicate R on $\Sigma^* \times \Sigma^*$, and a polynomial p, the *NP search problem* specified by R and p is to compute a value of the following function *search*:

$$search(x) \;=\; y \in \Sigma^{\leq p(|x|)} \text{ such that } R(x, y) \text{ holds.}$$

Notice again that *search* may be multi-valued in general. Let SearchP denote the class of functions like *search*.

Preliminaries for Average-Case Complexity Theory

A *probability function* μ on U is a total function from U to $[0, 1]$ such that $\sum_{x \in U} \mu(x) = 1$. We use λ to denote the uniform probability function on Σ^r, where r will be specified in each context.

Throughout this paper, only length-wise input distributions are considered. That is, for each $n \geq 0$, we consider probability function μ_n on Σ^n (hence, $\sum_{x \in \Sigma^n} \mu_n(x) = 1$), and discuss average-case complexity assuming that each instance $x \in \Sigma^n$ appears with probability $\mu_n(x)$. Thus, formally speaking, an *input distribution* (or, *distribution* in short) is specified as a family $\{\mu_n\}_{n \geq 0}$ of such length-wise probability functions μ_n. In this paper, however, we denote an input distribution by a single function such as μ, and for each $x \in \Sigma^n$, we use $\mu(x)$ to denote $\mu_n(x)$.

Definition 2.1. ($\mathrm{P}^{\mathcal{C}}$-computable distribution, $\mathrm{P}^{\mathcal{C}}_{\mathrm{tt}}$-computable distribution)
For any complexity class $\mathcal{C}$ and any input distribution μ, μ is a *$P^{\mathcal{C}}$-computable distribution* if its cumulative distribution μ^* is computed by some polynomial-time bounded randomized oracle Turing machine M relative to some oracle set $X \in \mathcal{C}$. The notion of "$\mathrm{P}^{\mathcal{C}}_{\mathrm{tt}}$-computable distribution" is defined similarly by considering oracle Turing machines that ask queries only nonadaptively.

Remark. We are using the randomized polynomial-time computability for the "polynomial-time computability" notion. Furthermore, we are using length-wise probability, and the *cumulative distribution* μ^* of μ is defined by $\mu^*(x) = \sum_{x' : x' \in \Sigma^n \wedge x' \leq x} \mu(x')$, where $\leq$ is the standard lexicographic order on Σ^*. Thus, the above definition is not

equivalent to the original one in [Lev86]. Nevertheless, the following argument does not change even if the above notion is defined by using the *deterministic* polynomial-time computability. In this case, it is easy to show that our definition is equivalent to the original one as long as the probability for each length is polynomial-time computable.

Definition 2.2. ($\mathrm{P}^{\mathcal{C}}$-samplable distribution, $\mathrm{P}^{\mathcal{C}}_{\mathrm{tt}}$-samplable distribution)
For any complexity class $\mathcal{C}$ and any input distribution μ, μ is a *$P^{\mathcal{C}}$-samplable distribution* if there exist a polynomial-time bounded randomized oracle Turing machine G, which is called a *generator*, and some set $X \in \mathcal{C}$ such that for each x (let $n = |x|$),

$$\mu(x) \;=\; \mathrm{Pr}_G\{\, G^X(\overline{n}) \text{ yields } x \,\}.$$

The notion of "$\mathrm{P}^{\mathcal{C}}_{\mathrm{tt}}$-samplable distribution" is defined similarly by considering oracle Turing machines that ask queries only nonadaptively.

We define distribution classes. For any complexity class $\mathcal{C}$, let $\mathrm{P}^{\mathcal{C}}$-comp denote the classes of distributions that are $\mathrm{P}^{\mathcal{C}}$-computable. Let $\mathrm{P}^{\mathcal{C}}$-samp (resp., $\mathrm{P}^{\mathcal{C}}_{\mathrm{tt}}$-samp) denote the class of $\mathrm{P}^{\mathcal{C}}$-samplable (resp., $\mathrm{P}^{\mathcal{C}}_{\mathrm{tt}}$-samplable) distributions. Though we defined notions in a general way, we will mainly consider distribution classes P-comp, P-samp, $\mathrm{P}^{\mathrm{NP}}_{\mathrm{tt}}$-samp, and P^{NP}-samp.

Note that by these definitions, the values of probability functions are always (binary) rational numbers. Thus, these definitions are weaker than the original ones [Lev86, BCGL92] that allow real numbers for probability. Nevertheless, it is shown [Gur91, Lemma 1.6] that we lose no generality by this restriction for discussing polynomial-time computability.

Levin [Lev86] gave a general and robust definition to the notion of "polynomial-time solvable on average". Levin's definition uses distributions on Σ^*; on the other hand, we are using length-wise input distributions in order to make our discussion more intuitive. Thus, we modify Levin's definition to a length-wise version, which is more intuitive but less robust. The following definition for "polynomial on μ-average" was suggested by Gurevich [Gur91].

Definition 2.3. (Polynomial on μ-average)
A function f is *polynomial on μ-average* if there exist constants $c, d > 0$ such that for all $n \geq 0$,

$$\sum_{x \in \Sigma^n} \frac{f(x)^{1/d}}{n} \mu(x) \;\leq\; c.$$

For showing "polynomial on μ-average", the following simple characterizations are useful.

Proposition 2.4. Let f be any function from Σ^* to $\mathbf{N}$, and μ be any input distribution. Then f is polynomial on μ-average if there exist polynomials p and q and a constant $d > 0$ that satisfy the following for all $n \geq 0$:

$$\sum_{x \in \Sigma^n} \frac{f(x)^{1/d}}{q(n)} \mu(x) \ \leq \ p(n).$$

Remark. The proposition is provable by using an argument similar to the proof of [Gur91, Lemma 1.5].

Proposition 2.5. Let f and t be any functions from Σ^* to $\mathbf{N}$ and from $\mathbf{N}$ to $\mathbf{N}$ respectively, and let μ be any input distribution. For any $k \geq 1$, assume that f is polynomial in $t(n)^k$ on μ-average; that is, for some constants $c, d > 0$ and for all $n \geq 0$, we have

$$\sum_{x \in \Sigma^n} \frac{f(x)^{1/d}}{t(n)^k} \mu(x) \ \leq \ c.$$

Then for any set $X \subseteq \Sigma^*$ such that $\mu(X \cap \Sigma^n) \leq 1/t(n)$ for all $n \geq 0$, f is polynomial on μ-average on X; or more specifically, for all $n \geq 0$ and $X' \subseteq \Sigma^n$ such that $\mu(X') \leq 1/t(n)$, we have

$$\sum_{x \in X'} f(x)^{1/(2dk)} \cdot \mu(x) \ \leq \ c + 1.$$

Proof. We split the sum depending on the value of $f(x)^{1/(2dk)}$.
(1) Let S be the set of all $x \in X'$ with $f(x)^{1/(2dk)} \leq t(n)$. Then

$$\sum_{x \in S} f(x)^{1/(2dk)} \cdot \mu(x) \ \leq \ \sum_{x \in S} t(n) \cdot \mu(x) \ = \ t(n) \cdot \sum_{x \in S} \mu(x) \ \leq \ t(n) \cdot \frac{1}{t(n)} \ = \ 1.$$

(2) Let T be the set of all $x \in X'$ with $f(x)^{1/(2dk)} > t(n)$, i.e $f(x)^{1/(2d)} > t(n)^k$. Then

$$\begin{aligned} \sum_{x \in T} f(x)^{1/(2dk)} \cdot \mu(x) \ &= \ \sum_{x \in T} \frac{t(n)^k \cdot f(x)^{1/(2dk)}}{t(n)^k} \mu(x) \\ &\leq \ \sum_{x \in T} \frac{f(x)^{1/(2d)} \cdot f(x)^{1/(2dk)}}{t(n)^k} \mu(x) \ \leq \ \sum_{x \in T} \frac{f(x)^{1/d}}{t(n)^k} \mu(x) \ \leq \ c. \end{aligned}$$

□

We say that a machine M runs in *polynomial-time on μ-average* if its (expected) running time time_M is polynomial on μ-average. It is shown [Gur91, Proposition 1.1] that the above definition is equivalent to Levin's original one for distributions satisfying a

certain natural condition. Furthermore, all the arguments in this paper can be modified for Levin's definition. Thus, we will lose no generality by using this definition.

For our notion of "average polynomial-time", we will use the above definition in this paper. Nevertheless, we should also note that there are weaker (but still natural and robust) ways to define this notion. The following is one such example, which has been used in the study of cryptographic one-way functions.

Definition 2.6. (Almost polynomial under μ)
A function f is *almost polynomial under* μ if there exist integer k such that for all $c \geq 0$ and for almost all $n \geq 0$,

$$\mu\{\, x \in \Sigma^n \mid f(x) \leq n^k \,\} \;\geq\; 1 - \frac{1}{n^c}.$$

The reducibility notion often helps us to discuss the implication of complexity assumptions such as P = NP. Here we use the following reducibility from [BCGL92].

Definition 2.7. (Random reduction)
Let (L_1, μ_1) and (L_2, μ_2) be respectively a pair of a language and a probability function. A *random reduction* (or, more specifically, $\propto_{\mathrm{T}}^{\mathrm{P}}$*-reduction*) from (L_1, μ_1) to (L_2, μ_2) is a randomized oracle Turing machine Q with the following properties:
(a) Q is polynomial-time bounded.
(b) For every $x \in \Sigma^*$,

$$\begin{aligned} x \in L_1 &\implies \Pr_Q\{\, Q^{L_2} \text{ accepts } x \,\} \;\geq\; 2/3, \text{ and} \\ x \notin L_1 &\implies \Pr_Q\{\, Q^{L_2} \text{ rejects } x \,\} \;\geq\; 2/3. \end{aligned}$$

(c) There exists a polynomial p such that for any $n \geq 0$ and for every $y \in \Sigma^*$ that is queried by Q^{L_2} on some $x \in \Sigma^n$, we have

$$\mu_2(y) \;\geq\; \frac{1}{p(n)} \sum_{x \in \Sigma^n} \mathrm{Ask}_Q(x, y; L_2) \cdot \mu_1(x),$$

where $\mathrm{Ask}_Q(x, y; L_2)$ is the probability $\Pr_M\{\; Q^{L_2}(x) \text{ queries } y \;\}$.
Remark. Here we modified the original definition for our length-wise probability functions. Some of the above conditions are slightly more restrictive than the original ones.

A $\propto_{\mathrm{T}}^{\mathrm{P}}$-reduction Q that asks queries only nonadaptively is called a $\propto_{\mathrm{tt}}^{\mathrm{P}}$*-reduction*. A $\propto_{\mathrm{T}}^{\mathrm{P}}$-reduction is extensively used from a (multi-valued) function to a set. It is easy to show that the following relation holds [BCGL92].

Proposition 2.8. Let (L_1, μ_1) and (L_2, μ_2) be respectively a pair of a language and a probability function. If (L_1, μ_1) is $\propto^{\mathrm{P}}_{\mathrm{T}}$-reducible to (L_2, μ_2), and L_2 is polynomial-time solvable on μ_2-average, then L_1 is polynomial-time solvable on μ_1-average.

Remark. This can be extended for a reduction from (f_1, μ_1) to (L_2, μ_2), for a (possibly multi-valued) function f_1.

3. Known Results and Some Simple Observations

For simplifying our statements, we will use the following complexity class introduced in [SY92].

Definition 3.1. (Class $\mathrm{P}_{\mathcal{D}}$)
For any distribution class $\mathcal{D}$, $\mathrm{P}_{\mathcal{D}}$ is the class of languages L such that for any distribution $\mu \in \mathcal{D}$, L is accepted by some randomized Turing machine whose running time is polynomial on μ-average.

Remark. We will also use this notation for discussing the complexity of computing functions. That is, a function f is in $\mathrm{P}_{\mathcal{D}}$ if for any distribution $\mu \in \mathcal{D}$, f is computable by some randomized Turing transducer whose running time is polynomial on μ-average.

For example, $\mathrm{P}_{\text{P-comp}}$ is the class of languages that are polynomial-time solvable on average under any polynomial-time computable distributions. Hence, the relation NP $\subseteq \mathrm{P}_{\text{P-comp}}$ is equivalent to the following statement: for all P-computable distribution μ and all L in NP, L is polynomial-time decidable on μ-average. Thus, by this notation, we can state our principal question as follows: Is it true that NP $\subseteq \mathrm{P}_{\text{P-comp}} \Longrightarrow$ OptP $\subseteq \mathrm{P}_{\text{P-comp}}$? Also two questions we asked in the introduction are stated as follows.

Q1. For which distribution class $\mathcal{D}$, do we have the following implication?:

$$\mathrm{NP} \subseteq \mathrm{P}_{\mathcal{D}} \implies \mathrm{OptP} \subseteq \mathrm{P}_{\text{P-comp}}.$$

Q2. For which distribution class $\mathcal{D}$, do we have the following implication?:

$$\mathrm{NP} \subseteq \mathrm{P}_{\text{P-comp}} \implies \mathrm{NP} \subseteq \mathrm{P}_{\mathcal{D}}.$$

Here let us review previous results.

Proposition 3.2. [BCGL92]
(1) NP $\subseteq \mathrm{P}_{\text{P-samp}} \implies \Theta_2^{\mathrm{P}} \subseteq \mathrm{P}_{\text{P-samp}}$ (where $\Theta_2^{\mathrm{P}} = \mathrm{P}_{\mathrm{tt}}^{\mathrm{NP}}$).
(2) NP $\subseteq \mathrm{P}_{\text{P-samp}} \implies$ SearchP $\subseteq \mathrm{P}_{\text{P-samp}}$.

Remark. The first fact, which is almost immediate from Proposition 2.8, is not explicitly stated in [BCGL92], but the idea has been used to show the second fact.

Proposition 3.3. [IL90] $\mathrm{NP} \subseteq \mathrm{P}_{\text{P-comp}} \implies \mathrm{NP} \subseteq \mathrm{P}_{\text{P-samp}}$.

By using these propositions, it is not so hard to show that if $\mathrm{NP} \subseteq \mathrm{P}_{\text{P-comp}}$, then every NP optimization problem has an average-polynomial-time approximation scheme.

Consider any NP optimization problem Π, and let *cost* and p be a cost function and a polynomial specifying Π. For any probability function μ, we say that Π has a *μ-average-polynomial-time approximation scheme* if for each ε, $0 < \varepsilon < 1$, there exists a randomized Turing machine M with the following properties:

(a) The running time of M is polynomial on μ-average.

(b) For any $x \in \Sigma^*$, with probability greater than $2/3$, $M(x)$ yields an *ε-approximation* y; that is, y satisfies

$$\frac{opt(x) - cost(x,y)}{opt(x)} \ \leq \ \varepsilon.$$

(Recall that we assumed that $cost(x,y) \geq 0$ and that we are considering maximization problems.)

Theorem 3.4. If $\mathrm{NP} \subseteq \mathrm{P}_{\text{P-comp}}$, then every NP optimization problem has a μ-average-polynomial-time approximation scheme for every P-computable distribution μ.

Proof. (The same idea has been used for proving [CG93, Lemma 4].)
Let Π be any NP optimization problem that is specified by *cost* and p. Also let μ be any P-computable distribution.

Consider any constant ε, $0 < \varepsilon < 1$, and let it be fixed. We show some machine exists that satisfies the above conditions (a) and (b) for ε. Formally, the problem is to compute a (multi-valued) function *ap-val* whose value on x takes every ε-approximation of x. We reduce this approximation problem to some NP search problem. Let $opt(x) = \max\{cost(x,y') \mid y' \in \Sigma^{\leq p(|x|)}\}$. We may assume that for every $x \in \Sigma^*$, $0 \leq opt(x) \leq 2^{q(|x|)}$ for some polynomial q. Also let r be a polynomial such that $\lfloor (1-\varepsilon)^{r(n)} 2^{q(n)} \rfloor = 0$ for any n. Now we consider the problem of computing the following function:

For each $\langle n, x, k \rangle_n$, where $x \in \Sigma^n$ and $1 \leq k \leq r(n)$,
$search(\langle n, x, k \rangle_n) \ = \ y$ such that $\lfloor (1-\varepsilon)^k 2^{q(n)} \rfloor \leq cost(x,y) \leq \lceil (1-\varepsilon)^{k-1} 2^{q(n)} \rceil$.

Clearly, if there is no solution for $\langle n, x, k-1 \rangle_n$, then any solution for $\langle n, x, k \rangle_n$ is an ε-approximation. Thus, the problem of computing *ap-val*(x) is solvable by computing *search* for all $\langle n, x, k \rangle_n$, $1 \leq k \leq r(n)$. That is, *ap-val* is polynomial-time reducible to *search*. Now it suffices to show some μ' in P-comp such that $(ap\text{-}val, \mu)$ is $\propto^{\mathrm{P}}_{\mathrm{T}}$-reducible (in fact, $\propto^{\mathrm{P}}_{\mathrm{tt}}$-reducible) to $(search, \mu')$. This can be done by simply defining $\mu'(\langle n, x, k \rangle_n) = \mu(x)/r(|x|)$. □

4. The First Question

Here we discuss our first question. Namely, we would like to obtain some sufficient condition (in terms of the relation NP $\subseteq \mathrm{P}_{\mathcal{D}}$) for all NP optimization problems to be polynomial-time solvable on average under any P-computable distribution. For example, it has been shown [SY92] that $\mathrm{P}_{\text{E-comp}}$ is exactly the same as P, where E-comp is the class of exponential-time computable distributions. Hence, NP $\subseteq \mathrm{P}_{\text{E-comp}}$ implies that P = NP even in the worst-case; thus clearly, all NP optimization problems are polynomial-time solvable (even in the worst-case). Here we will show that a much weaker condition NP $\subseteq \mathrm{P}_{\mathrm{P}^{\mathrm{NP}}\text{-samp}}$ is sufficient, and furthermore it is indeed necessary.

Before proving this result, let us first show that we lose no generality if we discuss our problem by using Δ_2^{P} class.

Theorem 4.1. For any distribution class $\mathcal{D}$ defined in Section 2, we have $\Delta_2^{\mathrm{P}} \subseteq \mathrm{P}_{\mathcal{D}}$ $\Longleftrightarrow$ OptP $\subseteq \mathrm{P}_{\mathcal{D}}$.

Proof. ($\Longleftarrow$) Let L be any set in Δ_2^{P}. By using the proof technique for [Kre88, Theorem 3.1], we can define some function *opt-val* in OptP such that $x \in L$ if and only if the last bit of *opt-val*(x) is 1. Thus, for any distribution $\mu \in \mathcal{D}$, if *opt-val* is polynomial-time computable on μ-average, so is L.
($\Longrightarrow$) Let *opt-val* be any function in OptP. We assume that for some polynomial r, $|$*opt-val*$(x)| \leq r(|x|)$ for all $x \in \Sigma^*$. It is easy to show that the following set L is in Δ_2^{P}:

$$L = \{ \langle n, x, i, d \rangle_n : x \in \Sigma^n,\ 1 \leq i \leq r(n),\ d \in \{0,1\},\ \text{and the } i\text{th bit of the lexicographically smallest } \textit{opt-val}(x) \text{ is } d \}.$$

Clearly, for any $x \in \Sigma^*$, some value of *opt-val*(x) (in fact, the lexicographically smallest value of *opt-val*(x)) is polynomial-time computable by asking at most $q(|x|)$ queries to L. Then it is easy to show that (*opt-val*, μ) is $\propto_{\mathrm{T}}^{\mathrm{P}}$-reducible (in fact, $\propto_{\mathrm{tt}}^{\mathrm{P}}$-reducible) to (L, μ') for some $\mu' \in \mathrm{P}_{\mathcal{D}}$. Then the proof follows from Proposition 2.8. □

Now we are ready to prove our first main result. That is, NP $\subseteq \mathrm{P}_{\mathrm{P}^{\mathrm{NP}}\text{-samp}}$ is sufficient and indeed necessary for showing $\Delta_2^{\mathrm{P}} \subseteq \mathrm{P}_{\text{P-comp}}$ (i.e., OptP $\subseteq \mathrm{P}_{\text{P-comp}}$). First we show the sufficiency. In fact, we can prove that NP $\subseteq \mathrm{P}_{\mathrm{P}^{\mathrm{NP}}\text{-samp}}$ is sufficient for $\Delta_2^{\mathrm{P}} \subseteq \mathrm{P}_{\mathrm{P}^{\mathrm{NP}}\text{-samp}}$.

Theorem 4.2. If NP $\subseteq \mathrm{P}_{\mathrm{P}^{\mathrm{NP}}\text{-samp}}$, then $\Delta_2^{\mathrm{P}} \subseteq \mathrm{P}_{\mathrm{P}^{\mathrm{NP}}\text{-samp}}$.

Proof. Let L_1 be any set in Δ_2^{P} and μ_1 be any P^{NP}-samplable distribution. Then we have a polynomial-time bounded deterministic Turing machine Q and an NP set L_2 such that Q^{L_2} accepts L_1. Also there exist a polynomial-time bounded randomized Turing

machine G and an NP set X witnessing μ_1 is P^{NP}-samplable. We may assume, for some polynomial q_1 and all $x \in \Sigma^n$, that $Q^{L_2}(x)$ asks exactly $q_1(n)$ distinct queries, and that it always asks queries of the form $\langle n, x, y \rangle_n$ for some y.

Now for any $\langle n, x, y \rangle_n$ that is queried by $Q^{L_2}(x)$, define $\mu_2(\langle n, x, y \rangle_n) = \mu_1(x)/q_1(n)$, and $\mu_2(z) = 0$ for any other z. Then μ_2 is a well-defined probability function. Also it is clear that μ_2 satisfies the condition (c) of Definition 2.7 for L_1, L_2, Q, and μ_1 by taking $p(n) = 1$ and $q(n) = q_1(n)$. Hence, (L_1, μ_1) is $\propto^{\mathrm{P}}_{\mathrm{T}}$-reducible to (L_2, μ_2).

On the other hand, for a given $n \geq 0$, one can generate each $\langle n, x, y \rangle_n$ with probability $\mu_2(\langle n, x, y \rangle_n)$ in the following way: First, simulate $G^X(\overline{n})$ to generate x. Then simulate $Q^{L_2}(x)$ to generate all queries. Finally, output one of the generated queries at random. Thus, μ_2 is $\mathrm{P}^{X \oplus L_2}$-samplable. Therefore, by assumption, L_2 is polynomial-time solvable on μ_2-average, which proves that L_1 is polynomial-time solvable on μ_1-average because $(L_1, \mu_1) \propto^{\mathrm{P}}_{\mathrm{T}} (L_2, \mu_2)$. □

Next we show that $\mathrm{NP} \subseteq \mathrm{P}_{\mathrm{P}^{\mathrm{NP}}\text{-samp}}$ is indeed necessary for showing $\Delta^{\mathrm{P}}_2 \subseteq \mathrm{P}_{\text{P-comp}}$. That is, the following relation.

Theorem 4.3. If $\Delta^{\mathrm{P}}_2 \subseteq \mathrm{P}_{\text{P-comp}}$, then $\mathrm{NP} \subseteq \mathrm{P}_{\mathrm{P}^{\mathrm{NP}}\text{-samp}}$.

Suppose $\Delta^{\mathrm{P}}_2 \subseteq \mathrm{P}_{\text{P-comp}}$, and let us first discuss very intuitively why this seems to imply $\mathrm{NP} \subseteq \mathrm{P}_{\mathrm{P}^{\mathrm{NP}}\text{-samp}}$. Consider any distribution μ_0 in P^{NP}-samp. By definition, there exist a randomized machine G_0 and a NP oracle set X_0 such that $G_0^{X_0}(\overline{n})$ produces $x \in \Sigma^n$ according to μ_0. Since $\Delta^{\mathrm{P}}_2 \subseteq \mathrm{P}_{\text{P-comp}}$, every P^{NP}-computation can be simulated by some randomized machine whose running time is polynomial on average; thus, the computation of $G_0^{X_0}$ can be simulated by some randomized machine G_0' whose running time is polynomial on average. That is, the distribution μ_0 itself is "on average" P-samplable, or, one can define a P-samplable distribution μ_0' that "approximates" μ_0. Since $\Delta^{\mathrm{P}}_2 \subseteq \mathrm{P}_{\text{P-comp}}$, clearly $\mathrm{NP} \subseteq \mathrm{P}_{\text{P-comp}}$. Thus, by Proposition 3.3, $\mathrm{NP} \subseteq \mathrm{P}_{\text{P-samp}}$; that is, NP sets are decidable in polynomial-time on μ-average for any P-samplable distribution μ. Hence, NP sets are decidable in polynomial-time on μ_0'-average. But this property seems to hold for μ_0 because μ_0' is a "good approximation" of μ_0.

This intuitive idea can be formalized to prove a similar relation for weaker "average polynomial-time" notions such as our almost polynomial-time criteria (Definition 2.6). On the other hand, for proving the above theorem, we need a more careful argument. We begin by defining the notion of "polynomial-time samplable on average".

For any probability function μ, we say that μ is *aveP-samplable distribution* if there exists a *deterministic* Turing machine G such that for each $n \geq 0$,

(1) for every $s, s' \in \Sigma^*$, if $G(\overline{n}, s) \neq \perp$, then $G(\overline{n}, ss') = \perp$.

(2) for every $x \in \Sigma^n$, $\mu(x) = \lambda\{\, s \in \Sigma^* \,:\, G(\overline{n}, s) \text{ yields } x \,\}$, and

(3) G runs polynomial-time on λ-average; that is, there exist constants $c, d > 0$ (independent from n) such that

$$\sum_{s \in \Sigma^* \wedge G(\overline{n}, s) \neq \perp} \frac{(\mathrm{time}_G(\overline{n}, s))^{1/d}}{n} \lambda(s) \;\leq\; c.$$

Here we use λ to denote the uniform distribution on Σ^*. That is, $G(\overline{n}, s)$ (with $G(\overline{n}, s) \neq \perp$) is regarded as the execution of *randomized* generator that halts after consuming all bits of s for its coin-tossing. In the following, we will refer G satisfying (1) and (2) as a *deterministic generator* for μ. Notice that the generator G in Definition 2.2 can be modified to this type of generator that halts *always* in polynomial-time. Thus, this notion is a natural generalization of P-samplable distribution. We use aveP-samp to denote the class of aveP-samplable distributions.

Now we prove the following theorem, which is a generalization of Proposition 3.3.

Theorem 4.4. $\mathrm{NP} \subseteq \mathrm{P}_{\text{P-comp}} \implies \mathrm{NP} \subseteq \mathrm{P}_{\text{aveP-samp}}$.

Proof. Consider any set L_0 in NP and any aveP-samplable distribution μ_0. We will construct some machine M_2 that recognizes L_0 in polynomial-time on μ_0-average.

Let G_0 be a deterministic generator for μ_0 that satisfies the condition (1) $\sim$ (3) above. We consider any $n \geq 0$; let us fix it for a while, and discuss only inputs in Σ^n. For simplifying notation, in the following, we write $G(\overline{n}, s)$ as $G(s)$.

Note that $L_0 \in \mathrm{NP}$ is solvable in exponential-time. Hence, for some polynomial e_0, we may consider only instances x with $\mu(x) \geq 2^{-e_0(n)+1}$; that is, we can use the exponential-time algorithm for L_0 for those instances x with $\mu(x) < 2^{-e_0(n)+1}$, and this does not affect the polynomial-time computability discussion. Thus, in the following, we may assume that $\mu_0(x) \geq 2^{-e_0+1}$. (Again we use e_0 to denote $e_0(n)$.) On the other hand, for each $x \in \Sigma^n$, we have $\lambda\{\, s \,:\, G_0(s) = x \wedge G_0(s) \text{ takes more than } (2^{e_0} c_0 n)^{d_0} \text{ steps } \} \leq 2^{-e_0}$. Thus, even if we modify G_0 so that it terminates after $(2^{e_0} c_0 n)^{d_0}$ steps, the probability that x is generated is reduced by only 2^{-e_0}, which is at most the half of $\mu_0(x)$, since we have just assumed that $\mu_0(x) \geq 2^{-e_0+1}$. Hence, without losing generality, we may assume that the worst-case running time of $G_0(s)$ is bounded by $(2^{e_0} c_0 n)^{d_0}$.

From the condition (2), G_0 runs in polynomial-time on λ-average, which is witnessed by constants $c_0, d_0 > 0$. This intuitively means that the average running time of $G_0(s)$ is $(c_0 n)^{d_0}$. That is, with $t(u) = (2^u G_0 n)^{d_0}$, On the other hand, as explained above, the worst-case running time of $G_0(s)$ is bounded by $(2^{e_0} c_0 n)^{d_0}$. Hence, with $t(u) =$

$(2^u G_0 n)^{d_0}$, we can claim that the running time of G_0 for producing some string of length n is bounded by $t(0)$ on average, and bounded by $t(e_0)$ in the worst case. Thus, intuitively, for most of $s \in \Sigma^*$ such that $G_0(s) \neq \perp$, $G_0(s)$ halts in $t(0)$ steps, and the number of s for which $G_0(s)$ needs more than $t(u)$ steps becomes less and less when u increases.

For any u, $0 \leq u \leq e_0$, define $G_0^{(u)}$, $\mu_0^{(u)}$, and $\widehat{\mu_0}$ as follows:

$$G_0^{(u)}(s) \;=\; \begin{cases} G_0(s), & \text{if } (2^{u-1}c_0 n)^{d_0} < \mathrm{time}_{G_0}(s) \leq (2^u c_0 n)^{d_0}, \text{ and} \\ \perp, & \text{otherwise.} \end{cases}$$

(For $u = 0$ use the condition $\mathrm{time}_{G_0}(s) \leq (c_0 n)^{d_0}$ instead.)

$$\mu_0^{(u)}(x) \;=\; \lambda(\,(G_0^{(u)})^{-1}(x)\,). \quad \widehat{\mu_0}(u) \;=\; \sum_{x \in \Sigma^n} \mu_0^{(u)}(x).$$

Then the following relations are clear from the definition.

Fact 1.

(1) For any $x \in \Sigma^n$, we have $\mu_0(x) \;=\; \sum_{u=0}^{e_0} \mu_0^{(u)}(x)$. Hence, $\sum_{u=0}^{e_0} \widehat{\mu_0}(u) \;=\; 1$.

(2) For any u, $0 \leq u \leq e_0$, we have $\widehat{\mu_0}(u) \times 2^{u-1} \leq 1$.

It follows from the above (1) that for any x, there exists some u, $0 \leq u \leq e_0$, such that $\mu_0^{(u)}(x) \geq \mu_0(x)/(e_0+1)$. Let u_x denote it. Intuitively, for most of random seeds generating x, G_0 runs about $t(u_x)$ steps.

Here for the proof, we make use of one key lemma, which is provable by a straightforward modification of [IL90]. Let us first review [IL90]. Consider any set L in NP and any P-samplable distribution μ. Hence, μ has a *polynomial-time bounded* generator G. From the assumption that $\mathrm{NP} \subseteq \mathrm{P}_{\text{P-comp}}$, Impagliazzo and Levin constructed some randomized machine M such that (1) M recognizes L, and (2) M runs in polynomial-time on μ-average. By modifying their argument (see Appendix for the detail), we can construct the following M_1 for our L_0 and G_0.

Lemma 4.5. There exists a randomized machine M_1 with the following properties:

(1) For all $x \in \Sigma^n$ and $u \geq 0$,
  (i) $M_1(\langle n, \overline{t(u)}, x\rangle_{n,t(u)})$ outputs either 0, 1, or $\perp$,
  (ii) if $M_1(\langle n, \overline{t(u)}, x\rangle_{n,t(u)}) \neq \perp$, then $M_1(\langle n, \overline{t(u)}, x\rangle_{n,t(u)}) = 1 \iff x \in L_0$, and
  (iii) if $\mu_0^{(u)}(x) \neq 0$, then $\Pr_{M_1}\{\, M_1(\langle n, \overline{t(u)}, x\rangle_{n,t(u)}) \neq \perp \,\} \geq 2/3$.

(2) M_1 runs in polynomial time on μ_1-average, where $\mu_1(\langle n, \overline{t(u)}, x\rangle_{n,t(u)}) = \mu_0^{(u)}(x)$ for any u, $0 \leq u \leq e_0$. (We artificially set $\mu_1(\langle n, 0, 0\rangle_{n,t(u)}) = 1 - \widehat{\mu_0}(u)$, and $\mu_1(x') = 0$ for any other form of x'.)

Now by using M_1, we define the following algorithm M_2.

prog M_2 (**input** x);
  $n \leftarrow |x|$; $N \leftarrow p_1(n)$; % p_1 is sufficiently large polynomial.
  in parallel do
    (0) simulate N executions of $M_1(\langle n, \overline{t(0)}, x\rangle_{n,t(0)})$ in parallel
        by using N randomly chosen sequences as M_1's random resource;
        if M_1 yields 0/1 **then accept/reject**;
        $\vdots$
    (e_0) simulate N executions of $M_1(\langle n, \overline{t(e_0)}, x\rangle_{n,t(e_0)})$ in parallel
        by using N randomly chosen sequences as M_1's random resource;
        if M_1 yields 0/1 **then accept/reject**;
    ($*$) determine $x \in L_0$ by brute force deterministic computation;
        if the computation accepts/rejects **then accept/reject**
  od.

Clearly, this M_2 recognizes L_0 correctly. Thus, it suffices to show that M_2 runs in polynomial-time on μ_0-average. In the following discussion, we consider any sufficiently large n, and let it be fixed.

Let c_1 and d_1 are constants witnessing that M_1 is polynomial-time on μ_1-average. Thus, for any u, $0 \leq u \leq e_0$,

$$\sum_{x\in\Sigma^n} \frac{(time_{M_1}(\langle n, \overline{t(u)}, x\rangle_{n,t(u)}))^{1/d_1}}{|\langle n, \overline{t(u)}, x\rangle_{n,t(u)}|} \mu_1(\langle n, \overline{t(u)}, x\rangle_{n,t(u)}) \ \leq \ c_1.$$

Recall that $\mu_1(\langle n, \overline{t(u)}, x\rangle_{n,t(u)}) = \mu_0^{(u)}(x)$ and that $|\langle n, \overline{t(u)}, x\rangle_{n,t(u)}|$ is less than some polynomial in $n + t(u)$, which is bounded by $(2^{u-1}n)^{d_2}$ for some d_2. Thus, for any u,

$$\sum_{x\in\Sigma^n} \frac{(time_{M_1}(\langle n, \overline{t(u)}, x\rangle_{n,t(u)}))^{1/d_1}}{(2^{u-1}n)^{d_2}} \mu_0^{(u)}(x) \ \leq \ c_1.$$

Note also that $\sum_{x\in\Sigma^n} \mu_0^{(u)}(x) = \widehat{\mu_0}(u) \leq 1/2^{u-1}$ for any u. Thus, for any u, $0 \leq u \leq e_0$, it follows from Proposition 2.5 that

$$\sum_{x\in\Sigma^n} \frac{(time_{M_1}(\langle n, \overline{t(u)}, x\rangle_{n,t(u)}))^{1/(2d_1d_2)}}{n^{d_2}} \mu_0^{(u)}(x) \ \leq \ c_1 + 1.$$

Now for a given input $x \in \Sigma^n$, estimate the running time of $M_2(x)$. Here we focus on the u_xth parallel step; that is, the simulation of $M_1(\langle n, \overline{t(u_x)}, x\rangle_{n,t(u_x)})$. Recall that the probability that $M_1(\langle n, \overline{t(u_x)}, x\rangle_{n,t(u_x)}) \neq \perp$ is greater than 2/3, and

that $M_1(\langle n, \overline{t(u_x)}, x\rangle_{n,t(u_x)})$ is simulated by using N random sequences in parallel. Thus, with very high probability, some simulation of $M_1(\langle n, \overline{t(u_x)}, x\rangle_{n,t(u_x)})$ yields 0 or 1, and furthermore it halts in $2 \cdot \mathrm{time}_{M_1}(\langle n, \overline{t(u_x)}, x\rangle_{n,t(u_x)})$ steps. (Recall that $\mathrm{time}_{M_1}(\langle n, \overline{t(u_x)}, x\rangle_{n,t(u_x)})$ is the expected running time over all random sequences of M_1). Thus, we can assume that the running time of $M_2(x)$ is bounded by that of the u_xth parallel step, which is bounded by $2N\mathrm{time}_{M_1}(\langle n, \overline{t(u_x)}, x\rangle_{n,t(u_x)}) + p_2(2^{u_x}n)$ for some polynomial p_2.

Hence, for some constant $d_3 \geq 2d_1d_2$ such that $(p_2(2^{u_x}n))^{1/d_3} \leq 2^{u_x-1}n$ we have

$$\begin{aligned}
&\sum_{x:\, u_x=u} \frac{(\mathrm{time}_{M_2}(x))^{1/d_3}}{n}\mu_0(x) \\
&\leq\ (e_0+1)\cdot \sum_{x:\, u_x=u} \frac{(2N\mathrm{time}_{M_1}(\langle n, \overline{t(u_x)}, x\rangle_{n,t(u_x)}) + p_2(2^{u_x}n))^{1/d_3}}{n}\mu_0^{(u_x)}(x) \\
&\leq\ (e_0+1)n^{d_2-1}\cdot \sum_{x:\, u_x=u} \frac{(2N\mathrm{time}_{M_1}(\langle n, \overline{t(u_x)}, x\rangle_{n,t(u_x)}))^{1/d_3}}{n^{d_2}}\mu_0^{(u_x)}(x) \\
&\qquad\qquad + (e_0+1)\cdot \sum_{x:\, u_x=u} \frac{(p_2(2^{u_x}n))^{1/d_3}}{n}\mu_0^{(u_x)}(x) \\
&\leq\ (e_0+1)n^{d_2-1}(2N)^{1/d_3}(c_1+1) + (e_0+1)\cdot \frac{2^{u_x-1}n}{n}\cdot\frac{1}{2^{u_x-1}} \\
&\leq\ (e_0+1)\left(n^{d_2-1}(2N)^{1/d_3}(c_1+1)+1\right).
\end{aligned}$$

Hence, summing up this for $u = 0$ to $u = e_0(n)$, we have $\sum_{x\in\Sigma^n}(\mathrm{time}_{M_1}(x))^{1/d_2}/n \leq p_3(n)$ for some polynomial p_3. Therefore, from Proposition 2.4, we conclude that M_2 runs in polynomial-time on μ_1-average. □

Now with Theorem 4.4, the proof of Theorem 4.3 is easy.

Proof of Theorem 4.3. It suffices to show that under the assumption that $\Delta_2^{\mathrm{P}} \subseteq \mathrm{P}_{\text{P-comp}}$, every P^{NP}-samplable distribution is aveP-samplable. Consider any P^{NP}-samplable distribution μ. Then there exists some P^{NP}-generator G^X for μ. That is, G is a polynomial-time bounded deterministic machine, X is a set in NP, and they satisfies the following for some polynomial r and all $n \geq 0$ and $x \in \Sigma^n$:

$$\mu(x) \ =\ \lambda\{\, s \in \Sigma^{r(n)} \,:\, G^X(\overline{n}, s) = x \,\}.$$

Then define L by

$$L \ =\ \{\, \langle n, s, i, d\rangle_n \,:\, n \geq 0,\, s \in \Sigma^{r(n)},\, 1 \leq i \leq n,\, d \in \{0,1\},\, \text{and } (G^X(\overline{n}, s))_i = d \,\},$$

where $(G^X(\overline{n}, s))_i$ is the ith bit of $G^X(\overline{n}, s)$. Clearly, L is in Δ_2^{P}. Thus, it is polynomial-time decidable on average under uniform distribution. On the other hand, the function G^X (under uniform distribution) is $\propto_{\mathrm{tt}}^{\mathrm{P}}$-reducible to L (under uniform distribution). Therefore, G^X is computed by some G' that runs in polynomial-time on average under uniform distribution. □

5. The Second Question

In this section, we discuss the second question. That is, from the assumption NP $\subseteq$ $\mathrm{P}_{\text{P-comp}}$, how far can we prove? Or more specifically, for which distribution class $\mathcal{D}$, can we prove NP $\subseteq \mathrm{P}_{\mathcal{D}}$?

We first show that the assumption implies NP $\subseteq \mathrm{P}_{\mathrm{P}_{\mathrm{tt}}^{\mathrm{NP}}\text{-samp}}$.

Theorem 5.1. If NP $\subseteq \mathrm{P}_{\text{P-comp}}$, then NP $\subseteq \mathrm{P}_{\mathrm{P}_{\mathrm{tt}}^{\mathrm{NP}}\text{-samp}}$.

Proof. The proof is essentially the same as the one for Theorem 4.3. It follows from Proposition 3.3 that the assumption implies that $\mathrm{P}_{\mathrm{tt}}^{\mathrm{NP}} \subseteq \mathrm{P}_{\text{P-comp}}$. Then by exactly the same argument as the one for Theorem 4.3, we can show, for every $\mathrm{P}_{\mathrm{tt}}^{\mathrm{NP}}$-samplable distribution μ, that μ is aveP-samplable. Thus, by Theorem 4.4, every NP set is polynomial-time solvable on μ-average. That is, NP $\subseteq \mathrm{P}_{\mathrm{P}_{\mathrm{tt}}^{\mathrm{NP}}\text{-samp}}$. □

Therefore, the difference between $\mathrm{P}_{\mathrm{P}_{\mathrm{tt}}^{\mathrm{NP}}\text{-samp}}$ and $\mathrm{P}_{\mathrm{P}^{\mathrm{NP}}\text{-samp}}$ is essential for our original question, namely, the question of whether it holds that NP $\subseteq \mathrm{P}_{\text{P-comp}} \Longrightarrow$ OptP $\subseteq$ $\mathrm{P}_{\text{P-comp}}$. This leads us to another type of question. That is, the relation between $\mathrm{P}_{\mathrm{P}_{\mathrm{tt}}^{\mathrm{NP}}\text{-samp}}$ and $\mathrm{P}_{\mathrm{P}^{\mathrm{NP}}\text{-samp}}$, or in more general, between distribution classes $\mathrm{P}_{\mathrm{tt}}^{\mathrm{NP}}$-samp and P^{NP}-samp.

Here we show that the class $\mathrm{P}_{\mathrm{tt}}^{\mathrm{NP}}$-samp is in fact not so small by proving that it (essentially) contains some distribution class, i.e., the class of #P-computable distributions, which seems much stronger than P-comp. Let us first define the notion of "#P-computable distribution". Intuitively, an input distribution μ is a #P-computable if μ is defined as some #P function. That is, there exists a polynomial-time computable binary predicate R and a polynomial q such that for each x (let $n = |x|$),

$$\mu(x) \;=\; \frac{||\,\{\, w \in \Sigma^{q(n)} \mid R(x,w)\,\}\,||}{2^{q(n)}}.$$

Let #P-comp denote the class of #P-computable distributions.

For any two input distributions μ_1 and μ_2, we say that μ_1 *approximates* μ_2 *within constant factor* if $c_1\mu_1(x) \leq \mu_2(x) \leq c_2\mu_1(x)$ for some constants $c_1, c_2 > 0$ and for all $x \in \Sigma^*$. We have the following theorem.

Theorem 5.2. Every input distribution in #P-comp is approximated within constant factor by some input distribution in $\mathrm{P}_{\mathrm{tt}}^{\mathrm{NP}}$-samp.

For the proof we use hashing functions. Here we use linear hashing functions of Carter and Wegman [CW79] for our concrete hashing functions. A *linear hash function* h from Σ^l to Σ^m is given by a Boolean (m, l)-matrix $A = (a_{i,j})$, and maps any string $x = x_1 \ldots x_l \in \Sigma^l$ to some string $y = y_1 \ldots y_m$, where $y = Ax$ under modulo 2. Let $H_{l,m}$ be the set of linear hash functions from Σ^l to Σ^m. The following facts are basic properties of linear hash functions.

Fact 2. For all $x \in \Sigma^l$ and for all $y \in \Sigma^m$,

$$\mathrm{Pr}_{h \in H_{l,m}}\{\, h(x) = y \,\} \;=\; \frac{1}{2^m}.$$

Fact 3. For all $x_1, x_2 \in \Sigma^l$, $x_1 \neq x_2$, and for all $y_1, y_2 \in \Sigma^m$,

$$\begin{aligned} &\mathrm{Pr}_{h \in H_{l,m}}\{\, h(x_2) = y_2 \mid h(x_1) = y_1 \,\} = \frac{1}{2^m}, \quad \text{and thus} \\ &\mathrm{Pr}_{h \in H_{l,m}}\{\, h(x_1) = y_1 \wedge h(x_2) = y_2 \,\} = \frac{1}{2^{2m}}. \end{aligned}$$

We make use of the following lemma.

Lemma 5.3. Let X be any subset of Σ^l of size $K = 2^k$ for some $k \geq 0$, and let $m = k+c$ for some integer $c > 0$. For all $x \in X$ define P_x by

$$P_x = \mathrm{Pr}\{\; (h(x) = y) \;\wedge\; (\forall x' : x' \in X \wedge x' \neq x \,[\, h(x') \neq y \,]) \mid h \in H_{l,m},\; y \in \Sigma^m \;\}.$$

Then P_x is bounded as follows:

$$\left(1 - \frac{1}{2^c}\right) \cdot \frac{1}{2^c} \times \frac{1}{K} \;\leq\; P_x \;\leq\; \frac{1}{K}.$$

Proof. In the following, we assume that hashing function h is chosen from $H_{l,m}$ randomly and that the domain of y is Σ^m. First we have

$$\begin{aligned} P_x &= \sum_y 2^{-m} \cdot \mathrm{Pr}\{\; \forall x' : x' \in X \wedge x' \neq x \,[\, h(x') \neq y \,] \mid h(x) = y \;\} \cdot \mathrm{Pr}\{\; h(x) = y \;\} \\ &= \sum_y 2^{-m} \cdot 2^{-m} \left(1 - \mathrm{Pr}\{\; \exists x' \in X \,[\, x' \neq x \wedge h(x') = y \,] \mid h(x) = y \;\}\right) \\ &\geq \sum_y 2^{-m} \left(1 - \frac{||X|| - 1}{2^{k+c}}\right) \times \frac{1}{2^{k+c}} \;\geq\; \left(1 - \frac{1}{2^c}\right) \frac{1}{2^c} \times \frac{1}{K}. \end{aligned}$$

On the other hand, we have

$$P_x \;\le\; \Pr\{\; h(y) = x \mid y \in \Sigma^m,\; h \in H_{l,m} \;\} \;\le\; \sum_y 2^{-m} \cdot 2^{-m} \;=\; 2^{-m} \;\le\; \frac{1}{K}. \qquad \square$$

Jerrum, Valiant, and Vazirani [JVV86] showed a method to generate, for any set X in P and a given l, a string of length l in X with almost the same probability in polynomial-time by using some NP oracle. The above lemma gives a much simpler method to do the same task provided we know the size of $X \cap \Sigma^l$; in fact, the method uses an NP oracle only nonadaptively. This point is crucial for proving our theorem.

Proof of Theorem 5.2. Let μ be any input distribution in #P-comp. Then by definition, there exists a polynomial q and a polynomial-time computable binary predicate R such that for any $n \ge 0$ and any $x \in \Sigma^n$, $\mu(x) = ||\{\, w \in \Sigma^{q(n)} \mid R(x,w) \,\}||/2^{q(n)}$. We first define the following sets X_1 and X_2:

$$\begin{aligned}
X_1 \;=\; \{\; \langle n, h, y\rangle \;:\; & n \ge 0,\, h \in H_{n+q(n),q(n)+1},\, y \in \Sigma^{q(n)+1}, \text{ and} \\
& \exists x, w, x', w' :\, x, x' \in \Sigma^n \;\wedge\; w, w' \in \Sigma^{q(n)} \\
& [\; xw \ne x'w' \;\wedge\; h(xw) = h(x'w') = y \;\wedge\; R(x,w) \;\wedge\; R(x',w') \;] \;\}. \\
X_2 \;=\; \{\; \langle n, h, y, i, d\rangle \;:\; & \\
& n \ge 0,\, h \in H_{n+q(n),q(n)+1},\, y \in \Sigma^{q(n)+1},\, 1 \le i \le n+q(n),\, d \in \{0,1\}, \text{ and} \\
& \exists x, w :\, x \in \Sigma^n \wedge w \in \Sigma^{q(n)} \;[\; h(xw) = y \;\wedge\; R(x,w) \;\wedge\; (xw)_i = d \;] \;\}.
\end{aligned}$$

(where $(xw)_i$ is the ith bit of w)

Next consider the following randomized machine G, and for any $x \in \Sigma^n$, define $\mu'(x)$ to be the probability that $G(\overline{n})$ generates x.

prog G (**input** $\overline{n}$);
  generate h from $H_{n+q(n),q(n)+1}$ randomly;
  generate $y \in \Sigma^{q(n)+1}$ randomly;
  if $\langle n, h, y\rangle \in X_1$ **then** output ? and halt;
  use oracle X_2 to find some $xw \in \Sigma^{n+q(n)}$ with $h(xw) = y$ and $R(x,w)$;
  if no such xw exists **then** output ? **else** output x.

Then clearly both X_1 and X_2 are in NP. Furthermore, it is easy to modify G so that G asks only nonadaptive queries to $X_1 \oplus X_2 \in \mathrm{NP}$. Thus, G can be considered as a $\mathrm{P}^{\mathrm{NP}}_{\mathrm{tt}}$-generator. On the other hand, it follows from Lemma 5.3 that μ' approximates μ within constant factor. This almost proves the theorem.

Here, precisely speaking, μ' is not a real input distribution because μ' may assign some positive value to error symbol '?'. It is possible, however, to modify G to use polynomially many y's in parallel and thereby reducing the probability to yield '?' to less than $2^{-q(n)}$. Then G can output any string of length n instead of '?' while keeping

the property that μ' approximates μ within constant factor. The detail analysis is left to the reader. □

Notice that if μ_1 approximates μ_2 within constant factor, then the polynomial-time solvability is equivalent between μ_1-average and μ_2-average. Thus, we have the following corollary.

Corollary 5.4. $\mathrm{P}_{\#\mathrm{P}\text{-comp}} \subseteq \mathrm{P}_{\mathrm{tt}}^{\mathrm{NP}}$-samp.

Acknowledgments

The second author thanks to Professor Uwe Schöning for inviting him to the Universität Ulm, which made this joint research possible, and to the people in the Abteilung Theoretische Informatik (in particular, Uwe Schöning and Thomas Thierauf) for their warm hospitality to him. The authors have benefitted very much from valuable discussions with Professor Johannes Köbler, Professor Uwe Schöning, and Professor Thomas Thierauf.

References

[BDG88] J. Balcázar, J. Díaz, and J. Gabarró, *Structural Complexity I*, EATCS Monographs on Theoretical Computer Science, Springer-Verlag, 1988.

[BDG91] J. Balcázar, J. Díaz, and J. Gabarró, *Structural Complexity II*, EATCS Monographs on Theoretical Computer Science, Springer-Verlag, 1991.

[BCGL92] S. Ben-David, B. Chor, O. Goldreich, and M. Luby, On the theory of average case complexity, *J. Comput. Syst. Sci.* 44 (1992), 193−219.

[CW79] J. Carter and M. Wegman, Universal classes of hash functions, *J. Comput. Syst. Sci.* 18 (1979), 143−154.

[CG93] R. Chang and W. Gasarch, On bounded queries and approximation, in *Proc. 34th IEEE Sympos. on Foundations of Computer Science*, IEEE (1993), 547−556.

[Gur91] Y. Gurevich, Average case completeness, *J. Comput. Syst. Sci.* 42 (1991), 346−398.

[IL90] R. Impagliazzo and L. Levin, No better ways to generate hard NP instances than picking uniformly at random, in *Proc. 31st IEEE Sympos. on Foundations of Computer Science* (1990), 812–821.

[JVV86] M. Jerrum, L. Valiant, and V. Vazirani, Random generation of combinatorial structures from a uniform distribution, *Theoret. Comput. Sci.* 43 (1986), 169–188.

[Joh84] D. Johnson, The NP-completeness column: An on going guide, *J. Algorithms* 5 (1984), 284–299.

[Kre88] M. Krentel, The complexity of optimization problems, *J. Comput. Syst. Sci.* 36 (1988), 490–509.

[Lev86] L. Levin, Average case complete problems, *SIAM J. Comput.* 15 (1986), 285–286.

[SY92] R. Schuler and T. Yamakami, Structural average case complexity, in *Proc. 12th Foundations of Software Technology and Theoretical Computer Science*, Lecture Notes in Computer Science 652 (1992), 128–139.

Appendix: Proof of Lemma 4.5

Let us first review the proof of the following theorem (i.e., Proposition 3.3) proved in [IL90]. Then the lemma is proved by a straightforward modification of their proof.

Theorem A.1. [IL90] $\mathrm{NP} \subseteq \mathrm{P}_{\text{P-comp}} \Longrightarrow \mathrm{NP} \subseteq \mathrm{P}_{\text{P-samp}}$.

Suppose that $\mathrm{NP} \subseteq \mathrm{P}_{\text{P-comp}}$. Consider any μ_0 in $\mathrm{P}_{\text{P-samp}}$ and any NP set L_0, and we show that L_0 is polynomial-time decidable on μ_0-average.

We may assume some polynomial-time computable predicate W_0 and polynomial p_0 such that for all $x \in \Sigma^*$ (letting $n = |x|$), $x \in L_0 \Longleftrightarrow \exists w \in \Sigma^{p_0(n)} [\, W_0(x,w)\,]$. Since μ_0 is P-samplable, there exist some polynomial-time bounded *generator* G_0 for μ_0. That is, for some polynomial r_0 and for all $x \in \Sigma^n$,

$$\mu_0(x) \;=\; \frac{\| \{\, s \in \Sigma^{r_0(n)} \mid G_0(\overline{n}, s) = x \,\} \|}{2^{r_0(n)}}.$$

Now consider any sufficiently large n, and let us fix it in the following discussion. For simplifying our notation, we write $G_0(\overline{n}, s)$ as $G_0(s)$, and $r_0(n)$ and $p_0(n)$ as r_0 and p_0 respectively.

Here again we use linear hash functions. Recall that $H_{l,m}$ is the set of linear hash functions from Σ^l to Σ^m. For any $x \in \Sigma^n$, $h_1 \in H_{l_1,r_0}$, and $h_2 \in H_{n,l_2}$, we say that (h_1, h_2, z) *determines* x if

$$\begin{aligned}&\exists v \in \Sigma^{l_1} \,[\; h_1(v) = s \;\wedge\; f_0(s) = x \;\wedge\; h_2(x) = z \;] \\ &\wedge\, \forall v' \in \Sigma^{l_1} \,[\; (\; h_1(v') = s' \;\wedge\; f_0(v') = x' \neq x \;) \;\Longrightarrow\; h_2(x') \neq z \;].\end{aligned}$$

We show that for appropriate choice of l_1 and l_2, $(h_1, h_2, h_2(x))$ determines x with high probability. In the following, let $k_x = \lfloor\!|| \log G_0^{-1}(x) ||\!\rfloor$. Note that $2^{k_x} \leq || G_0^{-1}(x) || < 2^{k_x+1}$; hence, $2^{-(r_0-k_x)} \leq \mu_0(x) < 2^{-(r_0-k_x)+1}$.

Lemma A.2. For any $x \in \Sigma^n$, let $l_1 = r_0 - k_x - 1$ and $l_2 = l_1 + 2$. Then we have

$$P_0 \;=\; \Pr\{\, (h_1, h_2, h_2(x)) \text{ determines } x \mid h_1 \in H_{l_1,r_0}, h_2 \in H_{n,l_2} \,\} \;\geq\; \frac{1}{16}.$$

Proof. Let us fix our x, and let $z = h_2(x)$. In the following, we assume that h_1 (resp., h_2) is chosen from H_{l_1,r_0} (resp., H_{n,l_2}) uniformly at random. Define P_1 and P_2 as follows (P_2 varies depending on h_1):

$$\begin{aligned}P_1 \;&=\; \Pr_{h_1}\{\, \exists v \in \Sigma^{l_1} \,[\, G_0(h_1(v)) = x \,]\,\}, \quad \text{and} \\ P_2 \;&=\; \Pr_{h_2}\{\, \exists v_1, v_2 \in \Sigma^{l_1} \,[\, G_0(h_1(v_1)) \neq G_0(h_1(v_2)) \wedge \\ &\qquad\qquad h_2(G_0(h_1(v_1))) = h_2(G_0(h_1(v_2))) = z \,]\,\}.\end{aligned}$$

Then clearly, $P_0 \geq P_1 - \max_{h_1} P_2$.

First from the following inequalities, we show that $P_1 \geq 1/4$. (In the following, the range of s's and v's are $G_0^{-1}(x)$ and Σ^{l_1} respectively.)

$$\begin{aligned} P_1 &\geq \sum_{(s,v)} \Pr_{h_1}\{\, h_1(v) = s \,\} - \sum_{(s,v),(s',v')} \Pr_{h_1}\{\, h_1(v) = s \ \wedge\ h_1(v') = s' \,\} \\ &= \frac{\overbrace{||\, G_0^{-1}(x) \,|| \times 2^{l_1}}^{\#}}{2^{r_0}} - \frac{\#(\#-1)}{2} \times \frac{1}{(2^{r_0})^2} \ \geq\ \frac{1}{2}\left(1 - \frac{1}{2}\right) \ =\ \frac{1}{4}. \end{aligned}$$

For estimating P_2, let F be the range of $G_0 \circ h_1$. Then, for all h_1, $||\, F \,|| \leq 2^{l_1}$ and therefore we have the following.

$$\begin{aligned} P_2 &= \Pr_{h_2}\{\, \exists x_1, x_2 \in F\, [\, x_1 \neq x_2 \ \wedge\ h_2(x_1) = h_2(x_2) = z \,]\,\} \\ &\leq \sum_{x_1 \neq x_2 \in F} \Pr_{h_2}\{\, h_2(x_1) = h_2(x_2) = z \,\} \\ &\leq ||\, F \,||^2/(2^{l_2})^2 \ \leq\ 2^{2l_1 - 2l_2} \ =\ \frac{1}{16}. \end{aligned}$$

□

Now consider a procedure Q_0 stated below. The idea of Q_0 is as follows. For a given $x \in \Sigma^n$, let $G_0^{-1}(x)$ be the set of strings y such that $G_0(y) = x$. Choose k randomly from $\{0, 1, ..., r_0\}$; then with probability $1/r_0$, we have $k = k_x$ ($k_x = \lfloor \log ||G_0^{-1}(x)|| \rfloor$). If $k = k_x$, then by Lemma A.2, for randomly selected hash functions h_1 and h_2, $(h_1, h_2, h_2(x))$ determines x with probability $\geq 1/16$. That is, we can indirectly specify x by $(h_1, h_2, h_2(x))$, and thus, Q_0 can ask a right query to an oracle L_0', which is essentially the same as asking x to L_0.

prog Q_0 (**input** $x \in \Sigma^n$);
choose k randomly from $\{0, ..., r_0\}$;
$l_1 \leftarrow r_0 - k - 2$; $l_2 \leftarrow l_1 + 2$;
choose h_1 randomly from H_{l_1, r_0}; choose h_2 randomly from H_{n,l_2};
$y \leftarrow \langle n, k, h_1, h_2, h_2(x)\rangle_n$, and $y_i \leftarrow \langle n, k, h_1, h_2, h_2(x), i\rangle_n$ for each i, $1 \leq i \leq n$;
if $y \in EX_1 \wedge y \notin EX_2 \wedge \bigwedge_{i=1}^n (x_i = 1 \iff y_i \in EX_3)$
then if $y \in L_0'$ **then accept else reject**
else output ?.

Here oracle sets are defined as follows (here let $l_1 = r_0 - k - 2$, $l_2 = l_1 + 2$, $x = G_0(h_1(v))$, $x_1 = G_0(h_1(v_1))$, and $x_2 = G_0(h_1(v_2))$):

$$\begin{aligned} EX_1 &= \{\, \langle n, k, h_1, h_2, z\rangle_n \ :\ \exists v \in \Sigma^{l_1}\ [\ h_2(x) = z\ ]\,\} \\ EX_2 &= \{\, \langle n, k, h_1, h_2, z\rangle_n \ :\ \exists v_1, v_2 \in \Sigma^{l_1}\ [\ x_1 \neq x_2 \ \wedge\ h_2(x_1) = h(x_2) = z\ ]\,\}. \\ EX_3 &= \{\, \langle n, k, h_1, h_2, z, i\rangle_n \ :\ \exists v \in \Sigma^{l_1}\ [\ h_2(x) = z \ \wedge\ x_i = 1\ ]\,\}. \\ L_0' &= \{\, \langle n, k, h_1, h_2, z\rangle_n \ :\ \exists v \in \Sigma^{l_1},\ w \in \Sigma^{p_0}\ [\ z = h_2(x) \ \wedge\ W_0(x, w)\ ]\,\}. \end{aligned}$$

Following the above argument, we can easily show that $Q_0(x)$ gives a correct answer with probability $\geq 1/16r_0$. Note also that when $Q_0(x)$ says accept/reject, then the answer is always correct.

From the assumption that NP $\subseteq$ $\mathrm{P}_{\text{P-comp}}$, sets EX_1, EX_2, and L_0' are all solvable by some machines N_1, N_2, and N_3 respectively in polynomial-time under the following uniform distribution (in the following, we use H_{l_1,r_0} and H_{n,l_2} to denote $||\, H_{l_1,r_0} \,||$ and $||\, H_{n,l_2} \,||$):

$$\nu_1(\langle n,k,h_1,h_2,z\rangle_n) \;=\; \frac{1}{r_0 \cdot H_{l_1,r_0} \cdot H_{n,l_2} \cdot 2^{l_2}}.$$

Similarly, EX_3 is solvable by some machine N_4 in polynomial-time under the following uniform distribution:

$$\nu_2(\langle n,k,h_1,h_2,z,i\rangle_n) \;=\; \frac{1}{r_0 \cdot H_{l_1,r_0} \cdot H_{n,l_2} \cdot 2^{l_2} \cdot n}.$$

We estimate the computation time of Q_0 when using these three machines $N_1 \sim N_4$. (In the following, only $N_1 \sim N_3$ (under distribution ν_1) are investigated; the analysis for N_4 (under ν_2) is almost the same and thus omitted.) Unfortunately, we cannot claim that Q_0 runs in polynomial-time on μ_0-average. The problem is that for wrong choice of k, h_1, h_2, $Q_0(x)$ may ask a query y to $N_1 \sim N_3$ with probability much larger than $\nu_1(y)$. It is, however, possible to show that if $(h_1,h_2,h_2(x))$ determines x, then the probability that y is queried is close to $\nu_1(y)$.

Define $(\mu_0 \times \lambda)$ to be the probability that a string y is queried by $Q_0(x)$ when x is given under distribution μ_0. That is,

$$(\mu_0 \times \lambda)(\langle n,k,h_1,h_2,z\rangle_n) \;=\; \mu_0(x) \times \mathrm{Pr}_{Q_0}\{\; Q_0(x) \text{ queries } \langle n,k,h_1,h_2,z\rangle_n \;\}.$$

Here suppose that k is chosen correctly; that is, $k = k_x$. Then for any $(h_1,h_2,h_2(x))$ that determines x, we have

$$\begin{aligned}
(\mu_0 \times \lambda)(\langle n,k,h_1,h_2,h_2(x)\rangle_n) \;&=\; \mu_0(x) \cdot \frac{1}{H_{l_1,r_0}} \cdot \frac{1}{H_{n,l_2}} \;\approx\; 2^{-(r_0-k)} \cdot \frac{1}{H_{l_1,r_0}} \cdot \frac{1}{H_{n,l_2}} \\
&\approx\; 2^{-l_2} \cdot \frac{1}{H_{l_1,r_0}} \cdot \frac{1}{H_{n,l_2}} \;\approx\; \frac{1}{r_0 \cdot H_{l_1,r_0} \cdot H_{n,l_2} \cdot 2^{l_2}} \\
&=\; \nu_1(\langle n,k,h_1,h_2,h_2(x)\rangle_n),
\end{aligned}$$

where $\approx$ means equal up to a polynomial factor. Thus, for those $\langle n,k,h_1,h_2,h_2(x)\rangle_n$ that determine x, machines $N_1 \sim N_3$ (and similarly N_4) return the answer within reasonable amount of time. Or more specifically, if we estimate the time complexity of $Q_0^{N_1,N_2,N_3,N_4}$ only for the case where $k = k_x$ and $(h_1,h_2,h_2(x))$ determines x, then we can show that the running time is polynomial on μ_0-average.

Now for a sufficiently large polynomial p, consider a new procedure Q_0' that runs Q_0 for $p(n)$ times in parallel and outputs an answer if one of the executions of Q_0 returns the answer. Recall that there is not so small chance that $k = k_x$ and $(h_1, h_2, h_2(x))$ determines x. Thus, for any input x, the probability that it is determined with the correct k by some execution of Q_0 is very high. This is enough to prove that $(Q_0')^{N_1,N_2,N_3,N_4}$ runs in polynomial-time on μ_0-average. Therefore we have Theorem A.1.

Lemma 4.5 is proved almost the same way as above. Here we need to consider a generator G_0 that may not halt in a fixed polynomial-time. However, by considering time bounded version $G_0^{(u)}$ of G_0, we can easily show that $(Q_0')^{N_1,N_2,N_3,N_4}$ indeed satisfies the conditions (1) and (2).