Simplified derandomization of BPP using a hitting set generator*

Oded Goldreich† Salil Vadhan‡ Avi Wigderson§

January 14, 2000

Abstract

A hitting-set generator is a deterministic algorithm which generates a set of strings that intersects every dense set recognizable by a small circuit. A polynomial time hitting-set generator readily implies $\mathcal{RP} = \mathcal{P}$. Andreev *et. al.* (ICALP'96, and JACM 1998) showed that if polynomial-time hitting-set generator in fact implies the much stronger conclusion $\mathcal{BPP} = \mathcal{P}$. We simplify and improve their (and later) constructions.

Keywords: Derandomization, $\mathcal{RP}$, $\mathcal{BPP}$, one-sided error versus two-sided error

---

*A preliminary version of this work has appeared in the proceedings of *Random99*.

†Department of Computer Science, Weizmann Institute of Science, Rehovot, Israel. `oded@wisdom.weizmann.ac.il`.

‡MIT Laboratory for Computer Science, 545 Technology Square, Cambridge, MA 02139. `salil@theory.lcs.mit.edu`. Supported by an NSF Mathematical Sciences Postdoctoral Research Fellowship.

§Institute of Computer Science, The Hebrew University of Jerusalem, Givat-Ram, Jerusalem, Israel. `avi@cs.huji.ac.il`.

1 Introduction

The relation between randomized computations with one-sided error and randomized computations with two-sided error is one of the most interesting questions in the area. Specifically, we refer to the relation betwen $\mathcal{RP}$ and $\mathcal{BPP}$. In particular, does $\mathcal{RP} = \mathcal{P}$ imply $\mathcal{BPP} = \mathcal{P}$?

The breakthrough paper of Andreev *et. al.* [1] (and its sequel [2]) gave a natural setting in which the answer is YES. The setting is a specific natural way to prove $\mathcal{RP} = \mathcal{P}$, namely via "hitting-set generators" (see exact definition below). Informally, such a generator outputs a set of strings that hits every large efficiently-recognizable set (e.g., the witness set of a positive input of an $\mathcal{RP}$ language). Having such a generator which runs in polynomial time enables a trivial deterministic simulation of an $\mathcal{RP}$ algorithm by using each of the generator's outputs as the random pad of the given algorithm.

The main result of [1] was that such a generator for 1-sided error algorithms already suffices to derandomize 2-sided error algorithms: the existence of polynomial-time hitting set generators implies $\mathcal{BPP} = \mathcal{P}$.

Definition 1 (hitting set generator): *An algorithm,* G*, is called a* hitting set generator for circuits *if for every $n, s \in \mathsf{N}$ (given in unary) generates as output a set of n-bit strings $G(n, s)$ with the following property: every circuit of size s on n input bits, which accepts at least half its inputs, accepts at least one element from the set* $\mathrm{G}(n, s)$.[1]

Since s is the essential complexity parameter ($n \leq s$), we let $\mathsf{t}_{\mathrm{G}}(s)$ denote the running time of the generator G on input (n, s), and $\mathsf{N}_{\mathrm{G}}(s)$ denote the size of its output set. Clearly $\mathsf{N}_{\mathrm{G}}(s) \leq \mathsf{t}_{\mathrm{G}}(s)$. The result of Andreev *et. al.* [1] is

Theorem 2 [1]: *If there exists a hitting-set generator G running in time* t_{G} *then* $\mathcal{BPP} \subseteq \mathcal{DTIME}(\mathrm{poly}(\mathsf{t}_{\mathrm{G}}(\mathrm{poly}(n)))$.

With the most important special case (i.e., $\mathsf{t}_{\mathrm{G}}(s) = \mathrm{poly}(s)$)

Corollary 3 [1]: *If G runs in polynomial time then $\mathcal{BPP} = \mathcal{P}$.*

Our main result is a simple proof of Theorem 2. To explain what simple means is not so simple, and we have to explain how the given generator assumed in the theorem is used to enable the derandomization of $\mathcal{BPP}$, in the proof of [1] and in later proofs. Indeed later proofs (of [2] and then [3]) were much simpler, but while proving Corollary 3, they fell short of proving Theorem 2.[2]

The reader is warned that the following discussion is on an intuitive level and some things cannot easily be made precise. The reader who doesn't like such discussions is welcome to skip to the formal proof in the next two sections.

The proof in [1] uses the generator in two ways. Once, literally as a producer of a hitting set for all large efficient sets. Second, and more subtly, as a hard function. Observe that the existence of such a generator G immediately implies the existence of a function on $O(\log \mathsf{t}_{\mathrm{G}}(s))$ bits which

[1] Usually generators are defined to output only one string; in terms of the above definition it means that on input an index $i \in \{1, ..., |\mathrm{G}(n, s)|\}$, the generator outputs the i^{th} string in $\mathrm{G}(n, s)$. However, we find the current convention simpler to work with in the current context.

[2] However, both [2] and [3] use their techniques to study the relationship between one-sided and two-sided error in additional respects not addressed by Theorem 2. In particular, [3] resolve the promise-problem analogue of the question "Does $\mathcal{RP} = \mathcal{P}$ imply $\mathcal{BPP} = \mathcal{P}$?" in the positive. See the discussion at the end of this section.

is computable in time $t_G(s)$ but cannot be computed by circuits of size s. These two ways are combined in a rather involved way for the derandomization of $\mathcal{BPP}$.

It is interesting to note that for the case $t_G(s) = \text{poly}(s)$, the resulting hard function mentioned above can be plugged into the pseudorandom generator of [6], to yield $\mathcal{BPP} = \mathcal{P}$ as in Corollary 3. However, [6] was unavailable to the authors of [1] at the time (the two papers are independent). Moreover, [6] is far from "simple", it does use the computational consequence which we are trying to avoid, and anyway it is not strong enough to yield Theorem 2.

A considerably simpler proof was given in [2]. There the generator is used only in its "original capacity", as a hitting set generator, without explicitly using any computational consequence of its existence. In some sense, this proof is more clearly a "black-box" use of the output set of the generator. However, something was lost. The running time of the derandomization is replaced by $\text{poly}(t_G(t_G(\text{poly}(n)))$.

On the one hand, this is not too bad. For the interesting case of $t_G(s) = \text{poly}(s)$ (which implies $\mathcal{RP} = \mathcal{P}$), they still get the consequence $\mathcal{BPP} = \mathcal{P}$, as in Corollary 3 (since iterating a polynomial function twice results in a polynomial). On the other hand, if the function t_G grows moderately so that $t_G(t_G(n)) = 2^n$, then we have as assumption a highly nontrivial derandomization of $\mathcal{RP}$, but the consequence is a completely trivial derandomization of $\mathcal{BPP}$.

The best (to our taste) way to understand the origin of the iterated application of the function t_G in the result above, is explained in the recent paper [3], which further simplifies the proof of [2]. They remind the reader that Sipser's proof [8] putting $\mathcal{BPP}$ in $\Sigma^2 \cap \Pi^2$ actually gives much more. In fact, viewed appropriately, it almost begs (with hindsight) the use of hitting sets!

The key is, that in both the $\forall\exists$ and $\exists\forall$ expressions for the $\mathcal{BPP}$ language, the "witnesses" for the existential quantifier are abundant. Put differently, $\mathcal{BPP} \subseteq \mathcal{RP}^{\text{pr}\mathcal{RP}}$, (where pr$\mathcal{RP}$ is the promise-problem version of $\mathcal{RP}$). But if you have a hitting set, you can use it first to derandomize the "oracle" part of the right-hand side. This leaves us with an $\mathcal{RTIME}(t_G(\text{poly}(n))$ machine, which can again be derandomized (using hitting sets for $t_G(\text{poly}(n))$ size circuits).

In short, the "two quantifier" representation of $\mathcal{BPP}$, leads to a two-level recursive application of the generator. It seems hopeless to reduce the number of quantifiers to one in Sipser's result. So another route has to be taken to prove Theorem 2 in a similar "direct" (or "black-box") manner as above, without incurring the penalty arising from this two level recursion.

We eliminate the recursion to have only one-level use of the hitting set, by "increasing the dimension to two": Following Lautemann's proof [7] of Sipser's result, for each input to a given $\mathcal{BPP}$ algorithm which uses $\ell(n)$ random coins, we consider a $2^{\ell(n)} \times 2^{\ell(n)}$ matrix whose (a, b)'th entry is the decision of the algorithm using random pad $a \oplus b$.[3] In this matrix, the fraction of incorrect answers in each row (resp., column) is small. The hitting set is used to select a small subset of the rows and a small subset of the columns, and the entries of this submatrix determine the result. Specifically we will look for "enough" (yet few) rows which are monochromatic, and decide accordingly. The correctness and efficiency of the test are spelled out in Lemma 6. It is essentially captured by the following simple Ramsey-type result, which is seemingly new and may be of independent interest.

Proposition 4 *For every n-vertex graph, either the graph or its complement has a dominating set of size $\lceil \log_2 n \rceil$. Furthermore, one can find such a set in polynomial time.*

[3] A preliminary version of this work [5] considered a different matrix whose (a, b)'th entry is the decision of the algorithm using random pad $a \circ b$. For that matrix to have the desired properties, it was necessary to first perform drastic error reduction (using extractors) on the $\mathcal{BPP}$ algorithm. The main simplification here is in avoiding this step.

We end by observing that (like the previous results) our result holds in the context of promise problems. Hence, the existence of hitting set generators provides an efficient way for approximately counting the fraction of inputs accepted by a given circuit within additive polynomial fraction. Formalizing this is standard and we leave it to the reader.

Perspective: As described above, Buhrman and Fortnow [3] prove that $\mathcal{BPP} \subseteq \mathrm{pr}\mathcal{RP}^{\mathrm{pr}\mathcal{RP}}$, and actually $\mathrm{pr}\mathcal{BPP} = \mathrm{pr}\mathcal{RP}^{\mathrm{pr}\mathcal{RP}}$. It follows immediately that $\mathrm{pr}\mathcal{RP} = \mathrm{pr}\mathcal{P} \Rightarrow \mathrm{pr}\mathcal{BPP} = \mathrm{pr}\mathcal{P}$, resolving the main question of this area for promise classes! This result suggests two natural extensions that remain open. The first is to obtain an analogue of their result for the standard language classes $\mathcal{RP}$ and $\mathcal{BPP}$. (In [3], it is shown that such an extension cannot relativize.) The second is to "scale" the result upwards. From the hypothesis $\mathrm{pr}\mathcal{RP} \subseteq \mathcal{DTIME}(t(n))$, they obtain the conclusion $\mathrm{pr}\mathcal{BPP} \subseteq \mathcal{DTIME}(\mathrm{poly}(t(t(\mathrm{poly}(n)))))$. Theorem 2, as proven in [1] and this paper, replaces the composition $t(t(\cdot))$ with a single $t(\cdot)$ for the (very) special case when the derandomization of $\mathrm{pr}\mathcal{RP}$ is via a hitting-set generator.

2 The Derandomization Procedure

Given $L \in \mathcal{BPP}$, consider a probabilistic polynomial-time algorithm A for L. Let $\ell = \ell(n)$ be a fixed polynomial denoting the number of coin tosses made by A on inputs of length n; similarly, define $s = s(n)$ so that the computation of A on inputs of length n can be implemented by circuits of size $s(n)$. We assume that A has error probability at most $1/2\ell(n)$; this can be achieved by straightforward amplification of any $\mathcal{BPP}$ algorithm for L.

Let $A(x, r)$ denote the output of algorithm A on input $x \in \{0,1\}^n$ and random-tape contents $r \in \{0,1\}^{\ell(n)}$, Our derandomization procedure, described below, utilizes a hitting-set generator H as defined earlier (cf., Def. 1).

Derandomization procedure: On input $x \in \{0,1\}^n$, letting A, ℓ, and s be as above.

1. Invoking the hitting-set generator G, obtain $H \leftarrow \mathrm{G}(\ell, \ell \cdot s)$. That is, H is a hitting set for circuits of size $\ell \cdot s$ and input length ℓ. Denote the elements of H by $e_1, ..., e_{\mathsf{N}}$, where $\mathsf{N} \stackrel{\mathrm{def}}{=} \mathsf{N}_{\mathrm{G}}(s)$ and each e_i is in $\{0,1\}^\ell$.

2. Construct an N-by-N matrix, $M = (v_{i,j})$, so that $v_{i,j} = A(x, e_i \oplus e_j)$. That is, run A with all possible random-pads composed by XORing each of the possible pairs of strings in H. (We merely use the fact that $a \oplus b$ is easy to compute and that for any a the mapping $b \mapsto a \oplus b$ is 1-1, and similarly for any b and $a \mapsto a \oplus b$.)

3. Using a procedure to be specified below, determine whether for every ℓ columns there exists a row on which all these columns have 1-value. If the procedure accepts then accept else reject. That is, accept if and only if

$$\forall c_1, ..., c_\ell \in [\mathsf{N}]\ \exists r \in [\mathsf{N}] \text{ s.t. } \wedge_{i=1}^{\ell} (v_{c_i,r} = 1) \tag{1}$$

We first show that if $x \in L$ then Eq. (1) holds, and analogously if $x \notin L$ then

$$\forall r_1, ..., r_\ell \in [\mathsf{N}]\ \exists c \in [\mathsf{N}] \text{ s.t. } \wedge_{i=1}^{\ell} (v_{r_i,c} = 0) \tag{2}$$

Note that this by itself does not establish the correctness of the procedure. Neither did we specify how to efficiently implement the procedure. To that end we use a general technical lemma which implies that it cannot be the case that both Eq. (1) and Eq. (2) hold, and in fact efficiently determines at least one which does not hold. These are deferred to the next section. But first we prove the above implications.

Proposition 5 *If* $x \in L$ (resp., $x \notin L$) *then* Eq. (1) (resp., Eq. (2)) *holds,*

Proof: We shall prove a slightly more general statement. Let χ_L be the characteristic function of L (i.e., $\chi_L(x) = 1$ if $x \in L$ and $\chi_L(x) = 0$ otherwise). Then we prove that for every $x \in \{0,1\}^n$, for every ℓ rows (resp., columns) there exists a column (resp., row) on which the value of the matrix is $\chi_L(x)$.

Fixing the input $x \in \{0,1\}^n$ to algorithm A, we consider the circuit C_x which takes an ℓ-bit input r and outputs $A(x,r)$ (i.e., evaluates A on input x and coins r). By our hypothesis regarding the error probability of A, we have

$$\mathrm{Pr}_{r\in\{0,1\}^\ell}[C_x(r) \neq \chi_L(x)] \ \le \ \frac{1}{2\ell}$$

It follows that for every $y_1, ..., y_\ell \in \{0,1\}^\ell$,

$$\mathrm{Pr}_{z\in\{0,1\}^\ell}[(\forall i)\ C_x(y_i \oplus z) = \chi_L(x)] \ge \frac{1}{2} \tag{3}$$

Let $\overline{y} = (y_1, ..., y_\ell)$, and consider the circuit $C_{x,\overline{y}}(z) \stackrel{\text{def}}{=} \wedge_{i=1}^{\ell}(C_x(y_i \oplus z) = \chi_L(x))$. Then, by the above $\mathrm{Pr}_z[C_{x,\overline{y}}(z) = \chi_L(x)] \ge 1/2$. On the other hand, the size of $C_{x,\overline{y}}$ is merely ℓ times the size of C_x, which was at most s. Thus, by definition of the hitting-set generator G, the set $H = \mathrm{G}(\ell\,,\, s)$ must contain a string z so that $C_{x,\overline{y}}(z) = \chi_L(x)$.

The above holds for any $\overline{y} = (y_1, ..., y_\ell)$. Thus, for every $y_1, ..., y_\ell \in H \subseteq \{0,1\}^\ell$ there exists $z \in H$ so that $A(x, y_i \oplus z) = C_x(y_i \oplus z) = \chi_L(x)$ for every $i \in [\ell]$. Thus we have proved that for every ℓ rows in M there exists a column on which the value of the matrix is $\chi_L(x)$.

A similar argument applies to sets of ℓ columns in M. Specifically, for every $z_1, ..., z_\ell \in \{0,1\}^\ell$

$$\mathrm{Pr}_{y\in\{0,1\}^\ell}[(\forall i)\ C_x(y \oplus z_i) = \chi_L(x)] \ \ge \ \frac{1}{2} \tag{4}$$

Again, we conclude that for every $z_1, ..., z_\ell \in H$ there exists $y \in H$ so that $C_x(y \oplus z_i) = \chi_L(x)$ for every $i \in [\ell]$. Thus, for every ℓ columns in M there exists a row on which the value of the matrix is $\chi_L(x)$. The proposition follows. ■

Perspective: The above procedure is a simplified version of the procedure given in a preliminary version of this work [5]. Specifically, the argument in [5] relies on explicit constructions of extractors for drastic error reduction of the $\mathcal{BPP}$ algorithm. Here, we only use a mild (and trivial) error reduction. The discrepency stems from the fact that the matrix considered in [5] is different. Step 3 in the above procedure is identical to the step in [5], and so is Lemma 6. Thus, our argument relies on two essential ingredients: The first ingredient, adopted from [2], is the use of auxiliary circuits (depending on C_x but not identical to it), in order to argue that a hitting-set must have certain strong properties with respect to C_x. The second ingredient is the constructive combinatorial result given by Lemma 6. (A third ingredient, using extractors as in [5], is eliminated here.)

3 Correctness and Efficiency of the Derandomization

Proposition 5 shows that for every x either Eq. (1) or Eq. (2) holds. But, as stated above, it is not even clear that Eq. (1) and Eq. (2) cannot hold simultaneously. This is asserted next.

Lemma 6 *Every n-by-n Boolean matrix, with $n \leq 2^k$, either has k rows whose OR is the all 1's row, or k columns whose AND is the all 0's column. Moreover, there is a (deterministic) polynomial-time algorithm that given such a matrix find such a set.*

We prove the lemma momentarily. But first let use show that Eq. (1) and Eq. (2) cannot hold simultaneously. We first note that in our case $n = \mathsf{N} = \mathsf{N}_{\mathrm{G}}(s)$ (which we may assume is at most 2^ℓ since it does not help for a hitting set to have repeated elements) and $k = \ell$. Then we just apply the following corollary.

Corollary 7 *For every n-by-n Boolean matrix, with $n \leq 2^k$, it is impossible that both*

1. *For every k rows there exists a column so that all the k rows have a 0-entry in this column.*
2. *For every k columns there exists a row so that all the k columns have a 1-entry in this row.*

Furthermore, assuming one of the above holds, we can decide which holds in (deterministic) polynomial-time.

Proof (of Corollary 7): Suppose Item (1) holds. Then, the OR of every k rows contains a 0-entry, and so cannot be the all 1's row. Likewise, if Item (2) holds then the AND of every k columns contains a 1-entry, and so cannot be the all 0's column. Thus, the case where both items holds stands in contradiction to Lemma 6. Furthermore, finding a set as in the lemma yields which of the two items does not hold. ■

Proof of Lemma 6: Let $S_0 = [n]$, $R = \emptyset$, and repeat for $i = 1, 2, \ldots$: Take a row j not in R which has at least $|S_i|/2$ 1's in S_i. Add j to R, and let S_{i+1} be the part of S_i that had 0's in row j. We get stuck if for any i, no row in current $[n] - R$ has at least $|S_i|/2$ 1's in S_i. Otherwise, we terminate when $S_i = \emptyset$

If we never get stuck, then we generated at most $\log_2 n \leq k$ rows whose OR is the all 1's row (as the i^{th} row has 1-entries in every column in $S_{i-1} - S_i$, and the last S_i is empty). On the other hand, if we got stuck at iteration i, let $S = S_i$. Note that every row has at least $|S|/2$ 0's in the columns S. (This includes the rows in the current R which have only 0's in the columns in $S \subset S_{i-1} \subset \cdots \subset S_0$.) But now picking greedily columns from S in sequence so as to contain the largest number of 0's in the remaining rows will clearly pick a 0 from every row after a set T of at most k columns from S were chosen.

Turning to the algorithmics, note that the above procedure for constructing R, S and T is implementable in polynomial-time. Thus, in case the "row" procedure was completed successfully, we may output the set of rows R, and otherwise the set T of columns. ■

Proof of Theorem 2: Proposition 5 shows that for every x either Eq. (1) or Eq. (2) holds, and furthermore that the former (resp., latter) holds whenever $x \in L$ (resp., $x \notin L$). By applying Corollary 7 as indicated above it follows that only one of these equation may hold. Using the decision procedure guaranteed by this corollary, we implement Step 3 in our derandomized procedure, and Theorem 2 follows. ■

Note that for a $\mathcal{BPP}$ algorithm which uses ℓ coin tosses and can be implemented by circuits of size s, our derandomization only invokes the hitting-set generator with parameters $(\ell, s \cdot \ell)$ and otherwise runs in polynomial time. However, if the algorithm only has constant error probability, we must first reduce the error to $1/2\ell$, which increases these parameters somewhat. Using standard error reduction (running the algorithm $O(\log \ell)$ times independently and ruling by majority), we obtain the following more quantitative version of our result:

Theorem 8 *Suppose there is a hitting set generator* G *such that* $\mathrm{G}(\ell, s)$ *is computable in time* $t(\ell, s)$*. Let* L *be a problem with a constant-error* $\mathcal{BPP}$ *algorithm that, on inputs of length* n*, uses* $\ell = \ell(n)$ *coin tosses and can be implemented by circuits of size* $s = s(n)$*. Then*

$$L \in \mathcal{DTIME}(\mathrm{poly}(t(O(\ell \cdot \log \ell), O(s \cdot \ell \log \ell)))).$$

The $O(\ell \log \ell)$ in the first argument to t can be reduced to $\ell + O(\log \ell)$ by using random walks on expanders for error reduction.

Acknowledgments

The second author thanks Adam Klivans for explaining [5] to him.

References

[1] A.E. Andreev, A.E.F. Clementi, and J.D.P. Rolim. A new general derandomization method. *Journal of the Association for Computing Machinery (J. of ACM)*, 45(1), pages 179–213, 1998.

Hitting Sets Derandomize BPP. In *XXIII International Colloquium on Algorithms, Logic and Programming (ICALP'96)*, 1996.

[2] A.E. Andreev, A.E.F. Clementi, J.D.P. Rolim and L. Trevisan, Weak Random Sources, Hitting Sets, and BPP Simulations. To appear in *SIAM J. on Comput.*. Preliminary version in *38th FOCS*, pages 264–272, 1997.

[3] H. Buhrman and L. Fortnow. One-sided versus two-sided randomness. In Proceedings of the *16th Symposium on Theoretical Aspects of Computer Science.* Lecture Notes in Computer Science, Springer, Berlin, 1999.

[4] S. Even, A.L. Selman, and Y. Yacobi. The Complexity of Promise Problems with Applications to Public-Key Cryptography. *Inform. and Control*, Vol. 61, pages 159–173, 1984.

[5] O. Goldreich and A. Widgerson. Improved derandomization of BPP using a hitting set generator. Proceedings of *Random99*, LNCS 1671, Springer, pages 131–137, 1999.

[6] R. Impagliazzo, A. Wigderson, P=BPP unless E has Subexponential Circuits: Derandomizing the XOR Lemma. *29th STOC*, pages 220–229, 1997.

[7] C. Lautemann. BPP and the Polynomial Hierarchy. *Information Processing Letters*, Vol. 17, pages 215–217, 1983.

[8] M. Sipser. A complexity-theoretic approach to randomness. In *15th STOC*, pages 330–335, 1983.

[9] D. Zuckerman. Simulating BPP Using a General Weak Random Source. *Algorithmica*, Vol. 16, pages 367–391, 1996.