Chicago Journal of Theoretical Computer Science

The MIT Press

Volume 1999, Article 4
25 March 1999

ISSN 1073–0486. MIT Press Journals, Five Cambridge Center, Cambridge, MA 02142-1493 USA; (617)253-2889; *journals-orders@mit.edu, journals-info@mit.edu*. Published one article at a time in LaTeX source form on the Internet. Pagination varies from copy to copy. For more information and other articles see:

- *http://mitpress.mit.edu/CJTCS/*
- *http://www.cs.uchicago.edu/publications/cjtcs/*
- *ftp://mitpress.mit.edu/pub/CJTCS*
- *ftp://cs.uchicago.edu/pub/publications/cjtcs*

The *Chicago Journal of Theoretical Computer Science* is abstracted or indexed in *Research Alert,®SciSearch,®Current Contents®/Engineering Computing & Technology*, and *CompuMath Citation Index.®*

©1999 The Massachusetts Institute of Technology. Subscribers are licensed to use journal articles in a variety of ways, limited only as required to insure fair attribution to authors and the journal, and to prohibit use in a competing commercial product. See the journal's World Wide Web site for further details. Address inquiries to the Subsidiary Rights Manager, MIT Press Journals; (617)253-2864; *journals-rights@mit.edu*.

The *Chicago Journal of Theoretical Computer Science* is a peer-reviewed scholarly journal in theoretical computer science. The journal is committed to providing a forum for significant results on theoretical aspects of all topics in computer science.

Editor-in-Chief: Janos Simon

Consulting Editors: Joseph Halpern, Stuart A. Kurtz, Raimund Seidel

Editors:

Martin Abadi	Greg Frederickson	John Mitchell
Pankaj Agarwal	Andrew Goldberg	Ketan Mulmuley
Eric Allender	Georg Gottlob	Gil Neiger
Tetsuo Asano	Vassos Hadzilacos	David Peleg
Laszló Babai	Juris Hartmanis	Andrew Pitts
Eric Bach	Maurice Herlihy	James Royer
Stephen Brookes	Ted Herman	Alan Selman
Jin-Yi Cai	Stephen Homer	Nir Shavit
Anne Condon	Neil Immerman	Eva Tardos
Cynthia Dwork	Howard Karloff	Sam Toueg
David Eppstein	Philip Klein	Moshe Vardi
Ronald Fagin	Phokion Kolaitis	Jennifer Welch
Lance Fortnow	Stephen Mahaney	Pierre Wolper
Steven Fortune	Michael Merritt	

Managing Editor: Michael J. O'Donnell

Electronic Mail: chicago-journal@cs.uchicago.edu

This paper is a contribution to a special issue on the 1996 Dagstuhl workshop on "Structure and Complexity" by Eric Allender.

The Complexity of Generating Test Instances*

Christoph Karg Johannes Köbler Rainer Schuler

25 March, 1999

Abstract

Abstract-1 Recently, Watanabe proposed a framework for testing the correctness and average-case performance of algorithms that purport to solve a given NP search problem efficiently on average with respect to some distribution on the instances. The idea is to randomly generate certified instances under some distribution that resembles the input distribution. Watanabe showed that unless RE = NE, test instances cannot be generated for some NP search problems. We further discuss Watanabe's approach and show, as an upper bound, that test instances can be generated for every NP search problem with non-adaptive queries to an NP oracle.

Abstract-2 We also introduce Las Vegas and Monte Carlo types of test instance generators and show that these generators can be used to find out (with high confidence) whether an algorithm is correct and efficient on average. It is shown that Monte Carlo generators can be easily constructed for all RP search problems and that Las Vegas generators exist for all ZPP search problems as well as for other problems such as prime factorization. On the other hand, we prove that Monte Carlo generators can only exist for problems in NP $\cap$ co-AM.

1 Introduction

1-1 The class NP is one of the most intensively studied classes in computational complexity theory, for it contains a large variety of problems that are of

*An extended abstract of the paper was presented at the Fourteenth Symposium on Theoretical Aspects of Computer Science (STACS), Springer-Verlag, *Lecture Notes in Computer Science* 1200, pp. 375–386, 1997.

practical importance. There are two types of problems that we are interested in, namely decision problems and search problems. Solving a decision problem means to decide whether a given instance has a solution, and solving a search problem means to actually find a solution. Although, at first glance, search problems seem more difficult to solve than decision problems, in many cases, they are equivalent under polynomial-time Turing reductions. For example, since any NP search problem is polynomial-time Turing reducible to any NP-complete (decision) problem, it follows that NP-complete problems have *self-computable witnesses* (in the sense of [BD76, Bal89]), meaning that solutions can be efficiently computed by asking oracle queries to the corresponding decision problem.

1-2 Since polynomial-time algorithms do not exist for NP-complete problems unless P = NP, it seems reasonable to look for algorithms that are *efficient in the average case* [Lev86] (see [Gur91] or [Wan97] for a survey). In practical applications, instances occur with certain probabilities. To model this process, we use probabilistic Turing machines that output instances in time polynomial in the length of the generated instances [BCGL92].

1-3 Besides designing efficient algorithms it is also important to find ways of testing the correctness (and performance) of a given algorithm. Ideally, one would like to give a proof that the algorithm is correct and efficient. However, in some cases, a mathematical proof may be hard to obtain, or the proof may be long and complex. Thus, in practical terms, it is reasonable to test algorithms by feeding them with carefully chosen instances for which the solutions are already known. An obvious approach is to use a probabilistic polynomial-time bounded algorithm that generates instances of the problem together with some solution. To be useful, these test instances should be generated according to some distribution that resembles the frequencies with which the instances occur in practice.

1-4 In this paper our objective is to verify the "global correctness" of a given algorithm A in the sense that, on a randomly selected input (according to the underlying distribution on the instances), the output of A is correct with high probability. This should be seen in contrast to the program checking approach in [BK95], where the outcome of the algorithm on a single instance has to be verified. Watanabe [Wat94] introduced a framework for globally checking the correctness and average-case behavior of algorithms. Intuitively, a generator for an NP search problem only needs to solve the problem on instances that are (randomly) *generated by the generator itself*, whereas an algorithm has to solve the problem on instances that are (randomly) *gener-*

ated "from outside." Thus, for some NP search problems it might be easier to find a generator than to find an algorithm that solves it. For example, it is not hard to construct a test instance generator for *prime factorization* under the standard distribution (which is uniform on each input length). On the other hand, no efficient on-average algorithm is known that on input of a binary number n, either outputs a factor of n in case n is composite, or outputs "prime" otherwise. In fact, this is the basis for the practical security of, for example, the RSA cryptosystem. Also, for certain random self-reducible problems [TW87, AFK89] it is possible to generate instances (along with some solution) according to the distribution induced by the self-reduction. Examples are the *quadratic residue* and the *discrete logarithm* problems.

1-5 A test instance generator, as introduced by Watanabe, is a probabilistic algorithm G that generates instances along with a solution; in particular, G only produces *positive instances*. Moreover, on input 0^n, G is required to output each positive instance of length n with sufficiently large probability (compared with the *conditional* distribution on the set of all positive instances of length n). As shown by Watanabe, there exist NP search problems for which test instances cannot be generated unless RE = NE. In fact, since for certain distributions the generator has to amplify the probability of the positive instances by an exponential factor, we can show that this is even true for very simple search problems.

1-6 In our model, a generator may produce positive as well as negative instances. As in Watanabe's approach, we require that the generator G output a witness along with every positively classified instance, i.e., G never misclassifies a negative instance. Moreover, each instance x has to be generated with a probability that is polynomially related to the probability $\mu(x)$ with which x occurs in practice. Thus, G is not required to amplify the probability of the positive instances. Furthermore, this condition implies that any algorithm is efficient on the generated instances if and only if it is efficient under the given distribution μ.

1-7 We define two types of test instance generators in this paper, namely, a Monte Carlo generator and a Las Vegas generator. While a Monte Carlo generator is allowed to misclassify a (positive) instance with small probability, a Las Vegas generator has to classify all generated instances correctly. We show how these generators can be used to decide efficiently (and with high confidence) whether a given algorithm A for an NP search problem is correct with respect to μ. Further, we describe how to construct Monte Carlo generators for all RP search problems as well as Las Vegas generators for all

ZPP search problems.

1-8 In general, whereas logarithmically many *adaptive* queries are useless, a polynomial number of *non-adaptive* queries to an NP set turn out to be sufficient to generate test instances for any NP search problem. This implies, in particular, that under the assumption NP = RP, any distributional NP search problem has a test instance generator.

1-9 Further, by an application of the universal hashing technique, we show that the domain of any search problem for which a Monte Carlo generator exists (under the standard probability distribution) necessarily belongs to co-AM. Similarly, Las Vegas generators can only exist for NP search problems whose domain belongs to NP ∩ co-NP. As a consequence, Monte Carlo or Las Vegas generators do not exist for search problems with an NP-complete domain, unless the polynomial-time hierarchy collapses.

1-10 Finally, we define a reducibility between NP search problems that, similar to the reducibility introduced by Levin [Lev86, BG93], preserves the probabilities of the instances. Via this reducibility we can obtain a test instance generator for a given NP search problem provided that we already have a test instance generator for some NP search problem to which the given one reduces.

2 Preliminaries

2-1 In this paper we use the standard notations and definitions of computational complexity theory (see, for example, [BDG95]). An introduction to the theory of computational average-case complexity can be found in [Gur91, BCGL92, Wan97]. Given below are some of the basic definitions we will use.

2-2 All languages considered here are over the alphabet $\Sigma = \{0,1\}$. The length of a string $x \in \Sigma^*$ is denoted by $|x|$. For a set A of strings, let $A^{=n} = A \cap \Sigma^n$ and $A^{\leq n} = \bigcup_{k=0}^{n} A^{=k}$. We denote the cardinality of a finite set A by $\|A\|$, and we write $A(x) = 1$, if $x \in A$, and $A(x) = 0$ otherwise. The pairing function $\langle \cdot, \cdot \rangle : \Sigma^* \times \Sigma^* \to \Sigma^*$ is defined as $\langle x, y \rangle = d(x)01y$, where $d(x_1 \dots x_n) = x_1 x_1 \dots x_n x_n$. Note that $\langle x, y \rangle$ is computable and invertible in polynomial time, and for all $x, y \in \Sigma^*$, $|\langle x, y \rangle| = 2|x| + |y| + 2$.

2-3 **Probability functions and distributions.** Let μ be a probability distribution on Σ^*. Associated with μ are a probability function that we also denote by μ and a distribution function, denoted by μ^*. More formally, μ and

μ^* are functions from Σ^* into the interval $[0,1]$ such that $\sum_x \mu(x) = 1$ and $\mu^*(x) = \sum_{y \leq x} \mu(y)$, where, as usual, $\leq$ denotes the lexicographic ordering on Σ^*. For any set $X \subseteq \Sigma^*$ let $\mu(X) = \sum_{x \in X} \mu(x)$. We refer to the distribution with the probability function $\mu_{\mathrm{st}}(x) = 1/(|x|(|x|+1)2^{|x|})$ as the standard distribution.

2-4 Let $\mu, \nu : \Sigma^* \to [0,1]$ be functions. We say that μ is polynomially dominated by ν (in symbols: $\mu \preceq \nu$) [Lev86] if, for some polynomial p and all x, $\mu(x) \leq p(|x|) \cdot \nu(x)$. In case $\mu \preceq \nu$ and $\nu \preceq \mu$, the distributions μ and ν are called equivalent (in symbols: $\mu \equiv \nu$) [WB95]. A function f from Σ^* to the non-negative integers is polynomial on μ-average [Lev86] if there exists a constant $\epsilon > 0$ such that

$$\sum_{x \neq \lambda} \frac{f^\epsilon(x)}{|x|} \cdot \mu(x) \quad < \quad \infty.$$

The set of functions that are polynomial on μ-average is closed under addition and multiplication. Furthermore, if μ is dominated by ν, then every function that is polynomial on ν-average is polynomial on μ-average [Lev86, Gur91].

2-5 A distribution is called p-computable if its distribution function μ^* is polynomial-time computable in the sense of Ko and Friedman [KF82], i.e., there exists a polynomial time bounded deterministic Turing transducer M such that, for all x and all k, it holds that $|M(x, 1^k) - \mu^*(x)| \leq 2^{-k}$. A distribution is called samplable [BCGL92] if there exists a probabilistic Turing machine M that on input λ outputs x with probability $\mu(x)$. If, in addition, M is polynomial-time bounded in the length of the output, then μ is called p-samplable. It is easy to see that the standard distribution μ_{st} is p-computable. Furthermore, every p-computable distribution is dominated by some p-samplable distribution [BCGL92, Theorem 7].

2-6 **Distributional NP search problems.** An NP search problem is specified by a binary polynomial-time decidable witness relation R and a polynomial q_R such that $R(x,w)$ implies $|w| = q_R(|x|)$. Any w for which $R(x,w)$ holds is called a solution (witness) for x. An instance x is called a positive instance in case it has a solution, and a negative instance, otherwise. We assume without loss of generality that λ is never a solution. A witness relation R also specifies an NP decision problem, namely $D_R = \{x \in \Sigma^* \mid \exists w : R(x,w)\}$. D_R is also called the domain of R. In case $D_R = \Sigma^*$, R is called a total NP search problem.

2-7 A distributional NP search problem is a pair (R, μ) consisting of a witness relation R and a p-samplable probability function μ. A (deterministic) al-

gorithm A efficiently solves (R, μ) if the running time of A is polynomial on μ-average and A computes a solution for all inputs x in the domain of R, whereas it outputs λ on all other inputs.

3 Generating Positive Test Instances

3-1 As pointed out in the introduction, the aim of test instance generation is to verify the quality (correctness and performance) of a given algorithm. In this section we will discuss Watanabe's approach to the test instance generation problem. Specifically, Watanabe requires the following properties of a test instance generator G for a distributional NP search problem (R, μ): G is a probabilistic polynomial-time algorithm, which, on input of 0^n, either outputs $\perp$ ("not successful") or generates some instance x of length n together with a solution w for x. Moreover, G has to output each positive instance with non-negligible probability compared with the conditional probability function ρ_n on the positive instances of length n. In particular, G is not allowed to output any negative instance, and hence, it is more appropriate to speak of G as a generator for the *promised* NP search problem restricted to the positive instances.

Definition 1 (Test Instance Generator [Wat94]**)** *Let (R, μ) be a distributional* NP *search problem and let D be the domain of R. A probabilistic polynomial-time Turing machine G is called a* test instance generator *for (R, μ) if there is a polynomial q such that for all n,*

1. *on every path, $G(0^n)$ outputs either $\perp$ or a pair $\langle x, w\rangle$ such that $R(x, w)$ and $x \in \Sigma^n$, and*
2. *if $\mu(D^{=n}) > 0$, then for every $x \in D^{=n}$, $G(0^n)$ outputs with probability at least $\rho_n(x)/q(n)$ a pair of the form $\langle x, w\rangle$, $w \in \Sigma^*$. Here, $\rho_n(x) = \mu(x)/\mu(D^{=n})$ is the probability of x under the condition that x belongs to $D^{=n}$.*

3-2 As shown by Watanabe, generators of this type can be used to detect, for a given input length n (in expected time polynomial in n and $1/\varepsilon$), that an algorithm makes an error, provided that errors occur with probability ε (with respect to ρ_n). Furthermore, they can be used to find out that an algorithm is not efficient (with respect to ρ_n) [Wat94]. More precisely, if G

is a test instance generator for a search problem (R, μ), then the quality of a given algorithm A with respect to ρ_n can be verified as follows.

1. If A makes an error on strings of length n with probability at least ε, then with probability at least ε, A makes at least one error on a polynomial-size sample S generated by G, i.e., the test instances in S are obtained by running $G(0^n)$ a polynomial number of times.

2. If A needs more than n^k steps on Σ^n with probability at least ε, then with probability at least ε, A needs more than n^k steps on some input from a polynomial-size sample S generated by G.

We note that, since in Definition 1 the probability that $G(0^n)$ outputs $\langle x, w\rangle$ is not bounded from above, an algorithm A may take exponential time on average under the distribution induced by the generator even if A is polynomial on ρ-average. In fact, it may happen that G produces with high probability instances that have probability zero with respect to ρ.

3-3 The following result, due to Watanabe, shows that it is unlikely that there exists a test instance generator for every distributional NP search problem.

Theorem 1 ([Wat94]) *If every* NP *search problem has a test instance generator under the standard distribution, then* RE = NE.

In fact, the proof shows that randomly computing instance/witness pairs for all NP search problems with a tally domain is impossible, unless RE = NE. Similarly, it can be seen that there are total NP search problems (i.e., with domain Σ^*) that do not have test instance generators under the standard distribution, unless NE∩co-NE ⊆ RE. This shows that generating test instances for an NP search problem for which decision is easy might nevertheless be hard.

3-4 We now show that generating test instances in Watanabe's model may be impossible even for trivial search problems since, for some distributions, it is necessary to amplify the probability of the positive instances of fixed length by an exponential factor.

Theorem 2 *Let* $R = \{\langle x, 1\rangle \mid x \in \Sigma^* - \{1\}^*\}$. *If for every p-samplable distribution* μ *there exists a test instance generator for* (R, μ), *then* RE = NE.

Proof of Theorem 2-1 **Proof of Theorem 2** Since RE = NE if and only if NP ∩ TALLY ⊆ RP [Boo74], it suffices to show that the assumption implies that every tally NP

set T is contained in RP. Let R_T be a binary relation and p be a polynomial such that for all n, $0^n \in T$ if and only if there exists a string $w \in \Sigma^{p(n)}$ with $R_T(0^n, w)$.

Proof of Theorem 2-2

Now consider the p-samplable distribution μ generated by the following probabilistic algorithm:

> First, guess a positive integer n with probability proportional to n^{-2}. Then, uniformly guess a string $w \in \Sigma^{p(n)}$ and output w in case $R_T(0^n, w)$ holds; else, output $1^{p(n)}$.

Then any test instance generator G for (R, μ) has the property that, if $0^n \in T$, then either $R_T(0^n, 1^{p(n)})$ holds or $G(0^{p(n)})$ outputs with probability $1/n^{O(1)}$ a pair $\langle w, 1\rangle$ for which $R_T(0^n, w)$ holds. Note that, in the latter case, $\rho_{p(n)}\{w \in \Sigma^{p(n)} \mid R_T(0^n, w)\} = 1$.

Proof of Theorem 2 □

3-5

An interesting question is whether the converse of Theorem 1 (or of Theorem 2) is also true. As shown below, non-adaptive queries to an NP set are sufficient to generate test instances for any distributional NP search problem. This implies in particular that under the assumption NP = RP, any distributional NP search problem has a test instance generator.

3-6

As shown in the following proposition, $O(\log n)$ many queries to any oracle are not helpful for a generator in Watanabe's model.

Proposition 1 *If, for a distributional* NP *search problem* (R, μ)*, test instances can be generated with* $O(\log n)$ *queries to some oracle, then there exists a test instance generator for* (R, μ)*.*

Proof of Proposition 1 Let M be a probabilistic oracle Turing machine that generates test instances by asking $O(\log n)$ queries to some oracle. Let M' simulate M where each oracle query is answered by a fair coin toss. If M outputs a pair $\langle x, w\rangle$, then M' outputs $\langle x, w\rangle$ only if $R(x, w)$ holds, and outputs $\perp$ otherwise. Then the probability that all oracle answers are guessed correctly is $1/n^{O(1)}$, implying that M' still fulfills the conditions of Definition 1.

Proof of Proposition 1 □

3-7 The upper bound on the complexity of generating test instances that we will give in Theorem 3 below is based on the universal hashing technique [CW79, Sip83, VV86]. Let $\mathcal{H}_{m,k}$ be the class of all linear hash functions from Σ^m to Σ^k. The following lemma is straightforward (cf. [Pap94]).

Lemma 1 *Let S be a subset of Σ^m of cardinality $\|S\| = s$ such that $2^k \leq 3s \leq 2^{k+1}$ and let $x \in S$. Then, for a uniformly chosen hash function h from $\mathcal{H}_{m,k}$, it holds with probability at least $2/(9s)$ that x is the only element in S with $h(x) = 0^k$. Hence, with probability at least $2/9$, there exists a unique $x \in S$ with $h(x) = 0^k$.*

Proof of Lemma 1 Since $\mathcal{H}_{m,k}$ is a universal class of hash functions, it follows for a uniformly chosen h that $\mathrm{Prob}\big[h(x) = 0^k\big] = 2^{-k}$ and that

$$\begin{aligned}&\mathrm{Prob}\big[h(x) = 0^k \wedge \exists y \in S - \{x\} : h(y) = 0^k\big]\\ &\leq \sum_{y \in S - \{x\}} \mathrm{Prob}\big[h(x) = 0^k \wedge h(y) = 0^k\big] \;<\; s2^{-2k}.\end{aligned}$$

Consequently, the probability that x is the only pre-image of 0^k in S can be calculated as follows.

$$\begin{aligned}&\mathrm{Prob}\big[h(x) = 0^k \wedge \forall y \in S - \{x\} : h(y) \neq 0^k\big]\\ &= \mathrm{Prob}\big[h(x) = 0^k\big] - \mathrm{Prob}\big[h(x) = 0^k \wedge \exists y \in S - \{x\} : h(y) = 0^k\big]\\ &> 2^{-k}(1 - s2^{-k})\\ &\geq 2/(9s), \text{ since } s2^{-k}(1 - s2^{-k}) \geq 2/9 \text{ for } 1/3 \leq s2^{-k} \leq 2/3.\end{aligned}$$

Proof of Lemma 1 □

Theorem 3 *For every distributional* NP *search problem (R, μ), test instances can be generated with parallel queries to some set in* NP*.*

Proof of Theorem 3 Let D be the domain of R and let q be a polynomial such that $|w| = q(|x|)$ for all solutions w for x. Let M be a probabilistic Turing machine witnessing that μ is p-samplable and let p be a polynomial time bound for M, i.e., the time used by $M(\lambda)$ to output a string x is bounded by $p(|x|)$. For any string $r \in \Sigma^{p(n)}$, let $M_r(\lambda)$ denote the output of M (if any) when using sequence r as random source. The test instance generator G for (R_D, μ) is defined as follows.

> On input 0^n, G first guesses two integers $k \in \{1, \ldots, p(n)+1\}$, $l \in \{1, \ldots, q(n)+1\}$ and two linear hash functions $h_1 \in \mathcal{H}_{p(n),k}$, $h_2 \in \mathcal{H}_{q(n),l}$. By asking in parallel the queries $y_i = \langle 0^n, i, h_1, h_2 \rangle$, $1 \leq i \leq m = 2p(n) + q(n) + 2$, to the oracle A, G obtains a sequence $s = A(y_1) \ldots A(y_m)$ of oracle answer bits. Then G determines strings $r \in \Sigma^{p(n)}$, $w \in \Sigma^{q(n)}$, and $x \in \Sigma^*$ such that $\langle r, w \rangle = s$ and $x = M_r(\lambda)$. Finally, if $|x| = n$ and $R(x, w)$ hold, then G outputs the pair $\langle x, w \rangle$; otherwise, G outputs the symbol $\perp$.

For every x, let $S_x = \{r \in \Sigma^{p(|x|)} \mid M_r(\lambda) = x\}$ denote the set of all random sequences r using which M outputs x, and let $S_n = \bigcup_{x \in D^{=n}} S_x$. For every $r \in S_n$ let α_r denote the probability that $G(0^n)$ guesses integers k, l and linear hash functions h_1, h_2 such that r is the only string in S_n with $h_1(r) = 0^k$ and there exists a unique solution w for $M_r(\lambda)$ with $h_2(w) = 0^l$. Then it follows by Lemma 1 that

$$\alpha_r \geq \frac{1}{(p(n)+1)(q(n)+1)} \cdot \frac{2}{9 \cdot \|S_n\|} \cdot \frac{2}{9}.$$

Now consider the following NP oracle set A defined as

$$\begin{aligned} \langle 0^n, i, h_1, h_2 \rangle \in A \;\Leftrightarrow\; & \exists \langle r, w \rangle : h_1(r) = 0^k,\ r \in S_n,\ h_2(w) = 0^l, \\ & R(M_r(\lambda), w), \text{ and the } i\text{th bit of } \langle r, w \rangle \text{ is 1.} \end{aligned}$$

Then it follows that, for every $x \in D^{=n}$, $G(0^n)$ outputs a pair of the form $\langle x, w \rangle$, $w \in \Sigma^*$, with probability at least

$$\sum_{r \in S_x} \alpha_r \;\geq\; \frac{4\,\|S_x\|}{81(p(n)+1)(q(n)+1)\,\|S_n\|} \;=\; \frac{4}{81(p(n)+1)(q(n)+1)} \cdot \rho_n(x).$$

Proof of Theorem 3 □

We end this section by observing that the above theorem can easily be extended to the Monte Carlo type of generators that we will introduce in the next section.

4 Test Instance Generation Without Amplification

4-1 In this section we propose a slightly different model of generating test instances. As explained above, our aim is to verify the quality of a given algorithm but to avoid the need for amplifying the probability of the positive instances by a possibly exponential factor. This goal is achieved by allowing the generator not only to output pairs $\langle x, w\rangle$, where x is a positive instance and w is a solution for x, but also to output pairs of the form $\langle x, \lambda\rangle$, where x might be negative as well. (Recall our assumption that λ is not a solution for any instance.) In the definition of a Monte Carlo generator, we also allow that positive instances x be generated (with small probability) in the form $\langle x, \lambda\rangle$, whereas this is totally forbidden for a Las Vegas generator. Before we proceed to the formal definition of these generators, let us explain why it might be easier to generate test instances for a distributional NP search problem than to solve the problem itself.

4-2 Intuitively, a generator for a distributional NP search problem can benefit from the advantage that it may intertwine the process of randomly generating an instance with the process of constructing a suitable solution. In contrast, an algorithm has to solve the problem on instances that are randomly generated "from outside." Thus, there might exist distributional NP search problems for which it is easier to generate instance/witness pairs than to solve the problem. In fact, this phenomenon is exploited in the design of many cryptographic protocols, such as in generating public keys together with some (secret) trapdoor information. For example, the security of the RSA cryptosystem is based on the practical hardness of the factorization problem, i.e., no efficient-on-average algorithm is known (under the standard distribution) that on input of n outputs a factor of n in case n is composite, and outputs "prime" otherwise. On the other hand, it is computationally easy to randomly generate composite numbers together with some factor. Indeed, it is not hard to construct a Las Vegas generator for the corresponding search problem (see Proposition 4).

4-3 Next we define the notion of an instance generator for a distributional NP search problem (R, μ). An instance generator G efficiently generates instances in accordance with the distribution μ. Furthermore, it provides, along with each generated instance x, either some solution w (implying that x belongs to the domain of R) or the empty string λ (implying absolutely

nothing on the membership status of x). In fact, an instance generator may only output pairs of the form $\langle x, \lambda \rangle$. In order to force G to generate "good" test instances, we will later impose an additional bound on the probability that G outputs a pair $\langle x, \lambda \rangle$ when x is positive.

Definition 2 (instance generator) *Let (R, μ) be a distributional* NP *search problem. A probabilistic Turing machine G (with output but with no input) is called an* instance generator *for R under μ if*

1. *on every halting computation, G outputs a pair $\langle x, w \rangle$ with the property that either $R(x, w)$ holds or $w = \lambda$,*
2. *the time needed by G to output a pair of the form $\langle x, w \rangle$, $w \in \Sigma^*$, is polynomially bounded in $|x|$, and*
3. *μ_G is equivalent to μ, where $\mu_G(x)$ is the probability that G generates instance x, i.e., $\mu_G(x) = \sum_{w \in \Sigma^*} \mu_G(x, w)$, where $\mu_G(x, w)$ is the probability that G outputs the pair $\langle x, w \rangle$.*

Note that if G were required to stop on all paths, then only finitely many instances could be generated. Actually, we allow that $\mu_G(\Sigma^*) < 1$, i.e., G might run forever with *positive* probability. Also, if $\mu(\Sigma^n) = 1/n^{O(1)}$, then G can easily be modified to produce instances of a given length according to the conditional probability of that length. (This is similar to Watanabe's model, but note that the generator might still output negative instances.)

4-4 Suppose that G is an instance generator for a distributional NP search problem (R, μ) and let A be an algorithm. Since the distributions μ and μ_G are equivalent, the running time of A is polynomial on μ-average if and only if it is polynomial on μ_G-average [Lev86].

4-5 We distinguish between two types of generators, depending on the quality of the generated test instances. A Monte Carlo generator is allowed to err with small probability, whereas a Las Vegas generator only generates test instances that are correctly classified.

Definition 3 (Monte Carlo generator) *Let G be an instance generator for a distributional* NP *search problem (R, μ) with domain D.*

1. *G is called a* Monte Carlo generator *for R under μ if for every polynomial q,*

$$\mu_G(x, \lambda) \leq \frac{\mu_G(x)}{q(|x|)}$$

holds for all but finitely many instances x in D.

2. *G is called a* Las Vegas generator *for R under μ if G never outputs a pair $\langle x, \lambda \rangle$ with $x \in D$.*

4-6 We note that if $\mu(D^{=n}) = 1/n^{O(1)}$, then any Monte Carlo generator for (R, μ) can be transformed into a test instance generator for (R, μ) according to Watanabe's model.

4-7 We next show how a Monte Carlo generator can be used to find out efficiently and with high confidence whether a given algorithm A for a distributional NP search problem (R, μ) is correct with respect to the underlying probability distribution μ. Without loss of generality, we can assume that A makes only errors on instances in the domain D of R. More formally, for any subset D' of D, we call the probability $\mu\{x \in D' \mid A(x) = \lambda\}$ that $x \in D'$ and A does not find a witness for x the error rate of A on D'. We say that a pair $\langle x, w \rangle$ convicts A on D' if $R(x, w)$ and $x \in D'$ hold, but $A(x) = \lambda$.

Theorem 4 *Let G be a Monte Carlo generator for a distributional* NP *search problem (R, μ) and let D be the domain of R. Then there exists a polynomial-time probabilistic transducer T such that the following statements hold for any search algorithm A.*

- *If the error rate of A on $D^{\leq l}$ is at least $1/m$, then for all n, T on input $\langle 1^n, 1^m, 1^l \rangle$ produces with probability at least $1 - 2^{-n}$ a sequence of pairs, at least one of which convicts A on $D^{\leq l}$.*
- *If additionally, A is polynomial on μ-average then for some polynomial t, the sequence contains with probability at least $1 - 2^{-n}$ a pair $\langle x, w \rangle$ that convicts A on $D^{\leq l}$ and for which $A(x)$ stops after at most $t(l, m)$ steps.*

Proof of Theorem 4-1 **Proof of Theorem 4** Let G be a Monte Carlo generator for (R, μ) and let q be a polynomial such that $|w| = q(|x|)$ for all solutions w of x. By Definition 2(3), there is a polynomial p such that for all instances x,

$$\mu(x)/p(|x|) \leq \mu_G(x) \leq p(|x|) \cdot \mu(x).$$

Since Definition 3(1) guarantees that for all but finitely many $x \in D$, $\mu_G(x, \lambda) \leq \mu_G(x)/2$, we can assume that, for all instances $x \in D$,

$$\sum_{w \in \Sigma^{q(|x|)}} \mu_G(x, w) \geq \mu_G(x)/2 \geq \mu(x)/(2p(|x|)).$$

Since the error rate of A on $D^{\leq l}$ is at least $1/m$, it follows that G outputs with probability at least $1/(2mp(l))$ a pair $\langle x, w\rangle$ with $x \in D^{\leq l}$ and $A(x) = \lambda$. Therefore, during $2mnp(l)$ independent computations, G will output with probability at least $1-(1-1/(2mp(l)))^{2mnp(l)} > 1-2^{-n}$ some pair $\langle x, w\rangle$ with $x \in D^{\leq l}$ and $A(x) = \lambda$. Note that since the running time of G is polynomially bounded in the length of the output, T can suspend the simulation of G after a polynomial number (in l) of steps.

Proof of Theorem 4-2

Note that since μ and μ_G are equivalent, the assumption that A is polynomial on μ-average implies that A is polynomial on μ_G-average, i.e., there exist constants $k, c > 0$ such that $\sum_{x\neq\lambda} \frac{\text{time}_A(x)^{1/k}}{|x|} \cdot \mu(x) < c$, where $\text{time}_A(x)$ denotes the running time of A on input x. Hence, it follows that

$$\mu_G\{x \in D^{\leq l} \mid A(x) \text{ runs for more than } t(l,m) \text{ steps}\} \leq 1/(4mp(l)),$$

where t is the polynomial $t(l,m) = (2cmlp(l))^k$. This implies that G outputs with probability at least $1/(4mp(l))$ a pair $\langle x, w\rangle$ that convicts A on $D^{\leq l}$ and for which $A(x)$ stops after at most $t(l,m)$ steps.

Proof of Theorem 4 □

4-8

It is interesting to note that the assumption in the above theorem can be weakened for certain distributions μ. In fact, if μ has the property that for some polynomial q and all n,

$$\textstyle\sum_{x:|x|>q(n)} \mu(x) \leq 1/n,$$

and if the error rate of A on D (instead of $D^{\leq l}$ as in the above theorem) is at least $1/m$, then it follows that $\mu\{x \in D^{\leq q(2m)} \mid A(x) = \lambda\} \geq 1/(2m)$. Hence, letting $T'(1^n, 1^m) = T(1^n, 1^{2m}, 1^{q(2m)})$, the above theorem implies that with probability at least $1 - 2^{-n}$, $T'(1^n, 1^m)$ produces an instance $x \in D^{\leq q(2m)}$ with $A(x) = \lambda$.

4-9

Next we consider distributional RP search problems (R, μ) (meaning that the binary relation R witnesses that its domain D_R belongs to RP, i.e., each $x \in D_R$ has at least $2^{q_R(|x|)-1}$ many solutions). As shown below, instance/witness pairs for (R, μ) can be easily generated by sampling a string x according to μ and randomly generating a witness for x.

Proposition 2 *For every distributional* RP *search problem* (R, μ) *there exists a Monte Carlo generator.*

Proof of Proposition 2 Let D be the domain of R and let q be a polynomial such that $|w| = q(|x|)$ for all solutions w of x. Let M be a probabilistic Turing machine witnessing that μ is p-samplable. Consider the following instance generator G for (R, μ).

> Simulate M. In case M outputs an instance x, $|x| = n$, guess randomly and independently a sequence $w_1, \ldots, w_n$ of strings of length $q(n)$ and determine the lexicographically largest string w in the set $\{\lambda\} \cup \{w_i \mid 1 \leq i \leq n \text{ and } R(x, w_i) \text{ holds}\}$. Output the pair $\langle x, w \rangle$.

It is easy to verify that G is a Monte Carlo test instance generator for (R, μ).

Proof of Proposition 2 □

4-10 As shown in the next theorem, it is unlikely that every NP search problem has a Monte Carlo generator under the standard probability distribution. In fact, Monte Carlo generators can only exist if the domain of the search problem belongs to co-AM. As a consequence, Monte Carlo generators do not exist for NP search problems with an NP-complete domain unless the polynomial-time hierarchy collapses to the second level [BHZ87, Sch88]. For definitions of the class AM and of Arthur-Merlin games we refer the reader to [BM88] or to a textbook, such as [BDG90] or [KST93].

Theorem 5 *If there exists a Monte Carlo generator for an* NP *search problem R under the standard distribution, then the domain of R is contained in* co-AM.

Proof of Theorem 5-1 **Proof of Theorem 5** Let G be a Monte Carlo generator for (R, μ_{st}) and let p be a polynomial bounding the running time of G. From Definition 2(3), we can conclude that, for some polynomial s,

$$\frac{1}{s(|x|)2^{|x|}} \leq \mu_G(x) \leq \frac{s(|x|)}{2^{|x|}}.$$

From Definition 3(1), we know that, for any polynomial q,

$$\mu_G(x, \lambda) \leq \frac{\mu_G(x)}{q(|x|)}$$

holds for all but finitely many x in the domain D of R. Now let S_x be the set of strings $r \in \Sigma^{p(n)}$ such that G outputs the pair $\langle x, \lambda \rangle$ when using r as

random source. Let $q(n)$ be the polynomial $2^5 s(n)^2$. Then, for all but finitely many x, the following implications are true (where $n = |x|$).

$$x \in D \;\Rightarrow\; \mu_G(x,\lambda) \leq \frac{s(n)}{2^n q(n)} \;\Rightarrow\; \|S_x\| \leq \frac{s(n)2^{p(n)}}{2^n q(n)} \;\Rightarrow\; \|S_x\| \leq \frac{2^{p(n)-n-5}}{s(n)},$$

$$x \notin D \;\Rightarrow\; \mu_G(x,\lambda) \geq \frac{1}{s(n)2^n} \;\Rightarrow\; \|S_x\| \geq \frac{2^{p(n)}}{s(n)2^n} \;\Rightarrow\; \|S_x\| \geq \frac{2^{p(n)-n}}{s(n)}.$$

Therefore, letting $k(n) = p(n) - n - \lceil \log_2 s(n) \rceil - 2$, it follows, for all but finitely many $x \in D$, that $\|S_x\| \leq 2^{k(n)-2}$ and for all but finitely many $x \notin D$, that $\|S_x\| \geq 2^{k(n)+2}$.

Proof of Theorem 5-2

Now consider for a fixed instance x (for which the above implications are true) the random variable Z that for a uniformly chosen h in $\mathcal{H}_{p(n),k(n)}$ gives the number of strings $r \in S_x$ for which $h(r) = 0^{k(n)}$. Then the expectation of Z is $E(Z) = 2^{-k(n)} \|S_x\|$ and the variance is $V(Z) = 2^{-k(n)}(1-2^{-k(n)}) \|S_x\| < E(Z)$ [VV86]. Thus, by using Chebyshev's inequality it follows that

$$x \in D \;\Rightarrow\; \text{Prob}[Z \geq 1] \leq E(Z) \leq 1/4,$$

$$x \notin D \;\Rightarrow\; \text{Prob}[Z = 0] \leq \text{Prob}[|Z - E(Z)| \geq E(Z)] \leq V(Z)/E(Z)^2 < 1/E(Z) \leq 1/4.$$

The Arthur-Merlin protocol for $\overline{D}$ proceeds as follows: On input x, $|x| = n$, Arthur randomly chooses a hash function $h \in \mathcal{H}_{p(n),k(n)}$ and asks Merlin to show him a string $r \in S_x$ with $h(r) = 0^{k(n)}$. Since Merlin succeeds with probability at least $3/4$, if $x \notin D$, and with probability at most $1/4$, otherwise, it follows that $\overline{D} \in$ AM.

Proof of Theorem 5 □

4-11

It is interesting to note that the above proof can easily be extended to any NP search problem (R, μ), where μ is positive and $1/\mu(x)$ can be *polynomially approximated* by an FP function $f(x)$, i.e., for some constant c and all x, $f(x) \leq 1/\mu(x) \leq f(x) \cdot |x|^c$.

4-12

By combining Theorem 5 with the result that NP is not contained in co-AM unless the polynomial-time hierarchy collapses to the second level [BHZ87, Sch88], we obtain the following corollary.

Corollary 1 *If there exists a Monte Carlo generator for some distributional* NP *search problem* (R, μ_{st}) *whose domain is* NP*-complete, then* PH $= \Sigma_2^P$.

4-13 We end this section by considering the Las Vegas type of generators. Define a ZPP search problem to be a RP search problem whose domain is in ZPP. Then a Las Vegas generator can be constructed for any ZPP search problem in a similar way as in the proof of Proposition 2.

Proposition 3 *For every distributional* ZPP *search problem there exists a Las Vegas generator.*

4-14 It is also interesting to observe that some search problems that are not known to be efficiently solvable have a Las Vegas generator. As an example, we consider the prime factorization problem under the standard distribution μ_{st}. We assume that numbers are given in binary, i.e., the string x is used to describe the number (denoted by n_x) with the binary representation $1x$. More precisely, let R_{prime} denote the witness relation defined as

$$R_{\text{prime}}(x,w) = \begin{cases} 1, & n_w \text{ is a prime factor of } n_x \text{ and } n_w < n_x, \\ 0, & \text{otherwise.} \end{cases}$$

Proposition 4 *There exists a Las Vegas generator for the prime factorization problem* (R_{prime}, μ_{st}).

Proof of Proposition 4-1 **Proof of Proposition 4** Recall that the domain of R_{prime} belongs to ZPP [AH87], i.e., there is a probabilistic polynomial-time algorithm A that on input of x either accepts (implying that n_x is prime) or rejects (implying that n_x is composite) or outputs "?" (with probability at most 1/2). Furthermore, since the standard distribution μ_{st} is equivalent to some p-samplable distribution, there is a probabilistic Turing machine M that on input λ outputs x with probability $\mu_M(x)$, where $\mu_{\text{st}}(x)/4 \leq \mu_M(x) \leq 4\mu_{\text{st}}(x)$ (cf. [BCGL92, Theorem 7]).

Proof of Proposition 4-2 Now consider the following generator G:

> With probability 1/2, execute step (1) or step (2).
>
> (1) Choose randomly under μ_M a string x in Σ^*. If $A(x)$ accepts, then output $\langle x, \lambda \rangle$. Otherwise, loop forever.
>
> (2) Choose randomly and independently under μ_M two strings $y, w \in \Sigma^*$. If $A(w)$ accepts and if $y \neq \lambda$, then output $\langle x, w \rangle$, where $1x$ is the binary representation of $n_y \cdot n_w$. Otherwise, loop forever.

It is easy to see that G fulfills conditions (1) and (2) of Definition 2 as well as condition (2) of Definition 3. Furthermore, it is not hard to verify that the distribution μ_G induced by G is equivalent to μ_{st} (i.e., G also fulfills Definition 2(3)), implying that G is a Las Vegas generator for $(R_{\mathrm{prime}}, \mu_{\mathrm{st}})$.

Proof of Proposition 4 □

4-15 On the other hand, Las Vegas generators can only exist for NP search problems whose domain belongs to NP ∩ co-NP.

Proposition 5 *If there exists a Las Vegas generator for an* NP *search problem R under the standard distribution μ_{st}, then the domain of R is contained in* co-NP.

Proof of Proposition 5 Let D be the domain of a search problem R and let G be a Las Vegas generator for (R, μ_{st}) whose running time is bounded by some polynomial s. We show that $\overline{D} \in$ NP. Consider the following NP machine M:

> On input x, simulate G for at most $s(|\langle x, \lambda\rangle|)$ steps and accept if G outputs the pair $\langle x, \lambda\rangle$.

Since $\mu_{\mathrm{st}}(x) > 0$ for all x and since G is a Las Vegas generator for (R, μ_{st}), M accepts x if and only if $x \in \overline{D}$.

Proof of Proposition 5 □

Corollary 2 *If there exists a Las Vegas generator for some distributional* NP *search problem (R, μ_{st}) whose domain is* NP*-complete, then* NP = co-NP.

4-16 As a final remark, it is not hard to see that, for every set L that has self-computable witnesses (in the sense of [BD76, Bal89]), there is a Las Vegas generator using L as an oracle. For example, since graph isomorphism (GI) and graph automorphism (GA) have self-computable witnesses and non-adaptively self-computable witnesses, respectively (cf. [Sch76] and [LT93, Lemmas 5.2 and 5.3]), there exist Las Vegas generators for the corresponding search problems that ask adaptive (non-adaptive) queries to GI (GA, respectively).

5 Hard Problems for Test Instance Generation

5-1 In this section we consider the question of how the test instance generation problem for one NP search problem (S, ν) can be reduced to the test instance generation problem for another NP search problem (R, μ). Inspired by [Wat94], we provide a way to transform a Monte Carlo or Las Vegas generator for (S, ν) to a generator for any problem (R, μ) that reduces to (S, ν) via the following kind of reduction.

Definition 4 *Let (R, μ) and (S, ν) be distributional* NP *search problems. Let D_R (D_S) be the domain of R (S, respectively). Then (R, μ) is* generator reducible *to (S, ν) if there exist functions $f, g \in$ FP and polynomials p, q such that:*

1. *For all y and r, if $f(y, r) \neq \perp$, then $f(y, r) \in D_R$ if and only if $y \in D_S$.*
2. *For all y, v, and r, if $f(y, r) \neq \perp$ and $S(y, v)$, then $R(f(y, r), g(y, v, r))$.*
3. *f is honest, i.e., for all y, r, if $f(y, r) \neq \perp$ then $p(|f(y, r)|) \geq |y|$.*
4. *μ is equivalent to ϕ, where ϕ is the distribution induced by ν, q, and f, i.e., $\phi(x) = \sum_{y,r} \nu(y) 2^{-|r|}$, where the sum ranges over all strings y, r such that $|r| = q(|y|)$ and $f(y, r) = x$.*

Two distributional NP search problems are called generator equivalent if they are generator reducible to each other.

5-2 In the sequel, we assume without loss of generality that $f(y, rr') = f(y, r)$ for all y, r, r' such that $|r| = q(|y|)$, and that $g(y, \lambda, r) = \lambda$ for all y, r.

Proposition 6 *If (R, μ) is generator reducible to (S, ν) and if there exists a Monte Carlo (Las Vegas) generator for (S, ν), then there exists a Monte Carlo (Las Vegas, respectively) generator for (R, μ).*

Proof of Proposition 6-1 **Proof of Proposition 6** Assume that (R, μ) reduces to (S, ν) via the functions f, g and the polynomials p, q. Let G be a generator for (S, ν). Consider the following algorithm G'.

> Simulate G. If G produces a pair $\langle y, v \rangle$, then randomly guess a string $r \in \{0,1\}^{q(|y|)}$. If $f(y, r) \neq \perp$, then output the pair $\langle f(y, r), g(y, v, r) \rangle$; otherwise, loop forever.

We claim that G' is an instance generator for (R, μ) (of the same type as G). By Definition 4 we know that if G' outputs $\langle x, w\rangle$, then either $R(x, w)$ or $w = \lambda$. Let $C(x) = \{(y, r) \mid f(y, r) = x, |r| = q(|y|)\}$ and let t and t' be polynomials such that $\mu_G(x) \geq \nu(x)/t(|x|)$ and $\phi(x) \geq \mu(x)/t'(|x|)$, where ϕ is the distribution induced by f, q, and ν. Then we have, for all x,

$$\begin{aligned}
\mu_{G'}(x) &= \sum_{(y,r)\in C(x)} \mu_G(y)\, 2^{-|r|} \\
&\geq \sum_{(y,r)\in C(x)} \frac{1}{t(|y|)}\nu(y)\, 2^{-|r|} \\
&\geq \frac{1}{t(p(|x|))} \underbrace{\sum_{(y,r)\in C(x)} \nu(y)\, 2^{-|r|}}_{\phi(x)} \\
&\geq \frac{1}{t(p(|x|))\, t'(|x|)}\mu(x).
\end{aligned}$$

Thus, $\mu \preceq \mu_{G'}$, and since $\mu_{G'} \preceq \mu$ can be derived in a similar way, $\mu_{G'} \equiv \mu$ follows. Furthermore, since f is honest, the time needed by G' to output a pair $\langle x, w\rangle$ is polynomially bounded in $|x|$.

Proof of Proposition 6-2

Assume now that G is a Monte Carlo generator for (S, ν). We have to show that for any polynomial t and almost all $x \in D_R$, $\mu_{G'}(x) \geq t(|x|) \cdot \mu_{G'}(x, \lambda)$. Let s be a polynomial such that $|f(y, r)| \leq s(|y|)$ for all y and r, $|r| = q(|y|)$. Since G is a Monte Carlo generator, it holds for almost all $y \in D_S$ that $\mu_G(y) \geq t(s(|y|)) \cdot \mu_G(y, \lambda)$. Then, for all sufficiently large $x \in D_R$ it follows that

$$\begin{aligned}
\mu_{G'}(x) &= \sum_{(y,r)\in C(x)} \mu_G(y)\, 2^{-|r|} \\
&\geq \sum_{(y,r)\in C(x)} t(s(|y|))\, \mu_G(y, \lambda)\, 2^{-|r|} \\
&\geq t(|x|) \sum_{(y,r)\in C(x)} \mu_G(y, \lambda)\, 2^{-|r|} \\
&= t(|x|) \cdot \mu_{G'}(x, \lambda).
\end{aligned}$$

Thus, it follows that G' is a Monte Carlo generator for (R, μ).

Proof of Proposition 6-3

In the case that G is a Las Vegas generator for (S, ν), it follows immediately from the properties of f and g that G' is a Las Vegas generator for

(R, μ).

Proof of Proposition 6 □

Another important property of the generator reducibility is transitivity.

Proposition 7 *The generator reducibility is transitive.*

Proof of Proposition 7 Let (R_i, μ_i), $i = 1, 2, 3$, be distributional NP search problems. Let D_i be the domain of R_i, $i = 1, 2, 3$. Assume that (R_i, μ_i) is generator reducible to (R_{i+1}, μ_{i+1}) via the functions $f_i, g_i \in \mathrm{FP}$ and the polynomials p_i, q_i, $i = 1, 2$. Consider the functions f and g defined as

$$f(y, r_1, r_2) = \begin{cases} f_1(f_2(y, r_2), r_1), & f_2(y, r_2) \neq \perp, \\ \perp, & \text{otherwise.} \end{cases}$$

and

$$g(y, v, r_1, r_2) = \begin{cases} g_1(f_2(y, r_2), g_2(y, v, r_2), r_1), & R_3(y, v), f(y, r_1, r_2) \neq \perp, \\ \lambda, & \text{otherwise.} \end{cases}$$

Since f_1 and f_2 satisfy the conditions of Definition 4, it follows that f and g fulfill the following conditions for all y, r_1, r_2 with $f(y, r_1, r_2) \neq \perp$.

1. $y \in D_3 \Leftrightarrow f_2(y, r_2) \in D_2 \Leftrightarrow \underbrace{f_1(f_2(y, r_2), r_1)}_{f(y, r_1, r_2)} \in D_1$,

2. $$\begin{aligned} R_3(y, v) &\Rightarrow R_2(f_2(y, r_2), g_2(y, v, r_2)) \\ &\Rightarrow R_1(\underbrace{f_1(f_2(y, r_2), r_1)}_{f(y, r_2 r_1)}, \underbrace{g_1(f_2(y, r_2), g_2(y, v, r_2), r_1))}_{g(y, v, r_2 r_1)}), \end{aligned}$$

3. $|y| \leq p_2(p_1(|f_1(f_2(y, r_2), r_1)|)) \leq p_2(|f_2(y, r_2)|)$, and

4. $\mu_1 \equiv \phi(x)$, where ϕ is the distribution induced by f, q, and μ_3. Indeed, since $|f_2(y, r_2)|$ is polynomially bounded in $|y|$, there exists a polynomial q such that $q_2(|y|) + q_1(|f_2(y, r_2)|) \leq q(|y|)$ for all y, r_2 with $f_2(y, r_2) \neq \perp$. For a given x, consider the following sets:

$$\begin{aligned} A(x) &= \{(y, r_1, r_2) \mid f(y, r_1, r_2) = x, |r_2| = q_2(|y|), |r_1| = q(|y|) - q_2(|y|)\}, \\ B(x) &= \{(y, r_1, r_2) \mid f(y, r_1, r_2) = x, |r_2| = q_2(|y|), |r_1| = q_1(|f_2(y, r_2)|)\}, \\ C_1(x) &= \{(z, r_1) \mid f_1(z, r_1) = x, |r_1| = q_1(|z|)\}, \\ C_2(z) &= \{(y, r_2) \mid f_2(y, r_2) = z, |r_2| = q_2(|y|)\}. \end{aligned}$$

For $i = 1, 2$, let ϕ_i be the distribution induced by f_i, q_i, and μ_{i+1}. Since $\mu_i \equiv \phi_i$, it follows that

$$\begin{aligned}
\phi(x) &= \sum_{(y,r_1,r_2)\in A(x)} \mu_3(y)2^{-|r_1r_2|} \\
&= \sum_{(y,r_1,r_2)\in B(x)} \mu_3(y)2^{-|r_1r_2|} \\
&= \sum_{(z,r_1)\in C_1(x)} 2^{-|r_1|} \cdot \underbrace{\sum_{(y,r_2)\in C_2(z)} \mu_3(y)2^{-|r_2|}}_{\phi_2(z)} \\
&\geq \sum_{(z,r_1)\in C_1(x)} 2^{-|r_1|}\frac{1}{p(|z|)}\mu_2(z) \qquad (\text{since } \mu_2 \equiv \phi_2) \\
&\geq \frac{1}{p'(|x|)} \underbrace{\sum_{(z,r_1)\in C_1(x)} 2^{-|r_1|}\mu_2(z)}_{\phi_1(x)} \qquad (\text{since } f_1 \text{ is honest}) \\
&\geq \frac{1}{p''(|x|)}\mu_1(x) \qquad (\text{since } \mu_1 \equiv \phi_1),
\end{aligned}$$

where p, p' and p'' are appropriately chosen polynomials. Thus, we have $\mu_1 \preceq \phi$, and since $\phi \preceq \mu_1$ can be shown similarly, it follows that $\mu_1 \equiv \phi$.

Using the properties derived above it is easy to see that (R_1, μ_1) is generator reducible to (R_3, μ_3) via the functions $\hat{f}, \hat{g}$, where

$$\hat{f}(y, r) = \begin{cases} f(y, \lambda, r), & |r| < q_2(|y|), \\ f(y, r_1, r_2), & r = r_1r_2, |r_2| = q_2(|y|). \end{cases}$$

and $\hat{g}$ is defined analogously.

Proof of Proposition 7 □

Next we present two problems from group theory that are generator equivalent. For this we need some notions from elementary group theory. For a more detailed introduction to group theory, we refer to [Hum96, Cam95]. Our reductions are based on the ones given in [Hof82].

5-3 S_n denotes the symmetric group of $\{1,\ldots,n\}$, i.e. the set of all permutations over $\{1,\ldots,n\}$. For a set $A \subseteq S_n$, we denote by $\langle A\rangle$ the smallest subgroup of S_n that contains A. A is called a generator set for $\langle A\rangle$. It is known that every subgroup of S_n can be generated by a set of at most $n-1$ permutations [Jer82].

Double Coset Membership $(\mathrm{DCM}, \mu_{\mathrm{DCM}})$

Instance. Positive integer n (given in unary), sets $A, B \subseteq S_n$ such that $1 \leq \|A\|, \|B\| \leq n-1$ and permutations $\pi, \psi \in S_n$.

Question. Do there exist $\alpha \in \langle A\rangle$, $\beta \in \langle B\rangle$ such that $\psi = \alpha\pi\beta$?

Distribution. Randomly and independently choose a positive integer n (with probability $\frac{1}{n(n+1)}$); then guess randomly and independently two integers $a, b \in \{1,\ldots,n-1\}$ under uniform distribution. Guess randomly and uniformly $a+b+2$ permutations from S_n. The density function is

$$\mu_{\mathrm{DCM}}(1^n, A, B, \pi, \psi) = \frac{1}{n(n+1)\,(n-1)^2\,(n!)^{\|A\|+\|B\|+2}}.$$

Group Factorization $(\mathrm{GF}, \mu_{\mathrm{GF}})$

Instance. Positive integer n (given in unary), sets $A, B \subseteq S_n$ such that $1 \leq \|A\|, \|B\| \leq n-1$ and a permutation $\psi \in S_n$.

Question. Do there exist $\alpha \in \langle A\rangle$, $\beta \in \langle B\rangle$ such that $\psi = \alpha\beta$?

Distribution. Randomly and independently choose a positive integer n as above; then guess randomly and independently two integers $a, b \in \{1,\ldots,n-1\}$ under uniform distribution. Guess randomly and uniformly $a+b+1$ permutations from S_n. The density function is

$$\mu_{\mathrm{GF}}(1^n, A, B, \psi) = \frac{1}{n(n+1)\,(n-1)^2\,(n!)^{\|A\|+\|B\|+1}}.$$

Let R_{DCM} (R_{GF}) denote the search problem of DCM (GF, respectively).

Proposition 8 *$(R_{\mathrm{DCM}}, \mu_{\mathrm{DCM}})$ and $(R_{\mathrm{GF}}, \mu_{\mathrm{GF}})$ are generator equivalent.*

Proof of Proposition 8-1

Proof of Proposition 8 We first show that $(R_{\mathrm{GF}}, \mu_{\mathrm{GF}})$ is generator reducible to $(R_{\mathrm{DCM}}, \mu_{\mathrm{DCM}})$. Since this reduction makes no use of the random bits, we omit the parameter r. For any instance $\langle 1^n, A, B, \pi, \psi\rangle$ of DCM and potential solution $\langle \alpha, \beta\rangle$, the reduction (f, g) is defined as

$$\begin{aligned} f(1^n, A, B, \pi, \psi) &= \langle 1^n, \pi^{-1}A\pi, B, \pi^{-1}\psi\rangle \\ g(\langle 1^n, A, B, \pi, \psi\rangle, \langle \alpha, \beta\rangle) &= \begin{cases} (\pi^{-1}\alpha\pi, \beta) & \alpha \in \langle A\rangle, \beta \in \langle B\rangle, \psi = \alpha\pi\beta, \\ \lambda & \text{otherwise.} \end{cases} \end{aligned}$$

Since for a given set A of permutations and a given permutation α it can be decided in polynomial time whether α belongs to $\langle A\rangle$ (see, for example, [Hof82]), it is easy to verify that this is indeed a generator reduction from $(R_{\mathrm{GF}}, \mu_{\mathrm{GF}})$ to $(R_{\mathrm{DCM}}, \mu_{\mathrm{DCM}})$. In fact, the density function induced by f and μ_{DCM} coincides with μ_{GF}.

Proof of Proposition 8-2

Next we argue that $(R_{\mathrm{DCM}}, \mu_{\mathrm{DCM}})$ is generator reducible to $(R_{\mathrm{GF}}, \mu_{\mathrm{GF}})$. For any instance $\langle 1^n, A, B, \psi\rangle$ of GF and potential solution $\langle \alpha, \beta\rangle$, the reduction (f, g) is defined as

$$\begin{aligned} f(\langle 1^n, A, B, \psi\rangle, r) &= \begin{cases} \langle 1^n, A, \pi^{-1}B\pi, \pi, \psi\pi\rangle, & r \text{ describes a } \pi \in S_n, \\ \perp, & \text{otherwise,} \end{cases} \\ g(\langle 1^n, A, B, \psi\rangle, \langle \alpha, \beta\rangle, r) &= \begin{cases} \langle \alpha, \pi^{-1}\beta\pi\rangle, & \alpha \in \langle A\rangle, \beta \in \langle B\rangle, \psi = \alpha\beta, \\ & r \text{ describes a } \pi \in S_n, \\ \lambda, & \text{otherwise.} \end{cases} \end{aligned}$$

Again, it is easy to verify that this is indeed a generator reduction. We remark that the density function induced by f and μ_{GF} is equal to μ_{DCM}.

Proof of Proposition 8 □

5-5

Finally, we show the existence of a hard problem in the sense that every distributional NP search problem (R, μ) is generator reducible to it, provided the distribution μ is p-computable and the probabilities $\mu(x)$ are not too small.

Bounded Halting Problem (K, μ_K)

Instance. A nondeterministic Turing machine M, a binary string x, and a positive integer k, given in unary.

Question. Does M accept x within k steps?

Distribution. Randomly and independently guess M, x and k under standard distribution, i.e., the density function μ_K is

$$\mu_K(M, x, 1^k) = \frac{1}{|M|(|M|+1)\,2^{|M|}\,|x|(|x|+1)\,2^{|x|}\,k(k+1)}.$$

The definition of the search problem R_K is straightforward: $R_K(\langle M, x, 1^k\rangle, v)$ holds if and only if v describes an accepting computation of M on input x of length $\leq k$.

Theorem 6 *Let R be an* NP *search problem and let μ be a p-computable distribution such that for some polynomial p and all x, $\mu(x) \geq 2^{-p(|x|)}$. Then (R, μ) is generator reducible to (R_K, μ_K).*

Proof of Theorem 6-1

Proof of Theorem 6 Let (R, μ) be a distributional NP search problem such that μ is p-computable. Let D be the domain of R.

Proof of Theorem 6-2

It is known [Lev86, Gur91, WB95] that D is many-one reducible to K via a function $h \in$ FP that is one-one, length increasing, polynomial-time invertible and for which $\mu(x)$ and $\mu_K(h(x))$ are equivalent. Let f be an FP function such that $h(f(y)) = y$ for all $y \in \mathit{range}(h)$. Furthermore, it is possible to compute in polynomial time from a given witness v for $h(x)$ some witness w for x, i.e., there exists a function $g \in$ FP such that $R(x, g(h(x), v))$ holds for every x, v with $R_K(h(x), v)$. Thus, for every $y \in \mathit{range}(h)$, the following statements hold.

1. $f(y) \in D \Leftrightarrow y \in K$,
2. $R_K(y, v) \Rightarrow R(f(y), g(y, v))$,
3. $p(|f(y)|) \geq |y|$ (since $h(f(y)) = y$), and
4. $\mu(x)$ is polynomially related to $\sum_{y, f(y)=x} \mu_K(y)$.

This implies that (R, μ) generator reduces to (R_K, μ_K) via $\hat{f}$ and $\hat{g}$, where

$$\hat{f}(y, r) = \begin{cases} f(y) & y \in \mathit{range}(h) \\ \perp & \text{otherwise} \end{cases}$$

and $\hat{g}$ is defined analogously.

Proof of Theorem 6 □

Remark. In the proof of Theorem 6 we used the completeness of the bounded halting problem for the class DistNP consisting of all pairs (L, μ) such that L is in NP and μ is p-computable [Lev86, Gur91]. In fact, the proof also works for other DistNP-complete problems such as the tiling problem, Post's correspondence problem, the word problem for Thue systems and groups, and LR(k) testing of context-free grammars [Lev86, Gur91, WB95, Wan95, Kar97].

5-7 On the other hand, it remains open whether Theorem 6 can be extended to p-samplable distributions μ. Note that there exists a similar open problem in average-case theory, namely, whether every distributional NP problem with a p-samplable distribution many-one reduces to the bounded halting problem [BG95]. Interestingly, under randomized truth-table reductions, every distributional NP (decision and search) problem with a p-samplable distribution reduces to the bounded halting problem [IL90, BCGL92].

Acknowledgments

We would like to thank the anonymous referees for helpful comments.

Acknowledgment of support: Christoph Karg's work was supported by the Deutsche Forschungsgemeinschaft, Research Grant Schö 302/4-2.

References

[AFK89] M. Abadi, J. Feigenbaum, and J. Kilian. On hiding information from an oracle. *Journal of Computer and System Sciences*, 39:21–30, 1989.

[AH87] L. Adleman and M. Huang. Recognizing primes in random polynomial time. In *Proceedings of the 19th ACM Symposium on Theory of Computing*, pages 462–469, New York, 1987. ACM Press.

[Bal89] J. L. Balcázar. Self-reducibility structures and solutions of NP problems. In *Revista Matematica*, volume 2, No. 2/3, pages 175–184. Universidad Complutense de Madrid, Madrid, 1989.

[BCGL92] S. Ben-David, B. Chor, O. Goldreich, and M. Luby. On the theory of average case complexity. *Journal of Computer and System Sciences*, 44:193–219, 1992.

[BD76] A. Borodin and A. Demers. Some comments on functional self-reducibility and the NP hierarchy. Technical Report 76-284, Department of Computer Science, Cornell University, 1976.

[BDG90] J. L. Balcázar, J. Díaz, and J. Gabarró. *Structural Complexity II*. EATCS Monographs on Theoretical Computer Science. Springer-Verlag, Berlin, 1990.

[BDG95] J. L. Balcázar, J. Díaz, and J. Gabarró. *Structural Complexity I*. EATCS Monographs on Theoretical Computer Science. Springer-Verlag, Berlin, second edition, 1995.

[BG93] A. Blass and Y. Gurevich. Randomized reductions of search problems. *SIAM Journal of Computing*, 22:949–975, 1993.

[BG95] A. Blass and Y. Gurevich. Matrix transformation is complete for the average case. *SIAM Journal on Computing*, 24(1):3–29, 1995.

[BHZ87] R. Boppana, J. Hastad, and S. Zachos. Does co-NP have short interactive proofs? *Information Processing Letters*, 25:27–32, 1987.

[BK95] M. Blum and S. Kannan. Designing programs to check their work. *Journal of the ACM*, 42(1):269–291, 1995.

[BM88] L. Babai and S. Moran. Arthur-merlin games: a randomized proof system and a hierarchy of complexity classes. *Journal of Computer and System Sciences*, 36:254–276, 1988.

[Boo74] R. Book. Tally languages and complexity classes. *Information and Control*, 26:186–193, 1974.

[Cam95] P. J. Cameron. Permutation groups. In R. Graham, M. Grötschel, and L. Lovász, editors, *Handbook Of Combinatorics*, chapter 12, pages 611–645. Elsevier, Amsterdam, 1995.

[CW79] J. L. Carter and M. N. Wegman. Universal classes of hash functions. *Journal of Computer and System Sciences*, 18:143–154, 1979.

[Gur91] Y. Gurevich. Average case completeness. *Journal of Computer and System Sciences*, 42(3):346–398, 1991.

[Hof82] C. M. Hoffmann. *Group-Theoretic Algorithms and Graph Isomorphism*, volume 136 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1982.

[Hum96] J. F. Humphreys. *A Course in Group Theory*. Oxford Science Publications, Oxford, 1996.

[IL90] R. Impagliazzo and L. A. Levin. No better ways to generate hard NP-instances than picking uniformly at random. In *Proceedings of the 31st IEEE Symposium on the Foundations of Computer Science*, pages 812–821, Los Alamitos, CA, 1990. IEEE Computer Society Press.

[Jer82] M. Jerrum. A compact representation for permutation groups. In *Proceedings of the 23rd IEEE Symposium on the Foundations of Computer Science*, pages 126–133, Los Alamitos, CA, 1982. IEEE Computer Society Press.

[Kar97] C. Karg. LR(k) testing is average-case complete. In *Proceedings of the 12th Annual IEEE Conference on Computational Complexity*, pages 74–80, Los Alamitos, CA, 1997. IEEE Computer Society Press.

[KF82] K.-I. Ko and H. Friedman. Computational complexity of real functions. *Theoretical Computer Science*, 20:323–352, 1982.

[KST93] J. Köbler, U. Schöning, and J. Torán. *The Graph Isomorphism Problem: Its Structural Complexity*. Birkhäuser, Boston, 1993.

[Lev86] L. Levin. Average case complete problems. *SIAM Journal on Computing*, 15:285–286, 1986.

[LT93] A. Lozano and J. Torán. On the nonuniform complexity of the graph isomorphism problem. In K. Ambos-Spies, S. Homer, and U. Schöning, editors, *Complexity Theory, Current Research*, pages 1–45. Cambridge University Press, Cambridge, 1993.

[Pap94] C. Papadimitriou. *Computational Complexity*. Addison-Wesley, Reading, MA, 1994.

[Sch76] C. Schnorr. Optimal algorithms for self-transformable problems. In *Proceedings of the 3rd International Colloquium on Automata, Languages, and Programming*, pages 322–337, Edinburgh, 1976. Edinburgh University Press.

[Sch88] U. Schöning. Graph isomorphism is in the low hierarchy. *Journal of Computer and System Sciences*, 37:312–323, 1988.

[Sip83] M. Sipser. A complexity theoretic approach to randomness. In *Proceedings of the 15th ACM Symposium on Theory of Computing*, pages 330–335, New York, 1983. ACM Press.

[TW87] W. Tompa and H. Woll. Random self-reducibility and zero-knowledge interactive proofs of possession of information. In *Proceedings of the 28th IEEE Symposium on the Foundations of Computer Science*, pages 472–482, Los Alamitos, CA, 1987. IEEE Computer Society Press.

[VV86] L. Valiant and V. Vazirani. NP is as easy as detecting unique solutions. *Theoretical Computer Science*, 47:85–93, 1986.

[Wan95] J. Wang. Average-case completeness of a word problem for groups. In *Proceedings of the 27th ACM Symposium on Theory of Computing*, pages 325–334, New York, 1995. ACM Press.

[Wan97] J. Wang. Average-case computational complexity theory. In L. A. Hemaspaandra and A. L. Selman, editors, *Complexity Theory Retrospective 2*, pages 295–328. Springer-Verlag, Berlin, 1997.

[Wat94] O. Watanabe. Test instance generation for promised NP search problems. In *Proceedings of the 9th Structure in Complexity Theory Conference*, pages 205–216, Los Alamitos, CA, 1994. IEEE Computer Society Press.

[WB95] J. Wang and J. Belanger. On the NP-isomorphism problem with respect to random instances. *Journal of Computer and System Sciences*, 50:151–164, 1995.