Electronic Colloquium on Computational Complexity, Report No. 34 (2010)

The Program-Enumeration Bottleneck in Average-Case Complexity Theory

Luca Trevisan[*]

March 7, 2010

Abstract

Three fundamental results of Levin involve algorithms or reductions whose running time is exponential in the length of certain programs. We study the question of whether such dependency can be made polynomial.

1. Levin's "optimal search algorithm" performs at most a constant factor more slowly than any other fixed algorithm. The constant, however, is exponential in the length of the competing algorithm.

   We note that the running time of a universal search cannot be made "fully polynomial" (that is, the relation between slowdown and program length cannot be made polynomial), unless P=NP.

2. Levin's "universal one-way function" result has the following structure: there is a polynomial time computable function f_{Levin} such that if there is a polynomial time computable adversary A that inverts f_{Levin} on an inverse polynomial fraction of inputs, then for every polynomial time computable function g there also is a polynomial time adversary A_g that inverts g on an inverse polynomial fraction of inputs. Unfortunately, again the running time of A_g depends exponentially on the bit length of the program that computes g in polynomial time.

   We show that a fully polynomial uniform reduction from an arbitrary one-way function to a specific one-way function is not possible relative to an oracle that we construct, and so no "universal one-way function" can have a fully polynomial security analysis via relativizing techniques.

3. Levin's completeness result for distributional NP problems implies that if a specific problem in NP is easy on average under the uniform distribution, then every language L in NP is also easy on average under any polynomial time computable distribution. The running time of the implied algorithm for L, however, depends exponentially on the bit length of the non-deterministic polynomial time Turing machine that decides L.

   We show that if a completeness result for distributional NP can be proved via a "fully uniform" and "fully polynomial" time reduction, then there is a worst-case to average-case reduction for NP-complete problems. In particular, this means that a fully polynomial completeness result for distributional NP is impossible, even via randomized truth-table reductions, unless the polynomial hierarchy collapses.

[*]`luca@cs.berkeley.edu`. U.C. Berkeley, Computer Science Division. This material is based upon work supported by the National Science Foundation under grant No. CCF-0729137 and by the BSF under grant 2006060.

ISSN 1433-8092

1 Introduction

We study three fundamental results in complexity theory, all due to Levin, which involve reductions whose running time is exponential in the *length* of certain programs.

1. Levin [Lev73] describes a *universal search algorithm* for any NP search problem. Instantiated, for example, to SAT, the algorithm has the property that for every other algorithm A there is a constant c_A such that for every satisfiable formula φ Levin's algorithm finds a satisfying assignment for φ in time at most $c_A \cdot t_A(\varphi) + |\varphi|^{O(1)}$, where $t_A(\varphi)$ is the running time of A on input φ.

   That is, compared to any other fixed algorithm, Levin's algorithm is at most a constant factor slower, plus a polynomial additive term, on *each* input. Unfortunately, the constant c_A is exponential in the *length* of the program A. So, far example, Levin's algorithm may be extremely slow on instances that are quickly solved by an algorithm that takes about 10 bytes to write down.

2. Levin describes a *universal one-way function* [Lev87, Lev03] f_{Levin}, with the property that if one way functions exist at all, then f_{Levin} is one-way. Specifically, if A is a polynomial time algorithm that inverts f_{Levin} on at least an inverse polynomial fraction of inputs and if g is any polynomial time computable function, then A can be used to invert g on an inverse polynomial (actually, even constant) fraction of inputs. The running time of the procedure that inverts g, however, is exponential in the bit length of the program that computes g in polynomial time.

   Again, this could mean that it is possible to have a strong attack on f_{Levin} which would not have effective applications to invert functions like AES or RSA, or indeed any function that requires more than, say, 10 bytes to specify.

3. A key result in Levin's theory of average-case complexity is a completeness result [Lev73] that also involves an "exponential dependency on program length," although of a nature that is slightly more complicated to explain, affecting the way the distribution produced by the reduction is "dominated" by the distribution of the complete problem.

   The end result, however is the following: if there is a polynomial-time-on-average algorithm for the complete problem then for every language L in NP and every "polynomial time computable" distribution D, there is a polynomial-time-on-average algorithm for L under distribution D. The complexity of the latter algorithm, however, is exponential on the bit length of the non-deterministic algorithm that decides L (and which puts L into NP).

It is natural to wonder if the exponential dependency is necessary in these application. (I first learned of this question from Charles Rackoff.)

We show that in (1), the universal search algorithm, achieving a polynomial dependency on the length of the program A would imply that $P = NP$. This is proved via a very simple observation that might have been noted before, although we are not aware of any such reference.

Regarding (2), the universal one-way function, we show that a polynomial dependency on the code length of g in the reduction would yield a reduction transforming "infinitely-often distributional one-way functions" into "infinitely-often one-way functions," a reduction which would be a surprising, and perhaps unlikely, result. We describe an oracle relative to which such a reduction is impossible. This means that a superpolynomial dependency on the code length of g in Levin's result is necessary in any relativizing approach.

We give a definitional treatment of the issues arising in (3), the average-case completeness result for NP. We show that an average-case completeness proof in which one reduces an NP distributional problem (L, D) to a complete problem via a reduction of complexity polynomial in the description of L[1], would yield a *worst-case to average-case reduction* for NP. In fact, it is not hard to see that the converse is also true, and so the issue of efficiency in Levin's average-case complexity theory for NP is essentially identical to the problem of existence of worst-case to average-case reductions for NP A result of Bogdanov and Trevisan [BT06b] shows that the existence of such reductions implies the collapse of the polynomial hierarchy, even if we consider randomized truth-table reductions.

A Remark on Models of Computation. In this paper we refer, informally, to *algorithms*, *running time*, and *length of an algorithm*. The arguments are model-indpendent, so any reasonable instantiation of the above notions could be used and any reasonable encoding (to measure the bit length of an algorithm) could be adopted. For concreteness, the reader may fix a universal (say, two-tape) Turing machine U, and take "algorithm A" to refer to a first input A for U, "running time of A on input x" to refer to the number of steps of the computation $U(A, x)$, and "length of A" as the bit length of A.

2 Levin's Universal Search Algorithm

2.1 The Question

The first result that we discuss, dating back to [Lev73], is a search algorithm due to Levin to find certificates of YES instances of problems in NP. Instantiated to SAT, the algorithm, given a formula φ, enumerates all possible algorithms and runs them in parallel, until one of the algorithm finds a satisfying assignment. If φ is not satisfiable, then the algorithm never terminates, but the "scheduling" in which various algorithms are tried and timed out can be optimized so that, for every algorithm A, and for every satisfiable φ for which A finds a satisfying assignment, Levin's algorithm finds an assignment in time that is at most a constant factor slower than the running time of A (plus an additive term related to the time that it takes to verify the validity of a solution). More precisely, we have the following statement.

Theorem 1 *There is an algorithm A_{Levin} such that for every algorithm A there is a constant c_A such that if φ is a satisfiable instance of 3SAT and A finds a satisfying assignment for φ in time $t_A(\varphi)$ then A_{Levin} finds a satisfying assignment for φ in time at most*

$$c_A \cdot t_A(\varphi) + |\varphi|^{O(1)}$$

[1]More precisely, polynomial in the bit length of the non-deterministic polynomial time machine that puts L in NP.

That is, Levin's algorithm has a running time that is at most linear (plus an additive term polynomial in the length of the input) in the running time of every other algorithm, on every input. The slowdown constant c_A, however, is exponential in the bit length of the algorithm A.

Open Question 1 *Is there a universal search algorithm whose slowdown compared to a fixed algorithm A is at most polynomial on the description length of A?*

Hutter [Hut02] describes and analyses a universal search algorithm that is at most 5 times slower, plus additive terms, than any fixed *provably correct* algorithm A on any specific instance. An algorithm A being provably correct means that there exists a proof of the statement "for every formula φ, if A outputs an assignment then the assignment satisfies φ." The additive term in the running time is exponential in the length of such a proof, and also exponential in the length of the algorithm A.

2.2 Our Result

We note a simple argument showing that if Question 1 has a positive answer, then P=NP.

Question 1 refers to the existence of an algorithm as follows:

Definition 2 (Optimal Search) *A fully polynomially optimal search algorithm for SAT is an algorithm S that given an satisfiable formula φ finds a satisfying assignment and such that for every Turing machine A and every formula φ of length n for which A finds a satisfying assignment in time t, S also finds a satisfying assignment in time that is at most a $t \cdot n^{O(1)} \cdot |A|^{O(1)}$.*

And we prove the following result:

Theorem 3 *Suppose that a fully polynomially optimal search algorithm for SAT exists. Then P=NP.*

Proof: For a satisfiable formula φ, consider the algorithm A_φ that on input a formula ψ runs an exponential-time search to find a satisfying assignment for ψ if $\psi \neq \varphi$, and outputs a satisfying assignment for φ which is hard-wired in the code for A if $\psi = \varphi$. Clearly, the code of A_φ is linear in the size of φ and its running time on input φ is also linear, so S must find a satisfying assignment for φ in polynomial time. $\Box$

We add a couple of remarks:

- A fully polynomial optimal search algorithm for SAT clearly exists if $P = NP$, so the existence of such an algorithm is in fact equivalent to $P = NP$.
- Regarding Hutter's result, note that each algorithm A_φ that we use in our proof of Theorem 3 is a provably correct algorithm, with a proof of correctness of length poly($|\varphi|$). This means that if the additive term in Hutter's analysis could be made polynomial in the length of the algorithm and of its proof of correctness, then we would also have $P = NP$.

3 Levin's Universal One-Way Function

3.1 The Question

Levin proves in [Lev87] (the result is stated more explicitly in [Lev03]) the existence of a *universal* one-way function, a result related to his *optimal search algorithm* [Lev73] discussed in the previous section.

The basic idea is to consider the function $u(M,x)$ that takes in input an algorithm M and a string x, simulates M on input x for $|x|^2$ steps, and then outputs $\langle M, M(x)\rangle$ if the computation of M terminates within the prescribed time and if $|M(x)| = |x|$. (The choice of a quadratic time bound is rather arbitrary.)

Suppose now that there exists a one-way function f, that is, a polynomial time computable, length-preserving function such that for every polynomial time computable inverting algorithm A, every polynomial $p()$, and every sufficiently large n we have

$$\mathop{\mathbb{P}}_{x\sim\{0,1\}^n}[A(f(x)) = x' : f(x') = f(x)] \leq \frac{1}{p(n)}$$

(See [Gol01] for more context.) By a padding argument, if f is a one-way function, then there is a one-way function f' that is computable at most in time n^2 on inputs of length n for all sufficiently large n.[2]

Let M' be the algorithm that computes f' and let ℓ be the length of M'. Then we have that for every polynomial-time algorithm A, every polynomial p and every sufficiently large n we have

$$\mathop{\mathbb{P}}_{x\sim\{0,1\}^n}[A(u(x)) = x' : u(x') = u(x)] \leq 1 - \frac{1}{2^\ell} + \frac{1}{p(n)} \tag{1}$$

This doesn't imply that u is a one-way function, but we can define the function $f_{\text{Levin}}(x)$ that, given an input x of length n^2, parses it as n inputs $x_1, \ldots, x_n$ each of length n and then computes

$$f_{\text{Levin}}(x) := u(x_1), \ldots, u(x_n)$$

The "hardness amplification" result of Yao [Yao82] (see [Gol01] for proofs and more context) implies that if u is a one-way function in the weak sense of (3), then f_{Levin} is a one-way function in the standard strong sense.

Intuitively, once $n >> 2^\ell$, then with high probability at least one of the x_i will be of the form (M', z), and in such a case inverting f_{Levin} becomes at least as hard as inverting f' in a random input.[3]

Note that, if we do not believe, say, in the existence of one-way functions that can be defined with less than 200 bits of code in a particular model of computation, then this construction does not guarantee the security of Levin's function on input of length less than 2^{200} (indeed, because of the

[2]If the computation of $f(x)$ requires time at most $t(|x|)$, then we can define f' to do the following: given an input of length n, find the largest m such that $t(m) \leq n^2/2$, and then apply f to the first m bits of the input.

[3]The rigorous proof is rather subtle, but it follows this intuition.

squaring, the need of a prefix-free encoding to encode tuples, and the need to provide an input, one has to get up to lengths that are actually around 2^{450}).

In light of the impossibility of using Levin's universal function with any practical set of parameters, Rackoff has raised the question of whether the exponential dependency on the length of the code is necessary.

We formalize Rackoff's question as the existence of a universal one-way function with the following kind of efficient security reduction.

Definition 4 (Efficient Universal One-Way Function) *A function $f_{\text{universal}}$ is a universal one-way function under* fully polynomial *and* fully uniform *reductions if there is an algorithm R (for reduction) such that for every polynomial q, every inverting oracle A such that*

$$\mathop{\mathbb{P}}_{x \in \{0,1\}^n} [A(f_{\text{universal}}(x)) = x' : f_{\text{universal}}(x') = f_{\text{universal}}(x)] \geq \frac{1}{q(n)} \ ,$$

every length-preserving function g computable in polynomial time $p(n)$ by an algorithm M_g, we have

$$\mathop{\mathbb{P}}_{x \in \{0,1\}^n} [R^A(g(x), M_g, q, p) = x' : g(x) = g(x')] \geq \frac{1}{2}$$

and R runs in time polynomial in $n = |g(x)|$, in $q(n)$, in $p(n)$, and in $|M_g|$.

Note the function f_{Levin} comes with a reduction R that has all the above properties except that the running time is exponential, rather than polynomial in $|M_g|$.

We are, thus, interested in the question

Open Question 2 *Is there a universal one-way function whose security is established by a fully polynomial and fully uniform reduction?*

We should clarify a point about the terminology that we use: a universal one-way function is not necessarily a one-way function. If one-way functions exist, then every universal one-way function is indeed one-way, but if one-way functions do not exist then no function can be one-way, including the universal "one way" functions. (This is a bit as if we were calling NP-complete problems "universal intractable problems.")

3.2 Some Additional Background on One-Way Functions

We begin by providing definitions of weak and strong one-way functions and of infinitely-often one-way functions.

Definition 5 (One-Way Function) *A polynomial time computable function f is:*

- *a (strong) one way function if for every probabilistic polynomial time algorithm A and every polynomial q we have*

$$\mathop{\mathbb{P}}_{x \sim \{0,1\}^n} [A(f(x) = x' : f(x) = f(x')] \leq \frac{1}{q(n)} \qquad (2)$$

for all sufficiently large n

- *an infinitely-often (strong) one-way function, abbreviated i.o.-one-way, if* (2) *holds only for infinitely many n*
- *a weak one way function if there exists a polynomial q such that for every probabilistic polynomial time algorithm A*

$$\mathop{\mathbb{P}}_{x\sim\{0,1\}^n}[A(f(x) = x' : f(x) = f(x')] \leq 1 - \frac{1}{q(n)} \tag{3}$$

for all sufficiently large n

- *an infinitely-often weak one-way function, abbreviated weak i.o.-one-way if* (3) *holds only for infinitely many n,*

If follows by hardness amplification results of Yao [Yao82] (see [Gol01]) that a strong one-way function exists if and only one a weak one-way function exists, and that a strong i.o.-one-way function exists if and only a weak i.o.-one-way function exists.

3.3 Our Results

We prove the following result.

Theorem 6 *There is an oracle relative to which Question 2 has a negative answer.*

Theorem 6 is proved in two steps: first we present a relativizing proof that a positive answer to Question 2 would show that the non-existence of i.o.-one way functions implies the seemingly stronger statement that a "fully polynomial universal inverter" exists, and then we construct an oracle relative to which i.o.-one way functions do not exist, but fully polynomial universal inverters do not exist either.

Definition 7 (Fully Polynomial Universal Inverter) *A fully polynomial universal inverter is a randomized polynomial time algorithm A such that for every circuit C computing a length preserving function $f : \{0,1\}^n \to \{0,1\}^n$ we have*

$$\mathop{\mathbb{P}}_{x\sim\{0,1\}^n}[A(C, f(x)) = x' : f(x') = f(x)] \geq \frac{1}{2}$$

We note that the existence of a fully polynomial universal inverter is implied by $NP \subseteq BPP$, and it implies the non-existence of i.o.-one-way functions, and even of non-uniformly computable i.o.one-way functions, but that such conditions do not appear to be equivalent.

Lemma 8 *Suppose that there is a universal one-way function under fully polynomial and fully uniform reductions, and suppose that i.o.-one-way functions do not exist. Then there is a fully polynomial universal inverter.*

Furthermore, the above statement is true relative to any oracle.

Perhaps it is worth clarifying again that a "universal one-way function" is not necessarily a one-way function, but it is a function that has the property of being one-way if one-way functions exist at all.

Proof: [Of Lemma 8] Let $f_{\text{universal}}$ be the universal one-way function, and R the fully uniform and fully polynomial reduction. If i.o.-one-way functions do not exist, then there is an algorithm I that inverts $f_{\text{universal}}$ on a $\geq 1 - 1/poly(n)$ inputs of length n. Given a circuit C of size s that computes a length-preserving function $f : \{0,1\}^n \to \{0,1\}^n$, it is possible to construct in time $s^{O(1)}$ a Turing machine M_C such that M computes f in time $t = s^{O(1)}$ and such that $|M| = s^{O(1)}$. Now define

$$A(C, f(x)) = R^I(f(x), M_C, 2, t)$$

For the "furthermore" part, note that the above proof relativizes. □

Our second main result, Theorem 6 is now a consequence of the following result and the above discussion.

Lemma 9 *There is an oracle relative to which there is no i.o.-one-way functions and no fully polynomial universal inverter.*

Proof: We pick a random ensemble of functions $f_n : \{0,1\}^n \to \{0,1\}^n$, and a random sequence of "keys" $k_1, \ldots, k_n, \ldots$, where $k_n \in \{0,1\}^n$. The oracle is the disjoint union of the oracle for a PSPACE-complete problem (say, QBF), and on oracle F that given the query (x, y), with $|x| = |y| = n$, returns $f_n(x)$ if $y = k_n$ and 0^n if $y \neq k_n$.

Relative to this oracle there is no fully polynomial universal inverter. Indeed, suppose that there was such an inverter A, and consider, for every n, the linear size oracle circuit C_n that computes f_n (the circuit C_n has the key k_n hard-wired into it); then A would able to invert f_n in polynomial time on half the inputs given only oracle access to f_n in the forward direction and given a PSPACE oracle. It follows from calculations of Impagliazzo and Rudich [IR89] on the hardness of inverting random functions that this happens with probability zero over the choice of the randomness in the oracle.

On the other hand, relative to this oracle, there is no i.o.-one-way function. Let n^c be any polynomial and g be any function computable in time n^c relative to the oracle. Define g' to be the function that is computed by simulating the algorithm for g in every step, except that queries to the F oracle of length $2m > 4c \log n$ receive the simulated answer 0^m. We claim that

$$\mathop{\mathbb{P}}_{x \sim \{0,1\}^n} [g(x) = g'(x)] \geq .99$$

This follows from the fact that $g(x)$ and $g(x')$ disagree only when the algorithm that computes g receives a non-trivial answer from F for a query of length $2m > 4c \log n$, that is, if g is able to find k_m for some $m > 2c \log n$, while making only n^c oracle queries. With probability 1 over the randomness in F, this happens for a $o(1)$ fraction of the inputs x.

Consider the probabilistic algorithm A that, given y, generates the uniform distribution over the set of pre-images $\{x : g'(x) = y\}$. Such an algorithm can be implemented in polynomial time given access to a PSPACE oracle, the algorithm for g, and the list of the n^{4c} oracle answers provided by F for queries (x, y) where $|x| \leq 2c \log n$. Finally, we use a result of [LTW05] showing that if $\mathbb{P}_{x \sim \{0,1\}^n}[g(x) = g'(x)] = p$, and A is an algorithm that given y generates a random preimage of y under g', then $\mathbb{P}_{x \sim \{0,1\}^n}[A(g(x)) = x' : g(x) = g(x')] \geq p^2$. $\square$

4 Levin's Average-Case Completeness Result

It is not completely straightforward to even state the question that we want to study in Levin's theory of average-case complexity. There are at least two reasons: (i) the definitions in the theory are rather subtle, and come in several variants, so that it takes some time even to just outline the standard definitional treatment and (ii) our question relates to the complexity of reductions in a completeness result as a *function of the bit-length of the machine that defines the problem we reduce from*, a question that is rather important but rarely explicitly studied, so that there isn't even a standard terminology to refer to questions of this form.

4.1 "Full Uniform" and "Fully Polynomial" Completeness Results

The Cook-Levin theorem, the prototype of all completeness results in complexity theory, states that for every language L in NP there is a polynomial time computable function f_L such that for every x we have $x \in L \Leftrightarrow f_L(x) \in SAT$.

The proof of the theorem is via a reduction that is *fully uniform* and *fully polynomial*, in the sense that we explain below.

- The theorem only requires the *existence* of a reduction function $f_L(\cdot)$ for every L in NP, and does not put any requirement on how to construct f_L given a specification for L. The proof, however, proceeds by defining a single uniform algorithm $f(\cdot, \cdot, \cdot)$ such that if M is a non-deterministic polynomial time machine that decides L, $t(n)$ is a polynomial upper bound to the running time of M on inputs of length n, and x is any input, then the mapping $x \rightarrow f(M, x, t(|x|))$ is a reduction from L to SAT. We refer to such a reduction as *fully uniform*.
- For a function $f(\cdot, \cdot, \cdot)$ as above to prove the Cook-Levin theorem, it is sufficient that $f(M, x, t(x))$ be computable in time polynomial in $|x|$ and $t(|x|)$. In the known proof, however, $f(\cdot, \cdot, \cdot)$ is computable in time polynomial in $|M|$ as well. We refer to such a reduction as *fully polynomial*.

Essentially all completeness results in complexity theory are proved via fully uniform and fully polynomial reductions. Furthermore, the existence of fully uniform and fully polynomial completeness results for all the complexity classes known to have complete problems *relativizes*.

As noted by Charles Rackoff, an important exception are completeness results in the theory of average-case complexity, specifically Levin's "universal one-way function" [Lev87, Lev03], but also Levin's completeness result for distributional NP [Lev86] and Levin's "optimal search algorithm" [Lev73]. Such results are proved via reductions that are fully uniform but that do not "fully polynomially" preserve average-case solvability. In order to discuss this efficiency issue in average-case completeness results, we need to briefly summarize the set-up. The reader is referred to the survey papers of Impagliazzo [Imp95], Goldreich [Gol97], and Bogdanov and Trevisan [BT06a] for a broader context.

4.2 Distributional Problems, Average-Case Tractability, and Reductions

We follow the definitional set-up of Impagliazzo [Imp95]. We want to prove the following completeness result: that there is a specific problem BH in NP such that if we have a good-on-average algorithm for BH with respect to the uniform distribution of instances, then we also have good-on-average algorithms for all problems in NP and with respect to all distributions coming from a rather general class of "computable" distributions of instances.

Definition 10 (Distributional Problem) *A* distributional problem *is a pair $\langle L, D\rangle$, where L is language and D is a distribution over input instances, or, more precisely, an* ensemble *of distributions $(D_1, \ldots, D_n, \ldots)$ where D_n is a distribution over $\{0,1\}^n$.*

Definition 11 (AvgP) *A distributional problem $\langle L, D\rangle$ is in* average polynomial time *if there is an algorithm A such that, on input x and $\varepsilon > 0$, A runs in time poly$(|x|, \varepsilon^{-1})$ and returns either the correct answer to $x \overset{?}{\in} L$ or the error message $\perp$; furthermore, for every n,*

$$\mathop{\mathbb{P}}_{x \sim D_n}[A(x, \varepsilon) = \perp] \leq \varepsilon \tag{4}$$

The class of distributional problems admitting an average polynomial time algorithm is denoted by AvgP.

We can define the class AvgZPP analogously, but allowing A to be randomized taking the probability over the randomness of A as well as over $x \sim D_n$ in the left-hand side of (4).

Other definitions and variants are interesting, for example we may allow A to output an incorrect answer, and then bound the probability that an incorrect answer is given, leading to the definition of the class of Heuristic Polynomial time, HeurP [Imp95]. The randomized class HeurBPP is defined analogously to HeurP, but allowing A to be randomized.

Definition 12 (Reduction) *A many-to-one reduction from a distributional problem $\langle L_1, D_1\rangle$ to a distributional problem $\langle L_2, D_2\rangle$, written $\langle L_1, D_1\rangle \leq_m^{avgP} \langle L_2, D_2\rangle$, is a polynomial time computable function f such that*

1. *$x \in L_2 \Leftrightarrow f(x) \in L_2$*
2. *There is a polynomial $d(\cdot)$ such that for every n, every m, and every $y_0 \in \{0,1\}^m$*

$$\mathop{\mathbb{P}}_{x \sim D_{1,n}}[f(x) = y_0] \leq d(n) \cdot \mathop{\mathbb{P}}_{y \sim D_{2,m}}[y = y_0]$$

That is, the distribution (or, rather, ensemble of distributions) generated by applying $f()$ to random element of $D_{1,n}$ is $d(n)$-dominated by the ensemble D_2.

In a randomized reduction, f is the output of a randomized algorithm, property (1) is required to hold with probability 1 *over the randomness of f, and property (2) is required to hold by taking the probability over the randomness of f on the let-hand side.*

If we have a reduction $\langle L_1, D_1\rangle \leq_m^{AvgP} \langle L_2, D_2\rangle$, and $\langle L_2, D_2\rangle$ admits an AvgP-type algorithm of running time $t(n,\varepsilon)$, then it is easy to see that $\langle L_1, D_1\rangle$ also has a AvgP-type algorithm, of running time about $r(n) + t(\ell(n), \varepsilon/(\ell(n)\cdot d(n)))$, where $r(|x|)$ is an upper bound to the running time of $f(x)$, $\ell(|x|)$ is an upper bound to the length of $f(x)$, and $d(n)$ is the domination parameter. A randomized reduction preserves AvgZPP and HeurBPP algorithms. It is important to note that the domination parameter affects the running time of the algorithm for $\langle L_1, D_1\rangle$, and that a reduction with a super-polynomial domination parameter does not preserve AvgP solvability. (Nor would it preserve AvgZPP or HeurP solvability, or most other reasonable definitions of average-case tractability.)

4.3 The Question

Consider the following problem BH: on input a triple (M, x, z), where M is a non-deterministic Turing machine, the question is whether $M(x)$ has an accepting branch that halts within $|z|$ steps. The triple (M, x, z) is encoded as a binary string by having M and x be encoded with a prefix-free encoding; we assume that we use an encoding that maps a string of length ℓ into an encoding of length $\ell + O(\log \ell)$. Consider the uniform ensemble $U = (U_1, \ldots, U_n, \ldots)$ where U_n is the uniform distribution over $\{0,1\}^n$.

First we note that $\langle BH, U\rangle$ is *complete* under randomized reductions, for the class $\langle NP, U\rangle$ of all distributional problems of the form $\langle L, U\rangle$, where L is in NP. Indeed, let L be in NP, and let M be a non-deterministic Turing machine that decides L in time $p(n)$, where p is a polynomial and n is the input length. Then consider the randomized reduction

$$x \to (M, x, r)$$

where r is a randomly chosen string of length $p(n)$. This clearly satisfies property (1) of the definition of reduction. About property (2), if we apply the reduction to a random element of U_n we obtain strings of length $\ell(n) = |M| + n + p(n) + O(\log |M|) + O(\log n)$, and each such string is generated with probability $2^{-n}\cdot 2^{-p(n)}$ by the reduction. This implies that the reduction has domination parameter

$$d(n) = 2^{|M|}\cdot |M|^{O(1)}\cdot n^{O(1)}$$

Note that while the domination parameter is polynomial in n, it is exponential in $|M|$, and that a price polynomial in the domination parameter (and hence exponential in $|M|$) is paid when we convert a good-on-average algorithm for $\langle BH, U\rangle$ into a good-on-average algorithm for $\langle L, U\rangle$. This is a comparable slow-down to having to *find* M via a complete enumeration.

Levin [Lev86] proves the existence of a problem complete for the class $\langle NP, Pcomp\rangle$, which includes all distributional problems $\langle L, D\rangle$ where L is in NP and D is P-computable, and $\langle BH, U\rangle$ is such a complete problem under randomized reductions.[4] Impagliazzo and Levin [IL90] prove that $\langle BH, U\rangle$

[4]See [Lev86, Gol97, BT06a] for definitions and details of the reduction.

is complete for the more general class $\langle NP, Psamp\rangle$ of all problems of the form $\langle L, D\rangle$ where L is in NP and D is polynomial-time samplable. In such results, too, the domination parameter in the reduction is exponential in the description length of the Turing machine that decides L. In the Impagliazzo-Levin result, the domination parameter is also exponential in the description length of the sampler for D, and the reduction is a more sophisticated type of truth-table reduction.

Since the domination parameter is a *bona fide* complexity parameter, it is natural to consider a completeness result in the theory of average-case complexity to be *fully polynomial* if both the reduction running time and the domination parameter are polynomial in x and in $|M|$, where M is the machine that decides the language that we want to reduce to the complete problem.

Even leaving aside the more sophisticated completeness results for $\langle NP, Pcomp\rangle$ and $\langle NP, Psamp\rangle$, the following question is natural:

Open Question 3 *Is it possible to prove that there is a $\langle NP, U\rangle$-complete problem via a fully uniform and fully polynomial many-to-one reduction?*

4.4 Our Result

We prove that Question 3 has a positive answer if and only if there is a many-to-one worst-case to average-case reduction for NP-complete problems. Bogdanov and Trevisan show that every non-adaptive worst-case to average-case reduction for NP-complete problems implies the collapse of the polynomial hierarchy, hence

Theorem 13 *If Question 3 has a positive answer, then the polynomial hierarchy collapses.*

Definition 14 *A* non-adaptive worst-case to average-case reduction *from a problem W to a distributional problem $\langle L, D\rangle$ is a polynomial time randomized non-adaptive oracle algorithm R with the property that there exists a polynomial q such that for every oracle A such that*

$$\mathop{\mathbb{P}}_{x\sim D_n}[A(x) = \mathbf{1}_L(x)] \geq 1 - \frac{1}{q(n)}$$

we have that for every w

$$\mathbb{P}[R^A(w) = \mathbf{1}_W(w)] \geq \frac{3}{4}$$

where the probability is taken only over the internal coin tosses of R.

Informally, the definition says that R converts a good-on-average HeurBPP-type algorithm for $\langle L, D\rangle$ into a worst-case BPP-type algorithm for W. The main result of [BT06b] is that such reductions are unlikely to exist from NP-complete problems.

Theorem 15 ([BT06b]) *Suppose that there is a non-adaptive worst-case to average-case reduction from an NP-complete problem W to a problem $\langle L, D\rangle$ in $\langle NP, Psamp\rangle$. Then the polynomial hierarchy collapses.*

Our first main result, Theorem 13 follows from the following result and the above discussion.

Lemma 16 *Suppose that $\langle L, D\rangle$ is a $\langle NP, U\rangle$-hard problem in $\langle NP, Psamp\rangle$ under a fully uniform and fully polynomial reduction f.*

Then there is a non-adaptive worst-case to average-case reduction from SAT to $\langle L, D\rangle$.

PROOF: Let $f(\cdot,\cdot,\cdot)$ be the fully uniform and fully polynomial reduction establishing the completeness of $\langle L, D\rangle$. Recall that this means that for every distributional problem $\langle L', U\rangle$ in $\langle NP, U\rangle$, if M' is a non-deterministic turing machine and $p()$ is a polynomial such that M' recognizes L' in time $p(n)$, then the mapping

$$x \to f(M', x, p(|x|)$$

is a reduction from $\langle L', U\rangle$ to $\langle L, D\rangle$.

We want to prove that there is a a non-adaptive worst-case to average-case reduction from SAT to $\langle L, D\rangle$. That is, we want to define an algorithm R and find a polynomial q such that if A is an oracle such that $\mathbb{P}_{x\sim D_n}[A(x) = \mathbf{1}_L(x)] \geq 1 - 1/q(n)$ for all n, then $R^A(\cdot)$ solves SAT on all inputs, with error probability at most 1/4.

Given an instance φ of SAT of size n, we define $R^A(\varphi)$ as follows.

- Construct a non-deterministic Turing machine M_φ that ignores its input and solves φ in $p(n) = n^{O(1)}$ time. (Note that $|M_\varphi| = n^{O(1)}$.) That is, if φ is satisfiable then $M_\varphi(x)$ accepts for every input x, and the language recognized by M_φ is $\{0,1\}^*$; and if φ is not satisfiable then $M_\varphi(x)$ rejects for every input x, and so the language recognized by M_φ is the empty language.
- Pick a random x also of length n, and compute $y = f(M_\varphi, x, p(n))$.
- Make the query y to the oracle A, and output the answer of the oracle

It follows from the property of the reduction $f(\cdot,\cdot,\cdot)$ that y is distributed with domination $d(n) = n^{O(1)}$ relative to ensemble D, so there is a polynomial q such that if A makes at most $1/q(m)$ mistakes on inputs of length m then there is probability at least 3/4 over the choice of x and the randomness of f (if f is a randomized reduction) that the oracle answer of A to the query $f(M_\varphi, x, p(n))$ is the same as the answer to the question "is φ satisfiable?" $\Box$

We note that the existence of worst-case to average-case reductions and the collapse of the polynomial hierarchy follows not just from a completeness result via fully uniform and fully polynomial *many-to-one* reductions, but also from any (appropriately defined) fully uniform and fully polynomial randomized non-adaptive reductions.

Interestingly, the converse is also true: if we have a worst-case to average-case reduction from SAT to a problem $\langle L, D\rangle$, then we can fully uniformly and fully polynomially reduce any problem $\langle L', D'\rangle$ with $L' \in NP$ to $\langle L, D\rangle$ by first reducing L' to SAT via Cook's theorem, and then using the worst-case to average-case reduction from SAT to $\langle L, D\rangle$.

What about a completeness result proved via a fully uniform and fully polynomial *adaptive* randomized reduction? By the same argument, such a result is equivalent to the existence of an *adaptive* worst-case to average-case reduction within NP. The latter result is not known to imply unexpected collapses in complexity theory. Regarding relativizing results, Impagliazzo and Rudich (referenced in [Imp95]) construct an oracle relative to which $BPP \not\subseteq NP$, but every distributional problem in $\langle NP, U\rangle$ (and hence $\langle NP, PSamp\rangle$) admits a HeurP algorithm. In particular, relative to this oracle there is no adaptive worst-case to average-case reduction within NP, and so there cannot be a completeness result for $\langle NP, U\rangle$ proved via even an adaptive randomized fully uniform and fully polynomial reduction.

Acknowledgments

I would like to thank Charles Rackoff for raising the question of the efficiency of the reduction in Levin's universal one-way function.

References

[BT06a] Andrej Bogdanov and Luca Trevisan. Average-case complexity. *Foundations and Trends in Theoretical Computer Science*, 2(1), 2006. Also arXiv:cs/0606037.

[BT06b] Andrej Bogdanov and Luca Trevisan. On wost-case to average-case reductions for NP problems. *SIAM Journal on Computing*, 36(4):1119–1159, 2006. Preliminary version in *FOCS'03*.

[Gol97] Oded Goldreich. Notes on Levin's theory of average-case complexity. Technical Report TR97-058, Electronic Colloquium on Computational Complexity, 1997.

[Gol01] Oded Goldreich. *The Foundations of Cryptography - Volume 1*. Cambridge University Press, 2001.

[Hut02] Marcus Hutter. The fastest and shortest algorithm for all well-defined problems. *International Journal of Foundations of Computer Science*, 13(3):431–443, 2002.

[IL90] Russell Impagliazzo and Leonid Levin. No better ways to generate hard NP instances than picking uniformly at random. In *Proceedings of the 31st IEEE Symposium on Foundations of Computer Science*, pages 812–821, 1990.

[Imp95] Russell Impagliazzo. A personal view of average-case complexity. In *Proceedings of the 10th IEEE Conference on Structure in Complexity Theory*, pages 134–147, 1995.

[IR89] Russell Impagliazzo and Steven Rudich. Limits on the provable consequences of one-way permutations. In *Proceedings of the 21st ACM Symposium on Theory of Computing*, pages 44–61, 1989.

[Lev73] Leonid A. Levin. Universal search problems. *Problemi Peredachi Informatsii*, 9:265–266, 1973.

[Lev86] Leonid Levin. Average case complete problems. *SIAM Journal on Computing*, 15(1):285–286, 1986.

[Lev87] Leonid Levin. One-way functions and pseudorandom generators. *Combinatorica*, 7(4):357–363, 1987.

[Lev03] Leonid Levin. The tale of one-way functions. arxiv:cs.CR/0012023, 2003.

[LTW05] Henry Lin, Luca Trevisan, and Hoeteck Wee. On hardness amplification of one-way functions. In *Proceedings of the 2nd Theory of Cryptography Conference*, pages 34–49, 2005.

[Yao82] Andrew C. Yao. Theory and applications of trapdoor functions. In *Proceedings of the 23th IEEE Symposium on Foundations of Computer Science*, pages 80–91, 1982.

ECCC ISSN 1433-8092

http://eccc.hpi-web.de