Randomness vs. Time: De-randomization under a uniform assumption

Russell Impagliazzo*
Department of Computer Science
University of California
San Diego, CA 91097-0114
russell@cs.ucsd.edu

Avi Wigderson†
Institute of Computer Science
Hebrew University
Jerusalem, Israel 91904
avi@cs.huji.ac.il

Abstract

We prove that if $BPP \neq EXP$, then every problem in BPP can be solved deterministically in subexponential time on almost every input (on every samplable ensemble for infinitely many input sizes). This is the first derandomization result for BPP based on uniform, non-cryptographic hardness assumptions. It implies the following gap in the average-instance complexities of problems in BPP: either these complexities are always sub-exponential or they contain arbitrarily large exponential functions.

We use a construction of a small "pseudo-random" set of strings from a "hard function" in EXP which is identical to that used in the analogous non-uniform results of [21, 3]. However, previous proofs of correctness assume the "hard function" is not in $P/poly$. They give a non-constructive argument that a circuit distinguishing the pseudo-random strings from truly random strings implies that a similarly-sized circuit exists computing the "hard function". Our main technical contribution is to show that, if the "hard function" has certain properties, then this argument can be made constructive. We then show that, assuming $EXP \subseteq P/poly$, there are EXP-complete functions with these properties.

*Research supported by NSF Award CCR-92-570979, Sloan Research Fellowship BR-3311, grant #93025 of the joint US-Czechoslovak Science and Technology Program, and USA-Israel BSF Grant 92-00043

†This research was supported by grant number 69/96 of the Israel Science Foundation, founded by the Israel Academy for Sciences and Humanities

1 Introduction, History, and Intuition

1.1 Motivation

The introduction of randomization into efficient computation has been one of the most fertile and useful ideas in computer science. In cryptography and asynchronous computing, randomization makes possible tasks that are impossible to perform deterministically. Even for function computation, many examples are known in which randomization allows considerable savings in resources like space and time over deterministic algorithms, or even "only" simplifies them.

But to what extent is this seeming power of randomness over determinism real? The most famous concrete version of this question regards the power of BPP, the class of problems solvable by probabilistic polynomial time algorithms making small constant error. What is the relative power of such algorithms compared to deterministic ones? This is largely open. On the one hand, it is possible that $P = BPP$, i.e., randomness is useless for solving new problems in polynomial time. On the other, we might have $BPP = EXP$, which would say that randomness would be a nearly omnipotent tool for algorithm design. *A priori*, neither extreme seems likely: there are some problems where randomness seems exponentially helpful, but many hard problems are not susceptible to randomized solutions.

In this paper, we show that the intuition that randomness is a resource basically incomparable to time is wrong. Either there is a non-trivial deterministic simulation of BPP, or $BPP = EXP$!

Either time can non-trivially substitute for randomness, or randomness can non-trivially substitute for time. In other words, either universal de-randomization is possible, or randomization is a panacea for intractability. (There are some technical provisos: the deterministic simulation only works for infinitely many input lengths, and may fail on a negligible fraction of inputs even of these lengths.) We consider the former much more plausible than the latter.

1.2 History: Hardness vs. Randomness

While counter to most people's first intuition, our result should be less surprising to those who are aware of the literature on de-randomization. The fundamental paradigm in de-randomization is to trade "hardness" for "randomness". This was first elucidated in the remarkable sequence of papers [22, 6, 24]. Roughly speaking, "computationally hard" functions can be used to construct "efficient pseudo-random generators". These in turn lower the randomness requirements of any efficient probabilistic algorithm, allowing for a "nontrivial" deterministic simulation.

In many such results, there is a quantitative trade-off between the hardness assumption and the time to perform the deterministic simulation. The stronger the assumption, the faster the simulation. Here, we are concentrating on the "low end" of the curve in this trade-off: what is the weakest assumption one can make and still have some version of universal derandomization? Our results also have some implications for the "higher end" of the curve, but these are much less clean, and we will not fully describe them in this abstract.

We will thus compare our results mainly to the "low end" version of the known results. In particular, we will use as our standard for "nontrivial" the class $SUBEXP = \cap_{\delta>0} DTIME(2^{n^\delta})$. The statement $BPP \subset SUBEXP$ (read "randomness is weak"), while falling short of $P = BPP$, would be a great result to prove unconditionally, and it certainly implies $BPP \neq EXP$. There have been a sequence of papers getting weaker and weaker hardness assumptions sufficient to prove such a result. These papers use one of three basic methods for converting hard functions into pseudo-random sequences: the "cryptographically secure" (BMY-type) pseudo-random generator based on one-way functions [6, 24, 16, 7, 8, 11]; the NW-generator based on a Boolean function with no circuit that approximates it [21, 3]; and the hitting set method [1, 2].

To state our results, we will need some notation for complexity classes. Let $Size(T(n))$ be the class of functions computable by circuit families where the number of gates in the circuit with n inputs is at most $T(n)$. For C a complexity class and $t(n)$ a function, let $C/t(n)$ be the class of functions computable in C with $t(n)$ bits of "advice" depending only on the input size, i.e., $f \in C/t(n) \iff \exists g \in C$ and a function $h : Z \to Z$ with $|h(n)| \leq t(n)$ and $f(x) = g(x, h(|x|))$. A result of [15] shows that $P/poly = \cup_{c\geq 1} Size(n^c)$. For C a complexity class, let $i.o.-C$ be the class of functions that agree with a function in C for all inputs of length n for infinitely many n.

The first set of papers construct a pseudo-random generator from a one-way function. The pseudo-random generator quickly converts a small random string to a polynomially larger string that seems random in the following sense: Any adversary that can distinguish an output of this generator from a truly random string of the same length can be used to invert the function. A BPP algorithm that had a markedly different behaviour on a pseudo-random input than a random one would be such an adversary. So if no such invertor exists, the deterministic algorithm that enumerates the multi-set of outputs of the generator and simulates the BPP algorithm on each, taking the majority answer, would always be correct. Informally, this is stated as:

Theorem A 1 *[6, 24, 8, 7, 11]*
If there are one-way functions that cannot be inverted with a non-negligible probability in $P/poly$, then $BPP \subset SUBEXP$

The NW-generator [20, 21, 3] considerably weakened the hardness assumption needed in the nonuniform setting. It achieves the same deterministic simulation of BPP, from any function in $EXP - P/poly$.

Theorem A 2 *[20, 21, 3]*
If $EXP \not\subset P/poly$, then $BPP \subset i.o.-SUBEXP$

This was the best result known at the "low end" of the hardness vs. randomness curve.

There has also been a sequence of papers [21, 1, 2, 14] at the "high end" of the curve, where the desired goal is to obtain $P = BPP$ under the weakest possible assumption. The strongest result in this sequence is stated below:

Theorem A 3 *[14]*
If $EXP \not\subset i.o. - SIZE(2^{o(n)})$, then $BPP = P$.

1.3 Average-Case Derandomization Under Uniform Assumptions

Both the high-end and low-end results above require non-uniform hardness assumptions, i.e., that the hard problems in question are hard for circuits. The reason for this, intuitively, is that if a function is hard uniformly but easy non-uniformly, there is some advice, or trap-door, that makes computing the function easy. Even if this trap-door is hard to find, there is no way to guarantee that a rare instance of the BPP problem does not code this trap-door information. Thus, it seems difficult to obtain a worst-case guarantee for the simulation.

However, we might still be able to get an "average-case" simulation under a uniform assumption. In fact, what such a result would say is that it is infeasible to find inputs for which such a simulation fails. We need to be somewhat careful in defining average-case complexity of a problem. We actually want to examine the difficulty of the problem for "most instances" rather than the average of the difficulties. We also should say what kinds of errors the algorithm is allowed to make on the exceptional instances. An algorithm allowed to output mistakes on a small number of inputs will be called a "heuristic" for the problem, whereas an algorithm that simply fails to give an answer in the alloted time will be called an algorithm with a certain average-case performance. Here, we'll use somewhat ad hoc definitions of these concepts that have certain technical advantages for our setting, especially when making statements about "infinitely many" sizes..

Definition 1 *A* probability ensemble $\mu = \{\mu_n | n \in Z^+\}$. *is a sequence of probability distributions on the set of strings of length n. The ensemble μ is* polynomially sampleable *if there is a polynomial p and a polynomial time computable function M so that if $R \in_U \{0,1\}^{p(n)}$, then $M(n, R)$ is distributed according to μ_n. As usual, we can extend this notion to allow M access to an oracle, in which case we say μ is polynomially sampleable given the oracle.*

Let T and ϵ be functions of n. $HeurTIME_{\epsilon(n)}(T(n))$ is the class of pairs (f, μ) of functions $f : \{0,1\}^* \rightarrow \{0,1\}$ and probability ensembles μ so that there is an algorithm $A(x)$ running in deterministic time $T(|x|)$ so that $\forall n$, for $x \in_{\mu_n} \{0,1\}^n$, $Prob[A(x) \neq f(x)] < \epsilon(n)$.

$AvgTIME_{\epsilon(n)}(T(n))$ is the class of pairs (f, μ) of functions $f : \{0,1\}^* \rightarrow \{0,1\}$ and probability ensembles μ so that there is an algorithm $A(x)$ running in deterministic time $T(|x|)$ so that $A(x) \in \{f(x), ?\}$ and for all n, for $x \in_{\mu_n} \{0,1\}^n$, $Prob[A(x) \neq f(x)] < \epsilon(n)$.

If membership in one of the above classes holds for f together with any polynomially sampleable ensemble, then we omit mention of the ensemble and simply say that f is in the class. The same abuse of notation will be used for complexity classes being subsets of the above classes.

Under cryptographic assumptions, the standard techniques give "average-case" simulations. From uniformly one-way functions, we can generate pseudorandom sequences that are hard for probabilistic algorithms, rather than circuits, to distinguish. An informal statement of the resulting derandomization is:

Theorem A 4 *[6, 24, 8, 7, 11]*
If there are uniformly one-way functions, then for every $c > 0$, $BPP \subset Heur_{1/n^c}SUBEXP$ and $ZPP \subset Avg_{1/n^c}SUBEXP$.

1.4 Our Results

The main result of this paper is a version of this theorem based on the much weaker assumption $BPP \neq EXP$.

Theorem 5 *If $BPP \neq EXP$ then for every $c > 0$, $BPP \subset i.o. - Heur_{1/n^c}SUBEXP$ and $ZPP \subset i.o. - Avg_{1/n^c}SUBEXP$.*

We also give a sharp converse:

Theorem 6 *There are functions in $EXP \cap P/poly$ that are not in $i.o. - HeurTIME_{2/3}(2^{o(n)})/o(n)$.*

Corollary 7 *If* $BPP \subseteq i.o. - HeurTIME_{2/3}(2^{o(n)})/o(n)$*, then* $BPP \neq EXP$*. And so* $BPP \subset i.o. - Heur_{1/n^c}SUBEXP$

Corollary 8 *If* $BPP = EXP \cap P/poly$ *then* $BPP = EXP$*.*

Summarizing, we get:

Corollary 9 *Exactly one of the following holds:*
(1) $BPP = EXP$
(2) $BPP \subseteq i.o.-HeurTIME_{1/n^c}(2^{n^\delta})$ *for every* $\delta > 0$ *and* $c > 0$*.*

This result is naturally interpreted as a gap theorem on derandomization - either no derandomization of BPP is possible at all, or otherwise a highly nontrivial derandomization is possible. A more precise statement of the gap is:

Corollary 10 *If* $BPP \subseteq i.o. - HeurTime_{1/3}(2^{o(n)})/o(n)$ *then* $BPP \subseteq i.o. - HeurTIME_{1/n^c}(2^{n^\delta})$ *for every* $\delta > 0$ *and* $c > 0$*.*

1.5 Why wasn't this paper written in 1988?

The rest of this section describes intuitively the obstacles to obtaining this result long ago, and the key ideas we use to overcome them. Attempts to find a uniform version of the above result (namely, replacing $P/poly$ by BPP in the hardness assumption) followed immediately after its discovery in 1988, both by the authors and many others. However, the following presented a psychological barrier:

The proofs of the before-mentioned theorems have the following structure. A (presumably hard) function f is used to construct a (hopefully pseudorandom) generator G. Equivalently, one starts with a hypothetical distinguisher for G, and constructs from it an efficient algorithm for f, obtaining a contradiction. In the nonuniform versions the distinguisher and algorithm are circuits, and only an existence proof of the later from the former is required. In the uniform case, both are probabilistic Turing machines.

The construction of an algorithm for f from a distinguisher of G, follows a sequence of steps. Various steps in this construction seem to inherently need values of f at many (often random) points. While this is easy if f was the inverse of a one-way function or nonuniformly (hard-wire these values), such values seem impossible to obtain uniformly for an arbitrary function in EXP.

Here we take a careful look at the steps mentioned above. It was already known that for some of them (Random Self Reducibility [19, 4, 5], Hard Core Bit theorem [8]) the circuit construction is already uniform. More importantly, in the other steps (the XOR Lemma [24], the generator conversion [20, 21]), function values of f are the *only* nonuniform construct needed. Thus, the first key idea is allowing our PPT (Probabilistic Polynomial Time) algorithm to have an oracle for f.

At first sight it seems ridiculous to give an algorithm trying to compute f an access to an oracle for f. However, we don't merely want to compute f on a single input, but to construct a circuit computing f on all inputs. This, one could state the issue of whether such a construction exists as whether f is *learnable from examples* in the sense of computational learning theory. We show that a distinguisher for the pseudo-random generator can be used to learn how to compute the hard function from examples.

So we get the following (informal) partial result: if an NW-generator based on any $f \in EXP$ is not pseudo-random, then a circuit for f can be constructed in PPT^f.

We still need to convert this into a construction of such a circuit with no oracle calls. This is not trivial. However, a crucial observation is that in the construction above the use of the oracle is limited in the following way: it is never called on larger input lengths than those of the circuit it constructs. How can this help to eliminate the oracle? The next key idea is assuming (for no good reason so far) that f happens to be downward self reducible (like SAT or PERMANENT). In such cases observe that the oracle for f is redundant: to construct a circuit for f_n, simply use the above PPT algorithm, and whenever it calls the oracle, use the downward self reduction and the inductively constructed circuits of smaller sizes.

It remains to justify the assumption that f is downward self reducible. This seems a strange assumption, since downward self-reducible problems are always in $PSPACE$ and our f is supposed to be complete for EXP. However, we have an advantage: we know that the only way the NW generator fails is if $EXP \subseteq P/poly$, and then it follows from [15] and [23] that $EXP = \Sigma^2 = P^{\#P}$. [1]

So if the NW generator fails with a standard EXP-complete problem, we try again with a downward self-reducible $\#P$-complete problem (which we then know is also EXP-complete). If the simulation still fails, we can use it to solve the $\#P$ complete problem, and hence any problem in EXP. We will pick f to be our favorite random-self-reducible and downward self-reducible function complete for $\#P$, a variation of the permanent function.

2 Proof of Theorem 5

2.1 Overview

We want to show that derandomization is possible given $BPP \neq EXP$. As mentioned above , since [21] show that $BPP \subseteq i.o.-SUBEXP$ assuming $EXP \not\subseteq P/poly$, we can assume $EXP \subset P/poly$.

Furthermore, [15] observed that if this is true, then $EXP = \Sigma^2$. The proof of this can be sketched as follows. Let f be the function that, given a Turing Machine that runs in time 2^n, an input of length n and the name of a cell in the tableau for the machine, outputs the contents of that cell. f is EXP-complete. Given a circuit C, the question "Does C compute f on all inputs of length n?" is in Co-NP, since if not, one can exhibit an input on which the circuit fails, and a previous inconsistent finite block in the tableau. Thus, if small circuits for f exist, one can non-deterministically guess one, and then co-non-deterministically verify it. So if $f \in P/poly$, then $f \in \Sigma^2$. [2]

By Toda's Theorem [23], then $EXP = P^{\#P}$ and so the permament function is complete for EXP. ([23]). Thus, we can assume that computing the permanent is not possible in BPP. The permanent has two nice properties that we will use. First, it is random self-reducible ([19]), so its average-case and worst-case difficulty for BPP are equivalent. Secondly, one can solve the permanent in polynomial time using an oracle for the permanent of smaller matrices. We call this property downward self-reducibility. We will assume we have a function $f \notin BPP$ with these properties.

Let f_n be f restricted to inputs of size n. For each input size n, we will construct a pseudo-random generator G_n from n^c bits for some fixed constant c, to n^d bits for an arbitrary $d > c$, that will be computable in polynomial time with an oracle for f_n. Given a circuit that distinguishes the output of this generator from truly random strings, we will be able to construct a circuit computing f on n bit strings, in polynomial-time *with an oracle for* f_n .

The simulation of a BPP algorithm is as follows. Let $\delta > 0$ be given. On inputs of size k, assume the BPP algorithm uses k^{c_1} random bits and time. Set $n = k^{\delta/2c}$. Using $d = 2cc_1/\delta$, we construct the range of G_n, a set of $n^d = k^{c_1}$ bit strings, in time $2^{O(n^c)} = O(2^{k^\delta})$. We then simulate the BPP algorithm on each element of the range and take the majority vote.

Assume the above algorithm is incorrect with probability $1/k^d$ with respect to some sampleable distribution μ_k on k-bit strings, for all but finitely many k. Then given n, we can set $k = n^{2c/\delta}$ and through random sampling μ_k, find instances $x_1, ...x_{k^{O(1)}}$ in probabilistic polynomial-time so that the with high probability the algorithm fails on at least one x_i. Translating the behaviour of the BPP algorithm on x_i into a circuit, we get a polynomial-time probabilistic construction which produces a collection $D_1, ..D_r$ so that at least one D_i distinguishes outputs of G_n from truly random strings.

Working through the constructions from [21, 3], we show that from such a distinguisher we can construct a circuit for f_n in probabilistic polynomial-time *using an oracle for* f_n. We then get out of this Catch 22 by using a bootstrapping argument as follows. Assume we have a circuit C_{n-1} for f_{n-1}. Since f is downward self-reducible, we can simulate an oracle for f_n using C_{n-1}. We construct a set $D_1, ..D_r$ as above that contains a distinguisher for G_n. We can find this distinguisher, since we can use our oracle for f_n to sample from the range of G_n and hence estimate the distinguishing probability. We then use

[1]The result from [15] does not relativize ([13]), which suggests that similar methods might lead to non-relativizing separations. However, we are unsure whether our main result relativizes.

[2][3] get the stronger result, If $EXP \subseteq P/poly$ then $EXP = MA$.

it to construct a circuit C_n computing f_n.

The key observation is that the size of C_n does not depend on the size of C_{n-1}, since C_{n-1} was only used as an oracle. So unlike the situation if we used the downward reduction directly in such a construction, we do not get an exponential blow-up in size.

2.2 Reductions between construction problems

Most of the algorithms we'll use will be probabilistic polynomial time algorithms whose inputs and outputs will be encodings of Boolean circuits. We'll be interested in statements, "If one can construct a circuit with property X , then from it one can construct a circuit with property Y". Because to formalize such a statement requires some quantification, it is helpful to have some general notation for such reductions. For our paper, one can think of each construction problem as specifying a type of circuit, and one can think of n as the number of inputs to these circuits.

Definition 2 *A* construction problem $A = \{A_n\}$ *is a family of non-empty subsets* $A_n \subseteq \{0,1\}^*$. *(Note that no upper bound is put on the sizes of members of* A_n *in terms of* n*.)*

In the below definitions, the probabilities are taken over the uniform distribution on n bit strings.

Definition 3 Important Construction Problems

Circuits approximating f **Let $f : \{0,1\}^* \to \{0,1\}^*$ and $\epsilon : N \to [0,1]$. Define the construction problem $C^{f,\epsilon}$ by $C_n^{f,\epsilon}$ contains all circuits C with n inputs satisfying $Pr[C(x) = f(x)] \geq \epsilon(n)$.**

Circuits computing f **$C^f = C^{f,1}$.**

Distinguishers **Let $m : N \to N$, $G = \{G_n : \{0,1\}^{m(n)} \to \{0,1\}^n$, and ϵ as before. Define the construction problem $D^{G,\epsilon}$ by $D_n^{G,\epsilon}$ contains all circuits D with n inputs satisfying $Pr[D(G(y) = 1] - Pr[D(x) = 1] \geq \epsilon$.**

Definition 4 *Let A and B be construction problems.*

A strong construction *for A is a probabilistic function $f(n, \alpha)$ so that $\forall n \geq 1, \alpha > 0$, $Prob(f(n, \alpha) \in A_n) \geq 1 - \alpha$, where the probability is over random choices made by f. A* weak construction *for A is a probabilistic function $f(n)$ and a constant $c > 0$ with $Prob(f(n) \in A_n) \geq n^{-c}$ for all $n \geq 1$. A is* weak/strong probabilistic polynomial time constructible *if there is a weak/strong construction f for A which runs in time polynomial in n or n and $1/\alpha$, respectively.*

A construction of B from A *is a probabilistic function $f(x, \alpha)$ so that for every $n, \alpha > 0, a \in A_n$, $Prob[f(a, \alpha) \in B_n] \geq 1 - \alpha$. B is* probabilistic polynomial time constructible *from A, written $A \to B$, if there is a construction f of B from A which runs in time polynomial in n and $1/\alpha$.*

The following definition of random self-reducibility is slightly non-standard, but is clearly implied by all the usual definitions:

Definition 5 *A function $f : \{0,1\}^* \to \{0,1\}^*$ is* randomly self reducible (RSR) *if $(C^{f,1-n^{-c}} \to C^f)$ for some $c \geq 0$.*

As usual, we can extend this definition by allowing the function f access to an oracle, O, which we will write $A \to^O B$. If, in addition, the queries made by the construction are all of binary length n, we write $A \to^{O_n} B$.

We also adopt the same length restriction for the familiar efficient Turing reduction among functions.

Definition 6 *Let $f, g : \{0,1\}^* \to \{0,1\}^*$, and let $\ell : Z^+ \to Z^+$. We say that f is* polynomial Turing reducible to g restricted to length $\ell(n)$, *and write $f_n \leq_{pT} g_{\ell(n)}$, if there is a deterministic*[3] *polynomial time oracle machine M^g that on every input x outputs $f(x)$ and queries the oracle g only on inputs of length at most $\ell(|x|)$. f is* downward self-reducible *if $f_n \leq_{pT} f_{n-1}$.*

Let $ModPerm$ be the following decision problem:

Instance: An integer k in unary, a prime $p > 2k$ in unary, a $k \times k$ matrix M of integers modulo p, and an integer t modulo p.

Problem: Is $Perm(M) mod p = t$?, where $Perm$ is the usual permanent function.

[3] A probabilistic version can be given and used as well, but we shall not need it

Note that it is easy to generate random valid instances of $ModPerm$ of a given length, and to place them in one-to-one correspondence with integers up to the number of valid instances. So without loss of generality, by "uniform" distribution on length n strings for $ModPerm$, we mean uniform on the valid instances of length n. $ModPerm$ is downward self-reducible by the usual method of computing permanent via minors. By using the Chinese Remainder Theorem, it is easy to see that computing the permanent for an arbitrary matrix of integers reduces to $ModPerm$, so it is complete for $\#P$. The self-reducibility for permanent modulo a prime $> 2k$ due to Lipton [19] (using the method of [4]) shows that $ModPerm$ is random self-reducible according to the above definition. In the sequel, the reader can think of f as $ModPerm$, although we prefer to state things in more general terms.

Lemma 11 *There is a $\#P$ complete decision problem that is downward self-reducible and random self-reducible.*

Now we can sketch the outline of the proof in terms of the definitions above. Let f be a random self-reducible and downward self-reducible function, like $ModPerm$. We will define in the next subsection the NW-generator G_f. Assuming that G_f is not pseudo-random, we will conclude that $f \in BPP$.

The sketch is as follows:

Lemma 12 $G_{f_n} \leq_{pT} f_n$.

This will be immediate from the construction of G, given in the next sub-section.

Lemma 13 *If $BPP \not\subseteq i.o. - HeurTime_{1/n^c}(2^{n^\delta})$ for some $c, \delta > 0$, then $D^{G_f, 1/4}$ is weakly probabilistic polynomial-time constructible.*

The proof was sketched before. Assume the simulation based on computing the range of G_f and taking the BPP algorithm on this range fails with probability $1/n^c$ for all but finitely many n. Then we can sample a random instance and use the corresponding circuit as our distinguisher.

Lemma 14 *If $D^{G_f, 1/4}$ is weakly probabilistic polynomial-time constructible, then $D^{G_f, 1/5}$ is strongly probabilistic polynomial-time constructible using oracle f_n*

This follows from the previous two lemmas. We can repeatedly sample from using the weak construction. For each circuit we construct, we can estimate its distinguishing probability by sampling from the range of G_f using the oracle for f_n, since $G_f \leq_{pT} f_n$.

Lemma 15 *If f is random-self-reducible, then $D^{G_f, 1/5} \rightarrow^{f_n} C^f$*

This is the main technical lemma, and will be proved in sub-section 2.4. However, it really just examines the proofs of the known results.

Lemma 16 *If f is downward self-reducible, and C^f is strongly polynomial-time constructible using oracle f_n then $f \in BPP$*

Proof. We recursively compute circuits $C_1 \in C_1^f, ..C_n \in C_n^f$. We then output C_n evaluated at our input. Say that we have computed C_i. We run the construction for C_{i+1}^f with oracle f_{i+1} (with $\alpha = 1/n^2$), simulating queries to f_{i+1} by M^{C_i}, where M is the poly-time oracle Turing Machine from the definition of downward self-reducibility.

Note that $|C_{i+1}| \leq$(Time taken by the construction not counting oracle queries)*(Time taken to simulate queries *not counting the time to evaluate oracle calls by M*). This is a fixed polynomial in n, independent of the size of C_i. Since each $|C_i|$ is bounded by this fixed polynomial in n, the time for each stage (including time to evaluate oracle calls by M) is a fixed polynomial in n. Also, the probability that $C_n \notin C_n^f$ is at most $\alpha * n = 1/n$, so the error is bounded. ∎

Combining the above, we get:

Lemma 17 *If $BPP \not\subseteq i.o. - HeurTIME_{n^{-c}}(2^{n^\delta})$ then $f \in BPP$ for every downward and random self-reducible function f. In particular, $ModPerm \in BPP$ so $BPP = \#P$.*

As described earlier, this suffices to prove Theorem 5 for the case of BPP. (For ZPP we just need to note that the simulation is error-free, but may not find a halting computation. The rest is identical.)

2.3 Construction of G_f

We view the construction of G_f as a sequence of three steps, in order to make the proof more modular. The sequence we use below is not exactly the one used in [21, 3], but the original one would work as well.

Let $d > 0$ be an integer. Let $c > 0$ be the constant so that $C^f \rightarrow C^{1-n^{-c},f}$.

Direct product function Let $n_1 = n^{c+2}$. View an n_1 bit string as n^{c+1} n bit strings. Define $g : \{0,1\}^{n_1} \rightarrow \{0,1\}^{n^{c+1}}$ by $g(x_1, ..x_n) = f(x_1), ..f(x_n)$.

Hard-core bit Let $n_2 = n_1 + n^{c+1}$. View an n_2 bit string as an input to g x and a string r of length $|g(x)|$. Then $h(x,r) = < g(x), r >$, where $< y, z >$ represents inner product modulo 2.

Almost disjoint sets generator Let $m = n_2^2$. Let $z \in \{0,1\}^m$ and let $S = \{s_1 < s_2 < .. < s_{n_2}\}$ be a subset of bit positions between 1 to m. Then define $z|_S$ to be the n_2 bit string $z_{s_1} z_{s_2} ... z_{s_{n_2}}$. In [21], an explicit construction of ℓ such sets $S_1, ..S_\ell$ is given so that $|S_i \cap S_j| \leq \log_n \ell$ for every $i \neq j$. We define $G : \{0,1\}^m \rightarrow \{0,1\}^\ell$ by $G(z) = h(z|_{S_1}), h(z|_{S_2}), ...h(z|_{S_\ell})$.

From the construction, it is clear that $G_m \leq_{pT} h_{n_2} \leq_{pT} g_{n_1} \leq_{pT} f_n$, which proves Lemma 12.

2.4 Proof of Lemma 15

We work through the construction in reverse order, showing how to construct, from a distinguisher for G_m, a circuit for f_n. There will be four stages in this construction, the first three corresponding to the three levels of the definition of G and the last stage to the random self-reduction of f. All stages are identical to those from the non-uniform proofs, but we need to verify that the use of non-uniformity can be replaced by an oracle for f_n. We'll just review the constructions from other papers, to see that they are polytime computable from such an oracle, and refer to the relevant papers for proofs of correctness.

The four stages we need are given by the following lemmas:

Lemma 18 $D^{G,.2} \rightarrow^{h_{n_2}} C^{h,1/2+O(1/\ell)}$

Lemma 19 $C^{h,1/2+O(\ell^{-1})} \rightarrow C^{g,O(\ell^{-3})}$

Lemma 20 $C^{g,O(\ell^{-3})} \rightarrow^{f_n} C^{f,1-n^{-c}}$

Finally, by the definition of random self-reducibility $C^{f,1-n^{-c}} \rightarrow C^f$

Proof. Lemma 18: The construction is from [21]. Let $D \in D_m^{G,.2}$. We construct a circuit to predict h as follows: Pick $i \in_U \{1, ..\ell\}$. For each $1 \leq j \leq \ell$ with $j \notin S_i$, pick $z_j \in_U \{0,1\}$. For each $i' < i$, query h at all $2^{|S_i \cap S_j|} \leq \ell$ strings that might be $z|_{S_{i'}}$ for a z consistent with the z_j's. Store the answered queries in a table T. Pick $b_{i'} \in_U \{0,1\}$ for $i \leq i' \leq \ell$.

Let C be the following circuit: on input x , set $z|_{S_i} = x$, while the other bits of z are fixed to the randomly chosen bits. Set $b_{i'} = h(z|_{S_{i'}})$ for $i' < i$, by looking up the appropriate entry in T. If $D(b_1, ..b_\ell) = 1$ output b_i; else output $\neg b_i$.

By random sampling using the oracle for h_{n_2}, estimate the probability that $C(x) = h(x)$; if greater than $1/2 + .05/\ell$, output C, else repeat.

[21] show that the expected probability of success for C is at least $1/2 + .1/\ell$, so the number of repetitions before outputting a C that has good advantage is at most $O(n\ell)$ with very high probability. ∎

Lemma 19 follows directly from [8].

Proof. Lemma 20 is the uniform version of a direct product lemma which has many proofs [16, 9, 14]. We present here the construction from [14] as being simple to describe.

Let $C \in C_{n_1}^{g,\delta}$. Construct C' as follows: Let $n_3 = n_1/n = n^{c+1}$. Repeat for $r = 1$ to n^3/δ. Pick $i \in_U \{1, ..n_3\}$. For each $j \neq i$, pick $x_j \in_U \{0,1\}^n$, query $f(x_j)$ and record the answer. Flip coins until a head arises or until n tails have been flipped; let t_1 be the number of flips.

Let C'_r be the following three-valued circuit: On input x, compute t , the number of bit positions $j \neq i$ where the j'th bit of $C(x_1, ..x_{i-1}, x, x_{i+1}, ..x_{n_3})$ disagrees with $f(x_j)$ (as recorded.) If $t < t_1$ output the i'th bit of $C(x_1, ..x_{i-1}, x, x_{i+1}, ..x_{n_3})$; otherwise output "reject".

Let C' be the circuit that outputs the majority answer from those C'_r that do not reject. [14] prove that, for non-neglegible δ, $C' \in C^{f,1-n^{-c}}$ with high probability. If δ is at least inverse polynomial, the construction takes polynomial time. ∎

2.5 Proof of Theorem 6

Proof. We construct a problem in $E \cap P/poly$ that cannot be approximated in $TIME(2^{o(n)})/o(n)$ as follows. The idea is that some function from a universal family of hash functions will serve our purpose.

For any input of size n, we first simulate all machines with descriptions of size $.1n$ on all advice strings of size $.1n$ for 2^n steps on all inputs. If they don't halt, record the answer as (say) 0. This gives us at most $2^{\cdot 2n}$ strings (truth tables) of $N = 2^n$ bits each. Let H be a family of $2^{O(n)}$ pairwise independent hash functions from n bits to 1 bit. (For example, $H = \{0,1\}^n, h_r(x) = < r, x >$). Cosider them too as N-bit strings. Using Chebyshev bounds, one can see that a random $h_r \in H$ agrees with a string in 2/3 N positions only with probability $1/O(N)$. Then since there are less than $O(N)$ strings in our collection, we can find a hash function that does not agree with any of them in 2/3 the bits. We pick this h_r and output $h_r(x)$. ∎

3 Conclusions and Open Problems

Ideally, we would hope that our results are a step towards proving $BPP \neq EXP$. However, our results provide reasons both to be optimistic and pessimistic about such a proof. On the one hand, our result makes such a result stronger, since it would show a positive simulation as well as a negative result. On the other, it clarifies that the best way of attacking this problem is to continue along the lines of the de-randomization papers. It also shows that non-relativizing techniques can be useful in this area, so we need not be depressed by oracles where $BPP = EXP$. It also indicates that we do not need to prove circuit lower bounds to get such a result, so we should also be undaunted by the negative results on Natural Proofs.

There are some more technical points our work raises. First, is an average-case derandomization all one can hope for under a uniform assumption? If $BPP \neq EXP$ but $EXP \subseteq P/poly$, we have a paradoxical situation: the simulation of randomness by determinism does not always work, but it is intractable to find instances where it fails. Can we somehow utilize this intractibility in yet another layer of hardness vs. randomness tradeoffs?

As can be seen, our main result is achieved essentially with no technical work. All that is taken from previous papers. On the other hand these papers are viewed from a somewhat different perspective in trying to make them uniform, which is subtle in some ways and raises some new questions regarding these issues.

The first one is that the classical "learning from a membership oracle" problem of computational learning theory arises naturally here. Let $LEARN$ be the class functions for which this can be done efficiently, namely all f for which C_n^f is constructible in PPT^{f_n}. This class is quite interesting, and we trivially have:

Fact 21 $BPP \subseteq LEARN \subseteq P/poly$

We also showed that any downward self-reducible problem in $LEARN$ is also in BPP. What more can be said about the class $LEARN$?

Finally, we should mention that gaps similar to the one obtained here are possible at higher levels of the polynomial hierarchy, such as for whether $MA \neq NEXP$.

References

[1] A. Andreev, A. Clementi and J. Rolim, "Hitting Sets Derandomize BPP", in *XXIII International Colloquium on Algorithms, Logic and Programming (ICALP'96)*, 1996.

[2] A. Andreev, A. Clementi, and J. Rolim, "Hitting Properties of Hard Boolean Operators and its Consequences on BPP", manuscript, 1996.

[3] L. Babai, L. Fortnow, N. Nisan and A. Wigderson, "BPP has Subexponential Time Simulations unless EXPTIME has Publishable Proofs", *Complexity Theory,* Vol 3, pp. 307–318, 1993.

[4] D. Beaver and J. Feigenbaum, "Hiding Instance in Multioracle Queries", *Proc 7th Symposium on TheoreticalAspects of Computer Science, LNCS 415*, pp. 37–48, 1990.

[5] L. Babai, L. Fortnow, C. Lund, "Non-deterministic exponential time has two-prover interactive protocols", in *31st FOCS*, pp. 16–25, 1990.

[6] M. Blum and S. Micali. "How to Generate Cryptographically Strong Sequences of Pseudo-Random Bits", *SIAM J. Comput.*, Vol. 13, pages 850–864, 1984.

[7] O. Goldreich, H. Krawcyk, M. Luby, "On the existence of pseudorandom generators", in *29th FOCS*, pp. 12–24, 1988.

[8] O. Goldreich and L.A. Levin. "A Hard-Core Predicate for all One-Way Functions", in *ACM Symp. on Theory of Computing*, pp. 25–32, 1989.

[9] O. Goldreich, N. Nisan and A. Wigderson. "On Yao's XOR-Lemma", available via www at ECCC TR95-050, 1995.

[10] S. Goldwasser and S. Micali. "Probabilistic Encryption", *JCSS*, Vol. 28, pages 270–299, 1984.

[11] J. Hastad, R. Impagliazzo, L.A. Levin and M. Luby, "Construction of Pseudorandom Generator from any One-Way Function", to appear in *SICOMP*. (See preliminary versions by Impagliazzo et. al. in *21st STOC* and Hastad in *22nd STOC*.)

[12] R. Impagliazzo, "Hard-core Distributions for Somewhat Hard Problems", in *36th FOCS*, pages 538–545, 1995.

[13] R. Impagliazzo, In preparation.

[14] R. Impagliazzo and A. Wigderson, "P=BPP unless E has sub-exponential circuits: Derandomizing the XOR Lemma", Proc. of the *29th STOC*, pp. 220–229, 1997.

[15] R. M. Karp and R. J. Lipton, "Turing Machines that Take Advice", *L'Ensignment Mathematique*, 28, pp. 191–209, 1982.

[16] L. A. Levin, "One-Way Functions and Pseudorandom Generators", *Combinatorica*, Vol. 7, No. 4, pp. 357–363, 1987.

[17] M. Luby, *Pseudorandomness and Cryptographic Applications*, Princeton Computer Science Notes, Princeton University Press, 1996.

[18] L.A. Levin, "Average Case Complete Problems", *SIAM J. Comput.*, 15:285–286, 1986; also *STOC*, 1984.

[19] R. Lipton, "New directions in testing", In J. Fegenbaum and M. Merritt, editors, *Distributed Computing and Cryptography*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science Volume 2, pp. 191-202. American Mathematical Society, 1991.

[20] N. Nisan, "Pseudo-random bits for constant depth circuits", *Combinatorica 11* (1), pp. 63-70, 1991.

[21] N. Nisan, and A. Wigderson, "Hardness vs Randomness", *J. Comput. System Sci.* 49, 149-167, 1994

[22] A. Shamir, "On the generation of cryptographically strong pseudo-random sequences", *8th ICALP, Lecture Notoes in Computer Science* 62, Springer-Verlag, pp. 544–550, 1981.

[23] , S. Toda, "On the computational power of PP and $\oplus P$", in *30th FOCS*, pp. 514–519, 1989.

[24] A.C. Yao, "Theory and Application of Trapdoor Functions", in *23st FOCS*, pages 80–91, 1982.