On Generating Solved Instances of Computational Problems (Extended Abstract)

M. Abadi* E. Allender† A. Broder* J. Feigenbaum‡ L. Hemachandra§

Abstract: We consider the efficient generation of solved instances of computational problems. In particular, we consider *invulnerable generators*. Let S be a subset of $\{0,1\}^*$ and M be a Turing Machine that accepts S; an accepting computation w of M on input x is called a "witness" that $x \in S$. Informally, a program is an *α-invulnerable generator* if, on input 1^n, it produces instance-witness pairs $\langle x, w\rangle$, with $|x| = n$, according to a distribution under which any polynomial-time adversary who is given x fails to find a witness that $x \in S$, with probability at least α, for infinitely many lengths n.

The question of which sets have invulnerable generators is intrinsically appealing theoretically, and the results can be applied to the generation of test data for heuristic algorithms and to the theory of zero-knowledge proof systems. The existence of invulnerable generators is closely related to the existence of cryptographically secure one-way functions. We prove three theorems about invulnerability. The first addresses the question of which sets in NP have invulnerable generators, if indeed any NP sets do. The second addresses the question of how invulnerable these generators are.

Theorem (Completeness): If any set in NP has an α-invulnerable generator, then SAT has one.

Theorem (Amplification): If $S \in$ NP has a β-invulnerable generator, for some constant $\beta \in (0,1)$, then S has an α-invulnerable generator, for every constant $\alpha \in (0,1)$.

Our third theorem on invulnerability shows that one cannot, using techniques that relativize, resolve the question of whether the assumption that P $\neq$ NP alone suffices to prove the existence of invulnerable generators. Clearly there are relativized worlds in which invulnerable generators exist; in all of these worlds, P $\neq$ NP. The more subtle question, which we resolve in our third theorem, is whether there are also relativized worlds in which P $\neq$ NP and invulnerable generators do not exist.

Theorem (Relativization): There is an oracle relative to which P $\neq$ NP but there are no invulnerable generators.

More generally, we ask which sets can be generated according to which distributions. In particular, the simplest requirement that can be imposed on a generator is that it produce some $x \in S$ of length n, if at least one such x exists; we call such a generator an *exemplifier*. It seems at first that the sets in P or even NP that one encounters "naturally" are all exemplifiable. Nevertheless, we show:

Theorem: There are sets in P that are not exemplifiable, unless E = NE. More generally, there are sets in Π_i^{P} that are not exemplifiable, unless E = $\Sigma_{i+1}^{\mathrm{E}}$.

*DEC Systems Research Center, Palo Alto, CA 94301.
†Rutgers University, New Brunswick, NJ 08903.
‡AT&T Bell Laboratories, Murray Hill, NJ 07974.
§Columbia University, New York, NY 10027.

1 Introduction

We consider the efficient generation of solved instances of computational problems. For example, if $S = \{x\colon \exists w.p(x, w)\}$ is a set in NP, we may wish to generate instance-witness pairs $\langle x, w\rangle$ according to a specified distribution. The relationship between the complexity of generating pairs $\langle x, w\rangle$ and the complexity of finding w given x is intrinsically interesting theoretically, and it is also important to the testing of heuristic algorithms for hard problems and the proposed applications of zero-knowledge proof systems.

One of the questions we ask is whether it is possible to generate what we call an *invulnerable* distribution of instance-witness pairs. For example, is it possible to generate pairs $\langle f, a\rangle$, where f is a boolean formula and a is a satisfying assignment, give the secret a to one user A, publish the formula f, and remain reasonably confident that a polynomial-time adversary would be unable to find a satisfying assignment a' for f and thus to impersonate A? Feige, Fiat, and Shamir proposed this use of "zero-knowledge proofs of identity" as a security mechanism; the specific scheme they suggest is based on the Quadratic Residuosity Problem (QRP, [7]). Zero-knowledge proofs of identity may still be useful even if the QRP turns out to be easier than is widely assumed; furthermore, even if the QRP is hard, it may be possible to base a scheme on another problem and achieve more security. Thus, it is important to have a complexity-theoretic framework in which to consider whether a scheme for generating instance-witness pairs produces a secure distribution.

When Goldwasser, Micali, and Rackoff first introduced zero-knowledge proof systems, they postulated an all-powerful prover ([11]). Since then, they and others (e.g., [4], [6]) have considered a model in which prover and verifier have the same computational resources, and the prover's only advantage is that he happens to know the witness w for a particular instance x of the hard problem at hand, perhaps because he constructed x and w simultaneously. This model, together with the proof that all sets in NP have zero-knowledge proof systems ([5], [12]), forms the basis for the "compilation" of multi-party protocols into "validated" protocols ([8], [12]). Thus, it is important to realize that the model is meaningful only if there is a way for an efficient program to generate harder instances than the verifier can solve.

Many NP-Complete sets have obvious, simple generation schemes. For example, Hamiltonian graphs on n vertices can be generated by choosing a random circuit and then adding each other possible edge independently with probability $1/2$. The probability of generating a particular graph is proportional to the number of Hamiltonian circuits it has. However, the following examples show that some natural methods of generating solved instances are not secure. The first method succumbs to a very simple algorithm; the second can be cracked by a sophisticated technique.

Example: 3SAT. A 3SAT instance is a set of variables $U = \{u_1, u_2, \ldots, u_n\}$ and a set of clauses $C = \{c_1, c_2, \ldots, c_m\}$, where each clause consists of three literals. The question is whether there exists a truth assignment that satisfies C. (See [9] for definitions.)

A "natural" way to generate solved 3SAT instances is as follows. Choose a truth assignment t uniformly from the 2^n possibilities. For each i between 1 and m, choose three distinct variables uniformly at random; of the eight sets of literals that correspond to these variables, seven are true under t. Choose clause c_i from those seven, uniformly at random. This scheme produces each set C of m clauses satisfied by t with equal probability.

A polynomial-time adversary can reconstruct t with high probability, if the number of clauses m is large enough. The basic observation is that if $t(u_i) = \text{TRUE}$ then

$$\frac{\Pr(u_i \in c_j)}{\Pr(\overline{u_i} \in c_j)} = \frac{4}{3}, \qquad \text{for every } i \text{ and } j.$$

Therefore, if $m \geq kn \ln n$ for a suitable constant k, then with probability $1 - o(1)$ for every i simultaneously, the literal u_i appears in C more often than the literal $\overline{u}_i$ if and only if $t(u_i) = \text{TRUE}$.

One can try to improve this generation scheme by choosing the literals in each clause so that at least one is FALSE and at least one is TRUE. Then the expected number of occurences of $\overline{u_i}$ is equal to the expected number of occurences of u_i, for all i. However, the improved scheme can be cracked easily if $m \geq kn^2 \ln n$ by observing statistics about pairs of variables. ∎

Example: Subset Sum. A Subset Sum instance consists of a finite set $A = \{a_1, a_2, \ldots, a_n\}$ of positive integers and a positive integer M. The question is whether there exists a set $A' \subset A$ that has sum equal to M. The difficulty of the Subset Sum problem is the justification of knapsack-type public key cryptosystems.

One can generate solved Subset Sum instances as follows. Choose a vector $e = (e_1, \ldots, e_n)$ of zeroes and ones, uniformly at random. Fix a positive integer B. Choose each $a_i \in A$ uniformly at random from $\{1, 2, \ldots, B\}$. Let $M = \sum_{1 \leq i \leq n} a_i e_i$.

This generation scheme can be cracked with an algorithm due to Lagarias and Odlyzko ([19]). If B is sufficiently large, then every instance is almost certainly solvable by their ingenious application of the LLL basis-reduction algorithm. ∎

In Section 3 below, we define precisely what it means for a generation scheme to be invulnerable. We then prove a Completeness Theorem that states that, if any set in NP has an invulnerable generator, SAT has one. In particular, under the Quadratic Residuosity Assumption, the Discrete Logarithm Assumption, or the Factoring Assumption, one can generate a hard distribution of SAT.[1] This is not surprising. What is more interesting is that, even if all of these assumptions turn out to be false, one can still generate a hard distribution of SAT, provided one can generate a hard distribution of anything in NP. Our construction of an invulnerable generator for SAT incorporates whatever invulnerability is present in any possible generator for an NP set and does not assume it knows where the invulnerability comes from (as it would be assuming if it built hard instances by multiplying distinct primes, as in [7], etc.). Section 3 also contains an Amplification Theorem, which shows how to enhance the invulnerability of any generable distribution, and a Relativization Theorem — the existence of invulnerable generators clearly implies that P $\neq$ NP, but the converse cannot be proven by techniques that relativize.

In Section 4, we impose the simplest possible requirement on a generator, i.e., merely that it produce, on input 1^n, some instance of size n if one exists. Although it seems that all of the usual sets in NP can be exemplified easily, we show that there are sets in P that cannot be exemplified, unless E = NE. The proof draws on the connection between exemplification and the complexity of recognizing tally sets (subsets of $\{1\}^*$) and sparse sets; we generalize it to obtain the structural consequences of assuming that all sets at any particular level in the polynomial hierarchy can be exemplified.

In Section 5, we discuss briefly the general question of which sets can be generated according to which distributions, consider several related works, and propose directions for future research. Section 2 contains terminology and notation that is used extensively in the rest of the paper. We have deferred full proofs until the final version of the paper in order to save space; whenever possible, we give sketches that convey some of the essential points.

[1] Various forms of these assumptions are ubiquitous in the cryptographic literature (see, e.g., [1], [3], [10], [23]), and we don't need precise statements of them for this informal discussion. For our purposes, it suffices to note that it is possible to generate instances of these number-theoretic problems in randomized polynomial time and that it is widely assumed that, for each of the three problems, for any constant fraction, each polynomial-time algorithm fails to solve that constant fraction of the instances of length n, for all sufficiently large n.

2 Terminology, Notation, and Conventions

We call a program that flips coins and terminates in worst-case polynomial time on all inputs a *randomized polynomial-time* program. Let $\{M_i\}$ denote a standard enumeration of the randomized polynomial-time programs. Let $\{N_j\}$ denote a standard enumeration of polynomial-time nondeterministic programs; thus, each NP set is recognized by at least one program in our enumeration. We use $L(N_j)$ to denote the set (or language) recognized by N_j.

Let N be a nondeterministic polynomial-time program and S be $L(N)$. We call each accepting path of N on input x a *witness* that the *instance* x is in S. We assume without loss of generality that, for any fixed program N, the length n of an instance determines the length m of a witness and that the function $n \mapsto n + m$ is one-to-one. We let ω_x^N denote the set of witnesses that $x \in S$.

We use PF to denote the class of polynomial-time computable functions; a function $f \in$ PF need not have range $\{0,1\}$, and thus PF is a proper superset of the functions that compute membership of strings in sets in P. We use E to denote the union, over all positive constants c, of the sets recognizable in DTIME(2^{cn}). We use NE to denote the analogous class for NDTIME and RE to denote the class of sets in NE that are recognized by nondeterministic machines with the property that, if x is accepted by one computation path, then it is accepted by at least half of all possible computation paths. Thus, NE is to RE as NP is to R. (Note that we are not using RE to denote the class of recursively enumerable sets.)

We let S_n denote the elements of S that have length n. The symbol Λ denotes the default output of a program; it may be used to indicate that the desired output does not exist or that the program failed to find it. All of the generation programs that we consider take as input the length n, written in unary, run in polynomial time, and produce elements of S_n; thus we have, by definition, restricted attention to efficient generation.

3 Invulnerable Generators

In this section, we provide a complexity-theoretic framework in which to consider the generation of hard, solved instances. We define precisely what it means for a distribution of instance-witness pairs to be "secure against polynomial-time adversaries." Our first theorem addresses the question of which sets in NP have invulnerable generators, if indeed any such sets have them. Theorem 2 addresses the question of exactly how invulnerable these generators are. Finally, Theorem 3 addresses the question of what complexity-theoretic assumptions are needed to prove the existence of invulnerable generators.

Definition: The $(i,j)^{\text{th}}$ *generation scheme*, which we denote $G_{i,j}$, is a program that, on input 1^n, first simulates M_i on input 1^n and obtains an output string y. If y is of the form $\langle x, w\rangle$, where $|x| = n$ and w is an accepting computation of N_j on input x, then $G_{i,j}$ outputs $\langle x, w\rangle$; otherwise, it outputs Λ.

Consider the following game, played between a generation scheme $G_{i,j}$ and an adversary f in PF. The input to the game is a string 1^n; the first move is a run of $G_{i,j}$ on input 1^n. If $G_{i,j}$ outputs a pair $\langle x, w\rangle$, then the second move is for f to output $f(x)$; if $G_{i,j}$ outputs Λ, then the game ends after the first move. The function f wins the game if the generator outputs Λ, or if the generator outputs $\langle x, w\rangle$ and $f(x)$ is an accepting computation w' of N_j on input x; otherwise, the generator wins. Note that w' need not equal w; for example, in the identification scheme of Section 1, the adversary f can compromise the security of user A if he computes any satisfying assignment for A's public formula — he need not discover the private assignment that A was given during key-distribution.

Definition: A generation scheme is *α-invulnerable*, where α is a constant in $[0,1]$, if, for all $f \in$ PF, there are infinitely many lengths n for which the probability that f wins on input 1^n is at most $1-\alpha$. This probability is computed over runs of the game on input 1^n.

Definition: A set S in NP is *α-invulnerable*, where α is a constant in $[0,1]$, if there is a pair (i,j) for which $G_{i,j}$ is *α-invulnerable* and $S = L(N_j)$.

Notice that invulnerable generators are closely related to cryptographically secure one-way functions. Let g be a length-preserving function in PF, and assume that any polynomial-time program fails to invert at least a constant fraction of g's outputs, on infinitely many lengths (where "invert" means "find some element of the preimage"). Then the image of g has an invulnerable generation scheme: on input 1^n, generate a random w of length n and let x equal $g(w)$. Similarly, an invulnerable generation scheme $G_{i,j}$ gives rise to a cryptographically secure one-way function. The program M_i can be viewed as a mapping from coin-toss sequences to pairs $\langle x, w\rangle$. Let g be the function that takes a coin-toss sequence to the first component x of the pair output by M_i. Then g must be hard for any polynomial-time adversary to invert on infinitely many lengths; if it weren't the adversary could discover a coin-toss sequence that gives rise to $\langle x, w\rangle$, and the scheme $G_{i,j}$ would be vulnerable. The same remarks apply if we require in both cases that adversaries fail on all sufficiently high lengths instead of just infinitely many lengths.

We do not claim that a generation scheme that is invulnerable according to our definition is necessarily useful in practice. For example, the key-distributor in [7] would certainly like to know more than that there *exist* infinitely many lengths on which a particular polynomially bounded adversary can be thwarted with high probability; he would also like to know that such lengths are of practical size and to have a procedure for finding them. Our definition of invulnerability does, however, provide a good place to start a complexity-theoretic investigation.

Theorem 1 (Completeness): If any NP set is α-invulnerable, for some positive α, then SAT is also α-invulnerable.

Proof (sketch): The full proof proceeds in three stages. First, we construct a "universal generation scheme" G_U that simulates all possible generation schemes, capturing a constant fraction of whatever invulnerability is present in any of them. Next we construct a generator for SAT that applies Cook's reduction to the set S_U generated by G_U in a way that preserves invulnerability. Finally, we show that the lost fraction of invulnerability can be recaptured.

For the universal generator G_U, we need one program M_U, whose running time is bounded by a specific polynomial, to simulate infinitely many programs, whose individual running times may be arbitrarily high degree polynomials. We overcome that obstacle with the following lemma; it guarantees that we need only consider generators $\{G_k\}$ in which the program M runs in quadratic time.

Lemma: If $G_{i,j}$ is α-invulnerable, then there is an α-invulnerable generation scheme $G_{i',j'}$ in which $M_{i'}$ runs in quadratic time.

We cannot use a "generic reduction" such as the one used in Cook's proof of the NP-Completeness of SAT in order to construct a universal generator. Such a reduction would not necessarily be length-consistent (i.e., map instances of the same length to instances of the same length). Furthermore, even if our generic reduction mapped instances of length n to instances of length n^k, it may not preserve invulnerability: informally, if the "hard instances" output by a particular generator G_m represent a constant fraction α of the probability mass at length n, their images do not necessarily represent a constant fraction of the probability mass at length n^k, simply because there are so many more instances of length n^k.

We use a nonstandard pairing function to overcome this difficulty. It partitions the positive integers into "columns" as follows: column m, consists of all integers of the form $2^{m-1} + k \cdot 2^m$, where $k \geq 0$. Each input length n falls into exactly one column — the one whose index is one more than that of the least significant "1"-bit in the binary representation of n. On input 1^n, G_U first finds m, the index of the column containing n, then chooses an integer l uniformly from the interval $[n - 2^m, n)$. Next, G_U simulates G_m on input 1^l to obtain $\langle x, w \rangle$, pads x, and outputs $\langle x10^{n-l-1}, w \rangle$.

Lemma: If G_m is α-invulnerable, then G_U is $(\alpha/2^m)$-invulnerable.

Informally, to show that, for all f in PF, there are infinitely many lengths n on which f fails to "crack" the output of G_U with probability at least $\alpha/2^m$, we show that any such f corresponds to a function f' that fails to crack the output of G_m on infinitely many lengths n' with probability at least α. The loss of a factor of 2^m occurs because the "hard length" n' (for f' and G_m) corresponds to the hard length n (for f and G_U) such that $n' \in [n - 2^m, n)$; thus G_U only chooses to simulate G_m on input $1^{n'}$ with probability 2^{-m}. (Note that $\alpha/2^m$ really *is* a constant, because m is just the (fixed) index of a generator in our enumeration $\{G_k\}$.)

To construct an $(\alpha/2^m)$-invulnerable generator G_{SAT} for SAT, we use the fact that the program M_U in generator G_U runs in cubic time. We modify Cook's reduction so that, when applied to NP machines that run in cubic time, it takes instances of length k and produces instances of length exactly k^4. This modified Cook's reduction r also induces a mapping from witnesses of membership in S_U to satisfying assignments of elements of SAT. Thus G_{SAT} behaves as follows on input 1^n. If n is not a perfect fourth power, it outputs Λ. Otherwise, it simulates G_U on input 1^k, where $k^4 = n$, obtains a pair $\langle x, w \rangle$, and outputs $r(\langle x, w \rangle)$. We prove in the full paper that G_{SAT} is at least as invulnerable as G_U.

Theorem 2, below, guarantees that, if SAT has an $(\alpha/2^m)$-invulnerable generator, then it also has an α-invulnerable generator. ∎

Corollary: Under the Quadratic Residuosity Assumption, the Discrete Logarithm Assumption, or the Factoring Assumption, there is an α-invulnerable generator for SAT, for some $\alpha \in (0, 1)$.

Remark 1: For cryptographic purposes, one would really want more than that "there exists an infinite set of hard lengths" for cryptographic purposes. Note that the proof of Theorem 1 gives some hope, because, if some G_m defeats an adversary on $t(n)$ lengths between 1 and n, then G_U defeats the corresponding adversary on $\Omega(t(n))$ lengths between 1 and n. (This would not have been true had we used a standard pairing function that stretches both of its arguments quadratically.)

Theorem 2 (Amplification): If an NP set S is β-invulnerable, for some positive β, then S is also α-invulnerable, for all $\alpha \in (0, 1)$.

Proof (sketch): It suffices to show that α-invulnerability implies $2\alpha/(1+\alpha)$-invulnerability, because the limit of the sequence defined by $\alpha_0 = \alpha$, $\alpha_i = 2\alpha_{i-1}/(1 + \alpha_{i-1})$ is 1.

Intuitively, we will show how to increase the level of invulnerability in the most natural way: generate instances, try to crack them, and throw out the cracked ones. Suppose that $G_{i,j}$ is α-invulnerable and that $S = L(N_j)$. If $G_{i,j}$ is $(\alpha + (1 - \alpha)/2)$-invulnerable, then we are done, because $(\alpha + (1 - \alpha)/2) > (2\alpha/(1 + \alpha))$; so suppose that it isn't. Then, by definition, there is some $f \in$ PF that wins against $G_{i,j}$ on all but finitely many inputs 1^n with probability greater than $(1 - \alpha)/2$.

Consider the generator $G_{i',j}$ that works as follows on input 1^n: first it runs M_i on input 1^n, just as $G_{i,j}$ does. If M_i outputs $\langle x, w \rangle$, then $G_{i',j}$ computes $f(x)$ and checks whether it is an accepting computation of N_j on input x. If it is, then $G_{i',j}$ runs M_i again on input 1^n; otherwise, $G_{i',j}$ outputs $\langle x, w \rangle$. If f wins a sufficiently large number of successive runs of the game, then $G_{i',j}$ outputs Λ.

Clearly, $G_{i',j}$ generates the same set as $G_{i,j}$, namely $L(N_j)$. In the full paper, we show that $G_{i',j}$ is $(2\alpha/(1+\alpha))$-invulnerable and derive a good enough bound on the number of runs of the game between f and $G_{i,j}$ that $G_{i',j}$ has to simulate. ∎

Remark 2: For simplicity, we have modeled the adversary as a deterministic polynomial-time function. Clearly, in practice one would have to guard against randomized polynomial-time adversaries. Theorems 1 and 2 as stated hold even if we quantify over all randomized polynomial-time functions in the definition of invulnerability. We give details in the full paper.

Is it possible, in Theorem 1, to weaken the hypothesis that at least one set in NP is α-invulnerable? There are clearly oracles relative to which invulnerable generators exist. Indeed a random oracle will do ([21]). In all of these relativized worlds, P $\neq$ NP. Is the assumption that P $\neq$ NP sufficient to prove that invulnerable generators exist? Our next theorem shows that such a proof would not relativize.

Theorem 3 (Relativization): There is an oracle B such that $\mathrm{P}^{\mathrm{B}} \neq \mathrm{NP}^{\mathrm{B}}$, and invulnerable generators do not exist relative to B.

Proof (sketch): Let B = QBF $\oplus$ K, where $\oplus$ is disjoint union, and K is an extremely sparse set of strings of maximum Kolmogorov complexity. Specifically, K contains one string of each length n_i, where the sequence $n_1, n_2, \ldots$ is defined by: $n_1 = 2$, n_i is triply exponential in n_{i-1}, for $i > 1$; if $x \in K$ and $|x| = n$, then x has Kolmogorov complexity n. The inclusion of QBF gives machines with access to B the full power of PSPACE.

It is straightforward to prove that $\mathrm{P}^{\mathrm{B}} \neq \mathrm{NP}^{\mathrm{B}}$ using the techniques in [14].

To show that no invulnerable generators exist relative to B, let $G_{i,j}$ be a generation scheme that has access to the oracle, and assume that it is α-invulnerable, for some constant α in $(0,1)$. We derive a contradiction by producing an adversary f in PF^{B} that can crack a higher fraction than $1-\alpha$ of all of the instances of any length. Here is an informal description of f and why it works:

The generator $G_{i,j}$ involves a randomized polynomial-time program M_i and a nondeterministic polynomial-time program N_j, both of which can query B at any step. Let n^{k_1} and n^{k_2} be bounds on the running times of M_i and N_j, and let k be an integer greater than $\max(k_1, k_2)$. When trying to crack an instance x of length n, f first constructs the set K' consisting of all elements of K that have length less than $\log(n^{ck})$, where c is a suitably chosen constant. Because K is so sparse, there is at most one string r in $K \setminus K'$ about which $G_{i,j}$ may have queried B in generating an x of length n.

Assume that N is the integer closest to n for which there is a string in K of length N. The difficult case is when $\log(N^{ck}) \leq n \leq 2^{N/ck}$; otherwise, f can construct a witness that $x \in L(N_j)$ by using queries to $\mathrm{B}' = \mathrm{QBF} \oplus K'$. So assume, for example, that $n = 2^{N/ck}$.

The cracker f first uses B′ to determine whether there is a coin-toss sequence s that would cause $G_{i,j}$ to output x on input 1^n if $G_{i,j}$ were using B′. If such an s exists, then f can use PSPACE to construct one and in turn to construct a witness; this construction may or may not involve the discovery of the random string $r \in K \setminus K'$. If such an s does not exist, then f is not able to construct a witness. We show, however, that the only time there is no such s (and hence the only time f fails) is when $G_{i,j}$ actually queried B about the membership of r in K. We complete the proof with a counting argument that shows that, if this happens with any constant probability α, then r cannot have maximum Kolmogorov complexity. ∎

4 Exemplifiers

In this section, we study generators with the simplest possible property, i.e., those that produce some instance of size n if one exists. We call these generation programs *exemplifiers* and distinguish three kinds:

Definition: The set S is *weakly exemplifiable* if there is a deterministic polynomial-time program that, on input 1^n, returns an element of S_n, if $S_n \neq \emptyset$, and returns an arbitrary answer if $S_n = \emptyset$.

Definition: The set S is *strongly exemplifiable* if there is a deterministic polynomial-time program that, on input 1^n, returns an element of S_n, if $S_n \neq \emptyset$, and returns Λ if $S_n = \emptyset$.

Definition: The set S is *randomly exemplifiable* if there is a randomized polynomial-time program that, on input 1^n, behaves as follows:

Case 1: If $S_n \neq \emptyset$, then, with probability at least $1/2$, the program returns an element of S_n. If it does not return an element of S_n, then it returns Λ.

Case 2: If $S_n = \emptyset$, then the program returns Λ.

We denote by WEAK–EX, STRONG–EX, and RANDOM–EX the classes of sets that are weakly, strongly, and randomly exemplifiable, respectively. All of the sets one encounters "naturally" in NP seem to be exemplifiable; for example, it is easy to produce one Hamiltonian graph of each size or one graph with a perfect matching of each size or one satisfiable formula of each size. Thus, one basic question is whether or not every set in a particular complexity class is exemplifiable. A straightforward diagonalization argument shows:

Theorem 4: If f is superpolynomial and fully time-constructible, then there is a set in DTIME(f) that is not in EX, where EX is any of WEAK–EX, STRONG–EX, and RANDOM–EX.

For sets in the polynomial hierarchy, the situation is less clear. We show the following:

Theorem 5: If $\Pi_i^{\mathrm{P}} \subseteq$ WEAK–EX, then $\mathrm{E} = \Sigma_{i+1}^{\mathrm{E}}$. If $\Pi_i^{\mathrm{P}} \subseteq$ RANDOM–EX, then $\mathrm{RE} = \Sigma_{i+1}^{\mathrm{E}}$. In particular, if all sets in P are weakly exemplifiable, then E = NE, and, if all sets in P are randomly exemplifiable, then RE = NE.

Proof (sketch): The following connection between tally sets and the exponential hierarchy is drawn in [2] and [15].

Lemma (Book, Hartmanis et al.): Every tally set in Σ_i^{P} is in P if and only if $\mathrm{E} = \Sigma_i^{\mathrm{E}}$, for $i \geq 0$. In particular, every tally set in NP is in P if and only if E = NE.

We extend their proofs to show the following.

Lemma: Every tally set in Σ_i^{P} is in R if and only if $\mathrm{RE} = \Sigma_i^{\mathrm{E}}$, for $i \geq 1$. In particular, every tally set in NP is in R if and only if RE = NE.

If T is a tally set in $\Sigma_{i+1}^{\mathrm{P}}$, then it has the form $\{1^n\colon \exists w_{i+1} \forall w_i \cdots Q w_1 . p(1^n, w_1, \ldots, w_i, w_{i+1})\}$, where Q is either $\exists$ or $\forall$, depending upon whether $i+1$ is odd or even. By assumption, $S = \{\langle 1^n, w_{i+1}\rangle\colon \forall w_i \cdots Q w_1 . p(1^n, w_1, \ldots, w_i, w_{i+1})\}$ is exemplifiable (either weakly or randomly), because it is in Π_i^{P}. We use an exemplifier for S to construct a membership test that shows that T is either in P or in R, depending upon the type of exemplifier. The conclusion of Theorem 5 follows from the lemma. ∎

The fact that E = NE if all sets in P are weakly exemplifiable was proven independently, in another context, by Rudich and Impagliazzo ([21]).

Theorem 1 shows that SAT is complete with respect to invulnerable generators, i.e., that, if any set in NP is invulnerable, then SAT is. The following theorem concerns completeness with respect

to exemplification. Let $\{N_i\}$ be an enumeration of the nondeterministic polynomial-time programs, and assume without loss of generality that N_i runs in time exactly $n^i + i$. Let $S_0 = \{\langle 1^i, x, pad\rangle$: N_i accepts on input x within $|pad|$ steps$\}$. The pairing function $\langle ., ., .\rangle$ has the property that $|\langle a, b, c\rangle| = |\langle A, B, C\rangle|$ if and only if $|a| = |A|$, $|b| = |B|$, and $|c| = |C|$. Then S_0 is complete for exemplification in the following sense.

Theorem 6: S_0 is in EX if and only if all sets in NP are in EX, where EX is any of WEAK–EX, STRONG–EX, or RANDOM–EX.

5 Discussion, Related Work, and Open Problems

Let S be an NP set and fix a specific machine N that accepts S. Recall that ω_x^N is the set of accepting paths of N on input x. We say that S is *canonically generable* if there is a randomized polynomial-time program that, on input 1^n, generates pairs $\langle x, w\rangle$, where $|x| = n$, such that the probability accorded x is proportional to $|\omega_x^N|$. The straightforward generation schemes given in Section 1 for Hamiltonian graphs, 3SAT formulas, and Subset Sum instances are all canonical, with respect to the usual types of witnesses for these sets.

We call these generators canonical mainly because, in a sense, all generators for NP sets are canonical. If $G_{i,j}$ is a generation scheme for S, then a coin-toss sequence that causes M_i to output $\langle x, w\rangle$ is a witness that $x \in S$, and the probability accorded a particularly x is clearly proportional to the number of coin-toss sequences that cause it to be output.

The straightforward canonical generation scheme for Hamiltonian graphs has this general form: generate w uniformly and then pick x uniformly from the set of all instances such that w is a witness that x is in S. In fact, many sets in NP (e.g., SAT, graphs with perfect matchings, graphs with cliques of size $|V(G)|/2$) have canonical generation schemes of this form with respect to the usual types of witnesses. This leads naturally to the question of whether every set in NP has such a canonical generation scheme with respect to every type of witness. The answer to this question is no, unless the construction problem for NP sets can always be solved in polynomial time. (The construction problem is: given an instance x, find a witness if x is a yes-instance, and say that there is no witness if x is a no-instance.)

An interesting area for further research is the relationship between generable (i.e., canonical) distributions and the "hard-on-average" distributions studied by Levin et al. ([20], see also [13], [18], and, more recently, [22]). Levin's randomized NP (denoted RNP) is a class of pairs (D, μ), where D is any decision problem in NP and μ is any probability function on $\{0, 1\}^*$ (interpreted as instances of D) for which the cumulative distribution function $\mu^*(x) = \Sigma_{z<x}\mu(z)$ is polynomial-time computable. In [22], Venkatesan and Levin extend the definition to construction problems in NP; the distributions they allow are still those with polynomial-time computable μ^*.

Venkatesan and Levin exhibit a construction problem that is RNP-hard, i.e., if there is an algorithm that can solve it in expected polynomial time, then all RNP-construction problems can be solved in expected polynomial time. The distribution of instances that they consider is easy to generate; however, it assigns positive probability to no-instances. This suggests some natural questions. Is there an RNP-hard distribution (of instances of a construction problem) that assigns positive probability only to yes-instances? Can that distribution be generated efficiently if one insists on generating witnesses along with the instances? Are the requirements that a distribution be efficiently generable and that it have an efficiently computable μ^* mutually exclusive? For example, our canonical generation scheme for Hamiltonian graphs produces a distribution that probably does

not have a polynomial-time computable μ^*: if it did, then the #P-Complete problem of computing the number of Hamiltonian cycles in a graph would be solvable in polynomial time.

Another interesting question is that of uniform generation of instances. If a random exemplifier outputs each $x \in S_n$ with equal probability, then we call it a *uniform generator* for S. Uniform generation is loosely related to the work in [16] and [17]. If S is *rankable* in the sense of [16], then S is uniformly generable. In [17], Jerrum, Valiant, and Vazirani consider the problem of uniform generation of witnesses, i.e., they seek a program that, on input x, generates elements of ω_x^N uniformly. For sets in P, our uniform instance-generation problem is just the problem of [17], applied to tally sets. Thus we know that sets in P can be generated uniformly by a randomized polynomial-time program equipped with a #P oracle or less directly by a randomized polynomial-time program equipped with a Σ_2^{P} oracle.

In view of Theorem 5, there are sets in P that are not uniformly generable, unless RE = NE. Theorem 5 says nothing, however, about the exemplifiability of specific sets in NP, and, in fact many "natural" sets seem to be exemplifiable. What can be said about the uniform generability of a specific NP set, e.g., SAT? Recall that every randomized generator for SAT is a canonical generator with respect to some NP program that accepts it. Thus, if SAT is uniformly generable, it is recognized by an NP machine with respect to which every element of a given size has exactly the same number of witnesses. What are the structural consequences of the existence of such a machine for SAT?

6 Acknowledgements

We are grateful for helpful discussions with Mike Foster, Steve Mahaney, Steven Rudich, and Mihalis Yannakakis.

References

[1] M. Blum and S. Micali. "How to Generate Cryptographically Strong Sequences of Pseudo-random Bits," SIAM J. on Comput. (13), 1984, 850–864.

[2] R. Book. "Tally Languages and Complexity Classes," Inf. and Control (26), 1974, 186–193.

[3] R. Boppana and R. Hirschfeld. "Pseudorandom Generators and Complexity Classes," to appear in Advances in Computer Research, Silvio Micali (ed.), JAI Press (pub.), 1987.

[4] G. Brassard and C. Crépeau. "Non-transitive Transfer of Confidence: A *Perfect* Zero-Knowledge Interactive Protocol for SAT and Beyond," Proceedings of the 27$^{\text{th}}$ FOCS, IEEE, 1986, 188–195.

[5] G. Brassard and C. Crépeau. "Zero-Knowledge Simulation of Boolean Circuits," Advances in Cryptology — CRYPTO86 Proceedings, Andrew Odlyzko (ed.), Springer-Verlag (pub.), 1987, 223–233.

[6] G. Brassard, D. Chaum, and C. Crépeau. "Minimum Disclosure Proofs of Knowledge," to appear.

[7] U. Feige, A. Fiat, and A. Shamir. "Zero Knowledge Proofs of Identity," Proceedings of the 19$^{\text{th}}$ STOC, ACM, 1987, 210–217.

[8] Z. Galil, S. Haber, and M. Yung. "Cryptographic Computation: Secure Fault-Tolerant Protocols and the Public-Key Model," Advances in Crytology — CRYPTO87 Proceedings, Carl Pomerance (ed.), Springer-Verlag (pub.), 1988, 135–155.

[9] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of* NP-*Completeness*, Freeman, San Francisco, 1979.

[10] S. Goldwasser and S. Micali. "Probabilistic Encryption," JCSS (28), 1984, 270-299.

[11] S. Goldwasser, S. Micali, and C. Rackoff. "The Knowledge Complexity of Interactive Proof Systems," to appear in SIAM J. on Comput.

[12] O. Goldreich, S. Micali, and A. Wigderson. "Proofs that Yield Nothing but their Validity and a Method of Cryptographic Protocol Design," Proceedings of the 27$^{\text{th}}$ FOCS, IEEE, 1986, 174–187.

[13] Y. Gurevich. "Complete and Incomplete Randomized NP Problems," Proceedings of the 28$^{\text{th}}$ FOCS, IEEE, 1987, 111–117.

[14] J. Hartmanis. "Generalized Kolmogorov Complexity and the Structure of Feasible Computations," Proceedings of the 24$^{\text{th}}$ FOCS, IEEE, 1983, 439–445.

[15] J. Hartmanis, N. Immerman, and V. Sewelson. "Sparse Sets in NP \ P: EXPTIME vs. NEXPTIME," Inf. and Control (65), 1985, 158–181.

[16] L. Hemachandra and S. Rudich. "On Ranking," manuscript.

[17] M. Jerrum, L. Valiant, and V. Vazirani. "Random Generation of Combinatorial Structures from a Uniform Distribution," TCS (43), 1986, 169–188.

[18] D. Johnson. "The NP-Completeness Column, An Ongoing Guide," JOA (5), 1984, 284–299.

[19] J. Lagarias and A. Oldlyzko. "Solving Low-Density Subset Sum Problems," JACM (32), 1985, 229–246.

[20] L. Levin. "Average Case Complete Problems," SIAM J. on Comput. (15), 1986, 285–286.

[21] S. Rudich, private communication.

[22] R. Venkatesan and L. Levin. "Random Instances of a Graph Coloring Problem are Hard," Proceedings of the 20$^{\text{th}}$ STOC, ACM, 1988, 217–222.

[23] A. C. Yao. "Theory and Applications of Trapdoor Functions," Proceedings of the 23$^{\text{rd}}$ FOCS, IEEE, 1982, 80–91.