# The Complexity of Generating Test Instances

Christoph Karg* Johannes Köbler
Rainer Schuler
Abteilung Theoretische Informatik,
Universität Ulm,
89069 Ulm, Germany

February 5, 1997

**Abstract**

Recently, Watanabe proposed a new framework for testing the correctness and average case behavior of algorithms that purport to solve a given NP search problem efficiently on average. The idea is to randomly generate certified instances in a way that resembles the underlying distribution $\mu$. We discuss this approach and show that test instances can be generated for every NP search problem with non-adaptive queries to an NP oracle. Further, we introduce Las Vegas as well as Monte Carlo types of test instance generators and show that these generators can be used to find out whether an algorithm is correct and efficient on average under $\mu$. In fact, it is not hard to construct Monte Carlo generators for all RP search problems as well as Las Vegas generators for all ZPP search problems. On the other hand, we prove that Monte Carlo generators can only exist for problems in NP $\cap$ co-AM.

## 1 Introduction

The class NP plays a central role in computational complexity theory since it contains a large variety of problems that are of practical importance. Although, by definition, NP only contains decision problems, i.e., to decide whether a given instance has a solution, in many applications it is necessary to solve the corresponding search problem, i.e., to actually find a solution in case one exists. In general, the search version of an NP problem appears to be less tractable than its decision version. However, in some cases, the decision and

*Supported by the Deutsche Forschungsgemeinschaft, research grant Schö 302/4-2.

search versions are equivalent under polynomial-time Turing reductions, as, for example, in the case of NP-complete problems.

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 [Wan] 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].

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 certify the quality of an algorithm by giving a proof that it is correct and efficient. However, in many cases, these proofs may be long (if they can be found) and rather complicated. Thus, 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 outputs instances of the problem together with some solution. To be useful, these test instances should be generated according to some distribution which resembles the frequencies with which the instances occur in practice.

In this paper our objectice is to verify the global correctness of a given algorithm. This should be seen in contrast to the program checking approach in [BK95], where the algorithm has only to be verified on single instances. A framework for globally checking the correctness and average-case behavior of algorithms has been introduced by Watanabe [Wat94]. In his model, a probabilistic algorithm $G$ generates instances along with some solution, i.e., $G$ only produces *positive instances*. More precisely, $G$ on input $0^n$ outputs each positive instance of length $n$ with non-negligible probability (compared with the *conditional* distribution on the set of all positive instances of length $n$). For some distributions, this requires the generator to amplify the probability of the positive instances by an exponential factor. We exploit this fact to show that under such distributions test instance generators do not exist even for very simple search problems, unless RE = NE. On the other hand, non-adaptive queries to an NP-complete set turn out to be sufficient to generate test instances for any NP search problem. This result is tight since logarithmically many adaptive queries are shown to be useless.

To avoid the need for the probability amplification mentioned above we dispense with the condition that all generated test instances must be positive. In our model, a generator has to produce positive as well as negative instances. As in Watanabe's model we require the generator $G$ to output a witness along with every positively classified instance, i.e., $G$ might misclassify only positive instances. Moreover, we require that each instance $x$ is generated with a probability that is polynomially related to the probability $\mu(x)$ with which that instance occurs in practice. This condition implies that any algorithm is efficient on the generated instances if and only if it is efficient under the given distribution $\mu$.

We define two types of test instance generators that provided test instances of different quality: Whereas 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 find out efficiently and with high confidence whether a given algorithm $A$ for a distributional NP search problem is correct with respect to $\mu$. 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.

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) *generated "from outside"*. Thus there might exist distributional NP search problems for which it is easier to find a generator than to find an algorithm that solves it. For example, it is not hard to construct a Las Vegas generator for the integer factorization problem. On the other hand, no efficient on average algorithm is known (under the standard distribution) that on input $n$ outputs a factor of $n$ in case $n$ is composite, and outputs "prime" otherwise (in fact, this is the basis for the practical security of, e.g., 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.

Further, by an application of the universal hashing technique, we show that the domain of any search problem that admits a Monte Carlo generator (under the standard probability distribution) necessarily belongs to co-AM. As a consequence, Monte Carlo generators do not exist for search problems with an NP-complete domain, unless the polynomial time hierarchy collapses.

Finally, we define a reducibility between NP search problems which, 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 know a test instance generator for another NP search problem to which the given one reduces.

# 2 Preliminaries

In this paper we use the standard notations and definitions of computational complexity theory (see, e.g., [BDG95]). An introduction to the theory of computational average case complexity can be found in [Gur91, BCGL92, Wan].

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 set $A$ by $\|A\|$. We use the pairing function $\langle\cdot,\cdot\rangle : \Sigma^* \times \Sigma^* \to \Sigma^*$, defined as $\langle x, y\rangle = d(x)01y$ where $d(x_1 \dots x_n) = x_1x_1 \dots x_nx_n$. Note that $\langle x, y\rangle$ is computable and invertible in polynomial time, and that for all $x, y \in \Sigma^*$, $|\langle x, y\rangle| = 2|x| + |y| + 2$.

**Probability functions and distributions.** A probability function $\mu$ is a total function from $\Sigma^*$ to $[0, 1]$ such that $\sum_{x\in\Sigma^*} \mu(x) = 1$. For example, we refer to $\mu_{st}(x) = 1/(|x|(|x| + 1)2^{|x|})$ as the standard probability function. The distribution function of $\mu$ is $\mu^*(y) = \sum_{x:x\leq y} \mu(x)$ where $\leq$ denotes the lexicographic ordering on $\Sigma^*$.

Let $\mu, \nu : \Sigma^* \to [0, 1]$ be functions. For any set $X \subseteq \Sigma^*$ let $\mu(X) = \sum_{x\in X} \mu(x)$. $\mu$ is polynomially dominated by $\nu$ (in symbols: $\mu \leq^p_d \nu$) if for some polynomial $p$ and all $x$, $\mu(x) \leq p(|x|) \cdot \nu(x)$. $\mu$ and $\nu$ are called equivalent (in symbols: $\mu \equiv^p_d \nu$) if both $\mu \leq^p_d \nu$ and $\nu \leq^p_d \mu$. A function $f$ from $\Sigma^*$ to the non-negative integers is polynomial on $\mu$-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 $\mu$-average is closed under addition and multiplication. Furthermore, if $\mu$ is dominated by $\nu$, then every function which is polynomial on $\nu$-average is polynomial on $\mu$-average.

A function $\mu$ is called p-computable if the distribution $\mu^*$ can be approximated via a polynomial time algorithm in the sense of Ko and Friedman [KF82]. A function $\mu$ is called samplable [BCGL92] if there exists a probabilistic Turing machine $M$ that on input $\lambda$ outputs $x$ with probability $\mu(x)$. If, in addition, $M$ is polynomial-time bounded in the length of the output, then $\mu$ is called p-samplable.

**Distributional NP search problems.** An NP search problem is specified by a binary polynomial-time decidable relation $R$ (called witness relation) and a polynomial $q_R$ such that $R(x, w)$ implies $|w| = q_R(|x|)$. Any $w$ with $R(x, w)$ 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. W.l.o.g. we can assume that $\lambda$ is never a solution. A witness relation $R$ also specifies an NP decision problem, namely $D_R = \{x \mid \exists w : R(x, w)\}$. $D_R$ is also called the domain of $R$.

A distributional NP search problem is a pair $(R, \mu)$ consisting of a witness relation $R$ and a p-samplable probability function $\mu$. A (deterministic) algorithm $A$ efficiently solves $(R, \mu)$ if the running time of $A$ is polynomial on $\mu$-average and $A$ computes a solution for all inputs $x$ in the domain of $R$, whereas it outputs $\lambda$ on all other inputs.

# 3 Generating positive test instances

In this section we discuss Watanabe's approach to the test instance generation problem. The aim is to verify the quality (correctness and efficiency) of a given algorithm on the positive instances. Specifically, Watanabe requires the following properties of a test instance generator $G$ for a distributional NP search problem $(R, \mu)$ (actually, for the *promised* NP search problem restricted to the positive instances): $G$ is a probabilistic polynomial-time algorithm which on input $0^n$, either generates some instance $x$ of length $n$ together with a solution $w$ for $x$ or outputs $\perp$ ("not successful"). Moreover, $G$ has to output each positive instance with non-negligible probability compared with the conditional probability function $\rho_n$ on the positive instances of length $n$.

**Definition 3.1** [Wat94] *Let $(R, \mu)$ 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 $R$ under $\mu$ if there is a polynomial $q$ such that for all $n$,*

1. *on every path, $G(0^n)$ outputs either $\perp$ or a pair $\langle x, w\rangle \in R$ with $x \in \Sigma^n$, and*
2. *for all $x \in D^{=n}$, $G(0^n)$ outputs with probability at least $\rho_n(x)/q(n)$ a pair of the form $\langle x, w\rangle$, $w \in \Sigma^*$, where*

$$\rho_n(x) = \begin{cases} \mu(x)/\mu(D^{=n}), & \text{if } x \in D^{=n} \text{ and } \mu(D^{=n}) > 0, \\ 0, & \text{otherwise.} \end{cases}$$

*is the probability of $x$ under the condition that $|x| = n$ and $x$ belongs to $D$.*

As shown by Watanabe, generators of this type can be used to detect for a given input length $n$ that an algorithm makes an error with probability $\varepsilon$ (with respect to $\rho_n$). Furthermore, they can be used to find out that an algorithm is not efficient (with respect to $\rho_n$) [Wat94]. We note that since in Definition 3.1 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 $\rho$-average.

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 3.2** [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 that don't have test instance generators under the standard distribution, unless NE $\cap$ co-NE $\subseteq$ RE. This shows that also in the case that the domain is $\Sigma^*$ it might be hard to generate test instances since finding a solution might still be nontrivial.

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

*Proof.* Since RE = NE if and only if NP $\cap$ TALLY $\subseteq$ 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 $R_T(0^n, w)$.

Now consider the p-samplable distribution $\mu$ generated by the following probabilistic algorithm:

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

Then any test instance generator $G$ for $(R, \mu)$ has the property that for all $n$, $G(0^n)$ outputs $\langle 0^n, 1\rangle$ with probability $1/n^{O(1)}$ if $0^n \in T$, and with probability 0, otherwise. $\square$

Before we give an upper bound on the complexity of generating test instances we observe that $O(\log n)$ many queries to an arbitrary oracle do not help in generating test instances in Watanabe's model.

**Proposition 3.4** *If for a distributional* NP *search problem $(R, \mu)$, test instances can be generated with $O(\log n)$ queries to some oracle, then there exists a test instance generator for $(R, \mu)$.*

*Proof.* Let $M$ be a probabilistic oracle Turing machine that generates test instances with $O(\log n)$ queries to some oracle. Let $M'$ simulate $M$ where oracle queries are answered randomly. 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 3.1. $\square$

In the proof of the next theorem we make use of the universal hashing technique [CW79, Sip83, VV86]. Let $\mathcal{H}_{m,k}$ be the class of all linear hash functions from $\Sigma^m$ to $\Sigma^k$. We will use the following well-known fact that can easily be derived using that $\mathcal{H}_{m,k}$ is a universal class of hash functions (cf. [Pap94]).

**Lemma 3.5** *Let $S$ be a subset of $\Sigma^m$ 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 $h(x) = 0^k$.*

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

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

Consequently, we get for the probability that $x$ is the only preimage of $0^k$ in $S$ (i.e., $h$ *isolates $x$ inside $S$*),

$$\begin{aligned} &\text{Prob}\left[h(x) = 0^k \wedge \forall y \in S - \{x\} : h(y) \neq 0^k\right] \\ = \; &\text{Prob}\left[h(x) = 0^k\right] - \text{Prob}\left[h(x) = 0^k \wedge \exists y \in S - \{x\} : h(y) = 0^k\right] \\ > \; &2^{-k}(1 - s2^{-k}) \; \geq \; 2/9s. \end{aligned}$$

□

**Theorem 3.6** *For every distributional* NP *search problem $(R, \mu)$, test instances can be generated with parallel queries to* SAT.

*Proof.* Let $D$ be the domain of $R$ and let $q$ be a polynomial such that $|w| = q(|x|)$ for all $\langle x, w\rangle \in R$. Let $M$ be a probabilistic Turing machine witnessing that $\mu$ 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 $p(|x|)$. Let $M_r(\lambda)$ denote the output of $M$ (if any) when using sequence $r$ as random source. The test instance generator $G$ for $(R_D, \mu)$ 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 linear hash functions $h_1 \in \mathcal{H}_{p(n),k}$, $h_2 \in \mathcal{H}_{q(n),l}$. Then $G$ asks in parallel the queries $\langle 0^n, i, h_1, h_2\rangle$, $1 \leq i \leq l(n) = 2p(n) + q(n) + 2$ to get a sequence $s = s_1 \ldots s_{l(n)}$ of oracle answer bits. Then $G$ determines strings $r \in \Sigma^{p(n)}$, $w \in \Sigma^{q(n)}$, and $x = M_r(\lambda)$ such that $\langle r, w\rangle = s$. Finally, $G$ outputs the pair $\langle x, w\rangle$, in case $|x| = n$ and $R(x, w)$, and $\perp$ otherwise.

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 $\alpha_r$ 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 $w \in \Sigma^{q(n)}$ with the property that $h_2(w) = 0^l$ and $R(M_r(\lambda), w)$. By Lemma 3.5 it follows that

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

Hence, when using as oracle the set $A$ defined as

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

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

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

□

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

In this section we propose a slightly different model of generating test instances. As explained above, our aim is to avoid the need for amplifying the probability of the positive instances by a possibly exponential factor. We first define the notion of a witness generator $G$ for a distributional NP search problem $(R, \mu)$ which efficiently generates instances in accordance with the distribution $\mu$. 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 $\lambda$ (implying absolutely nothing on the membership status of $x$). Below we will define two types of instance generators by additionally imposing some bound on the probability that $G$ outputs the pair $\langle x, \lambda\rangle$ in case $x$ is positive.

**Definition 4.1** *Let $(R, \mu)$ be a distributional* NP *search problem. A probabilistic Turing machine $G$ is called a* witness generator *for $R$ under $\mu$ 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. *$\mu_G$ is equivalent to $\mu$, where $\mu_G(x)$ is the probability that $G$ generates instance $x$, i.e., $\mu_G(x) = \sum_w \mu_G(x, w)$, 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 finitley 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 (similar as in Watanabe's model).

Suppose that $G$ is a witness generator for a distributional NP search problem $(R, \mu)$ and let $A$ be an algorithm. Since the functions $\mu$ and $\mu_G$ are equivalent, the running time of $A$ is polynomial on $\mu$-average if and only if it is polynomial on $\mu_G$-average [Lev86].

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 4.2** *Let $G$ be a witness generator for a distributional* NP *search problem $(R, \mu)$ with domain $D$.*

1. *$G$ is called a* Monte Carlo generator *for $R$ under $\mu$ 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 $\mu$ if $G$ never outputs a pair $\langle x, \lambda \rangle$ with $x \in D$.*

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 $(R, \mu)$ is correct with respect to the underlying probability distribution $\mu$. We can w.l.o.g. 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 $D'$.

**Theorem 4.3** *Let $G$ be a Monte Carlo generator for a distributional* NP *search problem $(R, \mu)$ and let $D$ be the domain of $R$. Then there exists a probabilistic polynomial time-bounded transducer $T$ such that for every search algorithm $A$ and all $n, m, l > 0$ the following holds:*

> *If the error rate of $A$ on $D^{\leq l}$ is at least $1/m$, then with probability at least $1 - 2^{-n}$, $T(1^n, 1^m, 1^l)$ produces an instance $x \in D^{\leq l}$, together with a solution $w$ such that $A(x) = \lambda$.*

*Proof.* Let $G$ be a Monte Carlo generator for $(R, \mu)$ and let $p$ be a polynomial such that for all instances $x$,

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

Since for all but finitely many $x \in D$, $\mu_G(x, \lambda) \leq \mu_G(x)/2$, we can w.l.o.g. assume that for all instances $x \in D$,

$$\sum_{w \neq \lambda} \mu_G(x, w) \geq \mu_G(x)/2 \geq \mu(x)/2p(|x|).$$

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

It is interesting to note that under the following reasonable condition on $\mu$ it suffices to assume in the above theorem that $A$ has error rate at least $1/m$ on $D$ (instead of $D^{\leq l}$):

For some polynomial $q$, $\sum_{x:|x|>q(n)} \mu(x) \leq 1/n$ holds for all $n$.

In fact, if under this condition the error rate of $A$ on $D$ 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 $x \in D$ with $A(x) = \lambda$.

Next we consider distributional RP search problems $(R, \mu)$ (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 half of the strings of length $q_R(|x|)$ as solutions). As shown below, instance/witness pairs can be easily generated in this case by sampling a string $x$ according to $\mu$ and randomly generating a witness for $x$.

**Proposition 4.4** *For every distributional* RP *search problem* $(R, \mu)$ *there exists a Monte Carlo generator.*

*Proof.* Let $D$ be the domain of $R$ and let $q$ be a polynomial such that $|w| = q(|x|)$ for all $\langle x, w\rangle \in R$. Let $M$ be a probabilistic Turing machine witnessing that $\mu$ is p-samplable. Consider the following witness generator $G$ for $(R, \mu)$.

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

It is easy to verify that $G$ is a Monte Carlo test instance generator for $(R, \mu)$. □

Intuitively, a generator for a distributional NP search problem can benefit from the advantage that it may intertwine the processes of randomly generating an instance and 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 it. In fact, this phenomen is exploited in the design of many cryptographic protocols, e.g., for 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 (e.g., under the standard distribution) that on input $n$ 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.

However, it is unlikely that any NP search problem admits of a Monte Carlo generator under the standard probability distribution. As shown in the next theorem, 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 some textbook, e.g., [BDG90, KST93].

**Theorem 4.5** *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.* Let $G$ be a Monte Carlo generator for $(R, \mu_{st})$ and let $p$ be a polynomial bounding the running time of $G$. From property (1) of Definition 4.2 we can conclude that for some polynomial $r$,

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

From property (3) 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 $2^5 r(n)^2$. Then we get for all but finitely many $x$ (letting $n = |x|$),

$$\begin{aligned} x \in D \;\Rightarrow\; \mu_G(x,\lambda) \leq \frac{r(n)}{2^n q(n)} \;&\Rightarrow\; \|S_x\| \leq \frac{r(n)2^{p(n)}}{2^n q(n)} \;\Rightarrow\; \|S_x\| \leq \frac{2^{p(n)-n-5}}{r(n)}, \\ x \notin D \;\Rightarrow\; \mu_G(x,\lambda) \geq \frac{1}{r(n)2^n} \;&\Rightarrow\; \|S_x\| \geq \frac{2^{p(n)}}{r(n)2^n} \;\Rightarrow\; \|S_x\| \geq \frac{2^{p(n)-n}}{r(n)} \end{aligned}$$

Now let $k(n) = p(n) - n - \lfloor \log r(n) \rfloor - 2$ and consider for a fixed instance $x$ the random variable $Z$ giving the number of strings $r \in S_x$ with $h(r) = 0^{k(n)}$, where $h$ is chosen uniformly at random from $\mathcal{H}_{p(n),k(n)}$. Then the expectation of $Z$ is $E(Z) = 2^{-k(n)} \|S_x\|$ and the variance is $V(Z) = 2^{-k(n)}(1 - 2^{-k(n)}) \|S_x\|$ [VV86]. Thus, we have that

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

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

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 $\overline{D} \in \mathrm{AM}$. $\square$

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

**Corollary 4.6** *If there exists a Monte Carlo generator for some distributional* NP *search problem* $(R, \mu_{st})$ *whose domain is* NP*-complete, then* $\mathrm{PH} = \Sigma_2^P$.

We end this section by considering the Las Vegas type of generators that never output a wrongly classified instance. It is not hard to construct a Las Vegas generator for any RP search problem whose domain is in ZPP (ZPP search problem, for short). On the other hand, Las Vegas generators can only exist for NP search problems whose domain belongs to NP $\cap$ co-NP.

**Proposition 4.7**

1. *For every distributional* ZPP *search problem there exists a Las Vegas generator.*
2. *If there exists a Las Vegas generator for an* NP *search problem $R$ under the standard distribution $\mu_{st}$, then the domain of $R$ is contained in* co-NP.

*Proof.* 1. can be proved analogously to Proposition 4.4.

2. Let $R$ be a search problem with domain $D$ and let $G$ be a Las Vegas generator for $(R, \mu_{st})$. It suffices to prove $\overline{D} \in \mathrm{NP}$. Consider the following nondeterministic Turing machine $M$:

On input $x$, $M$ simulates $G$ and accepts if $G$ outputs the pair $\langle x, \lambda \rangle$.

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

**Corollary 4.8** *If there exists a Las Vegas generator for some distributional* NP *search problem* $(R, \mu_{st})$ *whose domain is* NP*-complete, then* NP = co-NP.

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 [Sch76, LT93], there exist Las Vegas generators for the corresponding search problems that ask adaptive (non-adaptive) queries to GI (GA, resp.).

# 5 Hard problems for test instance generation

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

**Definition 5.1** *Let* $(R, \mu)$ *and* $(S, \nu)$ *be distributional* NP *search problems. Let* $D_R$ $(D_S)$ *be the domain of* $R$ $(S$*, resp.). Then* $(R, \mu)$ *is* generator reducible *to* $(S, \nu)$ *if there exist functions* $f, g \in \mathrm{FP}$ *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. $\mu$ *is equivalent to* $\phi$*, where* $\phi$ *is the distribution induced by* $\nu$*,* $q$*, 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.

In the sequel, we assume w.l.o.g. 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 5.2** *If* $(R, \mu)$ *is generator reducible to* $(S, \nu)$ *and if there exists a Monte Carlo (Las Vegas) generator for* $(S, \nu)$*, then there exists a Monte Carlo (Las Vegas, resp.) generator for* $(R, \mu)$*.*

*Proof.* Assume that $(R, \mu)$ reduces to $(S, \nu)$ via the functions $f$, $g$ and the polynomials $p, q$. Let $G$ be a generator for $(S, \nu)$. 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 $f(y,r) \neq \bot$, then output the pair $\langle f(y,r), g(y,v,r)\rangle$, otherwise loop forever.

We claim that $G'$ is a witness generator for $(R,\mu)$ (of the same type as $G$). Let $C(x) = \{(y,r) \mid f(y,r) = x, |r| = q(|y|)\}$. It follows that if $G'$ outputs $\langle x, w\rangle$ then either $R(x,w)$ or $w = \lambda$. 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 $\phi$ is the distribution induced by $f$, $q$, and $\nu$. 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, $\mu \leq_d^p \mu_{G'}$, and since $\mu_{G'} \leq_d^p \mu$ can be derived in a similar way, $\mu_{G'} \equiv_d^p \mu$ follows. Furthermore, since $f$ is honest, the time needed by $G'$ to output a pair $\langle x, w\rangle$ is polynomially bounded in $|x|$.

Assume now that $G$ is a Monte Carlo generator for $(S,\nu)$. 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 $(R,\mu)$.

In the case that $G$ is a Las Vegas generator for $(S,\nu)$ it follows immediately from the properties of $f$ and $g$ that $G'$ is a Las Vegas generator for $(R,\mu)$. $\square$

Another important property of the generator reduction is transitivity.

**Proposition 5.3** *The generator reduction is transitive.*

*Proof.* Let $(R_i, \mu_i)$, $i = 1, 2, 3$, be distributional NP search problems. Let $D_i$ be the domain of $R_i$, $i = 1, 2, 3$. Assume that $(R_i, \mu_i)$ is generator reducible to $(R_{i+1}, \mu_{i+1})$ 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 5.1, 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. $\mu_1 \equiv_d^p \phi(x)$, where $\phi$ is the distribution induced by $f$, $q$, and $\mu_3$. 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 $\phi_i$ be the distribution induced by $f_i$, $q_i$, and $\mu_{i+1}$. Since $\mu_i \equiv_d^p \phi_i$ it follows that

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

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

Using the properties derived above it is easy to see that $(R_1, \mu_1)$ is generator reducible to $(R_3, \mu_3)$ 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. $\square$

Next we present two problems from group theory which 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].

$S_n$ denotes the symmetric group of $\{1, ..., n\}$, i.e. the set of all permutations over $\{1, ..., n\}$. For a set $A \subseteq S_n$, we denote by $\langle A \rangle$ the smallest subgroup of $S_n$ which 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 $a, b \in \{1, ..., n-1\}$ 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 $a, b \in \{1, ..., n+1\}$ 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 $R_{\mathrm{DCM}}$ ($R_{\mathrm{GF}}$, resp.) denote the search problem of DCM (GF, resp.).

**Proposition 5.4** *$(R_{\mathrm{DCM}}, \mu_{\mathrm{DCM}})$ and $(R_{\mathrm{GF}}, \mu_{\mathrm{GF}})$ are generator equivalent.*

*Proof. 1. $(R_{\mathrm{GF}}, \mu_{\mathrm{GF}})$ is generator reducible to $(R_{\mathrm{DCM}}, \mu_{\mathrm{DCM}})$.* Since this reduction makes no use of the random bits we omit the parameter $r$. Let $\langle 1^n, A, B, \pi, \psi \rangle$ be an instance of DCM. The reduction $(f, g)$ is defined as

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

It is easy to verify that this is 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 $\mu_{\mathrm{DCM}}$ coincides with $\mu_{\mathrm{GF}}$.

*2.* $(R_{\mathrm{DCM}}, \mu_{\mathrm{DCM}})$ *is generator reducible to* $(R_{\mathrm{GF}}, \mu_{\mathrm{GF}})$. Let $\langle 1^n, A, B, \psi \rangle$ be an instance of GF. 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 $\mu_{\mathrm{GF}}$ is equal to $\mu_{\mathrm{DCM}}$. □

Finally, we show the existence of a hard problem in the sense that every distributional NP search problem with a p-computable distribution is generator reducible to it.

***Bounded Halting Problem*** $(K, \mu_K)$.

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

*Question.* Does $M$ accept $x$ in within $k$ steps?

*Distribution.* Randomly and independently guess $M$, $x$ and $k$ under standard distribution, i.e., the density function $\mu_K$ 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:

$$R_K(\langle M, x, 1^k\rangle, v) \quad \Leftrightarrow \quad \begin{array}{l} v \text{ describes an accepting computation of } M \\ \text{on input } x \text{ of length} \le k. \end{array}$$

**Theorem 5.5** *Let $R$ be an* NP *search problem and let $\mu$ be p-computable. Then $(R, \mu)$ is generator reducible to $(R_K, \mu_K)$.*

*Proof.* Let $(R, \mu)$ be a distributional NP search problem such that $\mu$ is p-computable. Let $D$ be the domain of $R$.

It is known [Lev86, Gur91] that $D$ is many-one reducible to $K$ via a function $h \in \mathrm{FP}$ 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 $g \in \mathrm{FP}$ such that $R(x, g(h(x), v))$ holds for every $x, v$ with $R_K(h(x), v)$. Thus we have for every $y \in \mathit{range}(h)$,

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 $\sum_{y,f(y)=x} \mu_K(y)$.

Hence, $(R, \mu)$ generator reduces to $(R_K, \mu_K)$ via $\hat{f}$ and $\hat{g}$, where

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

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

**Remark.** We have shown Theorem 5.5 for the Bounded Halting Problem $(K, \mu_K)$ which is complete for the class DistNP consisting of all pairs $(L, \mu)$ such that $L$ is in NP and $\mu$ is p-computable [Lev86, Gur91]. In fact, the proof also works for other DistNP complete problems as, e.g., the Distributional Tiling Problem, the Distributional Post Correspondence Problem, the Distributional Word Problem for Thue Systems and for groups [Lev86, Gur91, WB95] or LR($k$) Testing of context-free grammars [Kar97].

On the other hand, it is open whether Theorem 5.5 can be extended to the class of p-samplable distributions. Note that there exists a similar open problem in average case theory, namely, whether every distributional NP problem with a p-samplable distribution reduces to the Bounded Halting Problem [BG95].

# References


[AFK89] M. Abadi, J. Feigenbaum, and J. Kilian. On hiding information from an oracle. *Journal of Computer and System Sciences*, 39:21–50, 1989.

[Bal89] J.L. Balcázar. Self-reducibility structures and solutions of NP problems. In *Revista Matematica*, volume 2, pages 175–184. Universidad Complutense de 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(2):193–219, 1992.

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

[BDG90] J.L. Balcázar, J. Díaz, and J. Gabarró. *Structural Complexity II.* Springer, 1990.

[BDG95] J.L. Balcázar, J. Díaz, and J. Gabarró. *Structural Complexity I.* Springer, 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 of Computing*, 24(1):3–29, 1995.

[BHZ87] R.B. 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 that check their work. *Journal of the Association for Computing Machinery*, 42(1):269–291, 1995.

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

[Boo74] R.V. Book. Tally languages and complexity classes. *Information and Control*, 26:281–287, 1974.

[Cam95] P.J. Cameron. Permutation groups. In R. Graham, Grötschel, and L. Lovász, editors, *Handbook Of Combinatorics*, chapter 12, pages 611–645. Elsevier, 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 complexity. *Journal of Computer and System Sciences*, 42(3):346–398, 1991.

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

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

[Jer82] M. Jerrum. A compact representation for permutation groups. In *Proc. 23rd Symposium on Foundations of Computer Science*, pages 126–133. IEEE, 1982.

[Kar97] C. Karg. LR($k$) Testing is average-case complete. In *Proc. 12th Conference on Computational Complexity Theory.* IEEE, 1997. to appear.

[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: It's Structural Complexity.* Birkhäuser, 1993.

[Lev86] L.A. Levin. Average case complete problems. *SIAM Journal of 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 245–271. Cambridge University Press, 1993.

[Pap94] C. Papadimitriou. *Computational Complexity.* Addison–Wesley, 1994.

[Sch76] C.P. Schnorr. Optimal algorithms for self-transformable problems. In *Proc. 3th International Colloquium on Automata, Languages and Programming.* Lecture Notes in Computer Science, Springer, 1976.

[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 *Proc. 15th ACM Symposium on the Theory of Computing*, pages 330–335, 1983.

[TW87] W. Tompa and H. Woll. Random self-reducibility and zero-knowledge interactive proofs of possession of information. In *Proc. 28st Symposium on Foundations of Computer Science*, pages 472–482, 1987.

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

[Wan] J. Wang. Average-case computational complexity theory. In A. Selman and L. Hemaspaandra, editors, *Complexity Theory Retrospective II.* Springer. (to appear).

[Wat94] O. Watanabe. Test instance generation for promised NP search problems. In *Proc. 9th Conference on Structure in Complexity Theory*, pages 205–216. IEEE, 1994.

[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.